# Example-Based Stylization of Navigation Maps on Mobile Devices

Ondřej Texler*
*Supervised by: Daniel Sýkora*†

Department of Computer Graphics and Interaction
Faculty of Electrical Engineering
Czech Technical University
Prague / Czech Republic

## Abstract

In this paper, we present a system which allows fast transfer of artistic style from a hand-drawn exemplar to a navigation map. We demonstrate how to reduce the computational overhead of current example-based style transfer techniques by using fast bitmap operations which replace texture synthesis at places where it is not necessary to satisfy high-quality results. In addition, we also demonstrate how to utilize parallel processing and hardware acceleration on the GPU to further reduce the computational overhead and achieve interactive response on a mobile device.

**Keywords:** Stylization, Texture Synthesis, Example-Based, Occurrence Map, Parallelization, Mobile Device

## 1 Introduction

Example-based stylization is a complex problem. In modern computer science, there has been a lot of research in this field, and remarkable progress in both quality and speed of a texture synthesis has been made.

Many state-of-the-art stylization approaches work well on complex images, for example, photos, but fail on simple images like a screenshot from a map. Mainly convolutional neural network approaches suffer from this problem, see Figure 3. In a complex scene and a complex artistic style, it is easy to hide some glitches without notice. Based on the analysis of related and recent work, details can be found in the Section 2, StyLit [7] is currently the best and the most suitable technology for map stylization. This paper deals with adopting and extending the StyLit approach to navigation maps.

Texture synthesis has many artistic and entertainment applications. The visual quality often needs to be compromised, when the available computional budget is low. It is especially case of interactive computer games and mobile phones. When it comes to the quality of the synthesis, the StyLit approach works well. However, it is not fast enough to obtain an interactive response on mobile devices.

We present a new method reducing computational overhead by replacing stylization of some parts of the image by fast bitmap operations. Texture synthesis based on the StyLit approach is integrated into the existing mobile navigation Dynavix as a prototype. It allows the user to create his own map style, typically by painting on a paper and scanning it or by using any graphics editor. This custom map style is then used to stylize the given navigation map. Details of integration are also provided. Figure 1 shows example of map stylization, where the image *B* shows a real screenshot from the Dynavix navigation application and the image *B'* shows the final stylized result.

Organization of the paper is following. First, related work and current state-of-the-art approaches are mentioned briefly. Next, the texture synthesis is described in general and defined more formally followed by a solution of the texture synthesis with all previously mentioned improvements. Finally, implementation and integration into Dynavix is described and results are presented.

## 2 Related Work

In recent years, many different approaches dealing with digital paintings creation have been published. A generic example-based technique *The Lit Sphere* was introduced by Sloan et al. [19]. A shaded sphere painted by an artist is used as the style example and pixels from this exemplar are transferred to the target 3D model using texture mapping. Patch-based synthesis with more complex illumination guidance was proposed in StyLit [7].

*Image Analogies* is a concept proposed by Hertzman et al. [10]. There are two pairs of images, source images and target images, one image in the pair is filtered-stylized and one is unfiltered. The stylization is described by the source image pair. The algorithm iterates over unfiltered target pixels and finds the most similar location in the unfiltered source image and transfers the look from the filtered counterpart. However, this approach suffers from introducing visible seams, repetition and other artefacts and does not preserve the visual appearance of used art tool.

Another example-based region-growing approach is by Efros et al. [6], where the new texture is generated one

*texleond@fel.cvut.cz
†sykorad@fel.cvut.cz

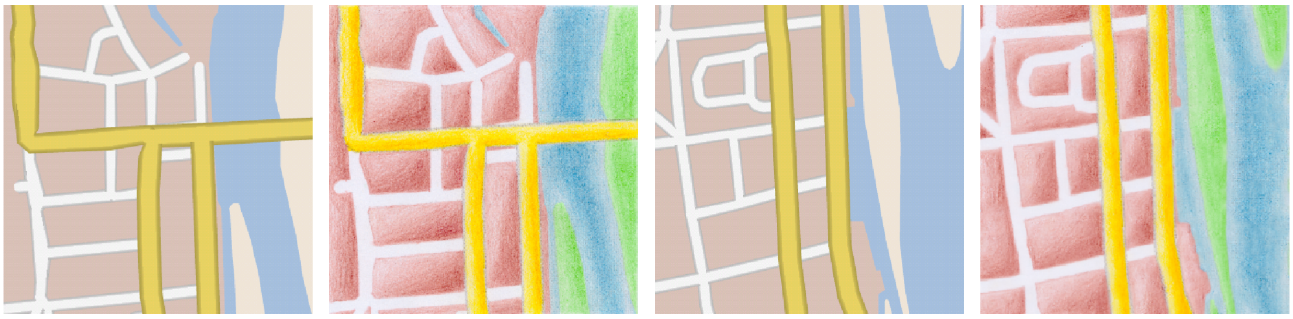


Figure 1: Real demonstration of navigation map stylization. Both A and B are screenshots from Dynavix. A' is a crayon drawing made by a human and scanned. B' is the result of the synthetization algorithm.

pixel at a time. Given a sample texture image, a new image is initialized by a patch from the sample texture and this patch in the new texture is grown pixel by pixel. Another approach is proposed also by Efros et al. [5]. The new texture is not growing one pixel at a time but one patch at a time. New pixels or patches in both previous approaches are determined based on their similarity with the sample texture.

Kwatra et al. [15] and Wexler et al. [20] introduced a texture optimization technique. They defined a similarity metric for measuring the quality of synthesized textures with respect to a given input sample by Markov Random Field. They then formulated the synthesis problem as a minimization of an energy function by using expectation-maximization-like algorithm. In contrast to the region-growing approaches, it is not synthesized one pixel at a time or one patch at a time, but the whole new texture is refined many times in iterations, from randomly initialized, through coarse texture to the sharp, fine and faithfully-looking texture.

In recent work of Fiser et al. [8] and of Barnes et al. [2], the original Hertzman's [10] algorithm was replaced by the texture optimization technique. This new approach is ideally suited for the controllable synthesis of textures and other adjustments, but it suffers from the so-called wash-out effect described by Newson et al. [16] and examined in more detail by Jamriška et al. [11]. The wash-out effect makes parts of the new texture smoothed or blurry. This undesired effect is caused by an extensive use of a small amount of the same or similar patches from which a new texture is composed. Many strategies have been developed to deal with the wash-out effect, e.g. color histogram matching by Kopf et al. [14], and bidirectional similarity by Simakov et al. [17]. These approaches work well in case of the source being mostly stationary without many nearly-homogeneous patches. Unfortunately, it does not work well on realistic style examples. More robust mitigation of the wash-out effect by encouraging uniform patch usage was recently published by Kaspar et al. [13] and Jamriška et al. [11].

Recently, many alternative approaches were developed to achieve computer-assisted stylization by using Neural Networks. Artistic Style [9] uses a deep convolutional neural network VGG [18] trained for object recognition, the VGG-Network is used for extracting information from a texture, it can extract information about both style and content. Texture synthesis is represented as the minimization problem solved by an iterative gradient descent. This approach has impressive results in some cases. Since the VGG network is trained on natural images, it does not work well on images with synthetic appearance - like computer-generated maps. Extension of this approach, along with the original Hertzman Image Analogy [10] concept, results in the Deep Image Analogy [12], that uses VGG as well.

## 3 Texture Synthesis

At first in this section, the Image Analogy concept is defined. Next, we focus on the texture synthesis in general, various texture synthesis methods and types are presented. The section also contains a formal definition of the problem, the definition of an energy and a description of global optimization approach.

### 3.1 Image Analogies

Image Analogies is a concept originally defined by Aaron Hertzman [10] and nowadays it is widely used to define example-based guided texture synthesis. It describes relation (analogy) between two pairs of images, see Figure 1. The first pair are the images $A$ and $A'$, the second pair are the images $B$ and $B'$. There is an analogy between the images $A$ and $B$ and between $A'$ and $B'$, meaning the image $A'$ has the same relation to the image $A$ as the image $B'$ to the image $B$. Hertzmann in Image Analogies [10] likens the problem of synthesis to the filtering and defines, that given a pair of images $A$ and $A'$, where $A$ is unfiltered and $A'$ is a filtered source image, along with some additional unfiltered target image *B*, a new filtered target image *B'* is synthesized.

The Hertzman's definition means that we have an example of some unknown filtration *f(A) = A'* which transforms the image *A* to *A'*. Since *A* and *A'* are known, filtration *f* can be reconstructed and applied to the another unfiltered image *f(B) = B'*. Image *B'* completes the analogy. In this paper we will also refer to the example image *A* as sample guide, the stylized image *A'* as sample style, the image *B* as output guide and the stylized image *B'* will be called output.

## 3.2 Texture Synthesis Taxonomy

The research in texture synthesis was quite active in the last two decades, and since there are many texture synthesis approaches, we can classify them into the following categories. Example-based vs. Procedural, Region-Growing vs. Global Optimization and Guided vs. Non-Guided texture synthesis.

From the texture generating point of view, we can distinguish between the example-based and the procedural approach. In case of procedural texture synthesis, the new texture is generated using predefined parts and methods following predefined patterns. For example, a texture is composed of a predefined set of brush strokes and polished using particular filters and effects. The Example-based approach is more generic, it tries to infer the generative procedures based on a set of examples, meaning it does not depend on the specific algorithm. An example of a style has to be provided and the new image is created based on this example.

We can also categorize texture synthesis approaches into the Region-Growing category, where the new texture grows one pixel or one patch at a time, or into the Global Optimization category, where the whole texture is refined in multiple iterations, from coarse initialization to a finite and faithfully looking texture. Global optimization approaches allow controlling the texture synthesis process more intuitively.

We can distinguish between guided and non-guided synthesis. In case, we synthesize simple texture in order to make it bigger, no guide with extra information of the content is needed, so it is called normal or non-guided texture synthesis. If more complex image is synthesized, guide images describing a content of the sample image and output image needs to be provided. Guide images allows texture synthesis distinguish between different parts of the image, so it is called guided synthesis.

Based on this categorization, the technique described in this paper falls into the Example-based, Global Optimization and Guided texture synthesis categories.

## 3.3 Global Optimization

This approach was originally published by Wexler [20] and later by Kwatra [15]. They define texture synthesis as the problem of energy minimization using the expectation-maximization-like algorithm. The energy function *E* of a texture *B'* with respect to the sample style *S* is defined as follows:

$$E = \sum_{x \in B'} \min_{s \in S} [SSD(x,s)] \tag{1}$$

In other words, energy *E* is the sum of distances of each patch from texture *B'* to its closest patch in texture *S*. In this paper, term *patch* means small squared region of pixels usually of size 5 x 5 pixels. The distance between two patches is computed using the Sum of Squared Distances (SSD) similarity measure method. As can be seen, energy *E* would be zero if for each patch from texture *B'*, a perfect match in the texture *S* is found. We will see later, that this definition of energy is not robust enough.

First of all, the Nearest-neighbor field (*NNF*) needs to be defined. It describes the relation between two textures. Let *A'* and *B'* be textures (e.g., *A'* can be our sample style texture and *B'* can be our output texture). Let $\varphi$ be some patch similarity metric (i.e., *SSD*). We can define *NNF* as follows:

$$NNF_{B' \to A'}(P) = \min_{Q \in A'} [\varphi(P,Q)] \tag{2}$$

*P* is an arbitrary patch from image *B'*. *NNF* is the mapping of all patches from image *B'* to some patches of image *A'*. This mapping is not injective, multiple patches from image *B'* can be mapped to one particular patch from image *A'*.

The Expectation Maximization algorithm is used for optimization in circumstances where both, the desired variables, and the parameters of the energy function being optimized are unknown. In terms of Expactation Maximization algorithm, image *B'* is the desired variable, while the *NNF* represents parameters. The estimation of image *B'* by refining it in multiple iterations in order to minimize the texture energy *E* corresponds to the *E-step*, while finding the *NNF* corresponds to the *M-step*.

In other words, given the sample texture, the new texture is synthesized. As long as the new texture does not look similar as the sample texture, the new texture has high energy and it is changed in a way to look more similar as the sample texture, meaning it is changed in such a way to have lower energy. In each iteration, each pixel from the new image is evaluated as the average of several pixels from the sample texture. Generally, more iterations lead to better results.

## 3.4 Problem Formulation

We will define the energy exactly how Kwatra [15] defines it using the *NNF* and we will extend it two times. First, we will augment this definition by guide images. Second, the definition will be augmented to mitigate the wash-out effect.

Suppose we have textures denoted as in Figure 1, $\varphi$ is some similarity measure method, energy $E$ using the *NNF* is defined as follows:

$$E = \sum_{s \in B'} \varphi(s, NNF^{\varphi}_{B' \to A'}(s)) \qquad (3)$$

This is the energy definition by Kwatra [15]. Since Kwatra's approach is not guided synthesis it does not deal with sample guide *A* and output guide *B*. This means, we have to extend this energy definition by guide images. $NNF^{\varphi}_{B \to A}$ is *NNF* coputed on guide images. $NNF^{\varphi}_{B' \to A'}$ returns style patch and $NNF^{\varphi}_{B \to A}$ returns guide patch. To simmplify a notation, we will use $NNF_s$ instead of $NNF^{\varphi}_{B' \to A'}(s)$ for style Nearest-neighbor field and $NNF_g$ for guide Nearest-neighbor field respectively.

Following is definition of texture energy $E$ extended by guide images:

$$E = \sum_{s \in B} \alpha \cdot \varphi(s, NNF_s) + \beta \cdot \varphi(s, NNF_g) \qquad (4)$$

Parameters $\alpha$ and $\beta$ are optional and can be used to make synthesis follow more guide than style or follow more style than guide.

Kwatra's original definition of energy is not robust enough and suffers from the so-called *wash-out effect* [16], an undesired effect which occurs when a small amount of same or similar patches from the sample texture *A'* are used too excessively when a new output texture *B'* is synthesized. To mitigate the wash-out phenomenon, Kaspar et al. [13] defines patch penalization. Energy function augmented by the patch penalization is following:

$$E = \sum_{s \in B'} \alpha \cdot \varphi(s, NNF_s) + \beta \cdot \varphi(s, NNF_g) + \lambda \cdot \frac{\Omega_{occ}}{\omega_{best}} \qquad (5)$$

Our energy definition is now the difference between patches in style plus the difference between patches in a guide plus patch penalization. $\Omega_{occ}$ returns the count of how many times was a particular patch used so far (its pixels respectively). $\omega_{best}$ is the ideal number of usage of this particular patch, it is computed automatically using the guide images, details are described in Section 3.6. Parameter $\lambda$ determinates how big is the penalization for patches which are used more often than they should be.

## 3.5 Texture Synthesis Algorithm

See Algorithm 1. Significant improvements in both speed and quality are achieved by using a multi-resolution approach. The sample texture, as well as both guide textures, are downsampled to half of their original resolution a few times, according to the parameter *LEVELS*. The synthesis then starts from the lowest resolution. Multiple iterations, given by the parameter *N*, are performed on each resolution level. When transitioning to a higher resolution level, the output texture is upsampled.

At first, output image *B'* is initialized by random colors, then it is refined in multiple iterations, upsampled and refined again to a fine and faithfully looking texture. In some literature, this iterative approach is called "coarse-to-finite". In each iteration, *NNF* is computed and each pixel of the new output texture is evaluated based on the mentioned *NNF* using the so-called *voting* method. In voting method, all patches from *NNF* are stacked on each other and its corresponding pixels are averaged in order to create new texture. Parameters *A*, *B*, *A'* and variable *B'* are images according to the image analogy, Figure 1.

**Algorithm 1** Texture Synthesis - Multi-Resolution

```
function TextureSynthesis(A, B, A', N, LEVELS)
$B' = random\ colors$
for lvl = 0 : LEVELS do
    $A_{\downarrow} = Downsample(A, 2^{LEVELS-lvl})$
    $B_{\downarrow} = Downsample(B, 2^{LEVELS-lvl})$
    $A'_{\downarrow} = Downsample(A', 2^{LEVELS-lvl})$
    for n = 0 : N do
        $NNF = findNNF(A_{\downarrow}, B_{\downarrow}, A'_{\downarrow}, B')$
        $B' = voting(NNF, A'_{\downarrow})$
    end for
    if lvl < LVLS then
        $B' = upsample(B', NNF)$
    end if
end for
return B'
end function
```

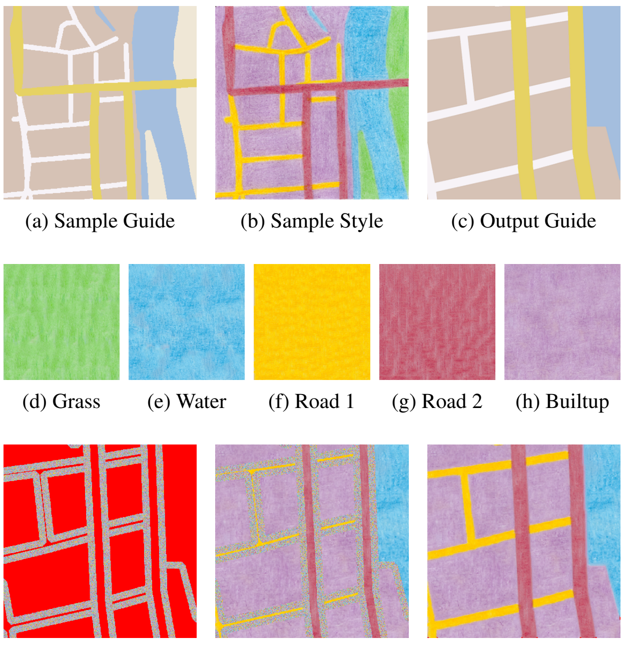
## 3.6 Map Segments Guidance

Since we do not synthesize only one texture but the entire image with more complex content, guidance is needed to distinguish between different parts of the synthesized image. A computer-generated map usually consists only of a small amount of regions and therefore only from a small amount of colors. Guide images in Figure 1 consist from five different colors - blue color for water, yellow color for bigger roads, white color for smaller roads, pink color for built-up areas and sand color for grass areas. This inherit segmentation of computer-generated maps is ideally suited to be used as the guide for synthesis. Later in this section, the term "segment" is used to refer to a map component like water, road, built-up area, etc.

In Eq. 5, parameter $\omega_{best}$ is mentioned. Finding a good value of $\omega_{best}$ in general case is a very complex problem. In our case, $\omega_{best}$ can be computed separately for each segment as follows:

$$\omega_{best}(segment) = \frac{output[segment].size}{sample[segment].size} \qquad (6)$$

If $\omega_{best}$ is set correctly, *NNF* will probably assign patches between same segments, for example, most of the water patches from the output guide will be assigned to water patches from the sample guide.



(a) Sample Guide (b) Sample Style (c) Output Guide

(d) Grass (e) Water (f) Road 1 (g) Road 2 (h) Builtup

(i) Mask (j) Initialization (k) Final result

Figure 2: Illustration of the Pre-synthesized Textures Speed-Up. At the top row, input images A', A and B are shown. The middle row contains pre-synthesized textures, and the bottom row demonstrates synthetization process.

## 3.7 Pre-Synthesized Textures Speed-Up

The novelty that we bring into the texture synthesis is reducing the computational overhead by replacing expensive synthesis operations by fast bitmap operations. As we describe in the Section 3.6, computer-generated maps consist of a small amount of segments, therefore, there is a possibility to pre-generate the texture of each segment and use these textures to speed-up the synthesis, see images (d) to (h) at Figure 2. Pre-synthesized textures are part of the input and are created using the texture synthesis algorithm described in Section 3.5.

At first, a mask around the edges of the output guide is computed, see image (i) at mentioned figure. Areas near to the edges are initialized with random colors, areas far from the edges are initialized by pre-synthesized textures based on the content of the output guide, see image (j). Since the texture synthesis approach used in this paper is the global optimization method, this coarse composed image is then optimized in multiple iterations in order to make borders between segments faithfully-looking, see final result (k). Synthesis is applied only around the edges, therefore a significant speed-up is achieved without losing the quality.

# 4 Implementation

Our ultimate goal was to integrate texture synthesis into the mobile navigation application Dynavix and optimize stylization to work well and fast on navigation maps. Beside using pre-synthesized textures described in the section above, another significant speed-up is achieved by parallelization of the most time-consuming parts on the GPU using OpenCL standard.

## 4.1 Dynavix Integration

Implementation of this paper was integrated into an existing mobile navigation Dynavix as a prototype. Dynavix offers GPS navigation and maps for Android devices. In this section, we will describe the details of this integration.

Dynavix uses OpenGL technology for both map and map widget rendering, and the Android Framework for the rest of its user interface. In the first phase, a map is rendered using OpenGL. In the second phase, map widgets are rendered over the previously rendered map, also by OpenGL. In the third phase, the remaining Android UI is rendered over it. The prototype of map stylization was inserted between the first and the second phase. See pseudocode 2 of initializing and running a map stylization in Dynavix.

**Algorithm 2** Dynavix Main

```
function DynavixMain()
textureSynthesis = create TextureSynthesis object
textureSynthesis.loadSampleAndSampleGuide(...)
textureSynthesis.loadPreSynthesizedTextures(...)
while Dynavix is running do
    Render Map by OpenGL
    B = Read pixels from OpenGL bufffer
    B' = textureSynthesis.synthesis(B)
    Write B' to OpenGL buffer
    Render Map Widgets
    Render Android Framework UI'
end while
end function
```

## 4.2 GPU Parallel Acceleration

The most time-consuming part of texture synthesis is finding the Nearest-Neighbor field. Texture synthesis runs in multiple iterations on multiple resolution levels and in each iteration, *NNF* is computed. *NNF* matches each patch from output texture *B'* to the closest patch (with respect to the patch penalization) in sample style *A'*. In our case *NNF* is computed by using a brute-force-like algorithm, meaning for each patch from *B'*, the entire image *A'* is searched. Suppose that the width and height of both images *A'* and *B'* is $n$, thus complexity of the *NNF* computing is $O(n^4)$.

For simplification, suppose we are not dealing with patch penalization, meaning this brute-force-like algorithm is ideally suited for parallelization. For each patch from image *B'*, texture *A'* can be searched independently. During computing of the *NNF*, texture *A'* is accessed only for reading, so there is no need for any explicit synchronization and the process can be naturally parallelized.

## 5 Results and Comparison

This section starts with a comparison of the quality of our synthesis implementation with nowadays very popular convolutional neural networks. Next, several experiments with the pre-synthesized texture improvement followed by experiments with infinite zooming demonstrate how wide the usage of texture synthesis really is.

### 5.1 Comparison with CNN Approaches

In past years, there has been a big expansion of the convolutional neural networks into the field of image stylization. Convolution neural networks have impressive results if they are used for the stylization of complex paintings. In case of simple style and simple guide, most of neural network approaches do not work well. Figure 3 shows a comparison of our results with three neural approaches. From left to right: Neural Doodle [4], Artistic Style[9] and Deepart.io [3]. Implementations of Neural Doodle and Artistic Style are available as python code on GitHub and Deepart.io has a web page application. The image used as the output guide is the same as image *B* on Figure 1. As can be seen, our result is significantly better than any other result. According to the Deepart.io web application [3], it has good results when the sample style and output guide are more complex, however in our simple map scenario, Deepart.io results are insufficient. Artistic Style completely failed to capture colors, while Neural Doodle failed to capture the content.

### 5.2 Pre-Synthesized Textures Experiments

In case of the pre-synthesized improvement, the output image is initialized with pre-generated textures and only areas around the edges are synthesized. The quality of the result is given by the size of the synthesized area around the edges. Pre-synthesis only has effect when the marked area around the edges is positive and does not cover the whole or most of the image. Figure 4 shows multiple results for different sizes of the synthesized area around the edges. As can be seen, results (a), (b) and (c) look almost the same, meaning we need to synthesize only a really small area around the edges to get a faithfully-looking result. Optimal mask thickness depends on the particular style, we consider mask of size 2px or 4px to be sufficient for most of the styles.

Table 1 shows computational times for different mask thickness. Four of the resulting images are shown at Figure 4. Resulting images has size 360 x 360 px, and measurements were performed on a single core CPU. As can be seen, synthesis using pre-synthesized improvement is orders of magnitude faster than full image synthesis.

Table 1: *Pre-synthesized speed-up*

| **Mask Thickness** | **Time** |
|---|---|
| Full image | $\sim$10 minutes |
| 8 px | 20859 ms ($\pm 100ms$) |
| 4 px | 9342 ms ($\pm 100ms$) |
| 2 px | 2376 ms ($\pm 100ms$) |
| 0 px | 2 ms ($\pm 1ms$) |

### 5.3 Zoom Experiments

Texture synthesis, as described in this paper, has a wide use. It can be used, for example, to achieve *infinite* zooming on a texture. Given the sample style, texture is synthesized on each zoom level, meaning the zoomed texture has the same texture appearance and quality as the original texture. See examples on Figure 5. Zoom is performed without losing the quality and resolution of the texture. Light-blue areas on the water may look like glitches, but they are a part of the style. In the original texture, there is light-blue water near the shore and dark-blue water in the center of the river. However, guide images do not distinguish between them, meaning the texture synthesis algorithm cannot differentiate between them.

## 6 Limitations and Future Work

Still, some options for future improvement are available. The algorithm can be improved in multiple ways and its implementation can be more effective. Although finding *NNF* is, in the case of the pre-synthesized improvement, computed only around the edges, there is still a possibility to achieve further speed improvement by using an approximative method [1]. Moreover, working with a mask during the *NNF* computing could be done more efficiently. Although this implementation uses OpenCV, only a few and very basic structures and functions from this library are actually used. By removing OpenCV, implementation can become even more independent. Many other minor things could also be improved, but mentioned were the most fundamental.

## 7 Conclusions

In the course of this paper, we have begun with a description of the stylization problem in general. We extended Kwatra's [15] optimization approach and definition of texture energy two times. At first, energy was extended from non-guided texture synthesis to guided texture synthesis. Second, energy was extended by Kaspar's [13] occurrence map $\Omega$ in order to mitigate the wash-out effect [16].

We then presented a basic method to solve the guided texture synthesis problem. It was further extended by map segment guidance based on the content of the stylized image and computation of $\Omega_{best}$ was introduced. The main

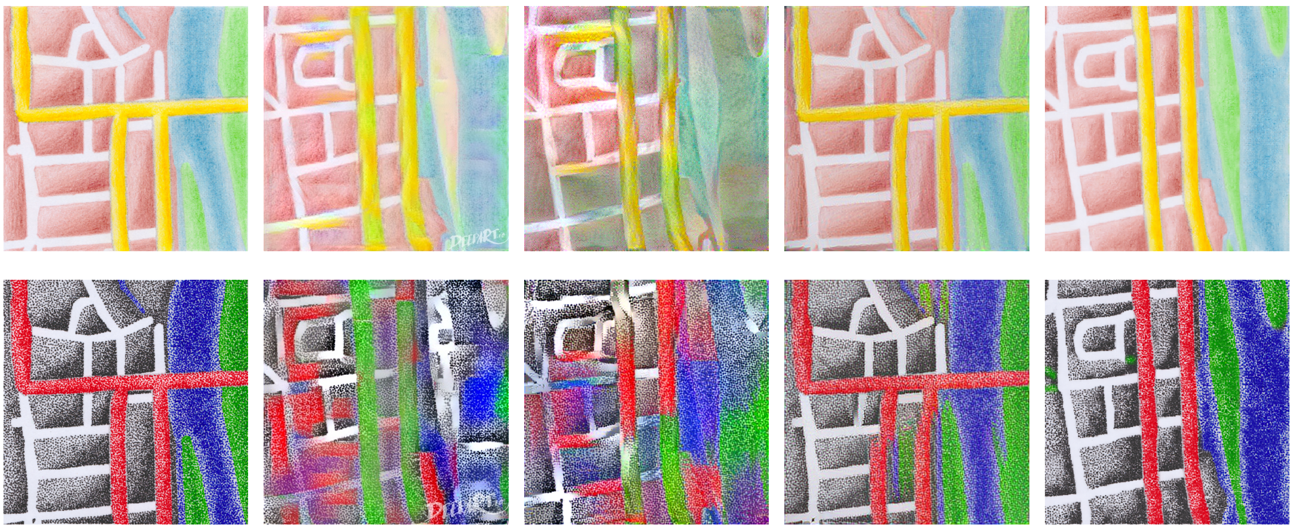

(f) Style (g) Deepart.io (h) Artistic Style (i) Neural Doodle (j) Ours

Figure 3: Comparison with CNN approaches - crayon style and pen style. The top left column shows input styles. Middle three columns show the results of three CNNs. And top right column is the result of our implementation.

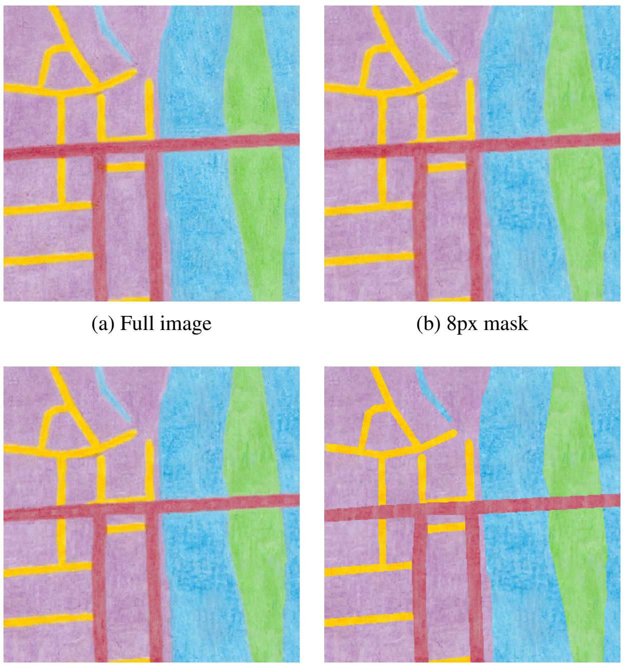

(a) Full image (b) 8px mask (c) 4px mask (d) No mask

Figure 4: Pre-synthesized improvement - different sizes of the synthesized area around the edges. Full image (a), 8 px (b), 4 px (c), no mask (d).

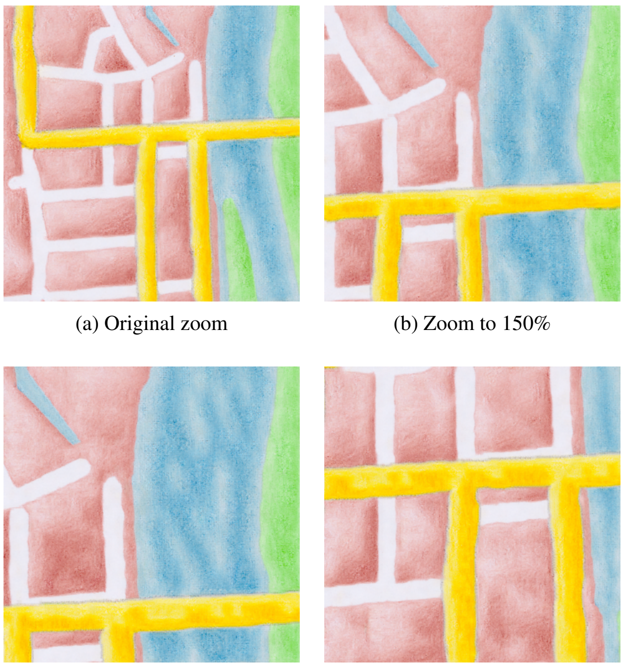

(a) Original zoom (b) Zoom to 150% (c) Zoom to 200% water (d) Zoom to 200% road

Figure 5: Infinite zooming. (a) shows original texture. (b) shows zoom to 150%. (c) shows zoom to 200% on the water region and (d) shows 200% zoom on the road.

contribution of this paper is an explanation of speed-up using pre-synthesized textures. It allows to synthesize only necessary parts of the image and parts which were once synthesized can be completely reused.

The texture synthesis algorithm was integrated into the Dynavix GPS navigation as a prototype. Details of integration are presented as well. A similar procedure can be used to integrate texture synthesis into other applications and pipelines, for example, computer games.

Finally, the results are presented and compared with other stylization approaches and implementations. Our results have significantly better quality in map stylization in most cases. Moreover, computational time is substantially lower, and quality is still within a satisfactory range. Our implementation does not have any dependencies on additional resources, like neural networks, databases, etc., meaning it is ideally suited for integration into mobile navigation applications or any other pipeline or device.

## 8 Acknowledgements

This research was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS16/237/OHK3/3T/13 (Research of Modern Computer Graphics Methods).

## References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. *PatchMatch: A randomized correspondence algorithm for structural image editing*. ACM Transactions on Graphics 28(3), 24, 2009.

[2] C. Barnes, F.-L. Zhang, L. Lou, X. Wu, and S.-M. Hu. *PatchTable: Efficient patch queries for large datasets and applications*. ACM Transactions on Graphics 34(4), 97, 2015.

[3] M. Bethge, A. Ecker, L. Gatys, L. Kidziński, and M. Warchol. Deepart.

[4] A. J. Champandard. *Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artwork*. Computing Research Repository, abs/1603.01768, 2016.

[5] A. A. Efros and W. T. Freeman. *Image Quilting for Texture Synthesis and Transfer*. SIGGRAPH Conference Proceedings, pp.341-346, 2001.

[6] A. A. Efros and T. K. Leung. *Texture Synthesis by Non-Parametric Sampling*. International Conference on Computer Vision, pp.1033-1038, 1999.

[7] J. Fišer, O. Jamriška, M. Lukáč, E. Shechtman, P. Asente, J. Lu, and D. Sýkora. *StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings*. ACM Transactions on Graphics, 35(4), 2016.

[8] J. Fišer, M. Lukáč, O. Jamriška, Y. Gingold, P. Asente, and D. Sýkora. *Color Me Noisy: Example-based rendering of hand-colored animations with temporal noise control*. Computer Graphics Forum 33(4), pp.1-10, 2014.

[9] L. A. Gatys, A. S. Ecker, and M. Bethge. *A Neural Algorithm of Artistic Style*. Computing Research Repository, abs/1508.06576, 2015.

[10] A. Hertzmann, Ch. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. *Image Analogies*. SIGGRAPH Conference Proceedings, pp.327-340, 2001.

[11] O. Jamriška, J. Fišer, P. Asente, E. Shechtman, and D. Sýkora. *LazyFluids: Appearance Transfer for Fluid Animations*. ACM Transactions on Graphics 34(4), 92, 2015.

[12] L. Jing, Y. Yuan, Y. Lu, H. Gang, and B. K. Sing. *Visual Attribute Transfer through Deep Image Analogy*. Computing Research Repository, abs/1705.01088, 2017.

[13] A. Kaspar, B. Neubert, D. Lischinski, M. Pauly, and J. Kopf. *Self Tuning Texture Optimization*. Computer Graphics Forum 34(2), pp.349-360, 2015.

[14] J. Kopf, C. W. Fu, D. Cohen-Or, O. Deussen, D. Lischinski, and T. T. Wong. *Solid texture synthesis from 2D exemplars*. ACM Transactions on Graphics 26(3), 2, 2007.

[15] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. *Texture optimization for example-based synthesis*. ACM Transactions on Graphics 24(3), pp.795-802, 2005.

[16] A. Newson, A. Almansa, M. Fradet, Y. Gousseau, and P. Pérez. *Video inpainting of complex scenes*. SIAM Journal of Imaging Science 7(4), pp.1993-2019, 2014.

[17] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani. *Summarizing visual data using bidirectional similarity*. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2008.

[18] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Computing Research Repository, abs/1409.1556, 2014.

[19] P. P. J. Sloan, W. Martin, A. Gooch, and B. Gooch. *The Lit Sphere: A model for capturing NPR shading from art*. Proceedings of Graphics Interface, pp.143-150, 2001.

[20] Y. Wexler, E. Shechtman, and M. Irani. *Spacetime completion of video*. IEEE Transactions on Pattern Analysis and Machine Intelligence 29(3), pp.463-476, 2007.