

Deep Interactive Evolutionary 3D Modelling

M.Sc. in Software Development

Master's Thesis

Authors:

Adrian Westh (adwe@itu.dk)

Simon Krabbe Munck (skra@itu.dk)

Supervisor:

Sebastian Risi (sebr@itu.dk)

ITU

June 03, 2019

Deep Interactive Evolutionary 3D Modelling

A Study in Deep Interactive Evolution and Its Application
in the 3D Modelling Domain

Authors

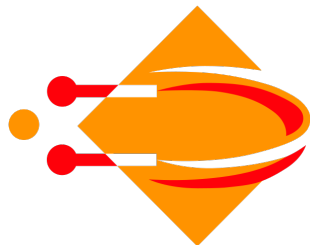
Adrian Westh (adwe@itu.dk)
Simon Krabbe Munck (skra@itu.dk)

Supervisor

Sebastian Risi (sebr@itu.dk)



M.Sc. in Software Development



IT-UNIVERSITETET | KØBENHAVN

Copenhagen, June 03, 2019

Acknowledgements

First, we would like to express our sincere gratitude to our advisor, Associate Professor Dr. Sebastian Risi, for the continuous support in regards to our master's thesis - for his patience, motivation and knowledge. His guidance greatly assisted this work.

Second, we must acknowledge the main location for both writing this master's thesis and developing the end product: Analog, the student orchestrated café at ITU. A café whose delicious coffee we would not have been without.

Lastly, three cheers to all the volunteers participating in the user testing and Sofia for proof reading without having any knowledge on the subject.

Declaration of Authorship

We, Adrian WESTH (Student No.: 12456) and Simon Krabbe MUNCK (Student No.: 12451), declare that this Master's Thesis titled "Deep Interactive Evolutionary 3D Modelling" and the work presented in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely our own work.
- We have acknowledged all main sources of help.
- Where the thesis is based on work done by ourselves jointly with others, we have made clear exactly what was done by others and what we have contributed ourselves.

Signed: Adrian Westh Simon Munck

Date: June 03, 2019

Abstract

This master's thesis introduces Deep Interactive Evolutionary 3D Modelling (DeepIE3D) as a novel alternative to traditional 3D modelling. DeepIE3D is an extended implementation of the Deep Interactive Evolution (DeepIE) approach, where the artifacts undergoing evolution are 3D voxel models. The DeepIE approach is a combination of generative adversarial networks (GAN) and interactive evolutionary computing. A GAN trained on a specific domain provides a generative model which enables a mapping of latent vectors to artifacts. The latent space used for artifact generation is well suited for evolutionary control. Exposing this evolutionary control to end users makes creating artifacts an intuitive process which mitigates user fatigue. The specific GAN framework used in DeepIE3D is PacWGAN-GP2. This framework ensures stable training and a generator able to produce a wide range of different artifacts. Alongside the DeepIE approach, this master's thesis introduces an alternative novel approach utilizing evolutionary principles. Furthermore, the benefits of adding Novelty Search (NS) to DeepIE are explored.

The main finding in this master's thesis is that the DeepIE approach successfully can be applied to 3D modelling. User testing reveals that end users are able to create subjectively satisfying 3D models: Both with the original approach and the novel approach introduced. NS in latent space does not yield any benefits, but NS in behavioral space does. However, NS in behavioral space is too computationally heavy.

Lastly, DeepIE3D is useful in practice: 3D models created via the publicly exposed web application can be downloaded as STL files and thus 3D printed.

Contents

1. Introduction	1
2. Background & Related Work	3
2.1. Generative Adversarial Networks	3
2.1.1. Distances	3
2.1.2. Original Generative Adversarial Networks	4
2.1.3. Deep Convolutional Generative Adversarial Networks	4
2.1.4. 3D Generative Adversarial Networks	5
2.1.5. Wasserstein Generative Adversarial Networks	6
2.1.6. PacGAN	7
2.2. Evolution	7
2.2.1. Deep Interactive Evolution	7
2.2.2. Novelty Search	9
3. Data	10
4. Approach & Implementation	12
4.1. Generative Adversarial Networks	12
4.1.1. Architecture	12
4.1.2. Training Ratios	13
4.1.3. Loss Functions	15
4.1.4. Modes and Overfitting	16
4.1.5. Optimizer	17
4.1.6. Hyperparameters	17
4.2. Evolution	19
5. Web Application	22
5.1. Technologies and Frameworks	22
5.2. Evolution	23
5.3. User Interface	24
6. User Testing	27
6.1. Motivation	27
6.2. Approach	27

7. Results	30
7.1. Evolution	30
7.1.1. Mutation	30
7.1.2. Crossover	30
7.1.3. Novelty Search	30
7.2. User Testing	35
7.3. 3D Printing	40
8. Analysis & Discussion	42
8.1. Generative Adversarial Networks	42
8.1.1. Training	42
8.1.2. 3D Model Distribution	45
8.2. Evolution	47
8.2.1. Mutation & Crossover	47
8.2.2. Novelty Search	48
8.3. User Testing	50
9. Future Work	55
10. Conclusion	57
Bibliography	58
A. Behavioral Novelty Search	61
B. Latent Vectors Used in Training Analysis	63
C. User Testing: Questionnaire	67
D. User Testing: Answers	81
E. Network Gradients	98

List of Figures

2.1.	An example of a generator in a DCGAN. This generator generates 2D image samples. It has no pooling or fully connected layers, but instead uses striding to upsample. Furthermore, it connects the input directly to the highest convolutional features (image from Radford et al. [1]) . . .	5
2.2.	Overview of the DeepIE approach in four steps constituting a cycle: First, a pre-trained generator is passed latent vectors to produce artifacts. Second, a user selects a range of artifacts based on subjective satisfaction. Third, the latent vectors associated with the selected artifacts are passed on. Fourth, the latent variables are evolved with a mutation rate determined by the user (image from Bontrager et al. [2]) . . .	9
3.1.	Three 3D models from different training sets. Each model is an example from a domain of training data	10
3.2.	A comparison of an original 3D model (a) and its downscaled 3D model (b)	11
4.1.	3D-GAN generator architecture. A latent vector \mathbf{z} is mapped to a 3D matrix representing a 3D object. The boxes between are volumetric fully convolutional layers (image from Wu et al. [3])	13
4.2.	Excerpt of the original DeepIE algorithm (image from Bontrager et al. [2])	19
5.1.	Overview of the DeepIE3D approach in four steps constituting a cycle: First, a pre-trained generator is passed latent vectors to produce artifacts. Second, a user either a) selects a range of artifacts based on subjective satisfaction, or b) customizes the next evolutionary generation. Third, the latent vectors associated with the selected artifacts are passed on. Fourth, the latent variables are evolved based on the evolutionary approach used with a mutation rate determined by the user. In the second step, choosing a) corresponds to Figure 2.2, while choosing b) corresponds to the novel custom approach	24
5.2.	The UI of DeepIE3D in <i>Normal mode</i> . The red circle indicates actions available	25
5.3.	The UI of DeepIE3D in <i>Advanced mode</i> . The red circles indicate differences from <i>Normal mode</i> and actions available	25

5.4.	Actions available on models in the UI of DeepIE3D. (a) are actions always available, (b) is the action only available in <i>Normal mode</i> , (c) are actions only available in <i>Advanced mode</i> . Descriptions taken directly from the web application	26
6.1.	Airplanes to recreate in the user testing	28
7.1.	Result of mutating the same 3D model nine times with a mutation rate of 1.0	31
7.2.	First result of crossovering two 3D models nine times	32
7.3.	Second result of crossovering two 3D models nine times	33
7.4.	Violin plots of similarity in 3D model distribution. The 3D models are generated in four ways: Randomly, novelty search in latent space and behavioral novelty search of 3D models with five or ten comparisons. Each data point is the mean of a generated 3D model’s similarities to the other generated 3D models	34
7.5.	3D models from user testing: <i>Normal mode</i>	36
7.6.	3D models from user testing: <i>Advanced mode</i>	37
7.7.	Histograms showing in which evolutionary iteration the subjectively most satisfying model occurred in each of the six tasks in the user testing	38
7.8.	Histograms showing subjective satisfaction levels of the best model found in each of the six tasks in the user testing. Rated on a scale from 1 to 10	39
7.9.	Histograms showing subjective satisfaction levels of the best of the last models found in each of the six tasks in the user testing. Rated on a scale from 1 to 10	39
7.10.	Histograms depicting the overall evaluation given after using the DeepIE3D web application throughout all six tasks. (a) shows the the volunteers’ perceptions of the user experience. (b) shows the volunteers’ perceptions of the ease of using the web application. Both are rated on a scale from 1 to 10	40
7.11.	Nine examples of 3D printed STL files exported from the DeepIE3D web application. The colors are arbitrary and has no meaning. (a) , (c) , (f) and (i) are printed on an Ultimaker 2 Extended; (b) , (g) and (h) are printed on an Ultimaker 2+; and (d) and (e) are printed on a Makerbot Replicator	41
8.1.	First example of 3D models generated during $epoch = 0$ through $epoch = 10,000$ when training the final PacWGAN-GP2	43
8.2.	Second example of 3D models generated during $epoch = 0$ through $epoch = 10,000$ when training the final PacWGAN-GP2	44

8.3.	Violinplot depicting the similarity between 512 randomly generated 3D models of airplanes and their best fit in the training set. Each generated 3D model is compared to all 3D models in the training set and the similarity is reported as a percentage of voxels in the same positions . . .	46
8.4.	Violinplots depicting comparisons of distributions between the training set and generated 3D models. (a) shows the comparison of internal mean similarity distribution between the training set and the generated 3D models. The mean similarity is calculated as a mean of the similarities in their sets. All 3D models in the training set compare voxel positions to all the other training set 3D models in order to produce a similarity percentage. The same is done for 512 3D models produced by the generator. (b) shows the comparison of voxel count distribution between the training set and the generated 3D models	47
8.5.	Lineage of chosen 3D models through ten evolutionary iterations when recreating Airplane 2 (Figure 6.1b) in <i>Normal mode</i> (Task 2)	49
8.6.	Line plot showing the performance of behavioral NS. The lowest amount of new models to generate is 14, yielding an extra wait time of 10 seconds to the web application user - or as high as 22 seconds when searching in 10 random 3D models which could be argued crucial to ensure novelty	50
E.1.	Gradients for the layers of the critic.	98
E.2.	Gradients for the layers of the generator.	98
E.3.	Gradient penalty	99

List of Tables

4.1. Hyperparameters used for GAN training and their impact	18
5.1. Available server endpoints in the backend of DeepIE3D (API)	23
8.1. Ten paired-sample t-tests. Iteration ratio is the ratio between the iteration in which the best 3D model occurred and total iterations used (10). A_i is the sample (a set of samples), μ_i is the sample mean and σ_i is the standard deviation of the sample	51

List of Abbreviations

Adam	Adaptive Moment Estimation.
API	Application Programming Interface.
BCE	Binary Cross-Entropy.
BN	Batch Normalization.
CNN	Convolutional Neural Network.
DCGAN	Deep Convolutional Generative Adversarial Networks.
DeepIE	Deep Interactive Evolution.
DeepIE3D	Deep Interactive Evolutionary 3D Modelling.
EM	Earth-Mover/ Wasserstein-1.
GAN	Generative Adversarial Networks.
IEC	Interactive Evolutionary Computation.
JS	Jensen-Shannon.
KL	Kullback-Leibler.
MLP	Multilayer Perceptron.
NA-IEC	Novelty-assisted Interactive Evolutionary Computation.
NS	Novelty Search.
PacWGAN-GP2	Packed Wasserstein Generative Adversarial Networks Gradient Penalty with doubled input.
ReLU	Rectified Linear Unit.
RMSProp	Root Mean Square Propagation.
WGAN	Wasserstein Generative Adversarial Networks.
WGAN-GP	Wasserstein Generative Adversarial Networks Gradient Penalty.

1. Introduction

3D modeling is an activity that requires a novice user to spend an excessive amount of time in order to learn the most basic principles. The learning curve of the process is steep which discourages many potential users. When a user becomes adequate at 3D modelling, the process can still be cumbersome and prolonged.

Deep Interactive Evolutionary 3D Modelling (DeepIE3D) is a novel framework introduced in this master's thesis (thesis). It enables novice end users to perform otherwise advanced and complex 3D modelling without prior knowledge of neither 3D modelling nor the underlying software. Furthermore, it allows easy exploration of 3D model domains (e.g. airplanes or chairs) making it a useful tool in concept development.

As stated in the name, DeepIE3D relies on the principles of interactive evolution. Interactive evolution is a niche in AI-assisted creation where the user serves as the fitness function in an evolutionary population. This concept implies that a user guides evolution through indirect interaction with artifacts undergoing creation. In the case of DeepIE3D, the artifacts created are 3D voxel models. During each evolutionary iteration, the system responds to artifact interaction by evolving user picked artifacts through mutation and/or crossover, creating a new evolutionary generation. Evolutionary iterations are repeated until an artifact becomes close to a subjective goal set by the end user or the end user becomes fatigued by the process. The latter termination of the process is known as *user fatigue*: A state in which the end user becomes fatigued and exhausted by a tedious evolutionary process.

The word "Deep" in DeepIE3D comes from the use of deep neural networks to enable a mapping between genotypes and phenotypes: From latent space (genotypes) to artifact space (phenotypes). The deep neural networks strive to provide meaningful artifact representations while permitting evolution in latent space. The importance of reasonable artifact representations in a specific artifact domain cannot be underestimated. It provides a space which is search-able by evolution, thus easing the process of the end users which are able to guide this evolution. To further mitigate user fatigue by a more varied exploration of the artifact space, DeepIE3D seeks to employ Novelty Search (NS). In DeepIE3D, NS is utilized as a method to expand the variation of the artifacts served to the end user.

The specific deep neural networks used in DeepIE3D are Generative Adversarial Networks (GAN). The GAN framework consists of two deep neural networks competing in an adversarial process: A generator and a discriminator. The discriminator distinguishes real artifacts from counterfeited artifacts generated by the generator. The objective of the generator is to deceive the discriminator. The generator of a pre-trained

GAN provides the above mentioned phenotype/genotype mapping in the pre-trained domain of artifacts.

DeepIE3D is an extended implementation of the Deep Interactive Evolution (DeepIE) approach [2]. The original approach consists of interactive evolution combined with deep neural networks used for artifact representation. In Bontrager et al. [2], the implemented example focuses on 2D images, where DeepIE3D is concerned with 3D voxel models. Throughout the thesis, DeepIE3D is compared to the implemented example from DeepIE in regards to approach, user testing and user fatigue. However, DeepIE3D is not merely an implementation adhering to the DeepIE approach: Alongside the DeepIE approach, DeepIE3D introduces an alternative novel approach utilizing evolutionary principles. Furthermore, the benefits of adding NS to DeepIE are explored in this thesis.

DeepIE3D is publicly available as a web application¹. The web application caters to the end users in regards to user experience and functionality: After a few easily performed evolutionary iterations, a subjectively satisfying 3D model can be downloaded and e.g. directly printed from a 3D printer². Along with the exposed web application, the source code is publicly available on GitHub³ including pre-trained models and instructions on how to use it.

The thesis is structured as follows:

Chapter 2 presents the background and related work followed by a description of the training data in Chapter 3. In Chapter 4 the overall approach and implementation of DeepIE3D is described, closely related to Chapter 5 describing the implementation of the web application. The motivation and approach of the user testing is presented in Chapter 6, while Chapter 7 displays the findings of this thesis including the ones from user testing - Chapter 8 analyzes and discusses these findings. Finally, Chapter 9 reflects on future work, and the whole thesis is concluded in Chapter 10.

¹DeepIE3D web application: <https://adrianwesth.dk>

²Short proof of concept video of DeepIE3D: <https://www.youtube.com/watch?v=qYwVxKVQZmE>

³DeepIE3D source code: <https://github.com/ukuleleplayer/DeepIE3D>

2. Background & Related Work

This chapter presents a detailed overview of the background and related work serving as a foundation of this thesis. It addresses Generative Adversarial Networks (GAN) and evolution.

2.1. Generative Adversarial Networks

This section concerns itself with GANs, focusing on the architectures and frameworks explored in this thesis.

GAN is a framework for estimating generative models. It originally consists of two deep neural network models competing in an adversarial process: A generator and a discriminator. The discriminator distinguishes real data samples from counterfeited samples generated by the generator. The objective of the generator is to deceive the discriminator [4].

2.1.1. Distances

When training GANs, a minimization of the difference between two probability distributions is pursued. Three metrics of probability distribution difference are introduced:

- The Kullback-Leibler (KL) divergence

$$D_{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x), \log \frac{P(x)}{Q(x)} \quad (2.1)$$

where D_{KL} is the divergence between probability distributions P and Q , and \mathcal{X} is a compact metric set [5]. It is used to determine the loss of information when Q is used to approximate P . KL divergence is asymmetric which entails it cannot be used as a distance metric.

- The Jensen-Shannon (JS) divergence

$$D_{JS}(P, Q) = D_{KL}(P||M) + D_{KL}(Q||M), \quad (2.2)$$

where $M = \frac{(P + Q)}{2}$ [5]. In contrast to KL divergence, JS divergence is symmetric and can be used as a distance metric.

- The Earth-Mover (EM)/Wasserstein-1 distance

$$W(P, Q) = \inf_{\gamma \in \Pi(P, Q)} \mathbb{E}_{(x, y) \sim \gamma} [\|x - y\|], \quad (2.3)$$

where $\Pi(P, Q)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively P and Q [5]. $W(P, Q)$ of two distributions P and Q indicates how much mass must be transported in order to transform Q to P ; it is described as the cost of transforming one to the other [5].

Different GAN frameworks use different metrics.

2.1.2. Original Generative Adversarial Networks

As stated above, the original GAN framework consists of a generator and a discriminator. The generator tries to learn the distribution p_q over the real data by creating a mapping from latent vectors $p_z(z)$ of random noise to the real data space [4]. The generator is denoted by $G(z; \theta_g)$, where G is a differentiable function represented by a neural network with the parameters θ_g . Similarly, the discriminator is denoted by $D(x; \theta_d)$ and often mirrors much of the generator as it takes inputs of the format produced by the generator. The discriminator outputs a single scalar denoting the probability of the input being real or fake [4]. The adversarial process can be staged as a two player minimax game as seen in Equation 2.4:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (2.4)$$

Equation 2.4 is the objective of the two networks: The discriminator strives to maximize the probability of predicting correct labels for both the real data points and fake generated samples. The generator strives to minimize the probability of the discriminator predicting the generated samples as fake.

The goal of the training is for G to converge on a probability distribution p_g serving as a good estimator of p_{data} [4]. This means that G should have global optimum where $p_g = p_{data}$. $p_g = p_{data}$ iff the JS divergence is zero and non-negative [4]. Thus, the objective of the generator can be described as minimizing the JS divergence. See Goodfellow et al. [4] for the theorem and proof.

2.1.3. Deep Convolutional Generative Adversarial Networks

In Goodfellow et al. [4] the GAN framework's correctness is shown with two multilayer perceptrons (MLP) as the discriminator and generator respectively. Radford et al. [1] introduce Deep Convolutional Generative Adversarial Networks (DCGAN), an alteration of the original GAN framework where the two MLPs are substituted with

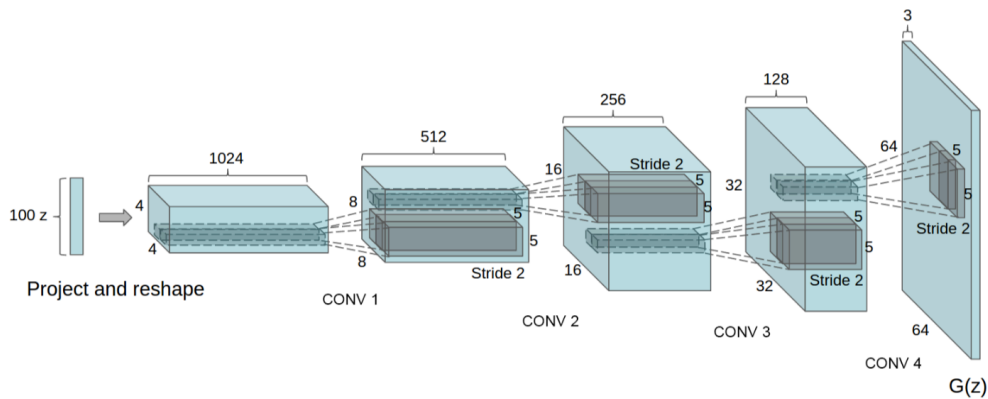


Figure 2.1: An example of a generator in a DCGAN. This generator generates 2D image samples. It has no pooling or fully connected layers, but instead uses striding to upsample. Furthermore, it connects the input directly to the highest convolutional features (image from Radford et al. [1])

convolutional neural networks (CNN). Past attempts to use two CNNs as the GAN architecture have been unsuccessful, as popular structural approaches such as max pooling layers and global average pooling were distorting the generated samples [1].

To remove the effects of pooling layers, DCGAN employs strided convolutions. This alternative is still able to make the networks learn their own spatial up-/downsampling. Global average pooling utilizes fully connected layers on top of convolutional features. This has been substituted with connecting the highest convolutional features directly to the input in the generator and to the output in the discriminator. The result is much faster convergence [1]. The training is stabilized by using Batch Normalization (BN) in all layers except the output layer of the generator and the input layer of the discriminator [1]. BN normalizes batches of input to each neuron, reducing the range the neuron values shift around [1, 6]. DCGAN utilizes the ReLU and Leaky ReLU activation functions. ReLU is applied in the generator which makes it saturate earlier. Leaky ReLU was found to work well in the discriminator [1].

An example of a generator in a DCGAN can be seen in Figure 2.1. The pictured generator adheres to the alternative structure described above. The corresponding discriminator is almost a mirrored version of this generator. The networks only differ in the activation functions as described above.

2.1.4. 3D Generative Adversarial Networks

The 3D-GAN framework was initially introduced as part of a system called 3D-VAE-GAN built to create 3D models from 2D images. 3D-GAN is a specific implementation of DCGAN, where traditional convolutional layers are replaced by volumetric convolutional layers capable of producing 3D models as output [3]. The discriminator and the generator mirror each other network-wise, while differing from one another in regards

to activation functions: ReLU in the generator and Leaky ReLU in the discriminator. No pooling layers or linear layers are added.

Wu et al. [3] employ an adaptive training strategy to keep the training of the generator and the discriminator in pace. This means that the discriminator only gets updated if its accuracy is below 80%.

2.1.5. Wasserstein Generative Adversarial Networks

The Wasserstein GAN (WGAN) introduces an alternate way of training GANs. As mentioned, original GAN training strives to learn a distribution estimating the data probability distribution. During training, the discriminator learns the distribution by discriminating real samples from fake samples, minimizing the JS divergence [4]. The JS divergence can, however, saturate locally when the discriminator becomes closer to optimality - this leads to vanishing gradient problems [5]: When the discriminator is near-optimal, the generator will experience its loss function to be incapable of providing context on how to train. Thus, a careful balance between training the generator and discriminator is needed.

WGAN proposes a different approach: It substitutes the discriminator for a *critic*. The critic does not discriminate between the inputs, instead it evaluates how far the distribution of the fake data is from the the real data distribution. To do this it utilizes the EM distance. Using the EM distance as the cost function leads to better generator training since the gradient of the critic is more well behaved [7]. When the critic is close to reaching optimality, the generator still receives a loss, describing how far it is from the desired distribution [5]. In order to use the EM distance as a cost function, the critic must lie within the space of 1-Lipschitz functions - this constraint is called the *1-Lipschitz constraint*. A *k*-Lipschitz constraint means that there is a constant *k* that limits the slope between any two points to have a gradient of at most *k* [8].

The original implementation of WGAN enforces the 1-Lipschitz constraint by weight clipping ensuring all weights are within a compact space [5]. This approach has a fault, as the parameter determining the clipping can affect the training in two ways: When too low, the network will take long to converge. When too large, the vanishing gradient problem will reappear [7].

WGAN Gradient Penalty (WGAN-GP), an improved version of WGAN, has been suggested in Gulrajani et al. [7]. Instead of weight-clipping, it introduces a gradient penalty in order to enforce the 1-Lipschitz constraint. A function *f* is 1-Lipschitz if its gradient norm is at most 1 in every point [7]. The gradient penalty constrains the critic by reviewing the gradient of straight lines between points of pairs sampled from the data distribution and the generated distribution. The optimal critic has straight lines from points in the data distribution to points in the generated distribution with gradient norm 1 [7]. The objective of WGAN-GP is a combination of the original critic

from WGAN and the gradient penalty:

$$L = \underbrace{\mathbb{E}_{\tilde{x} \sim \mathbb{P}_g} [D(\tilde{x})] - \mathbb{E}_{x \sim \mathbb{P}_r} [D(x)]}_{\text{Original critic loss}} + \lambda \underbrace{\mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} [(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{Gradient penalty}} \quad (2.5)$$

The part of Equation 2.5 described as the *Original critic loss* is the EM distance between two distributions \mathbb{P}_g and \mathbb{P}_r given by the Kantorovich-Rubinstein duality [9]. The part described as *Gradient penalty* is the penalty enforced when the gradient norms are different from 1. $\nabla_{\hat{x}} D(\hat{x})$ is the output of $D(\hat{x})$ differentiated with regards to \hat{x} , where \hat{x} is from the distribution sampled uniformly along straight lines between pairs of points sampled from the data distribution \mathbb{P}_r and the generator distribution \mathbb{P}_g [7].

2.1.6. PacGAN

The PacGAN framework introduces an approach to mitigate *mode collapse* when training GANs. Mode collapse is a common phenomenon when training GANs. A mode collapse is present, when a generative model fails to produce diverse artifacts: It only produces artifacts from a few of the expected modes [10]. A mode is best explained by example: A generative model trying to produce hand-written digits from e.g. the MNIST¹ dataset can cover 10 modes as there are 10 different digits.

The main idea behind PacGAN is to pass m "packed" or concatenated samples to the discriminator. These samples are from the same class: Either both real or both fake. Lin et al. [10] states that this approach allows the discriminator to do binary hypothesis testing which penalizes mode collapse.

To create such a packed discriminator, the size of its input layer must be increased by a factor of m . The packed discriminator is able to observe m samples, which intuitively helps it detect mode collapse as lack of diversity becomes easier to detect [10].

2.2. Evolution

This section concerns itself with the two evolutionary concepts used in this thesis: Deep Interactive Evolution (DeepIE) and Novelty Search (NS).

2.2.1. Deep Interactive Evolution

DeepIE is an approach combining deep learning with interactive evolutionary computing (IEC). This section only briefly touches the deep learning part, as the deep learning utilized in this thesis is covered in Section 2.1. However, it should be noted that the key insight in the main paper about DeepIE, Bontrager et al. [2], is concerned with deep learning: "... a GAN trained on a specific target domain can act as a compact and

¹The MNIST Database: <http://yann.lecun.com/exdb/mnist/>

robust *genotype-to-phenotype mapping ...*". Since the GAN mapping is well behaved, the genotypes are suited for evolutionary control.

IEC is an AI-assisted creative process. Instead of directly manipulating artifacts (e.g. pictures or 3D models) during creation, the users communicate their desired changes indirectly via feedback and suggestions to the software. When used in conjunction with GANs, the evolution is performed on the latent vectors, the genotypes, and the user feedback becomes the fitness function in regards to the generated artifacts, the phenotypes [2].

As stated by Bontrager et al. [2], this interactive AI-assisted creation has interesting applications:

- It empowers the novice user to create artifacts of higher quality than otherwise feasible - e.g. 3D modelling as this is not a trivial task and normally requires technical skills to perform
- It allows the user to create and explore artifacts with traits they would not normally strive towards

Traditionally, IEC has been used in open-ended domains with undefined objectives [2]. Finding the desired artifacts may likely be a cumbersome process taking a large number of iterations. This leads to the issue known as *user fatigue*, since users tend to tire after evaluating relatively few generations.

DeepIE tries to mitigate user fatigue by finding better artifact representations and restricting the class of artifacts to a certain domain. Both mitigations are achieved by the previously mentioned GAN.

DeepIE relies on two evolutionary principles: Mutation and crossover. Mutation is the process of randomly altering an artifact, crossover is the process of joining two artifacts by mating, thus creating an offspring with features from both artifacts.

The DeepIE approach is as follows: The generator from a pre-trained GAN takes randomly generated latent vectors as input. A number of artifacts (images in Bontrager et al. [2]) are produced which are shown to the user. The user selects the desired artifacts and the latent vectors of these undergo evolution: Some selected latent vectors are crossovered and new random latent vectors are generated. The selected latent vectors and the crossovered latent vectors are mutated and lastly passed to the GAN. Then the process loops. Figure 2.2 shows an illustrated overview of this process.

The evaluation method used to measure user fatigue and overall performance in DeepIE is user testing. Users are given tasks to reconstruct images of shoes and faces and the results produced are acceptable. The overall feedback from the users is a mix of amusement and frustration [2].

IEC is not the only way to perform interactive 3D modelling: Liu et al. [11] employ a different approach which utilizes the 3DGAN for artifact representation. Instead of evolving the 3D models, the end users make rough edits to them. After roughly editing a 3D model, the underlying system performs a *SNAP* command transforming

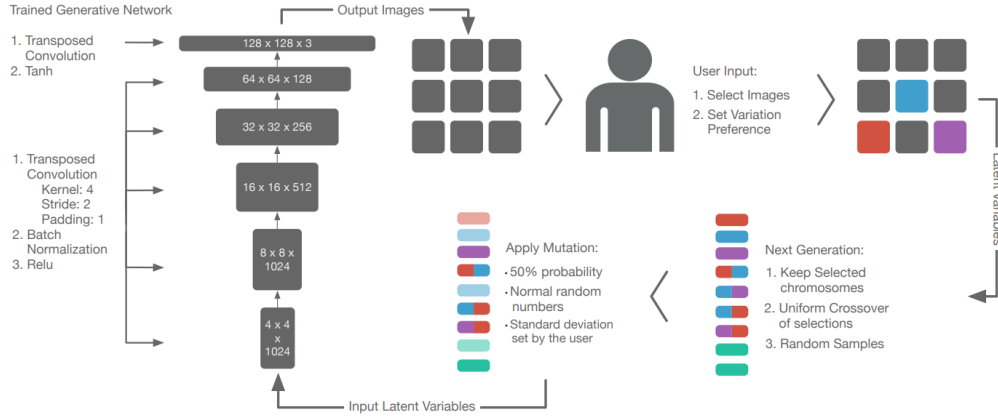


Figure 2.2: Overview of the DeepIE approach in four steps constituting a cycle: First, a pre-trained generator is passed latent vectors to produce artifacts. Second, a user selects a range of artifacts based on subjective satisfaction. Third, the latent vectors associated with the selected artifacts are passed on. Fourth, the latent variables are evolved with a mutation rate determined by the user (image from Bontrager et al. [2])

the rough 3D model into a more realistic one [11]. The results are promising, but the overall user experience, including user fatigue, is not assessed.

2.2.2. Novelty Search

NS is an algorithm used in evolutionary computing which favors novel behaviors in contrast to objective performance. Instead of rewarding candidates achieving high fitness, candidates exhibiting novelty are rewarded.

According to G. Woolley and Stanley [12], NS combined with IEC creates a complementary effect that offsets each other’s limitations. NS thrives with proper guidance and IEC requires ways to mitigate user fatigue. The guidance is provided by the user interaction from IEC and the user fatigue mitigation is provided by an easier explorable search space by NS. The approach combining NS and IEC is called novelty-assisted interactive evolutionary computation (NA-IEC). In this thesis, the initial version of NA-IEC is slightly modified since fitness evaluation is left *entirely* to the users, and since novelty is measured in the genotypes (latent space), not the phenotypes (behavioral space). Please see Section 4.2 for implementation details and Section 8.2 for a discussion of the results concerning NS.

3. Data

The training data used in DeepIE3D is 3D models in `binvox` format. A file of this format has a header describing dimensions, translations and scale of the subsequent voxel data.

A voxel represents a single data point in a regularly spaced 3D grid¹. The data points of the 3D models used for training contain only one value: 1 or 0.

All 3D models have been downloaded from `ShapeNet`²; more specifically the `ShapeNetCore v2` data set. During training, three model domains were used: Chairs, airplanes and cars - see Figure 3.1 for examples of the different model domains. To view raw `binvox` files, the `viewvox`³ program has been utilized. To read and write `binvox` files, the Python module `binvox_rw`⁴ has been utilized.

The 3D models from `ShapeNet` were not initially suited for training, leading to three preprocessing steps done in the below written order:

- **Downscaling**

The original 3D models from `ShapeNet` are of dimension $128 \times 128 \times 128$. Working with data of that size is computationally heavy and infeasible to render in the final application. Scaling the models down to $64 \times 64 \times 64$ reduces the data size eight times while still preserving similarity. An example of similarity preserv-

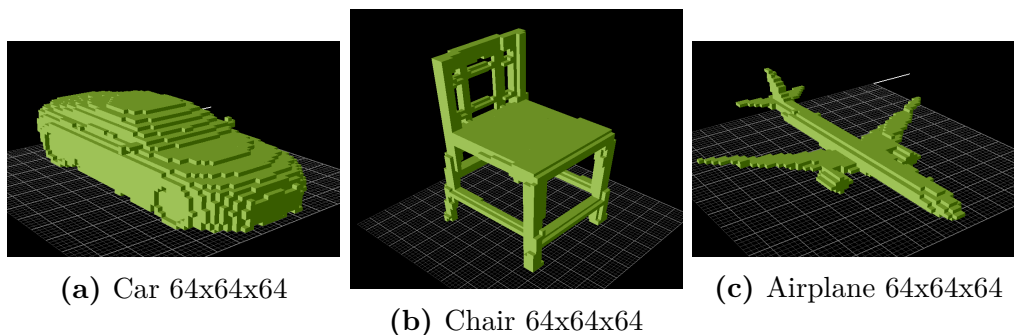


Figure 3.1: Three 3D models from different training sets. Each model is an example from a domain of training data

¹Definition of Voxel: <https://en.wikipedia.org/wiki/Voxel>

²ShapeNet: <https://www.shapenet.org/>

³Program for viewing `binvox` files: <http://www.patrickmin.com/viewvox/>

⁴Python module for reading and writing `binvox` files: <https://github.com/dimatura/binvox-rw-py>

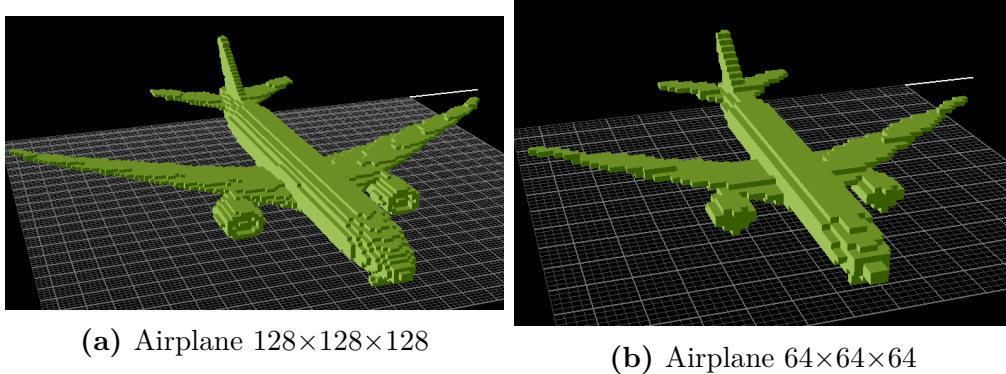


Figure 3.2: A comparison of an original 3D model (a) and its downsampled 3D model (b)

ing downscaling can be seen in Figure 3.2. All 3D models in DeepIE3D are of dimension $64 \times 64 \times 64$.

- **Centering**

The original 3D models from ShapeNet are not all situated in the same position. This unnecessarily increases difficulty of the training objective. Thus, all models have been centered in both the x - and the y -axis.

- **Removing outliers**

After downscaling and centering the 3D models, it was necessary to manually select usable models. Downscaling may result in incomplete models, e.g. a chair missing a leg. Furthermore, the original 3D models from ShapeNet have outliers in the sense that some models do not fit its model domain. In the airplane domain, some of the models have too much inspiration from "Star Wars", while other models have excessively varying sizes.

During training, data sets as small as 32 3D models are used for rapid testing, while the final model is trained on 512 hand-picked 3D models from the airplane domain.

When comparing similarity between two 3D models, the voxel similarity is used: Out of the total amount of voxels with respect to one of the models, how many voxels do they share. In other words: $\frac{\text{shared_voxels}}{\text{total_voxels}_m}$ where m is the model in focus.

4. Approach & Implementation

This chapter describes the approach and implementation details of the system implemented in this thesis: *DeepIE3D*.

The main idea behind DeepIE3D is to enable interactive evolution in the 3D modelling domain. This is achieved by: First, train a GAN on a specific 3D model domain (e.g. airplanes or chairs). Second, expose the generator from the GAN through an API, enabling 3D model construction directly from latent vectors. Third, expose evolutionary concepts through an API, enabling evolution of latent vectors. Fourth and last, encapsulate the above in a user friendly application, enabling intuitive manipulation of 3D models through a user interface. The source code of DeepIE3D, including two pretrained models and several preprocessed data sets, is publicly available at: <https://github.com/ukuleleplayer/DeepIE3D>.

The following sections will in detail elaborate on how this is achieved and Chapter 5 is dedicated to the web application.

4.1. Generative Adversarial Networks

This section describes the overall considerations in regards to the GAN implementations in this thesis. This mainly concerns the original Deep Convolutional GAN (DCGAN) and the Wasserstein GAN Gradient Penalty (WGAN-GP).

4.1.1. Architecture

The initial implementation was heavily influenced by the 3D-GAN implementation by the user *rimchang* on GitHub [13]. It is an adaption of the 3D-GAN framework described in Wu et al. [3] implemented in the Python library PyTorch. The 3D-GAN framework consists of a generator and a discriminator like the original GAN. The generator maps a latent vector to a 3D tensor representing a 3D model. The discriminator maps a cube corresponding to the output of the generator to a likelihood of it being real. The 3D-GAN framework uses neural network models with architectures of volumetric fully convolutional layers [3]. The generator consists of layers of kernel sizes $4 \times 4 \times 4$ and strides 2, utilizes batch normalization and lastly uses ReLU as activation function in each layer. The last layer ends with an elementwise sigmoid activation. The structure of the generator can be seen in Figure 4.1. The discriminator mirrors the generator, except ReLU is replaced by Leaky ReLU. The generator uses transposed

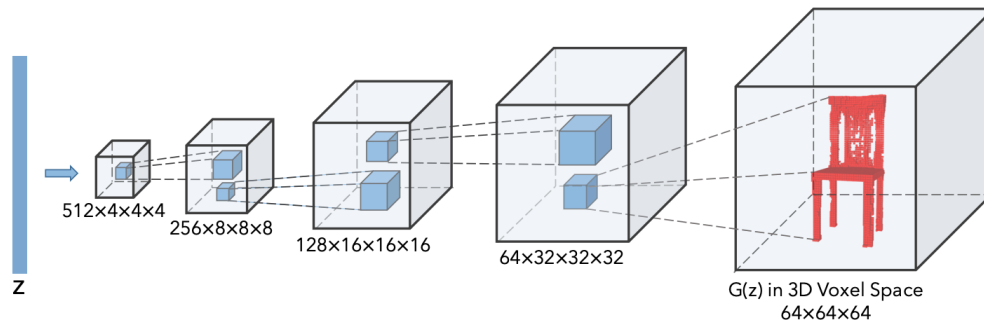


Figure 4.1: 3D-GAN generator architecture. A latent vector z is mapped to a 3D matrix representing a 3D object. The boxes between are volumetric fully convolutional layers (image from Wu et al. [3])

convolutional layers for upsampling, and the discriminator uses normal convolutional layers for downsampling.

The architecture of the models in this thesis is very close to the ones described in Wu et al. [3]. A distinction is made through the choice of using WGAN-GP as opposed to a traditional DCGAN. Batch Normalization (BN) can no longer be applied in the discriminator/critic (WGAN-GP’s counterpart to a discriminator), as the gradient penalty would otherwise be faulty: BN maps a batch of inputs to a batch of outputs through internal normalization, but gradient penalty depends on gradients from independent single inputs [7]. A layer-normalization scheme was attempted as it is suggested as a substitute for BN in Gulrajani et al. [7]. It was rejected as it results in the training never converging.

In order to avoid overfitting and mode collapse, a dropout strategy was enforced between convolutional layers. `PyTorch` has a `Dropout3D` layer which can be used without modifications in 3D convolutions; it randomly drops out entire channels [14]. This was, however, discarded as both models trained with or without dropout layer produced very similar results with a variety of modes.

The WGAN-GP architecture was extended to a PacWGAN-GP2 architecture. PacGAN changes the format of the input given to the discriminator/critic to fit a collection of generated and real data objects. PacWGAN-GP2 doubles the input ($m = 2$) which makes it take two data points (either both fake or both real) as input.

4.1.2. Training Ratios

When training a GAN, a ratio between training the discriminator and the generator has to be found. A ratio resulting in training the discriminator insufficiently will make the generator constantly able to fool the discriminator, even though it does not generate samples close to a desired artifact. Vice versa, a ratio resulting in insufficient generator training will result in a discriminator too good at discerning what samples are real. Thus, the generator will struggle to learn the data distribution.

Two factors can affect the training ratio: The loss function and the nature of the training set. The loss function, along with the learning rate, decides how much the network is back-propagated. If the loss function for either network is better at making its network converge, it will skew the training. Likewise, the nature of the training set can impact the GAN training: In the case of this thesis, distinguishing between $128 \times 128 \times 128$, $64 \times 64 \times 64$ and $32 \times 32 \times 32$ 3D models has an impact. A ratio favoring increasingly more discriminator training is needed as the voxel count grows.

The two following sections describe the final training ratios used in this thesis.

Original DCGAN

In the `PyTorch` implementation of 3DGAN, the ratio is determined by the discriminator’s prediction accuracy. As default, the discriminator is only allowed to train if its accuracy is less than 80% [13]. Thus, the generator is potentially allowed to train multiple times per discriminator training. This implicates that the discriminator may perform too well when the ratio is one to one.

Soumith Chintala (creator of `PyTorch` and co-author on WGAN and DCGAN papers [1, 5]) has created a `GitHub` repository providing *hacks* for training GANs. In section 14 in the repository’s `README.md`, he states that training the discriminator more often than the generator helps in some cases. The `PyTorch` implementation of 3DGAN references to Chintala’s `GitHub` repository [15]. The above training ratios are conflicting.

In this thesis, the initial implementation with the original DCGAN architecture from the 3DGAN framework was fine-tuned on small data sets of $32 \times 32 \times 32$ 3D models. When a higher resolution was needed, the training data was switched to $64 \times 64 \times 64$ 3D models.

The initial training ratio was the same as the default from the 3DGAN `PyTorch` implementation. This ratio is heavily skewed as the generator will always fool the discriminator and will not generate useful models within 1000 epochs.

A one to one training ratio was adopted but showed the same skewing as the prior ratio. This is consistent with the findings in the original implementation.

This, along with the *hacks* provided in Chintala [15], decided that a ratio favoring discriminator training is needed. The first approach was similar to training a WGAN (see below): Train the discriminator n times every epoch. This yields better results, but the discriminator can train multiple times even though it is already sufficiently good at discriminating. Furthermore, sometimes even n training cycles for the discriminator is insufficient for catching up with the generator.

This led to a ratio based on multiple discriminator trainings with a conditional on the discriminator’s accuracy: Every epoch the discriminator is trained n times as long as its accuracy is below a threshold (default: 80%). This allows the discriminator to catch up with the generator but not train unnecessarily.

With this ratio, the discriminator can become overly effective, leading to a generator unable to learn at the same pace. Thus, an upper bound on the discriminator’s accuracy

was instantiated: The discriminator is only allowed to train when its accuracy is below 100%.

The final training ratio of the DCGAN in this thesis is as such: Train the discriminator until its accuracy is above a specified threshold. If the accuracy is above another threshold (default: 100%), do not train the discriminator. Train the generator once.

WGAN-GP

The models with WGAN-GP and PacWGAN-GP2 architectures are always trained on $64 \times 64 \times 64$ 3D models.

An essential concept of WGAN-GP training is for the critic to be trained to optimality [5]. However, as the critic is not a discriminator, it does not have an accuracy on which to base the training ratio. This thesis adapts the training ratio from the WGAN-GP pseudocode to achieve critic optimality [7]. This approach simply trains the critic five times each epoch while training the generator once.

4.1.3. Loss Functions

The loss function in a neural network with backpropagation determines how much weights in a network update. When working with GANs, it is hard to decide which loss function results in the best training.

The below discussed loss functions are chosen based on published GAN papers.

Original DCGAN

The PyTorch implementation of 3DGAN utilizes Binary Cross-Entropy (BCE) as its loss function [13]. BCE, also called Sigmoid Cross-Entropy, is a Sigmoid activation followed by the distance from a classification. The discriminator concerns itself with two data classes: *real* = 1 and *fake* = 0. BCE calculates loss as a distance of the predicted class to the actual class. This is done independently for both classes, aggregating all losses of individual samples. The original objective function of the discriminator, described in equation 2.4, is BCE [16]. Thus, the objective of the discriminator is to assign low probabilities to fake samples and high probabilities to real samples. Furthermore, BCE can be used as the loss function of the generator: The generator's objective is to strive towards having assigned high probability to its generated samples by minimizing BCE loss between discriminator predictions and the real class.

As the implementation of the system in this thesis is based on the 3DGAN PyTorch implementation [13], the initial loss function was BCE. As stated in Subsection 4.1.2, training with the initial settings is ineffective as it does not converge within 1000 epochs. This is however not due to the loss function, but the training ratio: Deciding on a loss function is heavily reliant on finding the right training ratio. When a sufficiently good training ratio was found, BCE was deemed adequate.

One addition to BCE was tried: A heuristic which steers the generation of 3D models towards the number of voxels in the training set. This is done by adding a penalty when the number of voxels is outside the bounds of the highest and lowest amount of voxels found in the training set. The heuristic was not found effective. It slows the convergence of the training and does not result in better generated models.

WGAN-GP

WGAN, which uses the EM distance as its loss function, has been shown to yield better results than the original GAN [5]. It addresses the vanishing gradient problem, which the original GAN often experiences, by having a critic instead of a discriminator.

WGANs have been shown to create stability in training, ensuring a higher chance of convergence [5]. Gulrajani et al. [7] suggest an improved training process of WGAN: WGAN-GP. Instead of weight-clipping, it uses a penalty on the gradient of the critic with respect to its input in order to enforce the 1-Lipschitz constraint. This is an addition to the original critic loss function.

In this thesis, WGAN-GP is the chosen WGAN implementation. The loss function implemented is the objective seen in Equation 2.5. The original critic loss is implemented as the mean critic predictions of the fake samples subtracted by the mean of critic predictions of the real samples. The gradient penalty is implemented in the following way: Initially, real and fake samples are interpolated. The interpolated samples are fed to the critic and the resulting values' gradients, with regards to the interpolated samples, are found. The penalty is the vector norms of these gradients subtracted by 1, squared and finally averaged. The loss function of the generator is the negated mean of critic predictions of its generated samples.

4.1.4. Modes and Overfitting

In this thesis, mode collapse was often encountered. Thus, actions were taken in order to prevent it.

When the original DCGAN implementation became capable of producing models of a satisfactory quality, it became evident that it is victim to mode collapse: When trained extensively, it almost always produces the same model with very few alterations.

The choice to implement WGAN-GP relies on the premise that it prevents mode collapse [7]. Mode collapse in the original GAN is due to a generator collapsing towards a delta function at the data point(s), where the discriminator predicts the highest probability of being real [5, 17]. WGAN-GPs train the discriminator/critic to optimality to avoid this.

The WGAN-GP framework does remove mode collapse, as its generator produces a variety of different modes. The number of modes produced by WGAN-GP in this thesis is, however, still not the full set of modes.

The PacGAN framework is designed to mitigate mode collapse and is compatible

with various GAN architectures, including WGAN-GP. PacWGAN has been shown to discover more modes than WGAN-GP [10]. Thus, PacWGAN-GP2 has been implemented to increase the number of modes produced by the generator. Please note that when Lin et al. [10] mention a PacWGAN, the WGAN is actually a WGAN-GP, which explains why it is called a PacWGAN-GP2 in this thesis.

As stated earlier, a dropout-strategy was enforced in order to avoid overfitting. This, however, yields no significant results. Analysis of whether the final generative model is overfitting can be found in Subsection 8.1.2.

4.1.5. Optimizer

The original WGAN discourages the use of the popular Adaptive Moment Estimation (Adam) optimizer as training becomes unstable [5]. Instead, the Root Mean Squared Propagation (RMSprop) optimizer is encouraged. In Gulrajani et al. [7] an experiment is conducted with the Adam and RMSProp optimizer respectively. WGAN-GP achieves better performance when using the Adam optimizer compared to the RMSProp optimizer [7].

The final models with WGAN-GP architecture produced in this thesis use the Adam optimizer due to this.

The Adam optimizer, unlike normal stochastic gradient descent, maintains individual learning rates of all the network parameters. It determines the learning rates by utilizing two moments: The average mean of the gradients and the average uncentered variance of the gradients [18].

The Adam optimizer needs two coefficients: β_1 and β_2 . The coefficients are used for computing parameter learning rates for the two moments of the gradient. β_1 is the decay rate of the first moment, and β_2 is the decay rate of the second moment.

Kingma and Ba [18] state that $\beta_1 = 0.9$, $\beta_2 = 0.999$ are good default values for machine learning tasks. This is also evident as machine learning libraries such as `PyTorch` use these values as default [19]. When training the 3DGAN, Wu et al. [3] set $\beta_1 = 0.5$ as Radford et al. [1] find it stabilizes training. When training WGAN-GP, Gulrajani et al. [7] set $\beta_1 = 0.0$ and $\beta_2 = 0.9$.

The values of the betas in this thesis are the same as the suggested values found in the papers where the architectures are introduced.

4.1.6. Hyperparameters

The hyperparameters associated with the training of the models impact the training process. They need to be tuned in order to achieve optimal training behavior. This is not trivial and needs to be explored through trial and error.

In this thesis, the hyperparameters and their impact can be seen in Table 4.1.

	Description	Space	Impact
z distribution	The space of the values contained in the latent vector. Can have a uniform distribution or a normal (gaussian) distribution	<i>uni</i> : Uniform distribution [0.0;1.0] <i>norm033</i> : Normal distribution centered in 0.0 with <i>stdev</i> = 0.33 <i>norm1</i> : Normal distribution centered in 0.0 with <i>stdev</i> = 1.0 Default: <i>norm1</i>	Using a uniform distribution ensures all variables to be within a specified range. Using a normal distribution prevents diverging from a model's prior distribution and produces sharper artifacts [20]
Labels	The labels for the real and the generated fake data. They specify the value which the discriminator should strive to achieve. Only used when GAN type is <i>DCGAN</i>	<i>noisy</i> : Fake labels sampled from uniform distribution [0.0;0.15], real labels sampled from uniform distribution [0.85;1.0] <i>hard</i> : Fake labels = 0.0, real labels = 1.0 <i>d_hard</i> : Fake labels = 0.0, real labels = 0.9 Default: <i>noisy</i>	The purpose of <i>noisy</i> labels is to avoid overconfidence which can cause overfitting. To avoid overfitting with hard labels, <i>d_hard</i> can be employed. As the real labels are 0.9, overconfidence will be penalized
D Iterations	The number of times the discriminator is trained for each training of the generator	[0; ∞] Default: 5	In DCGAN, this can make the discriminator and generator progress similarly (often combined with <i>D Threshold</i>). In WGAN-GP, the number of iterations must be high enough to train the discriminator to optimality
D Threshold	The lower limit of accuracy of the discriminator. The discriminator is trained until the accuracy is above the threshold. Only used when GAN type is <i>DCGAN</i>	[0.0;1.0] Default: 0.8	This can make the discriminator and generator progress similarly (often combined with <i>D Iterations</i>)
Adam optimizer betas	Coefficients used for computing parameter learning rates by two moments of the gradients: The mean and the uncentered variance. The coefficients are the decay rates of the moments [19]	β_1 : [0.0;1.0] β_2 : [0.0;1.0] Default: [$\beta_1 = 0.0$; $\beta_2 = 0.9$]	When GAN type is <i>DCGAN</i> , $\beta_1 = 0.5$ and $\beta_2 = 0.999$ ensures stable training [4]. Gulrajani et al. [7] suggest using $\beta_1 = 0.0$ and $\beta_2 = 0.9$ when GAN type is <i>WGAN-GP</i>
GAN type	The architecture of the generative adversarial nets	<i>DCGAN</i> <i>WGAN-GP</i> <i>PacDCGAN2</i> <i>PacWGAN-GP2</i> Default: <i>PacWGAN-GP2</i>	<i>DCGAN</i> trains the fastest. In some cases, it experiences mode collapse. <i>WGAN-GP</i> addresses the vanishing gradient problem and should prevent mode collapse. <i>PacWGAN-GP2</i> takes two 3D models as input in the discriminator. This results in more modes
Training data size	The size of the training data set	[0; ∞] Default: 512	When the data size increases, the chance of a higher variety increases. This results in more modes
Batch size	The size of the batches of training data, which is used for one round of training both networks	[0;[Training data]] Default: 64	Often limited by (GPU) memory. A small batch size leads to slower epochs, as each batch has back propagation steps. However, the frequent training with a small batch size can cause the networks to converge at an earlier epoch
Learning rates	The learning rates for the networks	[0.0;1.0] Default: [$D_{lr} = 0.0001$, $G_{lr} = 0.0001$]	How much the gradient weighs in the backpropagation process
WGAN gradient penalty lambda	The lambda is the degree of the gradient penalty. The gradient penalty is multiplied with the lambda	[0; ∞] Default: 10	As the gradient penalty is part of the loss function of the discriminator, it affects how much the network is backpropagated

Table 4.1: Hyperparameters used for GAN training and their impact

4.2. Evolution

The DeepIE algorithm from Bontrager et al. [2] is implemented as true to the original as possible: First, latent vectors of selected 3D models are passed to the algorithm. Second, a share of the latent vectors are crossovered, and additional random samples are added. Third, the crossovered and selected latent vectors are mutated and returned along with the random latent vectors. Fourth and last, new 3D models are generated based on the previous result.

The excerpt implemented of the DeepIE algorithm can be seen in Figure 4.2. The two changes made are small but significant: First, the latent vectors used in DeepIE3D are associated with user selected 3D models, not user selected images. Second, the amount of latent variables in the latent vectors are 200, not 20. The amount of latent variables were chosen in accordance to the 3D-GAN implementation [3].

To further experiment with interactive evolution, a novel custom evolutionary approach is implemented alongside the original algorithm: Instead of choosing the 3D models to ones liking and running them through an algorithm, one decides entirely how the next generation should be evolved. In every evolutionary iteration, nine 3D models are generated. The custom approach offers four different actions to customize the next evolutionary generation: Create an entirely new 3D model, keep a 3D model as it is, mutate a 3D model, or crossover a 3D model with another 3D model. The underlying mutation and crossover implementations are identical to Bontrager et al. [2]. This custom approach has several implications:

- It is possible to use a single 3D model as baseline for the whole next evolutionary generation (e.g. mutating it nine times)
- It is possible to fully avoid evolving a 3D model, but still keep it in the next evolutionary generation as is
- It is possible to attempt to pass on the traits of a single model to all others by crossovering it with all other 3D models

```

8:  indices ← interface.getSelectedImages()
9:  selection ←  $Z_{indices}$ 
10:  $s^2$  ← interface.getMutationParameter()
11:  $\Delta$  ←  $m - \text{length}(\textit{selection})$ 
12:  $x$  ←  $\max(0, \Delta - \textit{foreign})$ 
13: cross ←  $x$  by  $n$  matrix where  $\textit{cross}_i \sim \textit{mutate}(\textit{uniform}(\textit{selection}), s^2)$ 
14:  $x$  ←  $\max(\textit{foreign}, \Delta)$ 
15: new ←  $x$  by  $n$  matrix where  $\textit{new}_{i,j} \sim \mathcal{N}(\mu, \sigma^2)$ 
16: selection ← apply mutate with  $s^2$  to each selection $i$ 
17:  $Z$  ← selection + cross + new

```

Figure 4.2: Excerpt of the original DeepIE algorithm (image from Bontrager et al. [2])

- It is possible to dictate the entire composition of the next evolutionary generation

In addition to the two evolutionary approaches, Novelty Search (NS) is implemented. The novelty metric used in this thesis is the novelty of the latent vectors. The degree of novelty corresponds to the distance in euclidian space. Two reasons led to this novelty metric, the first being most important: First, it is not viable performance-wise to compare 3D models (see Subsection 8.2.2 for a thorough analysis). Second, it was deemed capable of characterizing the space of unique behaviors in a way that is meaningful to 3D model generation - however, as described later in Subsection 8.2.2, this turns out to be wrong.

This latent space version of NS can easily be added to both the original DeepIE algorithm and the novel custom approach. When NS is performed, all new 3D models in the next evolutionary generation are generated using as different latent vectors as possible. If the next evolutionary generation consists of nine new 3D models, an initial random latent vector serves as a starting point. Please refer to the pseudocode in Algorithm 1 for implementation details - the actual implementation is done in Python.

Algorithm 1: Novelty Search in Latent SpaceDefaults: $m \leftarrow 9, n \leftarrow 200, min \leftarrow -4.9, max \leftarrow 4.9$

```

1  $Z \leftarrow$  m by n matrix where  $Z_{i,j} \sim \mathcal{N}(\mu, \sigma_2)$ 
2  $indices \leftarrow$  GetChosenModels() // UI is responsible for this
3 NoveltySearch( $Z_{indices}$ )
4 Function NoveltySearch( $Z$ ):
5   if  $Z$  length == 0 then
6     |  $noise \leftarrow$  vector of length n where  $noise_i \sim \mathcal{N}(\mu, \sigma_2)$ 
7     |  $Z \leftarrow Z + noise$ 
8   end
9   for  $i \leftarrow 0$  to  $m - Z$  length do
10    |  $Z \leftarrow Z +$  CreateNovel( $Z$ )
11  end
12  return  $Z$ 
13 Function CreateNovel( $Z$ ):
14   $novel \leftarrow$  EMPTY LIST
15  for  $i \leftarrow 0$  to  $n$  do
16    |  $numbers \leftarrow$  EMPTY LIST
17    | foreach vector  $v$  of the matrix  $Z$  do
18      |  $numbers.add(v_i)$ 
19    | end
20    |  $novel.add(NovelNumber(numbers))$ 
21  end
22  return  $novel$ 
23 Function NovelNumber( $numbers$ ):
24   $numbers.add(min)$ 
25   $numbers.add(max)$ 
26   $numbers.sort()$ 
27   $maxDist \leftarrow -1.0$ 
28   $number \leftarrow 0.0$ 
29  for  $i \leftarrow 1$  to  $numbers$  length do
30    |  $tempDist \leftarrow |numbers_i - numbers_{i-1}|$ 
31    | if  $tempDist > maxDist$  then
32      |  $maxDist \leftarrow tempDist$ 
33      |  $number \leftarrow numbers_{i-1}$ 
34    | end
35  end
36  return  $number + \frac{maxDist}{2.0}$ 

```

5. Web Application

This chapter concerns itself with the web application exposed to the end users of DeepIE3D. The web application is publicly available at this site: <https://adrianwesth.dk>.

5.1. Technologies and Frameworks

The web application consists of a front-end (the client) and a back-end (the server). The back-end is responsible for generating 3D models in addition to performing the evolution described in Subsection 4.2. As the original training of the GAN is done in Python using PyTorch, the whole back-end is written in Python.

The front-end is responsible for rendering the 3D models, storing the latent vectors and providing a seamless, easy-to-use user interface (UI). The whole front-end is written in Dart; a typed programming language by Google which transpiles to JavaScript.

The back-end relies heavily on a framework called Flask¹ which takes care of routing, sending data and generally serving the backend - it supports multithreading and is able to be hosted in a production environment.

For rendering using WebGL, the front-end utilizes the JavaScript library `three.js`². This adds a much-needed layer of abstraction, making it possible to render 3D models without writing a shader from scratch. Even with the abstraction, having to process as vast data volumes as the $64 \times 64 \times 64$ 3D voxel models still poses some challenges: It is too computationally heavy to render each voxel as a box. Instead, it is necessary to create vertices and faces, constructing the entire 3D model using only 2D triangles; one of the primitives in WebGL.

As mentioned, the web application is built around the classical client/server architecture. The list of available server endpoints and their functionality can be seen in Table 5.1. The server is hosted on Google App Engine³ which automatically scales the underlying hardware based on demand. It is easy to deploy and the scaling ensures low cost while being ready to perform in the event of unexpected interest in the web application.

The client is hosted on one of the authors' own domain. Only requests from this specific domain receive valid responses from the server.

¹The Flask framework: <http://flask.pocoo.org/>

²The JavaScript library `three.js`: <https://threejs.org/>

³Google App Engine: <https://cloud.google.com/appengine/>

Endpoint	HTTP Method	Request Data Type	Response Data Type	Description
/initialize_single/	POST	$model_type \leftarrow \text{string}$	$z \leftarrow \text{list of float}$ $coords \leftarrow \text{list of list of int}$ $camera \leftarrow \text{list of int}$	Initializes a single 3D model along with its latent vector and camera placement information
/generate_single/	POST	$model_type \leftarrow \text{string}$ $z \leftarrow \text{list of float}$	$coords \leftarrow \text{list of list of int}$ $camera \leftarrow \text{list of int}$	Generates a single 3D model based on a latent vector. Includes camera placement information
/evolve/	POST	$specifications \leftarrow \text{string}$ $mutation \leftarrow \text{float}$ $novelty \leftarrow \text{bool}$ $\sum_{n=0}^8 z_n \leftarrow \text{list of float}$	$\sum_{n=0}^8 z_n \leftarrow \text{list of float}$	Evolves all given latent vectors based on the specifications, mutation rate and novelty
/download_binvox/	POST	$model_type \leftarrow \text{string}$ $z \leftarrow \text{list of float}$	octet stream	Generates a 3D model from the latent vector and downloads it

Table 5.1: Available server endpoints in the backend of DeepIE3D (API)

5.2. Evolution

Two versions of evolution are implemented in the web application: The original algorithm from Bontrager et al. [2] and the novel custom evolutionary approach. The background behind the original algorithm is described in Section 2.2 while the implementation details of both algorithms are described in Section 4.2. The end user has easy access to both versions in the UI. All technical details are hidden from the user while providing powerful tools to perform evolution at the same time.

The front-end is only responsible for storing the current evolutionary generation as latent vectors, the back-end performs the evolutionary computation. Figure 5.1 shows an in-depth illustration of the evolutionary flows.

In the web application, the original DeepIE algorithm is called *Normal mode* and the custom evolutionary approach introduced in Section 4.2 is called *Advanced mode*. In both *Normal mode* and *Advanced mode* the user has the option to enable Novelty Search (NS). This forces the distribution of latent vectors to widen, thus opening up for more different models. The NS feature is disabled in the web application due to findings in this thesis; see Subsection 7.1.3 and Subsection 8.2.2 for further details.

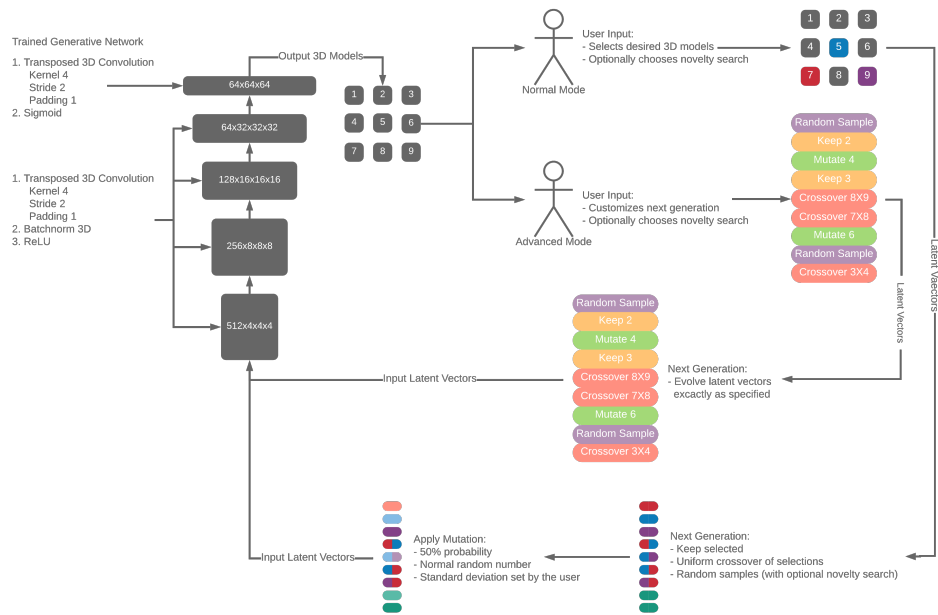


Figure 5.1: Overview of the DeepIE3D approach in four steps constituting a cycle: First, a pre-trained generator is passed latent vectors to produce artifacts. Second, a user either **a)** selects a range of artifacts based on subjective satisfaction, or **b)** customizes the next evolutionary generation. Third, the latent vectors associated with the selected artifacts are passed on. Fourth, the latent variables are evolved based on the evolutionary approach used with a mutation rate determined by the user. In the second step, choosing **a)** corresponds to Figure 2.2, while choosing **b)** corresponds to the novel custom approach

5.3. User Interface

The UI of the web application is tailored to the end users and requires very little knowledge about evolution, 3D modelling or artificial intelligence in general. As the web application is publicly available, the UI will only be described briefly. The overall UI can be seen in Figure 5.2 and Figure 5.3.

The main element on the web application is a grid of canvases displaying the generated 3D models. To allow easy navigation, each canvas itself exposes several actions. The general actions can be seen in Figure 5.4a, the actions available only in *Normal mode* can be seen in Figure 5.4b and actions available only in *Advanced mode* can be seen in Figure 5.4c. When in *Advanced mode*, a list shows how the next evolutionary generation will be evolved. Furthermore, an icon layer on each canvas shows how that 3D model contributes to the next evolutionary iteration.

The web application offers a thorough user guide, which introduces the end user to all necessary aspects of using DeepIE3D and suggests further reading. The user guide and all other text in the web application are kept easily readable and user-oriented.

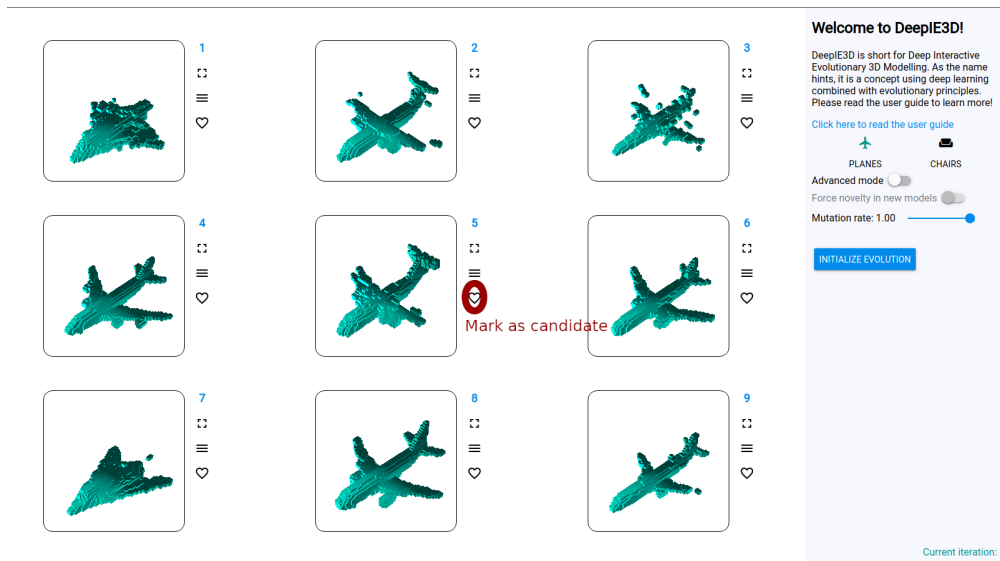


Figure 5.2: The UI of DeepIE3D in *Normal mode*. The red circle indicates actions available

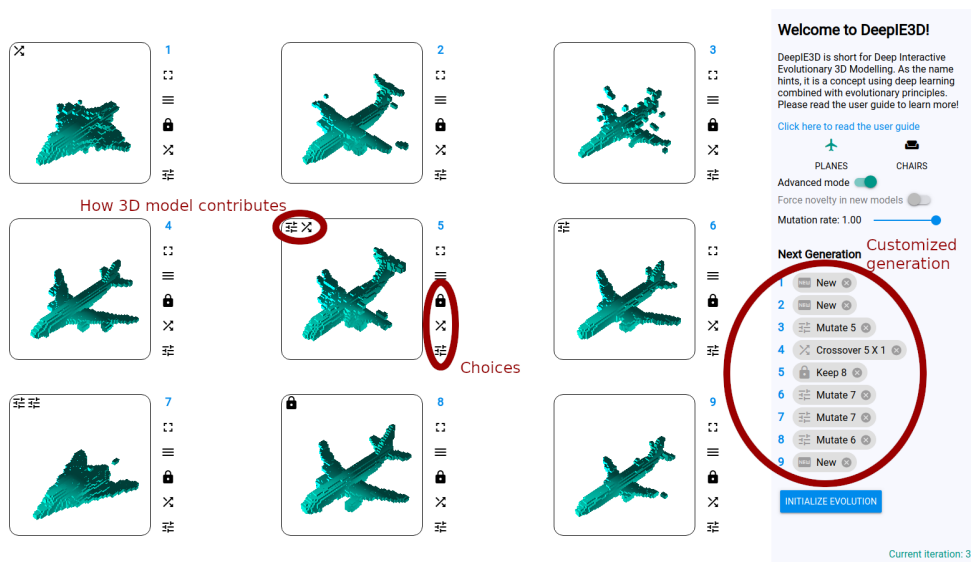


Figure 5.3: The UI of DeepIE3D in *Advanced mode*. The red circles indicate differences from *Normal mode* and actions available










-  View a large version of a 3D model
 -  Save a 3D model in various formats (.binvox, .stl, .png)
 - Force novelty in 3D models: Widens the distribution of newly created 3D models, resulting in more different 3D models (experimental feature, turned off for now)
 -  Adjust the general mutation rate, controlling the magnitude of mutations
 -  Select airplanes as 3D model type (will reload site)
 -  Select chairs as 3D model type (will reload site)
- (a)
-  Mark a 3D model as a candidate to the DeepIE algorithm
- (b)
-  Mark a 3D model as being kept in its original form in the next evolutionary iteration
 -  Mark a 3D model as being combined with another to-be-chosen 3D model in the next evolutionary iteration
 -  Mark a 3D model as being mutated in the next evolutionary iteration
- (c)

Figure 5.4: Actions available on models in the UI of DeepIE3D. (a) are actions always available, (b) is the action only available in *Normal mode*, (c) are actions only available in *Advanced mode*. Descriptions taken directly from the web application

6. User Testing

This chapter describes the motivation and approach of the user testing of DeepIE3D. The results can be found in Section 7.2 and are further discussed in Section 8.3.

6.1. Motivation

The main motivation to perform user testing of DeepIE3D is to obtain user feedback on how it performs in various tasks. As the end product primarily is aimed towards novices in the 3D modelling domain user testing is a critical part of evaluating its success. Success is defined as an outcome, where DeepIE3D delivers subjectively satisfactory results after process of 3D modelling without user fatigue.

Concretely, the goal of the user testing is to evaluate the following:

- How *Advanced mode* performs compared to *Normal mode*
- How DeepIE3D performs in recreation tasks
- How DeepIE3D performs in creative tasks
- How DeepIE3D operates in regards to user fatigue

6.2. Approach

The user testing approach is inspired by Bontrager et al. [2], but diverges in many ways.

The seventeen users participating (volunteers) were given a questionnaire. The exact questionnaire can be found in Appendix C, and the answers obtained from it can be found in Appendix D.

Guided by the questionnaire, the volunteers were given an introduction to DeepIE3D and were told to read the user guide in the web application. The goal of the user guide is to briefly introduce the purpose of DeepIE3D and give a thorough introduction on how to use it.

Concretely, the volunteers were assigned six tasks (four recreation, two creative):

Task 1 Using *Normal mode*, recreate the airplane shown in Figure 6.1a

Task 2 Using *Normal mode*, recreate the airplane shown in Figure 6.1b



(a) Airplane 1



(b) Airplane 2

Figure 6.1: Airplanes to recreate in the user testing

Task 3 Using *Advanced mode*, recreate the airplane shown in Figure 6.1a

Task 4 Using *Advanced mode*, recreate the airplane shown in Figure 6.1b

Task 5 Using *Normal mode*, create an imaginative airplane to ones liking

Task 6 Using *Advanced mode*, create an imaginative airplane to ones liking

All above tasks were done using ten evolutionary iterations. In all of the tasks, the volunteers were asked to do the following:

1. In the first evolutionary iteration, download the initial subjectively most satisfying 3D model. This is regarded as the initial current best
 2. During iterations, if an evolved 3D model is subjectively more satisfying than the current best, download it as the new current best. Keep track of the evolutionary iteration
 3. In the tenth iteration, download the subjectively most satisfying 3D model regardless of whether it is subjectively more satisfying than the overall current best
 4. Report on:
 - a) Subjectively most satisfying 3D model
 - b) Subjectively most satisfying 3D model from the tenth evolutionary iteration. This can be the same as the one from a)
 - c) In which iteration the subjectively most satisfying 3D model occurred
 - d) How satisfying the subjectively most satisfying 3D model turned out
-

- e) How satisfying the subjectively most satisfying 3D model in the tenth evolutionary iteration turned out. This can be the same as the one from d)

After the six tasks, the volunteers were asked to report on:

- The overall user experience: A scale from 1 to 10, where 1 is exhaustive and 10 is fun
- The ease of using the web application: A scale from 1 to 10, where 1 is hard and 10 is easy
- Their approach in recreation and creative tasks respectively
- The intuitiveness of *Normal mode* and *Advanced mode* respectively
- The user guide and the overall feature set of DeepIE3D

The majority of the answers (¹⁰/₁₇) from the questionnaire comes from one sitting. It was deemed necessary to be in the same location as the volunteers going through the questionnaire as questions may arise. Furthermore, overseeing the user testing may lead to valuable observations.

7. Results

This chapter describes the findings in this thesis including, but not limited to, results from the user testing. For an overall proof of concept, please see this video: <https://www.youtube.com/watch?v=qYwVxKVQZmE>. The video concisely shows the process from interactive evolutionary 3D modelling to a final 3D printed product.

7.1. Evolution

This section displays the results when utilizing mutation and crossover in DeepIE3D. Furthermore, it touches on the performance of novelty search.

7.1.1. Mutation

Applying mutation in DeepIE3D yields well-behaved results. In Figure 7.1 an initial model can be seen mutated nine times. The mutations undoubtedly resemble the initial model while at the same time having slightly different traits; the only outlier is Figure 7.1j.

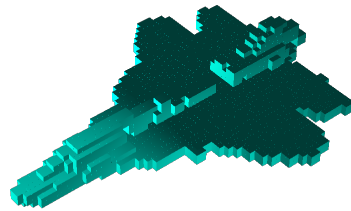
7.1.2. Crossover

Applying crossover in DeepIE3D yields well-behaved results. In Figure 7.2 and Figure 7.3 two 3D models can be seen crossovered nine times. Crossover has the *potential* of mixing traits. Figure 7.2g and Figure 7.3c both show this potential fulfilled: They undeniably expose traits from both initial models. While the other 3D models are well-behaved, they are, however, not as easily inferred as crossovers.

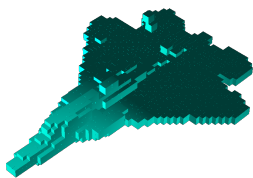
7.1.3. Novelty Search

Three different types of Novelty Search (NS) have been tested in DeepIE3D: Novelty in latent space (latent vectors) and novelty in behavioral space (generated 3D models), where the behavioral NS searches for novelty in five and ten randomly generated 3D models respectively.

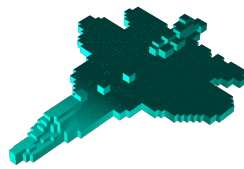
In Figure 7.4 the resulting 3D model distribution similarity of the different types of NS can be seen along with the standard distribution of random samples. The random sampling is the only method used in the DeepIE3D web application for reasons



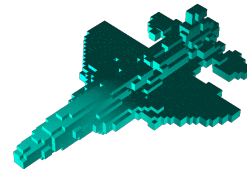
(a) Initial model



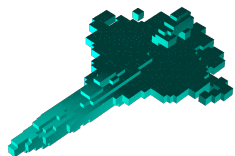
(b)



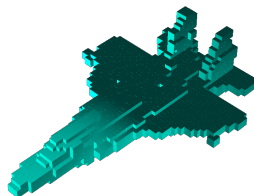
(c)



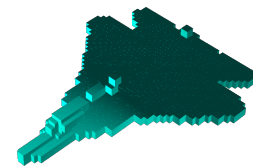
(d)



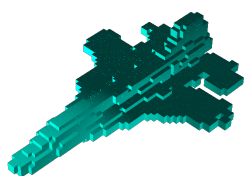
(e)



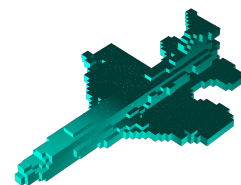
(f)



(g)



(h)



(i)



(j)

Figure 7.1: Result of mutating the same 3D model nine times with a mutation rate of 1.0

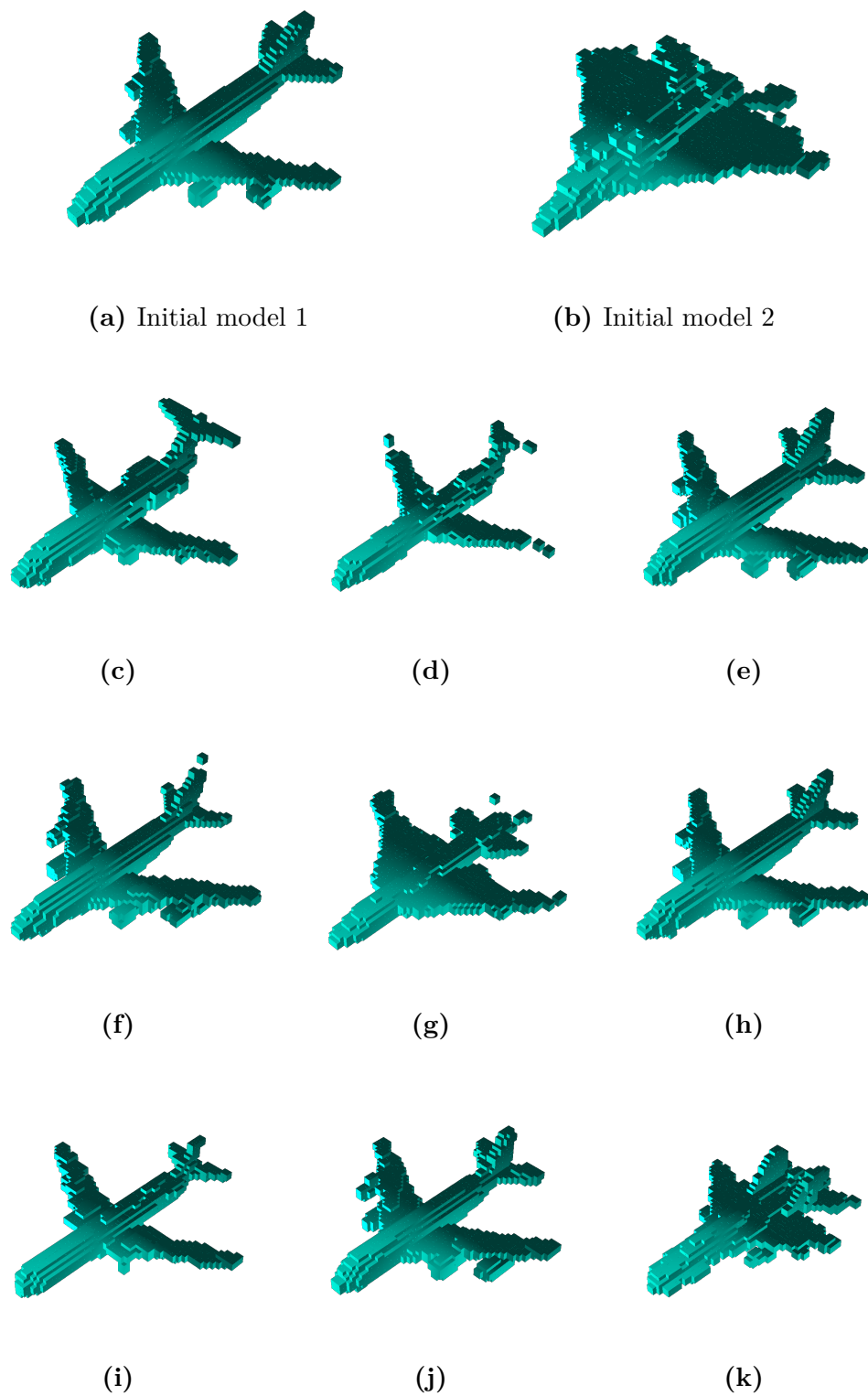
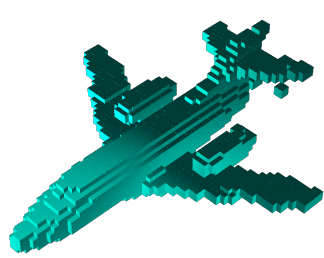
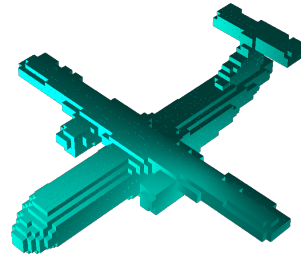


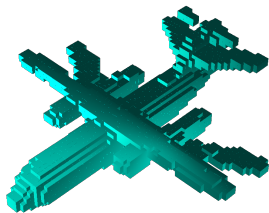
Figure 7.2: First result of crossovering two 3D models nine times



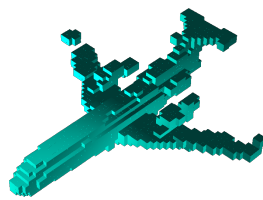
(a) Initial model 1



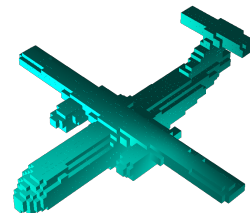
(b) Initial model 2



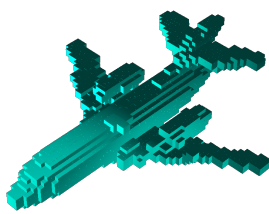
(c)



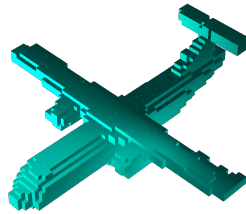
(d)



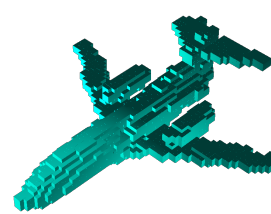
(e)



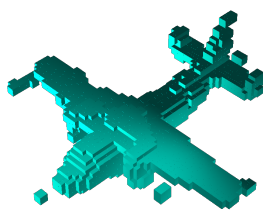
(f)



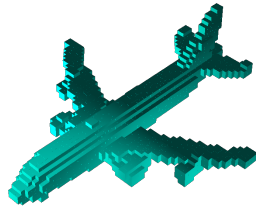
(g)



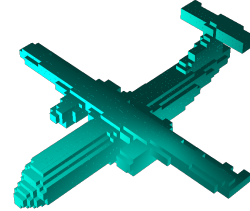
(h)



(i)



(j)



(k)

Figure 7.3: Second result of crossovering two 3D models nine times

described in Subsection 8.2.2 and the results seen in Figure 8.2.2. Figure 7.4 depicts the distribution similarity mean shown as violin plots¹.

The similarity mean distribution is found by:

1. Generate nine 3D models using one of the above mentioned methods
2. For each model, calculate the voxel similarity as described in Chapter 3 to all other 3D models. Average these similarities
3. Repeat 1 and 2 ten times

The above calculation yields ninety data points for each violin plot. It is clear that behavioral NS, which searches in ten randomly generated 3D models, outperforms all other versions. Additionally, it is clear that using novelty search in latent space does not improve novelty compared to simply generating random 3D models.

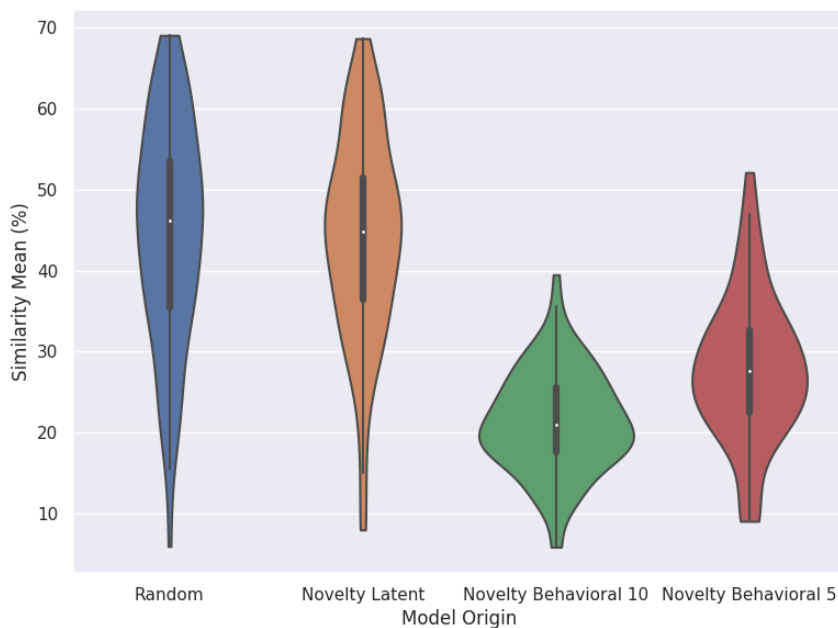


Figure 7.4: Violin plots of similarity in 3D model distribution. The 3D models are generated in four ways: Randomly, novelty search in latent space and behavioral novelty search of 3D models with five or ten comparisons. Each data point is the mean of a generated 3D model’s similarities to the other generated 3D models

¹A violin plot is a method of plotting numeric data. It is similar to a box plot, with the addition of a rotated kernel density plot on each side

7.2. User Testing

This section presents the results from the user testing. Seventeen volunteers participated in the user testing consisting of the six tasks seen in Section 6.2. After each task, the volunteers answered follow-up questions and uploaded models created in the tasks. After the six tasks, a row of questions about the overall experience concerning the web application were answered. The seventeen volunteers come from different backgrounds and have little to no knowledge about 3D modelling, evolution or computer science in general.

The questionnaire used in the user testing can be found in Appendix C along with the answers in Appendix D.

The goal of the tasks is to use ten evolutionary iterations to recreate or create an airplane. The volunteers uploaded images of their overall most satisfactory airplane and the most satisfactory airplane from the final (tenth) evolutionary generation. Figure 7.5 and Figure 7.6 show the images uploaded by the volunteers. The figures consist of three columns of pairwise comparisons between the final and best 3D model depicting Airplane 1 (Figure 6.1a), Airplane 2 (Figure 6.1b) and an imaginative airplane respectively. The rows in the figures are the results of a specific volunteer. Thus, volunteers are specified by the same row number in both figures. Figure 7.5 shows the results from **Task 1**, **Task 2** and **Task 5**, which are performed in *Normal mode*. Figure 7.6 shows the results from **Task 3**, **Task 4** and **Task 6**, which are performed in *Advanced mode*. The figures are mainly included to showcase the 3D model results from the user testing, but are further touched upon in Section 8.3.

After completing a task, the volunteers were asked to report on the following three considerations:

1. In which evolutionary iteration the subjectively most satisfying 3D model occurred
2. How satisfying the subjectively most satisfying 3D model turned out
3. How satisfying the subjectively most satisfying 3D model in the tenth evolutionary iteration turned out

Figure 7.7, Figure 7.8 and Figure 7.9 correspond to the results from consideration 1, 2 and 3 respectively. The subfigures correspond to the tasks from Section 6.2 in the following way:

(a) is **Task 1** (b) is **Task 2** (c) is **Task 5**
(d) is **Task 3** (e) is **Task 4** (f) is **Task 6**

The above ordering of tasks and subfigures is chosen in order to have all the tasks regarding *Normal mode* and *Advanced mode* aligned horizontally. Furthermore, it allows a vertical comparison of equivalent tasks performed in different modes.

Figure 7.7 shows the distribution of the evolutionary iteration producing the best 3D model in all six tasks.

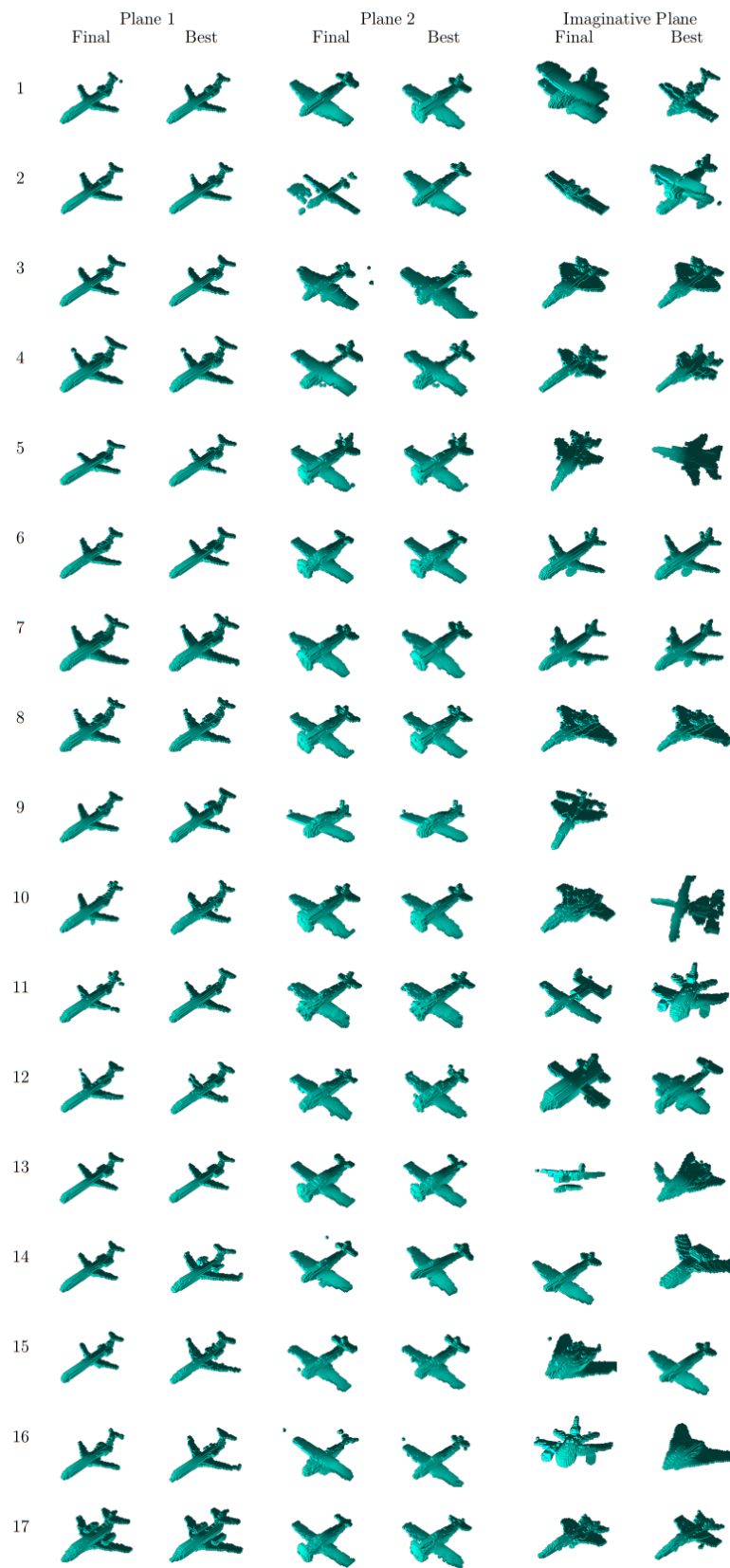
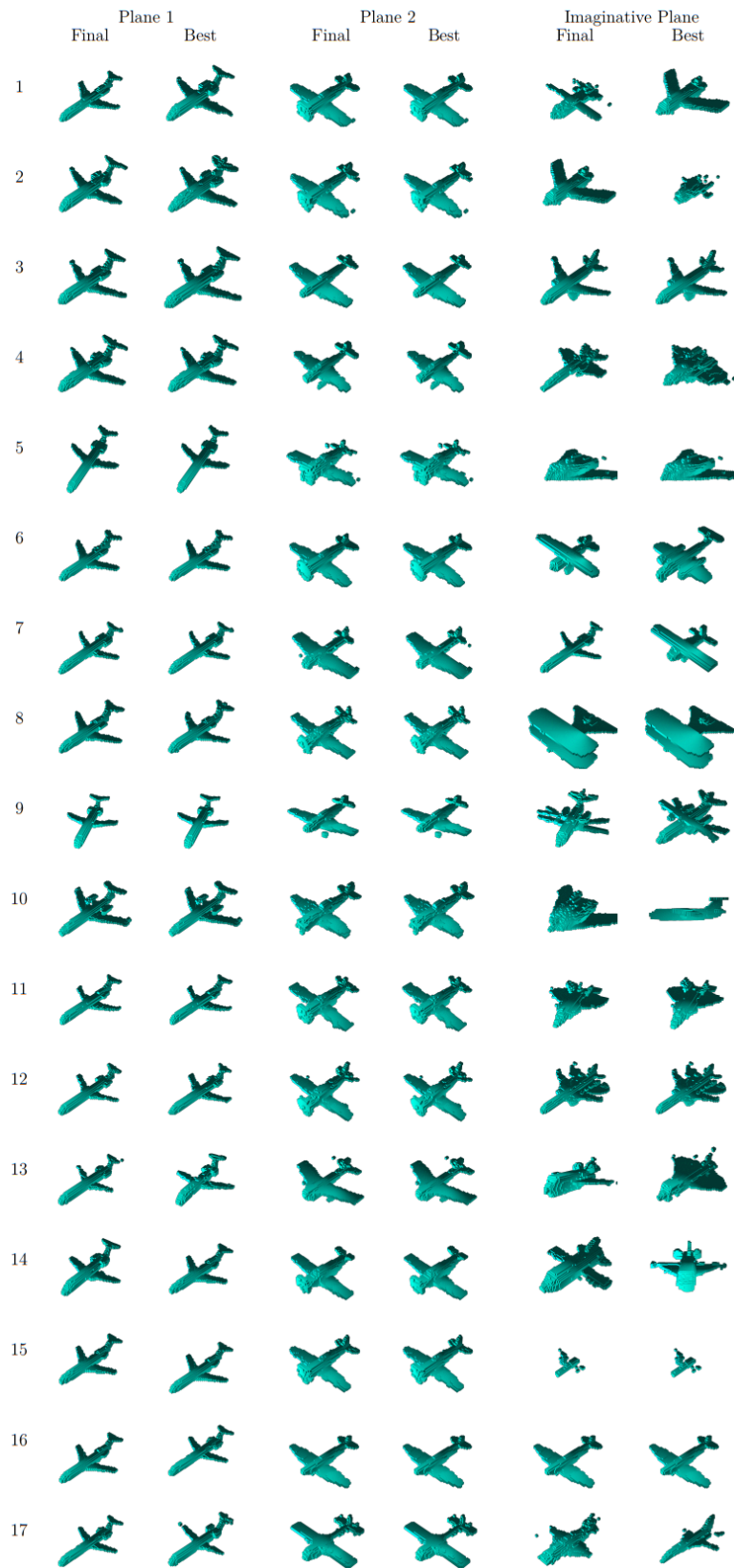


Figure 7.5: 3D models from user testing: *Normal mode*

Figure 7.6: 3D models from user testing: *Advanced mode*

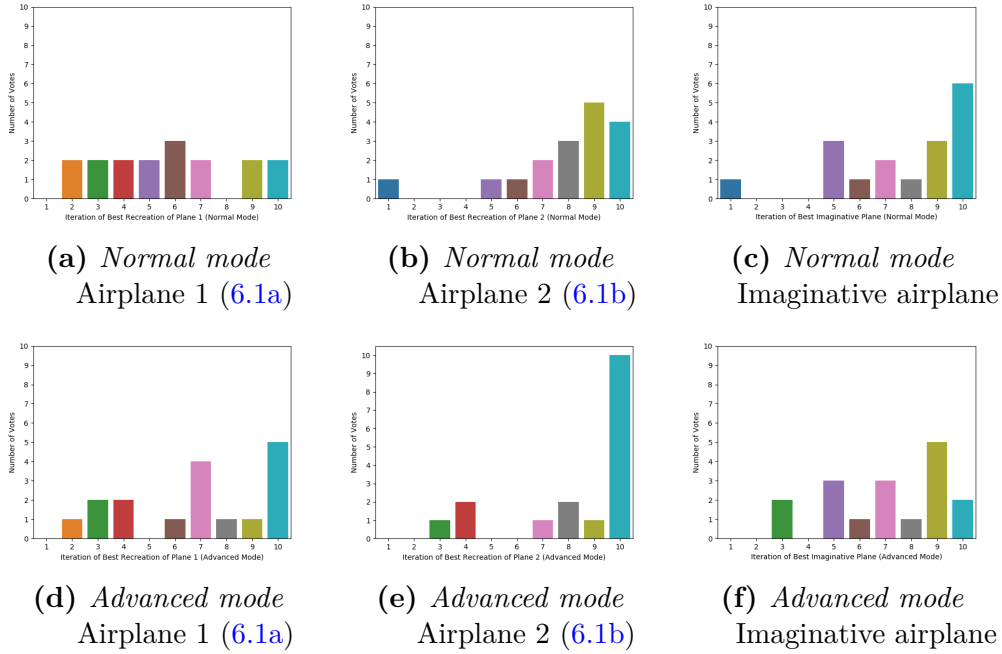


Figure 7.7: Histograms showing in which evolutionary iteration the subjectively most satisfying model occurred in each of the six tasks in the user testing

Figure 7.8 shows the distribution of subjective satisfaction level of the best 3D model in all six tasks.

Figure 7.9 shows the distribution of subjective satisfaction level of the best of the last 3D models in all six tasks.

After completing the six tasks, the volunteers were asked to evaluate the overall user experience and the ease of using the web application. The results can be seen in Figure 7.10a and Figure 7.10b respectively. These results are discussed in Section 8.3.

Lastly, the volunteers were asked questions about the following:

- Their approach in the recreation and the creative tasks respectively
- The intuitiveness of *Normal mode* and *Advanced mode* respectively
- The user guide and the overall feature set of DeepIE3D

The answers to the three questions along with all other raw answers and results can be seen in Appendix D. The findings are discussed in 8.3.

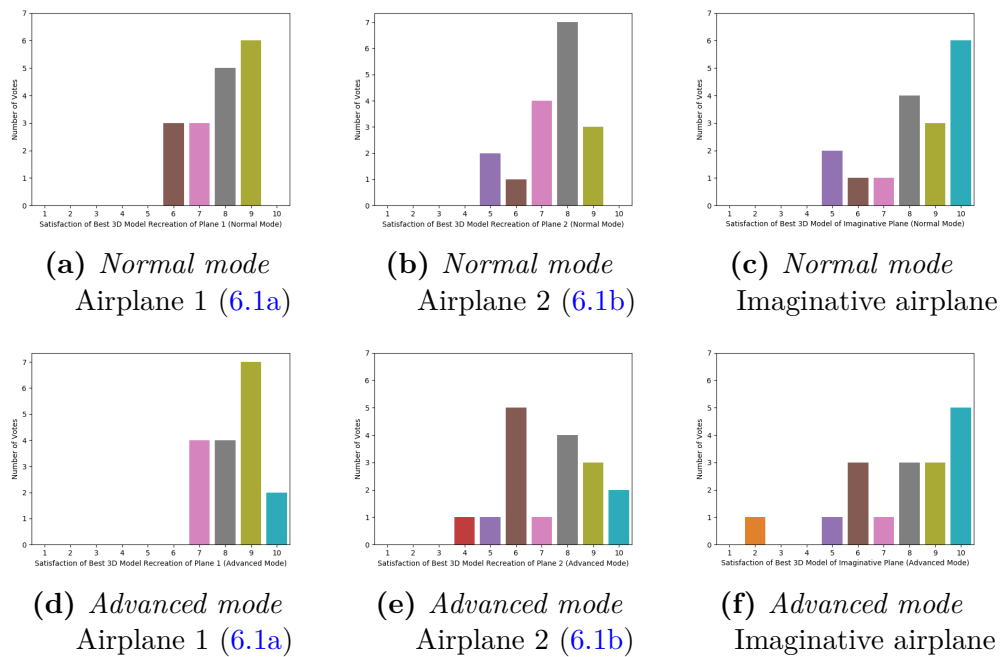


Figure 7.8: Histograms showing subjective satisfaction levels of the best model found in each of the six tasks in the user testing. Rated on a scale from 1 to 10

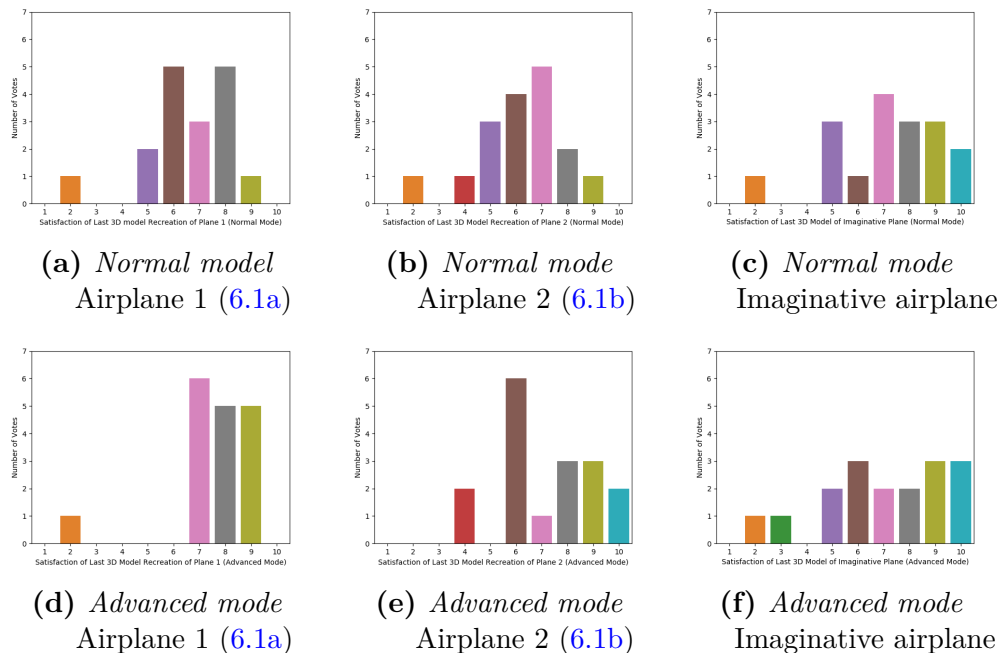


Figure 7.9: Histograms showing subjective satisfaction levels of the best of the last models found in each of the six tasks in the user testing. Rated on a scale from 1 to 10

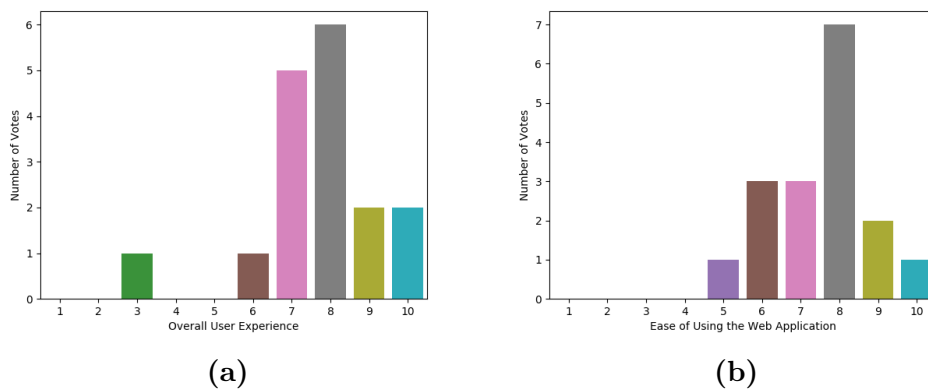


Figure 7.10: Histograms depicting the overall evaluation given after using the DeepIE3D web application throughout all six tasks. **(a)** shows the the volunteers' perceptions of the user experience. **(b)** shows the volunteers' perceptions of the ease of using the web application. Both are rated on a scale from 1 to 10

7.3. 3D Printing

One of the end products in DeepIE3D is an STL² file: After settling on a subjectively satisfying 3D model, it is possible to download it as an STL file. This has many usages including easy conversion to Minecraft³ models and 3D printing. Figure 7.11 shows nine prints made from STL files exported from DeepIE3D. The various 3D prints all have distorted features; some due to the nature of 3D printing and some due to the nature of the 3D models exported from DeepIE3D. When choosing a 3D model to export from DeepIE3D, the user has no way of knowing if the 3D model is solid inside. This can cause problems when printing.

²Definition of STL: [https://en.wikipedia.org/wiki/STL_\(file_format\)](https://en.wikipedia.org/wiki/STL_(file_format))

³Minecraft: <https://www.minecraft.net>



Figure 7.11: Nine examples of 3D printed STL files exported from the DeepIE3D web application. The colors are arbitrary and has no meaning. (a), (c), (f) and (i) are printed on an Ultimaker 2 Extended; (b), (g) and (h) are printed on an Ultimaker 2+; and (d) and (e) are printed on a Makerbot Replicator

8. Analysis & Discussion

This chapter is a discussion and analysis of the findings from Chapter 7. Additionally, it features an analysis of the PacWGAN-GP2 and the application of the evolutionary principles utilized in DeepIE3D.

8.1. Generative Adversarial Networks

This section analyzes the correctness of the GAN used in DeepIE3D; both the training of it and its ability to produce distinct 3D models.

8.1.1. Training

The final architecture in DeepIE3D is PacWGAN-GP2 trained on 512 hand-picked 3D models from the airplane domain.

Figure 8.1 and Figure 8.2 illustrate the training of the final PacWGAN-GP2. Both depict a timeline starting from $epoch = 0$ ending in $epoch = 10,000$. From 0 to 10,000, 3D models are generated in a pseudo-logarithmic fashion: Ten 3D models from 0 to 100 epochs (one every tenth), ten 3D models from 100 to 300 epochs (one every twentieth), ten 3D models from 300 to 700 epochs (one every fortieth) and so on. The two latent vectors used for generation are randomly generated using PyTorch with the same seed. This enables reproducibility while being able to explore how the generator responds to the same latent vectors throughout training. The specific latent vectors used for the two figures can be seen in Appendix B.

The two examples of training show that after relatively few epochs (around 80-100), the generator produces acceptable 3D models. Furthermore, the training process displays early stability as the 3D models resemble the overall same airplane type. These are among the key attributes of WGAN and thus WGAN-GP [5].

What cannot be seen in the two examples is the overall performance of the model. While generating 3D models during $epoch = 0$ through $epoch = 10,000$, the overall state of the 3D models was observed. It was seen that even though the 3D models in Figure 8.1 and Figure 8.2 resembled airplanes and stayed in almost the same shape, the overall state was not as stable: The 3D models improve throughout far more epochs than seen in the two examples. Furthermore, instances where the same latent vector generated different airplanes during many epochs were seen. In Figure 8.2 such a change is apparent on a minor scale. However, the mapping of the latent vector space to the 3D model space seems to stabilize when reaching around 6,000 epochs.

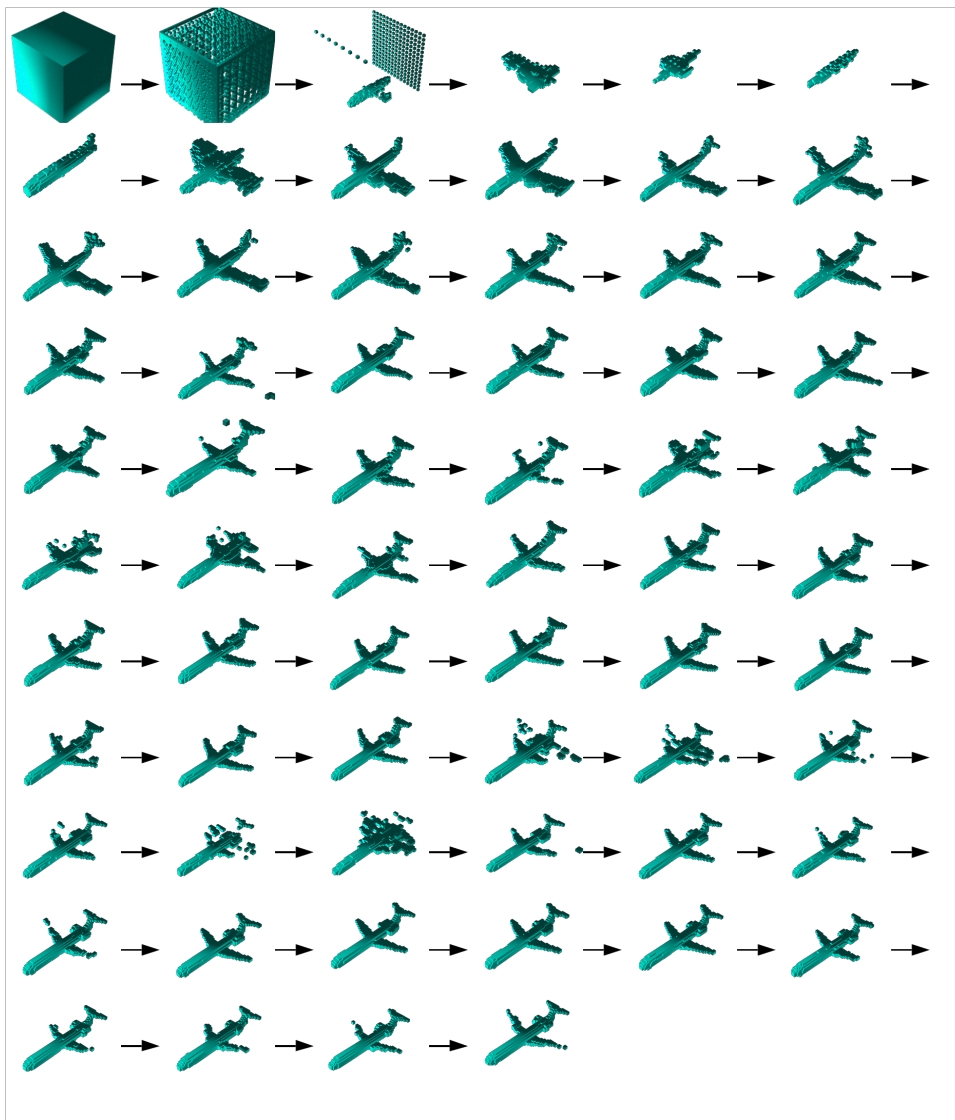


Figure 8.1: First example of 3D models generated during $epoch = 0$ through $epoch = 10,000$ when training the final PacWGAN-GP2

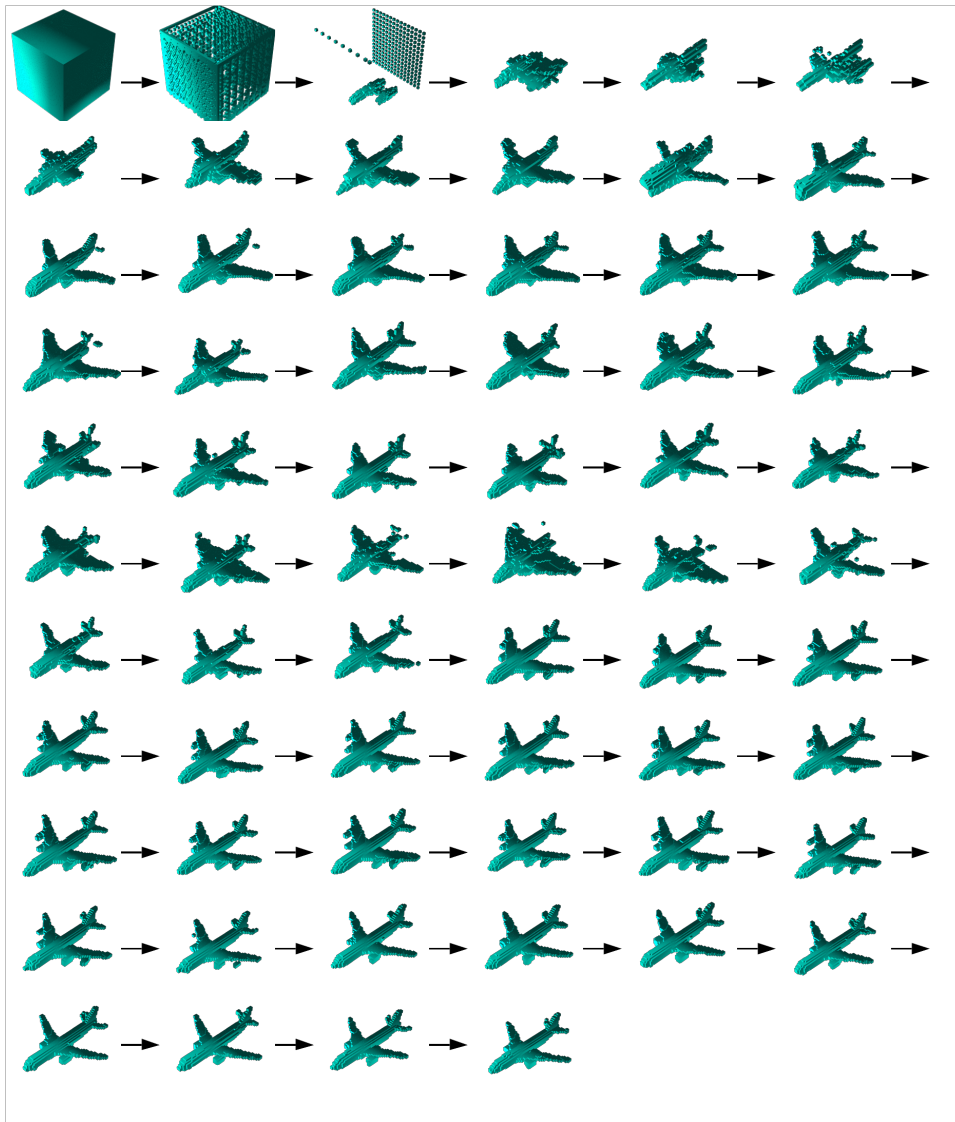


Figure 8.2: Second example of 3D models generated during $epoch = 0$ through $epoch = 10,000$ when training the final PacWGAN-GP2

As the 3D model quality is hard to measure quantitatively, the two figures and the simultaneous observations only provide an indication of how well the training process behaves.

The WGAN-GP framework is chosen in part because of its ability to address the vanishing gradient problem. The vanishing gradient problem results in a generator not being able to learn correctly. Hence, the generator is unable to progressively produce better samples. Further indication of a correct training process can thus be seen in the gradients. The gradients behave well during the entire training process, which indicates the generator keeps learning. The gradients and the gradient penalties, which are derived from the gradient norms, can be seen in Appendix E.

8.1.2. 3D Model Distribution

As mentioned, GANs are prone to mode collapse. To mitigate this, WGAN-GP is employed. To further extend the number of distinct modes, PacGAN is employed. The following demonstrates that mode collapse is mitigated in the final PacWGAN-GP2.

Figure 8.3 shows the similarity between generated 3D models and their most similar data point in the training set. None of the generated 3D models have a similarity of 100% which could have indicated overfitting. As the generator tries to produce 3D models from a specific domain (e.g. airplanes or chairs), the similarity should be high as the generated 3D models preferably are close to something that already exists.

Figure 8.4a shows the internal mean similarity distribution of the following: The training set consisting of 512 hand-picked 3D models and 512 generated 3D models. Figure 8.4b shows the distribution of the voxel count of the training set and the generated 3D models respectively. Figure 8.4a and 8.4b in unison demonstrate the distributions being very similar. This indicates that the number of modes, produced by the generator, is close to the number of modes found in the training set. It further demonstrates that these modes are alike.

These findings in themselves could indicate overfitting. However, Figure 8.3 shows that the similarity between the generated 3D models and the training set is not suspiciously high. Thus, the above demonstrates a generator capable of producing diverse and well-behaved 3D models while not overfitting.

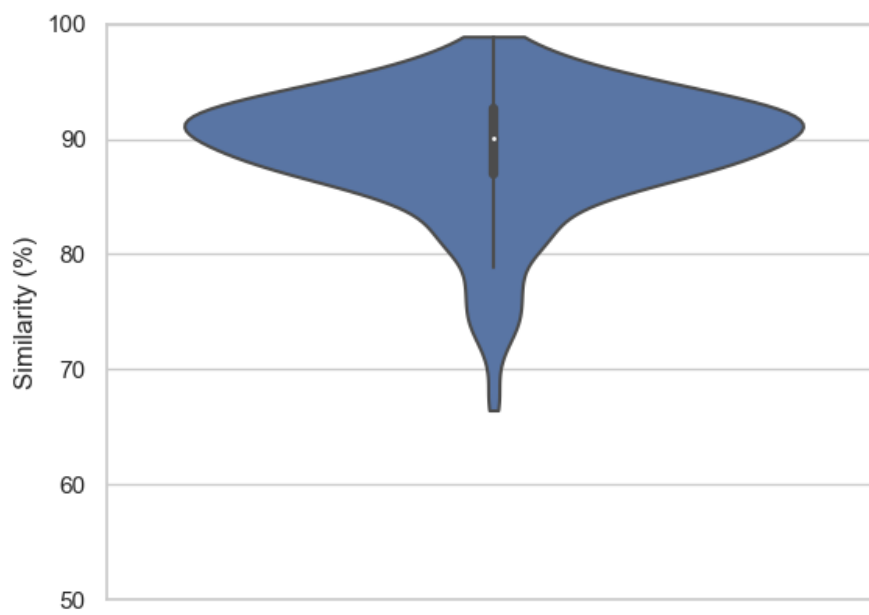


Figure 8.3: Violinplot depicting the similarity between 512 randomly generated 3D models of airplanes and their best fit in the training set. Each generated 3D model is compared to all 3D models in the training set and the similarity is reported as a percentage of voxels in the same positions

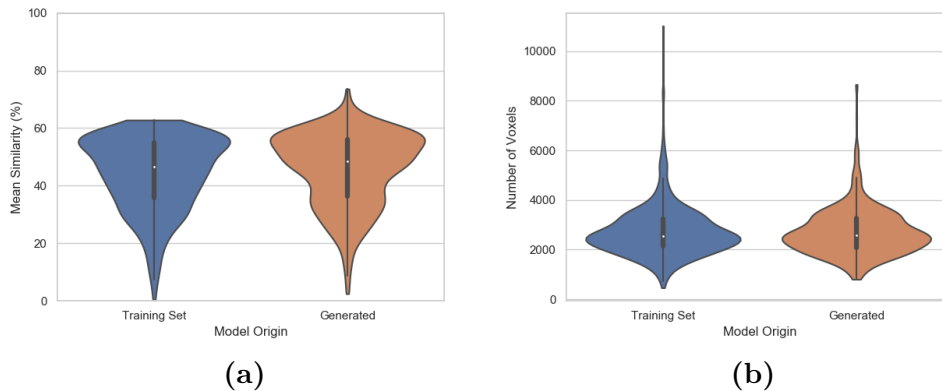


Figure 8.4: Violinplots depicting comparisons of distributions between the training set and generated 3D models. **(a)** shows the comparison of internal mean similarity distribution between the training set and the generated 3D models. The mean similarity is calculated as a mean of the similarities in their sets. All 3D models in the training set compare voxel positions to all the other training set 3D models in order to produce a similarity percentage. The same is done for 512 3D models produced by the generator. **(b)** shows the comparison of voxel count distribution between the training set and the generated 3D models

8.2. Evolution

This section considers the impact and correctness of the evolutionary principles employed in DeepIE3D. Furthermore, it addresses why behavioral Novelty Search (NS) is not included in the final web application.

8.2.1. Mutation & Crossover

Mutation and crossover is the foundation of the evolutionary flow in DeepIE3D. They are the only means of guiding the 3D models toward a target.

The purpose of mutating in DeepIE3D is to alter one or more 3D models to obtain modified but similar 3D model(s). Figure 7.1 shows that mutation works in the intended way, producing 3D models with a clear likeness but with distinct traits. As the mutation rate in Figure 7.1 is 1.0, the degree of difference between the original and its mutations is at its peak. Thus, with a lower mutation rate, the chance of having too heavily modified 3D models, such as Figure 7.1j, is likewise lower.

The purpose of crossovering in DeepIE3D is to obtain 3D models that are hybrids of distinct traits from two chosen 3D models. Figure 7.2 and Figure 7.3 show the potential of crossover, as both produce airplanes that clearly have traits from both initial 3D models. It does not perform as consistently well as mutation, since only a few airplanes in each figure contain traits from both its parents. When using the web application, only a limited number of crossovers can be performed: In *Normal mode*,

the number of crossovers is not transparently up to the user but is given by the original DeepIE algorithm. This limits the number of crossovers, leading to a lower chance of a successful one. In *Advanced mode*, the user can choose to make up to nine crossovers of the same two objects. This increases the chance of getting a crossover where traits from both parents can be seen. However, it will not produce as many clearly modified 3D models as nine mutations will. The benefits of crossover can be considerable, as it can modify 3D models towards a target 3D model in contrast to mutation. This definitely justifies the use of crossover.

Figure 8.5 shows the lineage of chosen 3D models through the evolutionary process of DeepIE3D. The lineage is recorded by one of the volunteers in the user testing but is done separately from the tasks in the questionnaire. It depicts the 3D models marked as candidate(s) for evolution in each evolutionary iteration and is performed in *Normal mode*. The goal of the lineage process is to recreate Airplane 2 (Figure 6.1b). A clear improvement can be seen in overall shape of the airplane from evolutionary iteration one to five. At evolutionary iteration five, the major missing trait is the propeller at the front of the airplane. This is found in evolutionary iteration six. The most satisfying airplane to the volunteer is expressed as being the second airplane in evolutionary iteration seven, meaning that the evolutionary process did not improve during the last three evolutionary iterations. This is possibly due to the generator not being able to produce a more satisfying airplane: There is a limit to how well a $64 \times 64 \times 64$ 3D voxel model can resemble Airplane 2.

Figure 8.5 proves that mutation and crossover through *Normal mode* is capable of producing the intended alterations wanted by a user in order to progress towards a target 3D model.

Advanced mode contains all evolutionary principles which the DeepIE algorithm employs. As the exact process happening behind the scenes in Figure 8.5 could be reenacted in *Advanced mode*, the lineage of that process is not shown. Furthermore, it would be extensive and confusing as the entire list of nine evolutions for each evolutionary iteration should be shown.

8.2.2. Novelty Search

The novelty metric used in DeepIE3D is the novelty of the latent vectors, not the novelty of the behavior, i.e. the generated 3D models. The performance when doing behavioral NS has naturally been measured, as this would have been the preferred metric. However, as seen in Figure 8.6, it will greatly increase the response time of each evolutionary iteration of 3D modelling, thus causing frustration and user fatigue; counteracting a main priority in this thesis. For comparison, exploring novelty in the latent space takes less than a second, even when nine new 3D models are needed. Please refer to Appendix A for the behavioral NS algorithm implemented in DeepIE3D.

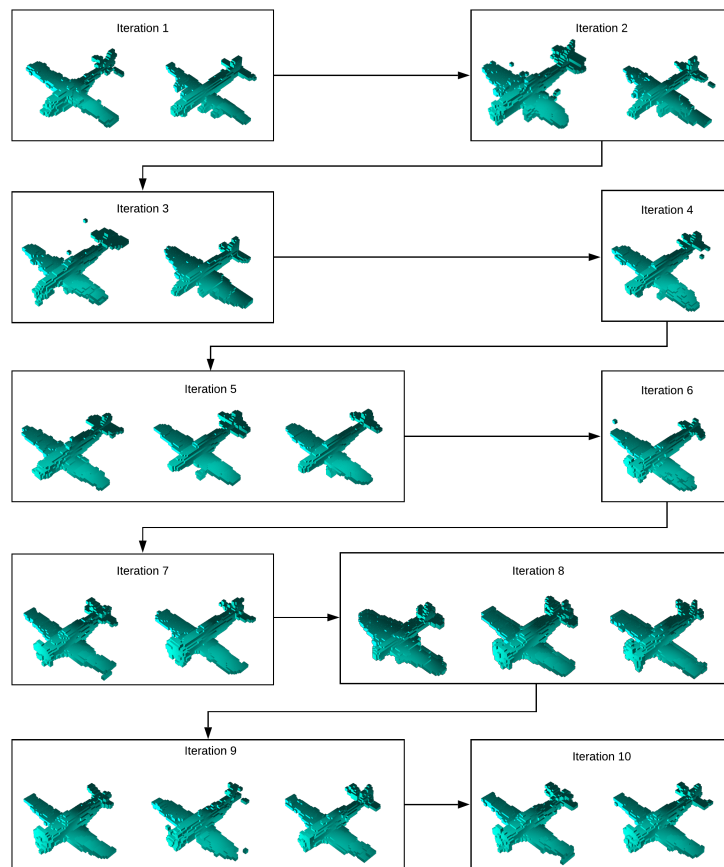


Figure 8.5: Lineage of chosen 3D models through ten evolutionary iterations when recreating Airplane 2 (Figure 6.1b) in *Normal mode* (**Task 2**)

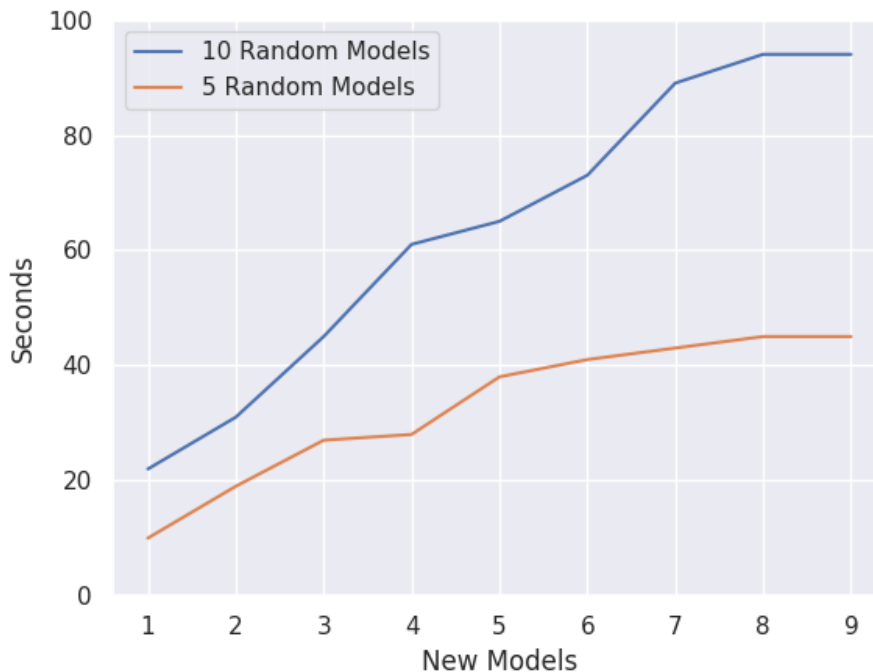


Figure 8.6: Line plot showing the performance of behavioral NS. The lowest amount of new models to generate is 14, yielding an extra wait time of 10 seconds to the web application user - or as high as 22 seconds when searching in 10 random 3D models which could be argued crucial to ensure novelty

Unfortunately, as seen in Subsection 7.1.3, the NS in latent space does not yield improvements compared to simply generating random 3D models. This leads to any kind of NS being disabled in the web application even though all are implemented.

The benefits of adding behavioral NS to DeepIE3D are clearly apparent, but the performance loss is too great. However, adding behavioral NS to DeepIE in less computationally heavy domains would likely be viable.

8.3. User Testing

The user testing of DeepIE3D was performed to assess its performance and user experience. This section elaborates on the results shown in Subsection 7.2.

Ten paired-sample t-tests¹ have been performed to evaluate the outcomes of the different user testing tasks; results can be seen in Table 8.1. With $p = 0.01$, none of the tests are statistically significant.

¹The paired-sample t-test is used to compare two sample means where there is a one-to-one correspondence between the samples [21]

A_1	A_2	μ_1	μ_2	σ_1	σ_2	p	Significant?
<i>Normal mode</i> Airplane 1 Iteration ratio (Task 1)	<i>Normal mode</i> Airplane 2 Iteration ratio (Task 2)	0.577	0.794	0.263	0.230	0.043	No
<i>Advanced mode</i> Airplane 1 Iteration ratio (Task 3)	<i>Advanced mode</i> Airplane 2 Iteration ratio (Task 4)	0.688	0.841	0.280	0.245	0.126	No
<i>Normal mode</i> Airplane 1 Iteration ratio (Task 1)	<i>Advanced mode</i> Airplane 1 Iteration ratio (Task 3)	0.577	0.688	0.263	0.280	0.237	No
<i>Normal mode</i> Airplane 2 Iteration ratio (Task 2)	<i>Advanced mode</i> Airplane 2 Iteration ratio (Task 4)	0.794	0.841	0.230	0.245	0.592	No
<i>Normal mode</i> Imaginative Iteration ratio (Task 5)	<i>Advanced mode</i> Imaginative Iteration ratio (Task 6)	0.770	0.711	0.259	0.228	0.498	No
<i>Normal mode</i> Airplane 1 Satisfaction (Task 1)	<i>Normal mode</i> Airplane 2 Satisfaction (Task 2)	6.588	6.176	1.660	1.667	0.502	No
<i>Advanced mode</i> Airplane 1 Satisfaction (Task 3)	<i>Advanced mode</i> Airplane 2 Satisfaction (Task 4)	7.588	7.176	1.660	1.878	0.515	No
<i>Normal mode</i> Airplane 1 Satisfaction (Task 1)	<i>Advanced mode</i> Airplane 1 Satisfaction (Task 3)	6.588	6.176	1.660	1.660	0.080	No
<i>Normal mode</i> Airplane 2 Satisfaction (Task 2)	<i>Advanced mode</i> Airplane 2 Satisfaction (Task 4)	6.176	7.176	1.667	1.878	0.128	No
<i>Normal mode</i> Imaginative Satisfaction (Task 5)	<i>Advanced mode</i> Imaginative Satisfaction (Task 6)	7.176	7.058	2.098	2.410	0.848	No

Table 8.1: Ten paired-sample t-tests. Iteration ratio is the ratio between the iteration in which the best 3D model occurred and total iterations used (10). A_i is the sample (a set of samples), μ_i is the sample mean and σ_i is the standard deviation of the sample

The relatively high p values lead to several implications: **1.** There is no significant difference between iterations used when recreating Airplane 1 or Airplane 2 in neither *Normal mode* nor *Advanced mode*. **2.** There is no significant difference between satisfaction level in the best recreation of Airplane 1 and Airplane 2 in neither *Normal mode* nor *Advanced mode*. **3.** There is no significant difference between iterations used in *Normal mode* and *Advanced mode* when recreating either Airplane 1 or Airplane 2. **4.** There is no significant difference between satisfaction level of the best recreations in *Normal mode* and *Advanced mode* when recreating either Airplane 1 or Airplane 2. **5.** There is no significant difference in neither iterations used nor satisfaction level when creating an imaginative plane in either *Normal mode* or *Advanced mode*.

The p values are however not the only thing apparent in Table 8.1. The average of the sample means of iteration ratio is 0.734 with the lowest mean being 0.577. This is a clear indication of the 3D models getting better via evolution. Furthermore, the sample means of satisfaction levels are all above 6.1. This is a clear indication of DeepIE3D being capable of producing subjectively satisfying 3D models. Another indication of DeepIE3D’s capability is the previously showcased Figure 7.5 and Figure 7.6 in Section 7.2. The authors of this thesis find that most of the best 3D models in the recreation tasks are satisfyingly close to the intended airplane targets. This is however another subjective evaluation.

None of the paired-sample t-tests show significant differences between any tasks done

in *Normal mode* and *Advanced mode*. This is a clear indication of the two modes being similarly suited for interactive evolutionary 3D modelling. Thus, users can choose between the original DeepIE algorithm from Bontrager et al. [2] and the novel custom approach introduced in this thesis - without compromising performance.

The three above mentioned indications are key insights in this thesis: DeepIE3D successfully employs evolution to such a degree that users achieve subjectively satisfying results. At the same time, the novel custom approach, *Advanced mode*, has been implemented successfully.

Subsequent paragraphs goes into depth with results from the user testing.

Strategies After completing the six tasks, the volunteers were asked to reflect on their choice of strategy in the recreation and creative tasks respectively.

In *Normal mode*, the volunteers simply choose their candidate(s) for evolution and instantiate the next evolutionary iteration. In *Advanced mode*, variations of strategies were observed. Some choose to fully exploit the list of actions for the next evolutionary iteration. This often entails many mutations/crossovers of one favored 3D model, few mutations/crossovers of less favored 3D models and keeping the best 3D model. Others either do not fully exploit the list or choose to generate many new 3D models in the next evolutionary iteration. Another variation in *Advanced mode* is the number of features utilized. One volunteer finds the crossover feature cumbersome, since it involves many clicks and thus chooses not to use it as extensively as the mutate feature. Few choose not to use it at all, as the effect is not clear to them. Most volunteers choose to use the keep feature. All use the mutate feature.

The volunteers do not modify their strategy much during the tasks. The strategies in the recreation tasks can be divided into four categories:

1. Select Best Likeness
2. Choose Specific Traits
3. Consider Both Specific Traits and Best Likeness
4. Select Best Likeness First, Then Choose Specific Traits

The most commonly applied strategy is Strategy 1 with $\frac{8}{17}$ volunteers following it. This strategy consists of marking one or more 3D models with overall likeness to the target as candidate(s) for evolution. The strategy corresponds to the 'Select Best Likeness' strategy from Bontrager et al. [2]. The way the strategy is applied varies between volunteers. Similarly to Bontrager et al. [2], two variations of Strategy 1 were observed: Choosing one similar 3D model or choosing some or all similar 3D models.

Strategy 2 is the second most applied strategy with $\frac{4}{17}$ volunteers following it. It focuses on distinct traits of the 3D models. Evolution candidate(s) are marked with

the intent of having the modified 3D models in the next evolutionary iteration possibly being closer to the target.

Strategy 3 and 4 are combinations of Strategy 1 and 2. Strategy 3 enforces an approach, where Strategy 1 and 2 are applied at the same time. It often entails marking more 3D models as candidates for evolution than both Strategy 1 and 2. Strategy 4 is only followed by one volunteer. It can be described as first following Strategy 1 until the 3D models become similar to the target. Then choosing 3D models with distinct traits in order to have the 3D models from Strategy 1 get the traits through evolution.

The strategies in the creative tasks are not consistent. The reason for this is twofold: First, the question about the strategy used in **Task 5** and **Task 6** is misunderstood as the strategy used in **Task 3** and **Task 4** by some volunteers. This might be due to the complexity of the questionnaire. Second, even though it is stated that the 3D models in the tasks should be without a preliminary view of the final airplane, some volunteers have just that. By reviewing the answers seemingly about the right tasks and by observing the volunteers, some indication of strategy can be formed: Strategies similar to the ones followed in the recreation tasks are applied. The goal of the evolutionary process is, however, different, as many volunteers want to make distorted airplanes after the four tasks concerned with precision.

One common observation throughout all strategies is that situations arise, where volunteers do not find any 3D models satisfying enough to mark as candidate(s) for evolution. Some volunteers expect to get similar results to the ones present when not choosing any candidate(s). This is intentionally not the case: All previous work is discarded and the next evolutionary generation consists solely of newly generated 3D models.

Experience After completing the six tasks, the volunteers were asked to reflect on the intuitiveness of *Normal mode* and *Advanced mode* respectively. $\frac{2}{17}$ did not find *Normal mode* intuitive, while $\frac{3}{17}$ did not find *Advanced mode* intuitive. While this difference is small, $\frac{6}{17}$ commented on *Advanced mode* being harder to use than *Normal mode*.

One volunteer said:

You do not really know what is happening behind the scenes. It is faster to use, as you don't have to decide on many different options.

- Volunteer 12

On the perception of *Normal mode*

This sums up the general chit-chat overheard while overseeing the user testing: The volunteers are happy about the ease of *Normal mode*, but do not have a clue about "... *what is happening behind the scenes ...*".

On the other hand, the perception of *Advanced mode* is quite the opposite. One volunteer said:

It was a little hard the first time.
Second and third it provided a lot of
info of what was happening. Took a
little longer.

- Volunteer 12

On the perception of *Advanced mode*

When grasping the amount of information displayed in *Advanced mode*, the volunteers seem to enjoy the added control. These opposite perceptions of *Normal mode* and *Advanced mode* is the characteristic clash between how user friendly an application is and how configurable it is. Luckily, DeepIE3D caters for both user types.

However, the perceptions all come from volunteers not being familiar with 3D modelling, evolution or computer science in general. It is highly likely that this creates a bias towards favoring *Normal mode*, but unfortunately this cannot be supported by evidence in this thesis.

After completing the six tasks, the volunteers were further asked to assess the overall user experience and the ease of using the web application. The user experience is graded from 1 to 10, where 1 is an exhaustive experience and 10 is a fun experience. The mean of the user experience rating is $\mu = 7.65$ and the median is $Md = 8$. This high rating suggests that DeepIE3D to a certain degree mitigates user fatigue - at least within ten evolutionary iterations.

The ease of using the web application uses the same scale (1 being hard, 10 being easy). The mean of the ease rating is $\mu = 7.53$ and the median is $Md = 8$. This compliments the user experience rating while consolidating DeepIE3D's ability to adapt a complex subject to a user friendly web application.

9. Future Work

DeepIE3D is already capable of performing deep interactive 3D modelling to a satisfying degree. Given more time, some areas of improvements may be beneficial to explore.

Data An obvious way to improve the generative model is to alter the data set. The training set used for the final models in this thesis consists of 512 handpicked 3D models of airplanes. The number of modes found in the data set naturally increases with the number of data points, thus a larger data set could lead to a generator able to produce a wider range of modes. Additionally in regards to the training set, raising the resolution of the 3D models in the training set will increase the resolution of the generated 3D models to the same degree. An increase in quality would ensure satisfaction in both real life 3D printing and software applications utilizing 3D models. However, both alterations entails substantially longer training time and render time in the web application.

Class-conditional GAN The generative model in DeepIE3D is trained to generate 3D models from one domain (e.g. airplanes or chairs). This is the case, as it strives to learn a probability distribution in a set of artifacts. If the artifacts in a training set are not in the same domain, the distribution is hard to imitate. Mirza and Osindero [22] introduce a conditional version of GAN (CGAN), which relies on auxiliary information such as class labels. This enables an option to direct the generation process towards producing artifacts from distinct domains. Thus, one-to-many mappings are used to produce a conditional predictive distribution [23]. The only alteration to the networks is that both take the auxiliary information in addition to the original input. Using a CGAN in DeepIE3D would enable a single model to produce 3D models from distinct domains (e.g. both airplanes and chairs). Another application of CGAN in DeepIE3D would be using descriptive user tags as the auxiliary information. These tags would make it possible to produce 3D models matching a selection of user tags. This could even be applied within a category such as airplanes in order to produce a more specific airplane (e.g. a WW2 fighter or a private jet).

A framework similar to CGAN is introduced by Brock et al. [24]: It is called BigGAN. In addition to being class-condition, BigGAN are capable of synthesizing 2D images of remarkable high resolution. This possibility of distinct artifact domains in combination with high resolution would be interesting to explore in 3D. However, it requires a vast amount of computational power.

Improving WGAN-GP Training The training process of WGAN-GP is not per-

fect. It addresses the mode collapse and vanishing gradient problem experienced in the original GAN and DCGAN. However, it does not create a perfect generator, as not all modes can be created. PacGAN helps the generator explore more modes, but it does not ensure that all modes are present. Wei et al. [25] introduce an improvement of WGAN-GP training. It points out a fault of gradient penalty: As the continuity is checked with interpolations of real and fake data points, it can neglect to check the continuity of regions close to the real data. Wei et al. [25] mitigate this by checking the continuity in two perturbed versions of a real data point. The two perturbations are close to the real data point. Using this improved training of WGAN-GP has in other cases led to more modes and higher quality images. Thus, applying it to the training process of the generative model used in DeepIE3D could widen the range of modes.

Scale-specific Controls Adding scale-specific controls in order to generate artifacts in DeepIE3D would be an interesting field to explore. Karras et al. [26] introduce StyleGAN, an alternative generator architecture. It provides an automatically learned, unsupervised separation of high-level attributes and stochastic variation in the generated samples. Karras et al. [26] have only shown this behaviour for GANs trained to generate images. The separation of high-level attributes and stochastic variation enables intuitive scale-specific mixing and interpolation operations [26]. If the generator architecture can be applied to generate 3D models in voxel space, it will enable the user to tune specific traits of a 3D model. As StyleGAN separates coarse to fine styles in the artifacts, many different traits can be modified. Coarse styles in the 2D image domain could be pose, general hair style, face shape etc.; where finer styles would be colors and specific facial features. Karras et al. [26] also introduce a term called style-mixing. Style-mixing enables mixes of two latent vectors in a crossover-like fashion in which coarse to fine styles can be applied from one artifact to another. The advantages of StyleGAN can be applied in multiple fashions in DeepIE3D: In *Normal mode*, the mutations and crossovers can differ in degree of style, thus creating a wider range of evolutions. In *Advanced mode*, the degree of styles can be decided for mutations and crossovers by the user. Crossovers could be much more specific, e.g. having coarse styles of one artifact and fine styles of another. This could make crossovers more useful.

Expert User Testing DeepIE3D is solely user tested on novice users with a limited understanding of the computer science field. Further user testing on advanced users would explore the capabilities DeepIE3D as a tool for real life and production applications. Testing users with much experience in 3D modeling would be especially impactful.

10. Conclusion

The novel framework, DeepIE3D, introduced in this thesis can successfully be applied as a 3D modelling tool. The user testing demonstrates that novice users are capable of producing subjectively satisfying 3D models using DeepIE3D while not suffering from user fatigue. This applies to both recreation and creative 3D modelling tasks. The results of the tasks in the user testing are similar to those found in DeepIE [2]. This indicates a correct implementation of the original approach with a favorable outcome. Furthermore, the tasks performed using the novel custom approach, *Advanced mode*, produce equally positive results. It is thus deemed to be just as adequate, while offering a more customizable approach to end users wanting more control.

The above clearly indicates the compatibility of the DeepIE approach in the domain of 3D models. Furthermore, it shows the correctness of the underlying architecture in DeepIE3D. In continuation of the observations from user testing, the results from a wide range of experiments ensures exactly this correctness. Thus, it can be concluded that utilizing PacWGAN-GP2 as phenotype/genotype mapping from latent space to 3D model space is sound. The experiments further demonstrate that the PacWGAN-GP2 used in DeepIE3D is capable of generating a plethora of different 3D models.

DeepIE3D also proves its worth in practical applications: The process of converting a 3D model exported from DeepIE3D to a 3D printed model is simple and easy.

The application of Novelty Search (NS) in DeepIE3D does not yield favorable results within a manageable time frame: Performing NS in latent space does not improve novelty in behavioral space, while performing NS in behavioral space is too computationally expensive, making it infeasible in practice. However, applying behavioral NS in less computationally heavy domains may be viable.

DeepIE3D is already capable of performing deep interactive 3D modelling to a satisfying degree, but possible improvement could increase its utility. A common category of possible improvements concerns the underlying architecture. These improvements can lead to a single generator capable of producing 3D models from distinct domains, the capability of scale-specific controls and improved resolution in the generated 3D models.

Bibliography

- [1] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, 2016. URL <https://arxiv.org/pdf/1511.06434.pdf>.
- [2] Philip Bontrager, Wending Lin, Julian Togelius, and Sebastian Risi. Deep interactive evolution. *European Conference on the Applications of Evolutionary, Computation (EvoApplications)*, 2018. URL http://sebastianrisi.com/wp-content/uploads/bontrager_evomusart18.pdf.
- [3] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *NIPS*, 2016. URL http://3dgan.csail.mit.edu/papers/3dgan_nips.pdf.
- [4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 2672–2680, 2014. URL <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>.
- [5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *CoRR*, 2017. URL <https://arxiv.org/pdf/1701.07875.pdf>.
- [6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML 2015*, 2015. URL <https://arxiv.org/pdf/1502.03167.pdf>.
- [7] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *Published in NIPS 2017*, 2017. URL <https://arxiv.org/pdf/1704.00028.pdf>.
- [8] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael Cree. Regularisation of neural networks by enforcing lipschitz continuity. *CoRR*, 2018. URL <https://arxiv.org/pdf/1804.04368.pdf>.
- [9] Cédric Villani. *Optimal Transport: Old and New (Grundlehren der mathematischen Wissenschaften Book 338)*. Springer, oct 2008. URL <https://www.xarg.org/ref/a/B00FC7F5OM/>.

-
- [10] Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. Pacgan: The power of two samples in generative adversarial networks. *NeurIPS 2018*, 2017. URL <https://arxiv.org/pdf/1712.04086.pdf>.
- [11] Jerry Liu, Fisher Yu, and Thomas A. Funkhouser. Interactive 3d modeling with a generative adversarial network. *CoRR*, abs/1706.05170, 2017. URL <https://arxiv.org/pdf/1706.05170.pdf>.
- [12] Brian G. Woolley and Kenneth Stanley. A novel human-computer collaboration: combining novelty search with interactive evolution. *GECCO 2014*, 07 2014. URL https://eplex.cs.ucf.edu/papers/woolley_gecco14.pdf.
- [13] GitHub user: rimchang. 3dgan-pytorch, 2017. URL <https://github.com/rimchang/3DGAN-Pytorch>.
- [14] PyTorch. Torch.nn, 2019. URL <https://pytorch.org/docs/stable/nn.html>.
- [15] Soumith Chintala. Gan hacks, 2016. URL <https://github.com/soumith/ganhacks>.
- [16] Daniel Sieta. Understanding generative adversarial networks, 2017. URL <https://danieltakeshi.github.io/2017/03/05/understanding-generative-adversarial-networks/>.
- [17] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *CoRR*, abs/1611.02163, 2017. URL <https://arxiv.org/pdf/1611.02163.pdf>.
- [18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CLR 2015*, 2014. URL <https://arxiv.org/pdf/1412.6980.pdf>.
- [19] PyTorch. Torch.optim, 2019. URL <https://pytorch.org/docs/stable/optim.html>.
- [20] Tom White. Sampling generative networks. *NIPS 2016*, 2016. URL <https://arxiv.org/pdf/1609.04468.pdf>.
- [21] Evie McCrum-Gardner. Which is the correct statistical test to use? *British Journal of Oral and Maxillofacial Surgery*, 46(1):38 – 41, 2008. ISSN 0266-4356. doi: <https://doi.org/10.1016/j.bjoms.2007.09.002>. URL <http://www.sciencedirect.com/science/article/pii/S0266435607004378>.
- [22] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014. URL <https://arxiv.org/pdf/1411.1784.pdf>.
- [23] Vincent Chen, Liezl Puzon, and Christina Wadsworth. Class-conditional super-resolution with gans. 2017. URL <https://pdfs.semanticscholar.org/e1de/667ec1dc1f255b3b3810af8dff4ad355c337.pdf>.
-

- [24] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018. URL <http://arxiv.org/abs/1809.11096>.
 - [25] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. Improving the improved training of wasserstein gans: A consistency term and its dual effect. *CoRR*, abs/1803.01541, 2018. URL <https://arxiv.org/pdf/1803.01541.pdf>.
 - [26] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018. URL <https://arxiv.org/pdf/1812.04948.pdf>.
-

A. Behavioral Novelty Search

Algorithm 2: Novelty Search in Behavioral Space

Defaults: $m \leftarrow 9$, $n \leftarrow 200$, $k \leftarrow 10$

```
1 Begin
2    $G_\theta \leftarrow$  pre-trained generator // DeepIE3D uses PacWGAN-GP2
3    $Z \leftarrow$  m by n matrix where  $Z_{i,j} \sim \mathcal{N}(\mu, \sigma^2)$ 
4    $indices \leftarrow$  GetChosenModels() // UI is responsible for this
5   BehavioralNoveltySearch( $Z_{indices}, G_\theta$ )
6 Function BehavioralNoveltySearch( $Z, G$ ):
7    $models \leftarrow$  EMPTY LIST
8   if  $Z$  length == 0 then
9      $noise \leftarrow$  vector of length n where  $noise_i \sim \mathcal{N}(\mu, \sigma^2)$ 
10     $Z \leftarrow Z + noise$ 
11  end
12  foreach vector  $v$  of the matrix  $Z$  do
13     $models.add(G(v))$ 
14  end
15  while  $Z$  length <  $m$  do
16    ( $tempZ, tempModels$ )  $\leftarrow$  GenerateModels( $G, k$ )
17     $similarity \leftarrow 100$ 
18     $index \leftarrow -1$ 
19    foreach ( $tempIndex, tempModel$ ) of  $enumerate(tempModels)$  do
20       $tempSimilarity \leftarrow$  GetSimilarity( $tempModel, models$ )
21      if  $tempSimilarity < similarity$  then
22         $similarity \leftarrow tempSimilarity$ 
23         $index \leftarrow tempIndex$ 
24      end
25    end
26     $Z \leftarrow Z + tempZ_{index}$ 
27     $models.add(G(tempZ_{index}))$ 
28  end
29  return  $Z$ 
```

```

30
31 Function GenerateModels( $G, k$ ):
32    $tempModels \leftarrow$  EMPTY LIST
33    $tempZ \leftarrow$   $k$  by  $n$  matrix where  $Z_{i,j} \sim \mathcal{N}(\mu, \sigma^2)$ 
34   foreach vector  $v$  of the matrix  $tempZ$  do
35      $tempModels.add(G(v))$ 
36   end
37   return ( $tempZ, tempModels$ )
38 Function GetSimilarity( $model, models$ ):
39    $nonzeros \leftarrow model.nonzero()$  // non-zero elements in model
40    $voxelCount \leftarrow 0$ 
41   foreach  $tempModel$  of  $models$  do
42     foreach  $(x,y,z)$  of  $nonzeros$  do
43       if  $tempModel[x][y][z] == 1$  then
44          $voxelCount \leftarrow voxelCount + 1$ 
45       end
46     end
47   end
48   return  $\frac{voxelCount}{models\ length} \cdot \frac{nonzeros\ length}{100}$  // Mean percentage similarity
49 end

```

B. Latent Vectors Used in Training Analysis

Latent vector used in Figure 8.1:

[
-1.0408011674880981, 0.9166041612625122, -1.3041905164718628,
-1.1096783876419067, -1.2187780141830444, 1.1675925254821777,
-1.0573921203613281, -0.11883937567472458, -0.9078398942947388,
0.34522223472595215, -0.5713335871696472, -0.2351086437702179,
1.0076055526733398, -0.7528814673423767, -0.22499920427799225,
-0.43268606066703796, -1.5071438550949097, -0.4585607945919037,
-0.8480006456375122, 0.5266043543815613, 0.029916180297732353,
-0.04983803257346153, 1.0650780200958252, 0.8860366940498352,
0.4640183746814728, -0.4986324608325958, 0.12886369228363037,
2.7630667686462402, 0.14047646522521973, 1.1191014051437378,
0.315231591463089, 1.7527765035629272, -0.7649639248847961,
1.8298852443695068, -0.27840104699134827, -0.2719452679157257,
-1.2944107055664062, -0.02431253343820572, -0.23535971343517303,
-0.7087094783782959, 1.1566312313079834, 0.42960160970687866,
-1.1873589754104614, -0.7467758655548096, -0.9319808483123779,
-0.8578645586967468, -0.9647331237792969, -0.0991455465555191,
-1.0189824104309082, 0.31567901372909546, -1.6035629510879517,
1.8493320941925049, 0.04472726583480835, 1.5852519273757935,
-0.5912226438522339, 1.1312177181243896, 0.7562121748924255,
-1.2022933959960938, -0.5833470225334167, -0.44068679213523865,
-1.9791470766067505, 0.7787133455276489, -0.7748500108718872,
-0.13975095748901367, 1.141386866569519, -0.635371744632721,
-1.4702459573745728, -0.21338605880737305, -0.8706575632095337,
1.6159112453460693, -0.23564793169498444, 0.9443864226341248,
2.113386631011963, -0.9754034876823425, 0.17569366097450256,
-0.13188815116882324, -0.27350300550460815, 0.3355262577533722,
0.18854671716690063, 2.143237352371216, 0.8527002930641174,
0.09647636860609055, -0.06249098479747772, 0.8268541097640991,
0.5598673224449158, -0.7775934338569641, 0.3339212238788605,
0.17591367661952972, 2.1108040809631348, 1.0702455043792725,

0.019540296867489815, 1.121282935142517, -1.487319827079773,
 -0.2043229192495346, -1.0466426610946655, -1.5772475004196167,
 0.10358445346355438, -0.3514097034931183, 0.2421167939901352,
 0.6462650895118713, 0.8729978799819946, -0.9275988936424255,
 0.17662756145000458, 1.0224436521530151, -0.48255372047424316,
 -0.542116641998291, -0.5341827273368835, -0.6412885189056396,
 0.035191990435123444, -0.4765148460865021, -0.040874917060136795,
 1.1992886066436768, 0.537361741065979, -0.19295910000801086,
 0.5936785936355591, 0.7203136086463928, 0.5061110258102417,
 1.5191761255264282, -0.4896864891052246, 0.9230629801750183,
 -0.6008041501045227, -1.1164305210113525, 0.2577155530452728,
 -0.7225574254989624, -0.924393355846405, 1.8736815452575684,
 1.012195348739624, -1.4481744766235352, -0.06435244530439377,
 0.32154718041419983, 0.5907514095306396, -1.4196633100509644,
 0.8279211521148682, -0.29685503244400024, 0.7120362520217896,
 -0.20681561529636383, -0.15475818514823914, 0.15530990064144135,
 -0.04888802021741867, 0.34295564889907837, 0.12640085816383362,
 0.15189151465892792, -1.3638681173324585, -1.6592905521392822,
 1.0311496257781982, -1.9556775093078613, -0.1482047438621521,
 1.7375658750534058, 2.2039496898651123, -0.6589270234107971,
 1.3314379453659058, -0.4497975707054138, 0.5493302345275879,
 0.053876012563705444, 0.2600640058517456, 0.8570226430892944,
 2.521064281463623, -0.04523999243974686, -0.31052321195602417,
 -0.9407292008399963, -0.003390141064301133, 1.5198700428009033,
 0.2654098570346832, -0.14413532614707947, 0.5406927466392517,
 -1.5476484298706055, 0.645520031452179, -1.1381512880325317,
 0.605167806148529, 1.1903902292251587, 1.2194879055023193,
 -0.04704220965504646, -1.0913935899734497, 1.0223486423492432,
 0.21128413081169128, 0.030614925548434258, 0.3603813350200653,
 0.3165828287601471, -0.8975173234939575, -0.639275848865509,
 0.6206748485565186, -0.168959379196167, -0.5815512537956238,
 1.263188123703003, -0.5647768974304199, -0.13118743896484375,
 0.6836482882499695, -0.25280386209487915, -1.0559577941894531,
 0.04088050499558449, 1.0751616954803467, -0.552819550037384,
 0.23216204345226288, 0.6321074962615967, 0.4733217656612396,
 0.2921431362628937, -0.23260068893432617, 1.8261481523513794,
 1.0807431936264038, -0.7225019931793213
]

Latent vector used in Figure 8.2:

[-1.2113046646118164, 0.6303586959838867, -1.4713038206100464,
]

-1.3351988792419434, -0.4896668791770935, 0.13174211978912354,
0.3294970691204071, 0.32642850279808044, -0.48055046796798706,
1.1031602621078491, 2.548506021499634, 0.30063533782958984,
-0.543218195438385, -1.0841293334960938, 1.4612364768981934,
-1.627929925918579, -1.4800710678100586, -1.0630654096603394,
0.3630143702030182, 0.3994666635990143, 0.14565789699554443,
-0.7344775199890137, -0.9872555732727051, 1.8512191772460938,
-1.3437334299087524, 0.853526771068573, 0.8811348080635071,
-0.6521903872489929, 0.5809689164161682, 0.3560802638530731,
0.016046009957790375, 0.40185678005218506, 1.9538429975509644,
-0.4460289478302002, 1.7102055549621582, 0.8944460153579712,
-0.5458323359489441, -0.6418041586875916, -2.052621364593506,
0.3467080593109131, -0.6968749761581421, -0.004699554294347763,
-0.31362661719322205, -1.2601573467254639, 0.697658360004425,
0.3720381557941437, -0.2606133818626404, -0.7613316774368286,
0.12767551839351654, 0.1522265523672104, -1.108310341835022,
-0.6451759338378906, -1.787100911140442, 0.6950071454048157,
-0.5825005173683167, -0.19255627691745758, 1.2356798648834229,
-0.8008244037628174, -0.2808338701725006, 0.8700663447380066,
-1.7343510389328003, -1.4346975088119507, -0.06284128874540329,
-0.5595195293426514, 1.0409702062606812, 0.13654087483882904,
1.8124992847442627, -0.4949057102203369, 0.8339433073997498,
-0.3967777490615845, 1.3932565450668335, 0.30117112398147583,
-0.2569783329963684, -1.3998596668243408, 0.7614684700965881,
0.8895881772041321, -0.39031508564949036, 0.8299045562744141,
0.2927452623844147, -0.6837154626846313, -0.22316285967826843,
-0.7009411454200745, -0.32141852378845215, -0.6802502274513245,
-1.3113828897476196, -1.5171953439712524, 0.628166913986206,
1.1301010847091675, 0.41947686672210693, 0.6914551854133606,
-0.8920358419418335, 0.2698323428630829, -2.215027093887329,
0.8058066964149475, 0.515622079372406, -0.04202818125486374,
-2.835059881210327, 0.3415207266807556, -0.585829496383667,
0.6763635277748108, -0.4266972541809082, 1.095131516456604,
1.24285888671875, 0.29533126950263977, -0.4089600741863251,
-0.6816136240959167, -0.7103906869888306, -0.11576968431472778,
-1.6770185232162476, 1.2439218759536743, 0.8127437233924866,
-0.8698902726173401, -0.716035008430481, -0.5617383718490601,
1.695793628692627, -0.37110382318496704, 0.004497247282415628,
0.14678776264190674, -0.277641624212265, 0.30988118052482605,
-0.7398537397384644, 0.6421766877174377, -0.025866955518722534,
-0.9036566019058228, 0.5111561417579651, 0.8201687335968018,
1.0510227680206299, -0.8726540803909302, -0.8036245703697205,

-0.18871735036373138, 1.729634165763855, 0.10785476118326187,
-0.6363270878791809, 0.051877859979867935, 2.0020792484283447,
-0.6008415818214417, 0.3244907259941101, 1.1994245052337646,
-1.1465080976486206, -0.930880606174469, -0.9179940819740295,
-0.7409375309944153, -1.0485632419586182, -1.3647245168685913,
0.34897133708000183, -0.5539128184318542, -0.44666504859924316,
1.7612162828445435, 1.0009013414382935, -0.5218213200569153,
1.4748759269714355, -0.683250367641449, -0.08347616344690323,
0.685611367225647, -0.7607735991477966, -3.1045682430267334,
-1.3223273754119873, 0.4691530764102936, -0.5240707993507385,
-0.37024804949760437, -0.396274209022522, -0.11807964742183685,
-0.9285410642623901, 0.3364261984825134, 1.0358314514160156,
0.9010142683982849, 1.1425833702087402, 0.35128119587898254,
0.953509509563446, 0.19746844470500946, 0.9650968313217163,
1.206891655921936, -0.24858546257019043, -0.18955329060554504,
2.187729835510254, 0.8639533519744873, -0.8377084732055664,
-0.1815066933631897, -0.8690583109855652, -0.22122421860694885,
-1.7606292963027954, 0.4089822471141815, -0.08567928522825241,
-0.7367804646492004, -0.11829587072134018, 0.3907356262207031,
-1.3913357257843018, -1.3956661224365234, 0.546488881111145,
-0.30829501152038574, -0.38502684235572815, 1.1338926553726196,
0.047884371131658554, 1.5241358280181885, 0.8959890007972717,
-2.4688668251037598, -0.2654874622821808, 1.0005842447280884,
-0.03324358910322189, 1.4986847639083862

]

C. User Testing: Questionnaire

The following pages are the questionnaire used in the user testing of DeepIE3D.



Deep Interactive Evolutionary 3D Modelling: Planes

<https://adrianwesth.dk>

Navnet og billedet, der er knyttet til din Google-konto, registreres, når du uploader filer og indsender denne formular. Er **zniwalla@gmail.com** ikke din? [Skift konto](#)

Introduction

This is the interview guide and instruction set for participating in experiments concerned with Deep Interactive 3D Modelling (DeepIE3D).


Experiments:

1. Create Convincing 3D Adaptions from 2D Images: Planes
2. DeepIE3D as a Tool for Creative Processes

Preliminary

Before performing the first experiment, please read the user guide on the website which can be accessed with the link at the top left of the page.

All the experiments will state a number of iterations. An iteration is a round in which you will pick the 3D model(s) you wish to evolve. The iteration ends and a new begins when the 'INITIALIZE EVOLUTION' button is clicked.

Taking a screenshot: During the experiments, you will be asked to take screenshots of the generated 3D models. When doing so, please use the 'Download .png file (take a picture)' feature found by clicking  next to a model. Please take the screenshot with the model in a position as close to its original position as possible. The desired position can also be seen in Figure 1.



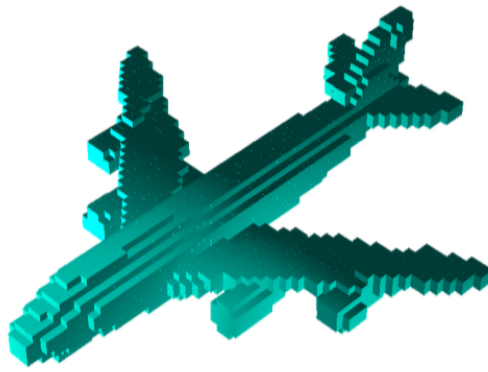


Figure 1: Optimal screenshot position.

During any task, please keep track of the following:

- At the initial iteration take a screenshot. This is the initial current best.
- At every iteration, after the initial one, consider if a model in the current iteration is more satisfactory than your current best. If so, screenshot it and refer to it as your new current best.
- Take a screenshot of the best 3D model from the final iteration.
- Which model was your favorite? (In which iteration did it appear)
- How many iterations were needed before the 3D models became satisfactory? (If they ever did)

You should not refresh the page during a task, as the iteration number will reset.

You have to refresh the the page when starting a new task to reset the iteration number.



Deep Interactive Evolutionary 3D Modelling: Planes

Navnet og billedet, der er knyttet til din Google-konto, registreres, når du uploader filer og indsender denne formular. Er zniwalla@gmail.com ikke din? [Skift konto](#)

*Skal udfyldes

Experiment 1: Part 1

<https://adrianwesth.dk>

Create Convincing 3D Adaptions from 2D Images: Planes

In this experiment, the objective is to recreate a plane from a 2D image as a 3D model. The 3D model will be created using DeepIE3D, which can be accessed with the link at the top of the page.

The experiment is divided into two parts: One using 'Normal mode' and one using 'Advanced mode'



(a) Plane 1



(b) Plane 2

Figure 2: Experiment: Planes

Part 1: Normal Mode

In this part, all tasks should be performed in 'Normal mode'.



Task 1: Plane 1

Recreate the plane seen in Figure 2a as a 3D model. Use 10 iterations.

Which iteration produced the best 3D model? *

Vælg 

On a scale from 1 to 10, how satisfactory was the best 3D model? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory

Very satisfactory

On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory

Very satisfactory

Please upload screenshot of best 3D model *

[TILFØJ FIL](#)


Please upload screenshot of best 3D model from final iteration *

[TILFØJ FIL](#)

Task 2: Plane 2

Recreate the plane seen in Figure 2b as a 3D model. Use 10 iterations.

Which iteration produced the best 3D-model? *

Vælg 

On a scale from 1 to 10, how satisfactory was the best 3D model? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory

Very satisfactory

On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory

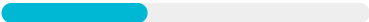
Very satisfactory

Please upload screenshot of best 3D model *

[TILFØJ FIL](#)

Please upload screenshot of best 3D model from final iteration *

[TILFØJ FIL](#)

 Side 2 af 5

TILBAGE

NÆSTE

Indsend aldrig adgangskoder via Google Analyse.

Dette indhold er hverken oprettet eller godkendt af Google. [Rapportér misbrug](#) - [Servicevilkår](#)

Google Analyse





Deep Interactive Evolutionary 3D Modelling: Planes

Navnet og billedet, der er knyttet til din Google-konto, registreres, når du uploader filer og indsender denne formular. Er zniwalla@gmail.com ikke din? [Skift konto](#)

*Skal udfyldes

Experiment 1: Part 2

<https://adrianwesth.dk>

Create Convincing 3D Adaptions from 2D Images: Planes

In this experiment, the objective is to recreate a plane from a 2D image as a 3D model. The 3D model will be created using DeepIE3D, which can be accessed with the link at the top of the page.

The experiment is divided into two parts: One using 'Normal mode' and one using 'Advanced mode'



(a) Plane 1



(b) Plane 2

Figure 2: Experiment: Planes

Part 2: Advanced Mode

In this part, all tasks should be performed in 'Advanced mode'.



Task 1: Plane 1

Recreate the plane seen in Figure 2a as a 3D model. Use 10 iterations.

Which iteration produced the best 3D model? *

Vælg 

On a scale from 1 to 10, how satisfactory was the best 3D model? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory Very satisfactory

On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory Very satisfactory

Please upload screenshot of best 3D model *

[TILFØJ FIL](#)

Please upload screenshot of best 3D model from final iteration *

[TILFØJ FIL](#)

Task 2: Plane 2

Recreate the plane seen in Figure 2b as a 3D model. Use 10 iterations.

Which iteration produced the best 3D-model? *

Vælg 

On a scale from 1 to 10, how satisfactory was the best 3D model? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory

Very satisfactory

On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory


Very satisfactory

Please upload screenshot of best 3D model *

[TILFØJ FIL](#)

Please upload screenshot of best 3D model from final iteration *

[TILFØJ FIL](#)

 Side 3 af 5

TILBAGE

NÆSTE

Indsend aldrig adgangskoder via Google Analyse.

Dette indhold er hverken oprettet eller godkendt af Google. [Rapportér misbrug](#) - [Servicevilkår](#)

Google Analyse





Deep Interactive Evolutionary 3D Modelling: Planes

Navnet og billedet, der er knyttet til din Google-konto, registreres, når du uploader filer og indsender denne formular. Er zniwalla@gmail.com ikke din? [Skift konto](#)

*Skal udfyldes

Experiment 2

<https://adrianwesth.dk>

DeepIE3D as a Tool for Creative Processes

In this experiment, the objective is to create a 3D model of an imaginary plane. The plane should be created entirely to your liking. The 3D model will be created using DeepIE3D, which can be accessed with the link at the top of the page.

The purpose of the experiment is to evaluate the DeepIE3D framework as a creative tool.

Task 1

This task should be performed in 'Normal mode'.

Create a plane you find interesting. This task requires that you have not pictured how the final product will look. In each iteration you should pick models with appealing features. Use 10 iterations.

Which iteration produced the best 3D model? *

Vælg

On a scale from 1 to 10, how satisfactory was the best 3D model? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory Very satisfactory



On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory Very satisfactory

Please upload screenshot of best 3D model *

TILFØJ FIL

Please upload screenshot of best 3D model from final iteration *

TILFØJ FIL

Task 2

This task should be performed in 'Advanced mode'

Create a plane you find interesting. This task requires that you have not pictured how the final product will look. In each iteration you should pick models with appealing features. Use 10 iterations.

Which iteration produced the best 3D-model? *

Vælg ▼

On a scale from 1 to 10, how satisfactory was the best 3D model? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory Very satisfactory

On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration? *

1 2 3 4 5 6 7 8 9 10

Very unsatisfactory Very satisfactory



Please upload screenshot of best 3D model *

TILFØJ FIL

Please upload screenshot of best 3D model from final iteration *

TILFØJ FIL

 Side 4 af 5

TILBAGE

NÆSTE

Indsend aldrig adgangskoder via Google Analyse.

Dette indhold er hverken oprettet eller godkendt af Google. [Rapportér misbrug](#) - [Servicevilkår](#)

Google Analyse





Deep Interactive Evolutionary 3D Modelling: Planes

Navnet og billedet, der er knyttet til din Google-konto, registreres, når du uploader filer og indsender denne formular. Er zniwalla@gmail.com ikke din? [Skift konto](#)

*Skal udfyldes

General Questions

How was the overall user experience? *

	1	2	3	4	5	6	7	8	9	10	
Exhausting	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Fun

How would you describe the ease of using the web app? *

	1	2	3	4	5	6	7	8	9	10	
Hard	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Easy

What strategy did you use to create the models in experiment 1? (What dictated which models you picked?) *

Dit svar

What strategy did you use to create the models in experiment 2? (What dictated which models you picked?) *

Dit svar

Did the user guide cover all features of the web application? If not, which were unsatisfactory and why? *

Dit svar



Is the 'Normal mode' intuitive? Is it hard to use? *

Dit svar

Is the 'Advanced mode' intuitive? Is it hard to use? *

Dit svar

Which features is the web application missing? (If any) *

Dit svar

 Side 5 af 5

TILBAGE

SEND

Indsend aldrig adgangskoder via Google Analyse.

Dette indhold er hverken oprettet eller godkendt af Google. [Rapportér misbrug](#) - [Servicevilkår](#)

Google Analyse



D. User Testing: Answers

The following pages are the answers from the questionnaire used in the user testing of DeepIE3D. Some pages are distorted, but this is due to Google Forms - the relevant data can be seen as graphs in Section [7.2](#) and the written answers are luckily not distorted.



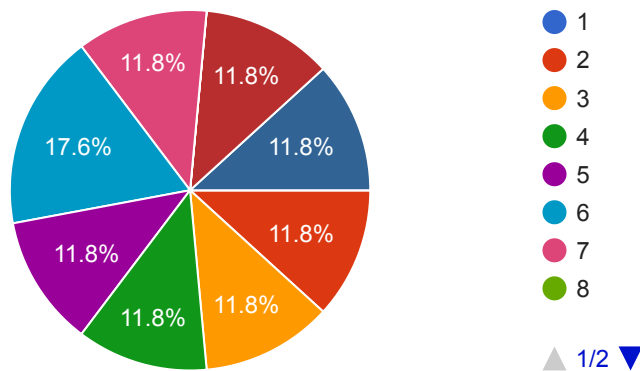
Deep Interactive Evolutionary 3D Modelling: Planes

17 responses

Experiment 1: Part 1

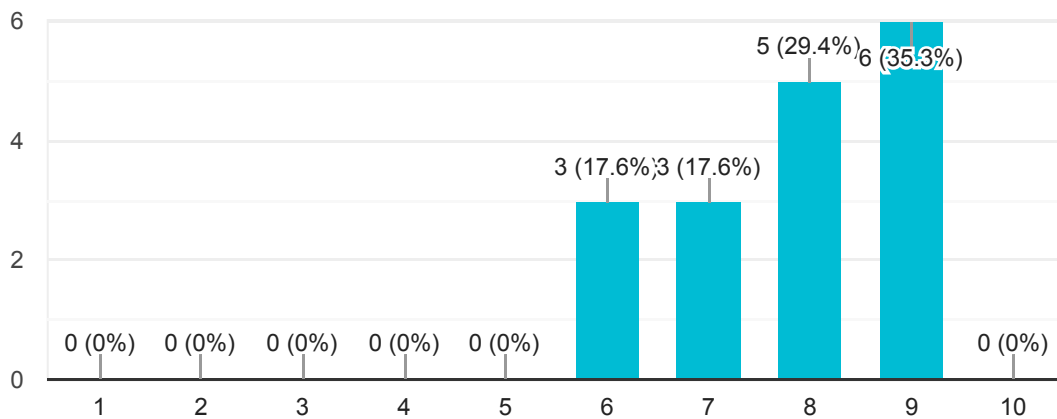
Which iteration produced the best 3D model?

17 responses



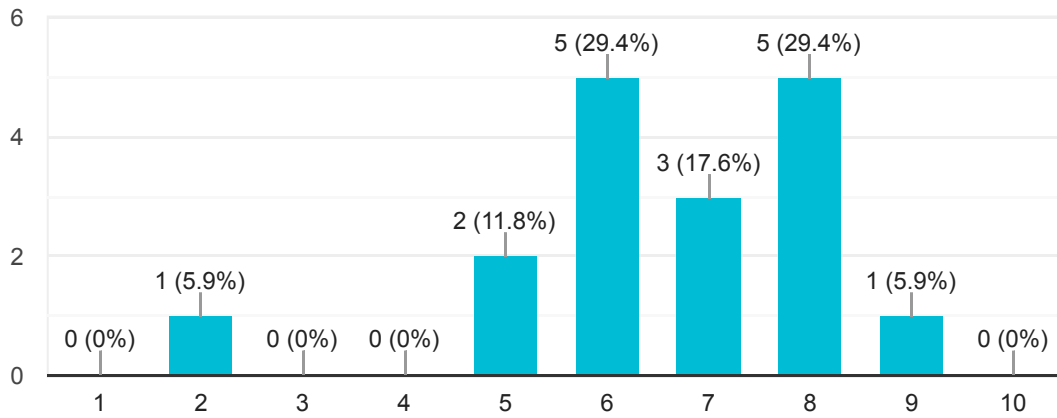
On a scale from 1 to 10, how satisfactory was the best 3D model?

17 responses



On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration?

17 responses



Please upload screenshot of best 3D model

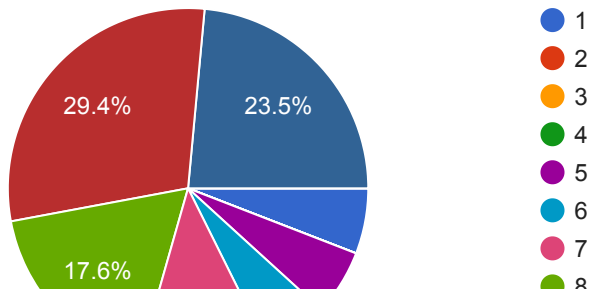
17 responses

Please upload screenshot of best 3D model from final iteration

17 responses

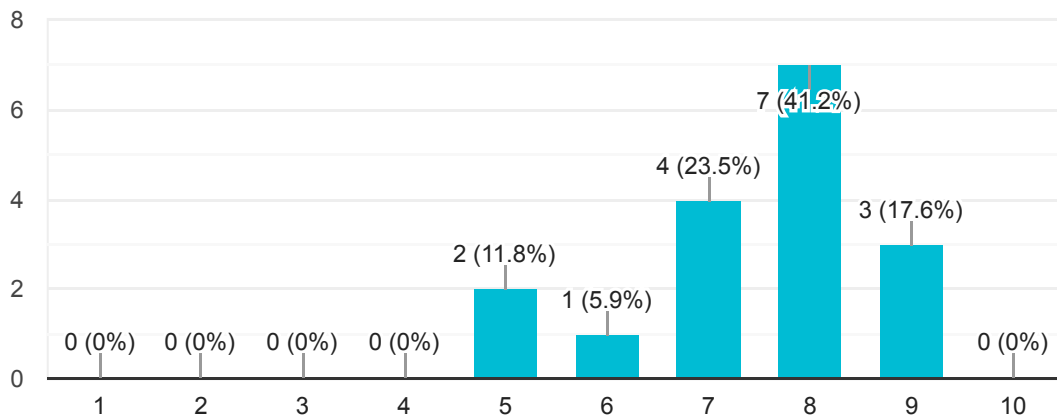
Which iteration produced the best 3D-model?

17 responses



On a scale from 1 to 10, how satisfactory was the best 3D model?

17 responses



On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration?

17 responses

Please upload screenshot of best 3D model

17 responses

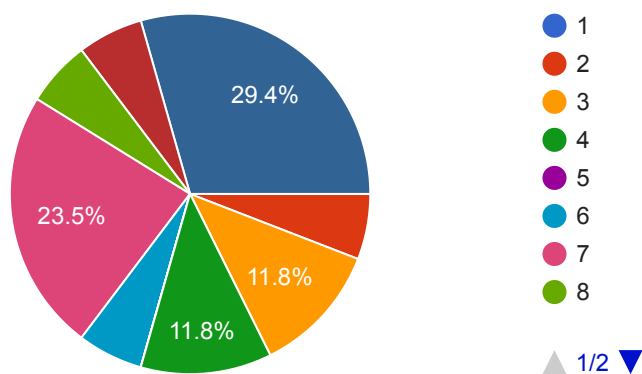
Please upload screenshot of best 3D model from final iteration

17 responses

Experiment 1: Part 2

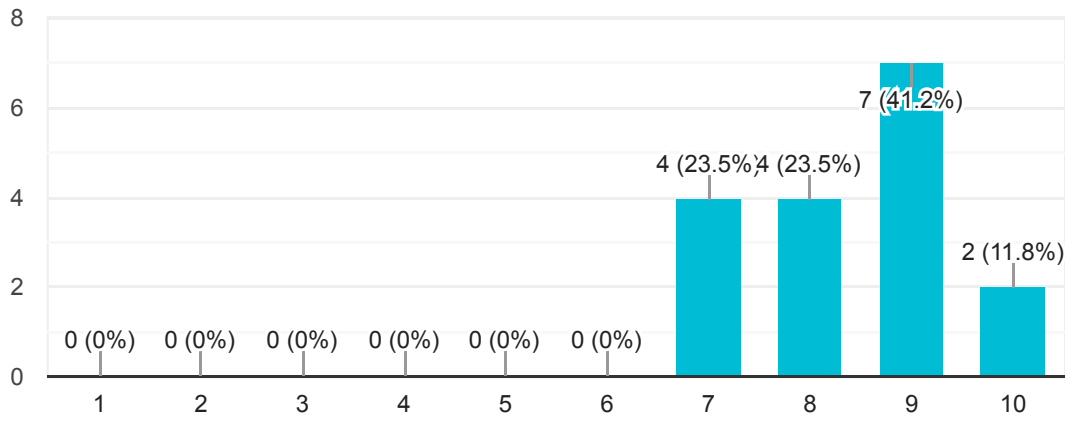
Which iteration produced the best 3D model?

17 responses



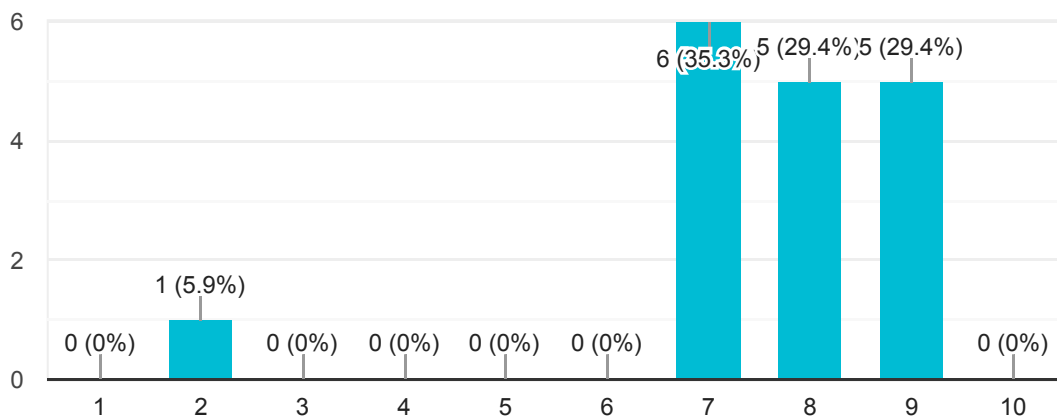
On a scale from 1 to 10, how satisfactory was the best 3D model?

17 responses



On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration?

17 responses



Please upload screenshot of best 3D model

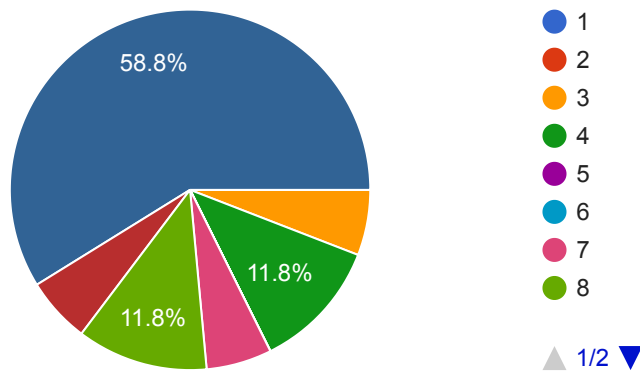
17 responses

Please upload screenshot of best 3D model from final iteration

17 responses

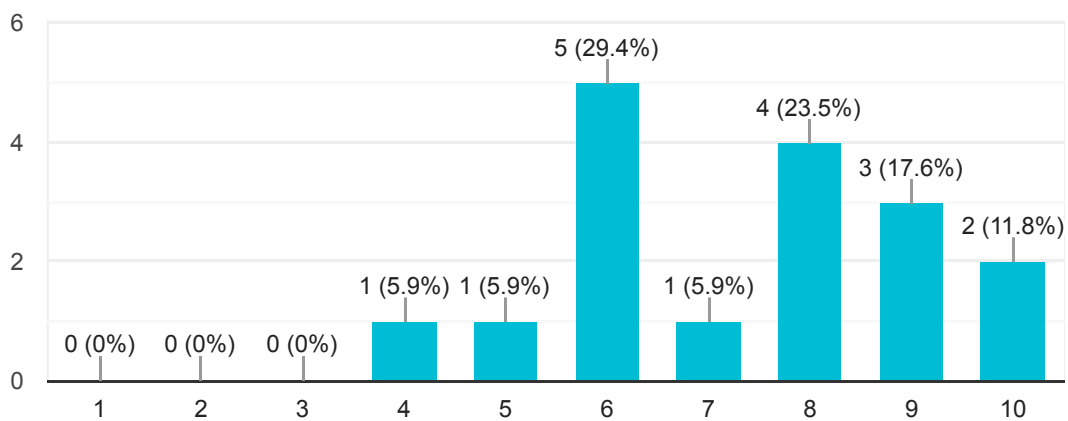
Which iteration produced the best 3D-model?

17 responses



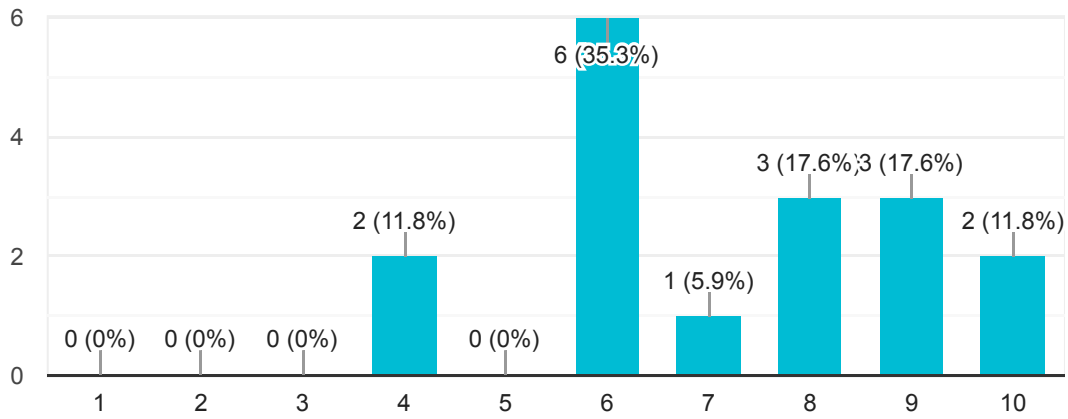
On a scale from 1 to 10, how satisfactory was the best 3D model?

17 responses



On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration?

17 responses



Please upload screenshot of best 3D model

17 responses

Please upload screenshot of best 3D model from final iteration

17 responses

Experiment 2

Which iteration produced the best 3D model?

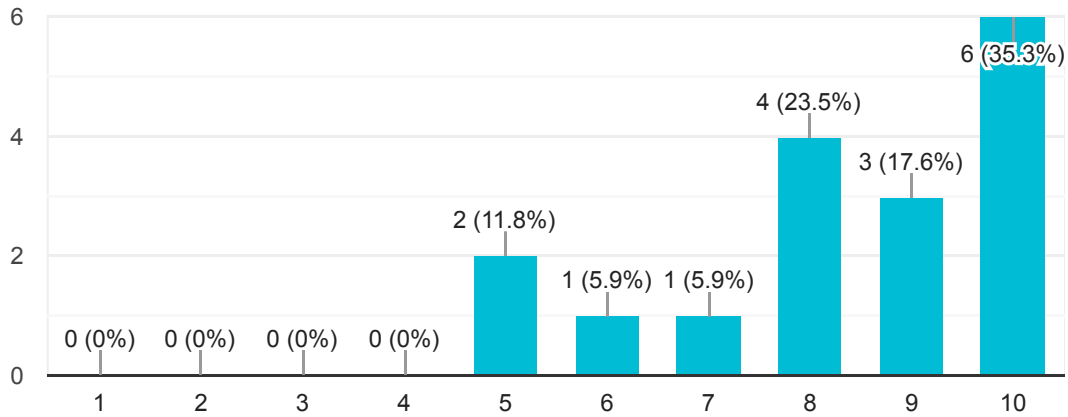
17 responses



- 1
- 2
- 3

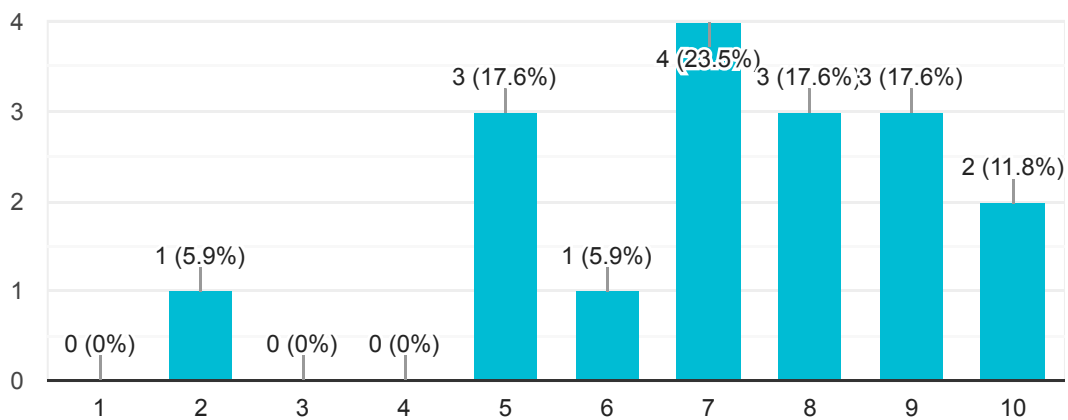
On a scale from 1 to 10, how satisfactory was the best 3D model?

17 responses



On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration?

17 responses



Please upload screenshot of best 3D model

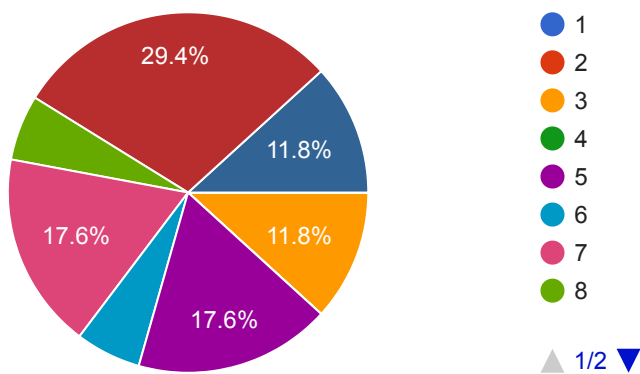
17 responses

Please upload screenshot of best 3D model from final iteration

17 responses

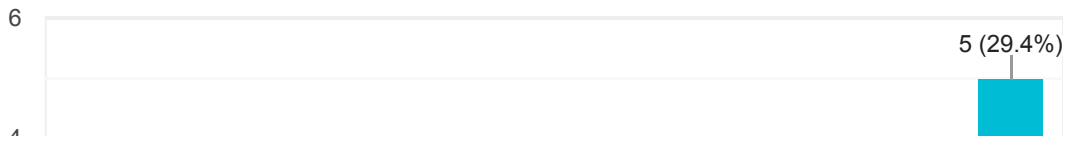
Which iteration produced the best 3D-model?

17 responses



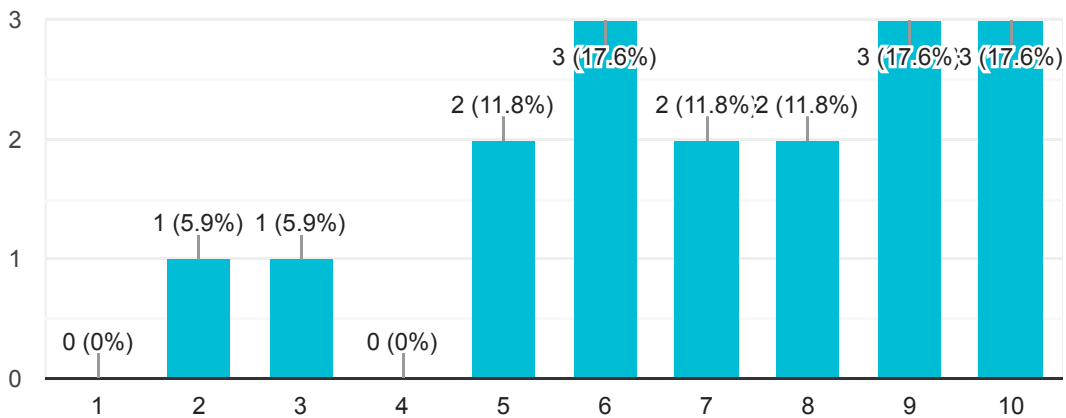
On a scale from 1 to 10, how satisfactory was the best 3D model?

17 responses



On a scale from 1 to 10, how satisfactory was the best 3D model in the last iteration?

17 responses



Please upload screenshot of best 3D model

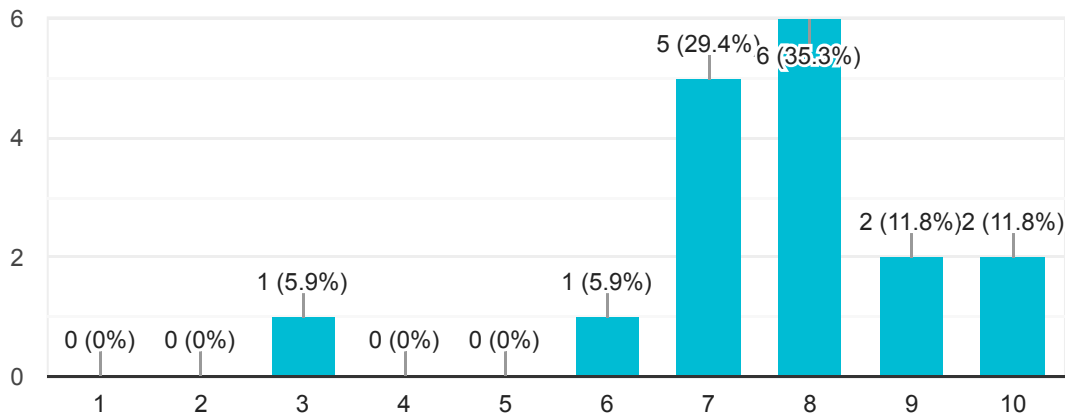
17 responses

Please upload screenshot of best 3D model from final iteration

17 responses

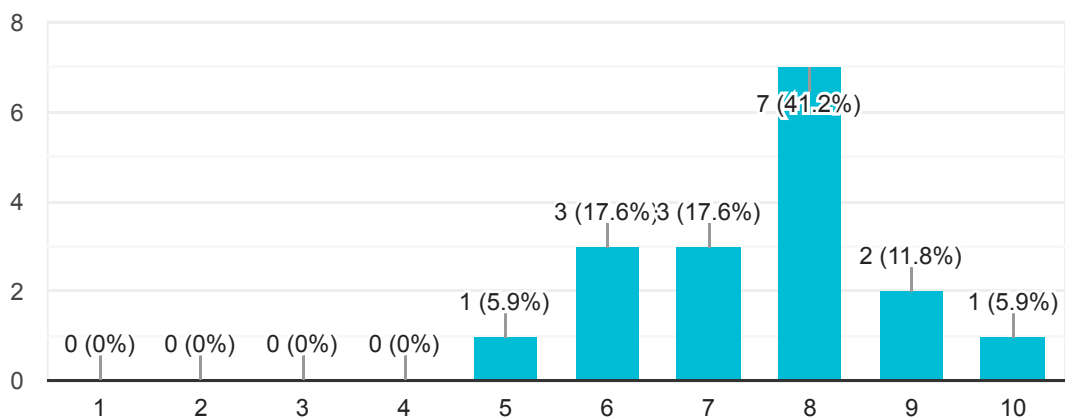
How was the overall user experience?

17 responses



How would you describe the ease of using the web app?

17 responses



What strategy did you use to create the models in experiment 1? (What dictated which models you picked?)

17 responses

Chose the ones that looked the most in regards to different features (pointy nose of the plane, edgy wings, motor-location)

I picked the ones that looked most like the pictures, but choose several ones if they were equally good

I chose the models that were most similar to the picture. For an example I chose the models that had some kind of propella (plane 2) or had motors in the back (plane 1)

Hearted the one that looked the most like the result I wanted. If none looked like the goal I didn't choose any. In advanced mode I locked the plane.

Whatever looked most like the pictures. Especially distinct features as wings and propels

I looked at specific features for the plane (pointy front, jet motors at the back of the plane etc.)

Clicked on the plane that resembled the most

Using hearts

Feature similiary, eg. wing/engine position and size

Looked for Engine location, engine shape, front propeller and width of wings. in normal mode "liked" all planes with these features. in advanced, tried to mutatae planes that had some of the different features, to combine them to a plane that had all features.

Picked the ones which had some resemblance with target

First few iterations, pick the plane that looked the most like the desired plane. When options to choose from chose the ones with desired features.

What looked most like a real plane

I just tried something out and chose the models that resembled the model the most.

Tried to improve the chosen planes

I picked the once that looked most like the plain

Choose the ones resembling the most

What strategy did you use to create the models in experiment 2? (What dictated which models you picked?)

17 responses

Chose the ones that looked the most action packed.

I lways kept the best one and mutated it, but also crossed it if the others had features that good make it more correct

If it was most similar to an airbus or a very thin plane.

I just went for my gut feeling. I like glitchy aesthetics.

See above.

Kind of the same one - looking for specific features

Clicked on the plane that resembled the most

mutating and crossovers

Feature similar, eg. wing/engine position and size

In normal mode I "Liked" planes that had different interesting features, to create a plane with a combination of these. For advanced I tried to mutate planes with many features, as well as crossover planes that had multiple interesting features, to make a combination.

Mutated 1-3 which looked promising

Tried to produce weird planes. Picked the weird ones and mutated them a lot.

What looked most like a real plane

I went with the flow and tried to get inspired while sticking to my creative instinct.

It should be able to fly!

I liked to combine planes where I didn't know how the result would be

Should look like a plane - the wings are a must

Did the user guide cover all features of the web application? If not, which were unsatisfactory and why?

17 responses

Yes

The user guide was a bit hard to follow

It did, but the language was difficult

Yes.

Didn't read the user guide.

It did, though it might require some re-wording as some of it was quite hard to wrap your head around

It was pretty fulfilling, yes

Could use examples/pictures

It did

Might have been a bit more detailed, but then it may have been a bit too long

I think so.

Dont know

I think it covered all.

-

I did not read it

Is the 'Normal mode' intuitive? Is it hard to use?

17 responses

Intuitive

Easy to use, very intuitive!

Without an oral introduction no..

Yes it's intuitive.

It's pretty easy to use when you know what to do, but there is some clutter and a lot of buttons I never used.

Very easy

I would say its pretty intuitive and not very hard to use

Very intuitive

It is simple and intuitive.

I think it is very intuitive

You do not really know what is happening behind the scenes. It is faster to use, as you dont have to decide on many different options.

no

it is intuitive.

Yes

It was simple

Yes, logical

Is the 'Advanced mode' intuitive? Is it hard to use?

17 responses

A bit harder to use. Had to think for longer over my choices.

Yes, after I read the user guide it was okay

It's kind of hard to use - it's difficult to figure out how much you can mutate.

I don't find advanced mode very intuitive but it helped that I had used it before.

A bit less easy, but after one try it's easy enough

This mode is a little harder to understand because it has more features but its not hard per se

Intuitive

a little harder to use, but still good.

Crossover requires lots of extra keypresses compared to everything else

Also pretty intuitive

After some iterations it became fine to use

It was a little hard the first time. Second and third it provided a lot of info of what was happening. Took a little longer.

A little more advanced

A little harder but still pretty intuitive - especially after having tried the normal mode.

Yeah

It was a little challenging

Yes, fine symbols

Which features is the web application missing? (If any)

17 responses

I don't know

A language not made for computer-scientists!!

A dummy guide.

3D model print out of the USB-port.

Colors, picture without background, ?????

It works pretty well overall, i wouldn't know what's missing

Dunno

It would be better if the mutations would start from 1 instead of 9, so if you lock a figure you know it will be the first model in the next iteration.

Keybindings

None that I can think of.

Could I model something else than planes?

The keep model feature in the advanced feature is very useful when recreating planes, but it is not present in the normal mode.

Advanced mode clearer description of next iteration. Maybe the planes of crossover

Would it be possible to have a picture continuously of ones favorite?

The final plane should be finished somehow

I would not know

Nope

This content is neither created nor endorsed by Google. [Report Abuse](#) - [Terms of Service](#)

Google Forms

E. Network Gradients

Gradients and gradient penalty throughout the training process of the final PacWGAN-GP2 used in DeepIE3D.

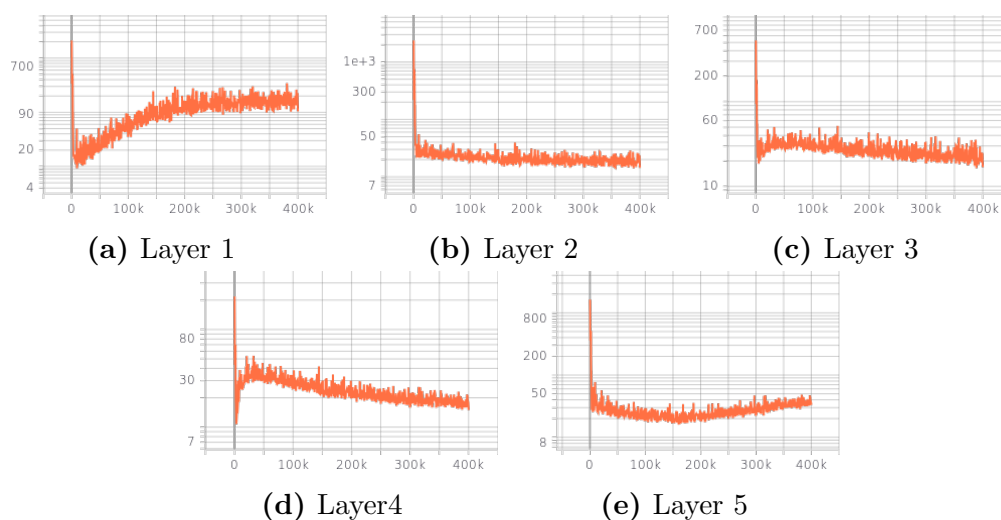


Figure E.1: Gradients for the layers of the critic.

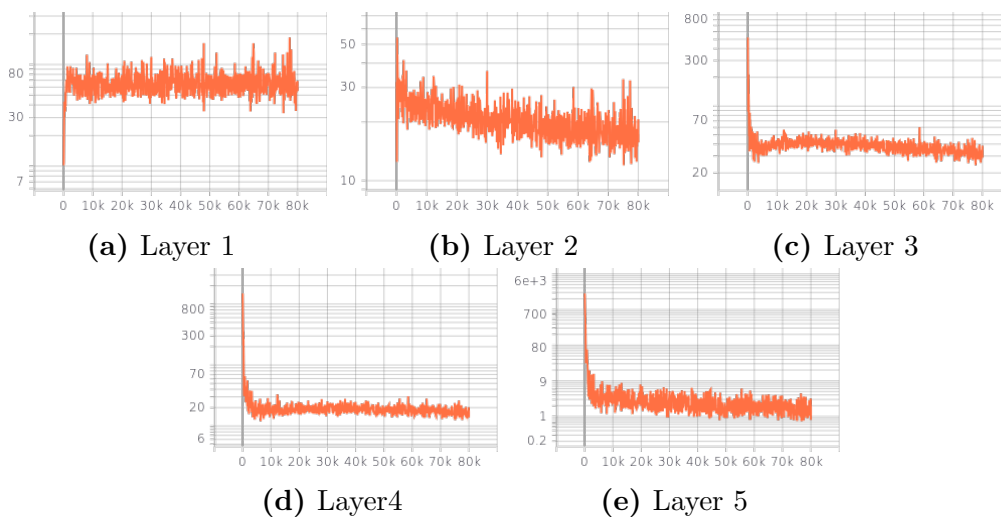


Figure E.2: Gradients for the layers of the generator.

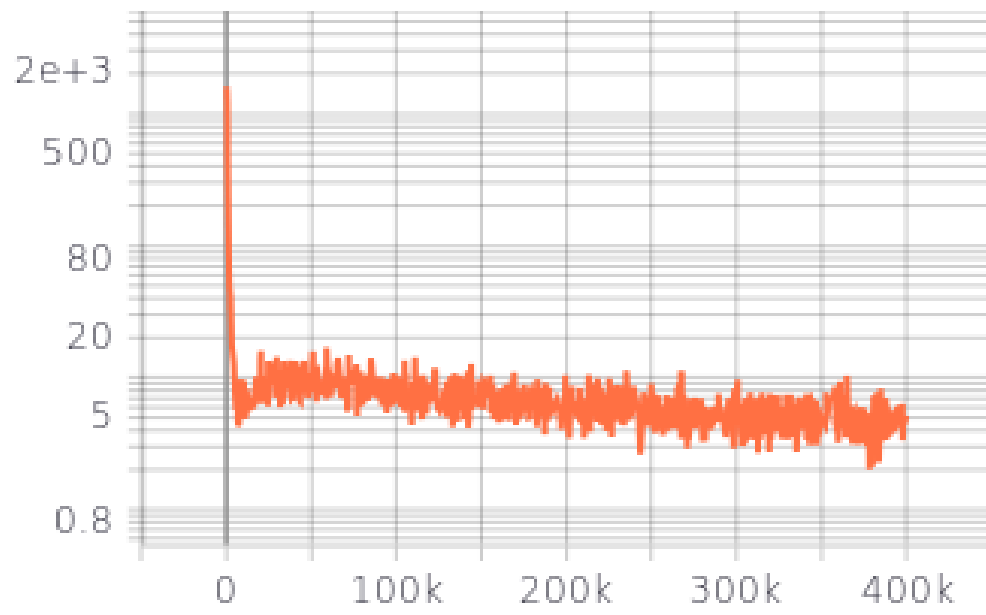


Figure E.3: Gradient penalty