EG3D Reproduction Group 67

Indy Dekker (5419018)
TU Delft
i.dekker@tudelft.nl

Sebastian Fernandez Ruiz de las Cuevas (5947626)
TU Delft
sfernandezruiz@tudelft.nl

Daniel Soler (6080758)
TU Delft
dsoleralvarezm@tudelft.nl

1 Introduction

In this report, we will reproduce the results from the paper "Efficient Geometry-aware 3D Generative Adversarial Networks" [1]. This paper describes improvements to the computational efficiency and image quality of 3D GANs.

This paper aims to reproduce the results from [1] and further study the characteristics of the proposed solution. This study consists of three parts. First, we look at the possibility of editing and controlling features in the generated images. Secondly, we will study the behaviour of the backgrounds in output images. Finally, we will look if it is possible to just render eyes instead of whole faces.

2 Figure Reproduction

The first part of the project is to reproduce the figures of [1] the first step is to set up a Python environment. To ensure all dependencies are configured right the repository contains a .yml-file. Running it with Conda will create the environment. To reproduce the images we can reuse the pretrained models which are made available by the authors. To reproduce figure 6 you use the model named "afhqcats512-128.pkl" and to reproduce the image containing the humans we use the model "ffhq512-128.pkl".

An important note with the creation of the environment is that we had to take to specific steps which were not mentioned in the readme.md file of the Github repository. Firstly, we need to install the CUDA toolkit separately within the created environment, this despite the fact that this should happen when creating the environment. To do this, run the following command: `conda install -c nvidia cuda`. In addition we need to set `CUDA_HOME` to the specific folder where the toolkit is installed.

Now it is possible to run the model and generate images with it. The commands to do so are provided in the readme file. The results of the gen_samples.py runs (for both cats and faces) are depicted in images 1 and 2

3 Feature Manipulation

This section covers the procedure to gain closer control over the features of an image. The first step is to form an idea of how the pictures are generated in the first place. Given the General Adversarial

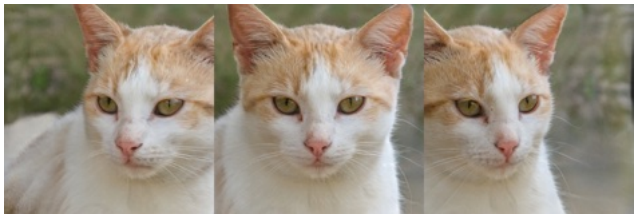

Figure 1: Resulting image of running the afhqcats512-128.pkl model

Figure 2: Resulting image of running the ffhq512-128.pkl model

Network (GAN) architecture is used this already gives us one piece of important information, namely, that there is a generative and a discriminating network.

Besides, we know that in the run command for the `gen_samples.py` file we have to input the value of a random seed. This value corresponds to a certain image being generated. When we combine these two givens we can locate a tensor that is generated named `z`. This tensor contains a set of random values that serve as an input to the network. As a result, the network synthesizes an image.

In this section, we study how altering the random tensor `z` changes the synthesized image. The goal is to be able to change specific features of the generated faces by manipulating `z`.

The experiment is divided into three parts. First, we apply a brute-force method to alter all the parameters of `z` separately with a significant amount to see how our baseline image 3 changes. We then pick a number of features and apply smaller changes to these specific z-values, for the same baseline person. Lastly, we apply the same changes but now for faces corresponding to different random seeds to see if we can find a generalization in how the output changes for altering the parameter in the z-tensor.

Figure 3: Baseline image for random seed = 5

3.1 Brute-force approach for altering z

The baseline image (figure 3) corresponds to running the `gen_samples.py` file using a value of 5 for the random seed (using the ffhq512-128.pkl model). The tensor `z` has a shape of [1, 512]. Hence, there are 512 parameters in z that we can work with.

To perform this experiment we add a loop in the `gen_samples.py`. For every iteration we make of the initial z-vector (based on the random seed) and subtract 20 from one of the 512 parameters. The result of running this results in a set of 512 random images. one example can be found in figure 4

Figure 4: Example of a resulting image after altering the z-tensor. In this case, we altered z[0][9]

.

The goal is to get an overview of which parameters control certain features in the images. As for the example in figure 4, we see that the facial expression stays more or less the same, however, the hair colour is changed. This might be an indication of the function of the specific parameter in `z`. Going over all the images we defined several interesting parameters, these parameters are examined in more detail as discussed in the following section.

3.2 Researching parameter functions

We now have a list of interesting parameters, the next step is to see if we can use these to manipulate specific features. We tried to alter the following characteristics of the baseline image:

- Hair colour
- Gender
- facial expression
- glasses on/off

To get more insights into what the parameters exactly do we synthesize multiple images where we alter the specific parameter in `z` with one of the following values: [-10, -5, -1, 1, 5, 10]. All the generated images can be found in appendix A.

One conclusion we can draw from this experiment is that larger changes in a parameter lead to larger changes in the output images. However, it seems like single parameters do not influence specific attributes of the faces. Changes in parameters seem to change multiple features of the faces at once. For example, if we look at 5 we see how besides the change of hair colour (when going in the negative direction) the background and glasses also change.

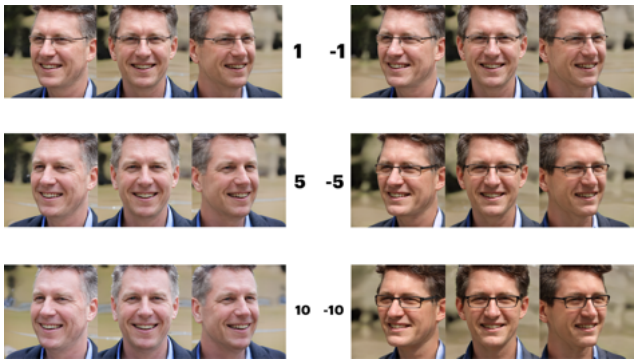

Figure 5: Altering parameter z[0][9] in more detail

.

Given the results, we can conclude that it is possible to change the features of the faces by altering the z-tensor. However, it is difficult to control specific traits of the generated images. We found that `z[0][9]`, when a value is subtracted, leads to a brown hair colour for our baseline image. Altering `z[0][30]` by adding 10 will lead to a change in gender. In addition, subtracting a value of 10 from `z[0][100` will lead to an image where the glasses disappear.

3.3 studying generalization of parameter alterations

So far we looked only at the image generated for random seed 5. In this section, we look at how the parameters influence faces generated with different random seeds. All the generated are, again, included in appendix A.

We run the same experiment as before, but we also check how extreme parameter changes impact the results by adding and subtracting the value 200. We run the test for the random seeds [4, 18, 148, 1812]. Due to time constraints, we only looked at the parameters [30, 102, 302].

An interesting finding is that altering a parameter with a high value like 200 or -200. Leads to the same output image for all tried random seeds. The reason this is significant is because a random seed means that z is completely different. So, we see how the influence of a parameter grows when its value increases/decreases significantly.

Figure 6 shows the result for a single random seed. We see how the generated faces morph towards the images for the values of -200 and 200. Other random seeds lead to the same faces being synthesized as the ones at the bottom of the figure.

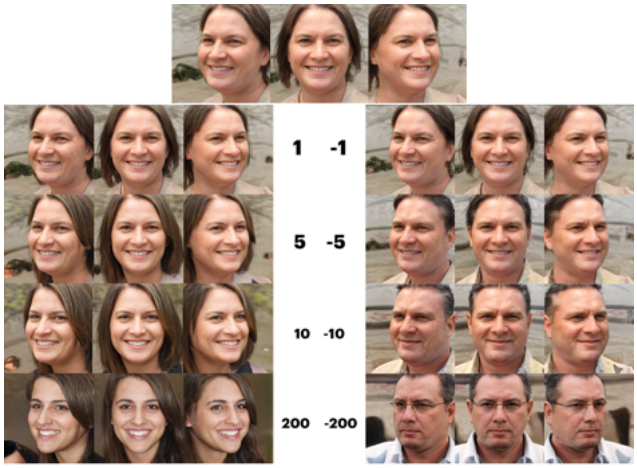

Figure 6: Altering parameter z[0][30] for random seed = 4.

.

In this experiment, we looked at three different parameters and what we see is how the pair to which the model converges differs for the parameters. The pair on every occasion consists of a woman and a man. With this knowledge, it is possible to better understand the model's behaviour because we now have an idea of which images the output is morphed when changing the specific parameter.

In conclusion, we found that the parameters in z do generalize. Whenever a single parameter is significantly higher or lower than the others we see that the resulting output images converge for multiple random seeds. These characteristics of the model can be utilized to alter specific features of the faces.

4 Background Analysis

As the paper explicitly mentions at the discussion, there is no "explicit background handling" meaning that the scene is modeled as a single 3D model. As a consequence the background is rendered as a single object with all the objects fused together, which is clear from the depth maps shown in the original paper (e.g figure 11 from [1]).

This comes with a series of problems as the authors themselves recognize. Firstly, without any modeling of the background it is left to the network to learn about it from the training dataset. Further, FFHQ, the dataset used for face modeling is predominately filled by front-facing subjects where the information about the background is limited, which means that the ability to generalize and generate accurate background information is hindered. Moreover, given the fact that the background is further from the camera than the face, it is more prone to present larger

Figure 7: Example of face yaw rotation with different backgrounds

feature changes for a give change in yaw angle, adding further difficulty to the task of modeling it, as seen on figure 7. In fact figure 5 of [1] shows that for steep yaw angles the background fills with solid colors as there is no detailed information on what to generate.

Furthermore we see that the 3D reconstruction tools use such as COLMAP give preference to the foreground object, as seen on figure 2 of the supplemental material of [1], where the point cloud of the background is less dense than for the face. In fact the tri-plane hybrid 3D representation method used relays heavily on feature extraction to accurately project a 2D image into 3D space. After a feature map is created and projected in 3D, a small MLP is used to interpret features as color and density. Therefore, as the faces take up more space of the image and have more recognizable feature, their location will be more accurately described than the low feature density background. Although this novel hybrid approach proposed on the paper works well for the foreground and reduces computational cost, it does not work very well for the background.

A possible solution would be to model the foreground and background separately like done in [2], or even go a step further and model each background object individually like [3] and [4]. This approach is in fact computationally more expensive than [1]. However it gives better results for the background. By modeling each background object individually a more detailed depth map could be generated and thus more features would be present that would result in higher quality representation.

5 Shape Artefacts

In this section we attempt to better understand the structure of the rendered data in order to isolate those regions in which artifacts are usually found. To demonstrate the challenges associated with this we focus on rendering the eyes and, later, we discuss the reason be-

hind these artifacts and possible solutions.
In order to represent different areas of the face, in this case the one in plot 8, we use tresholding. The intuition behind this is that the rendered data, obtained through the marching cubes algorithm, should follows a distribution in which different values correspond to different facial features.

Figure 8: Initial render with random seed 9.

To explore this hypothesis we perform a basic analysis of the distribution: some slices are plotted to intuitively look for patterns and, also, a histogram in order to assess the order of magnitude of those different values is plotted. The slices, shown in plot 9, show how the values indeed represent the facial features that we wish to extract.

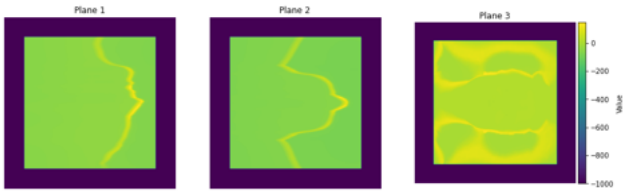

Figure 9: Slices of the marching cubes values.

These images have a significant padding added to them (purple areas) with value -1000 which are excluded from the histogram, shown in plot 10.

The histogram shows how most values are condensed into a narrow band of values, with a few data points extending to higher values. The hope here is that those higher values represent some of the facial structures that we wish to isolate from the rest of the data. After trial and error tresholding to get an idea of if that could be the case a K-Means clustering algorithm is ran on the data with the hope of isolating the different facial features. If the clustering algorithm found that each facial feature has a specific range of values we could easily treshold them and isolate a certain feature. The results of the clustering algorithm are shown on histograms in plot 11.

The first cluster clearly corresponds to the added padding (purple areas in figure 9). The second group only includes negative values that are not rendered.

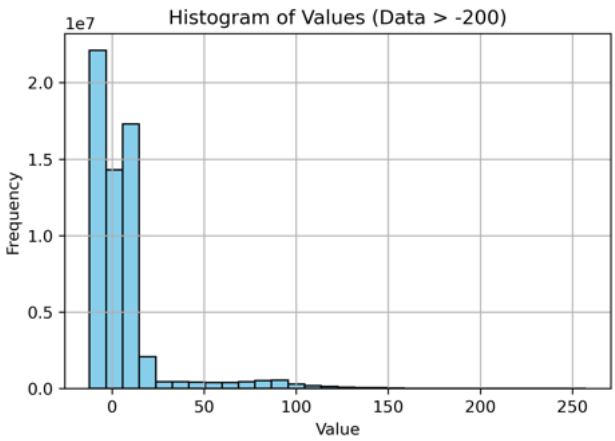

Figure 10: Histogram of marching cubes values.

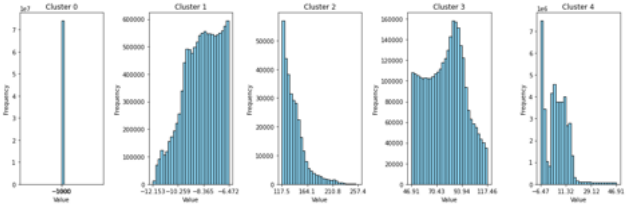

Figure 11: Histograms of the clustered values.

In order to find out if the rest clusters correspond to specific facial features, we render each one of them.

Cluster 2 Cluster 3 Cluster 4

Figure 12: Histograms for each cluster.

Here we can see two behaviors. First, we see that the hypothesis formulated that different facial features can be separated is true to some extent. Cluster 2 holds most of the information for the eyes, cheeks and mouth, as well as ears but has no data about the hair, neck or background. Cluster 3 has most of the forehead and hair as well as a lot of the neck and shoulders. Cluster 4 has mainly the neck and forehead. Therefore, performing clustering with a larger number of clusters could be able to further isolate each of the facial features. Nevertheless, this proves to be very computationally expensive for the complete dataset, and downsampling the data is recommended as a consequence. Another interesting approach to isolate this features would be to use location-informed clustering algorithms or other methods that retain a greater sense of spatial position. This methods are expected to be able to achieve better results than a regular clustering algorithm.

Second, we can see that most of the information

related to the eyes and teeth can be found in cluster 2. Consulting plot 11 shows how this cluster is the one with by far the least number of data points. We believe that this lack of information, and therefore of detail, is one of the driving forces of the artifacts observed in the eyes. If this were the case this issue could be solved by tweaking the settings of the eg3d algorithm, or by looking for another architecture that prioritizes these areas.

Another explanation of these artifacts could be that the issues related to progressive growing present in *StyleGAN* have not been completely addressed. As explained in [5] , the use that the *StyleGAN2* network makes of the so called "Progressive Growing" algorithm. As explained in [6] this method works by generating low resolution images and then progressively increasing the resolution of the images by adding layers to the networks. This process has been shown to stabilize the generation of high-quality images and is one of the cornerstones of *StyleGAN* success, but it also generates artifacts. This is because the progressively grown generator has a strong preference for location for details and with features such as eyes and teeth it fails to update them smoothly but rather it "jumps" when the next preferred location is reached. To solve this issue, a different architecture is proposed in [5], which retains the benefits of progressive growing without the problems associated with it. In such paper, they choose to use a skip generator and a residual discriminator instead of progressive growing and, in that manner, they are able to achieve good results whilst getting rid of these artifacts. These changes result in *StyleGAN 2* but, as claimed in the paper we are reproducing, these are still present. Therefore a plausible explanation is that these changes are unable to completely tackle the problem inherited from the first version of *StyleGAN*.

6 Conclusion

In this paper, we reproduced the results from the paper "Efficient geometry-aware 3D generative adversarial networks." Besides, we researched some flaws of the General Adversarial Network (GAN) that are proposed by the writers.

While reproducing the results we also opted to find ways to influence the features of output images. What we found is that by adjusting the input vector z it is possible to manipulate the output of the model. Each parameter of the vector converges to one of two images when the value is either significantly increased or decreased. The pair of images is different for each parameter in the vector z. This allows for more control over the output images and even allows the user to manipulate specific features of images.

A problem mentioned in the original paper is the background. The proposed model renders the scene as a single scene. So, the background is not handled explicitly, in the case of the FFHQ dataset (which does not contain a lot of information on backgrounds) this leads to problems with how the backgrounds of synthesized images are handled. A solution to this problem is the separate rendering of those backgrounds by a second network. However, this solution is computationally expensive.

Finally, the problem of rendering only a certain physical feature and assessing typical artifacts is discussed. Clustering the values shows some promise and position-informed methods which may be much more capable to isolate certain areas of the render are proposed. Then, the artifacts present on eyes are discussed. Observing the data of the render it can be seen how these areas have significantly fewer data points assigned to them compared to other facial features, which may be to blame for the artifacts. Another possibility is proposed, which is the fact that some of the problems associated with progressive growing in the first versions of *StyleGAN* are still present in the newer ones used in the *eg3d* algorithm. To solve it first the cause must be identified but the solution likely lies on architecture changes, such as those claimed to have fixed the progressive growing issues.

References

[1] Eric R. Chan et al. *Efficient geometry-aware 3D generative adversarial networks*. Apr. 2022. URL: https://arxiv.org/abs/2112.07945.

[2] Kai Zhang et al. "NeRF++: Analyzing and Improving Neural Radiance Fields". In: *CoRR* abs/2010.07492 (2020). arXiv: 2010.07492. URL: https://arxiv.org/abs/2010.07492.

[3] Michael Niemeyer and Andreas Geiger. "GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields". In: *CoRR* abs/2011.12100 (2020). arXiv: 2011.12100. URL: https://arxiv.org/abs/2011.12100.

[4] Michael Niemeyer and Andreas Geiger. "CAMPARI: Camera-Aware Decomposed Generative Neural Radiance Fields". In: *CoRR* abs/2103.17269 (2021). arXiv: 2103.17269. URL: https://arxiv.org/abs/2103.17269.

[5] Tero Karras et al. "Analyzing and Improving the Image Quality of StyleGAN". In: *arXiv preprint arXiv:1912.04958* (2019).

[6] Tero Karras et al. "Progressive growing of gans for improved quality, stability, and variation". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=Hk99zCeAb.

A Appendix A: Results of altering individual Z-parameters

A.1 Altering individual parameters of z

In this section, we show the results of synthesized images by the network when we alter specific parameters of the random seed vector z. For every one of the six iterations we take a copy of the original vector and add one of the values shown in each image (1,-1, 5, -5, 1-, -10). The pseudocode for this operation is the following:

```
entries = [-10,-5,-1, 1, 5, 10]
        parameter = [9, 28, 30, 51, 100, 102, 107, 244, 495]
        for i in parameter:
            for j in entries:
                z_copy = z.clone()
                z_copy[0][i] = z[0][i] + j
                synthesize image based on z_copy
```

the resulting images can be found in the figures below. The code is also available on a dedicated GitHub page https://github.com/idekker1/EG3D_reproduction_group67

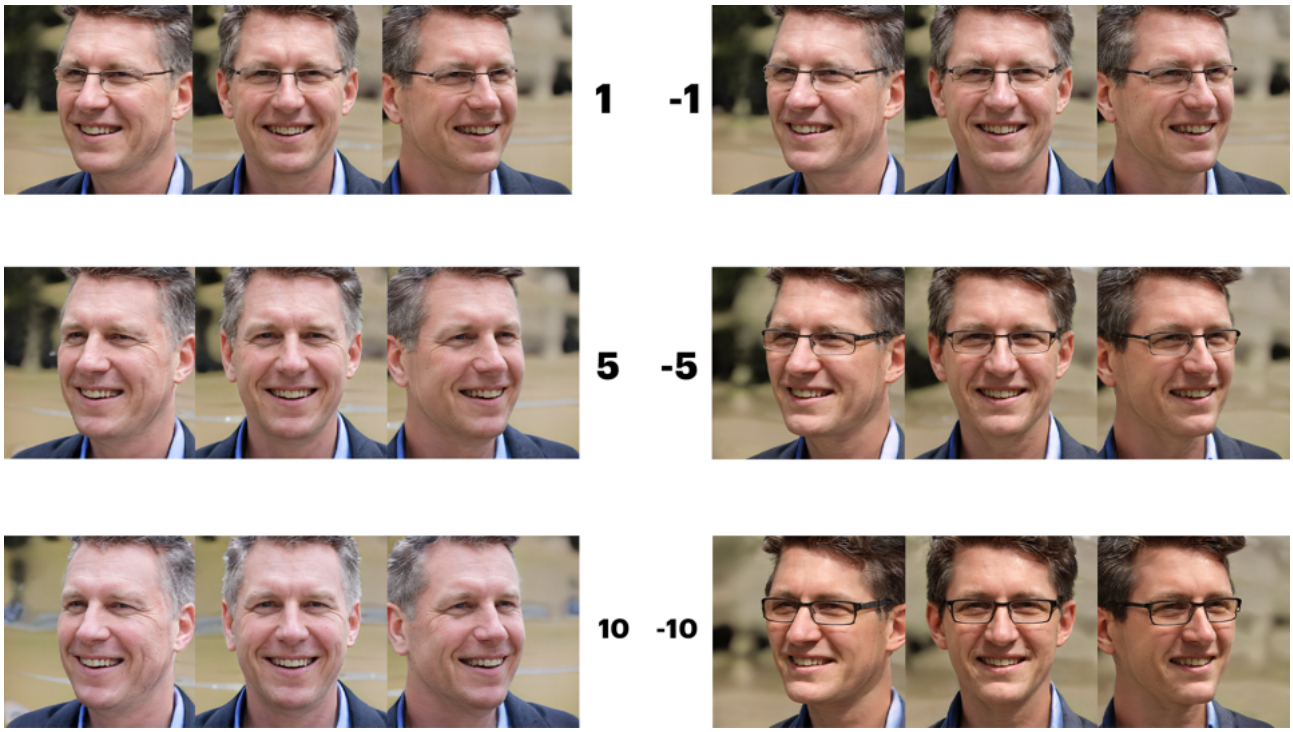

Figure A.1: Effects of altering z[0][9] on synthesized image.

.

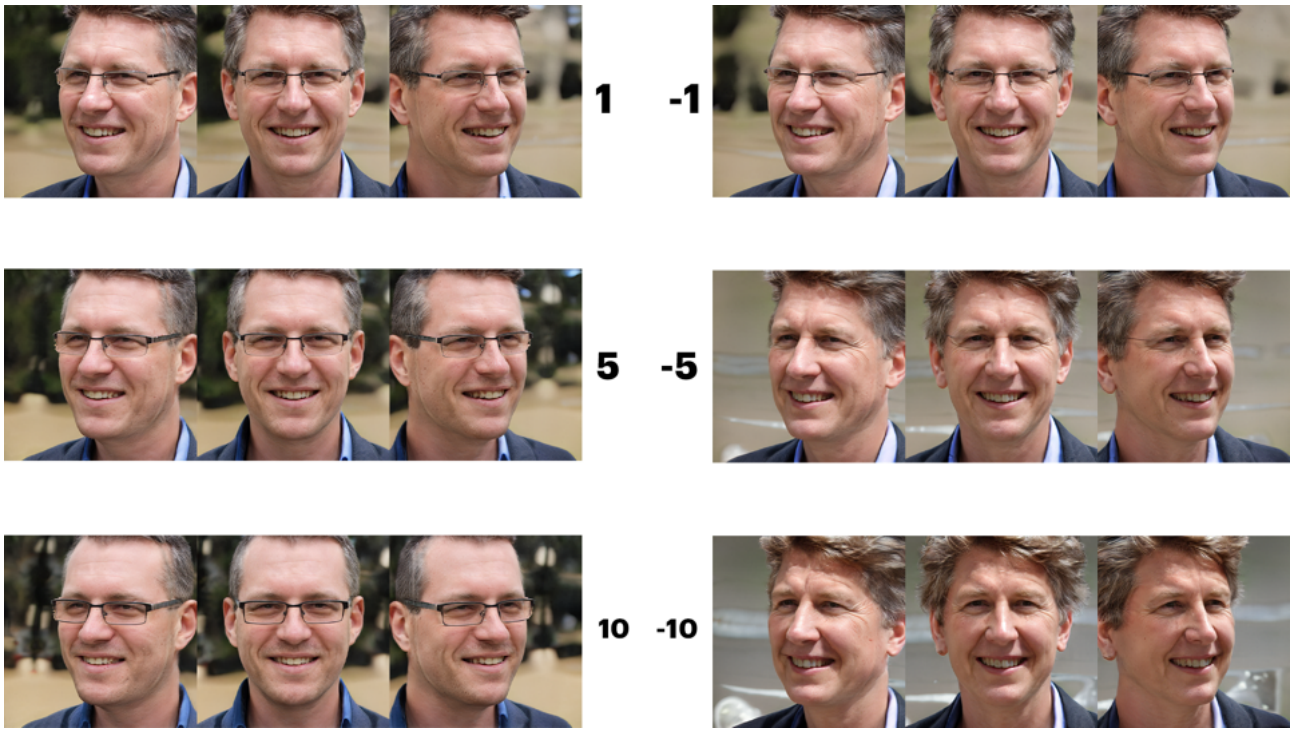

Figure A.2: Effects of altering z[0][28] on synthesized image.

.

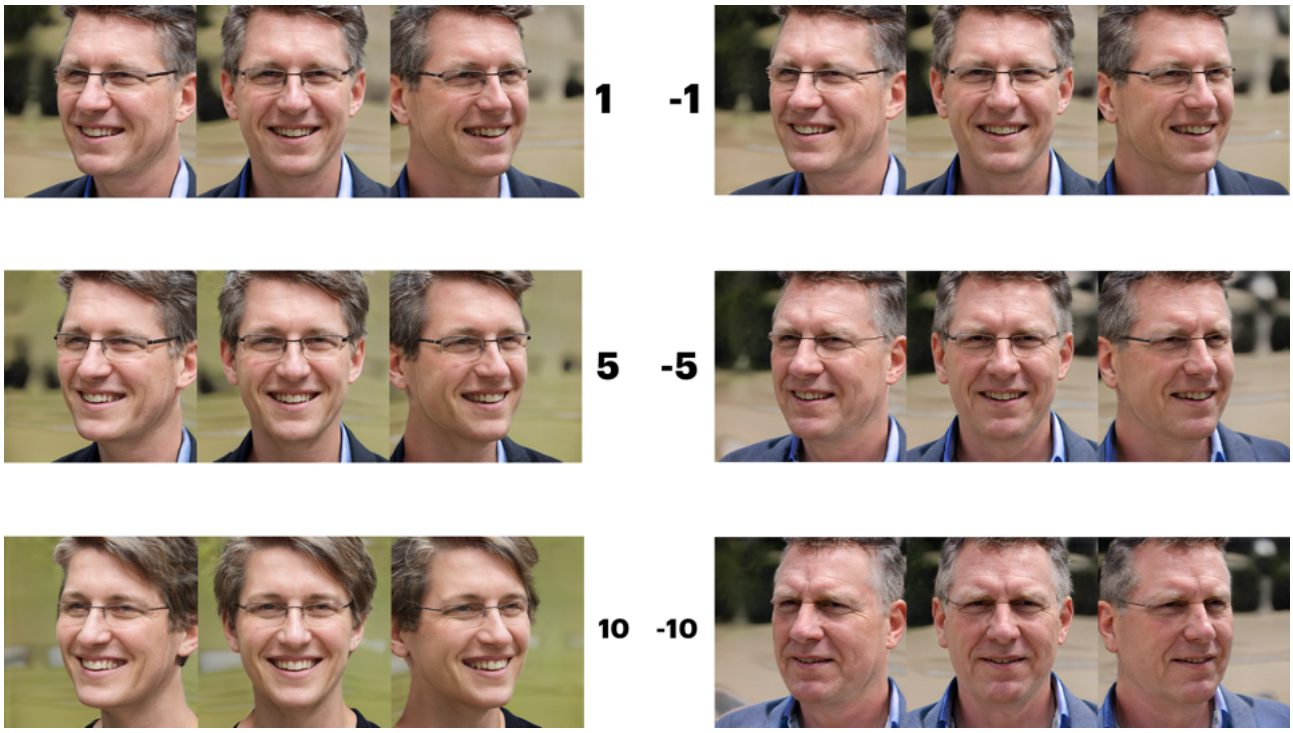

Figure A.3: Effects of altering z[0][30] on synthesized image.

.

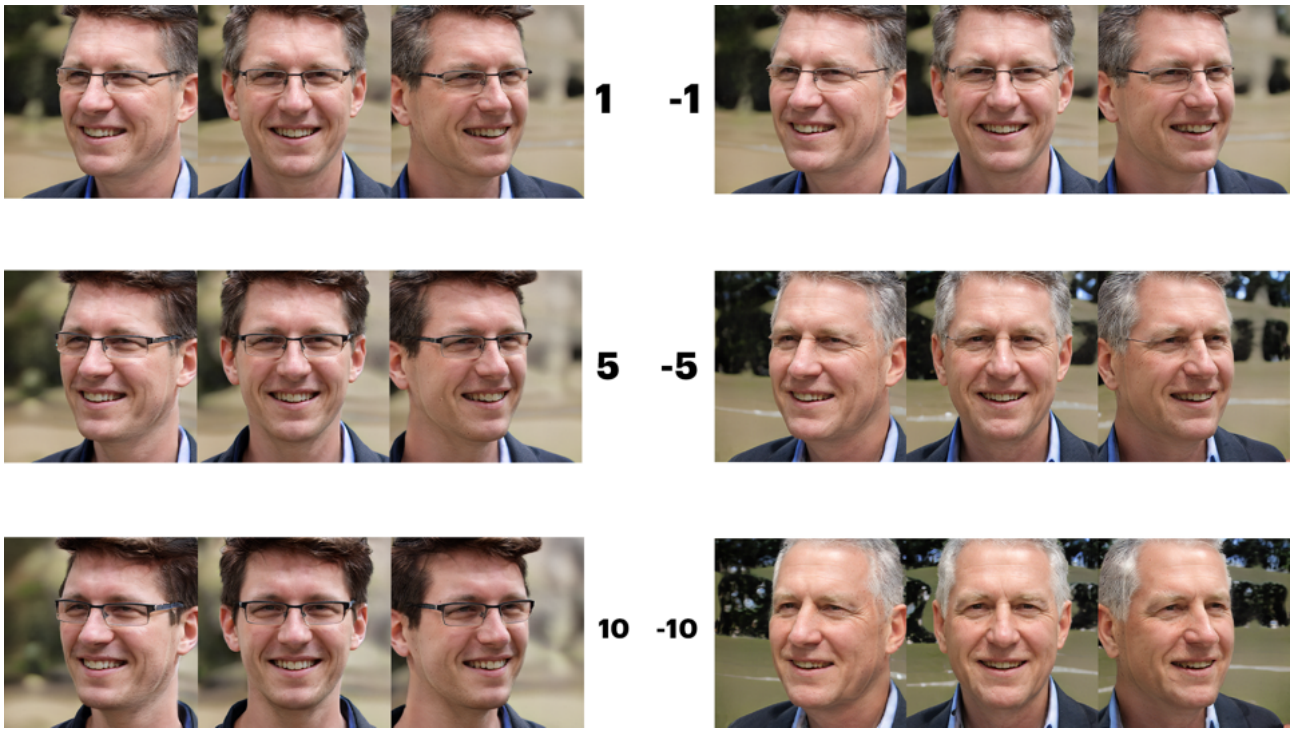

Figure A.4: Effects of altering z[0][51] on synthesized image.

.

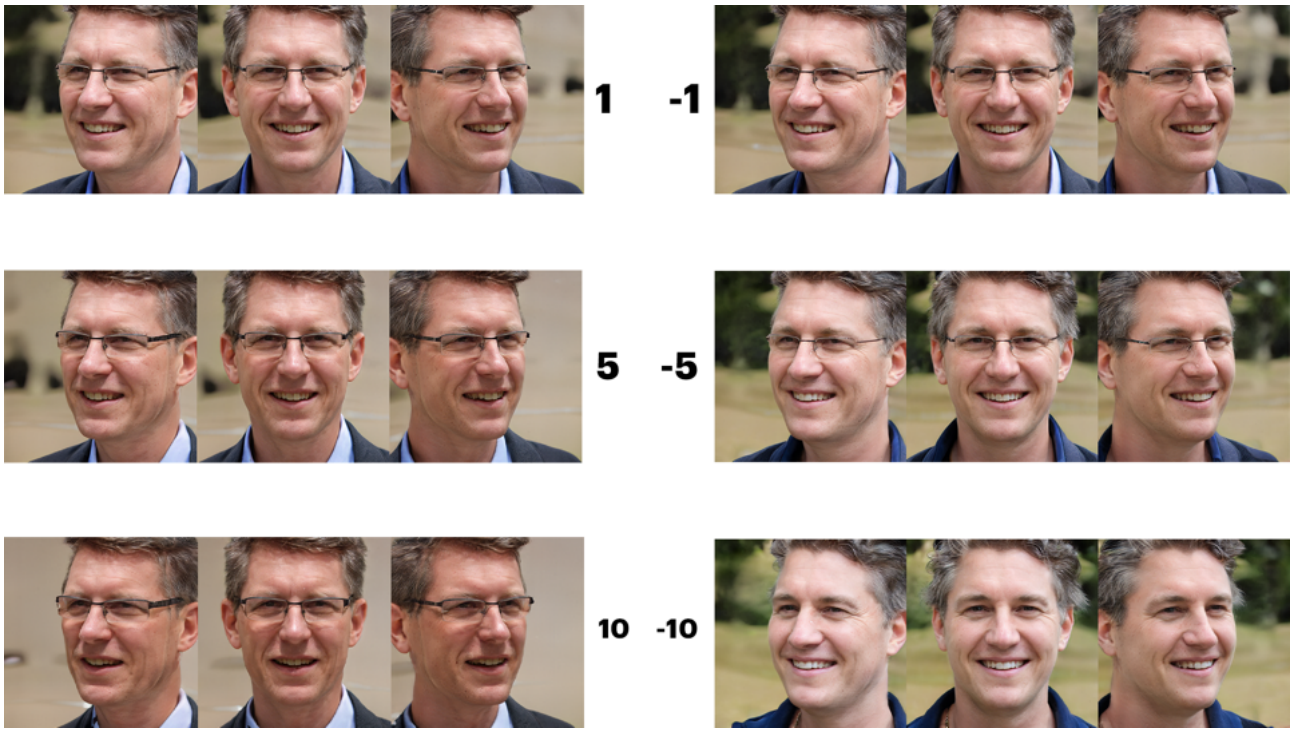

Figure A.5: Effects of altering z[0][100] on synthesized image.

.

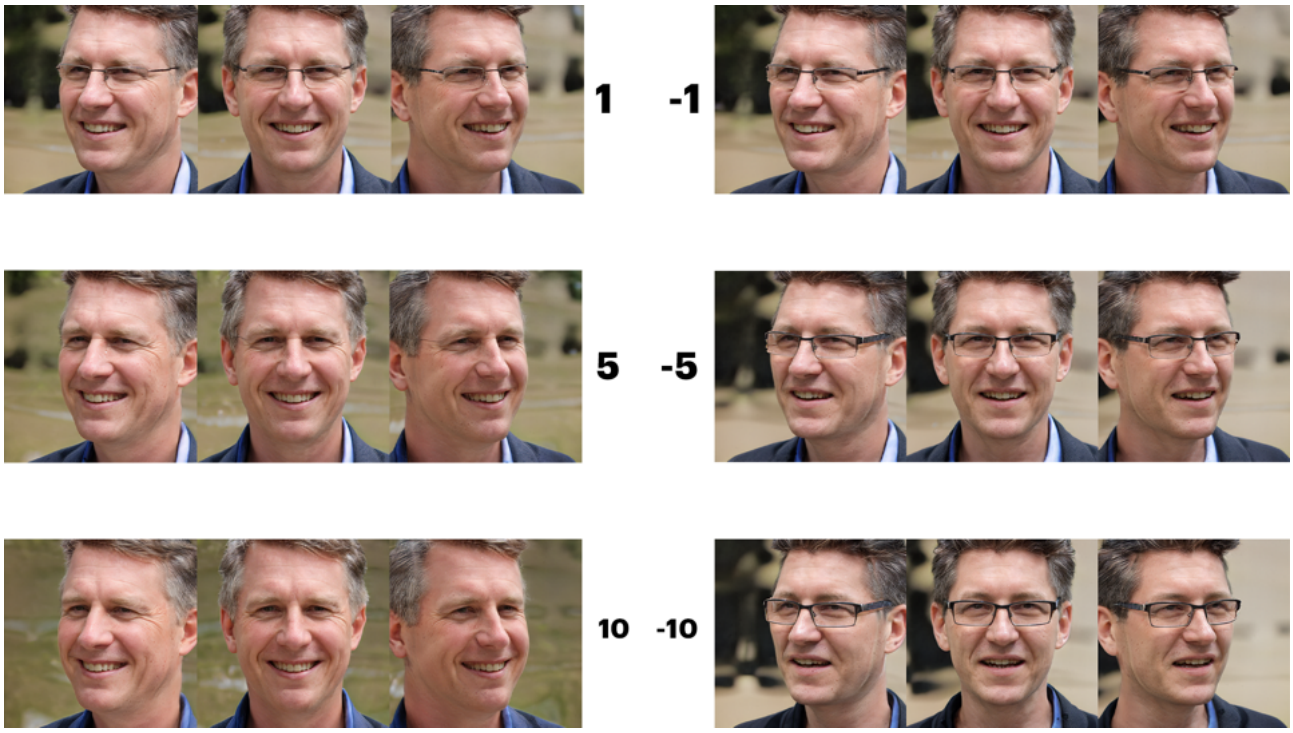

Figure A.6: Effects of altering z[0][102] on synthesized image.

.

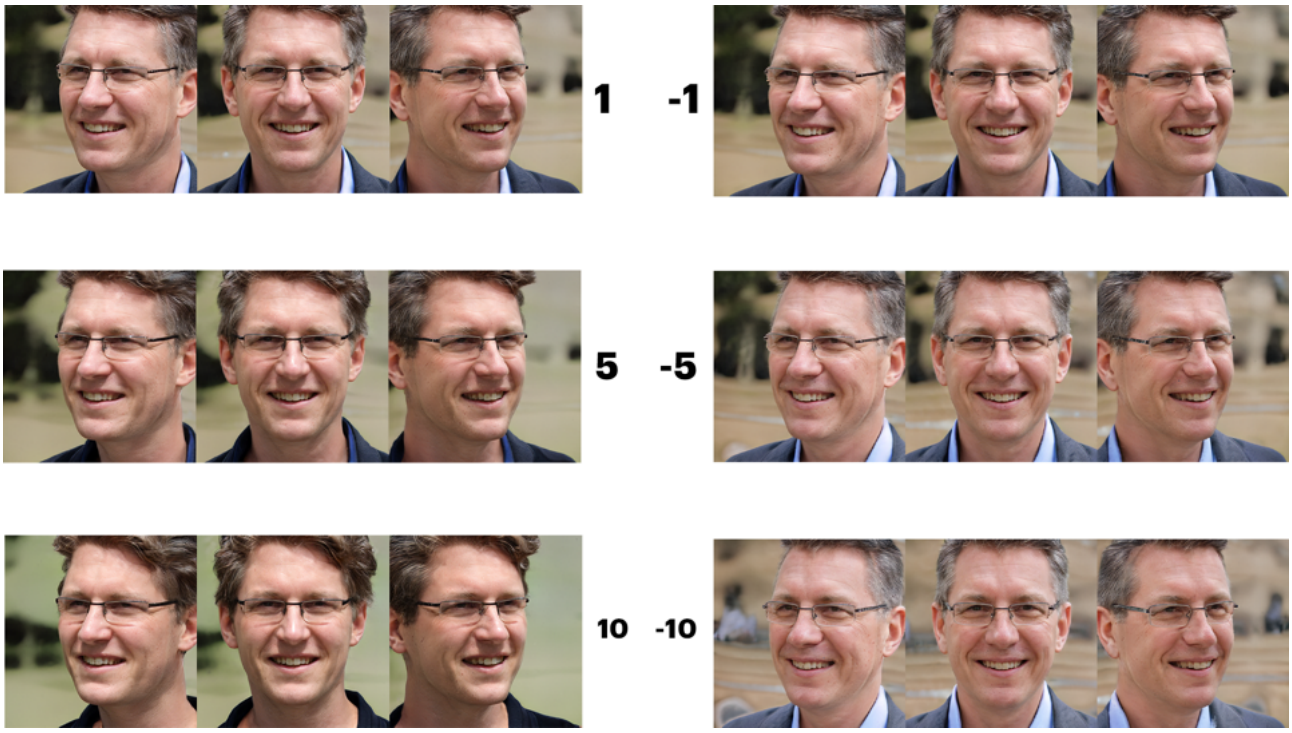

Figure A.7: Effects of altering z[0][107] on synthesized image.

.

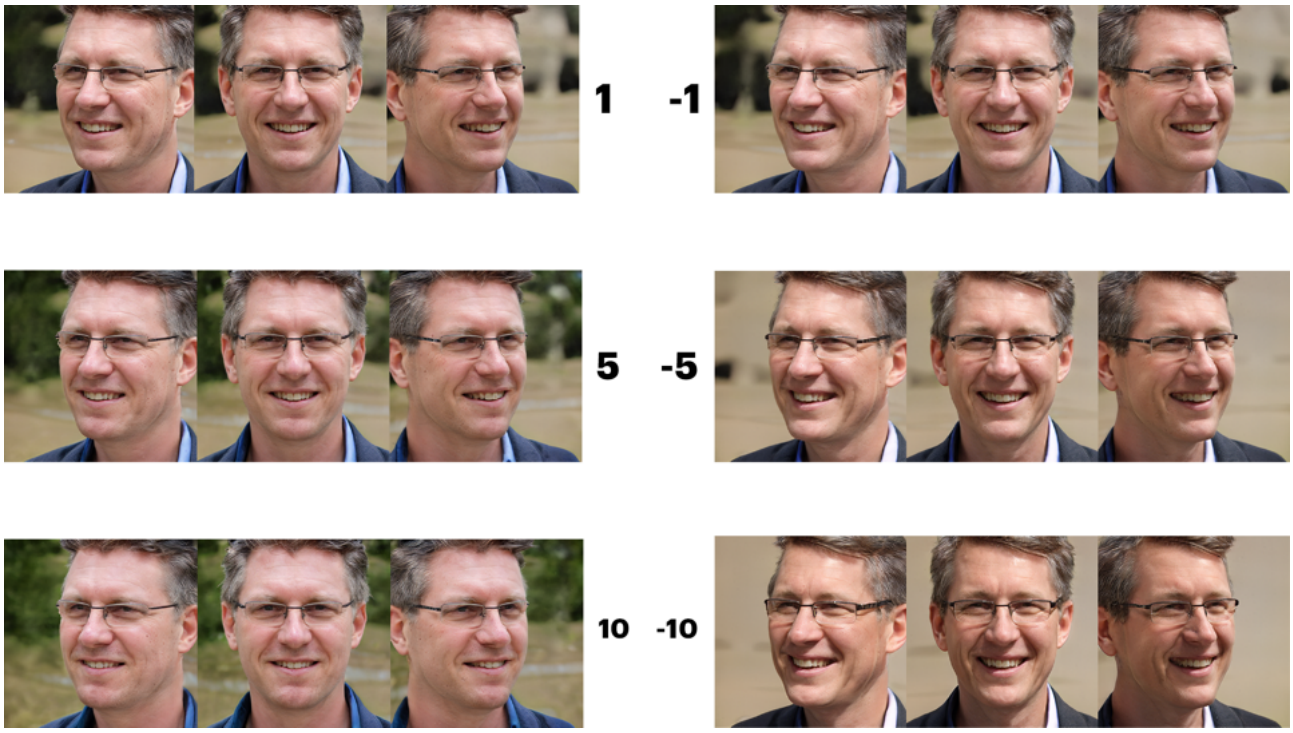

Figure A.8: Effects of altering z[0][244] on synthesized image.

.

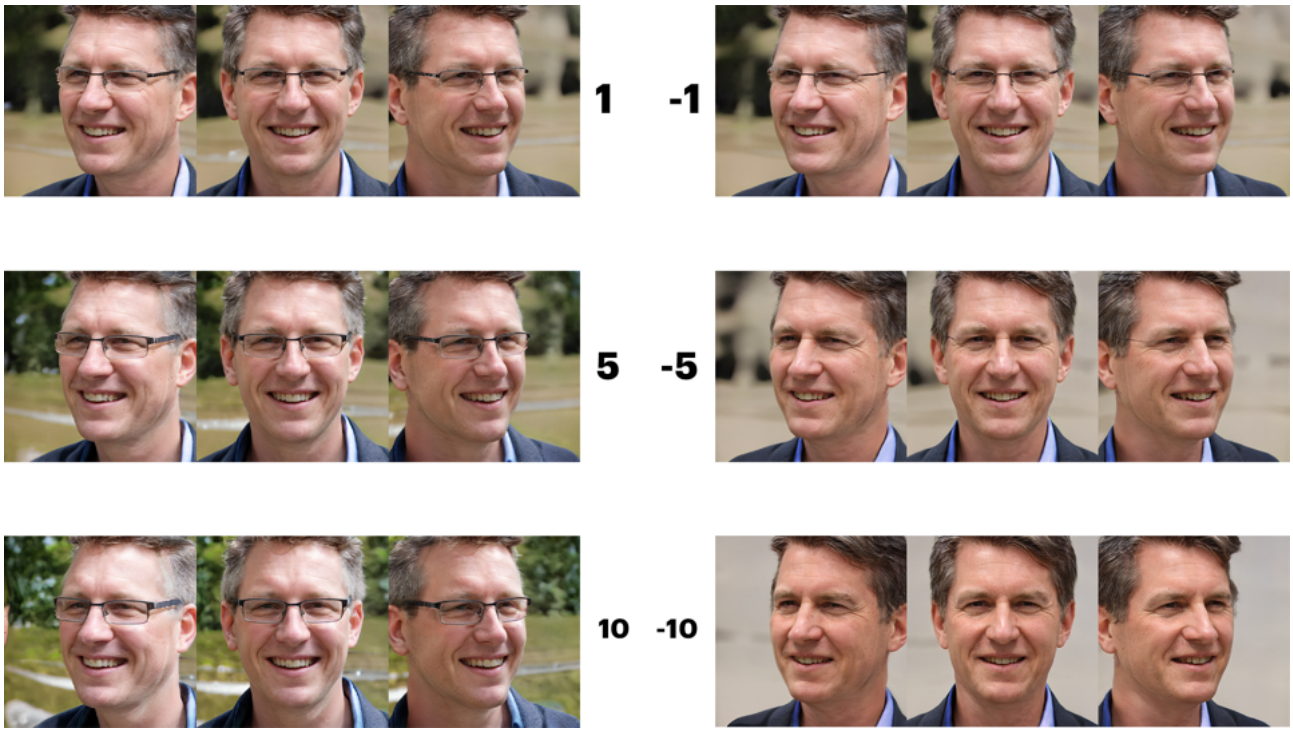

Figure A.9: Effects of altering z[0][495] on synthesized image.

.

A.2 Altering parameters for multiple random seeds

We used the same loop but this time we ran it multiple times for different random seeds. The top image in each figure represents the unaltered image belonging to that random seed.

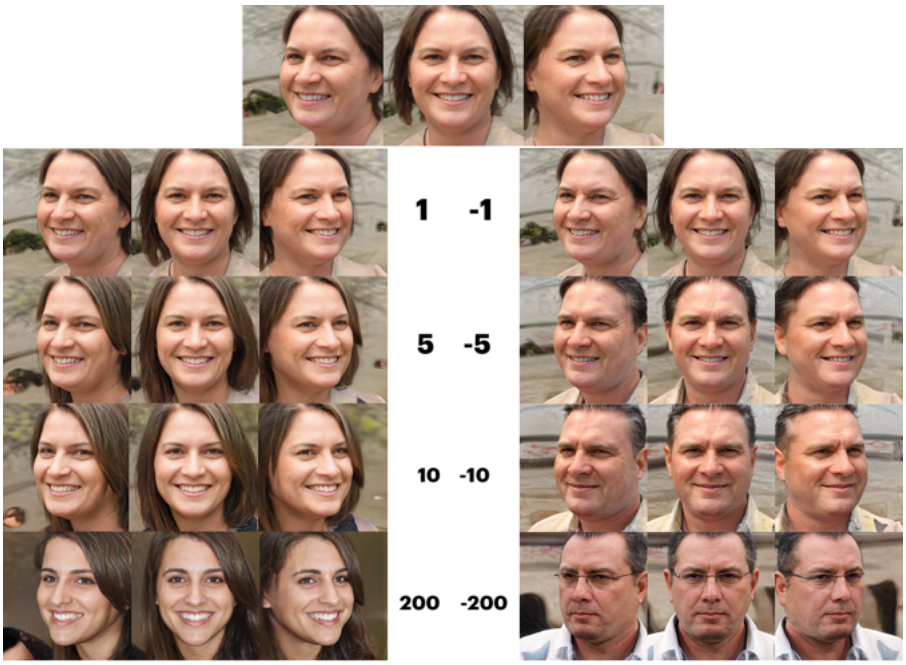

Figure A.10: Effects of altering z[0][30] using random seed 4

.

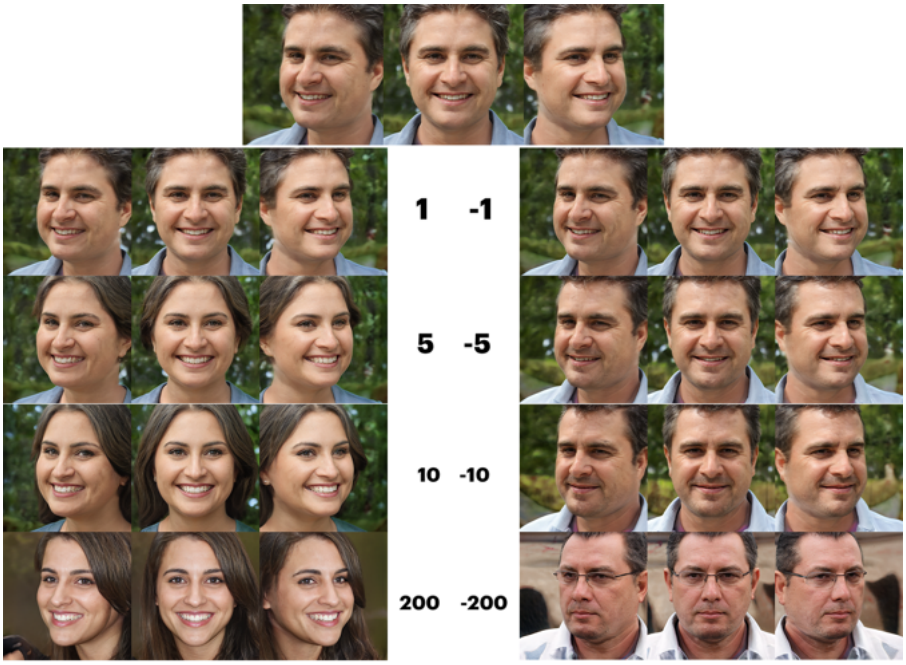

Figure A.11: Effects of altering z[0][30] using random seed 18

.

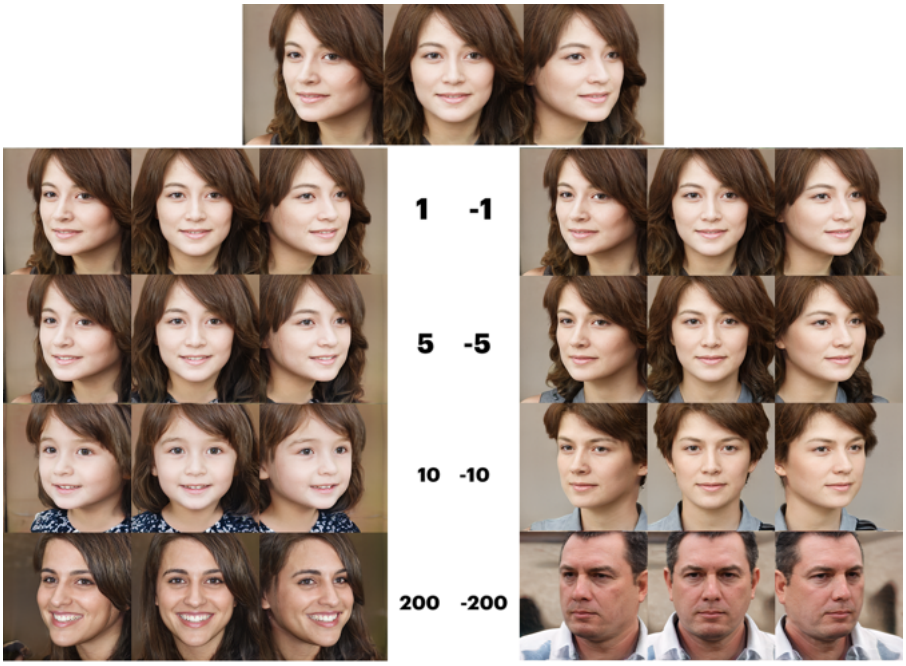

Figure A.12: Effects of altering z[0][30] using random seed 148

.

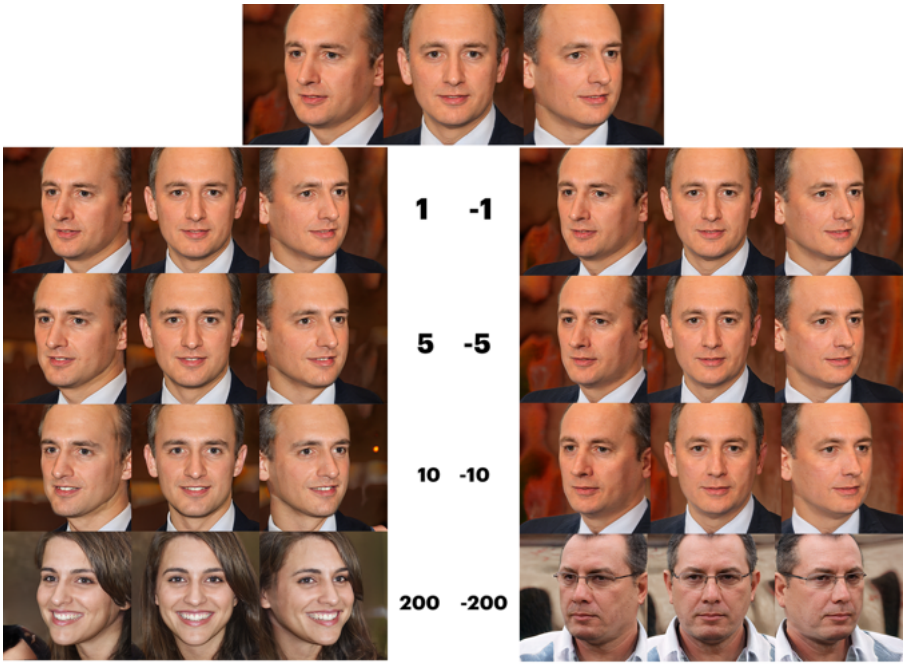

Figure A.13: Effects of altering z[0][30] using random seed 1812

.

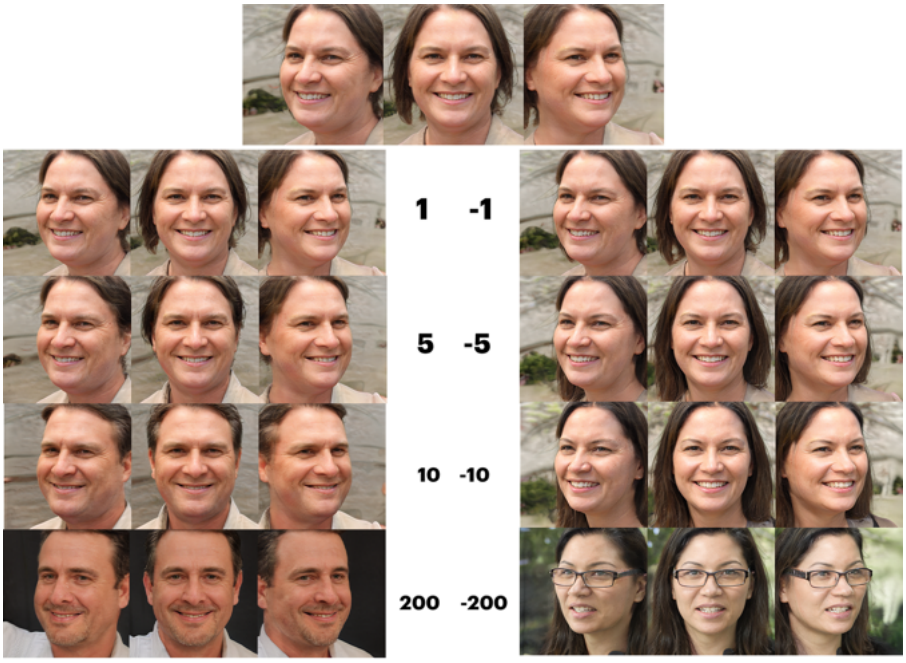

Figure A.14: Effects of altering z[0][102] using random seed 4

.

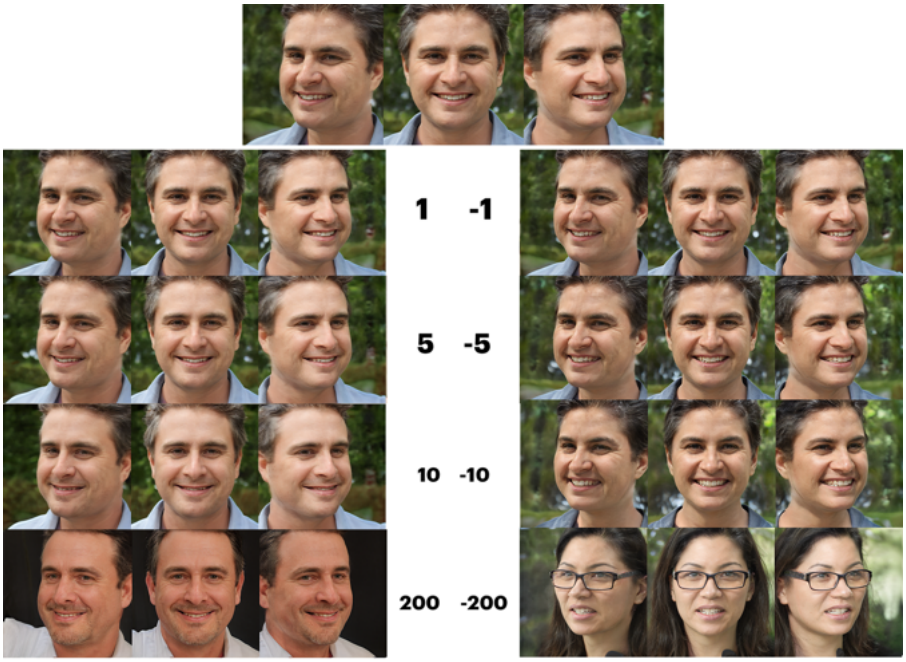

Figure A.15: Effects of altering z[0][102] using random seed 18

.

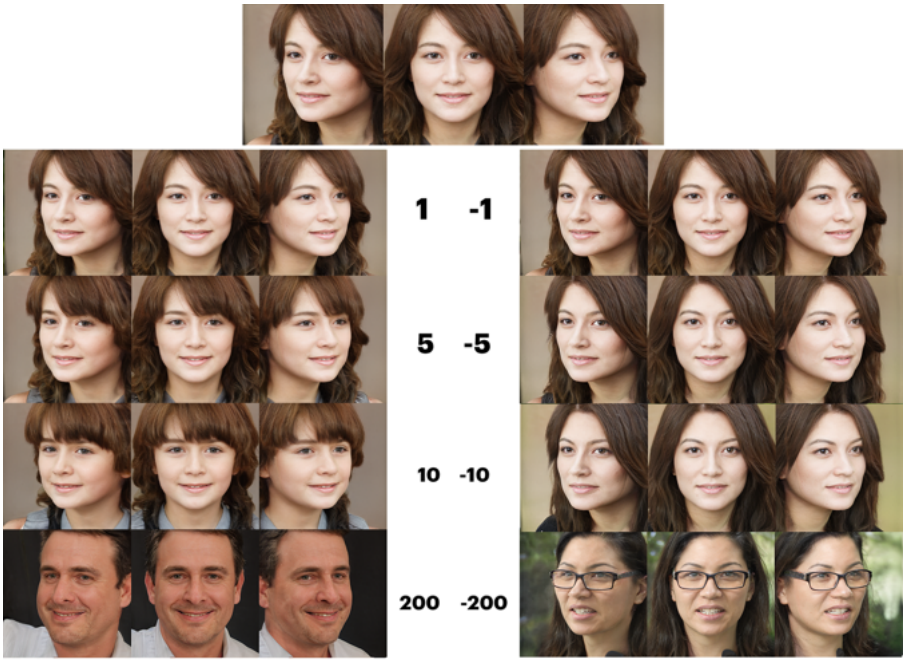

Figure A.16: Effects of altering z[0][102] using random seed 148

.

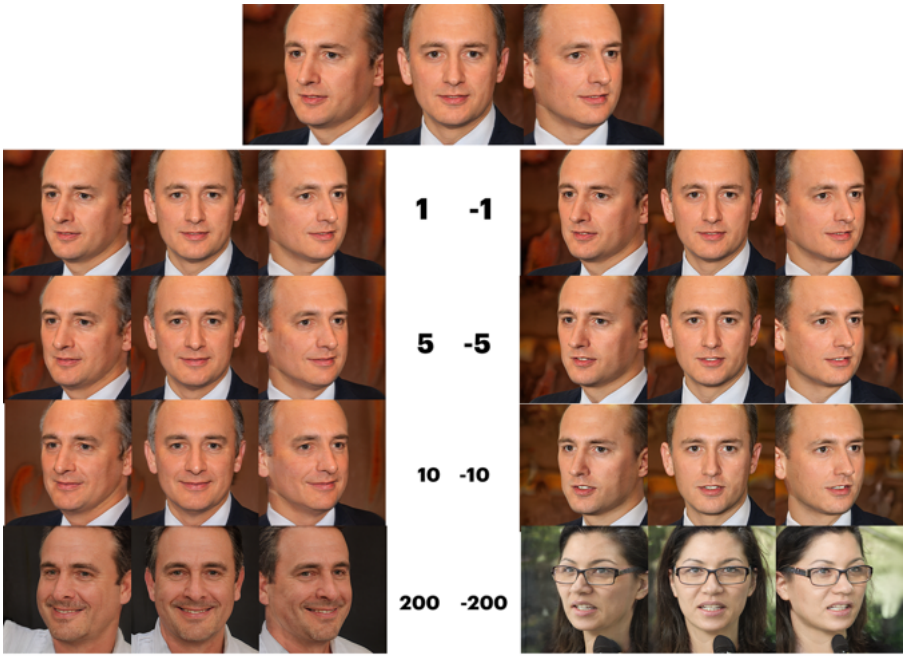

Figure A.17: Effects of altering z[0][102] using random seed 1812

.

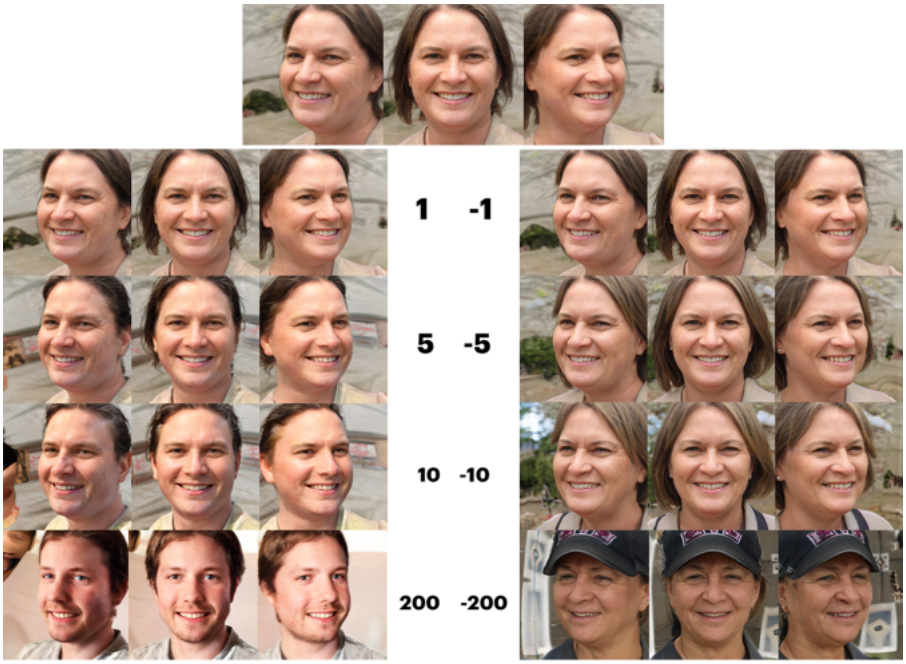

Figure A.18: Effects of altering z[0][302] using random seed 4

.

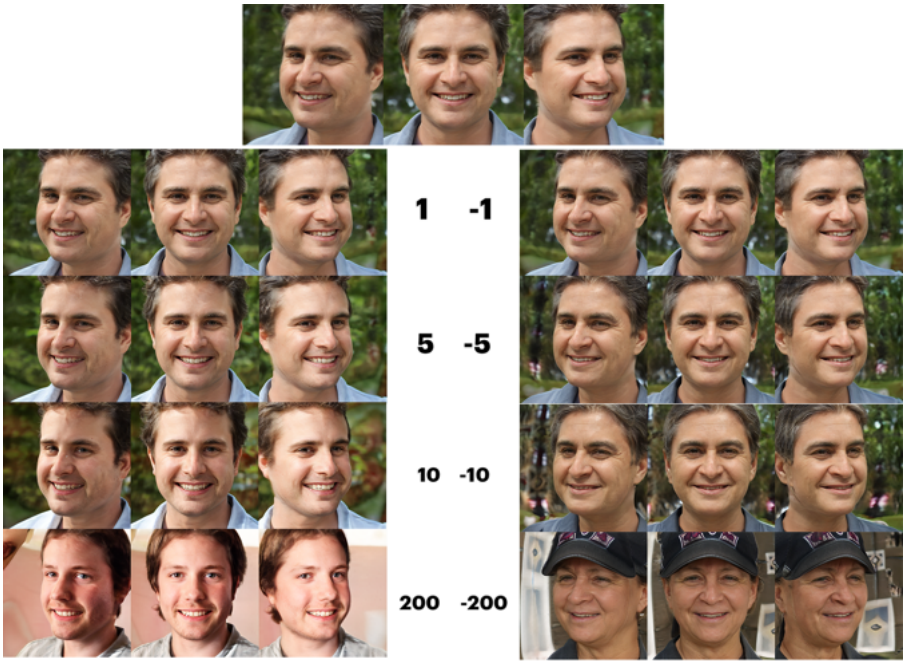

Figure A.19: Effects of altering z[0][302] using random seed 18

.

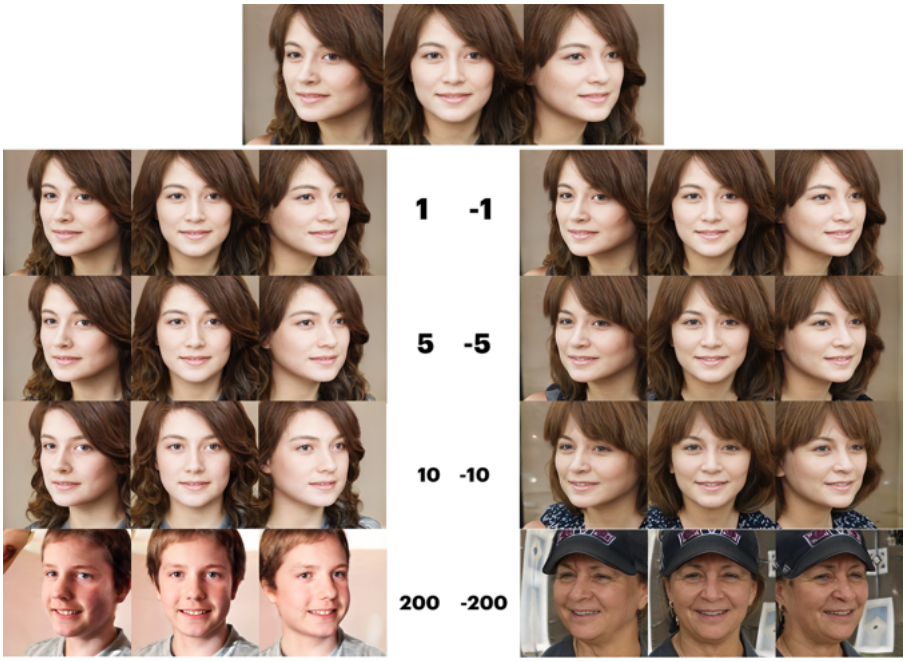

Figure A.20: Effects of altering z[0][302] using random seed 148

.

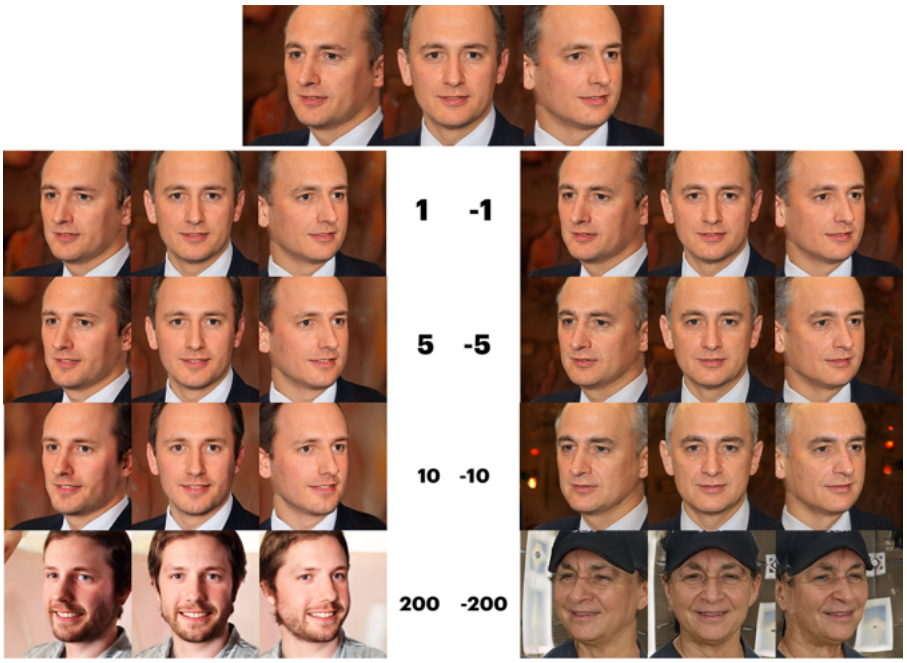

Figure A.21: Effects of altering z[0][302] using random seed 1812

.

B Appendix B: Task Division

B.1 Indy Deker (5419018)

Responsibilities:

- Initial reproducing of paper results
- Research into feature manipulation
- Setting up Blogpost document

B.2 Sebastian Fernandez Ruiz de las Cuevas (5947626)

Responsibilities:

- Initial reproduction of paper results
- Background analysis

B.3 Daniel Soler (6080758)

Responsibilities:

- Initial reproduction of paper results
- Shape artefacts research