

# Multiclass Texture Synthesis Using Generative Adversarial Networks

Maroš Kollár<sup>a</sup>, Lukas Hudec<sup>b</sup> and Wanda Benesova<sup>c</sup>

Faculty of Informatics and Information Technologies, Slovak University of Technology, Ilkovicova 2, Bratislava, Slovakia

**Keywords:** Texture, Synthesis, Multiclass, GAN, Controllability.

**Abstract:** Generative adversarial networks as a tool for generating content are currently one of the most popular methods for content synthesis. Despite its popularity, multiple solutions suffer from the drawback of a shortage of generality. It means that trained models can usually synthesize only one specific kind of output. The usual synthesis approach for generating N different texture species requires training N models with changing training data. However, few solutions explore the synthesis of multiple types of textures. In our work, we present an alternative approach for multiclass texture synthesis. We focus on the synthesis of realistic natural non-stationary textures. Our solution divides textures into classes based on the objects they represent and allows users to control the class of synthesized textures and their appearance. Thanks to the controllable selections from latent space, we also explore possibilities of creating transitions between classes of trained textures for potential better usage in applications where texture synthesis is required.

## 1 INTRODUCTION


The texture definition highly depends on the application area, in which we use this term (Haindl M., 2013). Nevertheless, textures generally describe object's surface properties like appearance, structure, consistency, or feeling from touch. Textures are an essential component of computer vision because they are used in tasks like classification, detection, or segmentation in medicine or the technology industry. Textures are also essential for the graphics and the entertainment industry since almost every animated movie, video game, or other product depends on its visual appearance.


We can classify textures based on their primary characteristic into groups like smooth, rough, glossy, matte, et cetera. There are also more general characteristics like stationarity and homogeneity (Zhou et al., 2018; Portilla and Simoncelli, 2000) that profile textures. It is possible to say that both of these features describe an aspect of texture complexity. Stationarity represents the regularity of structure. Homogeneity represents how many elementary textures are included in the evaluated texture. The more complex the texture structure is, the more difficult it is to synthesize.


Texture synthesis is a process of creating artificial textures that can be used to augment datasets needed for computer vision tasks. It can also replace texture photographing or painting with a more comfortable and less time-consuming content creation method. There are multiple texture synthesis approaches; however, current research orients on Generative adversarial networks (GANs) and diffusion networks. GANs proved their advantages in the quality of outputs and speed of generating. On the other hand, their disadvantages are long and challenging training accompanied by problems like vanishing gradients or mode collapse. Another drawback is that usual solutions are trained to synthesize one texture class, and multiple learned models are required to synthesize multiple texture classes (Zhou et al., 2018; Jetchev et al., 2016). That results in higher disk storage requirements for storing the network models and the inability of creating transitions between individual textures.

Our paper introduces the following contributions:

- We propose two approaches focused on controllable multiclass texture synthesis that uses a latent space as a control mechanism. Our solution is tuned to synthesize non-stationary textures from the natural environment.
- Latent space used for texture control is computed by pre-trained feature extractor before training the synthesis solution. This gives the advantage to modify the feature extractor based on class adja-

<sup>a</sup>  <https://orcid.org/0000-0002-1535-6830>

<sup>b</sup>  <https://orcid.org/0000-0002-1659-0362>

<sup>c</sup>  <https://orcid.org/0000-0001-6929-9694>

gency requirements to enhance the latent space independently from the generating process.

- We show that in the field of texture synthesis, the latent space can be used to create transitions between different classes of textures and also control the appearance of a specified texture class.

## 2 RELATED WORK

Texture synthesis has been an active field of research for multiple decades. Many approaches were introduced and categorized into groups during these years based on their main feature.

Non-parametric sampling is considered a traditional approach and, for a long time, was one of the most popular synthesis methods. This approach uses copying parts of sample textures to create a new one. Parts of sample textures are chosen based on their neighbourhood similarity with an area of an already synthesized part of the texture. There are two main types of non-parametric sampling based on the size of individual parts copied to synthesized texture. Pixel-based synthesis (Efros and Leung, 1999; Wei and Levoy, 2000; Shin et al., 2006; Ashikhmin, 2000) that creates texture pixel by pixel and patch-based synthesis (Praun et al., 2000; Liang et al., 2001; Kwatra et al., 2003) that copies whole patches. These approaches are intuitive and relatively easy to implement. On the other hand, their synthesis is quite slow, and there is a possibility that synthesized textures will contain visually duplicate parts.

In contrast to the non-parametric sampling, a parametric synthesis (Portilla and Simoncelli, 2000) uses parameters to describe texture statistics. The synthesized image is created by gradually changing random noise (Gatys et al., 2015b). Two textures should have identical statistics to be considered similar (Martin and Pomerantz, 1978).

In recent years textures have been synthesized mainly by using neural networks. Gatys et al. (Gatys et al., 2015b) created a parametric approach that used convolutional neural network VGG-19 (Simonyan and Zisserman, 2014) to extract Gram matrices at multiple layers as texture statistics. The new texture is synthesized from random noise. Noise is passed through the network and edited by gradient descent to minimize the difference between Gram matrices of example texture and new texture.

Variational autoencoders (Kingma and Welling, 2014; Chandra et al., 2017; Pesteie et al., 2019) are another approach that uses neural networks to synthesize textures. Variational autoencoders are similar to autoencoders, but their function is to create similar

output, not identical. They consist of an encoder part that maps input data to a low-dimensional representation and a decoder part that reconstructs this representation to output. A drawback of this synthesis solution is the quality of output that could be blurry.

Diffusion networks (Ho et al., 2020; Croitoru et al., 2022; Dhariwal and Nichol, 2021) are generative models inspired by nonequilibrium thermodynamics. They are based on two stages. A forward stage defined as a Markovian chain slowly destroys data by adding random noise to transform data into pure noise. A backward stage learns to recover the data by reversing the addition of noise. The new data is created by passing random noise to the learned model that synthesizes the final image by gradually predicting and removing noise. Diffusion networks produce high-quality images and are currently considered a state-of-the-art approach. Even though there is a disadvantage in long inference time due to the iterative approach.

Although the quality of diffusion network, generative adversarial networks introduced by Goodfellow et al. (Goodfellow et al., 2014) are still popular approaches for image synthesis. Generative adversarial networks contain two neural networks, a discriminator and a generator. These networks train themselves by min-max two-player game. That results in improved output quality of generated output. The generator's goal is to use random noise to create output that the discriminator would not reveal as fake. The goal of the discriminator is to determine which inputs are real and which are fake correctly. Since the original GAN solution introduction by Goodfellow et al., multiple GAN modifications have been created. Proposed modifications changed the architecture of generator or discriminator (Radford et al., 2016), used different adversarial loss functions (Arjovsky et al., 2017; Gulrajani et al., 2017), stabilized training (Karras et al., 2017), improve variation of outputs (Salimans et al., 2016) or create a new approach of training (Zhu et al., 2017).

Several GAN solutions have also focused on multiclass texture synthesis. Li et al. (Li et al., 2017) introduced solution DTS that uses one hot encoding vector as a control mechanism for the synthesis of multiple types of textures. The architecture of their proposed generator consists of two streams: one for synthesizing the final output and the second for processing input information that controls the class of synthesized texture. Their solution also showed that it is possible to create interpolations between learned textures that lead to new types of textures.

Another multiclass synthesis solution is PSGAN introduced by Bergmann et al. (Bergmann et al.,

2017). Their solution can learn multiple types of periodic and non-periodic textures from a dataset or high-definition images. A disadvantage of this solution is that the texture control mechanism (global dimensions of the input) is sampled as a random vector, so it does not allow control of which texture will be synthesized. The solution also does not provide complete coverage of the dataset.

Following these disadvantages, Alanov et al. (Alanov et al., 2020) proposed a solution that updates the creation of global input dimensions by using an encoder. The encoder is trained alongside the generator to learn textures' latent representation. Learned representation is then used as a control mechanism. They also ensure full dataset coverage by penalization for incorrect reproductions of a given texture.

### 3 METHOD

The main goal of our work is to create a robust controllable method for multiclass texture synthesis with a focus on non-stationary textures and maintaining the quality of generated textures. The controllability should ensure a change of texture type and its look. Proposed versions of the architecture of our solution are shown in figure 1.

#### 3.1 Generator

The core of our approach, the generator, uses the idea of a two-stream network from the work of Li et al. (Li et al., 2017). This architecture helps to force the network to synthesize a required texture class. The primary stream handles the synthesis process, and the secondary stream processes information on which texture class should be generated. Activations of secondary streams are merged as a 32-channel feature map to activations of the primary stream after every processed spatial upsample. This ensures that the primary stream has additional information about a class of synthesized textures at every resolution. An upsample function upsamples the resolution in both primary and secondary streams with scale factor 2. The exception is the first upsample of vector done by transposed convolution. After upsample, activations are processed by convolutional layers with instance normalization and leaky relu activation functions.

Because of the exertion of multiclass non-stationary texture synthesis, the solution suffered from visual artifacts and had difficulty learning all types of presented textures, even though there were only six. To deal with this problem, we implemented the generator as a progressively growing GAN (Kar-

ras et al., 2017). This approach helped to stabilize learning thanks to the ability of the network first to learn the color palette of synthesized textures and then increase resolution and learn details of textures. The current implementation of our solution generates outputs of  $128 \times 128$  pixels. However, thanks to the progressive, growing GAN implementation, more layers can be easily added to increase the resolution of the final output. We also changed batch normalization to instance normalization and added hyperbolic tangent as the final layer to improve the quality of outputs. Hyperbolic tangent limits values of output pixels between -1 and 1 to prevent very high or low values. The generated output is then used to train the discriminator without clipping to  $[0, 1]$  to force the generator to learn the correct interval of pixel values.

#### 3.2 Generator Inputs

The difference between alternatives of our solution is based on changes in inputs for the generator and the process of obtaining them. As a consequence of two generator streams, the generator requires two inputs.

In our first solution alternative, input for the secondary stream is realized as a one-hot encoding vector. This vector is inspired by the solution of Li et al. (Li et al., 2017) and encoded as a position of a single high bit, which clearly describes which texture should be synthesized. The primary input is inspired by the solution of Bergmann et al. (Bergmann et al., 2017). In contrast with their solution, our input is not constructed as a matrix of vectors but only as one vector. The input vector is concatenated from three parts: random, texture, and selection, whose lengths were set experimentally based on the quality and diversity of outputs.

- The random part is sampled from the uniform distribution on the interval  $[0,1]$ . This part provides variability between generated outputs.
- The selection part is a copy of the input for the secondary stream. One-hot encoding vector is included in primary input to contribute information about the selected texture even at the first layers of the generator.
- The texture part is created from an example image of the required texture class. An example image transformed to a grey scale is used as input for our pre-trained classification network. We selected the classification network and principal components analysis approach for initial experiments because of relatively straightforward network training and the possibility of identifying a well-trained classification network based on classification metrics. We took input activations of

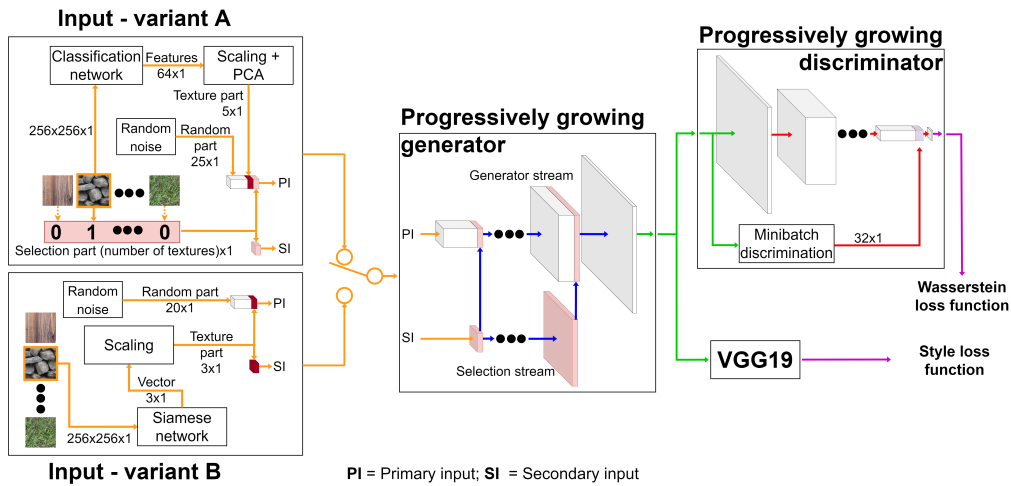


Figure 1: Visualization of the architecture of our solution. The left side of the figure shows variants of input for a generator. Variant A uses classification network as feature extractor with PCA for texture part of input. Variant B uses siamese network for texture part of input. The generator consists of two streams for image synthesis and keeping the information about the selected texture class. The discriminator is enriched with information from Minibatch discrimination. We use a combination of Wasserstein loss with Style loss computed by a pre-trained VGG19 network as a loss function.

the last fully connected layer as a feature vector and applied principal component analysis on this vector to obtain a representation vector of length five numbers. This part of the input assists the selection part with information about the selected texture and also brings variability to the input. To confirm that this vector carries information about the selected texture type, we visualized part of the vector in 3D space. As we can see in figure 2, every cluster of texture type could be approximately separated.

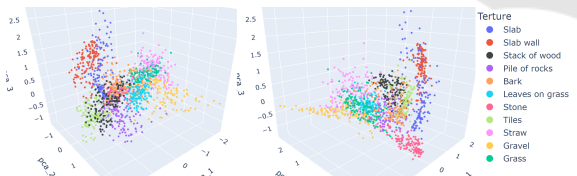


Figure 2: A visualization of the clusters of 3 out of 5 values from the texture part created by the classification network and PCA. The visualisation shows that individual clusters in the latent space are approximately separable.

In our second solution alternative, we replaced the classification network and PCA with a Siamese network and left out the selector part of the input. That leads to using the texture part as the only controlling mechanism in a primary and secondary input. Siamese network as the texture part extractor was selected for its ability to learn similarities that leads to a better embedding of textures in the latent space. Figure 3 shows clusters that refer to trained texture types in latent space created by our trained Siamese network. The selection part was left out based on testing

the first alternative. Tests showed that manual control of generator inputs for changing texture type or creating transitions between different textures is hard to use. That also leads to minimizing the length of the texture part from 5 to 3 values.

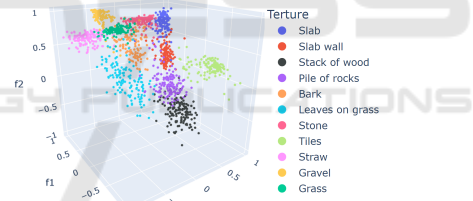


Figure 3: A visualization of the clusters from the texture part created by the Siamese network. The individual clusters are better separable than in the version that uses classification network and PCA.

The decision to use the pre-trained feature extractors was made because of the advantage of reviewing if the feature extractor is well-trained or needs to be trained more. In the case of the siamese network, we can affect the distribution to decrease distances between clusters of different texture classes or rearrange the cluster neighbors. For this purpose, during the training of the siamese network, we can define shorter distances between feature vectors for concrete pairs of textures that we want to be closer to in the latent space. This will allow us to create transitions between pairs of textures that we want. We can define the higher distance between feature vectors for other pairs of textures we don't want to be closer to.

### 3.3 Discriminator

The initial discriminator used in our solution was inspired by the DCGAN solution (Radford et al., 2016). We changed it by adding a sequence of batch normalization, leaky relu activation function, and convolutional layer with kernel size 5, stride 1, and zero padding behind the last convolutional layer. Because of the implementation of the progressive, growing generator, the discriminator also had to be implemented as progressively growing. That caused a change of the parameters of the first convolutional layer to kernel size 3, padding, and stride set to 1. A number of the following sequences of convolutional, normalization, and activation layers from the DCGAN solution are based on the current step of progressive growth. We also added a fade-in mechanism to handle adding new layers during progressive growth. Feature maps from the last convolutional sequence are flattened to a 1D feature vector. To deal with the mode collapse problem, we concatenated the final 1D vector of the discriminator with the 1D vector from minibatch discrimination (Salimans et al., 2016). This helps the network to identify a batch of generated images. Minibatch discrimination compares all images in a batch and creates a vector that references how similar images are. In case of high similarity, we can assume that images are synthesized and suffer from mode collapse. Based on executed tests, we noticed that minibatch discrimination helps mainly in the first two steps of progressive growth, which correspond to images of size  $8 \times 8$  and  $16 \times 16$ . The final change was to switch the sigmoid layer to a linear layer.

### 3.4 Loss Functions

Instead of the original loss function used in Goodfellow’s proposed GAN solution, we used Wasserstein loss (Arjovsky et al., 2017) with weight clipping to increase output quality and stability of training. That changed our discriminator to critic, which does not determine real or fake textures but tells the distance between the distribution of generated outputs and training data. We combined Wasserstein loss with style loss (Gatys et al., 2015a) during the training process to speed up the training process and bring a closer visual appearance of synthesized and real textures. The texture loss is calculated from feature vectors of the first five convolutional layers of the pre-trained classification network VGG19. Feature vectors of synthesized and real textures were compared using Gram matrices. Style loss  $L_{style}$  is added to Wasserstein loss  $W$  for generator loss  $L_g$  and could

be controlled by the weight factor  $\beta$ . Based on experimenting with the value of  $\beta$ , we found that the ideal value in our setup is 1. In the case of a higher style weight factor, the network learns style relatively fast, but outputs suffer from strong mode collapse and low quality. Formulas to calculate loss functions for the generator and the discriminator are shown in equations 1 and 2 with  $L_d$  as discriminator loss.

$$L_g = -W_{synth} + \beta * L_{style} \quad (1)$$

$$L_d = W_{real} - W_{synth} \quad (2)$$

### 3.5 Dataset

The dataset used for training our solution consists of eleven different classes of textures. Examples of all texture types are shown in figure 4.



Figure 4: Example of a patch from every 11 texture classes used in the training dataset. Top: slab, slab wall, a stack of wood, a pile of rocks, bark, leaves on grass. Bottom: stone, tiles, straw, gravel, and grass.

We gathered our data by capturing close-up photos with textures of natural environment objects and adding two publicly available textures from the Internet to balance the number of photos in individual classes. Every texture class contains 67 main images. In our solution, textures are perceived by the objects they represent. It means that, for example, barks from different types of trees (like a cherry tree, a linden tree, etc.) might be visually different, but we still see them as one class of texture. Our intention behind this is to increase the internal variation of classes. It also brings an opportunity to create a solution that could benefit from the simpler controllability of the texture class and the possibility of changing the appearance of texture by different subtypes.

Images in the dataset are used to create smaller cutouts of various sizes to augment the dataset. Because of differences in the scale of base texture structures in the images, we scaled various sizes of cutouts to generalize the model to the variability of input images. Cutouts positions are selected randomly to get numerous different cutouts of training data. Cutouts for training are then down-scaled to the size of data in mini-batch, which is always smaller than the size of created cutouts.

While creating the dataset, our primary goal was to select non-stationary homogeneous textures. Because of this condition, we mainly selected natural textures that are separable from their surroundings and do not have any regular pattern. However, in a few cases, we violated our intention to gain information about the behavior of our solution in exceptional cases. These cases are represented by adding the texture of tiles that is stationary and the texture of leaves on grass that is heterogeneous. We also added two similar textures, slab and slab wall, to determine how well our solution would generate multiple similar textures.

### 3.6 Training

Various techniques that affect the alternation of classes could be used during the training in multiclass synthesis. We decided to use a simple cyclic alternation that ensures the change of texture class after every batch. We implemented this alternation as a modulo operation of a count of batches and a total number of texture classes where the result corresponds to the currently synthesized/learned texture.

It is usual practice to violate the symmetry of learning and train the discriminator more often than the generator. The typical learning ratio between the discriminator and the generator is 5:1 (the generator is updated every fifth update of the discriminator). This helps the discriminator to be trained more than the generator and to provide better feedback for the training of the generator. We also implemented this technique as the modulo of the count of batches and learning ratio. Because of that, it is necessary to ensure that the generator is trained on every texture class. That could be achieved by setting the learning ratio and the total number of textures to be co-prime. Because of the typical learning ratio (5:1), the easiest way to preserve all conditions is to choose the number of texture classes indivisible by five.

Whereas our solution is progressively growing, we also needed to ensure a mechanism to control when to add new network layers to be trained. This issue is resolved using predetermined values assigned to each step of progressive growth, indicating how many epochs each step will be trained. This method, despite its simplicity, allows control over the length of training steps because later steps of the progressive growth need more extended training than the first steps with low output resolution.

Yet final training was trained for 2250 epochs unevenly divided into 5 steps from  $8 \times 8$  to  $128 \times 128$  pixels (100, 300, 450, 600, 800 epochs). The resolution of every step is double the previous resolution.

Every epoch consisted of 275 batches of 64 images. The training takes about six and a half days on a single NVIDIA GeForce RTX 3090 GPU. As optimizers, we used RMSprop with a learning rate set to  $5 \times 10^{-5}$  for the generator and  $9 \times 10^{-5}$  for the discriminator.

## 4 EVALUATION

To compare alternatives of our solution, we performed a quantitative comparison based on Fréchet Inception Distance (FID) (Heusel et al., 2017) that captures the similarity of real and synthesized image collections. FID uses an Inception network to get distributions of activations of real and synthesized data. Statistics (mean  $\mu$  and variance  $\Sigma$ ) of distributions are used for calculating the distance between them. A lower FID indicates more similar distributions, thus better quality of synthesized images than those with higher FID. FID could be calculated by equation 3.

$$FID(r, g) = \|\mu_r - \mu_g\|_2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{\frac{1}{2}}) \quad (3)$$

The quantitative comparison results (table 1) show that for similar training steps, the alternative that uses the classification network for obtaining the texture part of the input and uses the selection part achieves better results than the alternative with the siamese network. The table also shows that the quality of outputs could still be improved by more extended training of individual steps of progressive growth.

Considering the problem of non-stationary texture synthesis, empirical evaluation by the human eye is one of the most quality and accurate ways of evaluation that works even on a low number of generated outputs. This type of evaluation is a qualitative technique based on human observation and feelings of how accurate the texture is or how easily it could be identified as fake. Thanks to this evaluation, we could estimate the quality of our solution from the beginning of implementation and lead the architecture to better results.

Examples of synthesized textures by our solution that use the classification network are shown in figure 5. The figure also shows textures from our dataset synthesized by the solution of Li et al. (Li et al., 2017). Since their solution uses one image as a single texture class, for every class of our dataset, we selected 5 images preprocessed the same way as textures for our solution (random crop and resize). For training of the solution, we used implementation and parameters predefined on the GitHub project <sup>1</sup> with a

<sup>1</sup>[github.com/Yijunmaverick/MultiTextureSynthesis](https://github.com/Yijunmaverick/MultiTextureSynthesis)

Table 1: Quantitative comparison (FID) of alternatives of our solution.

	Epochs for steps of progressive growth					Total	FID
	1	2	3	4	5		
Siamese	100	200	250	375	450	1375	77.53
Classification + PCA	100	200	250	375	450	1375	62.71
Classification + PCA	100	300	450	600	800	2250	39.58



Figure 5: Examples of synthesized textures by our solution with input from classification network + PCA and solution by Li et al. (Li et al., 2017). Left section from top: slab, slab wall, stack of wood, pile of rocks. Middle section from top: bark, leaves on grass, stone, tiles. Right section from top: straw, gravel, grass.

change of the number of iterations to 455 000. As we can see, both solutions synthesize simple non-stationary textures like bark, stone or gravel well. On the results of Li et al., we can see artefacts on textures with stalks like grass or straw. However, these artefacts can possibly be removed by longer training. In results of their solution, there is also a decreased quality of textures that require specific features like gaps between individual slabs in slab walls or logs in pile of wood. It is challenging to compare these two solutions because both solutions have strong and weak aspects of texture quality. In that case we can compare these approaches by usability and controllability of generated texture.

For our model that uses the classification network, we created a survey focusing on the possibility of participants distinguishing if a texture is created by syn-

thesis or the camera. This model was selected based on better performance calculated by FID value. For testing purposes, we used ninety cherry-picked images from every synthesized texture class and ninety images per class from real data. Cherry-picking was done because in the artistic or prototyping process, the user can also decide if the texture looks as he wants and use it or synthesize a new texture and decide again. Cherry-picked textures were about 10% of all synthesized textures. However, this number does not represent the number of usable textures because we wanted the user survey to contain balanced classes. Because of this matter, 10% represents just a ratio of relatively usable textures from the weakest class, which were the classes of piles of rocks and tiles. The textures were shown to participants for three seconds without the name label of showed tex-

ture. These restrictions were used to rule out their overthinking about texture quality based on possible long-time analysis or their expectation of known class appearances. 104 persons participated in the survey; however, only answers from 93 participants were used for final statistics. We filtered out all participants who answered less than half of the survey. We also filtered out participants that identified correctly less than 45% of real textures because if they could not identify real textures correctly, their answers could have undesirably influenced the outcome of the evaluation. The histogram of correctly identified sources of textures is shown in figure 6. Distributions of correctly identified sources of textures are shown in figure 7. By Kolmogorov–Smirnov test, distributions were identified as normal. Based on Student T-test, we found out that the difference between distributions is statistically significant; thus, our solution could synthesize textures that can fool a person. Per texture percentage accuracy of determining the origin of the texture is shown in table 2.

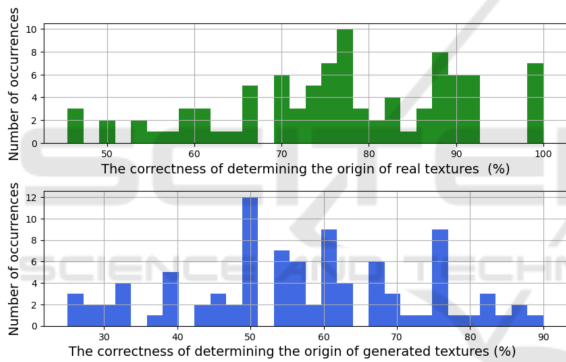


Figure 6: Histograms of correctly identified real and synthesized textures by participants of evaluation.

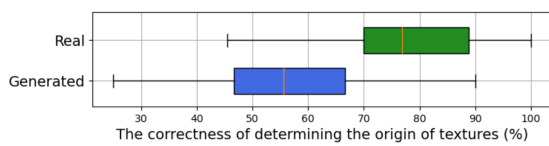


Figure 7: Distributions of correctly identified real and synthesized textures by participants of evaluation.

## 5 CONTROLLABILITY

In the first solution alternative (classification network, PCA, and selector part), the intention behind two different pieces of information on selected texture was to:

- Specify precisely what texture type should be synthesized (selection part).

Table 2: Percentage accuracy of determining the origin of the texture based on texture class.

Class	Real	Synthesized
Slab	69.90	43.27
Slab wall	66.99	46.39
Stack of wood	80.49	65.79
Pile of rocks	87.91	70.48
Bark	62.92	35.51
Leaves on grass	76.47	58.70
Stone	59.00	61.70
Tiles	80.00	76.29
Straw	90.48	58.82
Gravel	71.08	49.53
Grass	79.17	51.06

- Change the appearance of generated texture (texture part), like the colour or pattern of the concrete texture type.

Experiments of control mechanisms refuted the ability of the selection part to define texture class in every case. The texture part must also be located in the latent space area corresponding to the selected class for the expected output class. In some cases of mismatch between the selector and texture parts of the input, the selected texture class could be created with lower quality. In general, output could resemble the different texture types or be unidentified as concrete texture class. On the other hand, experiments confirmed the expected behaviour of changing texture parts to achieve different appearances. Examples of smooth changes of texture appearance created by changing the texture part of input are shown in figure 8. Because of multiple values that need to be set up during the synthesis and the necessity of compatibility between the texture and selector part of the input, manual control of all values is quite complex and hard to use. Thus also, the creation of transitions between different texture classes is complex. We also experimented with omitting one of the texture-controlling parts of the input. That led to a significant problem with overall quality or the inability to generate some texture types.

To eliminate the disadvantages of the first alternative, we simplified the controlling mechanism of the solution by removing the selection part and minimizing the dimensions of latent space. Thanks to the usage of the Siamese network as a feature extractor and its ability to learn similarities between textures (Hudec and Benesova, 2018), the solution synthesized all trained texture classes even though we used only the texture part of the input without the selector part. The controllability of this alternative is based only on changing values of the texture part. By that, we can control the change of texture class and appear-





Figure 8: Visualization of transitions between different subtypes of texture. From top: stone, grass, gravel, bark.

ance of texture and create transitions between neighbouring texture classes in the feature space. However, as shown by the comparison of FID, the alternative that uses the siamese network has lower quality. This problem can be solved by more extended training. Examples of transitions are shown in figure 9.

Experiments also showed that the appearance (not just variability) of the synthesized texture class could also be changed unintentionally by changing the random part of the input. This happens if the interval of the random part is set to  $[0,1]$ . By reduction of the interval, the texture will keep its appearance even if the random part is changed.

## 6 DISCUSSION

Compared to the solution of Li et al. (Li et al., 2017), whose approach takes every image as an individual texture, our approach perceives textures based on the objects they represent. This increases the inner variability of used classes and allows the user to select the exact texture type by class and adjust its appearance by changing the input from latent space.

Latent space as a controllable mechanism was also used in the work of Alanov et al. (Alanov et al., 2020). However, latent space in our approach is obtained from the pre-trained network rather than a network that is trained during the training of the generative model. Thanks to that, we can see how the texture classes will be distributed in latent space and consider whether this layout is suitable for our synthesis intentions or needs to be trained more or affected somehow.

Based on our survey results, we identified that the most unrealistic-looking textures are: Tiles and Pile of rocks. We assume that characteristics of these texture types or features of images used in the training

set could explain the low quality compared to other synthesized texture types.

In the case of tiles, the texture does not belong to a specific group of non-stationary homogeneous textures like most of the dataset. As human-created non-natural objects, tiles are strictly stationary because of their standard layout on roofs. Because of training mainly on non-stationary textures, our solution cannot periodically reconstruct repeating textures. The solution's full potential must be confirmed on a dataset with more stationary textures.

The problem with the texture of the pile of rocks could be caused by internal variation of images in the training set for this class. It means that the dataset contains a mix of images with different sizes and colours of rocks. During training with a subset of the dataset, where the class of rocks contained only one subtype, the model synthesized satisfying results that can be seen in figure 10. However, the texture of bark or stone also contains high internal variability, and we do not observe any problem with output quality. Another explanation behind the low quality of this texture could be that texture that represents multiple objects, like rocks, are more challenging than single-object textures. This could also be a case of low quality behind the Stack of wood that represents the layout of individual logs.

On the other hand, except for all other non-stationary homogeneous textures, the texture of leaves on grass, which is a heterogeneous texture, also achieved a satisfactory look. However, we can notice that leaves lack details like veins or midrib.

Possible improvements in our approach could be based on experiments using the siamese neural network. Because of the different subtypes of texture classes used during training, it can be beneficial to train the siamese network based on the strong similarity (between different classes) and weak similar-

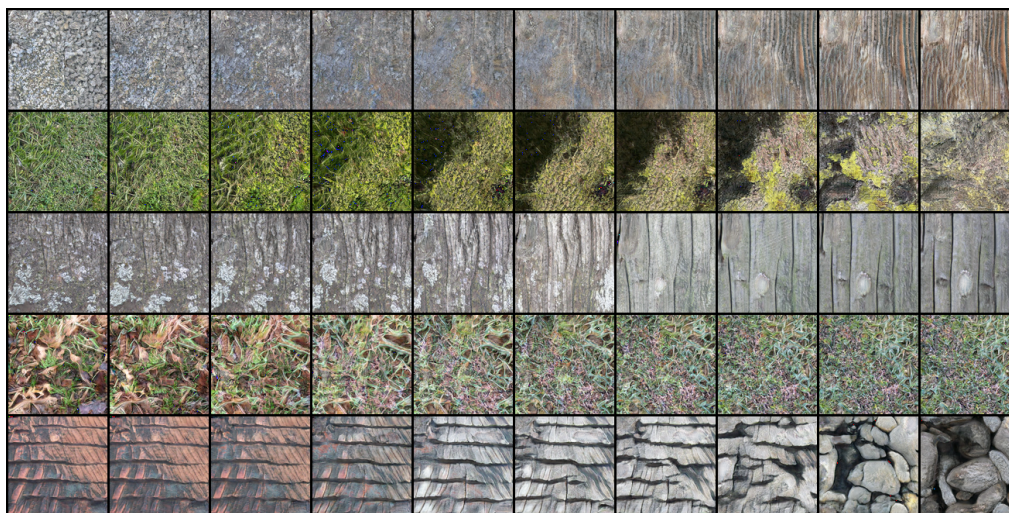


Figure 9: Visualization of transitions between different types of textures. From top: gravel-slab, grass-stone, bark-slab wall, leaves on grass-grass, tiles-pile of rocks.



Figure 10: Example of synthesized piles of rocks from a model trained on a dataset where the class of piles of rocks contained only one type of rock.

ity (between different subclasses of the same texture class). This can help create sub-areas in latent space based on the appearance of textures (for example, sub-area for individual barks, grass, etc.).

An interesting idea is also to increase the dimensional space of the texture part and force the siamese network to distribute the output so that it is possible to transit between every pair of textures. However, this can again lead to complex manual control of the solution. Another improvement could be focused on the tiling of textures or improvement based on adding bump or normal maps to synthesis. That would lead to synthesizing usable high-resolution textures for possible use in rendering or artistic prototyping.

## 7 CONCLUSION

This paper presents an alternative multiclass texture synthesis model based on generative adversarial networks. Our approach is tuned to synthesize non-stationary textures problematically synthesized by traditional approaches. Parts of existing solutions inspire our solution and GAN mechanics to obtain user controllability, preservation of information about selected texture along the entire length of the genera-

tor, training stability, and reduction of mode collapse. In our work, we perceive texture class based on the object it represents, which creates possible subtypes of individual classes. As a generator's input, we use information about the selected class from the latent space obtained by the pre-trained network. This helps us control the synthesized class and its subtype appearance.

We evaluated the results of our approach through the qualitative survey. This evaluation proved that textures synthesized by our solution could fool humans in determining the texture's origin. We also analyzed possible reasons for the low quality of individual classes that can lead to future improvements.

## ACKNOWLEDGMENT

This work was partially supported by STU Program to support Young Researchers and excellent teams of young researchers, and Cooperation (Financial support) with Siemens Healthineers Slovakia.

## REFERENCES

- Alanov, A., Kochurov, M., Volkhonskiy, D., Yashkov, D., Burnaev, E., and Vetrov, D. (2020). User-controllable multi-texture synthesis with generative adversarial networks. pages 214–221.
- Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan.
- Ashikhmin, M. (2000). Synthesizing natural textures. *Symposium on Interactive 3D Graphics*.

- Bergmann, U., Jetchev, N., and Vollgraf, R. (2017). Learning texture manifolds with the periodic spatial GAN. *34th International Conference on Machine Learning, ICML 2017*, 1:722–730.
- Chandra, R., Grover, S., Lee, K., Meshry, M., and Taha, A. (2017). Texture synthesis with recurrent variational auto-encoder.
- Croitoru, F.-A., Hondru, V., Ionescu, R. T., and Shah, M. (2022). Diffusion models in vision: A survey. *ArXiv*, abs/2209.04747.
- Dhariwal, P. and Nichol, A. (2021). Diffusion models beat gans on image synthesis. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc.
- Efros, A. A. and Leung, T. K. (1999). Texture synthesis by non-parametric sampling. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1033–1038 vol.2.
- Gatys, L., Ecker, A., and Bethge, M. (2015a). A neural algorithm of artistic style. *arXiv*.
- Gatys, L. A., Ecker, A. S., and Bethge, M. (2015b). Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. *CoRR*, abs/1505.07376.
- Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in Neural Information Processing Systems*, 3(January):2672–2680.
- Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved training of wasserstein GANs. *Advances in Neural Information Processing Systems*, 2017-Decem:5768–5778.
- Haindl M., F. J. (2013). *Visual Texture*, chapter Motivation. *Advances in Computer Vision and Pattern Recognition*. Springer.
- Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17*, page 6629–6640, Red Hook, NY, USA. Curran Associates Inc.
- Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*.
- Hudec, L. and Benesova, W. (2018). Texture similarity evaluation via siamese convolutional neural network. In *2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 1–5.
- Jetchev, N., Bergmann, U. M., and Vollgraf, R. (2016). Texture synthesis with spatial generative adversarial networks. *ArXiv*, abs/1611.08207.
- Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation.
- Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes.
- Kwatra, V., Schödl, A., Essa, I., Turk, G., and Bobick, A. (2003). Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286.
- Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., and Yang, M. H. (2017). Diversified texture synthesis with feed-forward networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:266–274.
- Liang, L., Liu, C., Xu, Y.-Q., Guo, B., and Shum, H.-Y. (2001). Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.*, 20:127–150.
- Martin, R. and Pomerantz, J. (1978). Visual discrimination of texture. *Perception & Psychophysics*, 24:420–428.
- Pesteie, M., Abolmaesumi, P., and Rohling, R. N. (2019). Adaptive Augmentation of Medical Data Using Independently Conditional Variational Auto-Encoders. *IEEE Transactions on Medical Imaging*, 38(12):2807–2820.
- Portilla, J. and Simoncelli, E. (2000). A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40.
- Praun, E., Finkelstein, A., and Hoppe, H. (2000). Lapped textures. In *Proceedings of ACM SIGGRAPH 2000*, pages 465–470.
- Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks.
- Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training GANs. *Advances in Neural Information Processing Systems*, pages 2234–2242.
- Shin, S., Nishita, T., and Shin, S. Y. (2006). On pixel-based texture synthesis by non-parametric sampling. *Computers and Graphics (Pergamon)*, 30(5):767–778.
- Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*.
- Wei, L.-Y. and Levoy, M. (2000). Fast texture synthesis using tree-structured vector quantization. *Computer Graphics (Proceedings of SIGGRAPH'00)*, 34.
- Zhou, Y., Zhu, Z., Bai, X., Lischinski, D., Cohen-Or, D., and Huang, H. (2018). Non-stationary texture synthesis by adversarial expansion. *ACM Transactions on Graphics*, 37(4).
- Zhu, J.-Y., Park, T., Isola, P., and Efros, A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. pages 2242–2251.