On Deep Learning for Procedural Techniques in Computer Vision

Ryan J. Spick

Doctor of Philosophy

University of York

Computer Science

April 2022

Contents

Chapter 1

Preface

1.1 Abstract

Deep learning has been shown to improve the generation of content within virtual environments, with a vast list of approaches. Typically content like terrains and objects are generated either by hand, or through the use of specifically designed algorithms such as using Perlin noise. Using generative adversarial networks we explore the use of sparse and abundant data sets for the generation of specific content within virtual environments. In this thesis we learn the applications of both prior and novel methods, focusing more on generative adversarial networks, of using deep learning to create content that would be plausibly used within a modern virtual environment. We explored 3 different learning approaches, and more typical non-data driven approaches, each area providing a novel contribution to the field. More specifically we explore the improved use of loss functions on certain data sets to improve efficiency and overall visual fidelity. We further evaluate visual generations using both statistical tests and qualitative evaluations. Primarily in this thesis, the work focuses on terrain generation, and how current terrain generation using noise based systems can be improved with more targeted generation of terrain areas. In this same chapter we collaborated with a leading video game company, and explored the use of GANs as an assistive tool for high quality texture generations. The final chapter of the thesis focuses on 3D generation using GANs, and follows an industry collaboration using point cloud data on human pose, with another experiment on voxel based generation using colour and sparse data sets.

List of Figures

List of Tables

1.2 Acknowledgements

I would like to thank everyone who has supported me through my PhD, especially my parents, and a special thanks to my Mum who read every single line to make sure any spelling errors were caught out, any remaining grammatical errors can be sent to her.

James Walker for being a great supervisor, and offering brilliant direction all throughout the PhD. Tim Bradley for being involved in supervision in the latter part, and his insight and contribution directly to one of the publications.

Everyone from IGGI 2019 who I spent much of my pre-covid time with in a brilliant working environment, competitive but light-hearted. Especially the RCH office 234 guys. To name you all in no particular order, Adrián Barahona-Ríos, Nathan Hughes, Mathew Whitby, Adam Katona, Evelyn Tan, Kevin Denamganai, Ozan Vardal and Oliver Scholten. And to those based down in London who I spent many months with during taught modules.

Oliver in particular for being there from the early days of the University of Hull and seemingly following me everywhere I went from then on.

Thanks to my wonderful girlfriend Becca for making sure I actually function on a daily basis.

Lastly, to all those involved in the many industry collaborations leading to much of the work in this thesis.

1.3 Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as references.

Some parts of this thesis have been published in conference proceedings; where items were published jointly with collaborators, the author of this thesis is responsible for the material presented here. For each published item the primary author is the first listed author.

Chapter 5 contains;

[6] R. J. Spick, P. Cowling, and J. A. Walker, "Procedural generation using spatial gansfor region-specific learning of elevation data," in 2019 IEEE Conference on Games (CoG), pp. 1–8, IEEE, 2019.134

[7] R. J. Spick and J. walker, "Realistic and textured terrain generation using GANs," in European Conference on Visual Media Production, pp. 1–10, 2019.

Chapter 7 contains;

[8] R. Spick, S. Demediuk, and J. Alfred Walker, "Naive mesh-to-mesh coloured model generation using 3d gans," in Proceedings of the Australasian Computer Science Week Multiconference, pp. 1–6, 2020.

[9] R. J. Spick, T. Bradley, N. Williams, and J. A. Walker, "Human point cloud generation using deep learning," in European Conference on Visual Media Production.

1.4 Abbreviations and Definitions

AI	Artifical Intelligence
BCE	Binary Cross Entropy
CGAN	Conditional GAN (cGAN)
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
CUDnn	CUDA Deep Neural Network Library
fBm	fractional brownian motion
FCN	Fully Connected Network
GAN	Generative Adversarial network
GCN	Graph Convolutional Networks
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
PC	Point Cloud
PCG	Procedural Content Generation
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SRTM	Shuttle Land Elevation Data
VAE	Variational Auto-Encoder

Chapter 2

Introduction

2.1 Deep Learning for Content Creation

Procedural content generation (PCG) is the creation of any content in a game or virtual world that implements some sort of algorithm. PCG has led to the creation of near-endless environments, with interactable content appearing entirely new while ensuring the visual fidelity of the content is high.

In this thesis we will explore the use of deep learning for PCG and where applicable compare direct results with more prominent non-deep learning alternatives. We also aim to answer the question of whether deep learning can be used for a visually *good* generalised content creation tool. We will use qualitative, quantitative, and purely non-subjective visual comparisons.

2.2 Motivation & Research Gap

Typically in current games and virtual environment industries, we see the use of pipeline-based methods. These methods use some sort of noise, or predefined data structure to build up a piece of content following certain constraints without evaluation or feedback. The main takeaway of these commonly used methods is the inability to create meaningful content without the use of human involvement for the rules and parameters of these algorithms.

Therefore we assert the motivation behind this thesis as the exploration and development of purely unsupervised output models, where once trained, a model can output a large array of varied content, that follows the pre-existing pattern of the training data.

Generalisability in the context of deep learning refers to the model's ability to adapt suitably to unseen data. Further, in the context of generative deep learning, this touches on the range of data within the constraints of the original data distribution, especially the latent interpolation between data.

The area of content generation is vast. We aimed to highlight small areas of the field that could benefit from the use of generative deep learning. We will be focused on unsupervised learning, this allows research to be iterated extremely quickly as for the lack of requirements to self-label, or even seek funding for labeling of data sets.

2.3 Hypothesis & Aims

We hypothesise that a feature rich trained generative model that can generalise will outperform algorithms specifically designed for certain content generation, and create an efficient replacement with more control over requirements.

To attain some answer to this, we will explore the use of generative deep learning for both 2D and 3D space content creation, comparing generations performed by trained networks to pre existing methods. Furthermore, we will explore the use of conditioning to achieve more control over generations compared to the typical randomness of latent space.

This thesis will aim to accumulate literature in the area of deep learning applied to content creation in virtual environments, and where our novel contributions fit into the current research field. We summarise the main aims of this thesis as:

- Improving the current state of generative neural networks, particularly more specific applications of GANs.
- To apply various quantitative and qualitative analyses to determine how strong results are, as opposed to purely subjective analysis.

- Showing real world application of methods outlined in this thesis, with direct engagement with industry partners. There were two separate collaborations during my PhD, one with a leading game development studio Revolution Games and another with a multinational video game and digital entertainment company, Sony Interactive Entertainment.

2.4 Contributions

Over the duration of the PhD and the subsequent contributions through conference papers, we explored multiple different architectural designs, where their contributions can be summarised as the following (those in bold are contributions used within this thesis, otherwise these publications were excluded through lack of relevance);

- **Novel method of generating height maps for 3d environments with GANs**
- **Novel method of texturing height maps using satellite data and GANs**
- **Sparse 3D textured voxel generation with mesh Poisson reconstruction**
- **Detailed 2D texture generation of hand drawn assets**
- **Dense human Point cloud-based generation and mesh reconstruction**
- **Collaborative patent submitted with Sony (SIE)**
- Classification prediction tool for death forecasting in Dota 2
- Using MAP-Elites for game space illumination
- Deep Learning for Wave Height Classification in Satellite Image for Offshore Wind Access

2.5 Thesis Overview

A brief overview of each chapter can be found here:

- Chapter 1: Preface to thesis - contains a list of Figures and Tables, publication declarations and abbreviations.

- Chapter 2: Introduction - This current section, outlining motivations, contributions and research questions.

- Chapter 3: Literature survey - Exploring pivotal research that has led the field to the current state of the art, beginning with basic deep learning leading up to generative networks. The field of PCG is then explored, with both non-deep learning, and deep learning-focused techniques outlined.

- Chapter 4: PCG in 2D space - First content chapter on novel deep learning techniques that were researched throughout the PhD, with a large focus on heightmap, or terrain elevation, data. Lastly an inclusion of an industry placement exploring high fidelity hand-draw images, and how a deep learning based pipeline was created for highly domain specific generations. This section also includes a sizeable survey performed to attain qualitative feedback on generative content.

- Chapter 5: PCG in 3D space - Second content chapter on generations in 3D space, this section focuses heavily on human pose generations using GANs as well as sparse coloured voxel models. We also explore a far more modern approach with graph convolutions and their power in unsupervised complex data.

- Chapter 6: Conclusion chapter - Final chapter outlining the thesis as a whole and what we discovered. The hypothesis is tested against the novel contributions of the thesis. Where we see the future of the field and any potential avenues of research that were not followed are discussed.

Chapter 3

Fundamentals of Deep Learning

3.1 Deep Learning

3.1.1 Neural Networks and Perceptrons

Neural networks are at the base of any deep learning model; these are essentially approximation functions that act to form a probability classification given some training data. A perceptron is the main component of a neural network, and on its own predicts a linear function to separate data points. The perceptrons role is to determine the decision boundary between some data, a linear decision boundary is useful in some low dimensional data such as height against age, but is practically useless in any higher dimensional data.

To alleviate the problems with singular perceptrons, they can be stacked together and connected to one another across layers. This leads to what is known as a multi-layered perceptron (MLP) and an example of a small MLP can be seen in Figure 3.1. Non-linear activation functions typically exist at the output of every node, increasing the capabilities of learning more complex and high dimensional data.

Each added set of perceptrons makes up a new layer in the network known as the depth, with each subsequent set of nodes added giving the network more abstract representation of the data being passed through. Increasing the number of nodes is known as the width of the network. One full pass through a network of nodes is known as a forward pass. The nodes of each network contain a weight value, this is updated when the

network produces a backwards pass. The backwards pass accounts for any error produced on the loss of the network, for the example in 3.1 this could be a simple loss based on how different the prediction of the network was for a specific pass of data, against what the actual real data is.

The example shown in Figure 3.1 has one hidden layer, and would work well for very small data sets of low dimensionality, but in practice these networks get a lot bigger. Increasing the depth and width of the network improves the learning capacity, and has been shown to produce more accurate prediction models.

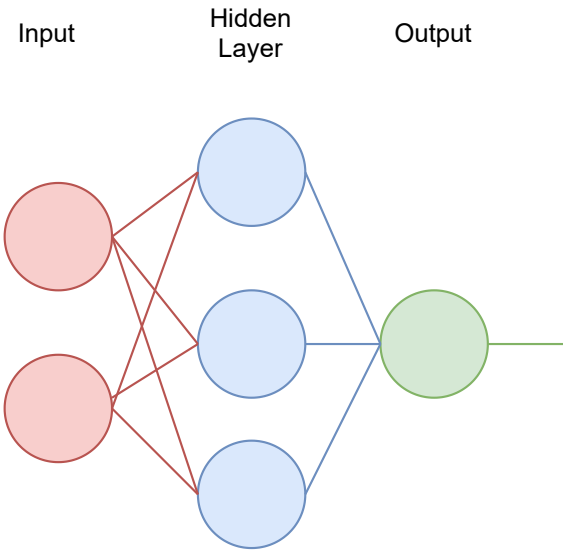


Figure 3.1: Neural network example, it consists of an input layer, an output layer, and some amount of hidden layers.

3.1.2 Backpropagation

During the backwards pass of a neural network, weights are adjusted to account for the error determined by the network. Backpropagation is a way of ensuring the nodes achieve an update based on this value. For a normal training pass of a network, or an iteration, the data is passed forward through the network. A gradient is calculated using the chain rule, the forward pass calculates a multiplicative value based on the data passed in and the weights of the nodes, finalising in some value at the end of the network. This value is compared to a target value, or label data, and the difference, known as the loss or error, is propagated back through the network, with the weights adjusted to positively affect the next iterations estimated loss. Over time the loss is reduced with each pass of the network,

which ensures decision boundaries generalise for the training data and subsequently when classifying unseen data.

3.1.3 Network Accuracy, Loss and Parameters

As previously discussed in a simplified way, loss is the fundamental relationship between the training data and the network's capacity to learn. Parameters make up the many variables within the network, with the weights of each node being one. The neural network learns a highly complex and abstract relationship between real data and the truth of the data by changing the trainable parameters.

There are many types of loss function, each offering different functionality. But the overall premise is to create a mathematical function that captures the performance with respect to the data of a network.

A loss function that is commonly used in regression tasks is mean squared error (MSE). MSE is the difference between the truth of the data and a networks prediction, which is then squared and and a mean of the values is taken. The formula below shows how the loss is calculated, with x_i being the ground truths, and y_i the predicted values. D is the number of samples we are testing against.

$$MSE(D) = \frac{1}{D}\sum_{i=1}^{D}(x_i - y_i)^2$$

It is also possible to create entirely novel loss functions for specific learning operations, such as adding an exponent to punish loss that is performing poorly, or positively weighting predictions of classes which are harder to predict.

Binary cross entropy is a standard loss function used across many classification tasks, this function differs from the regression based functions, and can only take one of two values, either 0 or 1.

Cross entropy is calculate as,

$$Entropy = -(y \log(p) + (1 - y) \log(1 - p))$$

However when the classes are greater than 2 a different loss is calculated for each label, denoted as,

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

There are many other parameters that govern the performance of a network that are not trainable, but may still be adjusted during training; these are known as hyper-parameters. Learning rate (LR) is a crucial hyper-parameter which determines the speed at which a model is controlled by loss updates, with each iteration the learning rate simply multiplies itself with the loss. Therefore a lower learning rate would lead to much smaller weight update per iteration, this is crucial to how networks converge on a useful optimal weights. Smaller learning rates require more training time (or epochs) due to the smaller changes to the networks trainable parameters, whereas larger learning rates require fewer steps, but can often lead to sub-optimal final network weights. An epoch indicates an entire pass through the entire dataset.

3.1.4 Data & Network Learning

There are 4 major types of learning that have been applied to tasks for learning representations. Figure 3.2 shows a visual example of the two main types of learning used in this thesis and how the data is represented;

Supervised Learning - Data is labelled with its true output and the true output is factored into the loss function. Each forward pass the loss is calculated accurately for the prediction, against the real data.

Unsupervised Learning - Learning of data is conducted with a function of the input, in other terms the input learns a representation of itself. However when averaged across

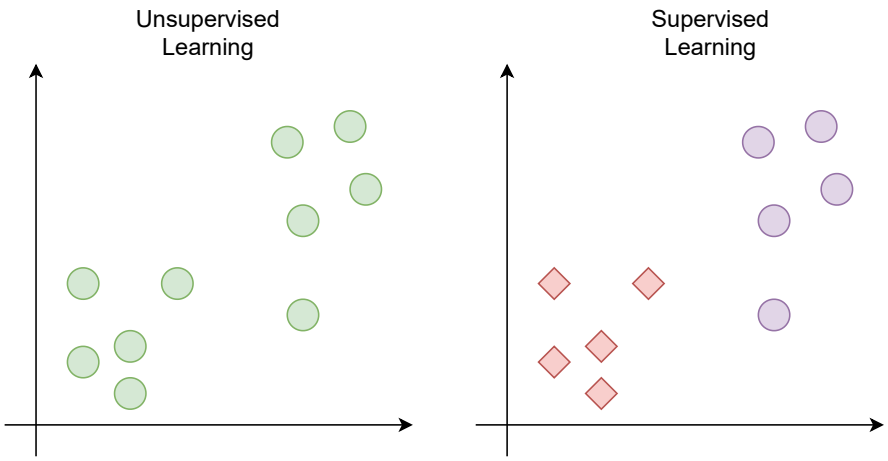


Figure 3.2: Example of supervised learning and unsupervised learning, and if the data has label information associated with it. With the unsupervised learning having no associated knowledge of the data. Whereas the supervised learning has label data (circle and diamond) for the two data classes.

multiple inputs a mean of the representation or a cluster is formed.

Semi-supervised learning - The learning here is performed using a small portion of label data alongside a large amount of unlabelled data. The large strength of this type of learning is the ability to label few samples, which could be time consuming to label manually, and allow the learning task to predict and subsequently label the rest of the data.

Reinforcement Learning - A learning method in which no data is typically used, but instead an environment with certain constraints is used, the model is trained using some actor-reward metric, with an actor performing actions which leads to certain amounts of reward. The aim of this model is to maximise reward for certain environment states.

In this thesis, we will mainly be touching on the use of unsupervised learning, though in some work we will cover supervised approaches. Unsupervised learning approaches encode data in a some form without the expressed need for labelling of data. With current data sets this approach creates a powerful dynamic, where large amounts of pre-existing data sets form games/artists/content creators may be used almost without time constraints that would otherwise be associated with labelling in supervised training.

3.2 Deep Learning Networks

The most basic of the networks is an Artificial Neural Network (ANN) [10]. This type of Network is a non-linear, feed forward network that takes in a defined input size and processes this through a number of fully connected hidden layers before finally outputting a value correlating to its prediction. Each layer contains a weight value that is updated with backpropagation. The fully connected layers learn representations of inputs that "activate" depending on the correlated strength of features that the nodes have learnt based on all prior training inputs. This model is the building block of many of the other divergent networks that tend to add customised layers and functionality on top of the "basic" fully connected model.

There are many other types of network architectures used for various data types. Data such as music, and text, those that contain some recuurrent data which relies havily on previous data to predict of classify future points require entirely different networks. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) Networks are two networks, with LSTMs building on top of RNNs, both of these excel at understanding past data with respect to current and future data. these networks are a feed-forward network that contains internal memory, and such can "remember" data which has been cycled through the network previously.

3.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) offer a highly efficient way of process image data, in ways that would scale inefficiently for networks like an MLP or ANN. CNNs have taken the computer vision field by storm, and are the basis of nearly every machine learning computer vision task, outperforming even hand crafted algorithms for both classification and generation tasks.

Recently CNNs have grown tremendously at recognition of various data tasks, where the automation that previously struggled to solve these challenges now surpasses even human recognition. There are two areas that have largely improved the way data is trained and predicted. The models that we use to pass our data through have become

deeper and can train far quicker than before. This is partially due to the increase in GPU power, and the methods employed to reduce the model sticking to certain features and beliefs of data known as over fitting. Convolutional Neural Networks are largely used for their capability of processing extremely large input data in a much more efficient way compared to the vanilla ANN, through various computational tricks.

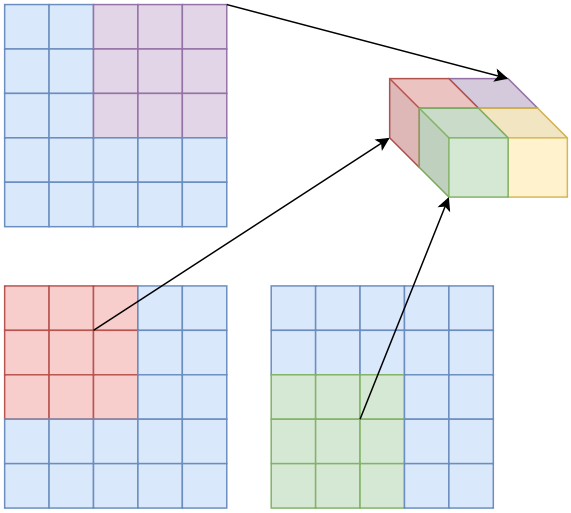


Figure 3.3: A 3x3 filter convolving over a 5x5 input grid, with a stride of 2. Resulting in a 2x2 output.

One of the first applications of CNNs successfully with the inclusion of backpropagation was [11] where Le Cun successfully trained a model to recognize handwritten zip codes. Later Le Cun massively improved this model [12]. This architecture was developed to read handwritten characters, an area that previously algorithms struggled to solve due to the huge variance in handwriting structures. As this network formed and evolved over time the accuracy output increased to a point where the network could predict the output of handwritten digits to a higher degree than a human. Although one of the oldest networks used for image classification using CNNs, the methods used in LeNet are still employed today by even the deepest, most advanced prediction models such as those use in ImageNet [13].

More recently challenges emerged pushing researchers to assemble and train networks on static data sets to achieve the highest accuracy. One of the more popular challenges which shows the power of supervised learning of CNNs is Image Net's ILSVRC Source, consisting of over 1.2M photos of 1000 different object classes for training with a further 150k for testing/validation. Each image has been hand labelled with a class so provides

perfect data for the testing of networks. The winners of each years challenge provide a good insight on to how to develop and train Deep Networks. [13]

There are various functions that CNNs utilise which separate them from the generic networks such as ANNs and are inherently good at working with large multi dimensional inputs. The following sections will discuss the terminology of these functions, how they apply to CNNs and what their purpose is.

3.3.1 Convolutions (Filtering)

The function that gives CNNs its name and does almost all of the heavy calculations for prediction in the network. This process essentially works as a vector of weights that convolve over the inputs in all dimensions. The filter's aim is to find certain features that the weights have learnt regardless of their position on the image. Each filter calculates a dot product against the width/height and **all of the depth** of an input for a given area of the filter, 5x5 for example. Leaving an output of an activation map with depth n (correlating to the number of filters specified) and with remaining output size calculated by equation 3.1. This exploits the potential compositionality of the data, between local areas which feed a feature representation to the next layer. However, if the data does not share common features amongst the image domain then the network would struggle to learn a combined hierarchical representation. As a rule of thumb the smaller resolution the input data the smaller the filter's size should be [14].

The filter output size from a layer in a CNN can be calculated as follows;

W = Input size (assuming height == width)
F = size of filter
S = Stride of filter
P = Padding (adds a border of *zero* values, allowing the control of the output spatial size)

$$OutputSize = \frac{W - F + 2P}{S + 1} \tag{3.1}$$

For example for a 33x33 input using a 3x3 filter with stride 1 and pad 0 would

result in a 31x31 output ((32-3+2*0)/1+1). With stride set to 2 the overall size would be reduced to 16x16 ((32-3+2*0)/2+1).

In Springenberg et al's work[15], a method of using solely convolution layers with certain parameters; for example the convolutions stride is set to 2, reducing dimensionality in the width/height by (k-1)/2 as shown in equation 3.1, allows the removal of the pooling function entirely, with results in cifar-10 almost identical to the current state of the art.

3.3.2 Pooling

Pooling is used to spatially reduce the size of the input data progressively, while retaining important features.[16] The pooling function works on each input dimension independently, where a specific area of the input dimension will be computed for one output value for each of the the pooling checks. The size of the selection is stated by the pool size, this is usually a 2x2 grid. The stride value will determine how many values in each depth grid the layer will pass the pool operation over, commonly set to 2. With these values there is a reduction in the data by a factor of 2 while maintaining important features that the network has identified.

3.3.3 Rectified Linear Unit (ReLU

Although this activation function is not constrained to CNNs, its uses are worth noting due to the popularity. This activation function is used in nearly every modern high performing CNN [13] [17] [18] [19] as a way of introducing non-linearities to ensure the CNN approximate in a much larger space.

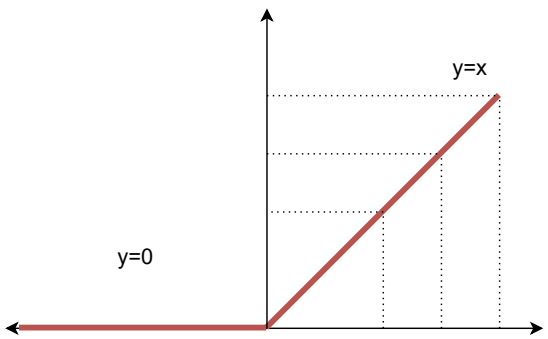


Figure 3.4: ReLU Function

ReLU is a non-saturating non-linear activation function. The purpose and need for using non-linear functions throughout the network is important, without which, all hidden layers could be summed and combined to one single layer perceptron [20]. Non-Linear means the output cannot be created from a linear combination of the inputs. Activation functions are required after most layers in order to decide if a neuron should activate for a specific input or not.

3.3.4 Fully Connected Layer (Dense)

A fairly simple concept and similar to a layer in an artificial neural network. The previous layer is connected to every number of neurons or nodes defined in the fully connected layer. This layer acts as a refinement for the thousands of parameters that exist throughout the network, and is required in order to use our activation function. Although this function can be used anywhere in the network architecture, it is essential for the last layer when wanting a discrete output, the number of classes in the networks definition are fed in as a parameter of the activation function (softmax for multi logistic and sigmoid for binary). The stronger a correlated node is "turned on" when passed in to the activation function defines which class it predicts.

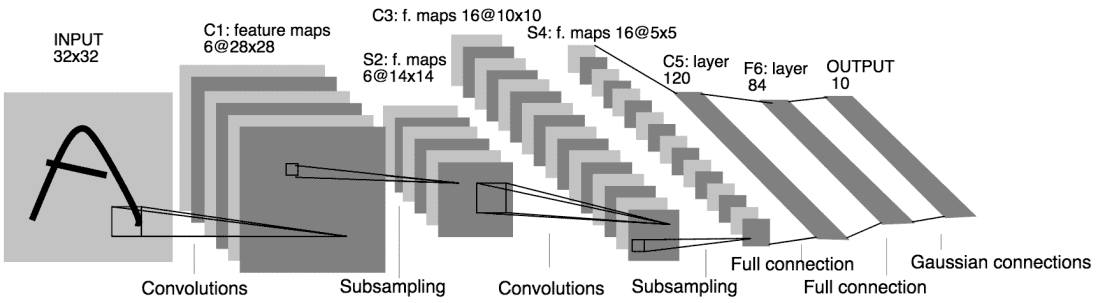


Figure 3.5: Full example of a CNN (Source)

These layers, combined with the use of padding to extend data to the same dimensions, creates a large degree of freedom with the type of data CNNs can process.

3.3.5 Advances in CNNs

Over time CNNs for classification have been massively improved, out performing nearly all previous approaches for image recognition. This is largely due to the increase in available

centralised data sources, datasets like ImageNet [21], and the architectures used to classify this data (cite alex net, vgg, inception, resnet).

Since 2016 architectures have been tending to get deeper (FNet [11], VGGNet [9], GoogleNet [10] and ResNet) whilst mostly keeping similar structures and designs for the network layout.

Due to the evolution of hardware, and more specifically the memory capacity of GPUs, it is now possible to process larger batches of data and pass these through even deeper networks. These deep networks contain more nodes and can learn a far better approximation of feature space. Though it is a broad statement to say deeper is better, the typical processing of particularly larger data sets have been shown to perform better on those networks which have the ability to hold a more complex representation for hard to classify classes.

ILSVRC is a research competition aimed at achieving the best possible results for classification of ImageNet, with the results measured using the top-5 error rate. This error rate is the number of images in the test set that are correctly classified in the top 5 identified classifications. This error rate has seen an exponential decline since the competition started in 2011, proving how quickly CNNs have improved at classification of images.

The use of multiple inputs to a layer showed promising results in the Inception model (cite), these intermittent additional inputs are known as Residual Connections. Inception "branched" every few nodes in the network, performing concurrent operations on the data, these operations would then be concatenated back in the a single node. This improvement lead to the network outperforming all previous architectures on the ImageNet data set. In 2015, ResNet (https://arxiv.org/pdf/1512.03385.pdf) used a massive 152 layer deep network that consisted of an ensemble, to achieve an error rate of 3.57%.

There are two paths that aim to improve how CNNs perform, one being optimization which looks at improving how the network performs and trains, such as decayed learning, batch norm and adaptive learning rates; Whereas another path is to improve on the feature extractions of CNNs. Typically CNNs are fantastic at a specific classification of singular images appearing within a defined region. However applying any of the previous imagenet

winning architectures to a wider image filled with various objects would show only one classification, which would probably be wrong.

3.3.6 Data Domains

Data domains are a broad and high level classification of types of data. With respect to deep learning, these are data types which are trainable by a type of deep learning network. This section will outline some of the data types that exist, and go in to more detail on the types that are used in this thesis. We will also explore how certain untrainable data can be processed in to a trainable form. Data in this thesis will draw from both outlined data domains in this section.

Euclidean Data

Euclidean domain data corresponds to data which exhibits a uniform structure. This can be imagined as a 2D image where pixels are aligned in a perfectly structured environment on a 2D grid, where data has to exist in every space. Similarly, sentence structures contain a 1D list of uniform characters or words. Voxels are another good example. A data type that forms a value in a uniform 3D grid, with the positional information inherently encoded in the voxel data structure. Voxels can represent and contain more than 3 channels of data, for example the colour of each voxel and a value of whether it exists or not, but nonetheless every square in a voxel grid must contain data and be the same size as each other data point.

The importance of this type of data when understanding typical deep learning architectures is how the input is processed. All Euclidean deep learning architectures need data to be in the same static length. The input layers of these networks are typically of predetermined dimensions and are not changeable. An example would be for some input image of size 33x33 pixels with 3 colour channels. Although since this data is an odd number and not a factor of 2 using a window stride of 2 would mean 1 pixel is always missed out. Therefore padding would be required for this type of odd data to ensure the window could properly cover the image.

Padding is not only used for data uniformity, and there are several important reasons that make padding a crucial part of deep learning models. The effects of padding vary dependant on the type of data used, but typically for image processing the use of padding can improve performance, as potential key information is retained around the image borders [22]. Padding can be used deeper in the network during concatenation operations, and many of the top performing ILSVRC such as Residual Net and Inception Net networks use this technique to ensure different sized kernels can be concatenated back together.

Non-Euclidean Data

Conversely to euclidean data, we see a large variation of data that doesn't fit to this conventional euclidean structure, this is known as non-euclidean data. Data such as Mesh obj, stl, 3ds etc. is typically not in correspondence, where all data points align with one another within the dataset, and can be of differing lengths compared to each other. An object file of a duck model created by one artist might be entirely different to that of a human model, they may have varying data channels (normal maps, texture maps) and varying vertex positions and numbers. It would be near impossible to have any sort of correlation between these two files. But having the ability to classify these types of data would unlock a plethora of unused data, however previously discussed conventional deep learning networks have no way of even remotely getting close to process this data in raw form, without some sort of data pre-processing.

Another example less relevant to content generation is social networks. These can be seen as data that cannot be mapped easily in to some linear space without losing high-dimensional information about the graph structure, and potentially introduce artifacts in the data that have *spurious* relations that did not exhibit these properties before. A good summary of non-euclidean data is "the shortest path between two close points might not be a direct line".

This social network example can be used as a straight forward way of representing the structure of a non-euclidean 3D data structure. This graph based approach can be used to reduce the complexity of an object file down to a set of points in a space, with

these points being used to form a graph.

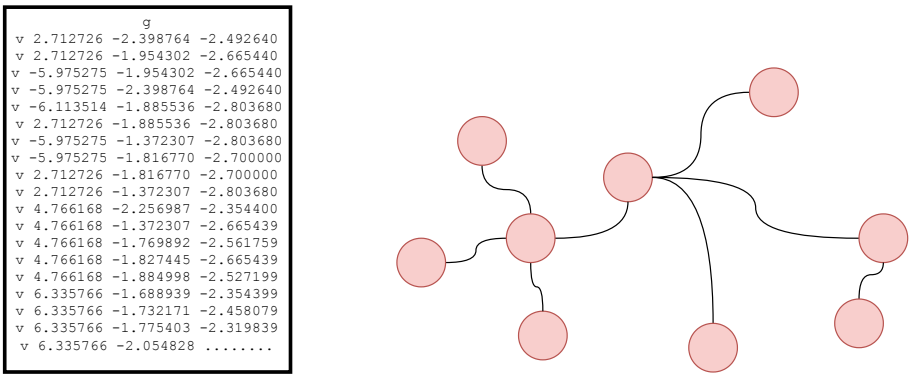


Figure 3.6: Naive conversion of an OBJ file in to a graph representation, using only vertices and their nearest neighbour as a connection. Indices can also be used to determine graph node connections.

3.3.7 Graph Neural Networks

Graph Neural Network (GNN) is a type of Neural Network which operates directly on the graph structure. It is a type of machine learning algorithm that can extract information from graphs using the nodes and edges. Graph data is typically not compatible with the discrete type architectures used with other data, such as image which contains fixed constrained input sizes.

Graph neural networks can be created like other neural networks, using fully connected layers, convolutional layers, pooling layers, etc. The GNN takes in the formatted graph data as an input and produces a vector of values that represent relevant information about nodes and their edges. This vector representation is called ”graph embedding.” Embeddings are often used in machine learning to transform non-conformative data into a structure that can be learned.

3.3.8 Graph based conv nets

Graph convolutions are being used more and more in research around images and typically difficult data strictures. An interesting and novel concept is the idea of using convolutional neural networks on graph structures,

However if we tried to use a standard 2D convolution on a graph, this would fail,

convolutions are designed to reference neighbouring pixels in a uniform data structure, something that graphs do not have.

With the exception of neighbouring nodes along edges, although even this would fail some of the time, as graphs usually are order invariant.

Furthermore, graphs have arbitrary sizes and can vary from one data point to another, something that with the use of typical CNNs would fail at.

3.4 Generative Adversarial Networks

GANs (2014) Provide a unique framework that utilizes two deep neural networks: a generator (G) network which attempts to capture the distribution of the training data, mapping this on to an input of variant noise (latent space). This is essentially the inverse of a classification network, such as an MLP, with the beginning of the generator consisting of a latent vector. This vector contains random feature values which are high dimensional and non-observable, the values are wide and deep enough to contain a description of the data and more often than not contain millions of variables which collectively make up the latent space. And a discriminator (D) network that will estimate the probability of its input being from the original training data or from the generators "recreated/forged" output, basically a discrete multilayer perceptron classifier [23]. The discriminator will be trained over time on both the real and fake data, and essentially become a binary classification network between what is real and fake.

Adversarial loss is a special type of loss used in GANs and is typically calculated using binary cross entropy, this loss rewards heavily when the generator outputs "perfect" data which fools the discriminator,

$$Loss = logD(x) + log(1 - D(G(z))) \tag{3.2}$$

,where x is some data, z is a randomly sampled noise vector from a distribution p(z). Usually Z follows a normal distribution of 0 - 1.

The power of adversarial networks lies in being able to mimic any distribution of

training data from a large amount of domain areas with a fully unsupervised approach. The real intrigue of generative networks was the lack of requirements for human involvement. Though more recently semi-supervised and fully supervised learning have been associated with GANs in order to achieve more control over generations, along with more statistically accurate results [24]. Supervision has also been shown to improve the training process of GANs [25]. This goes to show that although AI is in a fantastic place for generating content, the crucial direction of human involvement, through labelling or specific loss functions, will always lead to improved results.

Previous deep convolutional generative networks followed a trend of using specified probability distributions followed by using log likelihood to approximate these distributions to generate outputs. An example of such being the Boltzmann machine [26] - an undirected neural network, which aims to learn a representation of known visible variables against a hidden unknown variable.

Latent space represents the space at which classes are associated and grouped within a model. Sampling this space for outputs can provide a similar distribution of data to that of the learned inputs. Furthermore manipulating the latent vector inputs to the generator can create seamless transitions of multiple classified groups within a model. In the original DCGAN paper this can be seen with transitioning genders of faces [27].

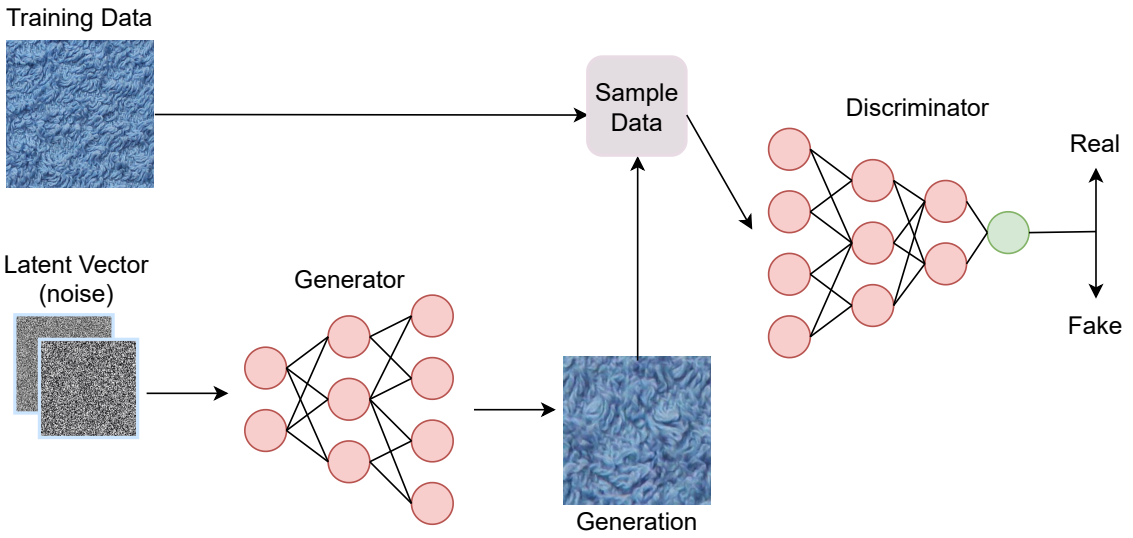


Figure 3.7: Example of a typical GAN training loop, using a dense neural network for both generator and discriminator. Using fabric textures as input and generation results.

3.4.1 Initial Contributions

Motivation behind GAN research came about as a broad method of obtaining representations of implicit data in the real world. Artificially manufacturing replica data has an extremely wide range of uses, from augmenting and translating image data, to animating paintings, or simply creating new recipes. All of these approaches uses some real-world data and draws on the distribution to create some abstract understanding of the data set.

The initial work on GANs by Ian Goodfellow used a multilayered perceptron model as their generators and discriminators networks [23]. They used several different data sets, MNIST, TFD, CIFAR-10, to prove the concept of GANs in their infant form, with very promising generated samples. Moreover an important part of their visualisations was the use of random sampling of data points, instead of traditional "cherry picked" results [23].

A method proposed in 2016 [27] built on the work done by Ian Goodfellow et al. by replacing the MLP networks with CNNs creating a new architecture known as Deep Convolutional GAN or DCGAN. This architecture allowed breakthroughs in the way images were generated with more consistent results, previously producing noisy and incomprehensible outputs. Using three differences over classical CNNs the GAN methodology was able to utilize the power of convolutions in a more stable environment; pooling layers were replaced with strided convolutions [15] (Strided convolutions can be useful for when spatial information is not relevant. Whereas traditional pooling allows the network to forget about spatial structure of inputs), batch normalization (Batch normalization normalizes inputs of a network to zero mean and unit variance.) is used on inputs of *both* networks, there are no fully connected layers in between input and output layers of the network.

In 2016 [28] proposed the use of 3D convolutions to learn representations of voxel based data. Previously 3D model generation was performed using more non-parametric algorithms to pool certain parts of models together. More background on 3D model generation can be seen in Chapter 7. This paper used the same architectural design as the original CNN-GAN [27], however extended the dimensions to accept a third data channel. These experiments showed highly promising results when training on the IKEA dataset, a dataset consisting of an array of different furniture in voxel format. They also show the ability of learning a mapping, using a VAE, of 2D images and their representative voxel

approximations in the GAN latent space.

Although super resolution using neural networks had been proposed in 2016 [29], work in 2017 [30] (SRGAN is a generative adversarial network for single image super-resolution) using GANs had shown to blow the results out of the water, with perceptually far more convincing results. SRGAN based architecture is the first framework to infer photo realism in natural images, leading to a network that can increase size by 4X of an original image. Their loss function contains two factors, typical adversarial loss and a content loss, which bypasses any connection of pixel similarity. The idea of detracting direct pixel loss in GANs for a more perceptual loss is a key finding that is commonly used in many different generative approaches today. Like many generative image tasks quantitative measures did not correlate to the actual visual fidelity of the images, with obviously less appealing images having higher Peak signal-to-noise ratio (PSNR) values. PSNR is a method of calculating an image similarity measurement between multiple images. Therefore they chose to use quantitative tests through perceptual evaluation from human users, leading to a mean opinion score (MOS) that disregarded the quantitative findings in favour of the GAN.

Published in 2018, BigGAN [31] pushed the boundaries of generating large scale high fidelity images. Whereas the previously discussed super resolution GAN would upsample an input image, BigGAN inherently generates high resolution images from a collection of low resolution (128x128) images. Their main contribution is the demonstration that GANs benefit massively from scaling, increasing the typical paramaters used by 4 times the conventional amount, when compared to prior work. This work offers a selection of particularly useful training improvements that help with increasing stability of GANs.

BigGAN showed extremely promising interpolation results, providing the best inception score at the time. Inception score is a popular metric for measuring perceived realism of an image, this score aims to determine if; generated images are varied in features and if an image looks like a class object from the training set. This technique was first used in [25] where the authors researched techniques to improve training of GANs, this metric was used as a determining factor in justifying improvements. The score is calculated through a series of summed distributions of classification probability for a sample of generated images. They also out perform the previous state of the art in Fréchet inception distance

score, FID is a metric that, unlike inception, compares distribution of generated images with the distribution that the network was trained with. These collection of improvements lead to an extremely detailed generative image, while retaining impressive performance. Alongside this more crucial work follows a substantial amount of smaller contributions that have paved the way for how GANs can perform, on progressively more complex and substantially smaller data sets. BigGAN added modifications that directly improve GAN training performance, from increasing stability, data augmentation and objective function optimization [32, 33].

Far more recently in 2019 we saw A Style-Based Generator Architecture for Generative Adversarial Networks [34], a style transfer based approach. This network adds many changes to the typical generator of a GAN, with the most crucial takeaway the ability to add control over synthetic images. These more modern networks utilise baseline progression; this is a method of training a generator progressively over time on large samples of the training data, for example we would train a network using 4x4 images, which we double in size each time stability is achieved in the initial training pass. Each time the size of the input images are increased, a new block is added to the generator and discriminator to acknowledge the size change. Progressive GANs have been used since late 2017 [35] where authors proved that stability and quality of outputs were far superior using these techniques. The style transfer based GAN uses nearest neighbour up-sampling method, as opposed to the commonly used transpose layers, boasting increased quality when up-sampling in the generator. Another method used, that is more common and not novel to this network, is the use of Gaussian noise at each activation map applying varied noise over the course of training which leads to an increased variation of generations.

Attention is all you need [36], was the first paper that showed the use of transformers, using what is known as an attention mechanism. Although the paper was for classification rather than generation, the importance of transformers for the future of deep learning could be enormous.

Attention takes motivation behind how we as humans pay attention to specific areas in an image, or words in a sentence. Human attention works by very quickly scanning key focal points in our vision, taking in as much important information as possible to make a quick inference of what we see. Similarly when reading over text we pull out

key pieces of information early in sentences inferring what should be coming next. The first attention mechanics used only dot products between input vectors, improved later to multi-head self-attention which instead of calculating the attention once, the dot product attention layer is run in parallel multiple times, essentially creating an ensemble inside the attention layer. The idea behind multiple heads is to allow the model to "jointly attend to information" [36] essentially pooling more information from each input at each layer. Attention nets are now commonly used in language models replacing traditional recurrent layers.

Far more recently in 2021 we see the emergence of sole transformers based GANs [37], typically previous work using attention or transformer like techniques have always contained some sort of convolutions. This work used the CelebA and CIFAR as their data input data. The most impressive contributions come from highly competitive IS and FID scores, when compared to prior state of the art GANs that contained convolutional layers, whilst using only transformer layers. The up sampling is performed, similarly to convolutional up sampling, where the input to the generator is a tensor of height by width with a certain number of channels deep, each up sampling layer reduces the channels by a factor of 4 to double the size of the height and width of the generated image. In the transformers paper for example, their input is an 8x8xC tensor where C is 48, the first up sampling layer transforms this to 16x16x(C/4) then the following up sampling transformers the tensor in to a 32x32x(C/16) which results in a 32x32x3 image tensor output. Each time an up scale occurs, the channels are locally unrolled to discover more pixels. Due to the network only using transformer encoding for the generator, the network suffers poorly from scaling, as each pixel is connected to every other pixel inside the transformers. This leads to only 64x64 generations being shown.

3.4.2 Variational Auto-Encoders (VAE)

GANs shared a similarity with auto-encoders, and so this section will briefly establish the key differences between the two.

VAEs are a type of deep learning network, not too dissimilar to GANs, that uses an encoder network and a decoder network, the encoder reducing the complexity of the

data down to a smaller representation through dimensionality reduction (similar to the discriminator of a GAN) and an encoder, which up-samples this representation to, usually, the same size constraints as the learned input. These learned representations are held within the weights of the decoder and encoder. However representational latent space can be saved and passed through the encoder network to produce reconstructed data.

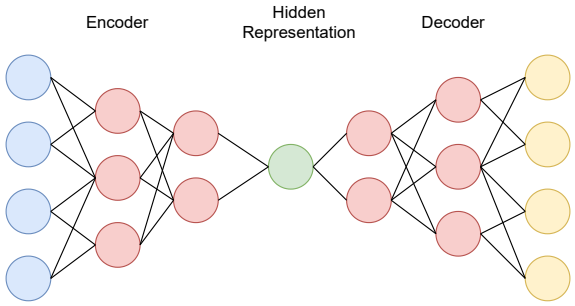


Figure 3.8: A fully-connected VAE, showing the encoding of input data to a hidden representation, then the up-sampling through the decoder.

VAEs were first discussed in *Auto-Encoding Variational Bayes* [38]. Experimental results showed the learning manifolds for MNIST and Frey Face data sets. The main purpose of this network architecture is to find the encoder and decoder pair that keeps the maximum amount of input information when encoding and the lowest error when reconstructing (decoding) the data.

VAEs perform well when latent modelling is required, where the explicit goal of a generative task is to reconstruct, while reducing certain variables. This makes VAEs perfect for data reduction in games, using networks to condense large scale models and textures down to a small latent representation, and passing it through an upscaling network either in real-time, or offline.

For clarity - the difference between a VAE and GAN; a VAE will compress its inputs down to a much smaller representation and then attempt to reconstruct the representation into an identically shaped tensor output. A GAN on the other hand instead of compressing the data, will learn the distribution of the data within the network's weights, and reconstruct through various upsampling layers a randomized noisy input into a coherent output. GANs generate **new** sampled data, VAEs generate **reconstructed** data.

3.4.3 Conditional and Supervised Approaches

Conditional GANs [39] were proposed by M. Mirza et al. and build on the somewhat stochastic nature of generative model's outputs, allowing more control over the way generations are pulled from latent space. An unconditional GAN typically has almost no control, expect for brute forcing outputs to match a certain loss constraint through exploration of latent space.

Conditional GANs use one or more additional labels, turning the traditional unsupervised GAN in to a somewhat supervised approach. These additional labels "condition" the network to learn sample data distributions with a bias towards the label data.

CGANs have been shown to perform particularly well on data sets with a large variation of data, but a small class size, such as MNIST or CIFAR-10. [39]. The use of conditions can also improve the efficiency and training capacity of the GAN itself, due to the nature of pre-defining boundaries in the latent space with regard to the label.

There are however, several limitations with this method, the obvious issue is the need for labelling, which in some cases is simply unfeasible, though a plethora of data sets exist which have been hand labelled.

The task of Image-to-image translation using deep learning is a very recent advance, and is the ability to learn to translate an input image in to an output image following some sort of representation, though has been around long before in computer vision. We first see applications of image-to-image translation using deep learning in [24]. In this paper the authors show a *generalised* method of learning mappings from one domain using the style of another. Using adversarial generator to discriminator loss over traditional L1 loss functions has resulted in a much higher visual fidelity.

In 2018 we saw the culmination of semantic maps and conditional GANs [40] building on top of the work in Image-to-Image Translation with CGANs [24]. This method allowed the synthesizing of high resolution images, with no need for any specific loss models, though a hand crafted loss did improve their results. The results show the ability of semantic mapping of generative results using conditions within the GAN. Importantly they use both quantitative and qualitative testing via human perception studies, proving

that the results shown in the paper aren't just cherry picked and that random generations appear realistic. Their results were slightly below the bench mark data.

3.4.4 GAN Downfalls

Although GANs have been shown to be an extremely powerful tool when used with both large and small unsupervised data, there are many issues that resolve from lack of data consistency, over fitting, mode collapse and failure to converge.

Overfitting is an error caused by modeling data that aligns too closely to a set of points. In terms of GANs from an image perspective, this would mean images matching too closely to the original data, or put plainly *distinct lack of variation*. Although all statistical and deep network problems can exhibit over fitting, it is particularly difficult to detect in generative networks. Since data is trained with respect to a discriminator's classification, the overfitting can be caused with little knowledge of it occurring [41]. Metrics, like euclidean distance of images, can be used to determine if there is lack of variety in the generated samples. Another suggestion in the paper Generative Adversarial Parallelization [42] suggests using multiple discriminators, and proves that it can help mitigate over fitting.

Mode collapse is often mistaken for overfitting and is caused by a generator producing only a few *highly performing* outputs. The generator then determines that these are the best results that fool the discriminator, resulting in a lack of variety in the outputs of the generator. At worst this can be only one output that had no variation. Technically, the discriminator should determine that this output is a generated sample, but if the discriminator gets stuck in some local minima then the generator can simply abuse this, and it becomes trivial to find another plausible output that would fool the discriminator while retaining the invariant generations. The generator can iterate over a relatively low number of generated samples that satisfies the performance against the discriminator.

There are plenty of research papers on alleviating mode collapse, such as using Wasserstein loss [43], an extension on a typical GAN architecture that uses a varied loss function and an adaptation on the discriminator's classification. Instead of a "real" or "fake" classification we see a shift towards a continuous based model that scores on a real-

ness metric. Wasserstein distance can otherwise be known as the earth mover's distance.

The main argument behind WGAN is the continuous gradient returns from the discriminator. As the critic type model can quickly converge to a linear function that returns less unstable gradients. This main argument is why WGAN type architectures have been seen to reduce, or entirely alleviate mode collapse in certain data domains.

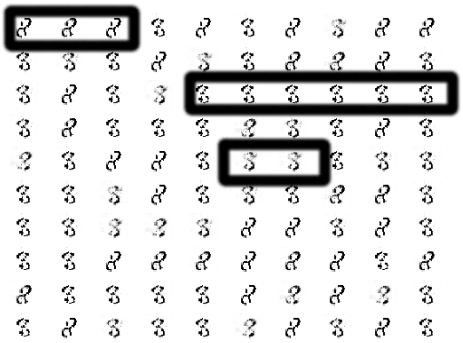


Figure 3.9: Illustrated example of a less severe mode collapse in the MNIST data set, a GAN outputs samples with some variation, however on the 1st, and 3rd line (highlighted) samples with exactly the same distributions can be seen [1]

Lastly, we look at decayed noise of inputs as a stability increase. Decayed noise is used in a vast array of GAN applications, from 2D images [44] to 3D models [8], and is the application of Gaussian or median noise blurs to real images of the discriminator. The fundamental idea behind this is to handicap the discriminator in the early iterations of training, where the generator network is particularly vulnerable to incorrectly generating plausible samples.

More often than not it can simply be an "unlucky" scenario, in which the discriminator is stuck in some minima. Therefore restarting training from a prior checkpoint where there is high variant in outputs or restarting training all together, but importantly changing the random seed, can be a resolution in its own right.

Lastly we look at failure to converge, or also known as the Saturation Problem, and is caused by the vanishing gradient problem. This phenomenon affects neural networks, such as GANs, that utilise gradient propagation to update their weights. The issue arises from the potential of small derivative error updates to weights in the network, in some cases gradients can become so small that the effective update of the weight becomes none.

Though this may only affect some weights in the network to a degree where the network can still function, at its worst case the network may cease to update entirely and thus no improvements will occur. For GANs this would mean consistently poor generations that do not improve over the course of the network's training.

3.5 Summary

The field of deep learning is extremely broad, and whilst an effort to narrow that field into separate applications to PCG have been made there was still a substantial amount of work covered. This section will aim to summarise some of the sections and points made in a less verbose manner. The literature has covered deep learning and its early research, followed by advancing in image classification using CNNs and finally on to the focal point of the thesis being GANs and their application to content generation.

Chapter 4

Procedural Content Generation & Deep Learning

4.1 Procedural Content Generation (PCG)

PCG or procedural content generation is a method of utilising techniques and algorithms to generate content through automated processes, with the definition explicitly excluding any content that has a manual creation process (using graphics engines, inbuilt editor tools) [45]. Though this is an arguable statement, as content generated with procedural techniques that are polished through manual involvement may still be classified as procedural, creating a slightly unclear divide on what PCG encompasses. The content itself can be in any form, from physical objects in a game to generating levels, economy, sounds, text, behaviours... the possibilities that PCG can be applied to are endless.

PCG was originally created as a way of compressing data, with Akalabeth: World of Doom (1979) being one of the first instances, though since setting a precedent for a vast amount of games to adopt this technique. Using terrain generation methods to condense endless maps into a few lines of code [46]. These early games (e.g. Rogue (1980)) would use PCG as a method of holding a larger static game world that would usually require more data storage than was available at the time, using a seeded random number generator the exact worlds the developers devised could be replicated. Another classic example is the adventure game Elite (1984), a game where thousands of planets

were generated procedurally, while using seeds to recreate prior interesting generations, reducing what would have been an intractable problem given the memory bottlenecks at that time into a remarkable, near-endless game. Though as storage mediums increased in space efficiency PCG methods were less likely to be used by developers, as the original reasoning of compressing data could now be stored for games within CD-ROMs. More modern games are still utilising this approach, even though storage has increased massively. No Man's Sky as an example is near 7GB in size but can produce a massive 18^{18} unique planets through randomised procedures. Furthermore, PCG is now used to generate and compress more than just world maps, with PCG being used to create entire entities from scratch [47]. Outerra is another game/simulation, this environment exhibits cutting edge terrain generation that is combined with real-world satellite elevation data to create a real-time, size replica of our planet. The environment also uses PCG to texture and programmatically add forest and vegetation to the surroundings.

Table 4.1 reproduced from [5] Procedural Content Generation for Games: A Survey shows a list of popular modern games that specifically use PCG to generate game space. In bold are the games that incorporate some kind of PCG for game bits and space, the two definitions that this thesis is aimed at. The table shows that nearly every game on the list, from a range of years from 1980 all the way until 2010, using either game bit or game space procedural generation. The sheer quantity of games prove that research on procedural techniques for these domains is of high value.

We see 4 main definitions of content in games that can be generated procedurally; game bits, game space, game systems and game scenarios. This taxonomy was created to help define and segment large amounts of literature on PCG. Game bits are referred to as non-interactable parts of the game, that act independently from one another; these are mainly the immersive parts of an environment such as texturing and sound. Game space is a slightly higher level definition of the overall environment that the user does interact with; this would mainly be classed as terrain maps and is the space that the environment exists in. Game scenarios are how the game is played out, what parts of the game such as storyline or challenges the player is given. Lastly, the definition of game design is given as a much broader control of states, rules and goals of a game; this could be directly controlling variables to increase or decrease a given difficulty level. All 4 of these

Table 4.1: Table reproduced from [5] of procedural game. This table shows the use of PCG, and their specified areas of use, in many modern and popular games.

Games (with year of release)	**Game Bits**	**Game space**	Game Systems	Game Scenarios
Borderlands (2009)	**x**			
Diablo I (2000)		**x**		
Diablo II (2008)		**x**		x
Dwarf Fortress (2006)		**x**	x	x
Elder Scrolls IV: Oblivion (2007)	**x**			
Elder Scrolls V: Skyrim (2011)				x
Elite (1984)		**x**	x	x
EVE Online (2003)	**x**	**x**		x
Facade (2005)				
FreeCiv and Civilization IV (2004)		**x**		
Fuel (2009)		**x**		
Gears of War 2 (2008)	**x**			
Left4Dead (2008)				x
.kkrieger (2004)	**x**			
Minecraft (2009)		**x**	x	
Noctis (2002)		**x**		
RoboBlitz (2006)	**x**			
Realm of the Mad God (2010)	**x**			
Rogue (1980)		**x**		x
Spelunky (2008)	**x**	**x**		x
Spore (2008)	**x**	**x**		
Torchlight (2009)		**x**		
X-Com: UFO Defense (1994)		**x**		

definitions offer some level of procedural involvement, however, for the area of computer vision, we have refined the 4 definitions down to *game space* and *game bits*.

PCG in its current state can offer large time savings in the design process in games, either as an assistive tool, or a replacement for the way content is produced. Once methods to generate entities within games are created they can be used countless times in different environments. Perlin noise is a fantastic diverse tool for PCG in which it has been adapted with fractional Brownian motion [48] to create highly realistic and detailed fractal terrains in many large scale modern games. PCG becomes far more useful as hardware increases in power, a study suggests that as graphical improvements occur the state of the art will become the benchmark for all games created then on, making games with lesser graphics harder to market [49]. This will be a space that perhaps generative methods can fill, allowing the creation of more tedious facets of graphical design to be created with a few lines of code rather than a few hours of design. Details that exist within games that are non-interactive and non-intrusive, such as grass or small foliage is a perfect example of the content that can be generated through autonomous methods with little impact on user play, whereas trees and landscapes should be more carefully generated as these become the interactive elements within the game.

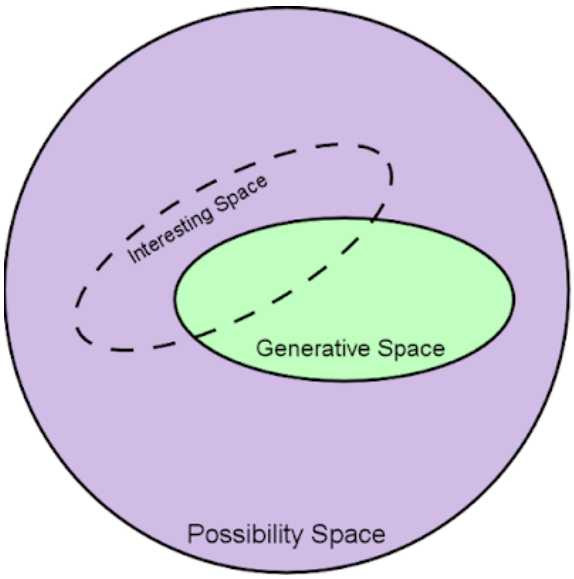


Figure 4.1: A very basic plot of the sheer amount of available possibility space for games, with the generation space taking up a smaller area of that, but the "interesting" generation space only slightly overlapping this generative space. Not all content generated procedurally is deemed interesting by the end user.

As discussed previously No Man's Sky is perhaps one of the more popular recent

games to rely heavily on PCG within its engine, with even interactive elements (creatures, trees, resources) generated through controlled procedural tools. Though a magnificent tool in that it can generate a vast amount of detailed worlds in an endless universe, a large amount of criticism was produced amongst a high percentage of the player base calling for less "repetitive" gameplay environments - edging to the sheer amount of interactive elements that were generated randomly. This type of problem can be visualised in Figure 4.1, where the possible generations within the game are near endless, but the user may find only a small amount of these combinations "interesting".

PCG has also seen use in filmography, with large scale multi-agent scenes sometimes being generated through PCG techniques. Films such as The Lord of the Rings, World War Z and I Robot have utilised an engine called MASSIVE. This engine uses state of the art PCG methods that are also used in games to render an array of unique background "actors".

4.1.1 Purpose

PCG has an obvious purpose of creating sheer amounts of content, but to what end does this contribute to a better environment?

Game content is a pivotal part of keeping people engaged with a virtual world [5]. Therefore which types of computer-generated content are used could massively and potentially adversely affect peoples interest when interacting with games content.

It is apparent that content production is becoming more expensive in terms of time and skill investments. Modern AAA title games can take several years to even close to a decade to develop, gameplay and other architecture designs are often not the bottlenecks for the time investment, more often than not it is the generation of high-quality content [50].

Therefore, given games increase in time investments for content creation, and the defining factor that content is directly correlated with engagement, PCG must be used in the future of game development to some degree.

In [5], the authors outline 6 main classes that are generated procedurally, with a

Table 4.2: Table from [5], their interpretation of a game environment split in to 6 main classes, each with their own generative requirements and rigour. Highlighted in bold are the 2 main classes that this thesis investigates in further detail.

Derived Content	News	Leaderboards
Game Design	System Design	World Design
Game Scenarios	Puzzles	Story
Game Systems	Roads	Entity Behavior
Game Space	Indoor/terrain maps	Bodies of Water
Game Bits	Textures	Sound

hierarchy of the quantity of each class. This can be seen in the table below, with most content generated at the bottom, to least at the top.

4.1.2 Algorithms

It is apparent that PCG is used in nearly every game to some extent, with varying degrees of effectiveness. In this section, we will discuss the various algorithms that have been used in video games and film. We will be focusing on three data types (as seen in Table **??**) that form up a vast majority of virtual environments; Terrain generation, which is the backbone of most outdoor worlds. Texture generation, is necessary to add another level of artistic design and realism. Lastly, 3D model generation for interactable humanoid models and other various 3D objects that could appear in a scene.

This section will focus on the background of non-deep learning approaches, in order to understand the properties that make these algorithms good at generating **specific** content. We will later discuss the various deep learning approaches focused on a more **generalised** theme.

Automating Terrain Generation

The terrain is an important part of any virtual environment, the ground beneath the player's feet is almost a necessity in any environment that exhibits gravity. Terrain has been generated via countless methods over the years, whether by hand or automation, the latter brings a great deal of research to how realistic terrain is generated. One of the most common methods of storing terrain data is using a height or terrain map. This implies a greyscale, or colour, image that represents a height at each pixel.

At the very basic level terrain can be created with tools that paint on varying intensity of greyscale values onto an empty image or grid. These tools are common on platforms like Unity and Unreal engine and provide a very quick way of prototyping terrain, however, lack the property of seamless-infinite terrain, but can be powerful when generating content that needs to follow a very specific constraint or layout.

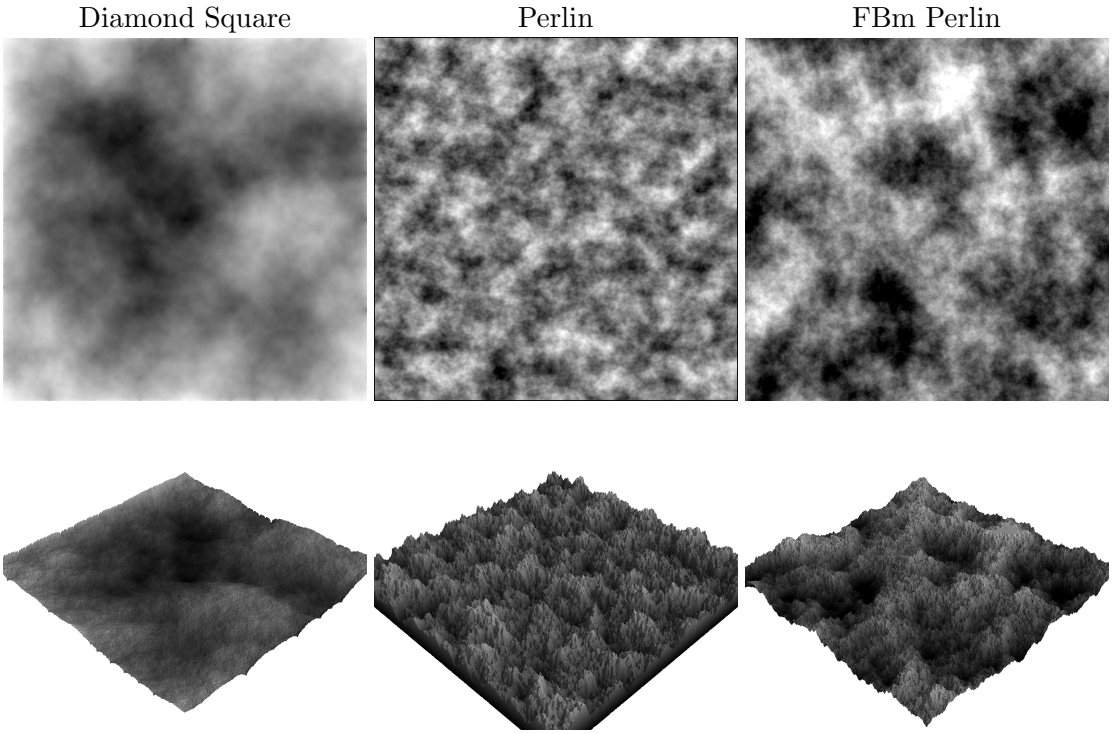


Figure 4.2: Diamond Square, Perlin, and FBm Perlin

Automating terrain generation through the use of algorithms is an extremely attractive design choice. Terrain can be condensed purely into a single algorithm that can represent a nearly endless amount of terrain data with very little overhead of memory or storage. This data can either be generated online or offline, an online generation being done in real-time as a user is traversing an environment, whereas offline could be preloaded either before the user enters the environment or generated and stored on a disk.

Two of the earliest terrain generation methods are diamond-square [51] and Perlin noise [52].

Diamond-square is a method of using midpoint-displacement over multiple iterations to create a fractal. The diamond-square algorithm uses a simple method of midpoint displacement, imaging this one a 1D line which has points 1,2,3,4,5 we would iteratively take two points, starting at either end of the line P1 and P5, and take an average of

these points with an added random to offset and apply this to the middle point in the set, P3. We would then iterate to the next set of points P1 and P3 and P3 and P5. This would occur until all points have been updated. The number of passes can also be used to determine how rigid the terrain will look.

Perlin noise is an extremely well known and common algorithm for exploiting realistic terrain generation, which exhibits random properties and can be completely seamless over a near infinite length.

Perlin noise was developed by Ken Perlin (cite) in 1983. Predominantly used for terrain generation in CGI and games in the early '90s to the present day. This technique is a type of gradient noise, essentially creating a smooth wave between two random points using an interpolation function.

Combining multiple single layers of noise with a parameterised function controlling the wave frequencies and amplitudes can help create very realistic textures, notably fractal pseudo-random texture maps. This is commonly known as fractional Brownian motion (fBm) [53]. These generated signal lines on their own may look smooth or rough, but adding multiple of these together creates a combination of all previous signals into one signal that appears to mimic real terrain structures.

Though Perlin noise is used heavily in terrain, it is not solely limited to terrains, and applications in many other texture domains such as clouds, fire and textures for objects have shown very strong visual results.

An approach that builds on the non-deterministic gradient-based noise is an interactive approach [54], where the author generates terrain landscapes from a series of drawn curves, spine and silhouettes. These regions are then populated with height values through noise propagation. The approach shows more active control over user-centred design compared to the previously discussed techniques. Perlin noise was later improved to simplex noise, which can be used with 3D (and beyond) space coordinates.

Many games employ some sort of noise generation for their terrain, mainly games that rely on procedural generation for endless or iterative worlds. For example, Minecraft uses a Perlin noise fBm generation style to create multiple biomes that exhibit lots of different mimicked real-world properties. Rust is another game that uses fractal noise to

create realistic mountain ranges and valleys. Also No Mans Sky, a game that offered a near-infinite number of fully procedural planets with procedurally generated terrain (along with a lot of other generated assets).

4.1.3 Automating Texture Synthesis

Texture generation exhibits very similar properties to that of terrain map generation, however, whereas terrains typically need to show seamless and fractal properties, texture map algorithms do not necessarily need to follow this same design.

Texture mapping has been very common since 1974 when Edwin Catmull originally created a method of mapping 2D textures to 3D objects [55]. Two-dimensional images are the most common forms of storing this type of data, and by using UV coordinates, which maps 2D pixel space onto each corresponding vertex on a 3D object, the colour data can be applied to the model. This can then be interpolated in the vertex shader to determine how the colour should be distributed amongst the primitive.

Textures can also be used for various other visual effects in virtual environments, such as generating fire or clouds etc. Using techniques like Perlin noise with an incremental variable such as time allows the pseudo animation of more complex texture structures to create moving effects within the textures. In Figure 4.3 an example taken from a rendered planet that uses Perlin noise on a low detailed sphere above the planet, using the Perlin value for the alpha value of the vertex, an illusion of partially covered clouds can be made. A time variable is then multiplied by the Perlin positions to create multiple cloud shapes that appear to be moving, when in reality the alpha value is changing over a fully rendered sphere.

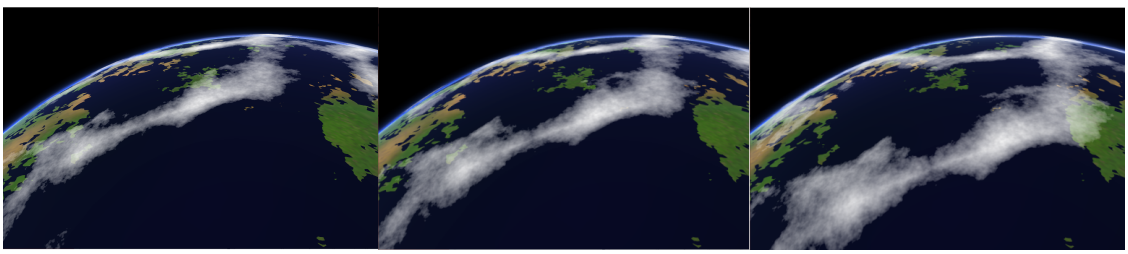


Figure 4.3: Example of PCG in a virtual world, Perlin noise is mapped onto a large sphere, the Perlin noise positional input is multiplied by a time T variable. Since Perlin noise is a smooth gradient function the effect mimics clouds rolling over the sky as time T increases.

Over the years we have seen more and more information associated with the original simple pixel-colour texture map, such as bump maps, normal maps, reflection maps etc. This has led to significantly more complex algorithms for the generation of each of these maps programmatically, though like terrain maps, it still remains a common choice to design these types of textures by hand, especially in environments where specific details are paramount. However in more generalized environments, where procedural generation can be used, we see a great shift towards the use of some sort of generation algorithm to either create more varied environments or as an assistance (or replacement) tool to reduce the pressure on texture artists.

Texture generation has been a long researched field, but not only in computer vision and stats. Work from Julesz in 1981 [56], which investigated the psychological observations of textures, showed that two textures would be perceived as identical if certain statistical distributions were the same across these textures. This work led to a great deal of research on the statistical properties of textures, and how these properties can be generated programmatically.

Probably the most impactful research on texture synthesis came as recently as 1999; Texture Synthesis by Non-parametric Sampling is one of the most highly cited non-deep learning-based texture synthesis methods [57]. Using a pixel-based approach this non-parametric model search-based approach picks random pixels at a time from an input texture and then attempts to find all neighbourhoods that exhibit similar properties to the neighbourhood of the chosen pixel, the random sample is then added to the synthesised texture. This work shows fantastic results for textures that exhibit strong global structures when using a large window for the neighbouring regions, however, fails in particular areas when using small window sizes, or the input texture is particularly small, as well as if the algorithm gets stuck in a particular neighbourhood of the input, the resulting texture would be garbage.

In 2000 work on A Parametric Texture Model Based [58] used pairwise statistical constraints. This approach starts with a set of random noise and uses some optimization procedure manipulating the noise into having a set of constraints from a statistical pool of properties to an original input texture, leading to, at the time, state of the art results for texture synthesis. In the event an input texture required uniformity i.e. a brick wall,

the method would struggle to keep the inhomogeneous properties, and the texture would collapse. Although local features would be captured, at a higher level the texture would appear extremely unordered and lack any sense of coherence.

Image inpainting is a slightly broad definition that covers repairing or altering images that may be missing areas or have imperfections, into a complete image. Given the repetitive nature of texture, this idea can be applied to the texture synthesis domain in digital inpainting, but instead of replacing missing information, a small sample of a texture can be used to generate far more with slight variations.

An example of such is Image Quilting 2001 [2], this algorithm uses a small sample of an input texture and fills a larger texture with overlapping blocks from the original, for each location the input texture is searched for a block that satisfies the overlap between the left and upper block. The minimum error boundary cut is calculated between overlapping neighbouring blocks at which point that block is added back onto the texture. This work remained pivotal in the field until the emergence of convolutional neural network architectures that are used to solve texture synthesis to higher visual fidelity.

Image Quilting not only creates textures but can be used to transfer textures. This is a very simple modification of the algorithm to allow the error to blend the old overlapping error and a new correspondence map. This map defines a coarse outline for the quilting algorithm to follow, creating a new texture with properties of the original texture, in a coarse outline of the correspondence map.

Although researched in 2001, Image Quilting is still one of the most straightforward and robust texture generation methods when given an original sample.

Though some other methods are Pixel-based algorithms, at a fundamental level are extremely straightforward, and use a pixel to pixel type mapping to find similar neighbourhoods of pixels to create a synthetic texture, these methods have also been used frequently for image inpainting. Some of which include using Markov fields [59] and image analogies [60].

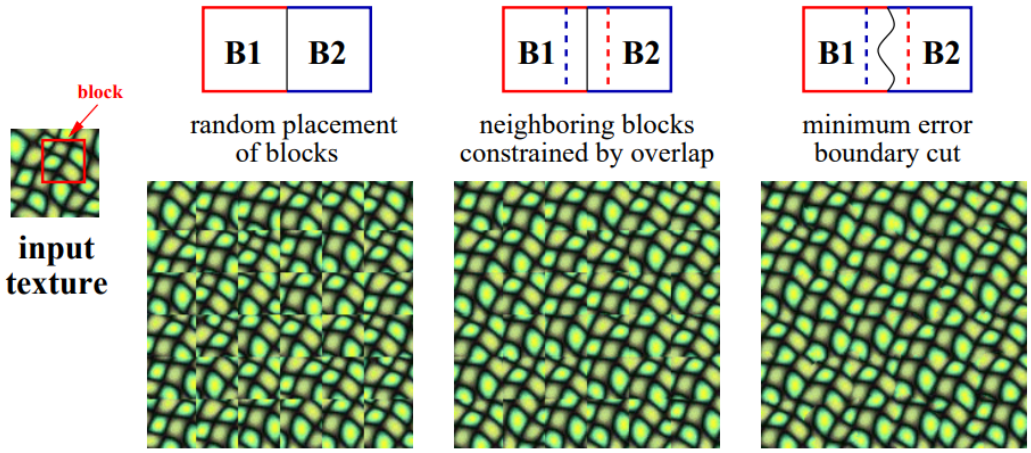


Figure 4.4: Image Quilting technique, Figure from [2] Quilting is a method of texture generation, sampling from original input textures to create a larger variant texture that has very similar patterns to the original.

4.1.4 3D Model Generation (Humanoid)

3D model generation is perhaps one of the larger areas of research out of the three topics covered so far, it ranges from the creation of mesh data, voxels data, point clouds, but the fundamental idea is the creation of coherent surfaces in a 3D space. There are typically three ways to create 3D models, scanning using 3D scanners or Lidar, creating them using some 3D modelling software, or to fully generate the model using a procedural method. This section will be focusing on the procedural methods of generating models.

Unlike modern deep learning approaches, mostly all of the research and literature covers a very specific topic of model generation. For example, we see a large branch of work on facial reconstructions, and a separate branch using different methods on pose reconstruction. This makes this section more of a wild west when compared to the very broad and generalisable deep learning methods later on.

3D models were first designed in the mid-1960s. Ivan Sutherland pioneered the creation of Sketchpad which began as a 2D drawing tool but quickly led to help create the first 3D models in the late 1960s [61].

Quite a while later in 1982 [62], a complex method of 3D object synthesis using patch intersections was proposed. Using preexisting shapes, synthesis could occur by combining these shapes and applying an intersectional algorithm. It checks each patch of each shape

on a patch by patch basis, the subdivision of patches describes a new shape which is a 3D model synthesised from both input models. This technique was extremely advanced for the year of release and is still cited in much modern modelling research today, but more importantly, led to a large collection of research in the 1980s and 1990s on surface synthesis.

Geometric hierarchy is something that is quite common in 3D model synthesis, the idea of using pre-existing 3D models and synthesising more using geometrically dissimilar, but functional connections. Shapes and 3D models can then be generated from a binary-tree like structure, which has a coarse to fine type representation. This representation is said to have a far more detailed understanding of the functionality of each shape's part while detracting from the explicit geometric detail [63].

This idea was proposed in the late 1990's [64, 65]. However more recently this application was shown to greater effect through the use of a co analysis of the representations of the shapes using a separate algorithm, this alleviated the presence of the typical large variability between parts in the shape set they used, leading to the authors producing results from geometrically dissimilar data, whilst assembling parts of these shapes into coherent structures.

There are commonly two types of generation methods, ones that incorporate rigging within the generation, and those that don't. Rigging is used in animation and simply keeps track of each component (body part) of the mesh. These can then be moved to simulate the animated movement of the human.

To summarise, the 3D modelling field prior to deep learning was extremely broad and offered no "one" solution fits all to the task of synthesising or generating new data. The solutions to specific problems are typical of high quality algorithmic based approaches, and where they provide solid results, require to be created with a very niche use case.

4.2 Deep Learning in PCG

4.2.1 Background and Applications

Recently, procedural content generation has seen a broad trend towards the use of deep learning. These prior specific PCG algorithms are quickly being replaced by the state of the art synthesis and generation networks that are assisting or replacing asset generation. This is mainly due to the generalised application of neural networks, and their ability to learn from small or large data sets. The feature-rich content of prior hand-generated or labelled data can be leveraged to create larger and more varied data by training neural networks.

Architectures such as fully convolutional neural networks have vastly changed the computer vision domain when applied to images, seeing classification and generation improvements [13]. Graph convolutional networks [66] are becoming more common and are starting to replace methods of generating 3D shapes. This type of network offers endless potential, due to the complex representations of shapes using their points as nodes, and neighbouring connections as edges. Graphs can be used to represent a vast amount of data formats, such as social networks, financial graphs and many other research areas. There are currently a vast amount of other network variations seeing a shift in other data domains, such as audio, data classification, finance and medical imaging. This section will explore where deep learning currently appears in PCG research and the potential areas that have not been explored in order to fill in gaps in the research field.

It is apparent that these large advances in computer vision are starting to trickle down into the games industry, and in particular procedural generation of content. This section will focus on the current research field, and explore any potential areas of interest for the application of modern deep learning techniques to age-old PCG problems.

In Procedural Content Generation via Machine Learning (PCGML) [67] the authors set out to survey the field, and the use of Machine Learning as a more broad term in PCG, compared to predecessor algorithms, such as search-based, noise-based or constructive methods. The paper itself outlines five main focal points of using ML; "Autonomous Generation, Co-creative and Mixed-Initiative Design, Repair, Critique and Analysis, and

Data Compression". In the next content chapters, the work of this thesis will be focused on data compression and autonomous design.

Later in 2020, we see another exploratory study investigating the increased use of deep learning in PCG [68], this paper shows more modern applications of PCG in games.

A method such as Latent Variable Evolution which combines a typical search based solution with some sort of generative solution such as a GAN or VAE, aims to search through a list of solutions from a network and find one that best matches a set of prior criteria.

We see a broad range of applications of generative deep learning for generating content procedurally, from audio to textures and 3D models. The possibilities of which domains deep learning has been applied to lies in the data availability.

In Adversarial Audio Synthesis [69] we see a drastic shift from the traditional application of GANs in the image domain to the audio domain. This work was more contextually applied to specific types of noise that would be represented in video games. In Synthesising Knocking Sound Effects Using Conditional WaveGAN the author trains an audio-based GAN with conditional classes, creating a deterministic method of generating variable, but highly realistic, sound effects. The author conducted multiple questionnaires finding that the audio samples were indistinguishable from their trained counterparts, however, specific members of the questionnaire who identified as skilled in the audio field were able to determine real from fake.

In a work titled Texture Synthesis Using Convolutional Neural Networks [70] the author uses the inherent learned properties of the convolutional neural network's feature space after training. During training, the features of the hidden layers learn to represent the natural images passed in. Though this process purely focuses on the discriminatory outputs and, as such each new single texture requires a full optimization of the network to minimize loss and create high-quality outputs. The feature maps are combined from multiple hidden layers to achieve less abstract representations; i.e. conv1, pool1, conv2 etc. The method uses a pre-trained network (VGG19) and by training on a specified input image, the feature extraction is performed as in any convNet. Lastly, with an input of the same dimensionality initialized with white noise, gradient descent can be performed

to find the lowest point of loss between it and the learned features.

Data compression is perhaps the origin of generating content procedurally, in a time where data storage was expensive, and often slow to read in large quantities. Even in modern times PCG is still used to save storage space, and in certain games create a practically infinite environment. Typically search-based and noise-based algorithms do not take up a lot of space but require computational resources when used to create new content on the fly. Wherewith deep learning algorithms we see an opposite trend, where networks can be bulky and expensive to train, however once trained the generation time for specific content creation can be minuscule in comparison. Using the data's underlying distribution the network can compress a vast amount of input data into a much more compact representation.

Early iterations of content generation using deep learning or GANs witnessed a difficulty with control of content, and although clever methods of controlling latent space were used post-training we lacked a way of generating class-based content. Recently conditional GANs are becoming more predominately used in generation tasks where labels exist for data.

Conditional GANs can create a lack of quality when there is a class sparsity, in which there is too little data to saturate a classes' weights within the network while keeping generations generalised as opposed to overfitting.

4.3 Summary

The work discussed in this literature chapter opens the door for the rest of the thesis. The field of deep learning in computer vision and PCG is exploding at a highly competitive rate, we wish to cement some novel ideas and methods in the area which can be conceivably built on in the future. We have touched on three main areas of PCG which we will explore further throughout the rest of the thesis, the narrative will follow improvement using deep learning for 3D models, Texture Synthesis and Height Map Data. These are the important novel contributions that we will be investigating deeper in to the thesis.

GANs and their application to the PCG field have thrown the door wide open, show-

ing improvements in nearly every area of generating content from images and music to 3D models. Their rapid learning of pre-defined data distribution has created an unparalleled generalised generation process. This literature chapter first discussed the methods used currently for very specific generations using mainly non-deep learning methods. We then explore how deep learning can be used to circumvent the need for specific algorithms for each task, and provide a far more generalised approach to PCG.

Chapter 5

Generation of 2D Content

In this chapter, we will discuss the different approaches of both 2D terrain map and texture generation with respect to game environments. We will compare different approaches both quantitatively and qualitatively, concerning the prior state of the art generation methods.

The motivation behind this work is the need for a more robust and targeted terrain and texture generation process, compared to noise based algorithms, which are iteratively changed with little control over the outcome. We also want to understand the reasoning behind algorithm choice and which methods perform more favorably, this will also aim to answer multiple meta-questions on individuals' enjoyment of terrain in virtual environments.

The overall benefit of this research should improve the future of terrain generation methods, with respect to deep learning, and the massive amounts of accessible data which can be readily trained.

In this chapter we discuss a real application of texture synthesis networks in order to create a process of generating large scale hand drawn art assets for a leading games company.

Finally, we explore the various deep learning networks used to learn real world terrain data to create high quality procedural generations of the target terrain. A small scale questionnaire is conducted in order to gauge how individuals respond to the varying algorithms discussed in this chapter, both to satisfy the need for justification of network

performance, but also to acquire more insight into what makes a "good" terrain. This questionnaire is later extended in Chapter 6 to build upon some inconsistencies and missed data collection opportunities.

We decided to focus on procedural terrain as a research area due to prior experience working with rendering of large scale terrains. The area was fairly stale, with repeatable perlin based noise generation, that made a good attempt to mimic real world terrain, but lacked a great deal of key structural similarities. An example of my past experiments with large scale terrain can be seen in Figure 5.1 where I rendered a large scale planet using fully procedural terrain using noise based techniques, however I noticed a lack of control-ability for the entire planet, and became aware of the repeating features even when using multiple noise layers.

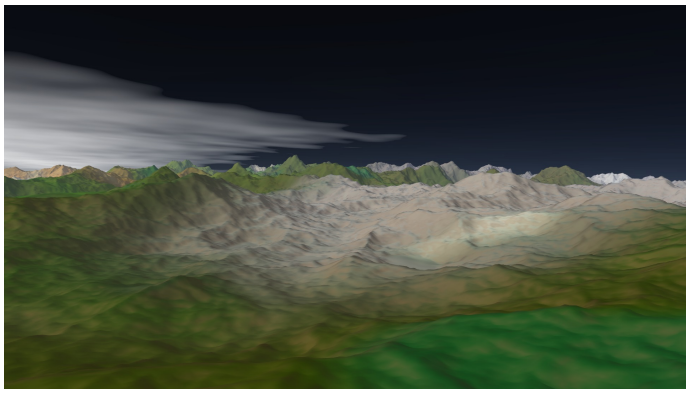


Figure 5.1: Render of work done for my BSc. Large scale planet rendering, showing how strong perlin noise can be, but also the repeating features, lack of inherent control and inconsistencies with real world terrain.

Further vocalisation made by industry talks, at events like COG and the IGGI Conference over several years only added to my want to investigate more interactive approaches to generating terrain.

Lastly, a placement opportunity arose during my PhD which lead to a collaboration on a real-world problem in the games industry, and that is the need for extremely high quality procedural texturing. The recent advancement to 4k resolution for most games means texture quality has to be better than ever, but artists' time has not become any less important, therefore this placement was to determine ways of semi-supervised approaches of generating the high quality textures used in their games.

5.1 General Texture and Hand-Drawn Texture Generation

5.1.1 Industry Partnership, Texture Synthesis and Real World Use

During my PhD, I undertook a supervised but fairly loose partnership with a company, Revolution Games. This was not only necessary for the PhD programme I was pursuing, but also a fantastic opportunity to incorporate all research up to that point into a real-world application of generative design for procedural content.

During this time the research goal was to create a robust method using some deep learning techniques to take highly specific and handcrafted artwork textures and create some automated pipeline to recreate these samples pragmatically. The initial goal was to survey the field for research on texture synthesis for very high-quality outputs, with the emphasis on speed, low cost, and perhaps most importantly, the ability to be used by someone with little to no domain knowledge. Therefore any preprocessing must be kept to a minimum, and where used, must be reproducible broadly speaking for any type of texture.

5.1.2 Introduction

In this Section, we will discuss the use of deep learning and generative methods to create textures. Although quite broad, this will be narrowed down to a discussion on a leading industry partner collaboration to create high-quality hand-drawn textures.

Although extremely similar to height map and elevation data, textures exhibit some fundamental differences; textures can be either uniform like a brick wall where each brick is uniformly positioned and could have the same texture at a lower level. Or non-uniform, similar to that of height data, where each feature of the texture exhibits a new or non-repeating pattern. The latter is far more difficult to accomplish but is far more visually pleasing.

Uniform textures are quite easy to create, and many different methods have been used to create wrapped textures that are formed from one ”starting block” texture.

However generating non-uniform textures from scratch, or from a reference point remains a challenge and a constantly researched area. This type of texture is what this section will focus on, and how deep learning and GAN approaches have performed exceeding well in all domains of texture generation.

The most crucial component of texture generation, excluding the obvious need for high quality, is the reduction of periodicity, a term that simply means how often repetition occurs in a domain. In the context of textures, this could be repeating features such as a brick that is identical to an adjacent brick in a wall texture.

We initially investigated current cutting edge deep learning-focused methods of generating textures, the two appearing most in the literature were using Variational AutoEncoders (VAEs) and Generative Adversarial Networks (GANs). Both of these had advantages and disadvantages, the main application of GANs is to generate new and unique data from the average data distributions of any given data, whereas VAEs would attempt to reconstruct data, but not generate new data.

Typically VAEs require a large amount of data to start to be functional with the use of latent space arithmetic which could have shown promising results for controlling output types, however as we only had 1 or 2 high-quality images for each domain, we opted against using this type of architecture. VAEs would also not be showing any variant generations from the original data distributions.

Whereas GANs are explicitly generating new data through the dynamic of indirect learning of data, given how densely detailed the data we were dealing with was, we opted to look deeper into the use of GANs.

There is a massive range of GANs, slightly altered in each piece of literature they appear in. Each one of these architectural designs is more useful for particular data. Our initial experiments were conducted using a DCGAN on the floorboard texture provided to us. The GAN was altered to take in small sub-sampled patches of the overall large texture file and attempts to learn the features of each subsection. The results were promising, with the highly detailed texture's features being represented in the generative network, allowing for generations of the training data which resemble some of the features of the floorboard texture. However, the generations themselves lacked any structure across the

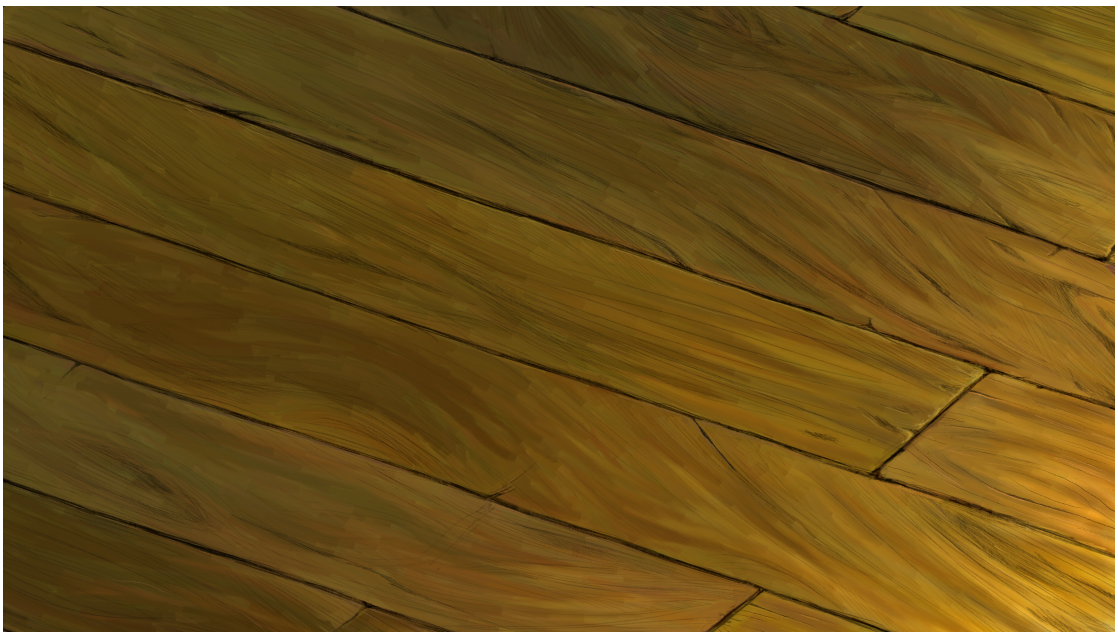


Figure 5.2: Example of a non pre-processed texture, requiring some pre-processing before being trained through a GAN

texture which meant that the cracks of the floorboard appeared all over the place and in no uniform way.

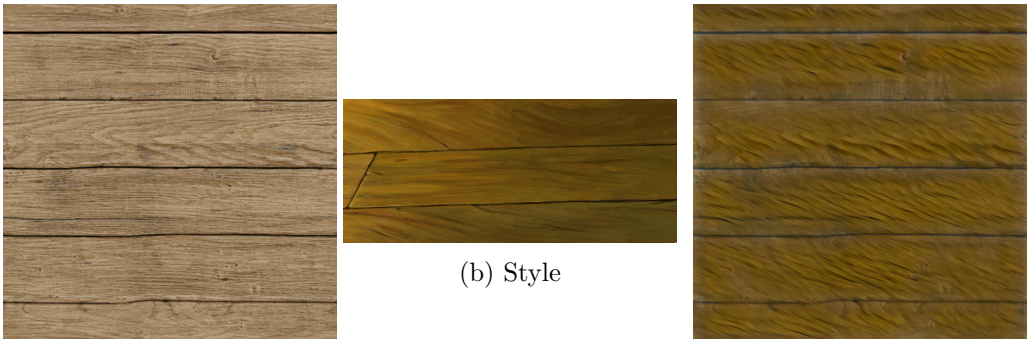
5.1.3 GAN Approach

The issue with an outright GAN approach is the need for pre-processing of data, in the vast majority of cases data cannot simply be thrown into a GAN with the expectation of coherent outputs. That was particularly the case here, with data in its initial form being rotated, no uniformity and large overall features. An example of a texture given to us of a floorboard can be seen in Figure5.2. In this figure, you can see the floorboard texture, which is rotated but also contains floorboard cracks in a completely non-uniform structure. A GAN typically needs to learn through a uniform centralised texture or object, with features that are relatively small but also abundant.

Given what we have discussed it was apparent that some preprocessing was needed in order to convert the texture into some appearance that is useable. Figure 5.3 shows a three-step process that rotates and crops the initial image, and then an optional step of removing the cracks between the floorboards, leaving a high-quality stroke stylised image. This was performed using basic image editing software paint .net, with the need

(a) Original texture (b) Rotated and cropped (c) Rotated, cropped and floor cracks removed

Figure 5.3: An example of some of the high quality textures we had to play with. Each texture was a hand drawn asset with extremely high visual fidelity. Texture preprocessing was applied to the original texture (a) to better create a trainable texture for a GAN.



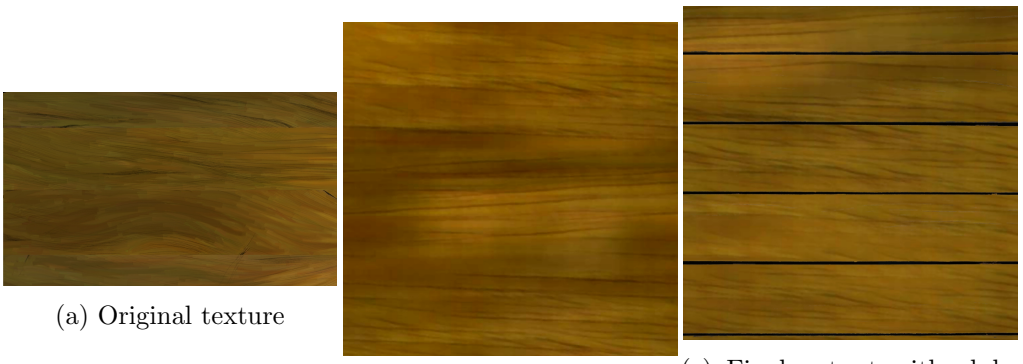
(a) Target (b) Style (c) Style Transferred

Figure 5.4: Style transfer used to transfer the style of the floorboard in figure (b) onto the content image of the figure (a) the result is a texture that has been optimized to blend both style and content into figure (c).

to satisfy the requirement of low domain knowledge for reproduction at a later date. The rotation was done in order to bring the floorboards to a more horizontal alignment, and the cropping was done to ensure the texture remained rectangular. Finally, this optional step used the paint .net magic wand tool to select the distinct black cracks and simply delete them, merging the texture back together.

Figure 5.4 ran into issues regarding the target style, since we were transferring the style onto a texture that already contained cracks in the wood, artefacts could be observed in the final output. This was not high enough visual fidelity to be used in a game, and on top of this affected the optimization step of the style transfer network.

Style transfer offers a unique approach of applying a set style from one texture to another content texture. We applied a basic neural style transfer model initially to determine how well this type of data would perform, given distinctive style-images that are usually used in this type of network, we were uncertain of how it would perform. However, the results from style transfer were visually good and ticked the boxes for the



(a) Original texture

(b) GAN output

(c) Final output with alpha mask cracks added

Figure 5.5: Generated sample using a pre-processed texture, trained through a GAN to generate varied samples. The large GAN output could then have cracks drawn on, or in our case a pre-made texture with cracks extracted through a binary colour bound calculation.

necessary requirements, however, the additional requirement for a target texture that would need to be produced complicated the automation process, and would require artists to produce just as many assets.

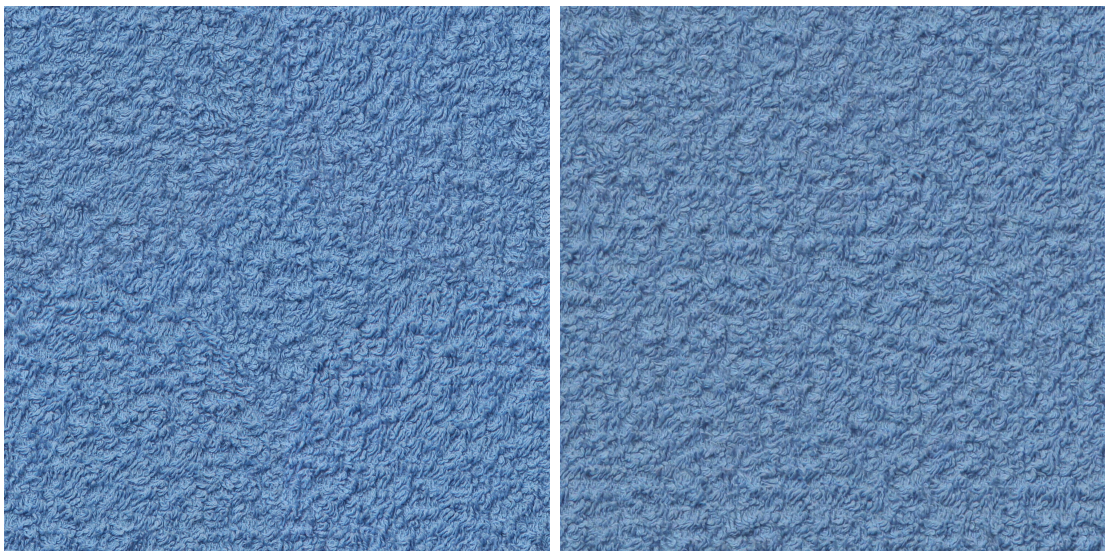
Later experiments were performed using a DCGAN. Our thoughts were, instead of generating a texture using a style, we learn the entire style manifold and then apply some alpha masked crack image on top of the large generated texture.

This can be seen in Figure 5.5, using a very crude outline of a floorboard which are just horizontal lines extracted from a texture online, we can mask over the large texture manifold and produce a realistic floorboard in the exact style that the artists use in their game. This could potentially save hours of time when compared to their pipeline before of hand painting each instance of a floorboard (of which there are many!), this would instead be replaced by drawing the crack outlines and laying this over the generated texture.

Final Design Flow

We eventually came up with two main designs for commonly used textures given to us as part of the collaboration:

Floorboards were used in nearly every indoor scene in the games they produced, each room could have an entirely different style of a floorboard. An example of a typical scene created by the studio can be seen in Figure 5.7. In this scene, a large portion of the screen contains floorboard, the distance from the camera means that the high detail



(a) Original texture (b) GAN output

Figure 5.6: An example of a much more straightforward texture to train, an elaborate hand-drawn piece of fabric. Trained through a GAN to generate extremely high-quality variations of the original.

of the floorboard in this case isn't necessary, but the capability should be there.

To summarise the floorboard creation, the initial texture was processed to remove as many of the cracks as possible, leaving behind a hand-drawn texture that contained "wood" like material but with no floorboard cracks. We then used this large texture as training data for the GAN, this would internally subdivide the texture into randomly cropped chunks of size 256x256. Since the network was fully convolutional these crops provide the filters manageable information for the larger textures manifold . The fully trained network could print out large textures of up to size 4k x 4k, depending on the GPU memory offered. The final stage of this process was to add in an overall of floorboard cracks, for the examples we used were simple binary masked images of floorboards which would crudely extract the cracks from a pre-existing texture. However, artists could spend this time outlining the exact blueprint of floorboard design they want, and then simply paste in the generated texture.

The fabric was perhaps the easiest texture for our GAN to learn, the fractal-like features were straightforward for the filters of the network to learn. The inherent lack of overall structure needed for this type of texture also added to the ease of training, it did not matter whether local features were rotated or in which position they appeared in. The process for this type of texture was as simple as moving the texture into the corresponding

Figure 5.7: An example of a scene from the game Broken Sword 5, a typical indoor scene, with around 30% of the scene containing floorboard textures.

folder and hitting train. The time taken to train was considerably less than the previous textures. Similar to the floorboard we could output textures of around 4k x 4k. Like this texture, there are many other textures with the same properties used by the studio.

Bricks provided some limitations that meant the quality of outputs for the experiments using this texture were not sufficient quality to be used in any capacity for the game studio.

Limitations

The final process offers a robust method of generating two types of commonly used texture for Revolution Games, the design was adapted so that there is no requirement for knowledge of how the GAN works, and that texture can be fed into a single folder and run with a push of a button. However, there are some limitations, the lack of ability to control what features appear on the texture is apparent, and something that some sort of conditional or semantic element to the GAN could have assisted with, an idea of giving the user a "paintbrush control" to paint on certain features of the floorboard texture for example.

Another limitation is the type of texture, we have shown that we had difficulty

learning certain textures, such as the brick texture, which would not work with the process used on the floorboard such as removing the cracks and then adding them back in post-process.

Lastly, the network size and GPU necessity is probably the largest drawback and requires the studio to either build their own deep learning PC with additional GPU support or use some cloud GPU server to run the GAN on. Once the networks are trained on the textures required the generation is relatively simple and requires next to no time or GPU memory. We could however reduce the size and capacity of the network, in doing so would mean the network could be trained on relatively cheap hardware but could see a reduction in the quality, which is extremely important to this project.

Summary

The results shown here provide some good insight into the limitations of using generative deep learning in the real world. Whereas in research we have full control over the data, cleaning, outputs and time, in the real world it is not anywhere near as practical.

The client we worked with had very real constraints, on the practicalities of the generation methods, along with being of high enough quality to be directly used in a game, the network was also required to be able to be trained with minimal domain knowledge. Time and system spec requirements meant that instead of using large networks, and training them for substantial amounts of time to achieve greater detail, we had to revert to using small networks, such as they do in TinyGAN, where the results are still competitive, with a slight quality degradation, for the relative loss of computational power.

5.2 Terrain

5.2.1 Introduction

Terrain generation is a large part of game and virtual environment scene creation. It is usually the foundation of many game worlds in which the rest of the content is built on top of it. Our challenges are the need to create a generative network that can both support the

level of detail at a low level, but also the structural information at a higher level, piecing together the parts of terrain data which makes real-world terrain interesting. The process should be able to take in target height map data, and output procedural variations of the target. This could be either unsupervised or supervised.

Currently, heightmaps are largely generated via procedural noise-based systems such as Perlin noise [52], real-world height data [71] or a more algorithmic approach like the diamond square algorithm [51]. Perlin noise is arguably the most versatile at generating procedural height values, with the ability to continuously produce smooth transitions of height values at any point on a surface, though using Perlin noise entirely on its own can create non-realistic height maps, due to the repeating non-structured noise features. Fractal Brownian Motion (FBM) [72] is a method of controlling the intensity of differing noise layers, sampling and averaging points from Perlin noise over multiple levels of varying height and frequency creates a height map of increasingly detailed but less noticeable features. The diamond-square algorithm [51] avoids the need for a noise-based reliance. By averaging points continuously between certain seeded points on a grid, this method can produce highly detailed fractal terrains that can mimic mountain ranges within the real world. This type of algorithm can generate non-uniform distributed structures, though with an element of randomness within the seeded points, can be equally as difficult to control the resulting terrain outputs. None of these methods provides direct control over the ability to create variant procedural terrain based on a target objective without the need for human supervision.

Certain developers of games have expressed the need for additional tools for generating terrain. The team behind *Hellblade*, at a Keynote talk at the IGGI conference on games 2018 [73], expressed the need for a more interesting and interactive world map environments that can be generated with little input through follow up questions to their talk. Another example is an author of a highly detailed 2D map generation tool called *Here Dragons Abound* [74] expressing concerns with the level of "interesting" generations Perlin noise is capable of producing, with users soon becoming bored of the repetitive outputs that the noise-based algorithms can generate.

The ability to generate terrains of a specific type could provide artists with a base to add the structures and areas necessary for a game environment, while at a broader level

the generation tool could entirely replace creations of height maps through automated learning of regions that the creators want the game to exist within. The method proposed is a novel approach to generating heightmaps learnt through a generative deep learning model which can output similar regions to that which they were trained on, essentially creating an environment where users can specify what their procedurally generated terrains should resemble.

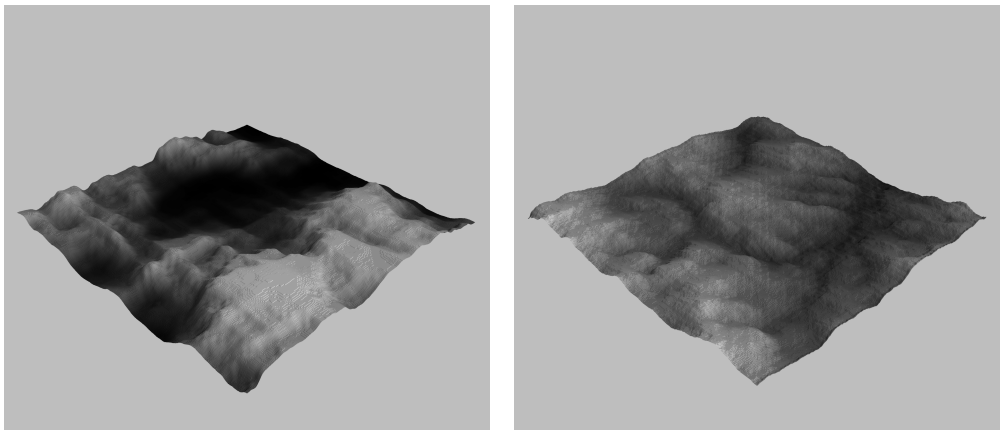
The network used for the results is built upon the Spatial GAN [75], a network designed for spatially independent feature representation for texture synthesis. This network shows comparisons to a popular Image Quilting technique [70] with results of arguable higher visual fidelity. We make some important modernisation adaptations to this network, offering an incremental improvement that incorporates decayed noise, hyper parameter optimization and adaptive learning rate.

5.2.2 Background

GANs have been discussed in literature Section 3.4, and the methods discussed in this section and further sections will build on that background. A similar task and the original idea of generating terrains using GANs from regions of the planet was first mentioned in [3], where the authors attempt to generate similar outputs from a data set from the Alps, utilizing multiple state of the art deep learning optimizations combined with a deep convolutional GAN (DCGAN) [27], a type of GAN built around the augmentation of convolutional layers in place of the multilayered perceptron [10] structure of the original GAN. With the filters of the convolutional layer producing a model focused more at spatial correlations, they show human interaction results with their output rendered height maps which show promising potential, though the inherent lack of upscaling from a DCGAN shows a limitation from a usability perspective. Furthermore, the terrain outputs produced appear fairly noisy and lack an amount of similarity to their derived regions, perhaps due to the oversaturation of varied data points used.

Observing their results, we found that when using multiple small 32x32 inputs of a vast array of real world satellite data, the outputs more closely resembled that of Fractal Brownian Motion noise. The DCGAN output contains a large number of dips and troughs

with a seemingly repeating pattern across larger generations. However as per their initial experiment the detail of the generations were simply too low to be used in any meaningful context, alongside the distinct lack of output size, being 32x32. Although it would be possible to apply some interpolation between height values, the result would still be a low quality blocky terrain render. Figure 5.8 shows a DCGAN render along side a BigGAN generation rendered.



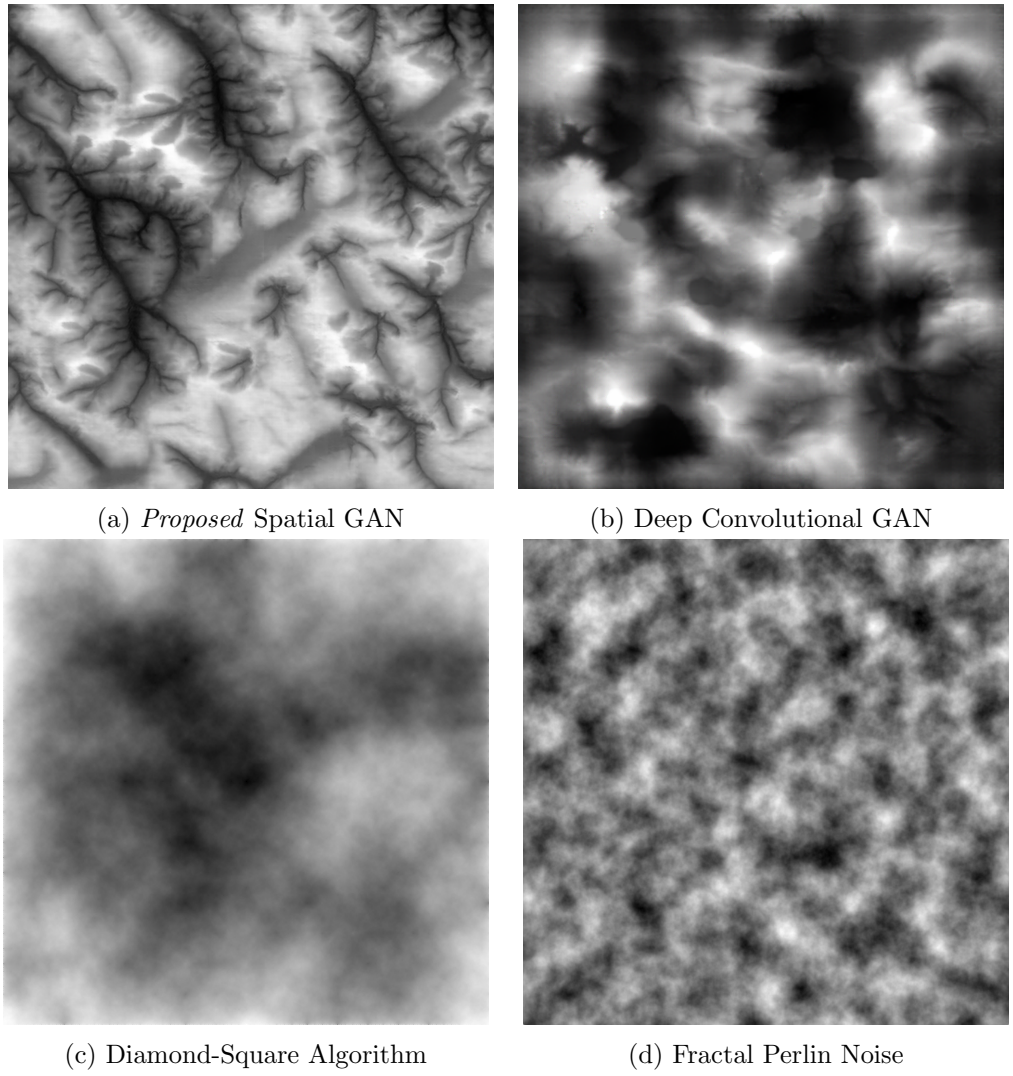
(a) DCGAN Generation Render (b) BigGAN Generation Render

Figure 5.8: Reproduced experiment from [3], included BigGAN as a proposed improvement to the smaller and lower resolution DCGAN. The BigGAN offers an improved feature representation, this can be seen by the connected valleys, compared to that of the DCGAN which has practically no structural features.

We decided to rerun their experiment but using larger input patches, and a different network architecture. Their use of DCGAN was slightly outdated, and the patch size fo 32x32 somewhat small.

We opted to use BigGAN which acts similarly to that of a DCGAN but with much larger network capacity. We also increased the size of inputs from 32x32 to 256x256, our thoughts were that this would help satisfy the increased number of network weights. The BigGAN output can be seen in Figure 5.8 (b). Although the quality of the BigGAN is visually better than that of the DCGAN, it is obvious these methods provide limitations to the overall *quality* of the rendered height maps when compared to any other approach. The batch size of this experiment was 32 and we trained the network for 20 hours, until the visual appearances stopped changing. We follow BigGANs settings for filters and layer setup.

5.2.3 GAN Approach



(a) *Proposed* Spatial GAN (b) Deep Convolutional GAN

(c) Diamond-Square Algorithm (d) Fractal Perlin Noise

Figure 5.9: The proposed method using Spatial GANs (a) vs. other common approaches (b,c,d). The comparison shows a large improvement on the learning structure of the SGAN over the previously attempted DCGAN results - which was trained on similar elevation data, while providing a contender to the predominantly used fractal Perlin noise, with structures learned directly from real terrain.

Heightmap generation is currently a fairly stagnant topic with the majority of generation using Perlin noise which forms a reliable, but sometimes repetitive output. In this Chapter, a method of generating height maps from real-world digital elevation data taken from specific regions of the planet is proposed. Raw elevation data sourced from NASA's SRTM (30m) data set is transformed into a height map format, this data is then passed into a two type unsupervised model. The method uses a type of generative adversarial network to learn the spatially-invariant features within the input regions, producing a network model that can output an extensive amount of varying, but visually and structurally

similar height maps to that of the input regions.

Figure 7.3 shows a comparison of 4 different algorithms, with our proposed algorithm shown in (a). Importantly, there is no defined metric for use on data such as this to quantify the level of quality during comparison. Later we do use some more generic image similarity metrics such as MSE and SSIM, which will be discussed further. A lot of the early work was performed using subjective and domain knowledge to determine improvements, we had a clear defined goal of creating a method that could produce variations of a target and set out on that path. A second idea we had was to run a qualitative experiment to subjectively prove from the perspective of an end-user the preference of method.

With this experiment, we wanted to pick up on some of the inconsistencies with the DCGAN approach, while still maintaining a semblance of control of what a designer may want. We took many of the ideas from [3], however instead of using a large selection of small resolution satellite samples, we opted to use larger resolution inputs, then crop these during training. This allowed the network to learn interconnecting regions of the overall structure of a sample, this was particularly important for satellite data. Stemming from the artificial appearance of the initial DCGAN work, if we learned these interconnecting smaller regions, then the real-world structures would remain in our outputs.

An important consideration when creating terrain is the use of texturing to add to the aesthetic of the terrain. A computationally cheap method of texturing a terrain is using an algorithm that uses a combination of gradient slope and height values. Textures can be added to produce realistic yet structurally repetitive textures across the entire terrain in a straightforward manner. Terrain authors can also be used to manually "paint" terrains within 3D engines using various brush tools. We also extend this work to use real-world colour data from the derived region's satellite image data. With advances in satellite imagery, high detailed terrain maps can also be acquired from sources such as Copernicus Sentinel satellite or Google Earth. We can pinpoint the exact geo-coordinates to acquire a satellite map that matches the height data, essentially creating a texture map on top of the height map. This can be used as a texture map straight away, or simply as a colour reference pointer for additional texturing. With both elevation and texture data, a powerful data set of real-world image data can be obtained.

Recently in 2017, a novel framework was proposed for the use of Spatial GANs (SGAN) [75]. Spatial GANs modifies the DCGAN [27] to remove fully connected layers, this allows for the scalability of outputs to any size and has properties that enable learned features to be mapped translation-invariant on to outputs. The SGAN architecture also follows DCGANs by removing all pooling layers, replacing them with strided convolutions. The strides within the generator and discriminator networks inherently learn the up and down scaling respectively, with 1/2 stride in the generator and 2 stride in the discriminator (Stride size affects how far the kernel will move when calculating outputs - stride of 1/2 will double the output size, 2 will halve the output size etc.). The input noise distribution of an SGAN begins with multiple vectors of noise compared to a commonly used single vector, providing a much larger range of possible spatially-variant structures for generations of the network. Overall SGAN boasts an accurate texture specific solution with an extremely fast output time compared to the previous methods discussed, due to the simplicity of network design - removing the aforementioned fully connected layers.

Although height data is essentially a type of texture data, there is a much larger emphasis on individual feature space, whereas textures usually have some sort of repeating pattern, that doesn't necessarily resonate with height data, and something we would tend to want to avoid given the apparent number of repeating features in noise based generation methods. After initial experimentation the trained SGAN produced good results, but there was a lack of large feature connection. We therefore aimed to adapt the SGAN with a few small, but imperative improvements.

As discussed in the Chapter 3.1, improvements to training GANs are being published frequently, and as such many "older generation" GANs have some techniques that can be modernised to increase training stability, but more importantly to improve the generation and variability of results.

Increasing the generators filter sizes offered a much richer feature set, but increased memory usage throughout the GAN. Typically smaller filter sizes would reduce the amount of unrelated features learned, however in this case increasing from a 5x5 filter to a 7x7 filter allowed a slightly better understanding of the localised feature space. In the context of the terrain maps, this would be the peaks and valleys. We would have loved to increase the size of the generator filters further but simply did not have the hardware capacity to

do so, due to the size of filters affecting memory usage at a quadratic power.

Adding a type of exponential decayed Gaussian noise to every input to the discriminator is something that is commonly used to prevent the discriminator over training[44]. The use of Gaussian noise can be seen in Fig 5.10, where sigma is reduced over the course of training, to a value of zero such that there is no Gaussian noise left in the training data. This was slightly subjective on how long to apply this function for, but for the first 200 epochs visibly assisted in reducing both mode collapse and also some regions that were rich in varied features.

There are various other noise functions that could have been used, such as salter-pepper, Poisson and speckle noise to name a few. Gaussian noise however fits in with the narrative of reducing discriminator performance, whilst still maintaining a rough overall pixel distribution similar to that of the training data.

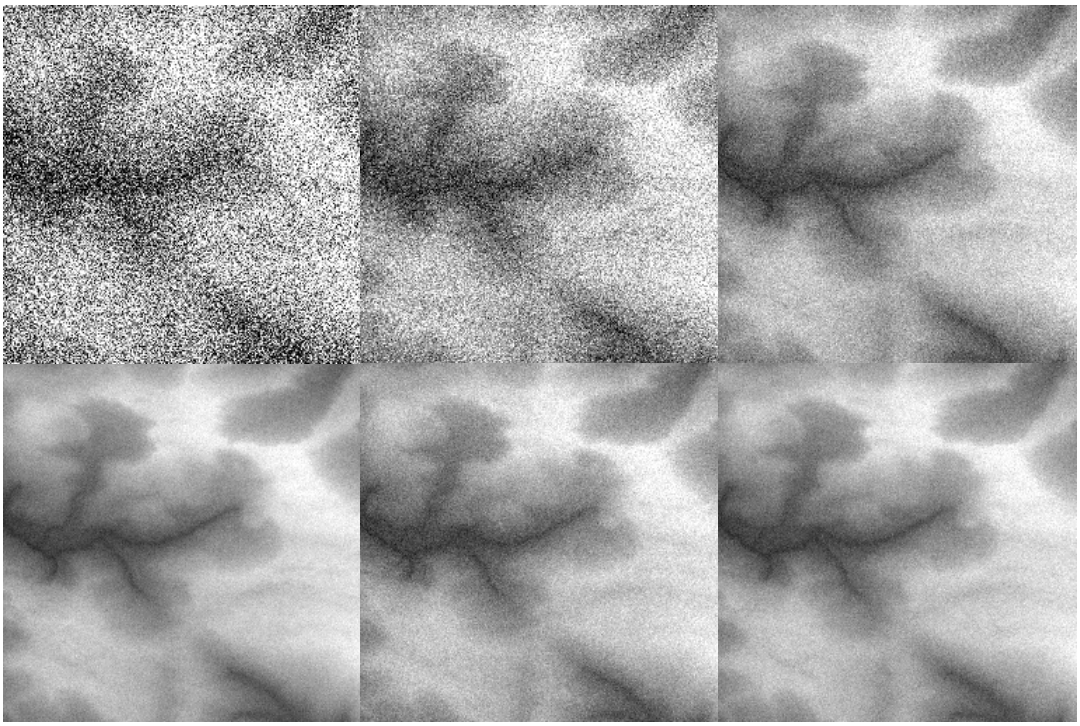


Figure 5.10: Example of a small 256x256 patch from a larger height map with decayed noise applied. Gaussian noise is used with a high sigma (100), decayed exponentially over 200 epochs until the sigma is less than 1, then the noise is removed entirely.

5.2.4 Data Collection and Pre-processing

The data was collected from an open source elevation map data set Shuttle Radar Topography Mission 30m (SRTM) courtesy of NASA [76]. This data provides a 30m to 1-pixel spatial resolution of the elevation topology of the planet, providing enough detail for the generative model used to learn the underlying features. Each region was approximately taken from a corresponding 100km^2 region. The data set itself provides the greatest resolution for real world terrain data that is publicly available.

Using the python library Elevation.py, a module for; accessing, downloading and processing SRTM elevation data, an automated tool was created to quickly process large areas of the planet given a rectangle area defined by longitude and latitude points, allowing the system to explore terrain patches for random locations of the planet, to selectively choose inputs to the network that are structurally interesting and different enough from other examples to train. This created easily manipulatable data which was banded from its raw continuous height data into a grayscale (1 colour channel) heightmap format, which was required as the input shape for the network.

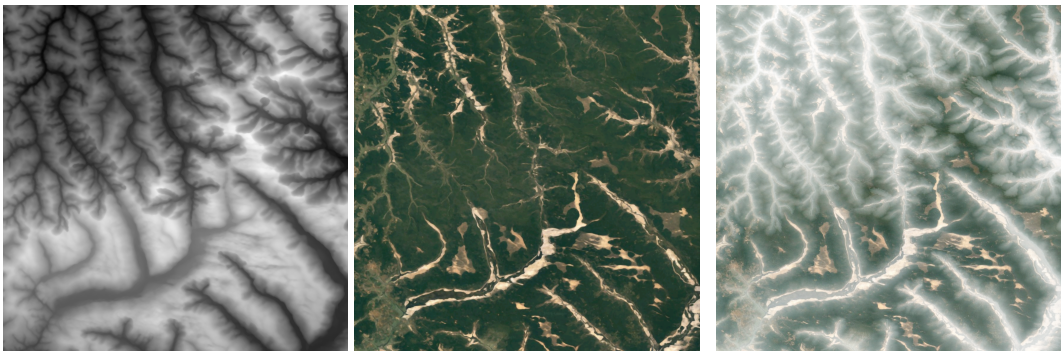
Data in its raw format was too large of a resolution for filters to learn the underlying structures of the height map. We trained the raw data initially and found that the generated outputs were very sparse and contained few structural features, we believe this is due to the filter size not being correlated to the resolution of the height data. The alternative is to increase the filter sizes, but was infeasible due to hardware constraints. The image was resized by a factor of 0.5, using anti-alias resizing, compressing some large areas of terrain which initially created a near intractable task for the network to learn. This value was chosen through a subjective initial run through of varying sized inputs to the network. Resizing by a half was a value such that the input features were still recognizable, but the relatively small filters of the network would still learn. The height map data was largely clean, though when initial runs were conducted there was a small amount of pixel noise within the downloaded height maps. To reduce the chance of this noise appearing in the outputs a small median blur of kernel size 3x3 was applied to remove the noise without affecting the structural integrity of the height map. An example of the final clean height map can be seen in Figure. 5.11 (a) alongside the real region (b) the elevation data was

extracted from (Taken from Google maps). Median blur was favorable here over a typical Gaussian due to the properties of the data. Since the resolution was 30m per pixel we noticed at a very low level grainy pixels that had large variation from the neighbouring pixels, creating some noise in the data.

All height data was normalized based on a min and max height variable that we defined at the start of data collection. Leaving all of out height maps in a range of 0-255, these were normalized to 0-1 upon training

The actual longitude-latitude values were generated entirely randomly, which created a large data set of randomly selected regions of the planet - this was to avoid a bias in the data selection process, since the SRTM mapping also includes oceanic regions, which were almost entirely flat height map regions. After inspecting an initial pass of data collection, the average height of oceanic height values was below an average threshold of 0m across all elevation points. Therefore if any region in the data set had a minimal range of height values (average height of the elevation under 0m Mean sea level (MSL)) they were discarded. However it was apparent that most of the regions showed little to no interesting features, and mainly produced relatively flat land. Therefore we manually select regions that we wanted to train, these can be seen in any of the height map figures in this section.

5.2.5 Texturing Height Data from Satellite Images



(a) Height map (b) Texture (c) Height and texture layered

Figure 5.11: A randomly sampled height map image (a) and it's aligned corresponding texture map (b) acquired using google maps - alongside the final training data texture (c) showing alignment of height values with the texture pixel values.

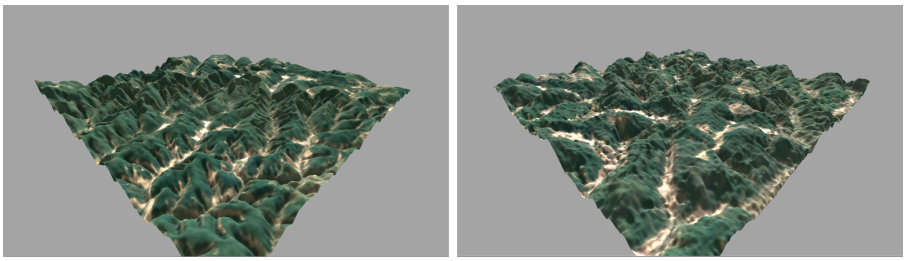
This section will briefly touch on extended work from the main focus of this chapter,

the work was published and boasts an improvement to the generation by using satellite data. We also show some crucial comparison of other methods used in the area.

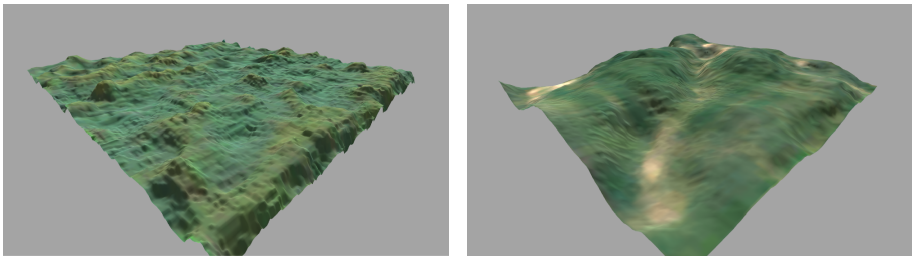
For the corresponding texture data, Google maps API was used for directly acquiring texture data. The API was used to create calls based on the longitudinal and latitudinal positions of the chosen rectangle area that would visually align with the gathered height map data. An example of the textured data can be seen in Fig. 5.11b. This was an additional step as part of our initial experimental design, the aim of this step was to add another layer of procedural generation, allowing the network to not only learn the height features, but also the colours with relation to what would appear in the real world.

Although the emphasis wasn't on the quality of the texture, as google maps is inherently blurry at pixel level, the colouring of the terrain could be used as a reference point for artists or users wanting to author terrain, using the pixel colour at each height as a reference to a separate texture map.

It was important to compare the current method against other commonly found generative networks. We decided to use the textured data samples for this comparison as it shows a more visual representation of the problem. The visual comparison results can be seen in Figure 5.12, where we compare a DCGAN and Gatys et al. method of texture synthesis against our method of generating height data.



(a) Baseline (Training region) (b) Proposed SGAN



(c) Gatys et al. (d) DCGAN

Figure 5.12: Visual comparison of 3 methods with the baseline training region rendered as a 3D terrain with generated texture.

Table 5.1: Structural similarity index (SSIM) and mean squared error (MSE) analysis of height and texture data used to render the comparisons in Fig. 5.12 against the base line region. With an average of both MSE and SSIM against SGAN shown.

Method Compared to SGAN	SSIM	MSE	Average Fidelity Loss
SGAN	0.587	5876	-
DCGAN	0.410	10515	43%
Gaty *et al.*	0.310	11948	70%

We observe that when adding in SSIM and MSE tests which are performed on a batch of generated samples, against the original training data, the SGAN performs slightly better on its structural similarity compared to DCGAN and far better than Gatys method. But when comparing MSE which is the overall pixel-to-pixel comparison SGAN outperforms massively the other two. This small quantitative analysis shows that Gatys method fails to capture much of the terrain in its generation, with both the colour and structure being far off the target, whereas the DCGAN produces slightly better results.

The visual analysis shows a similar story, with the SGAN producing some impressive results, with DCGAN producing a good structural copy, but lacking the scale and number of features. Gatys method falls through in our experiment, and fails to capture any structure, but some of the texture. In a future section we will investigate comparisons further, but will exclude Gatys' method, but keep the DCGAN based on this performance.

5.2.6 Experiments

Experimental Setup and Network Optimisation

In order to create a more optimal model to the parameters used as default, we used parameter exploration to determine which set of parameters would produce the most similar results compared to the initial inputs. These parameters were chosen from a range of predefined variables based on current standards in deep learning. We witnessed that the depth of the network was extremely important with regard to the interconnected regions and larger features of the input, however would directly increase the size of the GPU memory required.

The parameters we explored were as follows -

Learning rate [**0.0005**,0.0001,0.005,0.001]

Generator filters size [(3,3), (5,5), **(7,7)**]

Discriminator filters size [**(3,3)**, (5,5), (7,7)]

Batch Size [8, 16, **32**, 64]

Depth [5, **6**]

We ran this exploration a total of 40 times, each time picking a random selection of preset parameters which had not been picked before. The network was trained for 1 hour each time until it was stopped, the quality of the networks performance was determined using SSIM as discussed below. The best performing parameter combination is highlighted in bold.

We used SSIM (structural similarity index) as our baseline comparison algorithm, and a way of monitoring the parameters exploration. SSIM was chosen as it differs from non-spatial absolute error estimators like MSE or PSNR, where in this context the structural similarities are crucial to the generation of height maps from preexisting data. We also used MSE as a baseline measure, to ensure that over an amount of epochs the MSE value was declining. This has been shown to still be somewhat useful, even in non spatially similar comparisons. Both of these measures were calculated against the input data.

The SSIM index is calculated on various randomly selected windows from the image. The windows are measured between two ranges x and y.

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

An important part of deep learning is knowing when to stop training a model. Images are inherently difficult to statistically understand, unlike processing other data types. Early stopping is a way of determining when the model is trained to a point that any further training could only cause detrimental effects. The network training took into account the constant loss of the generator, where it was normal for the loss to steadily decrease as the discriminator improved, until a point at which loss plateaued. Therefore, the gradient of the loss was calculated constantly throughout training for a window of 25 previous epochs, when the gradient was 0 ± 0.01 for 10 consecutive epochs, the network

training was halted. These values were found to be accurate when analysing a network that was trained for an exaggerated amount of time - where once the network's generator loss started to plateau there were no visual or statistical improvements in the image quality - and in some cases, degradation occurred where filters would experience mode collapse causing features to be entirely lost.

Usually, the end of training values were signalled at epoch 600-700 of training. This can be confirmed with the use of statistical measures of similarity. For each output of the network, the statistical comparison values using structural similarity (SSIM) and mean squared error (MSE) were used. Their results averaged across 10 trained regions can be seen in Figure 5.13 (a), where a stagnation in change can be seen at around the 600 epoch mark. SSIM is a method of detecting similarity in structures between two images, this detects additional information that might not be attributed with MSE, where MSE attempts to perceive the actual pixel error. SSIM is an important measure with the data in this section, as the structural features of the data are not spatially dependent, meaning they can be located anywhere in an output image.

Lastly, a technique used to optimally train a generative network is the balancing of one over-performing network compared to the other. In our case, the loss of the discriminator was always far lower than that of the generator, indicating that the generator part of the network was struggling, or simply the discriminator was getting too good at predicting real against fake data. In this case, the solution was to train the generator twice for every update of the discriminator, this meant the discriminator loss reduced far less dramatically. This is a common technique in deep learning where otherwise one part of the network may outperform by a large enough margin to the other part, and cause instability during training.

After the testing discussed in the first part of this section we found that the parameters in Table 5.2 worked optimally on creating high quality comparative, yet variant data. These parameters were used for the resulting outputs that can be seen in the rest of this section.

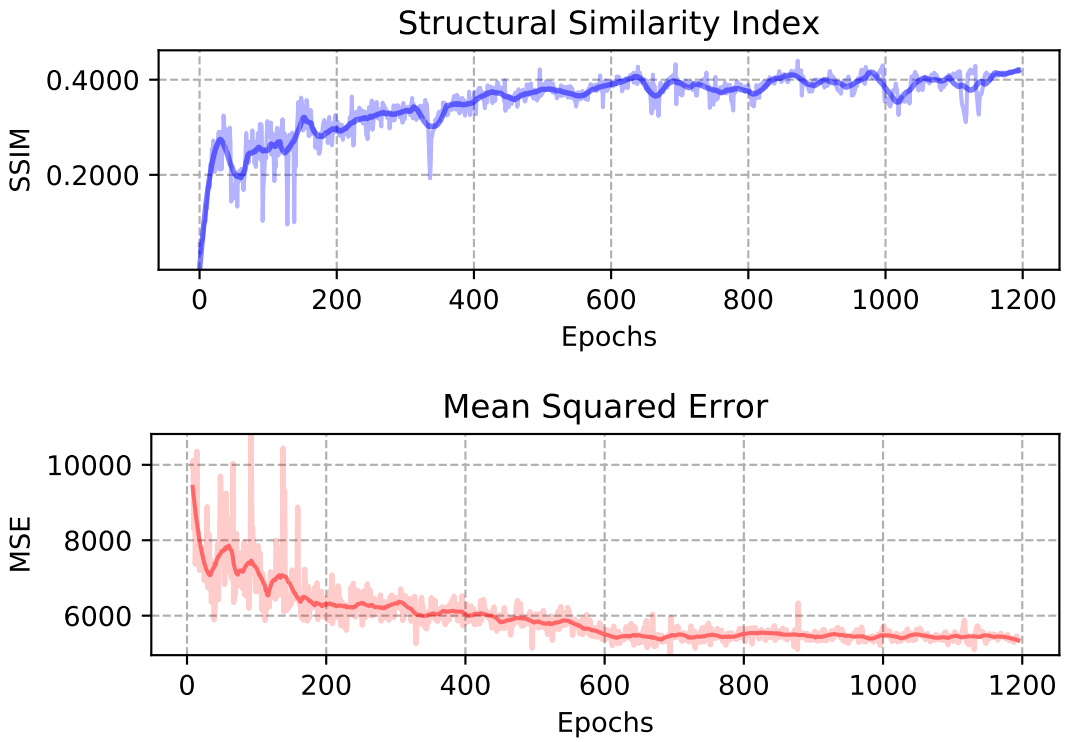


Figure 5.13: Analyse over epochs of the average structural and MSE similarity of 10 training regions. Determines a point at which results no longer visually increase in quality.

Network Training

The training models were conducted on a cluster of 8 GTX 1080 TI's using one card per region. Each input region could be fully trained within around 4 hours of training to a point of an output that resembled the original training point's features; while training to a point of completion took around 4 times that time (16 hours total), though the final 12 hours of training showed relatively little visual improvements and would only be required for extremely detailed results. Training was deemed complete when no noticeable visual changes were seen over several epochs using our SSIM measurement. This could also be quantified by exploring the loss graph, when the plot of the loss for the generator started to plateau with no more changes this signalled the end of training. The whole training process could easily be scaled back to one or more lower end card, though training times would take significantly longer.

Following the final training of a model, the network could then be reloaded and produce replica samples of the original region almost instantaneously, with the output size of the image only limited by GPU memory. Figure 5.14 shows a grid comparison of one of

Table 5.2: List of the main SGAN hyperparameters for the results shown in this section. N^i represents the current layer.

Parameter	Value
Optimizer	ADAM
Learning rate	0.0005
Momentum	0.5
L2 regularization	1e-5
Batch size	32
Depth (N)	5 & 6
Discriminator Filter's	(3,3) * $2^{(N^i + 6)}$
Generator Filter's	(7,7) * $2^{(N^{max} - N^i + 6)}$

the generated samples, each generated region taking roughly 50ms to generate on a GTX 1080Ti. These generations were output at a resolution of 2.4k x 2.4k, roughly 2x the size of the input texture which is a similar resolution prior to downscaling the input.

Network Results

Several regions of the planet were chosen based on their ability to convey the different features and structures that the method is capable of understanding, though the selection of coordinates to acquire regions was relatively random. Figs. 5.14, 5.15, 5.16 and 5.17 show four chosen trained regions alongside their general location and more specifically their longitude-latitude position. These figures show the direct output of a fully trained network based on the corresponding input, and for comparison an example of how they would appear in a very basic 3D render, mimicking how a generated sample might be visualised within a specific use case scenario, minus the texturing and entities.

When analysing the figure results in individual detail it is apparent that there are some limitations with feature connections. In Figs. 5.14 and 5.15 the input contains a large number of valleys and ridges, the generations fail to completely connect these features together. Whereas in Figs. 5.16 and 5.17 that have less connected regions in the input, the generations appear to have less issues with the transfer of features.

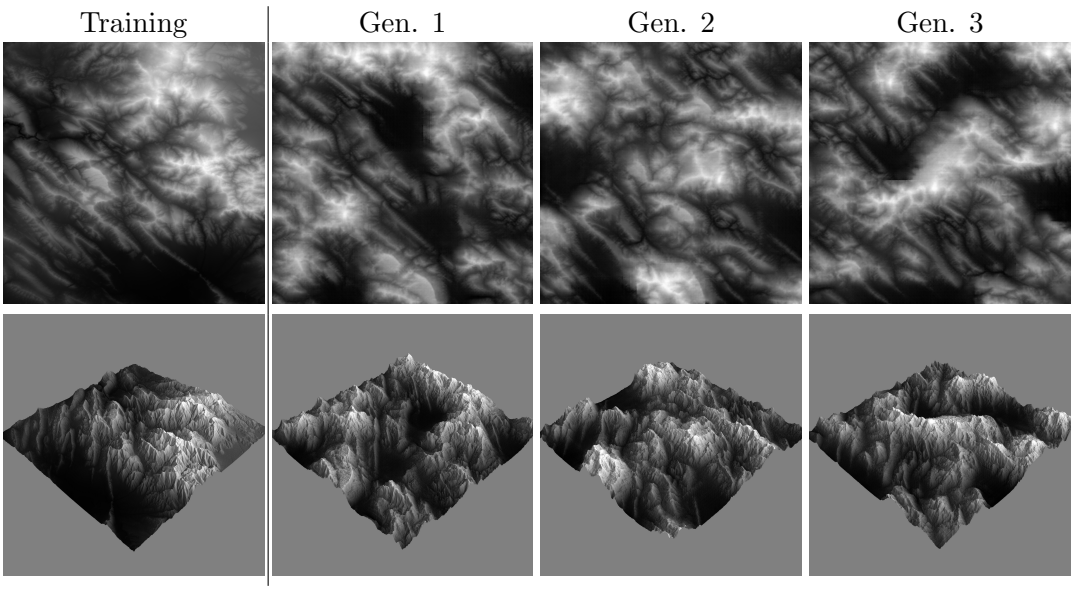


Figure 5.14: Height map (top) and 3D render (bottom) representation for training and 3 generations regions for (44.3,36.1,45.1,36.9)

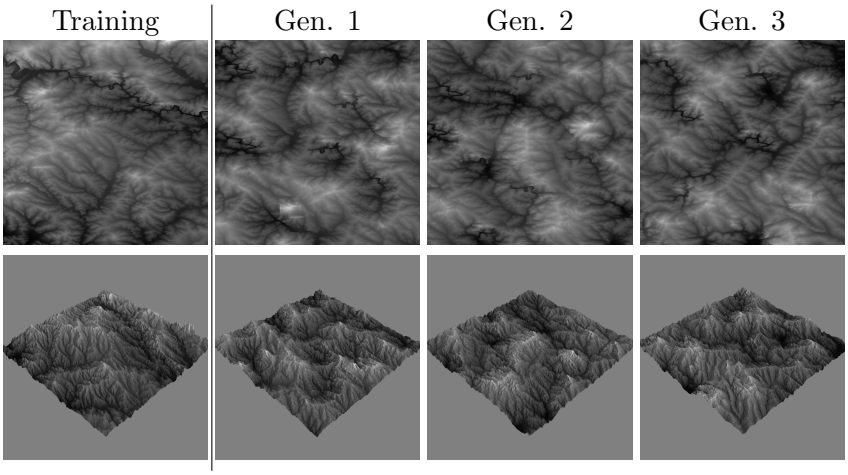


Figure 5.15: Height map (top) and 3D render (bottom) representation for training and 3 generations regions for (125.4,53.1,126.2,53.9)

5.2.7 Comparison of Different GANs

In this section we discuss, visualise and compare the results of different techniques for texture synthesis with regards to height map data. These approaches are not designed specifically for the type of data in this Chapter, but are used to justify the results from the previous sections and their strengths and weaknesses. Importantly, methods and comparisons discussed here are reference and used later in Chapter 6.

We chose to use three different approaches for comparisons, one cutting edge GAN (BigGAN), one base line GAN (DCGAN) and a non deep-learning technique (Image Quilt-

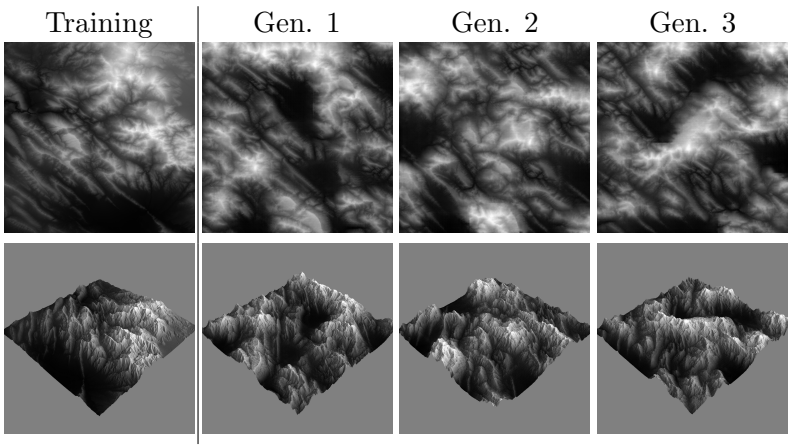


Figure 5.16: Height map (top) and 3D render (bottom) representation for training and 3 generations regions for (44.3,36.1,45.1,36.9)

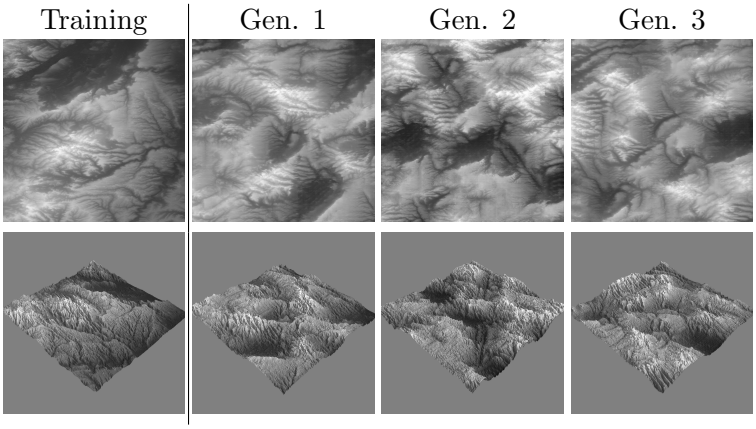


Figure 5.17: Height map (top) and 3D render (bottom) representation for training and 3 generations regions for (-1.1,52.3,-0.3,53.1)

ing). These three methods were also used alongside the SGAN data and real data as part of a participant perception test in Section 6.1.1. These methods were chosen as we felt it would provide a wide variety of performance, whilst incorporating the more commonly used methods in general computer vision.

BigGAN learns spatial dependencies which by default produces unfavorable results with the terrain data as an input. However if we pass in random crops of the height data we can augment a sizeable data set of very similar feature space. This leads to the model producing plausible height data outputs. This GAN takes advantage of attention maps, allowing the generator and discriminator to focus on different parts of the image. The attention maps allow the GAN to update areas of an image that are performing poorly against the discriminator. The BigGAN also combines a large amount of empirical improvement that appear in GAN research over the years. The draw back to using this

method is the sheer memory requirements for training even a medium sized output, our limit on an Nvidia Titan was 256x256 pixels. This meant using resizing to bring the height data inline with other methods.

The DCGAN has been discussed in many sections of this thesis, but appears in 2015 [27] and is the first paper using convolutions alongside the GAN idea. The first iteration of DCGAN produced very small outputs due to GPU memory limitations at the time, however during our experiments we simply increased the number of layers inline with the amount of GPU memory we had access to. The original DCGAN produced outputs of size 64x64, our DCGAN could output much large resolutions at size 256x256.

Image quilting is a non deep learning approach which provides a novel method of synthesising textures by stitching together small patches from existing images. This method boasts promising results on a range of textures, at competitive speeds. The quilting used in this section for the height maps is a direct out of the box application of the method shown in their paper [2].

5.2.8 Evaluating Training & Generation Times

A metric we have yet to discuss is time, a quality that is not as important for the majority of applications of height data. Although in the instance of real-time (online) application it can be a substantial bottleneck. In this section, we evaluate the time of a fully trained generative network, against some commonly used noise-based approaches. We tried to maintain consistency with variables of each algorithm, where possible bringing the adjustable quality of noise and search-based algorithms to that of the output of the generative network.

Although Perlin noise can be generated on a GPU using GLSL through a shader program, we chose not to include these values in our comparison, as there is no practical way of pulling the height data off the GPU to create a heightmap for additional performance needs like ground collision. But for interest a GTX 1080, using 35 octaves to achieve similar quality to that of our generated samples, at a resolution of 2000x2000 would have taken 0.06 seconds or 60 milliseconds.

5.2.9 Comparing to Other Methods

We wanted to provide a comparison to methods which could be used for similar content generation. These techniques would be used to show comparison, both visually and statistically, and potentially support the approach that we took in this chapter.

Image quilting was chosen as non-deep learning comparison, alongside this we used a basic DCGAN as a baseline and a more modern variation called BigGAN, although the DCGAN and BigGAN were designed more for non spatially varied data. This was done by taking random crop samples from a larger height map sample of an area. We then applied an SSIM calculation between all cropped samples, the SSIM would find structurally similar data which contained similar features to one another. The matrix of SSIM comparisons can be seen in Figure 5.20. The general idea here was to cluster the cropped samples in to groups which were highly comparable to each other in terms of feature space, this produced the "best" set of cropped images to be used in the training process. The alternative is to use all crop data, however from an early test, and based on how the DCGAN and BigGAN operate, we noticed results from this training were poor and feature space was not being learned. An example of a BigGAN poor performing training output can be seen in Figure 5.18.

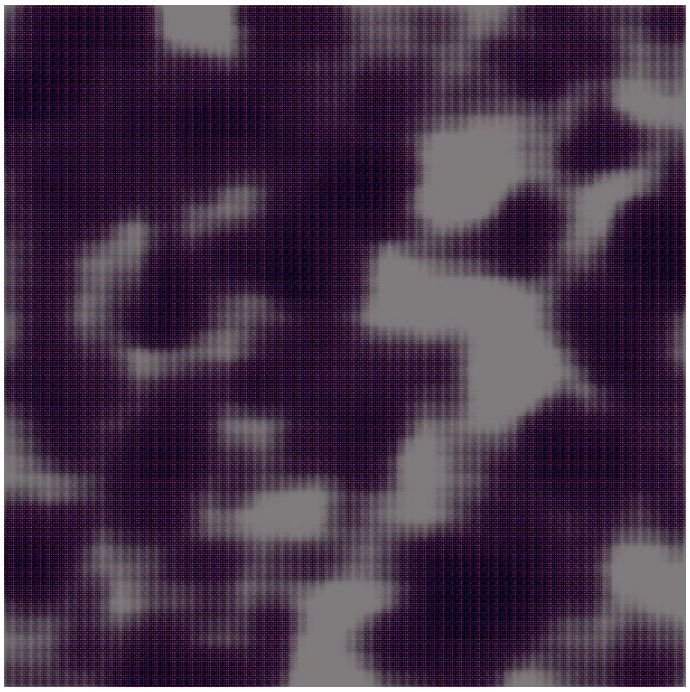


Figure 5.18: Result of BigGAN trained on completely random cropped data from a target height region, resulting in a poor understanding of feature space.

Both the non-spatial networks (DC and Big) were both trained with the completely random crop data, and data which was picked using the SSIM matrix shown in Figure 5.20.

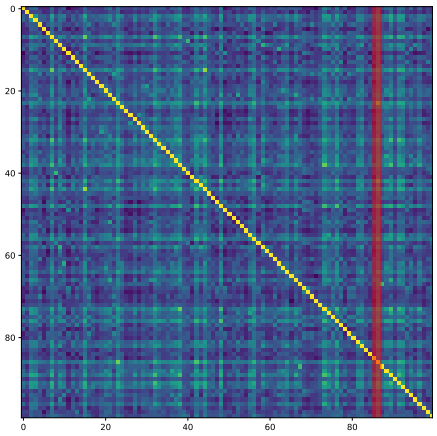


Figure 5.19: 100x100 comparison matrix of random cropped samples of a height map. Red overlay shows the highest cumulative SSIM value at index 86. The samples with highest similarity from this column were used as training data for the improved DCGAN approach.

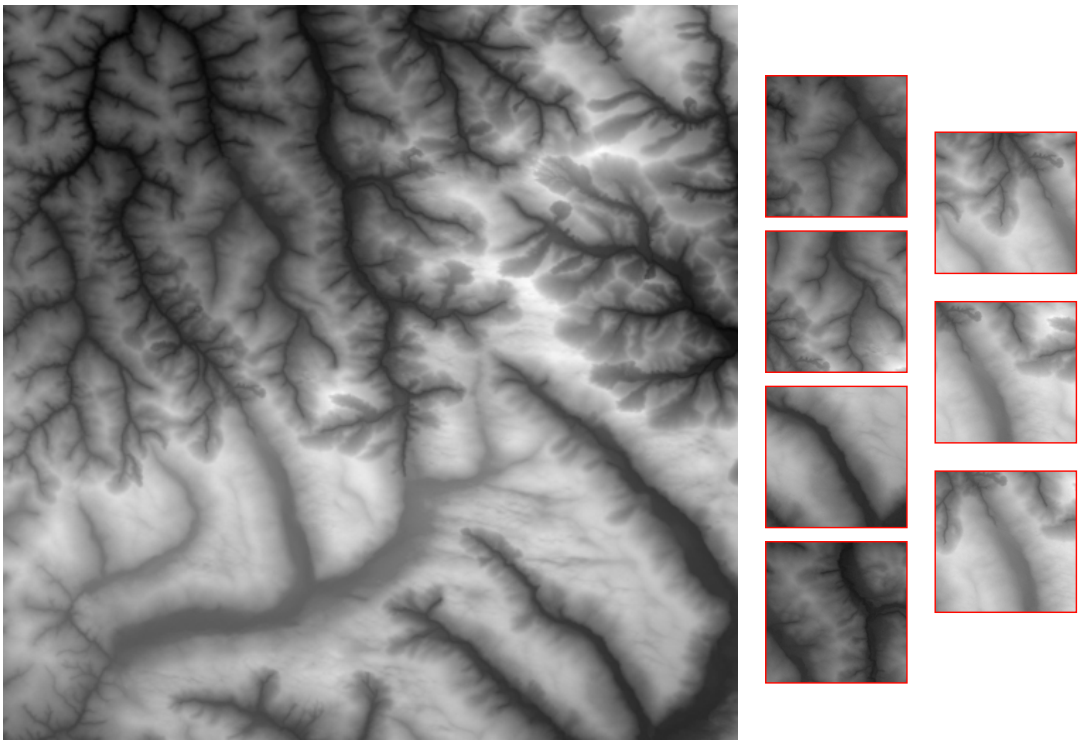


Figure 5.20: Several crops (right) of the larger input image (left) using the SSIM matrix shown in Fig 5.20. All cropped images have similar feature space to one another.

5.3 Results, Evaluation and Comparisons

Many research papers use only quantitative methods to describe their network's performance and results, a lot of the time using extremely specific algorithms that show high performance, but only in certain context.

From my experience, there is a distinct lack of qualitative methodology in an area where the end user will more often than not be an average human. Quantitative algorithms prove not to be as functional as are commonly used in telling the story of an end user's experience. I believe qualitative studies in the rapidly moving field of ML are simply infeasible to achieve, and in most cases requiring months of work post results from experiments. And in a field where publications may already be 6 months behind where the current state of the art is, simply leaves little time for this type of slow moving work.

This section is intended as a deep dive in to the rich data that can be uncovered with surveys and perhaps some surprises with comparison to algorithmic performance of networks.

There are obvious draw backs to this type of approach, and time and resources are the main two. I was lucky enough to have time to conduct this survey whilst writing my thesis, but for the fast paced field of deep learning it is apparent that research needs to be published as soon as possible. Though the study itself is directed at human participants, the resulting analysis was quantitative. The study paved the way for a much larger survey later in this thesis, which aimed at far more qualitative methods of analysis.

5.3.1 Initial Exploratory Survey - Height Comparison

Creating an argument for visual results is extremely subjective as to what appears to look more visually appealing/similar, excluding algorithmic comparison techniques such as Keypoint Matching [77] or inspecting distribution of pixels, which appear biased for the data produced in this section, as the output images contain inherently similar distributions to their training regions. Therefore to cement the justification of useability within virtual environments, an experiment was devised to discover participants' ability to distinguish between the multiple fake regions that were generated. The results of the

study show a strong correlation of the generated samples with the original training points with 90.15% classification accuracy when asking participants to classify a sample against multiple regions, one of which the sample was derived from.

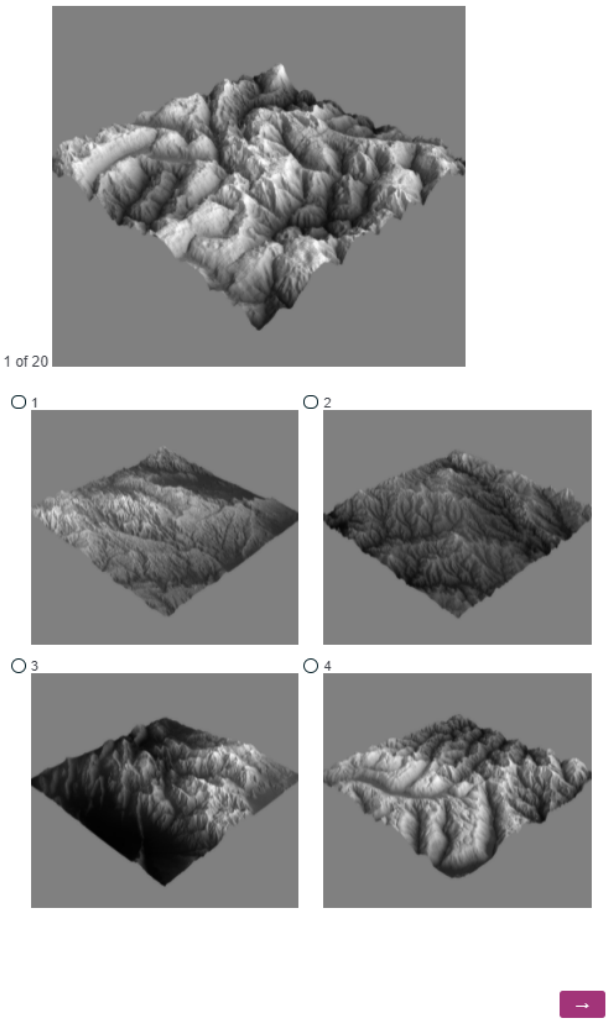


Figure 5.21: Panel from first participant survey on perception of *similarity* between procedurally generated rendered terrain and their derived training data. Users were asked to choose their perceived most similar region. Each question chose a random baseline rendered region, and 4 regions from a pool of 100 pre rendered terrains for each.

262 individuals were tested through various online forums, notably Reddit forum r/proceduralgeneration. From the 4 trained regions shown in Fig. 5.14, 5.15, 5.16 and 5.17 the model generated 20 random samples for each region, leaving a pool of 80 samples. During the survey the participant would be shown a total of 20 samples, with each sample individually shown alongside the 4 training regions, these were drawn from a uniform distribution of the 80 generated samples. The survey explicitly asked the participants to record their answer based on which of the 4 original training regions they thought more

Table 5.3: Table showing the participant classification for the 4 regions used in this survey, numbered 1 - 4 referencing the layout in Figure 5.21. Showing region 1 was the easiest to classify, whilst 2 and 4 were slightly harder. Although all classification percentages were extremely high, and greater than expected.

Region	**1**	**2**	**3**	**4**
Correct	210	205	195	217
Incorrect	10	14	16	17
Total	220	219	211	234
Percentage	95.45%	93.61%	92.42%	92.74%

closely resembled the shown sample, since each sample was derived from one of the training regions the accuracy could be recorded for each choice. The survey took on average just over 2 minutes (standard deviation(SD) of 90 seconds) for each participant to complete, giving around 6 seconds (SD 4.5 seconds) per answer.

A non-parametric Wilcoxon test revealed a significance of $\mathbf{p < 0.001}$; showing a highly significant result for users managing to classify generated height maps with the regions they were derived from.

This was the first iteration of survey design for the height map data, it showed a real interest with the amount of responses gathered, with a very quick turn around given the ease of access to respondents, and especially those of relevant respondents. We hypothesised that the generated samples would be strongly matched to their derived and trained regions. The results shown proved that the generated samples were extremely similar to that of their training regions, this satisfied our requirement at the beginning of the section to design a network that could produce variants using the same terrain features as that of a target region.

It was however not a perfect study, and drew no comparisons to other methods which could be used to produce similar results. We also collected next to no meta information, excluding time taken to complete. Important information such as age, sex, experience in the area etc. were all key points that were not considered.

Discussion

The network captures most of the local structures, but fails to connect these small local structures into the larger and more impactful areas: large valleys, long mountain ridges

etc. This can be seen within [75] where large connected structures of satellite images fail to be learned. The structure of terrain can be more accurately connected by increasing the depth of the generator and discriminator though doing so requires a larger amount of memory and training time. Furthermore, the method struggled with regions that didn't exhibit visually repeating features, as the filters would be constantly updated in a different direction to the general feature trend. This can be observed through the results of the section always containing a general feature trend across the training images.

Something touched on but not explored, is the use of conditions and semantic maps to add a user control to the design process. Something that I have had direct feedback on from people using this method of height generation, was the quality was fantastic and the ability to generate from very specific terrain types was great. But there was the need for some finer detail control, such as where these features would be mapped on a generated sample. The idea, which we discuss in much more detail in Section 8.2.1, is to use some mapping of different features in a terrain map before training and to train the network using both the texture and the feature map. Then once trained the generation would be produced alongside a user-generated feature map, controlling the layout of the features in the heightmap.

5.4 Summary

This chapter explored the use of a wide variety of generative networks for textures and height data in the 2D space. We have shown novel applications of GANs for high quality and region-specific heightmap generation, texturing of height map regions using satellite data alongside elevation data, and in-depth, comparisons using quantitative methods. Various other cutting edge and historical GANs were also trained in order to visually show how our GAN approach performs, alongside the advantages of each method.

We explore how textures of very specific designs can be used in generative methods to create a pipeline that could be used within a real-life game development company. These textures have been produced using artist assistance.

Chapter 6

Height Map Survey

6.1 Further Surveys and Perceptual Evaluation

6.1.1 Background

For this study we wanted to further understand peoples perception on different generative techniques, and how the results of these methods compare to each other. It was extremely important to build on the previous study shown in the section above. We knew that respondents were abundant, and it was easy to find people willing to offer their time for such a study. The main focus of the survey was to obtain as much relevant information possible first, then proceeding the main survey section we would ask for voluntary additional information which could be used to explore the types of people that answered the way they did.

Typically when metrics are used to compare GAN results, perceptual test are rarely used. This leaves a large amount of deviation for the choice of metric used, and in some cases can be cherry picked to produce more favorable results. Calculations of these specific numerical values are used to summarize the quality of generated images.

Advantages and disadvantages of GAN evaluation measures [4] is a paper that investigates 24 different quantitative metrics, and can be seen in Figure 6.1, that are currently used to evaluate GANs. The insight they show is that a lack of a defining metric seems to only hinder the progression of GANs and leads to bias results and a saturation of unnec-

essary GAN related research. Furthermore an extensive study in 2017 [78] showed that there is no empirical improvements from the original GAN architecture to many of the self defined "improved" models in today's work.

It is obvious that quantitative measures are far easier to run and compare on large scale GAN data sets, compared to running perceptual studies. However the main issues seem to be the arbitrary picking of comparison metrics in order to satisfy a narrative given by the authors.

	Measure	Description
Quantitative	1. Average Log-likelihood [18, 22]	• Log likelihood of explaining realworld held out/test data using a density estimated from the generated data (*e.g.* using KDE or Parzen window estimation). $L = \frac{1}{N}\sum_i \log P_{model}(\mathbf{x}_i)$
	2. Coverage Metric [33]	• The probability mass of the true data "covered" by the model distribution $C := P_{data}(dP_{model} > t)$ with t such that $P_{model}(dP_{model} > t) = 0.95$
	3. Inception Score (IS) [3]	• KLD between conditional and marginal label distributions over generated data. $\exp(\mathbb{E}_{\mathbf{x}}[\mathbb{KL}(p(y \mid \mathbf{x}) \parallel p(y))])$
	4. Modified Inception Score (m-IS) [34]	• Encourages diversity within images sampled from a particular category. $\exp(\mathbb{E}_{\mathbf{x}_i}[\mathbb{E}_{\mathbf{x}_j}[\mathbb{KL}(P(y\|\mathbf{x}_i)\|P(y\|\mathbf{x}_j))]])$
	5. Mode Score (MS) [35]	• Similar to IS but also takes into account the prior distribution of the labels over real data. $\exp\left(\mathbb{E}_{\mathbf{x}}\left[\mathbb{KL}\left(p(y \mid \mathbf{x}) \parallel p\left(y^{train}\right)\right)\right] - \mathbb{KL}\left(p(y) \parallel p\left(y^{train}\right)\right)\right)$
	6. AM Score [36]	• Takes into account the KLD between distributions of training labels vs. predicted labels, as well as the entropy of predictions. $\mathbb{KL}(p(y^{train}) \parallel p(y)) + \mathbb{E}_{\mathbf{x}}[H(y\|\mathbf{x})]$
	7. Fréchet Inception Distance (FID) [37]	• Wasserstein-2 distance between multi-variate Gaussians fitted to data embedded into a feature space $FID(r,g) = \|\mu_r - \mu_g\|_2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}})$
	8. Maximum Mean Discrepancy (MMD) [38]	• Measures the dissimilarity between two probability distributions P_r and P_g using samples drawn independently from each distribution. $M_k(P_r, P_g) = \mathbb{E}_{\mathbf{x},\mathbf{x}'\sim P_r}[k(\mathbf{x},\mathbf{x}')] - 2\mathbb{E}_{\mathbf{x}\sim P_r,\mathbf{y}\sim P_g}[k(\mathbf{x},\mathbf{y})] + \mathbb{E}_{\mathbf{y},\mathbf{y}'\sim P_g}[k(\mathbf{y},\mathbf{y}')]$
	9. The Wasserstein Critic [39]	• The critic (*e.g.* an NN) is trained to produce high values at real samples and low values at generated samples $\hat{W}(\mathbf{x}_{test}, \mathbf{x}_g) = \frac{1}{N}\sum_{i=1}^{N} \hat{f}(\mathbf{x}_{test}[i]) - \frac{1}{N}\sum_{i=1}^{N} \hat{f}(\mathbf{x}_g[i])$
	10. Birthday Paradox Test [27]	• Measures the support size of a discrete (continuous) distribution by counting the duplicates (near duplicates)
	11. Classifier Two Sample Test (C2ST) [40]	• Answers whether two samples are drawn from the same distribution (*e.g.* by training a binary classifier)
	12. Classification Performance [1, 15]	• An indirect technique for evaluating the quality of unsupervised representations (*e.g.* feature extraction; FCN score). See also the GAN Quality Index (GQI) [41].
	13. Boundary Distortion [42]	• Measures diversity of generated samples and covariate shift using classification methods.
	14. Number of Statistically-Different Bins (NDB) [43]	• Given two sets of samples from the same distribution, the number of samples that fall into a given bin should be the same up to sampling noise
	15. Image Retrieval Performance [44]	• Measures the distributions of distances to the nearest neighbors of some query images (*i.e.* diversity)
	16. Generative Adversarial Metric (GAM) [31]	• Compares two GANs by having them engaged in a battle against each other by swapping discriminators or generators. $p(\mathbf{x}\|y=1;M_1')/p(\mathbf{x}\|y=1;M_2') = (p(y=1\|\mathbf{x};D_1)p(\mathbf{x};G_2))/(p(y=1\|\mathbf{x};D_2)p(\mathbf{x};G_1))$
	17. Tournament Win Rate and Skill Rating [45]	• Implements a tournament in which a player is either a discriminator that attempts to distinguish between real and fake data or a generator that attempts to fool the discriminators into accepting fake data as real.
	18. Normalized Relative Discriminative Score (NRDS) [32]	• Compares n GANs based on the idea that if the generated samples are closer to real ones, more epochs would be needed to distinguish them from real samples.
	19. Adversarial Accuracy and Divergence [46]	• Adversarial Accuracy. Computes the classification accuracies achieved by the two classifiers, one trained on real data and another on generated data, on a labeled validation set to approximate $P_g(y\|\mathbf{x})$ and $P_r(y\|\mathbf{x})$. Adversarial Divergence: Computes $\mathbb{KL}(P_g(y\|\mathbf{x}), P_r(y\|\mathbf{x}))$
	20. Geometry Score [47]	• Compares geometrical properties of the underlying data manifold between real and generated data.
	21. Reconstruction Error [48]	• Measures the reconstruction error (*e.g.* L_2 norm) between a test image and its closest generated image by optimizing for z (*i.e.* $min_{\mathbf{z}}\|\|G(\mathbf{z}) - \mathbf{x}^{(test)}\|\|^2$)
	22. Image Quality Measures [49, 50, 51]	• Evaluates the quality of generated images using measures such as SSIM, PSNR, and sharpness difference
	23. Low-level Image Statistics [52, 53]	• Evaluates how similar low-level statistics of generated images are to those of natural scenes in terms of mean power spectrum, distribution of random filter responses, contrast distribution, etc.
	24. Precision, Recall and F_1 score [23]	• These measures are used to quantify the degree of overfitting in GANs, often over toy datasets.

Figure 6.1: An exert from*Pros and cons of GAN evaluation measures* [4] shows the sheer quantity of evaluation metrics currently used across many popular authors and researchers within the field of deep learning. A lack of universal metric is only hindering advancement in GAN research.

6.1.2 Introducing the Methods, Survey Methodology

Given what we have discussed we decided to run a survey to determine what actual end users of these types of algorithms think. With this in mind we set out to design a fair experiment that was easy enough to attain a large number of results online, but contained every avenue to target specific terrain generation methods. To this end we chose to use the following algorithms, with the real data acting as a baseline, in which all of the terrains were trained.

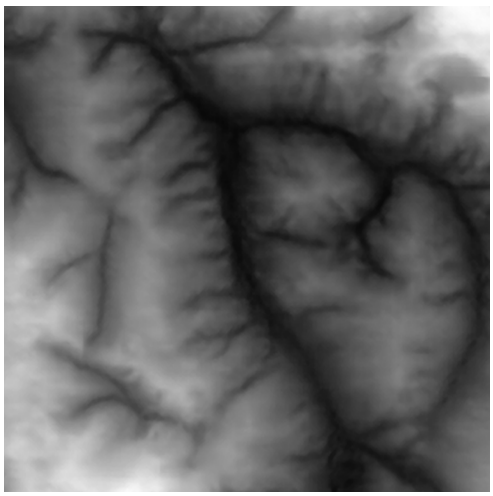
These methods have been discussed in Chapter 5, but are the comparison methods for generation of height data that were used in this survey: SGAN, BigGAN, Image Quilting, DCGAN and the real data.

Each generated height map was of varying sizes due to the nature and parameters

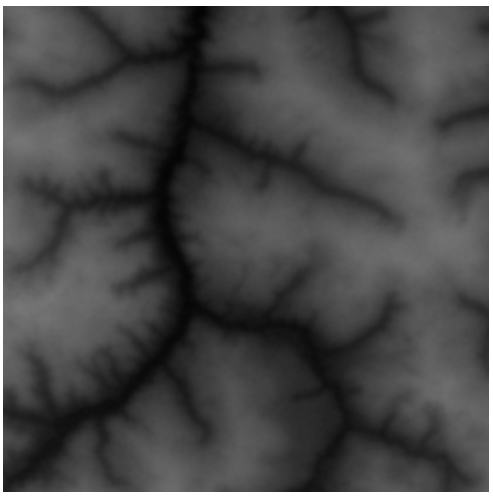
of the given algorithm. Therefore we opted to upsampling each height map to the largest output size. For this we chose to use Bicubic resizing, which for height map data ensured a smooth average between set height points on the image. These resized height maps were passed directly into the rendering pipeline. The rendering of each image was performed using mayavi. After rendering a screen shot was captured in the mayavi render window and were captured from the same camera position for each render. Mayavi was used as it allowed for quick rendering of the huge amount of height data we had, importantly we had to ensure any emphasis on colouring and lighting was either removed, or normalised across all terrains and this was possible with mayavi.

We ensured that the data added to the survey was evenly distributed, with 150 samples from each of the method's data. 150 samples per method led to a selection of 750 data points to use per question across techniques, although repeating data could not be seen over multiple questions, meaning each respondent would see 50 unique data points or roughly 6.7% of the data for a completed survey. We were confident that we would gain enough respondents to ensure that each data point would appear in multiple surveys.

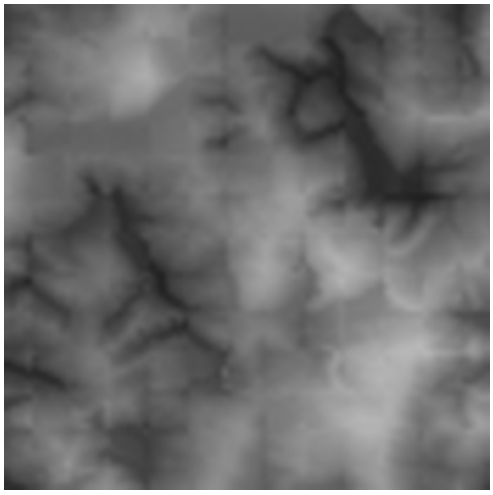
An example of the data prior to rendering can be seen in Figure 6.2. The full renders were resized to 400x350 using Bicubic resampling, maintaining the resolution of the originals but reducing size, this was simply due to how large the images were, hindering load speeds of the survey data. This was found out in early runs of the survey with a few participants, and for a normal internet speed loading 5 high resolution images was extremely slow and created a poor experience for the user. This created a trade off between the high quality terrains we were aiming to investigate against how many respondents we could gather. The size to which we resized the images still showed the general feature space we were interested in gauging against.



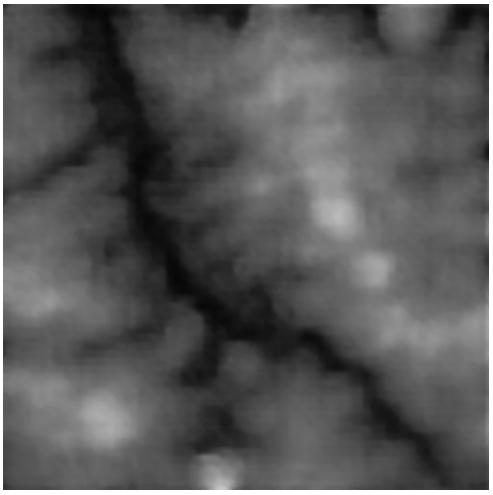
(a) Improved SGAN



(b) Real Data



(c) Image Quilting



(d) BigGAN

(e) DCGAN

Figure 6.2: The 4 types of height map generation shown in the survey, alongside the real world data from which all of the generations were derived.

An example panel from the survey can be seen in Figure 6.3. In this image users were asked to drag with their mouse, or finger on smart phone, each rendered image of

a height map into an order that they believe has the highest to lowest perceived quality. The user would then be shown up to 10 sets of terrains to rank, before moving on to some question investigating how and why they ranked the data they way they did. But also asking questions on their age, experience, and their favourite game that has a terrain with explanation of why they enjoy it.

We hypothesised that the survey would produce a ranking where the results produced by the SGAN and the real data source would be ranked the highest. We expected BigGAN and quilting results to be slightly behind these, with DCGAN in a clear last place. These assumptions were taken from the pre-rendered visual quality and resemblance to the real data as seen in Figure 6.2 which visualises the height data produced by each method. Importantly, we were looking at the overall quality of feature space and how connected each region of features was, as would be expected in the real data to have long corridors of features resembling mountains and valleys for example.

6.1.3 Data Cleaning and Timings

Of the respondents almost all would have been from reddit, with the exception of shared links between friends or colleagues, the exact numbers for survey traffic are not known. The survey was posted on 5 subreddits (proceduralgeneration, SampleSize, gamedev, MachineLearning and gaming) within the reddit community, very similar forums were used to the survey in the previous section, as this is where we found the most readily available and sensible responses. These were also chosen due to their assumed understanding of the terrain data.

The survey was taken 516 times. Of those responses, 462 responses were ethically sound, in which the participant was over 18 and was happy to consent to their data being used.

To begin with any progress made of 0% was immediately removed, these were responses where the user had passed the initial ethical checks, but had made 0 progress towards answering any questions on the survey. 80 responses were removed during this validation, leaving a total of 382 responses where questions had been attempted at least once.

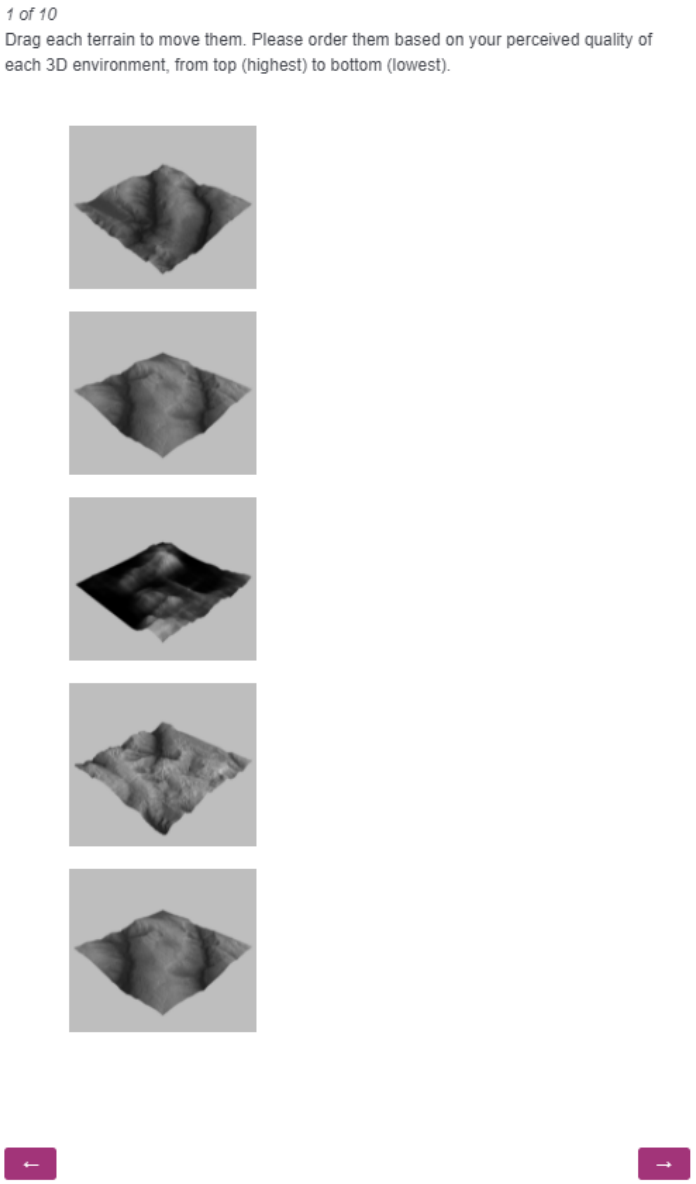


Figure 6.3: Layout of a randomly generated survey panel, users were asked to drag each of the rendered height maps in to an order that they perceive from highest quality to lowest. Each rendered height map was randomly sampled from a pool of prior rendered data as follows - BigGAN, DCGAN, Image Quilt, SGAN or real data.

The next important decision was whether to remove incomplete surveys from the results or not. Since our questions were entirely random and drawn from a normal distribution of samples per choice, we opted to keep only those results where at least 2 more more questions were completed. This was purely people who had started the questionnaire but not ranked any of the data and exited out, the data would still be recorded as 1 question being complete, but storing their answer as the random order it was shown in. This was largely permitted due to the independence between questions. Interestingly over 60% of respondents did not finish the survey and so an added factor to the decision was the

massive amount of lost data. Finishing the survey was defined as people who had reached the end of the main ranking questions, and had either answered or passed on the optional textual question responses. Therefore those that had ranked all of the questions but had closed the survey at the optional mark would be marked as unfinished, even though they had contributed to the survey. This would have inflated the number of unfinished surveys, as over half of the unfinished surveys had completed all of the ranking questions.

Overall after full cleaning 194 responses were left, and from these responses 1075 questions were answered of a maximum of 1200 for 6 question per participant.

Time is an important part of visual comparison studies, it is always hard to tell how much time people spent actually completing each question when conducting questionnaires over the internet. But what we do know is time completion for the entire survey, therefore we aimed to create a reasonable range of values which would remove any data which may not have been answered seriously.

The time data was not normally distributed, and had a median completion time of 119 seconds, with a mean of 6765 seconds and standard deviation of 3596.

Although theoretically people could have answered questions over the course of several hours or days, some responses simply took too long, and was probably down to people leaving the survey open. We used SD as an upper bound measure, removing 12 responses from the data set, or roughly 2.5% of the data set. Figure 6.4 shows a comparison of the removal of these upper bound data points, bringing the range of completion times in to a much more sensible range.

Based on prior timing experiments with both myself and several colleagues, serious completion of the survey took at least 30 seconds to complete. This value lined up well with the lower quartile range of the actual responses we gathered at 26.5 seconds. It is very obvious based on some timings that participants must have skipped through and not completed the survey as intended, many people with completion times of 2 seconds.

It gets slightly trickier as we are including data which could have come from incomplete survey responses; something that we did was normalise the times complete against how many questions they had answered. This allowed us to thoroughly see who had skipped through the questionnaire. 80 responses were removed based on their normalized

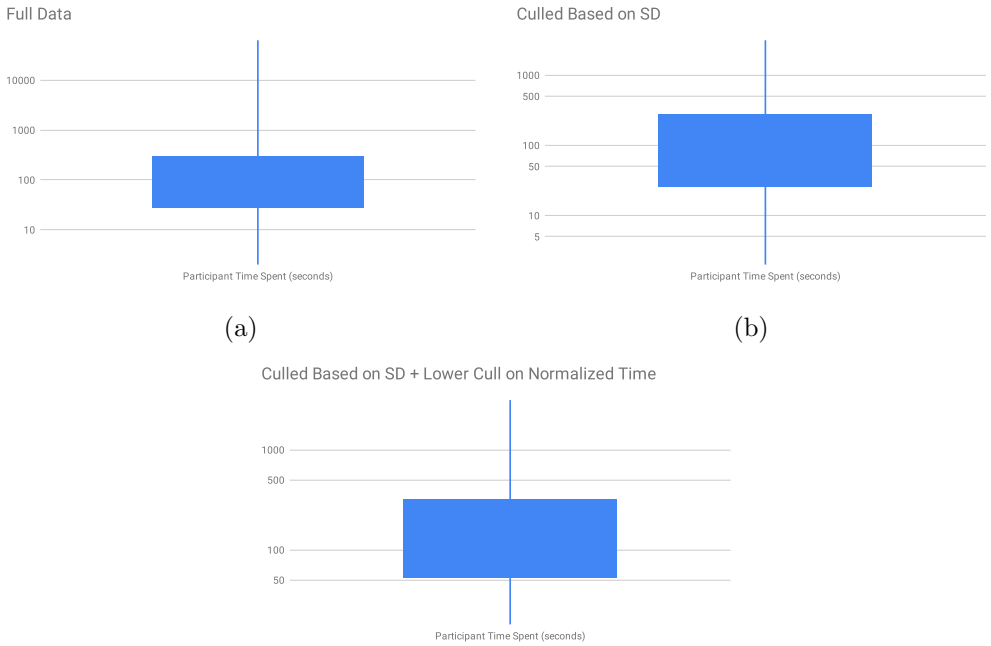


Figure 6.4: Time spent doing survey as a box and whisker plot, on a logarithmic scale.a) is the raw data without any refining. Graph b) shows data after removing results that were over 1 standard deviation of the entire data. c) shows the final timings after removing anomalous data, with the final removal completed using normalized time completion against questions answered.

Table 6.1: Table of percentage distributions for results in the rendered height map perception survey. Shown is the ranking of each method from 1 to 5 and the ranking frequency

	1	**2**	**3**	**4**	**5**
DCGAN	2.03	3.50	7.19	29.68	57.60
BigGAN	8.18	16.75	29.44	25.87	19.76
Quilt	25.14	27.89	23.62	16.51	6.83
Real	26.13	26.13	21.32	15.70	10.71
SGAN	**40.15**	26.42	18.11	10.92	4.39

question response time, where anyone under 5 seconds per question was removed. We chose 5 seconds as completion of 6 questions at 5 seconds each would reach our minimum reasonable completion time of around 30 seconds. Any time spent under this would be deemed to have been rushed with little thought to the ranking.

6.1.4 Survey Response Evaluation

The distribution of results per method can be can be seen as absolute value in Table 6.2 and as a normalized per method percentage representation in Table 6.1.

Table 6.2: Table of the number of occurrences for the results in the rendered height map perception survey. Shown is the absolute values for each of the rankings, and the summed occurrences for each method.

	1	**2**	**3**	**4**	**5**	**Occurrence**
DCGAN	22	38	78	322	625	1085
BigGAN	87	178	313	275	210	1063
Quilt	265	294	249	174	72	1054
Real	288	288	235	173	118	1102
SGAN	430	283	194	117	47	1071

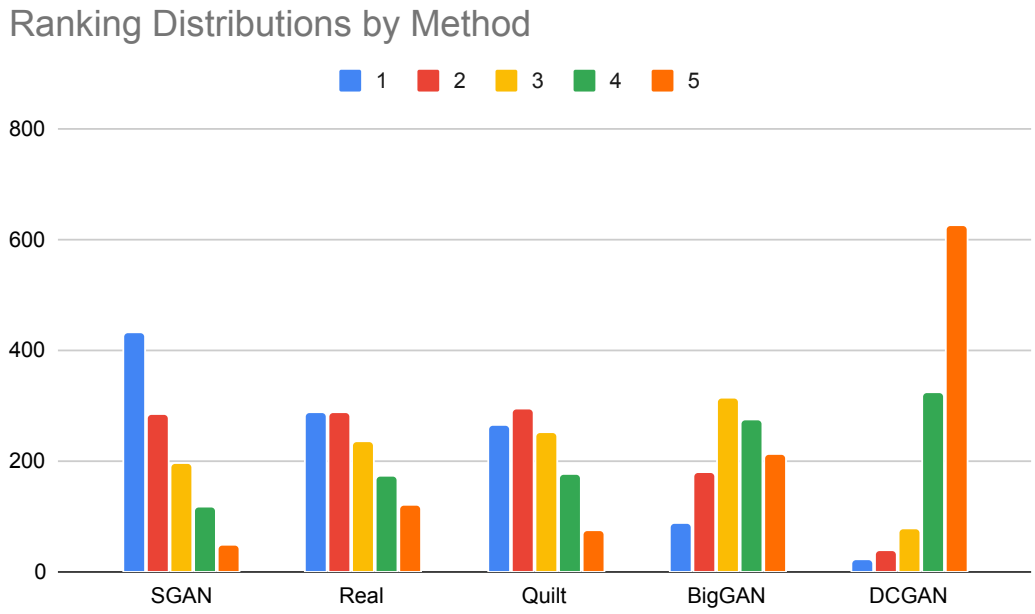


Figure 6.5: Overview of results in a tabulated form. Showing the distributions from Table 6.2

As part of additional collected information, each respondent could optionally answer any of 5 additional questions. These questions were posed in a way that could help understand why certain methods would be ranked the way they were, and also to understand what people generally perceive when looking at game terrains.

These questions asked were as follows -

1. "What is your gender?"
2. "When judging the images, what were you focusing on?"
3. "How long have you been playing video games?"
4. "Name a game that has a terrain/environment that you enjoyed recently"

5. "Describe a feature of the terrain/environment that you enjoyed"

All were open response questions, except the question on how long had you played video games, which was 5 preset options of "Under 1 year", "1-5 Years", "5-10 Years", "15-20 Years", "20+ Years".

This section will explore results from each of the questions individually; we will then discuss and explore the results with respect to the main ranking in order to further understand some of the complexities behind the results. Each heading will refer to the list of questions outlined above in the exact order shown.

6.1.5 Question 1 - Gender

Since we conducted the survey almost entirely on reddit, although the link could have been shared with colleagues/friends through word of mouth, we expected to see a predominately male response, as this is not only the demographic of reddit but also the subreddits that we used [79].

From the 192 responses, 151 left their gender, with 90% identifying as male, 4.6% female, 2.6% agender and 2.6% leaving a response that was considered an insincere response.

This however was far more than the original demographic data that Reddit puts forward, where it shows 69% of users are male [79]. This is more than likely an artifact of the communities that the survey was posted in, we targeted mainly game development sub-Reddits, where it could be assumed there are high numbers of male users.

Gender certainly provides some differences in visual perception tests, however most studies show only a slight difference in the speed of which men pick up visual perception than women, even more so as the age of the participant increases [80]. Since there was, within reason, no maximum time limit for participants this can essentially be ignored.

6.1.6 Question 2 - What participants focused on

Question 2 was probably the most important question in gaining insight into why people had ordered the questions the way they did.

In this section there are 2 figures, showing the frequency of words within comments. Figure 6.6 shows a simple keyword grab, where each comment has its unique words counted across all of the comments. The importance of this figure is how people were judging the terrains that were shown to them and highlights some significant differences between participants. Although this was an automated process, we have added in an additional column which contains the sum of the term real, realism, and realistic, which all have the same connotation, this column is in light red.

It is apparent that two key words stand out beyond any of the others, being "detail" and "realism". Interestingly they both have completely separate meanings in the context of ranking terrains. Realism implies the terrain mimics features of the real world. Whereas detail may pertain to the method of rendering, and how detailed each of the 3D renders were, rather than the features of the height maps themselves.

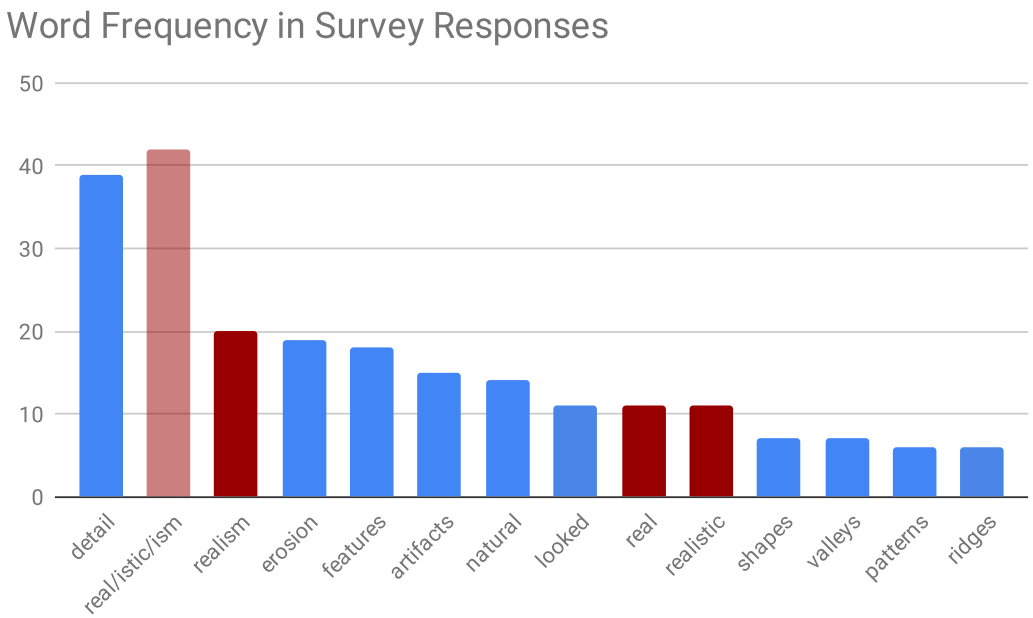


Figure 6.6: Figure showing the extracted key words for question 2. Removing any punctuation and stop words from the list. Any key word with over 3 occurrences were then plotted to form this Figure. In red are very similar key words with slightly different phrasing, with the opaque column showing the combined values of this "real" keyword.

Importantly, although Figure 6.6 shows the number of keywords without a labelling bias, many of the comments varied in length, and some contained multiple keywords and even multiples of the same keywords. For example the comment *"Amount of detail, apparent repetition and grid artifacts. I might have overvalued detail and contrast, as flat terrains are common in the real world.."* which contains detail twice, but also the keyword real. Whereas a smaller comment such as "Detail" contains just as much insight into how they ranked their questions, but with 3x less occurrence of keywords.

Because of this inconsistency, we added tags and categories to each of the comments manually, of course adding some subjection to the process, but also reducing the comments down to a single tag which should more accurately reflect a comment, regardless of length. For these, we preset 6 tags, realistic, detail, features, erosion, texture and variation.

After tagging each comment 42% were ranked based on realism, 24% on details and 20% on features with the rest of the tags below 10%. These can also be seen plotted in Figure 6.7.

This proved to be extremely insightful into the reasoning behind the performance of each method, we see a large focus on realism, detail and features. This is also more prominent in the first method we used to tag, where detail and realism dominated the keyword plot.

The first question to answer is why SGAN performed so well in relation to these comments, and also why real data was not top, if realism is so important.

To answer the first question we must look deeper into the method of SGAN, and the added functionality we added for specific height map data. SGAN provides a way of generating new data from a preexisting texture, but unlike most learning methods, subsamples the input into smaller features and learns from these. Therefore it can be asserted that SGAN learns the "interesting" feature space within an input, and in our case the interesting parts could be valleys, rivers and the detailed ravines. This then coincides with the second and third reason people ranked the way they did, being the detail of the render, and the features of the render. This is something which the SGAN did very well on mimicking, to the point where people could always tell which input a random generation was trained from, as seen in the first study in Section 5.3.

The next question to answer is more complicated, and if realism was so important of a reason behind ranking, why was the real data not top, and only slightly better than image quilting.

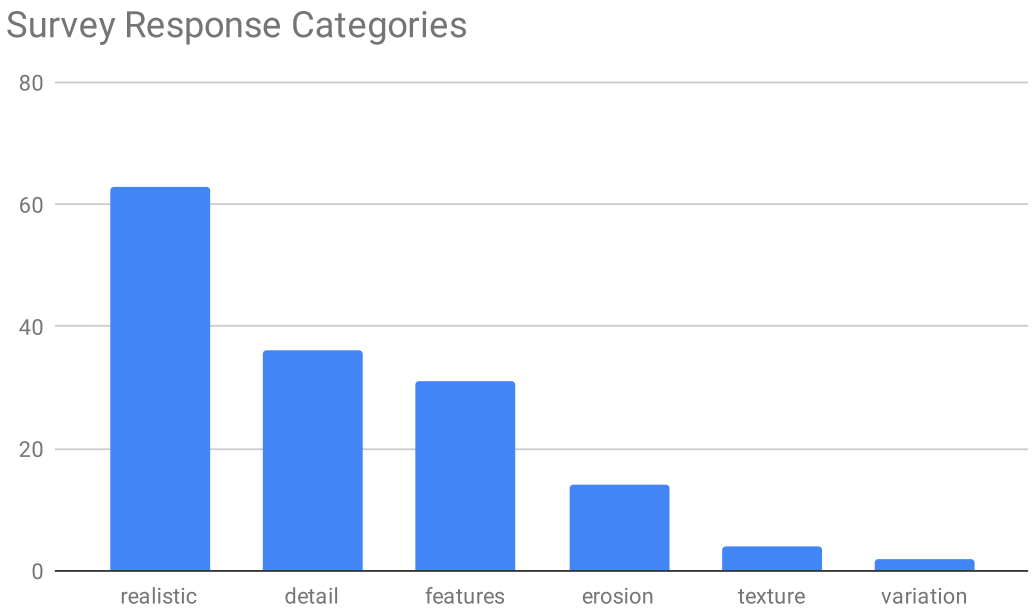


Figure 6.7: Figure showing subjective categories of key words for question 2. Each key word is formed from an entire comment from each respondent and takes in to account how they framed their response.

Observing how image quilting produces height data shown in Figure 6.2, we can see that the features in the quilting technique replicate that of real-world data but on a much lower resolution. Whereas if we compare it to the real data directly, the quality is definitely less, but going back to how people ranked based on features, the vast spread of features on the quilting could be the reason why it performed nearly as well with much less quality.

It became quickly evident that although people thought they were judging based on the realism of the terrain, when plotting the data of those respondents who were tagged with "Realistic" performed similarly to those that were tagged with "detail". This further proves why SGAN performed as well as it did, with the feature of the SGAN being more pronounced, and being near identical to that of the real terrain, but containing far more of these perceived "detailed" regions.

6.1.7 Question 3 - How long have you played video games?

This question was intended to gauge the experience that the users have had with video games, a fairly important component of the survey, and something that would make a big difference in the way people rank the terrains if they were relatively new to video games as a whole.

This question is deliberately broad but leads into a much more specific question in the next section, the combination of these two questions makes for a powerful meta understanding of the types of people who participated.

The questions were from a fixed response, from under 1 year, 1-5, 5-10, 10-20 and 20+ years.

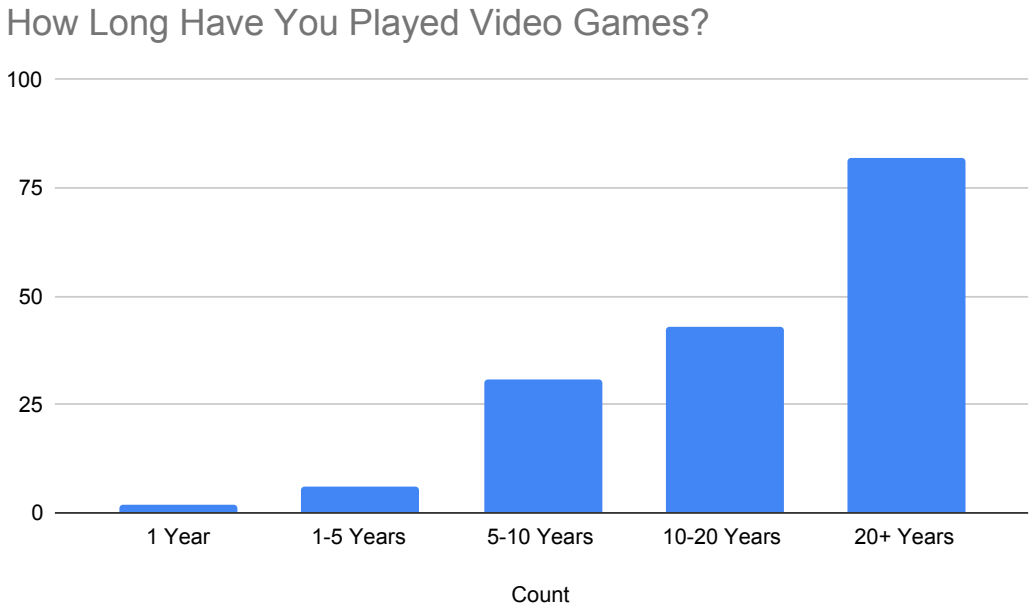


Figure 6.8: Figure with data from survey on how long participants have played video games for with a fixed response number of years

The results can be seen in Figure 6.8, with 50% of the participants having over 20 years of experience playing video games. Only around 5% of the participants had played video games for *less than* 5 years, making the overall respondents very experienced with video games. This is important, as we initially were expecting a higher level of performance in the way individuals could discern the terrain differences with the more experience they had previously had, regardless of which games, but had been exposed to games for a long

period of time.

6.1.8 Question 4 and 5 - Recent Game Played & Enjoyed Features

Question 4 - Name a game that has a terrain/environment that you enjoyed recently
Question 5 - Describe a feature of the terrain/environment that you enjoyed

Responses for question 5 were tied to the answering of question, therefore we will discuss both of these together in this section. These two questions asked firstly, "Name a game that has a terrain/environment that you enjoyed recently" and then "Describe a feature of the terrain/environment that you enjoyed".

At this point in the survey the data was already very clean, however, as with all open box responses, there was some noise within this section; of the 138 people who chose to leave a response here we removed 3 answers that were either empty or contained filler characters. We also slightly altered some responses, where the participant had missspelt a game name. Lastly, any response with acronyms for games, i.e. BOTW meaning the game Breath of the Wild, was changed to match the full name. 2 responses from the data did not leave a game name, but responded with the fact that they could not think of a game they had enjoyed "I have not played a game with a terrain/environment that I enjoy" and "generated land almost always sucks compared to hand-crafted features, other than it's infinite"

Out of the 133 responses that were left, there were 71 unique game title. Although most game titles were only mentioned once, there were 8 games mentioned 4 or more times. The list below shows the game and the number of occurrences.

1. Minecraft - 19
2. Breath of the Wild - 6
3. Red Dead Redemption 2 - 5
4. The Witcher 3 - 5
5. Valheim - 5

6. Far Cry 5 - 4

7. No Man's Sky - 4

8. Skyrim - 4

Interestingly most of these games exhibit non-procedural terrain, where only Minecraft, Velheim and No man's Sky have terrain that is procedurally generated. The type of game that people were thinking of whilst ranking the terrains in the survey would have had an impact on the results.

Furthermore whilst taking the ranking of those who were using procedural games such as Minecraft as a reference point, to more curated terrain-based games, there was no noticeable change in both the overall score and the ranking within the technique. This shows that the type of terrain in the game makes little difference to the ability to detect real-world data from that of generated terrains using cutting edge techniques.

Further Key Word Analysis on Question 5

The next stage is analysing the comments left regarding the type of game that people referred to in their previous comments. For this analysis, we initially did a quick keyword ranking to determine what people were generally thinking about with their favourite games terrain.

This can be seen in Figure 6.9. From this plot, we can see a large difference in the words that people used to describe how they ranked the terrains in question 3 and Figure 6.6. In fact, the highest occurring word "detail" has entirely disappeared from the word list. Whereas some words such as features and realism were still important descriptors.

From this, we can summarise that participants whilst ranking and comparing terrains held detail as a very important factor, second to realism. However, when asked to name a game terrain they enjoyed without comparing it to anything, detail suddenly became a non-factor, along with realism to some degree being much lower down the list.

This is a very interesting find, and it may isolate that people are looking for immersion over detail regarding game environments. The words described all follow a similar

pattern of specific features within the terrain. This is the opposite to what we assumed coming in to this study, thinking that detail and overall features would be of high importance.

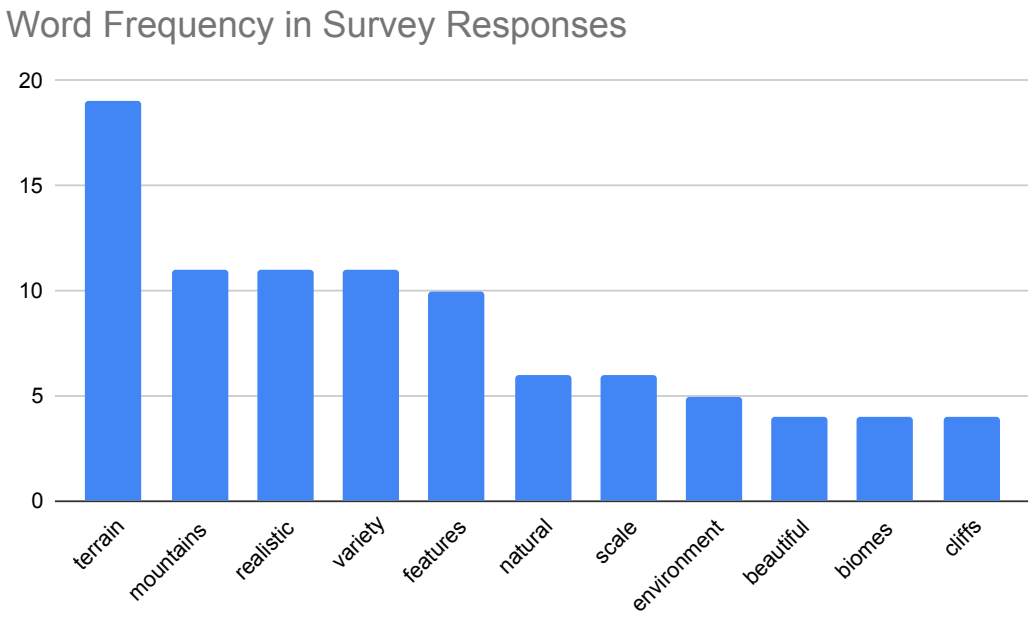


Figure 6.9: Figure showing extracted keywords for question 5, removing any punctuation and stop words from the list. Any key word with over 3 occurrences was then plotted to form this figure.

PCG vs non-PCG Comments

Next, we wanted to further analyse the difference in procedural content against real-world data, as this would open up some more answers on why the rankings fell the way they did. For this, we split each game into a category of whether it uses PCG in any way to generate its terrain and games that use handcrafted terrain. For example, Skyrim would be categorised as non-PCG, whereas no man's sky would be categorised as PCG.

We chose to only label those games that occurred 3 or more times in the data set, of those which were left, there were 28 games that used PCG and 24 that did not for their terrain. We then explored the use of keywords in the responses for each set of games. Compared to our previous analysis, the reduction in the number of games that were analysed led to a far smaller textual data set.

What we found was a very clear difference between the two categories; the people

who referenced non-PCG terrain-based games were very much focused on the specific detail within the environment, things that would be hand-tailored by game developers. To contrast that, we found those commenting pertaining to PCG terrain were far more interested in the deviations and biomes associated with those games.

When exploring individual data on a case by case basis to explore the above finding, we found that the types and specific details for each type of player was extremely varied, whereas PCG players very much had key words associated with vastness and endlessness, the non-PCG players were more focused on small details, with some keywords such as detail, exploration and confined. We believe the two distinct categories of players form a completely different and non-overlapping focus.

6.2 Summary

Overall the survey provided an entirely surprising insight, into both the amount of collected data and the quality of the responses. The previous section has set to lay out all of the data we acquired in a meaningful way that we can discuss in more detail in this section. This section will aim to bridge the understanding of why the SGAN outperformed everything, along with investigating why the BigGAN, which appeared to have high-quality outputs, performed poorly.

To begin with setting out this experiment we controlled every part of the terrain generation process between methods, using the same real-world heightmap input for all methods. This real-world heightmap was also used entirely for the real-world examples in the survey. Cropping was an important part, and a process that would ensure the resolution of each method's generations would appear the same. For this, we tried to match the features of each method to the same scale, whilst keeping the same size of the heightmap. Any resizing was performed using nearest neighbour resizing. Figure 6.2 visualises the conformity between the height data, ensuring all features were of the same scale. The only anomaly is the generations from the DCGAN, which were of such low-quality that individual features could not be matched with the other generations.

We analysed many different combinations of the data to determine what our par-

ticipants were focusing on with ranking the terrains, but also a plethora of metadata surrounding their interests in video games and specific terrain-based environments.

From our findings, we believe that the SGAN performed so well because of the inherent replication of features, but not only how it replicated real-world features, it also enhanced the number of features and detail of features. When we compare the example in Figure 6.2 we can see that although both real data and SGAN are similar, the SGAN produces far more permutations from the branched valleys and ridges.

Furthermore we believe that the real data performed similarly to the image quilt purely down to the fact that there are similar quantities of detailed features. Contrasting to the worse performing BigGAN whose features are extremely sparse in detail and lack the fractal variations that the SGAN and real-world data offer.

This summary is particularly useful for the future of texture and terrain generation algorithms, and something that should be investigated further, with perhaps the use of added user input on top of computer-generated imagery to enhance the detail even further. This survey has also proven that image quantitative measures are not the only metric that should be used in these studies, something that plenty of other research in this area forgoes.

Chapter 7

Generations in 3D Space

This Chapter focuses on work done during an industry placement with Sony, and the work contained here was with guidance from them following quite a challenging and specific research topic they had on the use and generation of point cloud data for animation and pose of human models

Generating 3D assets is a much more complex problem than in 2D space. Apart from the extra dimension, we also see a shift from typical representations in image format (a grid of pixels) to a large variety of data formats. For example, Voxel, Point Cloud, Mesh etc. each of these formats offers a slightly different problem, but also the ability for more robust generation approaches.

Due to these reasons, generative methods will need to be refined on a per data format basis, and many pre-existing GANs focused on 2D images will not work or will be needed to be adapted to access an additional dimension. 3D procedural content generation could lead to some significant improvements to the way we generate content in real time. Early work in this chapter proves there is a solid base line for generation quality, and with more research would allow games to directly incorporate deep learning models in to their generation pipeline.

In this chapter, we explore GANs and algorithmic approaches for the three data types, point cloud, voxel and mesh and how deep learning can be used to create visually and statistically probable generated samples. The chapter is structured as follows, Section 7.1 will discuss the data formats and where deep learning can fit into this narrative. We

then look at a series of experiments in Sections 7.2, 7.3 and 7.4 using Deep Generative approaches. Limitations and drawbacks of using deep learning are then discussed in Section 7.4.5. Finally, we will summarise the Chapter in Section 7.5.

7.1 Data Representations

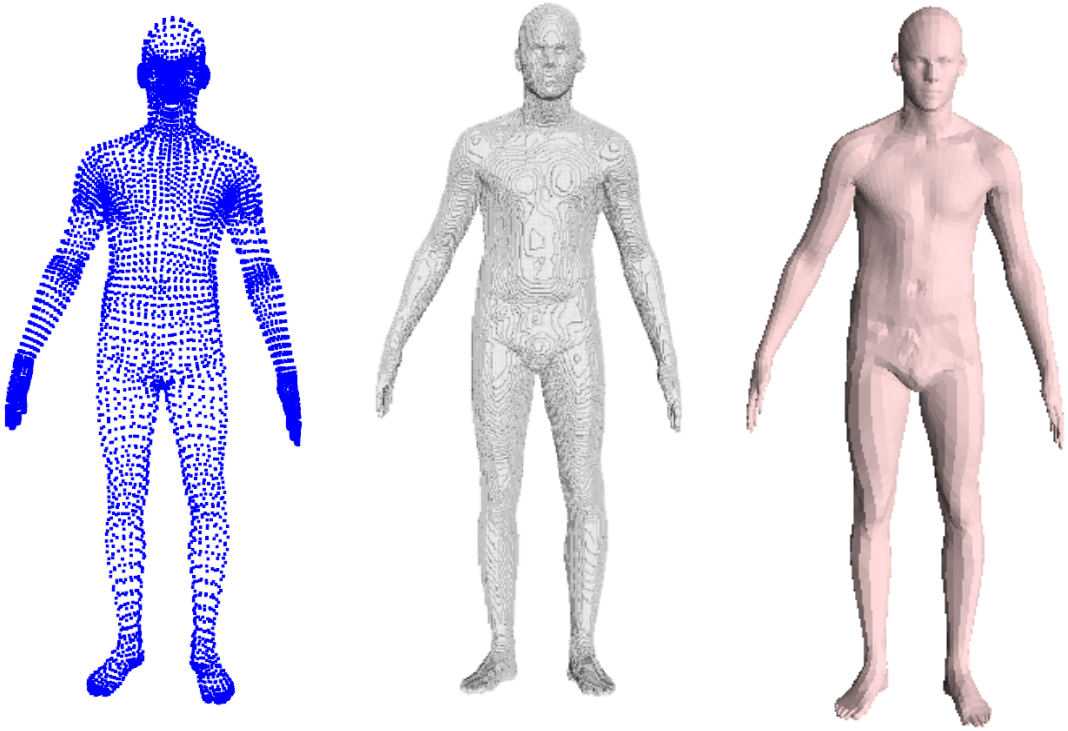


Figure 7.1: Example of three common data formats for representing 3D objects. In this case human pose models. (LR) Point cloud, voxel and mesh representations.

3D data presents a difficult challenge when compared to many data representations in computer science in general, even more so when approaching deep learning. Continuous space data is not easy to convert to some discretised format and in many cases contains other metadata in some entirely different format, such as 2D texture maps or animation scripts. There are ways of converting mesh data to and from different useable formats, however more often than not quality is lost in the process. For example, converting from mesh to voxel, and then back to mesh again using some marching cubes algorithm would see a drastic degeneration in the quality of the mesh.

This Chapter aims to investigate common representations of mesh data, and other variations, and how deep learning can be used to effectively understand and generate new

Figure 7.2: A 1.0 scale point cloud from human mesh, represented in a connected non directional graph of closest points N where N <0.1. The vague outline of a non euclidean human form can be made out within the graph.

procedural data.

It is important to understand the limitations of each representation with respect to deep learning. Some representations offer much less freedom as a data input for specific network architectures.

Figure 7.1 shows three common representations, point cloud, voxel and mesh, with the first two being commonly used as data inputs for deep learning. In this Chapter we will discuss how we used both point cloud and voxel data to contribute to the field of deep learning in computer vision.

There are a few main benefits of each data representation; Voxel data being per-

haps the most robust, it is entirely uniform (similar to an image but in 3D space) and data must exist in each grid space. This creates a near-identical data format to image processing with the addition of another dimension. However, can be extremely expensive in terms of GPU memory with the size scaling N^3. This data is very easy to use with traditional GANs, dense networks and convolutions layers, both have been shown to produce successful generation, classification and segmentation (cite).

Voxel data is typically much lower quality than that of its mesh derivative, and although there are methods of converting back to a mesh such as Marching cubes [81], a method of constructing new polygons between the discrete cubes of the Voxel, the conversion process tends to a much lower fidelity mesh result than perhaps was used to create the voxel.

Point cloud data is a set of points that are representational of a 3D coordinate system, where each point lies somewhere in this space. This type of data is usually captured from some real-world scan such as LiDAR, though it can also be produced through the downsampling of mesh data. Point clouds are a very straightforward way of containing a uniform data structure, which when combined with many points can form a complex view the surface of an object or scene.

Although memory can be an issue with point-based data, this can be alleviated with the dynamic ability to apply point decimation to randomly decimate points in the point cloud, this can help lower the points to a more reasonable amount while keeping the overall distribution of data. There are many ways of decimating point clouds, either by clustering, distance or simply randomly. This technique can also be used to sub-sample multiple samples from one point cloud, by taking a random smaller set of points from the larger point cloud multiple times.

Point cloud data can be converted back to a mesh using surface reconstruction algorithms, such as Poisson surface reconstruction [82].

Lastly, we touch on mesh data itself; a mesh file can be represented in hundreds of different formats, but the most common being STL or OBJ, STL standing for standard triangle language. Perhaps the most straightforward though is OBJ, which contains a list of vertices proceeded by a list of faces, alongside a whole host of other potential information

regarding colour, lighting etc. Vertices simply point to triangle positions in local space, and the triangles themselves are then constructed using a pointer to each of these vert positions.

There are tremendous difficulties with attempting to create some sort of meaningful connection between vertices and faces in these types of files. Not only do the vertices and faces vary between models, which for the majority of deep learning methods would fall over at that stage.

The use of essentially two information types within an obj file is where the problems really lie and would require some sort of two-part neural network, one for learning of the vertices and then a separate one for the connection of these faces. All in all, if it was plausible to train this type of data accurately the applications would be endless, however as of now the limitations are too great.

7.2 Voxel Generation of Coloured Objects

7.2.1 Introduction

Voxel data provides a straightforward method of interacting with standard GAN architectures in a generalised volumetric manner. Voxel data is inherently straightforward to train and represent due to its static nature of a fixed-sized 3D array of points, each of these points can contain as much additional data as possible. At its simplest form a voxel data representation may only contain a boolean variable of whether or not data exists in that data field, this is primarily used in voxel-based learning, however, data such as face normals, colour and even opacity can be included.

There are 2 main advantages of using adversarial training to generate 3D voxels from pre-existing methods, firstly 3D objects can be sampled from a prior probabilistic latent variable, creating unseen and novel objects from a data prior. Secondly, the latent space can be easily interpolated in order to traverse the data manifolds, creating smooth animations and control of outputs by controlling the input latent variable.

3D model creation forms a large part of the development process in 3D graphical

environments such as games or simulations. If an unsupervised approach can be used to generate high-quality textured models the turnaround in these areas could be greatly improved. Advances in generative deep learning have been shown to understand even complex 3D structures, allowing neural networks to output generations learned from abundant model data. In this section, we investigate the advancement of voxel-based deep learning and show how our method works to improve on previous state of the art methods.

7.2.2 Previous Voxel Work

Voxel-based data structures allow for a far simpler augmentation with current generative networks, where there is a uniform data structure, whereas mesh data files usually contain complex arrangements of data, which don't inherently work with more general network inputs.

The 3D field has seen an explosive development of deep learning applied tasks. Using multiple sections of pre-existing models to reconstruct variant samples of the original data points [63], these appear to be variant but it's easy to notice over time if exact sections of the selected models are used frequently.

Another interesting approach is the translation from the commonly used mesh data into a format that is much simpler to represent, mesh files have a complex list of faces, vertices, textures and sometimes normals, into a 3D array of values known as a voxel [83]. A voxel is a data structure that usually contains a binary value for each position in a 3D grid. With such a simple structure complex shapes can be rendered with no prior knowledge to object file data structures, however, depending on the complexity of the grid used to represent the mesh, the converted voxel quality could be reduced. An example of a voxel data structure can be seen in Figure 7.1.

With this representation technique more modern-day approaches have been applied to voxel data, with generative adversarial networks being shown to accurately map and reconstruct input voxel data. This can be seen in [83] and [84] where a 3D variant of a Deep convolutional GAN can learn the structure of the voxels, which are inherently simple for the network to understand with only one boolean variable per position in a uniform grid.

A technique of avoiding the voxel conversion type learning has been shown in [85], where an end-to-end style learning approach is applied, using two differing frequency generators and averaging the outputs results in a smooth surface mesh.

Consequently, none of these methods produces output results with colours learned within the network structure, some show post-processed texturing but these require manual or selected processing. The closest work related to this in the game-industry use is the use of the WaveFunctionCollapse algorithm [86], which is often applied to 3D models that were intentionally authored at the level of a voxel.

7.2.3 3D Convolutional Networks

In this section, we present a method of extending 3D Deep Convolutional Generative Adversarial Networks (DCGAN) by adapting the pre-processed data stage to a 4 channel voxel-based design. The data input allows the 3-dimensional DCGAN to learn both the original voxel data format and an extended colour representation for each voxel block. This also goes to prove the addition of many other elements specific to mesh data could be converted and added to this voxel representation.

There is also an exploration of the use of sparse data sets, and how generative networks perform in creating new varied samples when only exposed to a relatively small number of inputs. The number of models used in this section is 24 high-quality models from various artists. The process shows how the variant art styles between artists could be concatenated when inputting to a generative network, while still creating high-quality procedural outputs. The relatively low number of models explores the overfitting and vast artistic style differences. The intuition behind using varied 3D model styles was the ability for 3D artists to collaborate and combine their various art styles together. This would give a large advantage in games and other environments, where artists need not be trained up to a very specific set of constraints for the games restrictive design process. Instead, the artists could create models which conform to some data set, but need not be accurately similar to the broader art style, and retrain the generative network.

Voxel-based data structures allow for a far simpler augmentation with current generative networks, where there is a uniform data structure, whereas mesh data files usually

Table 7.1: List of the main DCGAN hyperparameters for the results shown in this paper.

Parameter	Value
Optimizer	ADAM
Learning rate (G)	0.005
Learning rate (D)	0.00005
Momentum	0.7
L2 regularization	0.0001
Input Size	64x64x64
Batch size	8
Depth (N)	5

contain complex arrangements of data, which don't inherently work with more general network inputs.

Finally, the adaptation of commonly used algorithms for voxel-mesh translation known as Marching Cubes [87] will be shown to be applied to the voxel data, extrapolating the face colours from the voxel grid across the mesh's face colours.

Methodology

The models were all run through a voxelization process, by running a uniform 3D kernel over the mesh to calculate if a block should exist there or not. An example of a low-resolution non- textured conversion can be seen in Figure 7.5b. Different to usual conversion methods, the colour of the meshes face at the point of calculation was also carried over. This allowed the voxel to contain the additional information of the texture file. This conversion can be seen in Figure 7.5c.

The voxel data was then 'hollowed' out to remove the data from the central point of the voxel data. Only the edge data was necessary for the purpose of learning the model's shape and colour associated with each voxel. Without removing the inner part of the voxel we had no way of applying colour data to these voxels, and so would have to add one block colour or an averaged colour based on the voxel's shell colours.

Previously mentioned are issues regarding the use of 3D convolutions in their capability to scale with data size. This is an intuitive problem that comes from the added dimension when compared to 2D convolutions applied to images, which are already fairly memory intensive. We had access to some mid-range powerful hardware during the time

of experiments, and so a number was chosen in the range of the GPU memory for the size of the architecture parameters and data input size. Typically factors of 2 are used for input size on data for images and in 3D voxel applications. This is important for two reasons, kernel stride should be even as to incorporate all of the data, but more importantly because of the upsampling layers, which require some factor of 2. We chose to use 64 as the size dimensionality. An interesting note on the voxelisation process is that any size ranging from 16 all the way to extremely detailed voxels at size 2024 can be generated from a mesh.

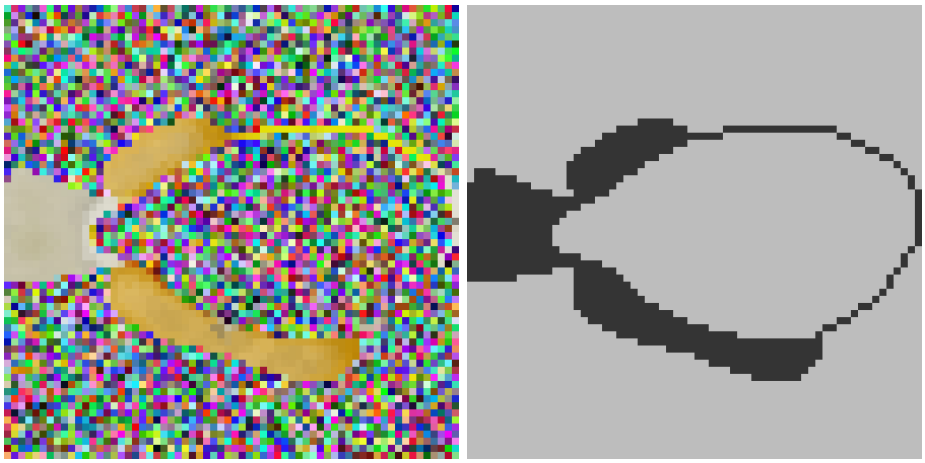
Data size was particularly important, not only because of the dimension increase but also because of the added data channel increase. In 2D image deep learning the formulae for data size would be relatively small at $n^2 * 3$ for an RGB image and $n^2 * 1$ for a greyscale image. With the addition of the colour channel, we see an increase from the size formula $n^3 * 1$ to $n^3 * 4$. The size of 64 fit all of the constraints presented to us, whilst maintaining a high visual fidelity of the object compared to the mesh, in which it could still be more than discerned from the original.

An important part of deep learning is data uniformity. Since we are using a very large box of mostly empty values, with only a small percentage of the data filled containing useful colour information, we had to come up with a representation for the data which posed no information. There were two options here, either fill in all of the empty voxels with a flat colour value, black for example, and set their voxel boolean to 0, or come up with a method of uniformly distributing the colour values over the empty voxels. Both methods were tried, and the latter produced far superior results, this was most probably due to the small distribution of important colour data being saturated by the black colour used to fill in the empty voxels.

Instead, the approach that was used was to generate a random set of colours each with a normalised value between -1 and 1, following the same convention as the normalisation of colours in the voxel shape itself. This data can be visualised as RGBA with the A channel being either -1 or 1 based on if a voxel appeared in that position and corresponding RGB channel containing the colour of the voxel between -1 and 1. An example of a cross-section of the voxel data can be seen in Figure 7.3. Further, the distribution of colour can be seen in Figure 7.4, where the histogram shape remains similar to the original

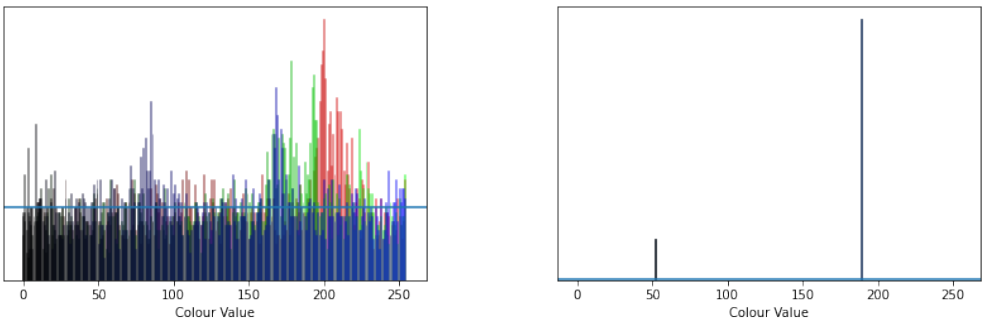
colour.

The training of the network took 10,000 epochs to fully train the relatively small data set, taking 5 hours on one GTX 1080. The addition of the colour channels emphasised the time in which the network took to train. The network parameters can be seen in 7.1.



(a) RGB Colour Channel (b) Alpha Colour Channel

Figure 7.3: Central slice of an example of one of the training 3D voxels in the X-axis. With a separated visualisation of the data preparation of a 3D model. This shows the assignment of a random pixel colour for spaces that do not contain data, to reduce the chance of sparsity of the learned region's colour.

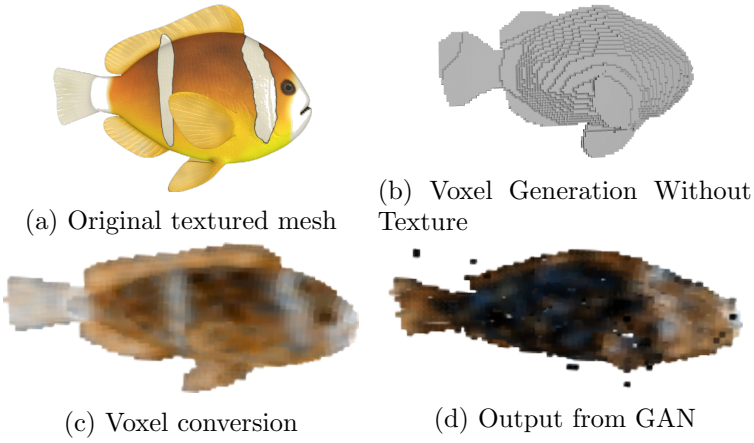


(a) RGB histogram (b) Alpha histogram

Figure 7.4: Colour histogram analysis of the data shown in Figure 7.3. Showing how the data distribution remains a similar shape, but with an increase in colour due to the added random colour variables.

Marching Cubes Mesh Conversion

Marching cubes is an algorithm for constructing 3D surface models (meshes) from 3D voxel data [87]. Marching cubes uses eight points on a cube and determines a set of polygons that could represent the points taken from the input, this operation "marches" or iterates



(a) Original textured mesh

(b) Voxel Generation Without Texture

(c) Voxel conversion

(d) Output from GAN

Figure 7.5: Data pipeline from left to right. Showing the original textured mesh converted to a voxel which is used as an input to the network. Finally, an output from the fully trained network can be seen.

over the entire grid. Polygons are generated based on how much of the eight points exist within the cube it is testing.

Marching cube examples typically extend only the surface mesh, excluding any face colours. While if the face colour is stored alongside the surface mesh a naive extrapolation of the voxel's colours could be transferred into the mesh output. An example of this can be seen in Figure 7.6.

Though the appearance of the marching cubes mesh is currently coarse, this could be alleviated through the use of hardware with a larger memory capacity. This would allow inputs to exceed the input size of 64 used in this section, which would produce a more smooth marching cubes conversion.

Results

The results show a promising contribution to model generation with inherent learned texturing within the network. In Figure 7.5 the pipeline of this technique can be seen, with the final image Figure 7.5d showing a typical output of the network. With the network only able to learn from a sparse collection of inputs, the output models were remarkably varied in both colour and shape.

There was a varying amount of noise that existed around the output image which was an artefact of training. While applying a Gaussian blur in 3D space of kernel 3x3x3

Figure 7.6: Low resolution mesh conversion using marching cubes with extrapolated face colours from a network generation.

and a low sigma of 0.05 pushed these artefact values below the alpha thresh-hold to show when rendered. It is worth noting that this noise is a typical part of the training process for voxel data, and may only be entirely removed if we over fit the model. A Gaussian blur was possible in this data due to the visible noise clusters existing in sparse areas away from the main model's distribution, this clustering approach was entirely unsupervised, and did not need to be adapted on a per model basis.

As mentioned previously the use of a marching cube algorithm provided a translation technique to transform the 3D voxel-based data into a naive low-quality mesh render. This type of mesh could be used within background renders of virtual scenes, where necessarily high detail is not needed, but the overall shape and colours can still be recognised from afar.

Previous results in the field share the same structural similarities in the model [83]. However, they lack the importance of texturing/colouring in the model. The results shown

here visualise both structurally sound model outputs alongside an interesting application of the learned textures of the models. Though it is difficult to measure the "similarity" of the generated outputs compared to the inputs, a random selection of the network generations can be seen in Figure 7.7.

7.2.4 Discussion

Current limitations with this technique revolve around the quality of the outputs, whether it be the voxel or the mesh conversion, there is a distinct lack of size and quality of learned generations compared to the original.

With this in mind future work will be centred on applying super-resolution techniques that have been used within 2D face generation using GANs [88], to provide a computationally cheap upscaling to the voxel-based generations. The better quality of voxel provides a more detailed mesh conversion when applying marching cubes. Furthermore, many voxel games use direct block-based models through the use of PCG, in which case the generative outputs could be used "out of the box" without any prior manipulation.

Although we propose the use of marching cubes to convert back to meshes and as a proof of concept is certainly an interesting idea, it is apparent that the mesh wouldn't be usable in any scenario where the model would be front and centre, however for background assets, and assets that do not require such high quality such as those far away from the camera, then the mesh conversions could certainly have some use case.

This project was originally intended to be an exploratory study and branched out into a novel contribution, however, the amount of open-source textured voxel data is simply not large enough and of not high enough quality to perform any further experiments on different data sets.

The combination of many different artistic styles into one network proves to be a very abstract and exciting concept, the generations shown in this Section and in Figure 7.7 show a wild variation of colour and model style that the various artists have applied.

Figure 7.7: A random set of output generations from the network.

7.3 Convolutional and Dense Networks on Human Point Cloud Data

7.3.1 Introduction

In this section, a novel method of generating accurate pose and animation of human point cloud data using generative deep learning methods is presented, which uses dense correspondence based data, in which all points within the point cloud align with every other point in corresponding data points. The Faust-MPI data set [89] is used, which consists of 10 actors, each holding 10 different poses. This sparse data set posed a difficult task when training a GAN, due to how varied each sample is in terms of pose, shape, and size over a large number of points (6890) per sample. But the results presented show that it is possible to generate new pose samples between the learned pose data points. Initially, the use of two GAN architectures was explored, a convolutional and a fully connected approach. It was observed that the convolutional GAN did not perform well on this type of symmetrical data compared to the fully connected GAN. The metrics for these comparisons were calculated based on a mean absolute error between correspondence from every generation at each epoch, to the closest match in the original data set.

The motivation behind this work is to help improve the process of creating content for virtual environments and video games. Currently, large amounts of time and resources are spent with designers and artists creating models for games. The proposed method generates high quality varied point clouds from existing data, which can be directly rendered with point-based rendering (PBR) or converted to a mesh using Sampled Poisson reconstruction.

To the authors' knowledge, the use of dense correspondence data has not been

explored with the use of discrete-data-based GANs. Therefore, it is shown how accurate generations can be compared to their training samples, and how new data samples can be generated when interpolating through the latent space, creating human pose animations along the way.

Furthermore, it is shown how conditional approaches can be incorporated into the model to produce both pose specific and actor specific generated data, with the focus of this section on how a user's control over the generated point cloud models can be improved.

Lastly, an important improvement to the typical binary cross-entropy loss function is presented. It is demonstrated that a pre-computed dot product value calculated across the entire data can be mapped directly onto the generated model during training. By applying the same dot product calculation to these generated samples during training, an improved loss can be backpropagated, which consequently converges to a better fit model as shown in Figure 7.10, thereby improving the results of the network.

This work compares both a trained Multi-Layered Perceptron (MLP) GAN and 1 Dimensional (1D) convolutional GAN in the early stages of the experiments to establish a baseline. However, issues were discovered surrounding the 1D convolutional GAN, pertaining to the way the 1D convolutional network generated samples that lacked symmetry, due to the lack of correspondence from points further along in the data set. Because of this limitation, the MLP GAN was solely used for the remainder of the experiments, in which optimal parameters for this network were also explored.

7.3.2 Pose and Body Generation Background

Recent work in the area of deep learning-based human model generation can be split into several sub-categories; the generation of realistic humans in images [90, 91]; the generation of realistic human motions in video [92]; recovering pose from images or video [93]; and the generation of plausible poses within a 3D scene [94, 95], though this work focuses on the pose and either uses simple stick figures [95] or variants of the SMPL model [93, 94] to represent people in scenes.

The work most similar to that presented in this section is that of Achlioptas *et al.* which [96] applies a variety of generative learning models directly to point cloud data,

including an autoencoder and a GAN. Their GAN shows relatively poor results on various household objects and was not applied to human reconstruction. In their supplementary material, they apply their autoencoder to human body reconstruction and interpolate in the latent space to generate intermediate poses between two reconstructed models. This varies significantly from the work presented here, as it focuses on merely reproducing models through an autoencoder, rather than generating new models. They also take an interesting hybrid approach, training a GAN, and separately a Gaussian Mixture Model [97], to filter the latent space of an autoencoder, before the autoencoder's decoder is applied to it. The specific function of this is unclear from the published work.

This work focuses on the holistic generation of human models, which can be used as assets in video games, animation production, or virtual social environments. While this is a more challenging problem than generating only pose, it requires only one lightweight network and no costly optimization step of fitting a human body model.

7.3.3 Methodology

Data

This work utilises the MPI-Faust data set [89] as the input data for the experiments and deep learning models in this section. MPI-Faust offers a selection of human pose data in different formats, including the dense correspondence data which is leveraged in this work.

Each model file contains exactly 6890 points, where every point corresponds with every other point in the data set. The data set consists of 10 unique models, of varying body structures and shapes. There are 10 poses mirrored across every model. This dense formatting of the data allows for uniform learning of complex floating-point data through standard generative approaches, such as a Convolutional neural network or a static fully connected network.

In the data set, there are various poses, from models with their hands in the air to body rotation poses. An example of four randomly sampled poses can be seen in Figure 7.13. The points are uniformly distributed when rendered into world space. There are large variations in the data set which pose a challenge to train through a GAN, such as the

size variation of the models, and the pose structure of the models. The goal of the network is to generate structurally similar models while keeping visual fidelity and generalisability high.

Network

This section will discuss the comparison of the 1D convolutional and the Multi-Layered Perceptron (dense) approach. Based on the dense ordering of the data used within this section, two networks that were order specific were chosen, an MLP fully connected network and a 1D convolutional network. These network architectures allow for a quick iterative training process to determine optimal parameters while retaining the important dense correspondence of the data structure when creating outputs. The specific performance of both of these networks on the human pose point data is investigated.

The use of a 1D convolutional network on this data is roughly equivalent to applying a 2D convolutional network to a 2D image, as all pixels in an image are ordered; this type of filter network convolves across the entire data set. Similarly, the proposed 1D convolutional neural network learns the same filter patterns across the entire dense correspondence Faust data set but in one dimension.

The MLP network is simply a fully connected network where every node in subsequent layers is connected to the previous layer's nodes. For the generator, the number of nodes increases with a factor of 2 for each layer. The final output of the network increases to the size of the point cloud data, 6890, which is the size required for the input to the discriminator to match the real samples. The discriminator is a reverse of the generator, leading down to a node size of 1 with a Sigmoid activation, signalling if a sample passed in is determined real or fake. The network layers and size used can be seen in Table 7.2.

Comparison and Issues

Initially, the use of a 1D convolutional neural network was tested, as it was determined this would be a better model to use for the type of data, where close neighbouring points were deemed to be an important part of the data set. Data preprocessing was necessary for the 1D convolutional approach, due to upscaling constraints in the generator, the data

had to be in a factor of 2. Therefore the dataset was zero-padded to achieve a size that the discriminator could handle. This increased the data size from 6890 to 6912.

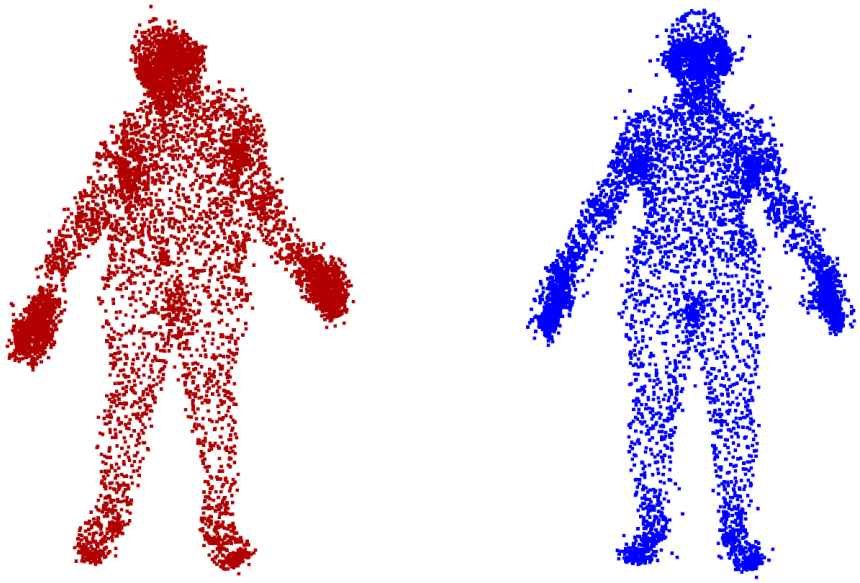


Figure 7.8: 1D Convolutional generation on the left (red) compared to the MLP generation on the right (blue). Notice the lack of symmetry in the 1D convolutional approach, with the left side of the point cloud slumping. Though the structure of the point cloud is picked up, the 1D struggles with more dense areas such as the hands and face when compared to the MLP network's generation.

However, from the beginning issues were found while using the 1D convolutional neural network, these were systematic to the data, given how the data was ordered. Because the points were flowing in a clockwise manner from the left-hand side of the model, it was found that when interpolating between the generated samples the models would slump to one side. An example of this can be seen in Figure 7.8.

This problem was alleviated when moving back to the simpler MLP approach. Because all weights were connected to every other weight, the problem shown in Figure 7.8 was nonexistent, and the symmetry of the data was maintained. Due to the network learning the data points relative positions to all other points, rather than the relative positions to the neighbouring points, this problem stemming from the convolving window approach.

The difference between the performance of both networks can be seen in Figure 7.10. This metric was calculated as the average distance between every point in a generation against each original training input.

Parameters and Parameter Exploration

A random parameter exploration based on a set of different network sizes and specific network hyperparameters was conducted. The mean absolute distance calculation shown in Section 7.3.3 determined how well each parameter set performed. Table 7.2 shows a list of the parameters which performed the most optimally together, giving the lowest mean absolute error, and consequently the parameters used in the final model. A range of preset random parameters was investigated, with the main exploration focusing on the depth of the network and the width of nodes used in each layer.

Parameters were rated based on how quickly a run would reach convergence. Convergence was defined in this exploration as when a model was producing structurally similar results, with the mean absolute error metric less than 100 for an average of 50 epochs. The closer the mean absolute error approached zero, the closer to an identical match to the original data the generation would be. It was found that stopping training when the error was around 100 allowed the network to generate a varied selection of human point cloud structures.

Each layer in the generator had a Relu activation. With the final layer a TanH activation was used, and a dense layer of size 6890 mimicking the original data size, while each layer in the discriminator used Leaky Relu, with the final layer condensed down to 1 dense layer with a Sigmoid activation.

The generator and discriminator also contained dropout [98] layers before the final layer, with a drop out of 0.4 (40%) used. The dropout function was removed from the generator post-training. This dropout assisted in the reduction of overfitting to the relatively small MPI-Faust data set [89]. Dropout temporarily removes weights of the network's layers during training to reduce the reliance on certain highly activated nodes in a layer.

Each model was trained on a single GTX 1080, with an average run to 50K epochs taking roughly 90 minutes, though the training was often stopped before this mark due to the potential of overfitting. The comparison of training times can be seen in Table 1.

Table 7.2: List of the main MLP GAN hyperparameters for the results shown in this section. ^{i} represents the current layer indexed at 1. These were determined by epochs to convergence, and verified with a visual evaluation of resulting generations.

Parameter	Value
Avg. Epochs to Converge	13500
Optimizer	ADAM
Learning rate	0.0005
Momentum	0.5
Batch size	16
Gen. Depth (N)	6
Disc. Depth (N)	5
Latent Dimension	100
Generator Layers	128 *(2^{i-1})
Discriminator Layers	1024 *($1/2^{i-1}$)

Loss Function

Initial experiments were performed with a standard binary cross-entropy loss, however, using this loss meant convergence to the human model shape took a large number of epochs. Therefore we investigated methods of efficiently accelerating this process. The only caveat is that the function must be calculable on the generated samples independently.

The idea of using the dot product is to add conformity in the early stages of training, specifically around the distance and angle between points. Because the data is in dense correspondence, with every point moving in the same direction to form the human model, a pre-computed dot product across the training data can be used to determine how well the generated samples conform. A sample of the dot product of all of the training data was taken, where each dot product was taken for every point in each data point. This came out at an average of 0.33 ± 0.055.

Initially the calculation would take a point (x) and the next point in the data file (x+1) - where $D = (N + 1)$. Indexing all of the points proved far too inefficient to use during training, so random jittering was employed. Jittering can be seen as adding a small stochastic displacement to the initial index. This was changed to a stochastic approach where now - $D = Dot(N, N + step)$ with the step being a random value between roughly 0.8% and 1.2% of the data set size, the dot calculation was performed until N >= data length. This helped with performance without drastically reducing the quality

of convergence to the pose shape.

These comparisons can be seen later in Figure 7.9, where the addition of the proposed Dot product based calculation impacted the average number of epochs on which the network would converge from random noise to the human pose. The final equation for loss used 20% of the dot calculation (D_i) and 80% binary cross-entropy (BCE) and where the average dot product for the data set is 0.33, shown as -

$$Loss = abs(0.33 - D_i) * 0.2 + BCE * 0.8$$

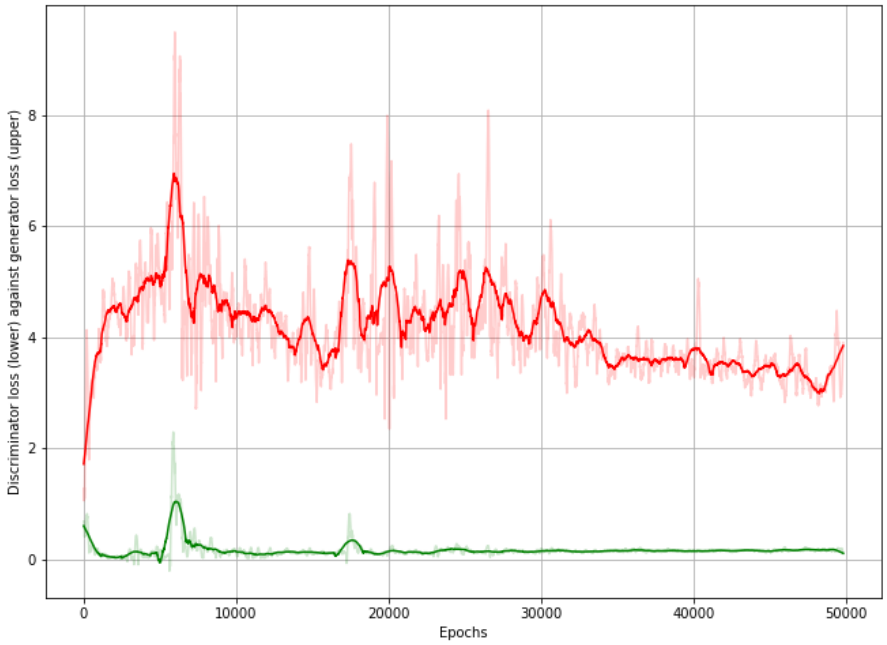


Figure 7.9: Loss plot of 5 training runs averaged. Upper (red) line shows generator loss, lower (green) line showing discriminator loss, over 50000 epochs of training. This showed the point of over fitting, around 35000 epochs, when training was stopped for the results shown. This value is comparable to Figure 7.10, where at this epoch the mean absolute error starts to slowly descend to 0 signifying the start of, or complete, overfitting.

7.3.4 Results and Evaluation

This section will discuss the results both visually and quantitatively, with a focus on how the results were varied enough from the original data, but with the ability to generate data with a structured point distribution. Also, it will be shown how the points can be reconstructed into a mesh using Poisson reconstruction, including some of the shortcomings

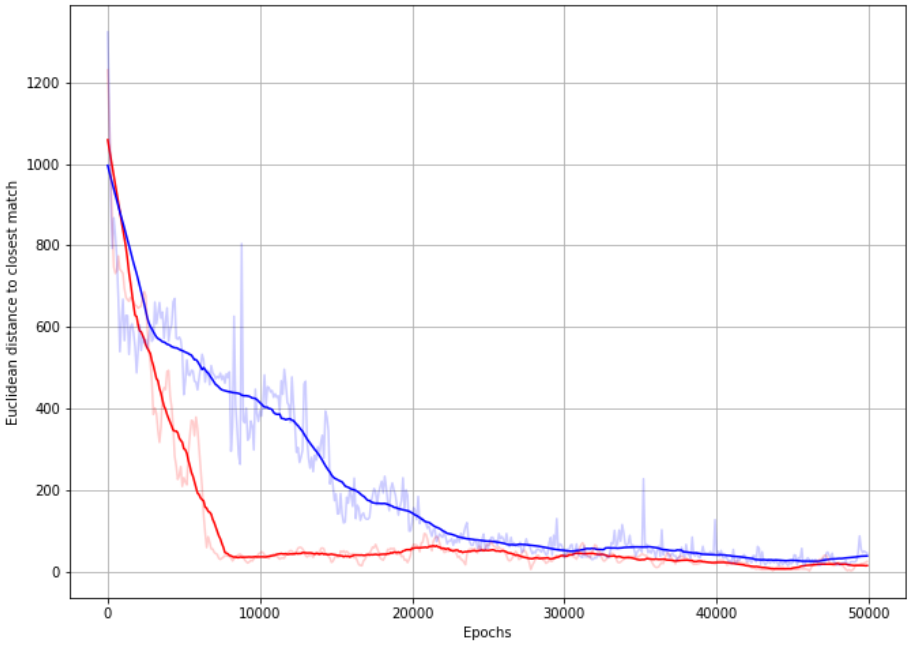


Figure 7.10: Mean absolute error to the closest sample metric. Performance of Dot product loss (red) against standard binary cross entropy loss (blue). Data gathered over 5 full runs per loss model of 50,000 epoch, with the solid line showing an averaged smoothed plot using a Savgol filter, and the feint line the actual averaged values.

of using such a method on point cloud data. Lastly, the exploration of latent space interpolation to create new samples, and the possibility of using an interpolation method to create animations.

Generations

An example of the generated data and original data can be seen in Figures 7.12 and 7.13 respectively. An example of sampled generations during training can be seen in Figure 7.11. These visual renderings of the point data show the differences between the original data and the generated samples from the network. These points were rendered with no other lighting/colour information added.

There is a clear overall quality loss between the original data and the generated data, however, the structure remains to a high enough degree to accurately recognize the human pose and body shape. As an overall visual evaluation, the model learned the pose and shape well. However, at a lower level, it is apparent that certain heavily dense areas

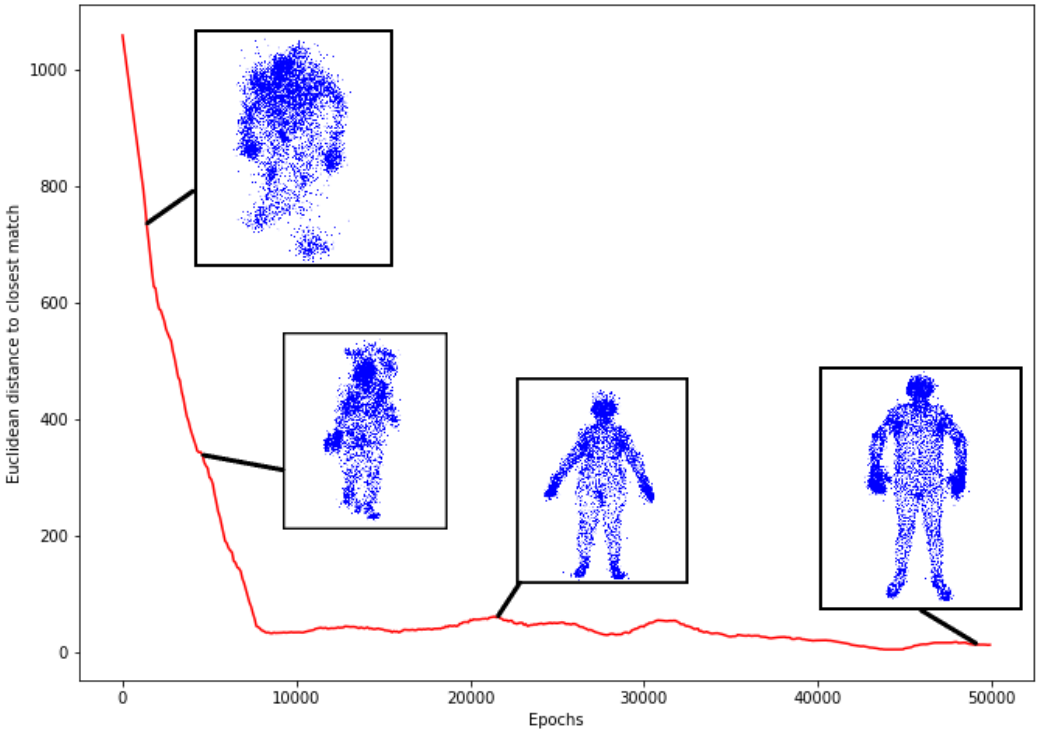


Figure 7.11: An example of the training plot over time, with the mean absolute error plotted. On the plot are 4 example generations through points of the training process.

of points posed a more difficult problem; The fingers, feet, and face, although structurally similar have some distinct lack of quality. There are several points in the generations that float away from the main structure. These points can be cleaned up with a median filter to improve the quality, however, the authors decided to leave the results as a direct output from the generator network. One can observe the uniformity of the points in the point cloud which appears less structured compared to the input data.

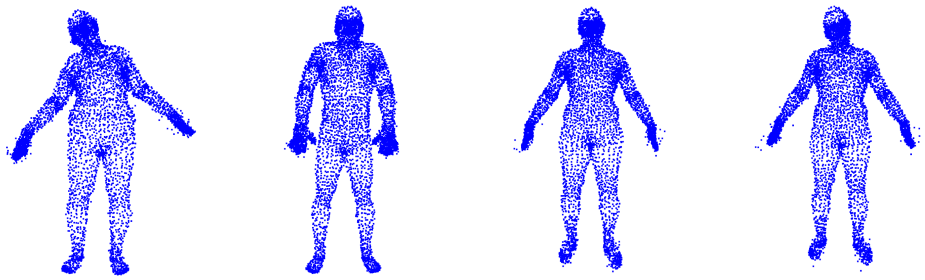


Figure 7.12: Multiple randomly sampled generated examples from the MLP GAN. Each model shows a different type of pose and body shape. More varied results are shown in the interpolations in Figure 7.15

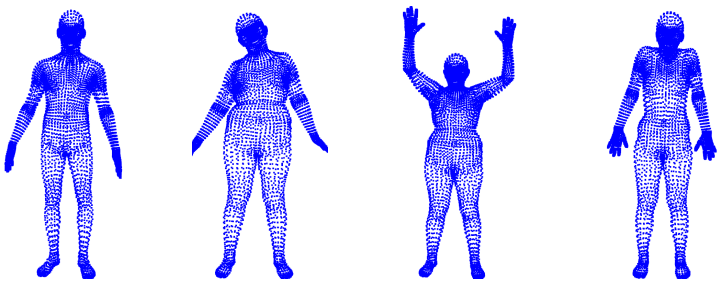


Figure 7.13: Randomly sampled data from the original data set

Reconstructed Point Cloud

Poisson reconstruction has been used previously [82] to reconstruct meshes from point cloud data. Screened Poisson [99] aims to improve the original algorithm by reducing the importance of noise through interpolating between values in surrounding areas of the point cloud.

The Screened Poisson technique can be applied to the generated point cloud. Below, in Figure 7.14, is a visualization of a randomly generated sample with Screened Poisson [99] applied. It highlights a few issues with this technique, and while the large structures are captured by the GAN, some intricate and more complicated areas, such as the hand and face, have not been captured well by the network. Though it is worth noting, even when reconstructing the original point cloud data to mesh, the quality was also lost in these areas.

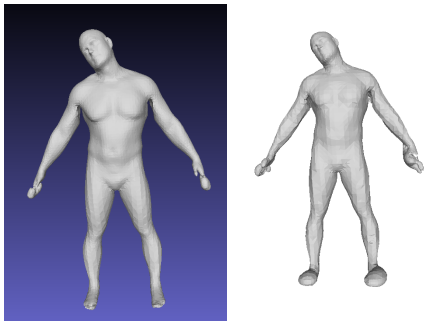


Figure 7.14: Point cloud generation reconstructed using Screened Poisson reconstruction. Parameters - Tree Depth of 13, minimum number of samples of 3 and interpolation weight of 4. A real data conversion so shown on the left and generated on the right. Both exhibit issues when sampling the hands, and more of an issue with the feet in the generated sample.

Interpolating In Latent Space

Latent space is the hidden variable space where high-dimensional features are encoded through the network's training process. New samples can be generated from the network by generating random latent noise and passing it through the generator. Interpolating between two or more of these random noises can generate new samples that lie between the two representative data boundaries.

For human pose generation, this type of latent space generation is extremely useful in the creation of new poses, and even GAN based animation between one pose and another.

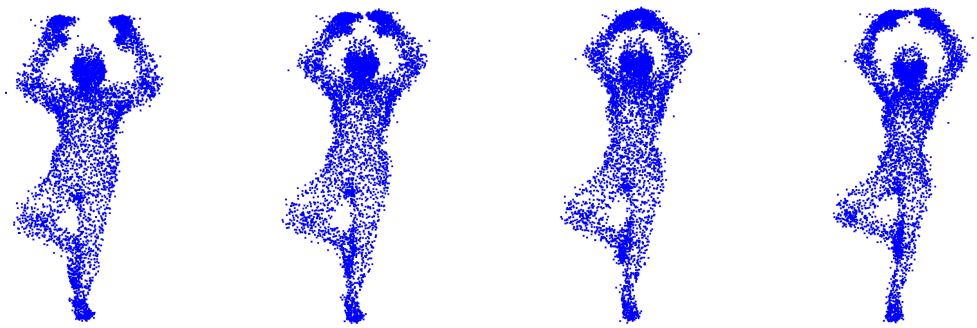
An example of several random latent generations can be seen in Figure 7.15. A Sigmoid based interpolation function was used, where more values were included at the midsection of the interpolation compared to the edges. This helped provide a smoother interpolation than that of a linear interpolation, with the context of creating an animation style transition between generated values. This interpolation function can be seen in Figure 7.16.

There are 4 samples shown in Figure 7.15, varying from body pose interpolation, to limb movement interpolation. The figures show several frames of generated samples between the left most and rightmost generation. These interpolation examples behaved as expected, and each generation moves the necessary points in the point cloud to control the limbs or body shape when transitioning through latent space.

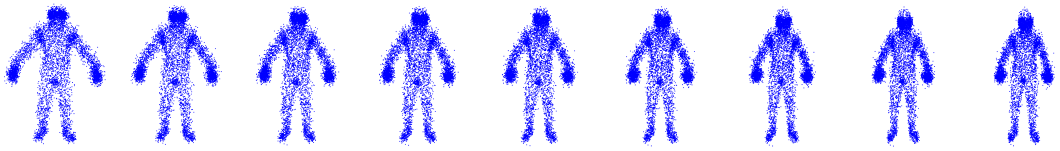
7.3.5 Conditional Approach

Conditional GANs [39] provide the ability to train a GAN with one or more additional elements of supervision. Using these labels aids the GAN in generating examples that match a specific label. This would help potential users of this tool to generate a specific pose or size of the model that they require.

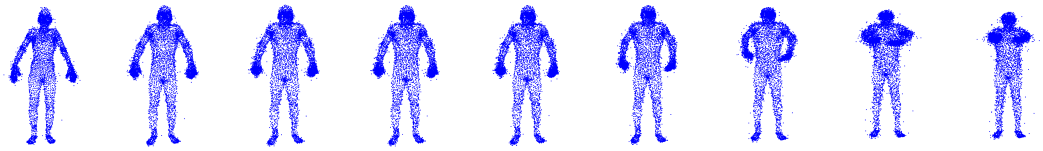
Having determined the quality of the proposed model after training on the data purely unsupervised, it is demonstrated that given a set of conditions, the model can be trained to output generations constrained to these conditions. Though, given the sparsity of the data, conditions were limited to the type of label that would fit over the entire data



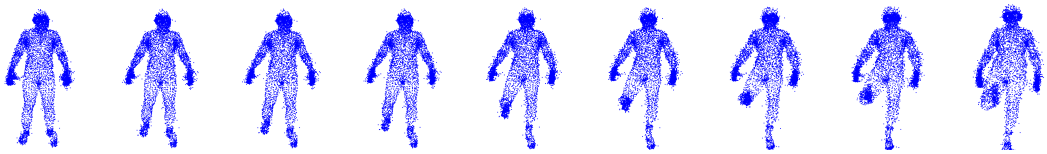
(a) Body size interpolation - shows a slightly larger model reduce in size to a smaller model, while retaining pose



(b) Body pose interpolation - slight arm movement moving model arms inwards.



(c) Body pose interpolation - Moving arms upwards towards a horizontal arm alignment



(d) Body pose interpolation - Single leg movement with the leg bending behind the model.

Figure 7.15: Selection of sampled interpolations, showing the variability of the interpolation function of the GAN. Different poses and body shapes are interpolated between in the latent space of the GAN.

set. This label needed to constrain the data to enough samples per label, while maintaining a spatial difference in each label's associated point data. The pose was explored as the main condition element, each of the 10 poses was given a unique label for each of the actors. An additional input and concatenate layer was added to the MLP GAN that was used for the experiments in section 7.3.4 of this section. However, the input to the generator and discriminator was adapted to include the additional conditional labels. Embedding layers were also used in the generator to create a tensor of the same size as the latent noise. The generator followed the same pattern of using an embedding, but this embedding was reshaped to the same size as the input data and concatenated together. Due to the increase in data from the labels and embedding layers, the capacity of the nodes in all of the layers was increased by a factor of 2.

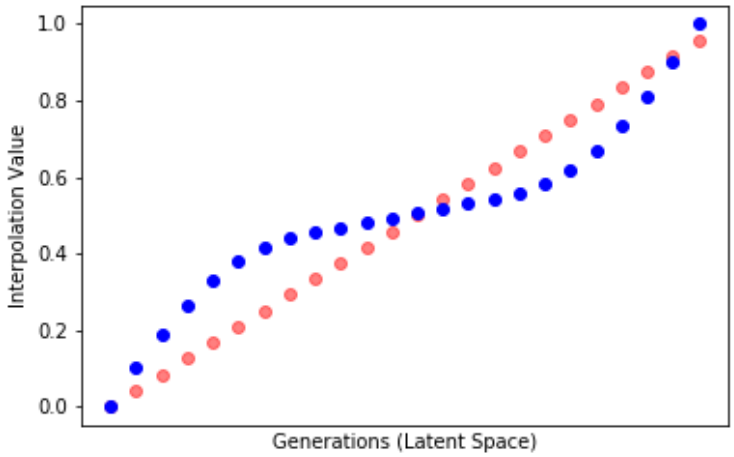


Figure 7.16: Improved use of a weighted interpolation (blue) vs linear interpolation (red). Where more values are sampled from the middle points of latent space compared to the edge values. This assisted in a smoother generation of animation style interpolation between generations from noise N_1 to N_2

.

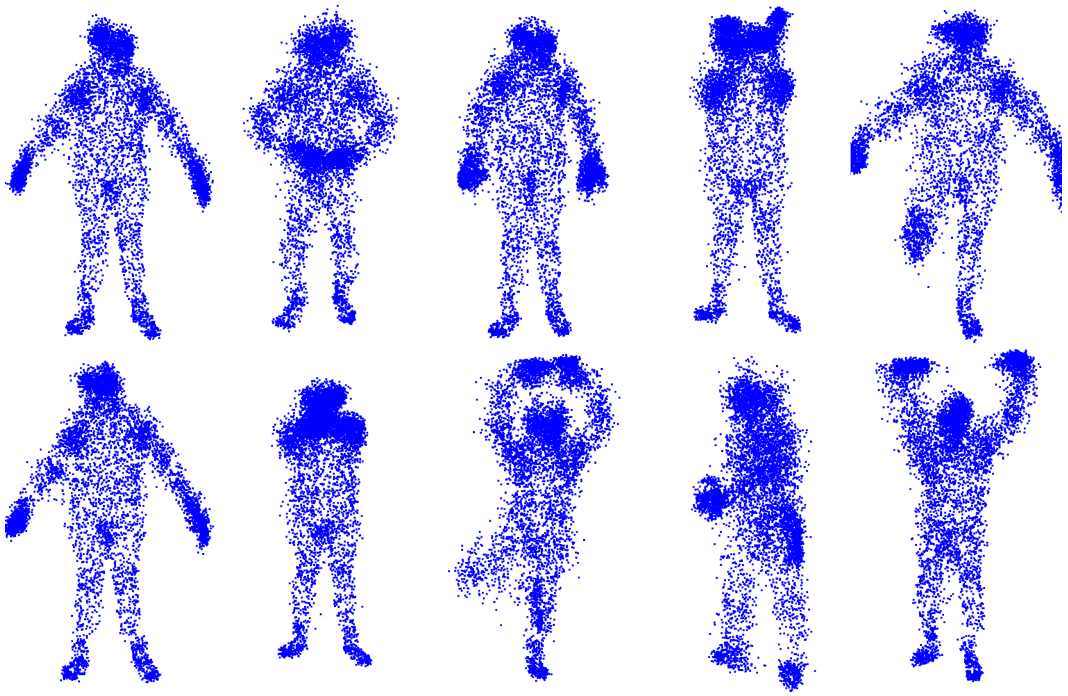


Figure 7.17: Conditional GAN output of the 10 poses, showing varying body shape and size. Each generation is the result of using one of the 10 trained labels as an input.

The results for the conditional approach can be seen in Figure 7.17. Each image in the figure shows a unique class of pose learned from the data set, and although the pose is correctly learned for each model, the quality of output compared to initial experiments (shown in Figure 7.12) that had no conditional labelling, has been largely reduced. It is

believed that the reduced quality can be attributed to the small data size of 100 samples, which proved to be a difficult task to train a GAN with the additional label data.

7.3.6 Discussion & Future Work

This tool would be used to quickly generate procedural human point clouds in real-time, allowing for games that previously-stored large amounts of model files to simply generate them through the users' CPU or GPU during a video game, or simulation.

This work demonstrates that interpolating this type of data can provide very interesting animation style groupings; where previously the data would have to be rigged for animation by manually selecting body parts, one can now simply pick two poses as the start and end frame of an animation. This offers artists and designers a new type of design tool to potentially assist in the creation of human models with specific poses and body shapes, with a large list of interpolated generations to pick from.

Something not explored in this body of work was the inclusion of additional data channels alongside the positional data. Work has previously explored going from single points of data to two or more additional points [100] and found no loss in quality of generations. Therefore specular, colour or even normals could be examined as potential additional data points. This would allow models with clothes or skin colour to be generated from the network. More importantly with the use of accurate normals, Poisson reconstruction could be much more accurately applied to the data alleviating some of the visual issues shown in Figure 7.14.

This section outlined a new method of loss calculation for ordered point cloud data, together with a deep parameter exploration of two neural network architectures has resulted in a robust method of generating new human pose and human poses animation from existing point cloud data. The interpolation function shows a smooth generation of human point clouds between two randomly sampled poses and shaped models.

The idea of taking a prior dot product calculation and incorporating it into a weighted binary cross-entropy loss function provided a large stability increase when training the generator of the network, subsequently improving the visual fidelity of human pose outputs and early training convergence. Two different approaches were compared,

demonstrating the presumed better performing 1D convolutional network to act poorly on this type of directed dense-ordered point data. Lastly, conditional GANs was explored as a potential solution to the inherent lack of control that exists when generating random samples without conditioning, showing that, while possible, the quality of the generations were below the authors' initial experiments, which the authors believe stemmed from the small size of the dataset.

Overall the applications shown here are extremely computationally cheap and effective methods of generating varied but lower quality generations of point-based data. The ability to interpolate using GAN based architectures opens the door wide to the interpolation exploration, and the use of GANs for a specific animation. Added conditional controls through the latent variables could be used to directly control how these animations occur, and for example use in games and other virtual environments could provide procedural and entirely new ways to animate the point-based data. Work is definitely needed on the reconstruction and texture-skinning of these point-based generations.

7.4 Graph Convolutions on Human Point Clouds

7.4.1 Introduction

Graph Convolutions (GCNs) are introduced in [101]. GCNs are a type of convolutional neural network that can work directly with graph data, using the structural properties of a graph's nodes and edges as information. The idea of GCNs is to take data from a node, and also its immediate neighbouring nodes, at the most basic level one could take an average of these nodes and aggregate them to form one value on the main node which holds a representation of the immediate connections. In a way we can compare these to normal CNNs which, when applied to data like images, are fixed graphs with a fixed number of neighbouring pixels. CNNs are spatial models and the operation is performed understanding the spatial elements of the data, such as an image. Graphs on the other hand have no spatial capacity.

The layers of a GCN usually correspond to the degree of neighbouring nodes, with layer one only taking a weighting of one neighbour, layer two would incorporate two degrees

of neighbouring nodes, and so on. Up to N layers which would incorporate N degrees of connected neighbouring nodes.

Typically prior knowledge is needed for graph compositions, in particular classification; this is not needed for generation as it has to be non-deterministic to ensure the vast shape and size of model types can be generated. Up-sampling in the generator is done by using branching, which essentially starts to create the graph structure, from the root node in layer 1. The graph convolution operations are then applied at each layer

TreeGCN is a graph convolutional GAN that aims to improve on prior work by using a tree structure, this tree structure allows the use of ancestral nodes within the graph convolutions and such can understand segmentation data using the learned graph dependencies. We used this architecture to show how human point data can be generated in an entirely unsupervised setting, with purely *unordered* point data.

7.4.2 TreeGAN

Data Parsing and Initial Limitations

Figure 7.18: Training data parsed in to a usable format, reduced down from the original graph size to 2048 and 4096 points respectively.

Based on experiments outlined in Section 7.2.4 we aim to investigate how graph-based deep learning can be leveraged to improve the quality, and overall effectiveness of this type of data through a more refined unsupervised approach, when compared to the data in dense correspondence, used previously.

Data used in this section is the same MPI-Faust human point correspondence data

that was used in the previous section, comprised of 100 samples with 10 models in 10 different poses.

The initial advantage we obverse when using GCNs and parsing point data into a graph, is that once data is automatically processed into some graph representation the data can be trained in no particular order, allowing for larger data sets that would previously have been unusable in the current field of generative deep learning to now be accessed and freely trained.

One difficulty however is generating graphs from point data in a way that disregards the data's correspondence. The method we chose was to use a Euclidean distance calculation on each point in the point cloud, this would determine a cluster of points around a localised point which would act as a node, each of these connections would form edges, and a graph would begin to form.

The obvious flaw here is the scalability of generating a graph this way; if we use the initial point cloud data of size 6980 then the number of connections in the graph will soon explode to an unmanageable number. This can be further seen when plotting the data across various reduction scales, Figure 7.19 shows the exponential increase when averaged across all samples in the human point data set. The reduction was performed with a simple common sampling technique, which shuffles the data, and then takes that number of samples from the beginning of the array. As long as enough points still exist then the distribution of the point sample will remain and that is all we need to construct the graph with enough data to learn its prior.

Another limitation is the ability for points which have no direct relation with one another to be connected via the distance metric; this could occur with the hand being close to the torso for example, potentially making one or more non-relational connections. For this to occur the hand would have to be practically touching the torso, in our data none of the models exhibited this behaviour and so we do not know the affect that this would cause on the outputs. But could speculate as long as data is varied enough in the data set, it should not matter.

Figure 7.2 shows the final construction of a human point cloud parsed into graph form. From this figure it is straightforward to see where clusters lie within the densely

connected nodes of the graph. This graph was formed using 0.1 as the local distance of all connected nodes, where the data was normalised in the range of 0 to 1. Figure 7.18 shows the renders of two size reduced point clouds, in to a smaller format use able by our network.

With a way of reducing point values to constrain to that of memory usage for our hardware, it was important to find several points which would suffice the quality needed but would not spike the graph connections. 2048 and 4096 were two sampling frequencies that were found to work well within our constraints, with the latter taking much longer to train, but producing slightly better quality generations.

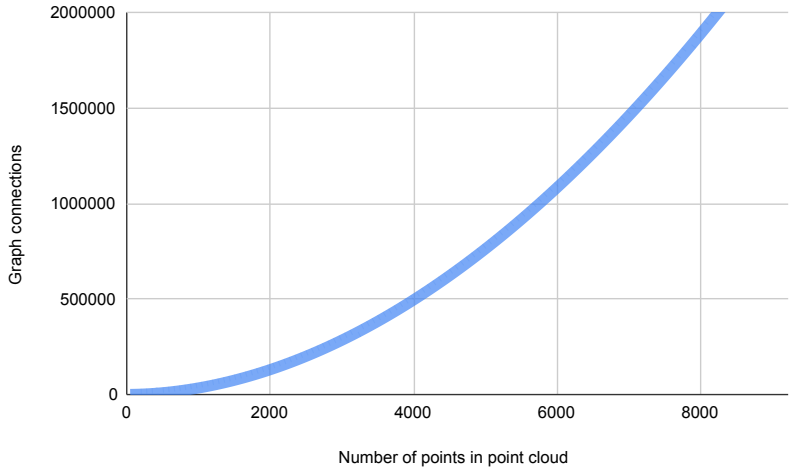


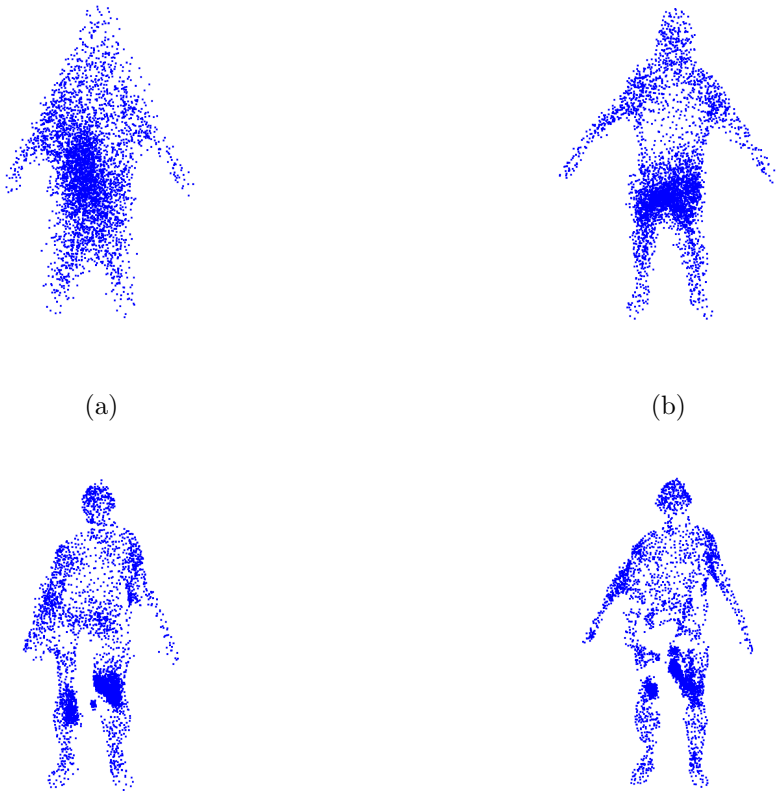
Figure 7.19: Visualisation of the exponential graph connection increase as the number of points in the sample input increase

7.4.3 Training

Network training was done using a GCN tree-GAN [102], TreeGCN, a pre-existing novel GAN that converts the typical graph based structure in to a tree data structure. TreeGCN works extremely well on spatially hierarchical data, with the tree having direct access to ancestor information

During training we used Adam for optimization of the network, with a learning rate of 0.005. This learning rate was decayed over time. Given the small amount of data available in this set we set the batch size to 10, with 100 samples being used in the training process.

Our early experiments indicated the need for early stopping, and that there is a fine line between a well trained model and one that collapses. We can see in Figure 7.20 where the generations over epochs are shown, with the early training showing poor dependencies but a good structure, with later generations showing a far too great dependency. The fourth figure in the set which was at epoch 20,000 shows the network starting to degrade, losing dependency information between key points around the groin area.



(a) (b)

(c) (d)

Figure 7.20: Training of human point data overtime using a GCN, generations are recorded at uniform epoch stages of 5000 (a), 10,000 (b), 15,000 (c) and 20,000 (d)

7.4.4 Results

Although graph convolutions allowed for an unsupervised approach on point cloud training, we observe a distinct lack in quality namely around specific regions and point clumping. Contrasted to the point cloud approach in the previous section, where all points were required to be in dense correspondence, which is the complete opposite of the unsupervised approach here, boasted much higher quality generations.

One technique we could use here is to remove points that appear in highly dense areas, this would result in the large masses around the thighs being removed and a much smoother point model left. We are currently unsure of why this occurs in the data set, as most other research papers show results which are far more consistent.

It appears that large amounts of detail from the hands and arms have been lost, and these points have gravitated towards the thighs. Though it is worth noting that the point distribution in early training shows more detail in the arms and hands but still suffers from the pooling of points around the groin/thigh.

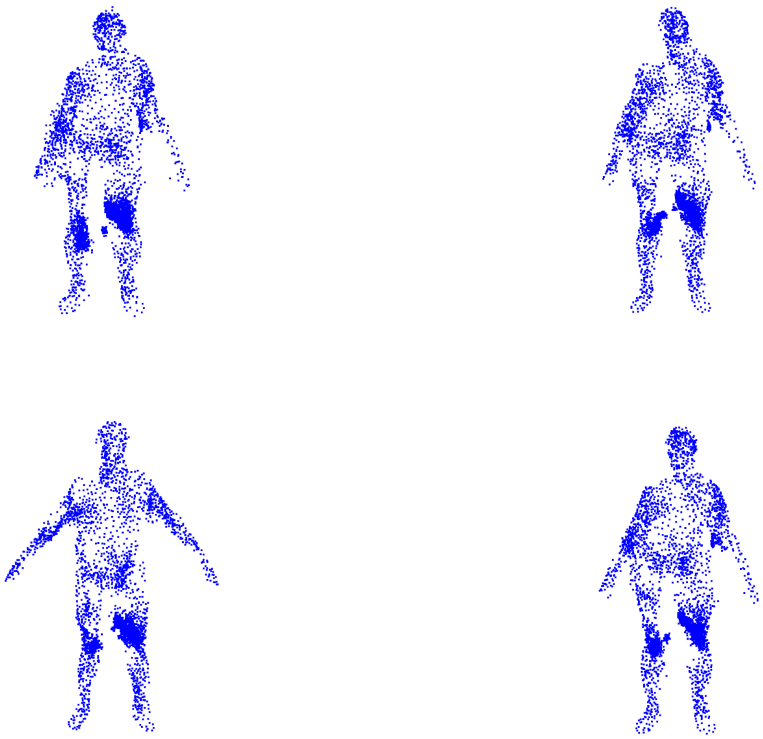


Figure 7.21: An example of multiple outputs from the GCN trained on human point data at 4K point resolution. Two different pose and body shape mopel generations.

7.4.5 Comparisons to Prior Approaches & Limitations

Figure 7.22 shows the three generated models using three different techniques outlined in this section. We can observe that the MLP approach produces a much cleaner model with a consistent distribution of points, with similar point density to original samples. The 1D convolutional approach boasts similar results but lacks the intricate detail around

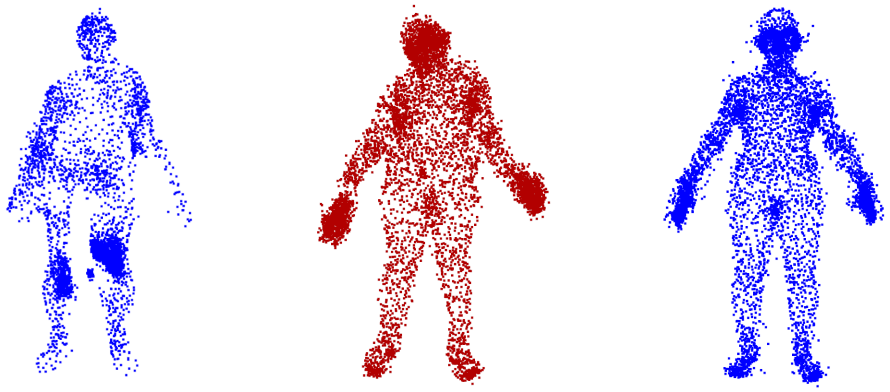


Figure 7.22: Visual comparison of GCN (left), with supervised approaches from 1D (middle) and MLP GAN (right) for renders of the human pose

sensitive areas such as the face and hands; this method as outlined above also suffers from symmetry issues due to the nature of the convolutions and the points fed through the network, we observe one half of the model not aligned with the other. Lastly, the GCN generation exhibits good point distribution but fails to distribute large blocks of points resulting in a model which has a patchy and inconsistent appearance.

We believe that the best results shown here are the MLP GAN approach, this method boasts the most visually similar distribution of points when compared to the initial data. Furthermore, the interpolation within the latent space allows for pseudo animation style handling of the point cloud generations, which when combined with conditioning creates a powerful idea for animation handling in game environments using pre-trained GAN networks. However the need for data pre-structuring is noted and taken in to account, and something that unsupervised approaches like GCNs will certainly outperform as this research area is refined.

PCG would benefit from this type of rendering style, with pre-trained networks able to be deployed to games and ran in real time, creating entirely new models on demand. The fact that the processing power can be front-loaded to the training process, with very little processing power needed to actually run the generator component. However during intensive games GPU power may be restricted and so wouldn't be possible, however perhaps networks could be loaded and activated during times of GPU in activity, with data stored ready for future use.

7.5 Summary

Overall we have shown 3 different approaches to 3D model generation, with 2 focusing on point data and the third on voxel data. 3D content generation is an extremely broad field. We explore the use of different generative networks, from convolutional to simple multi-layered networks, and how their properties exploit the various data types we chose to explore in this Chapter. With relation to deep learning networks, both data sources were straightforward enough to train, as far as 3D data sources go they can often be far more complex, such as .obj file representations or complex non-corresponded point data.

Voxel Data, has an inherent size constraint, meaning the use of convolutions and full connections are trivial to apply, however, their capability has many limitations with GPU memory, and scale to the power of 3. Though at smaller dimensions are very fast to train, and can even incorporate many more data channels of information, such as colour, as included in this Chapter, but also lighting properties, or other surface related information. The voxel training process was very straight forward, and almost always yield solid generation models from a training period. Something we would liked to have done was experiment with different type of voxel data, such as human models, however to bring the complex human model data in to a visually understandable voxel form required a much larger voxel depth than the hardware could manage, therefore we were relatively strapped for what model could fit in to this generation method. Results shown in Section 7.3 of this Chapter were done in collaboration with SIE Sony.

Point data is a continuous form of voxel data, that unlike voxel data has no wasted data containers. We found that training this type of data was not trivial however, and unlike voxel data require some completely varied GAN architecture, or to find a method of data pre-processing. The discovery of dense correspondence data made the initial experiments fairly straight forward, and allowed the use of two discrete data GANs, with the MLP out performing the 1D convolutional GAN. Later experiments would look in to graph based networks to focus on non-ordered point data and boasted some impressive results for the lack of labelling, but would ultimately have some inconsistencies that meant the technique as a PCG network would be lacking.

All methods exhibited in this chapter show a strong use case for PCG, providing, once trained, lightweight networks that can create procedural variations of training data. The timings for these generations are also extremely competitive with the current state of the art non-deep learning techniques.

The field of PCG is moving on from the roots of highly specific content creation algorithms, to a more generalised data-training approach, something that takes in to account the near endless amount of publicly accessible data, and crowd sourced labelling of data.

Chapter 8

Conclusion of the Thesis

8.1 Summary of the Thesis

In this thesis there has been a presentation of many novel techniques that directly contribute to the field of computer vision, and more specifically procedural content generation.

Where possible we aimed to back up any work with rigorous tests, both quantitatively and qualitatively. One of our aims being to push qualitative testing more into the realm of deep learning, an area that currently suffers from a surge of extremely specific equations to maximise scores for comparisons of image and model data in niche scenarios, something that qualitative testing alleviates, and provides a much broader picture of techniques strengths and weaknesses. This is especially useful when content is applied to an area of direct user interaction such as games, or other interactable environments.

We have shown a novel application in both 2D and 3D fields of image deep learning, both of these fields are backed up by peer reviewed conference papers that show clear outlined novel contributions, with the papers restructured and composed in a way that fits in to a single narrative throughout the thesis.

Height and terrain maps became an important and fairly large work inside the thesis, this was largely due to the lack of research in this area, and as of writing this becomes my second highest cited piece of work, with many people contacting me asking for future ideas and help setting up the experimental pipeline themselves.

The 3D work in this thesis acts as a way of tying in some additional experiments and collaborations done during the PhD.

The approach I took for the PhD was to encompass as much cool and interesting research as possible, regardless of the area, whilst ensuring any of the work could and would be added into the thesis. This allowed for a very free approach to research, where the direction wasn't necessarily fixed throughout.

8.2 Future Work & What Didn't Work

This section will aim to outline any ideas and research avenues that I simply did not have the resources to investigate. Many of these stem from the main block of the thesis being height map and real world terrain authoring and form a continuation of the field of PCG with deep learning.

8.2.1 Semantic Control

Conditional applications of deep learning have shown the ability to allow coloured representations of particularly in areas of an image, which when drawn on to a blank canvas and added in to the generator during GAN reconstruction can assist in the creating of a more tailored image. These applications can be seen in Neural Doodle [103], a method of semantic style transfer.

An example of the early experimentation of conditional control can be seen in Figure 8.1, where we segregated height data images based on height banded channels, however the learning of this type of segregation map didn't produce results of high quality which can be seen in Figure 8.2; In the figure the overall structure of the semantic reference is true, but the underlying detail of the terrain has almost no detail, and repeats small patterns from the terrain.

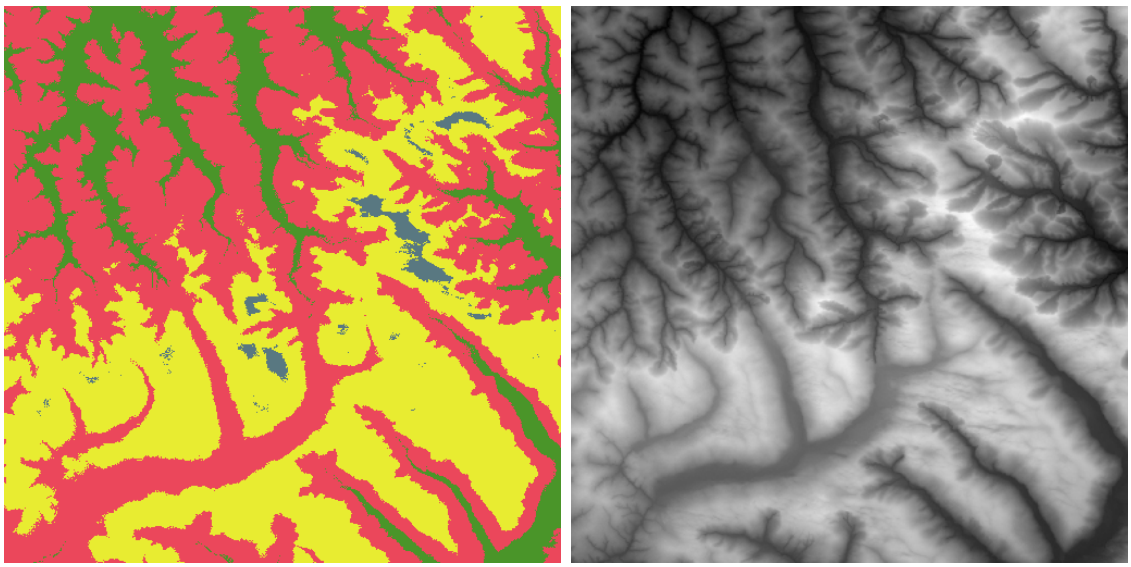


Figure 8.1: An example of the attempted application of image semantic segmentation for Neural Doodle. Colour channels are generated using a set of bands from the heights shown in the height map.

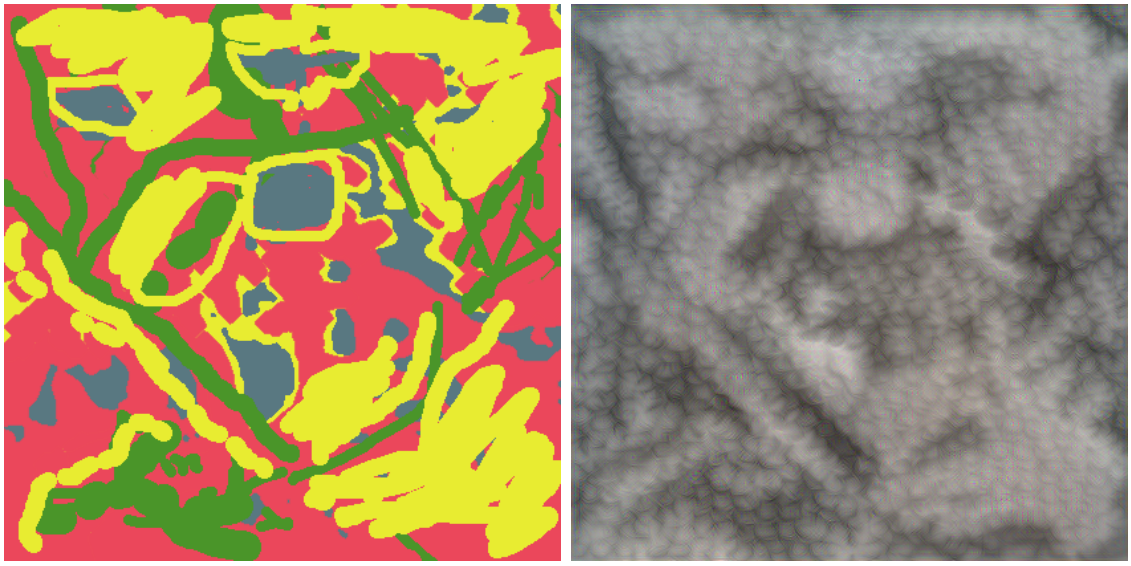


Figure 8.2: An input semantic reference with the resulting output. Succeeds in learning the overall structure of the reference image, but fails the more intricate details.

8.2.2 Human Voxel

Our work shown in this thesis on 3D models, and more specifically voxels, showed a great deal of promise. Increases in the capacity of voxel size is an obvious improvement, and something, that with access to GPUs, with larger memory and/or multiple connected GPUs could allow these experiments to be run with larger dimensions.

We also wanted to experiment with human mesh data transformed in to voxel, with

the added exploration of skin colour, something that would need to be uniform across a single generated voxel model, as unlike our example of colourful fish models skin colour is nowhere near as extremely varied across a given human model.

8.3 Future of the Field

I would like the direction of the deep learning and generative deep learning field to take a much needed turn towards controlled quantitative testing, measuring not just the "quality" of very specific and sometimes cherry-picked data, but the overall perceived quality from both end users, and the general public. In my opinion with the advent of easy to access crowd testing tools such as Amazon turk, and the increased accessibility of end users through online surveys on sites such as Reddit, there is no excuse for lack of qualitative data in research. This is something that provides a very rich analysis of techniques from a different perspective to the hundreds of finely tuned and custom algorithms for ranking performance.

8.4 Conclusion

Finally, I would like to finish the thesis by concluding the main works; we have established some novel areas in the field, with several experiments having never been conducted before. Although these areas are small, we have shown the practical application of the methods outlined in real world environments, something that was extremely important to me at the beginning of the PhD.

Terrain generation has been presented in a more exciting way. A method of using preexisting terrain through high-resolution topographic data from NASA's Shuttle Radar Topography Mission (SRTM) to quickly generate procedural terrain with more variation. We extended this method to incorporate satellite image data as textures, adding to realism of terrains and allowing creators to add their own textures using the generated colours as a reference points.

Model generation was a highly competitive topic area, but we offered several improvements to the preexisting literature, one which incorporates colour data attached to

voxel models alongside extremely sparse training samples, with the use of Marching Cubes to convert the generated voxels back in to a low resolution textured mesh.

Furthermore we experiment with dense correspondence point cloud data using human models as training data, we outline a novel loss function which works to stabilise training and show the difference between two common GAN architectures proving 1D convolutions to perform poorly when compared to MLP.

We apply cutting edge research with GraphGANs to the problem, and show how they perform well given their lack of need for supervision, but lack the structural similarities that the MLP GAN offers. While it is unfair to make this comparison, as the supervised approaches require data to be in correspondence, the GraphGAN approach is still relevant since non-ordered data could theoretically be manually reordered.

Bibliography

[1] "practical-guide-to-gan-failure-modes." `https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/`. Accessed: 2021-05-10.

[2] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 341–346, 2001.

[3] A. Wulff-Jensen, N. N. Rant, T. N. Møller, and J. A. Billeskov, "Deep convolutional generative adversarial network for procedural 3d landscape generation based on dem," in *Interactivity, Game Creation, Design, Learning, and Innovation*, pp. 85–94, Springer, 2017.

[4] A. Borji, "Pros and cons of gan evaluation measures," *Computer Vision and Image Understanding*, vol. 179, pp. 41–65, 2019.

[5] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 9, no. 1, pp. 1–22, 2013.

[6] R. J. Spick, P. Cowling, and J. A. Walker, "Procedural generation using spatial gans for region-specific learning of elevation data," in *2019 IEEE Conference on Games (CoG)*, pp. 1–8, IEEE, 2019.

[7] R. J. Spick and j. walker, "Realistic and textured terrain generation using gans," in *European Conference on Visual Media Production*, pp. 1–10, 2019.

[8] R. Spick, S. Demediuk, and J. Alfred Walker, "Naive mesh-to-mesh coloured model generation using 3d gans," in *Proceedings of the Australasian Computer Science Week Multiconference*, pp. 1–6, 2020.

[9] R. J. Spick, T. Bradley, N. Williams, and J. A. Walker, "Human point cloud generation using deep learning," in *European Conference on Visual Media Production*.

[10] R. J. Schalkoff, *Artificial neural networks*, vol. 1. McGraw-Hill New York, 1997.

[11] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[14] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, contour and grouping in computer vision*, pp. 319–345, Springer, 1999.

[15] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[16] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 111–118, 2010.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, *et al.*, "Going deeper with convolutions," Cvpr, 2015.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[20] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, 2013.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[22] "Cs231n convolutional neural networks for visual recognition." `https://tex.stackexchange.com/questions/3587/how-can-i-use-bibtex-to-cite-a-web-page`. Accessed: 2021-05-04.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[24] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *arXiv preprint*, 2017.

[25] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *CoRR*, vol. abs/1606.03498, 2016.

[26] R. Salakhutdinov and H. Larochelle, "Efficient learning of deep boltzmann machines," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 693–700, 2010.

[27] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[28] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," *CoRR*, vol. abs/1610.07584, 2016.

[29] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1646–1654, 2016.

[30] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.

[31] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[32] S. Nowozin, B. Cseke, and R. Tomioka, "f-gan: Training generative neural samplers using variational divergence minimization," *arXiv preprint arXiv:1606.00709*, 2016.

[33] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[34] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410, 2019.

[35] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[37] Y. Jiang, S. Chang, and Z. Wang, "Transgan: Two transformers can make one strong gan," *arXiv preprint arXiv:2102.07074*, 2021.

[38] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[39] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[40] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional gans," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8798–8807, 2018.

[41] I. Durugkar, I. Gemp, and S. Mahadevan, "Generative multi-adversarial networks," *arXiv preprint arXiv:1611.01673*, 2016.

[42] D. J. Im, H. Ma, C. D. Kim, and G. Taylor, "Generative adversarial parallelization," *arXiv preprint arXiv:1612.04021*, 2016.

[43] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, pp. 214–223, PMLR, 2017.

[44] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann, "Stabilizing training of generative adversarial networks through regularization," *arXiv preprint arXiv:1705.09367*, 2017.

[45] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, "What is procedural content generation?: Mario on the borderline," in *Proceedings of the 2nd international workshop on procedural content generation in games*, p. 3, ACM, 2011.

[46] A. Amato, "Procedural content generation in the game industry," in *Game Dynamics*, pp. 15–25, Springer, 2017.

[47] R. Moss, "7 uses of procedural generation that all developers should study." `http://www.gamasutra.com/view/news/262869/7_uses_of_procedural_generation_that_all_developers_should_study.php`, 2016. [Online; accessed 22-01-2019].

[48] B. B. Mandelbrot and J. W. Van Ness, "Fractional brownian motions, fractional noises and applications," *SIAM review*, vol. 10, no. 4, pp. 422–437, 1968.

[49] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

[50] R. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, "Integrating procedural generation and manual editing of virtual worlds," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pp. 1–8, 2010.

[51] A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models," *Communications of the ACM*, vol. 25, no. 6, pp. 371–384, 1982.

[52] K. Perlin, "An image synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.

[53] D. P. Kroese and Z. I. Botev, "Spatial Process Generation," *arXiv e-prints*, p. arXiv:1308.0399, Aug. 2013.

[54] J. Gain, P. Marais, and W. Straßer, "Terrain sketching," in *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pp. 31–38, 2009.

[55] E. Catmull, "A subdivision algorithm for computer display of curved surfaces," tech. rep., UTAH UNIV SALT LAKE CITY SCHOOL OF COMPUTING, 1974.

[56] B. Julesz, "A theory of preattentive texture discrimination based on first-order statistics of textons," *Biological Cybernetics*, vol. 41, no. 2, pp. 131–138, 1981.

[57] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2, pp. 1033–1038, IEEE, 1999.

[58] J. Portilla and E. P. Simoncelli, "A parametric texture model based on joint statistics of complex wavelet coefficients," *International journal of computer vision*, vol. 40, no. 1, pp. 49–70, 2000.

[59] R. Paget and I. D. Longstaff, "Texture synthesis via a noncausal nonparametric multiscale markov random field," *IEEE transactions on image processing*, vol. 7, no. 6, pp. 925–931, 1998.

[60] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 327–340, 2001.

[61] I. E. Sutherland, "Sketchpad a man-machine graphical communication system," *Simulation*, vol. 2, no. 5, pp. R–3, 1964.

[62] W. E. Carlson, "An algorithm and data structure for 3d object synthesis using surface patch intersections," in *Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pp. 255–263, 1982.

[63] O. Van Kaick, K. Xu, H. Zhang, Y. Wang, S. Sun, A. Shamir, and D. Cohen-Or, "Co-hierarchical analysis of shape structures," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 69, 2013.

[64] L. De Floriani, "A pyramidal data structure for triangle-based surface description," *IEEE computer graphics and applications*, vol. 9, no. 2, pp. 67–78, 1989.

[65] H. Samet and R. E. Webber, "Hierarchical data structures and algorithms for computer graphics. i. fundamentals," *IEEE Computer Graphics and applications*, vol. 8, no. 3, pp. 48–68, 1988.

[66] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[67] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (pcgml)," *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 2018.

[68] S. Risi and J. Togelius, "Increasing generality in machine learning through procedural content generation," *Nature Machine Intelligence*, vol. 2, no. 8, pp. 428–436, 2020.

[69] C. Donahue, J. McAuley, and M. Puckette, "Adversarial audio synthesis," *arXiv preprint arXiv:1802.04208*, 2018.

[70] L. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," in *Advances in neural information processing systems*, pp. 262–270, 2015.

[71] J. Becker and D. Sandwell, "Srtm30_plus: Srtm30, coastal & ridge multibeam, estimated topography," *Electronic journal. URL: http://topex. ucsd. edu/WWW_html/srtm30_plus. html*, 2006.

[72] B. B. Mandelbrot and J. W. Van Ness, "Fractional brownian motions, fractional noises and applications," *SIAM review*, vol. 10, no. 4, pp. 422–437, 1968.

[73] T. Antoniades, "Iggi 2018 conference key note," Sep 2018.

[74] AboundDragons, "Perlin Noise, Procedural Content Generation, and Interesting Space." `https://heredragonsabound.blogspot.com/2019/02/perlin-noise-procedural-content.html`, 2019. [Online; accessed 6-February-2019].

[75] N. Jetchev, U. Bergmann, and R. Vollgraf, "Texture synthesis with spatial generative adversarial networks," *arXiv preprint arXiv:1611.08207*, 2016.

[76] R. Bamler *et al.*, "The srtm mission: A world-wide 30 m resolution dem from sar interferometry in 11 days," in *Photogrammetric week*, vol. 99, pp. 145–154, Berlin, Germany: Wichmann Verlag, 1999.

[77] L. Daoud, M. K. Latif, and N. Rafla, "Sift keypoint descriptor matching algorithm: A fully pipelined accelerator on fpga(abstract only)," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, (New York, NY, USA), pp. 294–294, ACM, 2018.

[78] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are gans created equal? a large-scale study," *arXiv preprint arXiv:1711.10337*, 2017.

[79] "the demographics of reddit: who uses the site?."

[80] A. Shaqiri, A. Brand, M. Roinishvili, M. Kunchulia, G. Sierro, J. Willemin, E. Chkonia, L. Iannantuoni, K. Pilz, C. Mohr, *et al.*, "Gender differences in visual perception," *Journal of Vision*, vol. 16, no. 12, pp. 207–207, 2016.

[81] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM siggraph computer graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[82] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006.

[83] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," in *Advances in neural information processing systems*, pp. 82–90, 2016.

[84] W. Wang, Q. Huang, S. You, C. Yang, and U. Neumann, "Shape inpainting using 3d generative adversarial network and recurrent convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2298–2306, 2017.

[85] C. Jiang, P. Marcus, *et al.*, "Hierarchical detail enhancing mesh-based shape generation with 3d generative adversarial network," *arXiv preprint arXiv:1709.07581*, 2017.

[86] I. Karth and A. M. Smith, "Wavefunctioncollapse is constraint solving in the wild," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, p. 68, ACM, 2017.

[87] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *ACM siggraph computer graphics*, vol. 21, pp. 163–169, ACM, 1987.

[88] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.

[89] F. Bogo, J. Romero, M. Loper, and M. J. Black, "FAUST: Dataset and evaluation for 3D mesh registration," June 2014.

[90] L. Ma, X. Jia, Q. Sun, B. Schiele, T. Tuytelaars, and L. Van Gool, "Pose guided person image generation," in *Advances in neural information processing systems*, pp. 406–416, 2017.

[91] A. Siarohin, E. Sangineto, S. Lathuiliere, and N. Sebe, "Deformable gans for pose-based human image generation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3408–3416, 2018.

[92] C. Yang, Z. Wang, X. Zhu, C. Huang, J. Shi, and D. Lin, "Pose guided human

video generation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 201–216, 2018.

[93] A. Kanazawa, M. J. Black, D. W. Jacobs, and J. Malik, "End-to-end recovery of human shape and pose," in *Computer Vision and Pattern Regognition (CVPR)*, 2018.

[94] Y. Zhang, M. Hassan, H. Neumann, M. J. Black, and S. Tang, "Generating 3d people in scenes without people," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6194–6204, 2020.

[95] X. Li, S. Liu, K. Kim, X. Wang, M.-H. Yang, and J. Kautz, "Putting humans in a scene: Learning affordance in 3d indoor environments," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12368–12376, 2019.

[96] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. J. Guibas, "Learning representations and generative models for 3d point clouds," *arXiv preprint arXiv:1707.02392*, 2017.

[97] D. A. Reynolds, "Gaussian mixture models.," *Encyclopedia of biometrics*, vol. 741, 2009.

[98] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[99] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 3, pp. 1–13, 2013.

[100] r. r. spick and j. walker, "Realistic and textured terrain generation using gans," in *European Conference on Visual Media Production*, CVMP '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[101] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[102] D. W. Shu, S. W. Park, and J. Kwon, "3d point cloud generative adversarial network based on tree structured graph convolutions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3859–3868, 2019.

[103] M. Lu, H. Zhao, A. Yao, F. Xu, Y. Chen, and L. Zhang, "Decoder network over lightweight reconstructed feature for fast semantic style transfer," in *Proceedings of the IEEE international conference on computer vision*, pp. 2469–2477, 2017.