FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Stylizing Vector Animation Using Natural Dynamics of Artistic Media
Student:	Bc. Adam Platkevič
Supervisor:	prof. Ing. Daniel Sýkora, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2020/21

Instructions

Explore current state-of-the-art in the field of automatic transfer of appearance and movement into a video that is based on non-linear optimization, generative neural networks, and meta-learning [1, 2, 3, 4].
Design an algorithm that can mimic motion in the target vector animation using dynamic behavior of artistic media captured by time-lapse recording (e.g., watercolor diffusion).
Try to extend the method to handle several different dynamic effects (e.g., drying or crumbling).
Verify the functionality of the developed algorithm on various vector animations supplied by the supervisor of the thesis.

References

[1] Jamriška et al.: LazyFluids: Appearance Transfer for Fluid Animations, ACM Transactions on Graphics 34(4):92, 2015.
[2] Jamriška et al.: Stylizing Video by Example, ACM Transactions on Graphics 38(4):107, 2019.
[3] Wang et al.: Video-to-Video Synthesis. Proceedings of Advances in Neural Information Processing System, pp. 1144–1156, 2018.
[4] Wang et al.: Few-shot Video-to-Video Synthesis, Proceedings of Advances in Neural Information Processing Systems, pp. 5014–5025, 2019.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 14, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Stylizing Vector Animation Using Natural Dynamics of Artistic Media

Bc. Adam Platkevič

Department of Theoretical Computer Science
Supervisor: prof. Ing. Daniel Sýkora, Ph.D.

January 7, 2021

Acknowledgements

First and foremost, I would like to sincerely thank my supervisor Daniel Sýkora for his invaluable advice and active help throughout my work on this thesis. Also, I would like to thank him for arranging financial support from a grant from The Grant Agency of the Czech Technical University in Prague (grant No. SGS19/179/OHK3/3T/13 (Research of Modern Computer Graphics Methods)), which allowed me to devote more time to the thesis's realization.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on January 7, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Platkevič, Adam. *Stylizing Vector Animation Using Natural Dynamics of Artistic Media.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

V této práci představujeme vlastní metodu pro automatickou stylizaci vektorových animaci s použitím vzhledu běžných uměleckých médií, primárně zaměřenou na akvarely. Naším hlavním cílem je co nejlépe zachovat vizuální charakteristiky dynamického chování daného média.

Nejdříve poskytujeme stručný přehled již publikovaných metod pro přenos stylu a uměleckou stylizaci statických obrázků a videa. Následně podrobně popisujeme přístup ke stylizaci založený na přenosu spojitých regionů z předlohy. Mimo to se zaměřujeme i na některá důležitější příbuzná témata a řešení častých problémů. Nakonec formulujeme vlastní přístup k řešení stanoveného úkolu založený na předchozích konceptech, a vyhodnocujeme jeho výstupy. Zahrnujeme i popis některých implementačních detailů a kroků ke zrychlení implementovaného programu. Soudíme, že přes některé nedostatky, rovněž zmíněné v této práci, poskytuje navrhovaná metoda přesvědčivé výsledky odpovídající zadání.

Klíčová slova přenos stylu, animace, syntéza textury, dle předlohy, umělecká stylizace

Abstract

In this thesis, we present a novel approach to the automatic stylization of vector animation with the appearance of artistic media, primarily focused on, but not limited to, watercolor. The main goal is to preserve the visual characteristics of the given media's dynamic behavior as much as possible.

We provide a quick overview of previously published methods for style transfer and artistic stylization of still images and video. Subsequently, we describe the patch-based approach to example-based stylization in detail. We also focus on several important concepts related to this approach and solutions to common problems. Finally, we formulate our approach to the given task based on those concepts and evaluate its results. Details of our implementation and its optimization are included. We conclude that despite several shortcomings also mentioned in this thesis, our method produces compelling results in line with the set goal.

Keywords style transfer, animation, texture synthesis, example-based, artistic rendering

Contents

List of Figures

Introduction

For several decades, 2D animation has been a vital part of popular multimedia and art. With the rising utility of computers, a number of software solutions have emerged, that allow artists to design and animate a scene in terms of vector graphics. Both vector-based animation and earlier animation drawn by hand frame by frame often share common aspects to their visual style. One of primary components of typical cartoon-like animation are larger contrasting regions painted with flat color or smooth shading. A way to add to the visual appeal of such animation is to slightly deviate from this common look and add a fine texture or generally a different style to these regions. The goal of this thesis is motivated by the need of an automated approach to such stylization, as doing it by hand is too time consuming with a reasonably large sequences.

Automatic stylization of various imagery, including computer-generated animation, is a widely and actively researched topic in computer science. In this thesis, we choose to explore example-based approach to stylization of animation. That means not mimicking a single particular style, but rather transferring the look of an additional input – an exemplar image sequence – to the target animation. The field of example-based texture synthesis and image stylization has seen remarkable advancements in the past years, both in terms of the visual quality of the results and the computation speed. We build upon the state-of-the-art and formulate a method focused on example-based stylization of a flat-colored image sequence with the look of watercolor or other common artistic media, while maintaining their characteristic appearance as closely as possible.

Compared to still image stylization, dealing with continuous animation brings several additional problems to be addressed. Particularly maintaining temporal continuity (most commonly referred to as *temporal coherence*) of the stylized image sequence is of great interest. When each frame of a sequence is synthesized independently, there is typically some amount of temporal noise present. We are familiar with this effect from traditional hand-colored animation, which is also done in a frame-by-frame fashion. However, longer periods

of watching such flickering animation cause eye strain and generally result in an unpleasant viewing experience. On the other hand, methods that enforce a strong temporal coherence by propagating some information between frames often produce artificially looking results, where textures are essentially stuck to moving surfaces. Our goal in this thesis is to try to approach this problem in a unique way and transfer not only the static look of the exemplar to the target animation, but also the appearance of the material's natural movement over the surface. For example, a short video time-lapse of an ink drop soaking into a piece of paper would be a suitable exemplar of a material dynamically interacting with a surface. The aim is, given such exemplar sequence, that the movement in the synthesized animation also resembles contiguous bleeding of a color through fibres in paper.

Structure of the thesis

In the first chapter, we provide an overview of already published methods for style transfer and automatic artistic stylization of still images and video.

The second chapter is dedicated to the patch-based approach to texture synthesis. We formulate it as a global optimization problem in detail and introduce the spectrum of related concepts and problems.

In the next chapter, we describe our approach exhaustively. Building upon the concepts from the previous chapter, we formulate the problem rigorously and describe all individual parts of the final algorithm in detail.

The fourth chapter provides some details of our implementation and describes the steps taken towards optimizing the computation time.

In the last chapter, we evaluate the achieved results and discuss the limitations of the method that can be addressed in future work.

Chapter 1

Related work

1.1 Stroke-based rendering

A large body of work focuses on methods in the category of *stroke-based rendering*. They are based on an ordered placement of dabs of paint, single strokes [1, 2, 3], or textured curves [4, 5] onto the output image to form a stylized version of the given input (see figure 1.1). The placement of these primitives can be determined either via local criteria [1, 4] or global optimization [5]. Temporal coherence in video is typically achieved by moving the already placed strokes according to the movement in the input video (e.g., its optical flow [3] or a 2D projection of objects' movement in a 3D scene [2]). This approach effectively prevents the "shower door" effect, that is, when a texture of a moving object stays fixed relative to the viewport, making the object look like it was observed through a glass door [2]. Methods in this category generally allow for a wide range of customization via the selection of the painting primitives and textures and are able to produce compelling stylization with various artistic media. However, the painting primitives used are most often static, and in the case of animation stylization, they only change position or shape between frames. Therefore, they are not able to convey the dynamic behavior of a given artistic medium.

Figure 1.1: Stylized images generated with stroke based methods: Meier [2] (left), Hertzmann [4] (center) and Hays, Essa [3] (right)

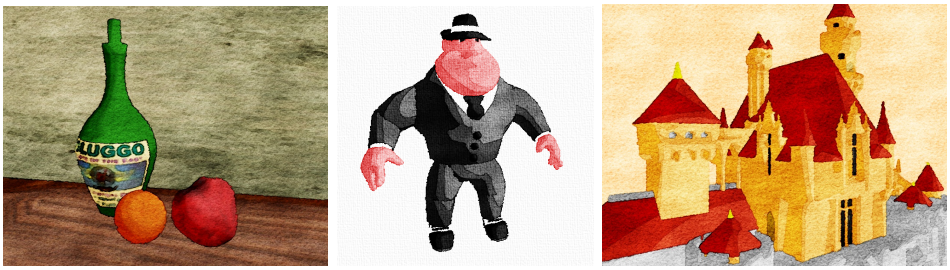


Figure 1.2: Watercolorization results from [7]

1.2 Image filtering and processing

Another substantial subset of procedural takes on artistic stylization comprises various filtering and processing of the input imagery and possibly blending it with a procedurally generated texture. Such methods are widely used in consumer photo-manipulation and graphics software. A framework for real-time 3D scene stylization is implemented in [6]. A common set of local semantic parameters for various stylization effects is established while enabling users to interactively vary these parameters in screen-, texture- and object-space. Each of those effects, however, needs to be separately implemented for usage in the framework. A pipeline for *watercolorization* of both still images and video is presented in [7] (see figure 1.2). They employ a series of pigment distribution heuristics, abstraction filters, and surface texture synthesis steps to produce results looking as if they were painted with watercolor. They also address temporal coherence in animation stylization in a fashion similar to [2] (see above). Although their method produces compelling results, it is limited to only a particular look. Also, realistic movement of watercolor is not addressed.

In [8], in addition to extending the watercolorization pipeline from [7], a method is presented to maintain temporal coherence in stylized video using temporal morphological filtering and texture advection (i.e., warping a texture along a motion-field). Similarly, in [9], a 2D pattern is successively transformed in a shape-preserving manner to match the movement of objects in an animated 3D scene. The video stylization techniques that rely on texture advection do not fit our goal of convincingly capturing the interaction of an artistic medium with a static surface, because in such setting, the texture features imparted by the surface should not undergo any deformation between consecutive frames.

1.3 Physical simulation

Physical simulations have been employed to achieve a realistic look of watercolor imagery. A layered model for water flow and pigment deposition on the

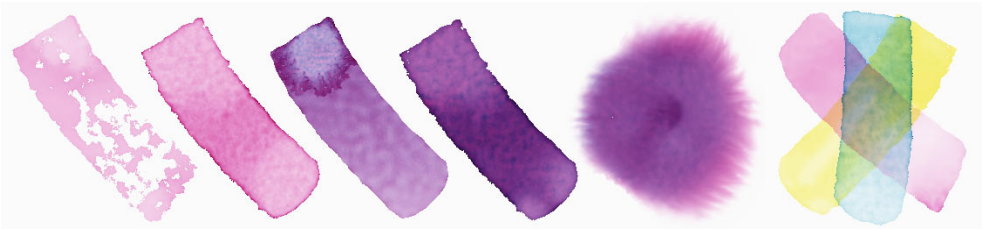


Figure 1.3: Simulated watercolor effects created using the system from [10]

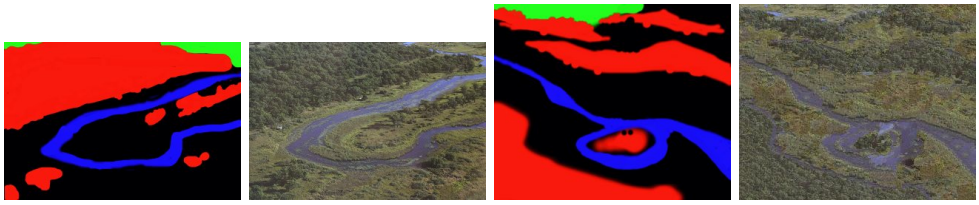
surface is presented in [10]. All washes over the paper are simulated separately and subsequently composited to form the final image. Their method enables a wide range of effects observable in watercolor art, such as dry-brush, blooming, and bleeding (see figure 1.3). It is usable for both interactive painting and automatic stylization. Several commercial products for digital painting simulate the dynamic behavior of watercolor with high visual authenticity, notably Fresco by Adobe [11] and Rebelle by Escape Motions [12]. None of these approaches, however, is designed for use in animation stylization as-is.

1.4 Pixel-based and patch-based stylization by example

Each of the methods mentioned so far has some degree of limitation to the range of styles it can produce. Arguably more versatile are example-based approaches to image stylization. The idea of example-based techniques is that instead of focusing on a particular look, the stylization is learned from an additional input — an exemplar. The seminal algorithm Image Analogies presented in [13] uses the concept of additional guidance channels to stylize an image based on a pair of a different image and its stylized version. This particular method is a pixel-based one, as it generates one pixel of the final image at a time. The pixel value is sampled from a probability distribution of center pixels lying in visually similar rectangular patches [14]. Their Texture-by-Numbers application shows the algorithm's general ability to steer patch selection towards particular source patches in different areas of the output image and thus produce a stylization of a scene (see figure 1.4).

Patch-based methods, comprehensively covered in [15], go a step further, and instead of copying single pixels from the exemplar, they transfer contiguous patches of it. These patches can be of irregular shapes, and the optimal seam between them is sought for as part of the synthesis [16, 17, 18]. Possible overlapping of such patches is addressed by some kind of blending between them [15].

Patch-based texture synthesis is often formulated as a global energy minimization problem [19, 20]. In [21], [22], and [23], various additional constraints



(a) Unfiltered source (b) Filtered source (c) Unfiltered target (d) Result

Figure 1.4: Texture-by-numbers: application of Image Analogies [13] for manually guided texture synthesis

are introduced to avoid repetitive usage of only a few patches, which leads to overly smooth areas — washout — in the synthesized results. As in Image Analogies, the objective energy function can be designed to take additional guidance channels into account [24, 22, 25]. Bénard et al. in [24] stylize animations originating from a traditional CG workflow, transferring the style of (partially) stylized keyframes to the rest of the frames while preserving temporal coherence. They rely on user-provided velocity fields and orientation fields, which can be computed automatically. The velocity fields are used to advect the result of the previously synthesized frame to guide the synthesis of the current frame. In the method for general video stylization presented in [26], the necessary guidance channels and vector fields are generated automatically.

The problem at hand can also be seen as a fluid texturing one. In the specific case of watercolor, we are actually working with a thin layer of fluid, which moves over the surface by the action of physical forces. This view can intuitively be generalized to other artistic media as well. In [20], Kwatra et al. propose a method for synthesizing a temporally coherent sequence of images, which move according to a user-provided flow field. A single exemplar image is used to texture every frame of the sequence. A method for texturing a surface of a three-dimensional body of fluid is presented in [27], where a bump map is synthesized along with the texture to generate complex microstructures. More recently, LazyFluids [23] enabled users to stylize a two-dimensional fluid animation based on a provided flow-field and a video exemplar of a moving fluid. They ensure that the exemplar's movement is authentically represented in the resulting sequence. While this leads to visual enrichment of temporal dynamics in the animation, the resulting movement does not strictly follow the target flow-field.

All of the mentioned fluid stylization methods utilize advection to maintain temporal coherence, which is not suitable for our purpose for reasons mentioned in section 1.2.

(a) Style exemplar (b) Content exemplar (c) Result

Figure 1.5: Image Style Transfer Using Convolutional Neural Networks [28]

1.5 Neural-based style transfer

Recent research on convolutional neural networks also led to their utilization in style transfer. Gatys et al. in [28] showed that responses on different layers of a VGG object recognition network can be separated into style- and content-related ones. Optimizing a white noise image to match a style exemplar in the style domain and an input image in the content domain leads to an image capturing the desired content in the style of the style exemplar (see figure 1.5). However, as stated in [28], it is problematic, both for a human and a computer, to properly define what constitutes the style of an image. Therefore this method often produces a mere recoloration of the input image or, conversely, forces parts of the style exemplar's content to appear in the output (for example, the stars in figure 1.5c).

A general-purpose method combining neural and patch-based approaches was recently presented in [29]. They first synthesize the global, semantically meaningful style using a neural approach and then add fine details free of artifacts in a patch-based fashion. The method also allows for synthesizing very high-resolution images in a reasonable time, overcoming another limitation of purely neural-based methods, which is the upper limit on image resolution.

None of those methods, however, is designed for generating temporally coherent animation. A method for general video-to-video translation based on generative adversarial networks is presented in [30]. It is capable of, for example, learning a mapping of segmentation masks to a photorealistic video. The method is, however, mainly focused on larger-scale features and would not be feasible for synthesizing imagery with fine detail. Also, as with other neural-based approaches, the learning phase takes large amounts of time (it is reported to take several days on multiple GPUs in [30]), which makes introducing new appearance exemplars a non-trivial task.

Chapter 2

Background

Despite the previous chapter's conclusion that no previously published method can be used for our purposes as-is, many of their key ideas and solutions to common problems are relevant to the goal of this thesis. Before looking into them in detail, let us first establish some basic notions.

For purposes of this thesis, we define an image I in the domain D as a two-dimensional array of values from D. For a pair of integer coordinates $p = (x, y)$, $I(p)$ will denote the value of a pixel at coordinates x, y in an image I. The notation $I_2 \subset I_1$ means that the image I_2 is a rectangular window cropped from I_1. If I is an image with finite dimensions w and h, by denoting $p \in I$ we mean that $p \in \{0, \ldots, w-1\} \times \{0, \ldots, h-1\}$.

When an image represents a boolean mask, that is, the domain D is $\{0, 1\}$, we will denote the count of pixels with the value of 1 as $|I|$ and the fact that $I(p) = 1$ will alternatively be denoted as $p \in I$.

In computer graphics, color is typically expressed in terms of its coordinates in a chosen color space. Values in the linear RGB color space map to intensities of the red, green, and blue components. In texture synthesis, the choice of CIELAB color space is common because it is designed so that a relative change in the value roughly corresponds to the perceptual color change consistently over the whole space. This is beneficial for evaluating visual similarity of two images. As well as RGB, CIELAB also uses three components. Therefore, we will refer as color images to images in the $\mathbb{R}^3$ domain.

2.1 Texture synthesis

Various methods for example-based texture synthesis have been published. In this section, we will look into the basic approaches and concepts behind the problem and finally show how the task of texture synthesis can be formulated as global optimization of a synthesized image's color values.

2.1.1 Region-growing

The way the texture synthesis problem is approached in [14] serves as an important basis for patch-based methods. The texture exemplar S is viewed as a sample from an infinite stationary texture S_{real}. (A texture is said to be stationary when the probability distribution of a pixel value is independent of its location.) The goal is to produce another sample $T \subset S_{real}$ from the same texture. Let I be an image and $w(p) \subset I$ a square window of an odd size centered on the pixel $p \in I$. The texture S_{real} is modeled as a *Markov random field* (MRF), i.e., it is assumed that the probability distribution of a pixel's values is independent of the values outside its spatial neighborhood. In other words, the probability $P(S_{real}(q) = v)$ of pixel q having a value of v is equal to the conditional probability $P(S_{real}(q) = v \,|\, w(q))$. Let $d(p, q)$ be an unspecified perceptual dissimilarity measure between two equally sized square windows centered on pixels p and q, respectively. The probability distribution function of values in pixel q is approximated as the histogram of $\{S(p') \,|\, d(p', p_{best}) < \epsilon\}$ for some small threshold ϵ, where $p_{best} = \arg\min_{p'' \in S} d(p'', q)$. Sampling from this estimated distribution gives the synthesized value of pixel q given a complete window $w(q)$. For this to be useful in texture synthesis, the dissimilarity measure d must take into account the pixels in $w(q)$ that are yet to be synthesized. The complete texture synthesis then starts from a small seed and grows it by synthesizing the rest of the texture one pixel at a time in the way described above.

The *sum of squared differences* (SSD) of pixel values is often used as the window dissimilarity measure. The pixel values in a $n \times n$ window $w(p)$ from a color image can be concatenated to form the vector $\boldsymbol{w}(p) \in \mathbb{R}^{3n^2}$. The SSD formula then takes the form

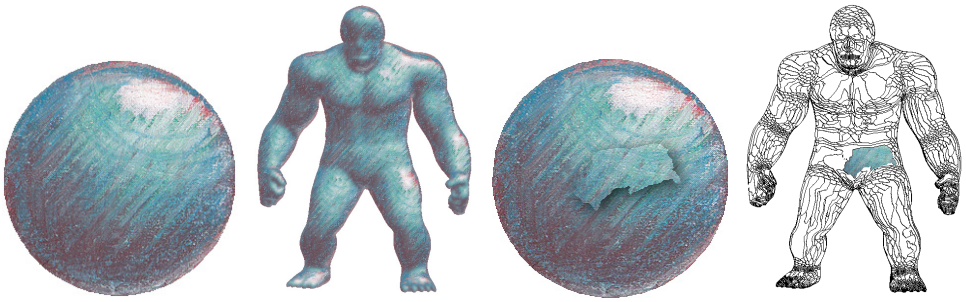
$$d(p, q) = \|\boldsymbol{w}(p) - \boldsymbol{w}(q)\|^2.$$

In this sense, finding the most similar windows can be thought of as finding the *nearest neighbors* in $\mathbb{R}^{3n^2}$ according to the metric represented by d. To account for the yet unknown pixel values, one can sum only over the known ones and weight the result accordingly.

This approach is well suited for synthesizing highly stochastic textures but fails to preserve fine details in more structured exemplars. The patch-based techniques we are interested in do not operate in terms of single pixels but rather aim to produce continuous patches copied verbatim from the source image (see figure 2.1). Ashikhmin in [31] shows that encouraging the formation of such continuous regions leads to better preservation of the source texture's structure on a fine scale.

2.1.2 Texture optimization

The problem of patch-based texture synthesis can alternatively be posed as a global optimization problem with a straightforward iterative algorithmic



(a) Style exemplar (b) Result (c) A patch from the exemplar (d) Placement of the patch in the result

Figure 2.1: An irregular patch transferred from exemplar to target [18]. The black lines in (d) represent patch boundaries.

solution [19, 20]. One of its added benefits over the region-growing method mentioned above is that it overcomes the unidirectional nature of the patch formation and therefore does not end up deviating from the source as the synthesized texture grows. The objective function that is subject to minimization, also called the texture energy, is defined as

$$E = \sum_{q \in T} \min_{p \in S} d(p, q). \tag{2.1}$$

In plain words, it is a sum over all windows in the target T of distances to their closest matching windows in the source S. As noted in [18], inside a coherent patch copied from the source, these distances are zero. Therefore minimizing this energy implicitly leads to such patches forming in the resulting image with their edges seamlessly blended. The algorithm starts with an initial guess of T and iteratively refines it in a fashion similar to the *expectation-maximization* (EM) algorithm [19].

In the first step of the iteration, the *nearest neighbor field* (NNF) is constructed. It is a mapping $NNF : T \to S$ defined as

$$NNF(q) = \arg\min_{p \in S} d(p, q).$$

Note that the energy function in equation 2.1 can now be formulated as

$$E = \sum_{q \in T} d(NNF(q), q).$$

Therefore the NNF represents a set of parameters w.r.t. which the energy function is minimized when T is fixed.

The second iteration step conversely minimizes E w.r.t. T while the NNF stays fixed. When using SSD as the energy function, the minimization is a convex problem which can be solved analytically by setting the derivative of

E to zero. The result of this minimization is simply setting the new value of a pixel q to the average of contributions of all windows overlapping at q according to the NNF, formally

$$T'(q) = \frac{1}{n^2} \sum_{-\frac{n}{2} \leq x,y \leq \frac{n}{2}} S(NNF(q + (x, y)) - (x, y)),$$

where n is the window size. This step is commonly referred to as the *voting step*, as the source windows vote on the final pixel values.

Since the new pixel values in T result from blending several source windows, the NNF might have changed after the second step. Therefore these two steps are repeated until the NNF converges. In practice, however, it is usually sufficient to set a fixed number of iterations.

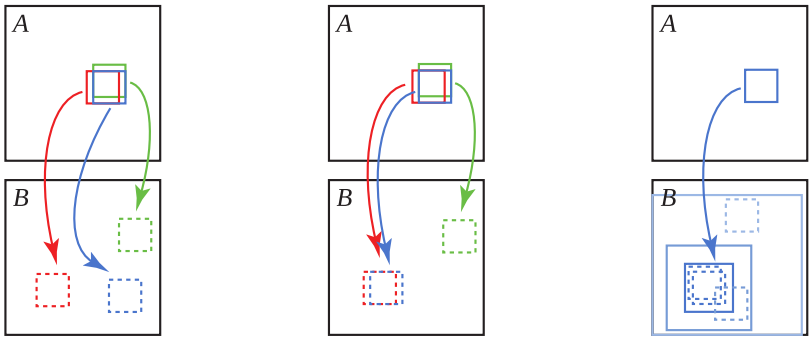
A detail we did not cover yet is getting the initial guess of T. A straightforward solution would be to use pseudorandom values for each pixel. However, doing so would introduce a bias towards source windows close to this arbitrary random initialization. Results better corresponding with the source texture can be achieved by randomizing the NNF instead and producing the initial T with a single run of the voting step. This way, the global distribution of pixel values in the initial guess corresponds with the exemplar.

2.2 PatchMatch

Various methods for retrieving the nearest neighbor of a window can be employed. The naive approach would be to exhaustively iterate over source pixels and choose the one with the smallest distance. Substantial speed up can be achieved using more elaborate methods. k-dimensional binary search trees [32] offer logarithmic time complexity of the query operation and therefore are a popular choice for this kind of task in various fields. The Winner-Update algorithm [33] is another valid choice, and it does not suffer from the *curse of dimensionality* as much. Note that the topic of dimensionality is very relevant to us because, as we will see later, the actual search space is typically of much higher dimension than $3n^2$. Some form of spatial hashing can be used as well.

The methods mentioned above enable retrieval of an exact nearest neighbor. For our use-case, an approximate match is sufficient. Again, many methods for finding approximate nearest neighbors exist, but we will focus on one specifically designed to produce a complete approximate NNF between two images, PatchMatch [34]. The key idea of the algorithm is that, in a sufficiently large image, a random assignment of a source pixel to a target one is in many cases correct by itself, and given the inherent high correlation between adjacent windows, good matches can be propagated between them.

The algorithm starts with a randomized NNF (figure 2.2a). Then an iterative improvement of it is performed. In each loop of the iteration, the following two steps are taken successively for each pixel in T.



(a) Random initialization (b) Propagation (c) Random search

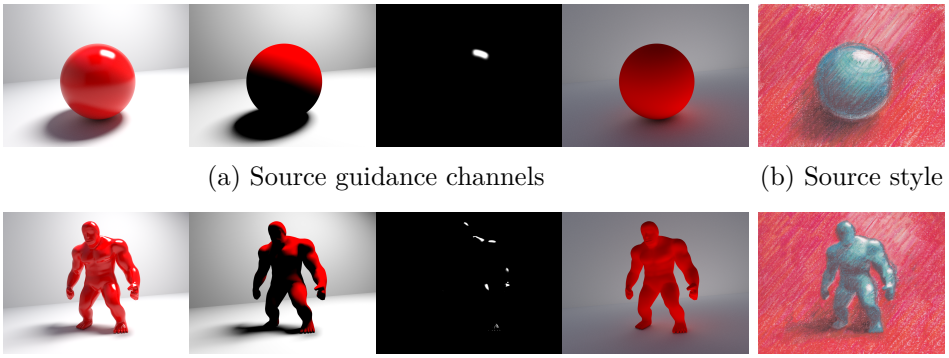
Figure 2.2: The phases of the PatchMatch algorithm [34]. The window whose mapping is being optimized is in blue.

First, an attempt is made to improve the mapping of a pixel q by propagating the mapping of the pixel's left and top neighbors $q-(1,0)$ and $q-(0,1)$. The assumption is that if a neighbor pixel already has a good match in S, the appropriate translation of its mapping is likely to be a good match for the current pixel (figure 2.2b). Therefore the mapping of q is updated to $\arg\min_p d(p,q)$ for p from $\{NNF(q), NNF(q-(1,0))+(1,0), NNF(q-(0,1))+(0,1)\}$. Additionally, the direction is reversed on even iterations, and the mapping is propagated from right and bottom neighbors instead.

In the second step done for a pixel q, the source S is randomly searched for potentially better matches. Each candidate $p_i \in S$ is randomly chosen from a window $w_i \subset S$ of size $\alpha^i n_{max}$ centered at the current $NNF(q)$, where α is a real positive coefficient less than one, n_{max} is the maximum search radius, and i goes from 0 to the point where the window size is less than one (see figure 2.2c). Again, $\arg\min_{p_i} d(p_i,q)$ is chosen as the new value of $NNF(q)$. A typical choice for α is 0.5, and for n_{max} it is the maximal dimension of S.

This per-pixel improvement is repeated until the NNF converges. However, the convergence is shown to be very fast, so performing just a small fixed number of iterations such as 4 or 6 is generally enough for a good approximation.

Furthermore, the search space can be extended with spatial transformations of the source windows, such as rotation and scale about the center [35, 36]. Such mapping can be expressed by extending the source pixel coordinates with additional parameters, for example, $NNF(q) = (x, y, \theta, s)$ when both rotation and scale are to be searched over. In the random search phase, the candidates get sampled from gradually contracting hypercubes in the parameter space. Special care must be taken in the propagation phase, as the translations of the neighbors' mappings have to be first transformed by the Jacobian of the source window transformation. Note that if the rotation is not constrained to multiples of 90 degrees, or the scaling allows for non-integer



(a) Source guidance channels (b) Source style

(c) Target guidance channels (d) Result

Figure 2.3: Various guidance channels used in StyLit [22]

factors, the x and y coordinates have to be real numbers as the translation may result in non-integers. In the voting step, interpolation of the pixel values has to take place when sampling the source at non-integer coordinates.

2.3 Guided synthesis

The perceptual distance measure d is not limited to just distances between vectorized windows. As described in [20], it can, for example, match the image gradient as well, as in

$$d(p,q) = \|\boldsymbol{w}(p) - \boldsymbol{w}(q)\|^2 + \mu\|\nabla\boldsymbol{w}(p) - \nabla\boldsymbol{w}(q)\|^2,$$

where $\nabla w(p)$ is the discrete gradient at p and μ is a parameter defining the relative weight of the gradient difference. In that case, minimizing the texture energy from equation 2.1 w.r.t. T requires solving a system similar to Poisson's equation. In general, any function of T can be used as long as it can be minimized w.r.t. T.

A special case is when additional channels are introduced which only serve as a guide for the synthesis (in terms of being present in the formula for the distance measure d) but do not get synthesized along with T. Each of those channels can be given an independent weight. The formula for weighted SSD thus becomes

$$d(p,q) = \sum_i weight_i\|\boldsymbol{w}_i(p) - \boldsymbol{w}_i(q)\|^2,$$

where $weight_i$ is the weighting coefficient of the i-th channel and $\boldsymbol{w}_i$ is the concatenation of values in the i-th channel of w.

For example, in Image Analogies [13], the luminance channel and responses of several differently oriented derivative filters of the unfiltered images are used as the guidance channels (referred to as feature vectors in the paper). In StyLit

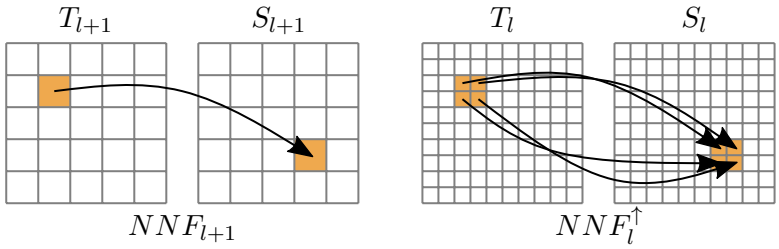


Figure 2.4: The result of upscaling an NNF mapping

[22], several separately rendered illumination effects (such as diffuse lighting and indirect reflections) are used (see figure 2.3).

In general, whether they are user-provided or automatically generated by pre-processing the images, the guidance channels serve the purpose of guiding the target-to-source mapping towards appropriate locations in the exemplar, based on information not directly present in the images themselves.

2.4 Multi-scale synthesis

A common way to speed up the convergence of synthesis while preserving high-level structures is to perform it in a coarse-to-fine fashion. For both the source image S and target image T, a sequence of images is generated such that $S_0 = S$, $T_0 = T$, and S_i, T_i are their lowpass filtered versions with cutoff frequency decreasing with $i = 1, \ldots, L$. Per the Nyquist-Shannon theorem, the filtered images can be downsampled to lower resolutions, thus producing what is called the *Gaussian pyramid*. The texture optimization algorithm then starts at the coarsest level of the pyramid and proceeds as described above. When moving a level up, the algorithm is initialized with the results from the coarser level in the following way.

First, the resulting NNF_{l+1} from the coarser level gets upscaled to the size of the current level. The most common and most straightforward choice is to double the size at each level, so the upscaled mapping $NNF_l^{\uparrow}$ is assigned as

$$NNF_l^{\uparrow}(q) = 2NNF_{l+1}(\lfloor q/2 \rfloor) + (q \bmod 2),$$

where the division, floor, and modulus operators act per component of the coordinate pair q. This essentially means that a mapping of a single target pixel gets upscaled to a block of 4 pixels mapped to 4 neighboring source pixels, as illustrated in figure 2.4.

The initial guess of T at level l is then obtained by simply performing the voting step with the upscaled NNF. Performing the upscaling of T in this manner, leveraging image information at the current pyramid level, ensures that the image is crisp, which would not be the case if it was merely upsampled. The upscaled NNF is also used as the initial instead of a random one.

Figure 2.5: Results from [21] comparing no spatial uniformity enforcement (left), using BDS (middle), and their spatial uniformity constraints (right)

The single performance bottleneck of texture optimization methods is finding the NNF mapping. The computational complexity of PatchMatch grows linearly with the size of the target and logarithmically with the size of the source. Therefore, doing the computation on coarser levels, which are orders of magnitude smaller than the full resolution images, brings substantial speedup. Furthermore, given the good guess used for initialization at finer levels, fewer subsequent iterations are needed to improve it.

Another advantage of the multi-resolution approach is enhanced preservation of patterns in the texture that are larger than the window size. Since equally sized windows are used on all pyramid levels, their size is essentially enlarged in terms of the full resolution. Therefore, the windows in coarser levels capture a larger context of the source texture.

2.5 Spatial uniformity constraints

A common undesirable phenomenon may occur with the approach described so far, when the optimization process ends up using only a small number of source patches and therefore not representing the source faithfully. The problem stems from the fact that only local neighborhood similarities are taken into account. Typically, source patches with a low variance of pixel values are cheapest in terms of the texture energy, so the results tend to manifest large overly smooth areas and not capture the overall appearance well (see figure 2.5 left).

Histogram matching, introduced in [37], is a way to enforce some global statistics of the resulting image, namely the histogram of individual color channels. It takes place in the voting step, where color contributions of all overlapping windows are weighted to penalize contributions that would lead to an increase of difference between the source and target histograms.

In [38], a measure of *bidirectional similarity* (BDS) is presented, that extends the energy function 2.1 with a completeness term:

$$E = \frac{1}{N_T} \sum_{q \in T} \min_{p \in S} d(p, q) + \frac{1}{N_S} \sum_{p \in S} \min_{q \in T} d(p, q),$$

where N_S and N_T are the sizes of the source and target images respectively. Minimizing w.r.t. this energy function enforces all source pixels to be represented somewhere in the target in addition to target patches being similar to source ones.

The BDS measure optimization, however, still does not enforce uniform usage of source pixels. A step towards this goal is taken in [21], where excessive usage of just a small number of source pixels gets penalized through an additional term in the distance function. An occurrence map Ω is introduced, which keeps track of how many times each source pixel is used in the target:

$$\Omega(p) = |\{q \in T \,|\, p \in w(NNF(q))\}| \,.$$

The distance function d is then extended with a term reflecting the usage count of pixels in the source window:

$$d'(p,q) = d(p,q) + \lambda \frac{\Omega(w(p))}{\omega_{best}},$$

where $\Omega(w)$ is the average occurrence count of pixels in the window w, that is $\frac{1}{n^2}\sum_{p \in w} \Omega(p)$, ω_{best} is the expected pixel occurrence count equal to $n^2|T|/|S|$, and λ is a parameter controlling the weight of uniformity enforcement. Results obtained with this approach are seen in figure 2.5.

In [23], the uniform pixel usage is strictly enforced by the introduction of an additional criterion

$$\sum_{p \in S} \delta(p) = |T| \text{ and } \delta(p) - K \in \{0, 1\},$$

where $\delta(p)$ is the number of usages of the source pixel p and $K = \lfloor |T|/|S| \rfloor$. A modified NNF retrieval scheme is used, where the mapping gets retrieved in a reversed direction, i.e., $S \to T$. When more than one source pixel gets mapped to a single target one, the assignment with the least distance is kept. This process is repeated until all target pixels are assigned a mapping to a source pixel. In order to fulfill the uniformity constraint, the reversed NNF retrieval is constrained to target pixels with yet unassigned counterparts and source pixels which satisfy $\delta \leq K$. Every time a source pixel p gets assigned, for which $\delta(p) = K$ holds, a counter R, originally set to $|T| \bmod |S|$, gets decremented. Once R reaches zero, only pixels satisfying $\delta(p) < K$ are considered.

The concept of adaptive error budget is introduced in [22]. Again, repeated retrieval of reversed NNF is performed. An observation is made that the plot of all distances in a mapping sorted in ascending order typically resembles a hyperbole (see figure 2.6). In order to determine a feasible error budget, a hyperbolic function f is fitted to the sorted distances, and a point k, such that $f'(k) = 1$, is retrieved. The point k represents a knee in the function, after which errors start increasing quickly. Only those mappings with an index below k in the ascending order are assigned, and the procedure is repeated for the rest of the target pixels. While this approach does not enforce strict uniformity, it still gives better results than using an occurrence map from [21].

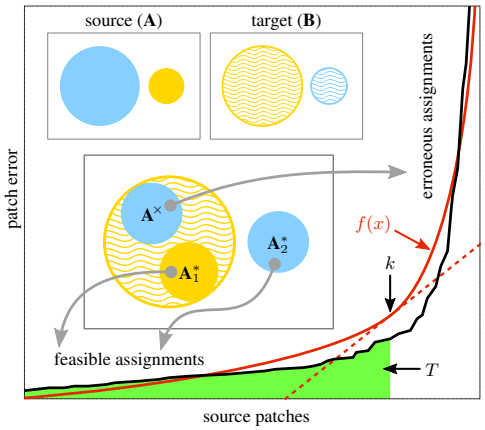


Figure 2.6: StyLit: Fitting a hyperbolic function to sorted distances in order to determine a feasible error budget [22].

2.6 Temporal coherence

Texture synthesis methods can also be used to generate animation, i.e., sequences of individual frames. For basic texture synthesis tasks, the notion of an image can be easily extended to three dimensions, as was seen, for example, in [19]. Their method for spatio-temporal hole infilling works with three-dimensional windows, which vote for pixel values in several frames at once.

Synthesizing one frame at a time without proper guidance typically leads to unpleasant temporal artifacts. The approaches described so far do not guarantee convergence to any particular local minimum, so successive frames may, and most often do, vary heavily. This leads to overall temporal noise in the resulting animation and causes a sensation of flickering. While this effect may be desirable to some degree (see [39]), too much noise is generally seen as undesirable as it causes eye strain or even nausea in the observer.

Another unwanted effect that often appears in various video stylization applications is the so-called *shower door effect* (coined in [2]). This term refers to the case when a texture of a moving object stays fixed relative to the viewport, therefore making the object look like it was observed through a glass door. This can be seen as the opposite of the temporal noise problem.

To maintain temporal coherence and avoid the shower door effect, some information has to be propagated between successive frames. A possible way is to use the synthesized result of the previous frame advected according to the motion or flow in the target scene as an additional guidance channel (see [20, 24, 23]). This way, the synthesis of the current frame is guided towards results somewhat similar to the previous frame (thus minimizing flickering) while avoiding the shower door effect (thanks to advection).

Chapter 3

Our approach

Principles relevant to the goal of this thesis were introduced. In this chapter, the complete proposed algorithm will be presented, which builds on some of those principles.

3.1 Problem formulation

The patch-based algorithm presented in this thesis aims to provide a way to automatically stylize a vector animation with the appearance and motion characteristics of a traditional painting medium, primarily focused on, but not limited to, watercolor.

The synthesis is guided only by the shape of objects in a target animation and not, for example, by their color. Therefore, we generalize the target guidance to a sequence of boolean masks. Parts outside the mask are not synthesized as a part of the algorithm, and it is up to the user to composite the result with a background afterward (possibly using some other method for texture synthesis to generate the background and blend it with the result seamlessly). It is assumed that the exemplar is captured perpendicular to the surface. The inputs to the algorithm are following (see figure 3.1):

- $S^{rgb} = \{S_i^{rgb}\}_{i=1}^{l_S}$ — a sequence of images serving as the style exemplar,
- $S^{mask} = \{S_i^{mask}\}_{i=1}^{l_S}$ — a sequence of boolean masks corresponding to the occurrence of the material in the exemplar,
- $T^{mask} = \{T_i^{mask}\}_{i=1}^{l_T}$ — a sequence of boolean masks defining the desired placement of the material in the resulting sequence.

Some simplifying assumptions are made about the process that causes motion in the source and target sequences:

- The motion field is as smooth as possible in both the spatial and temporal domains.

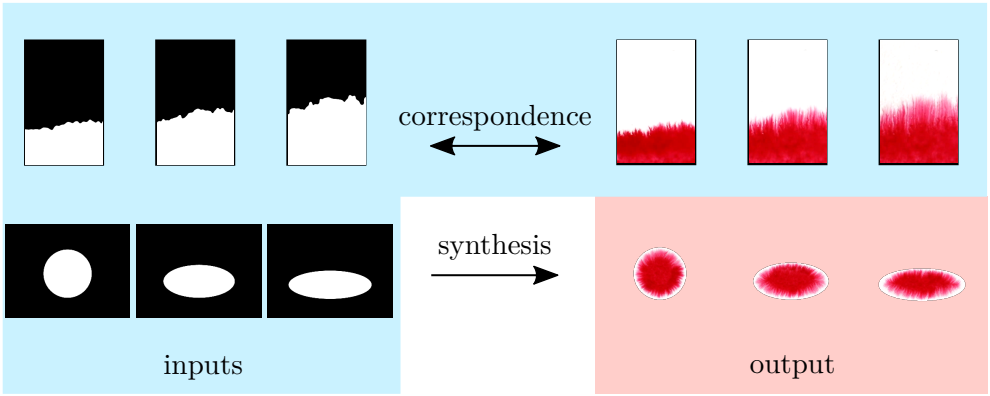


Figure 3.1: A demonstration of the inputs and outputs of the algorithm on several frames from the respective sequences

- The process is fully reversible and time-independent. In other words, the motion can be arbitrarily scaled along the temporal axis even by negative factors.

- The process is also independent of direction and position in screen space.

- All movement in the sequence is planar and is caused exclusively by this process and not, for example, by movement of the camera.

Note that these assumptions typically do not strictly correspond with the real physical phenomena manifested in the captured exemplars (e.g., diffusion does not fulfill the reversibility assumption). However, our aim is not to achieve physically accurate renderings (for example, a splotch of ink would not behave like a walking figure under natural circumstances), and these assumptions will help us stylize variously moving shapes while preserving some degree of visual coherence.

The look of an artistic medium is analyzed as an interaction between two different textures — the static texture of the surface and the dynamic texture of the artistic material. In this setting, the static nature of the surface texture should be preserved in the target sequence as much as possible. This criterion rules out using texture advection for achieving temporal coherence.

Even though the algorithm produces one frame at a time, the whole source sequence serves as an exemplar for each frame. Additionally, in connection with the direction independency assumption, spatial rotations of the source patches are searched over. In contrast with some other approaches, this avoids the necessity of enriching the source with its precomputed rotations.

3.2 Texture synthesis

The texture optimization algorithm of Kwatra et al. [20] described in section 2.1 is used as the basis for the texture synthesis. It is ran for each target frame individually, starting with the first one. In its basic form, it can be described with the following pseudocode:

Algorithm 1 Texture Optimization

Input sequences S^{rgb} and S^{mask} (see section 3.1), the mask of the current frame T_i^{mask}, the initial mapping NNF_{init}
Output the optimized mapping NNF, the synthesized image T_i^{rgb}

```
function OPTIMIZE(S^rgb, S^mask, T_i^mask, NNF_init)
    NNF ← NNF_init
    T_i^rgb ← VOTE(S^rgb, NNF)
    for j ← 1 ... opt_iters do
        NNF ← FINDNNF(S^rgb, S^mask, T_i^rgb, T_i^mask)
        T_i^rgb ← VOTE(S^rgb, NNF)
    end for
    return NNF, T_i^rgb
end function
```

The function FINDNNF returns a mapping of every target pixel $p \in T_i^{mask}$ to the nearest neighbor of its corresponding window in the sequence S^{rgb}. The nearest neighbor is represented as a quadruple (x, y, k, θ), where $1 \leq k \leq l_S$ is a source frame number, θ is a rotation of the mapped source window, and $(x, y) \in S_k^{mask}$.

3.3 Spatial uniformity

To encourage spatially uniform usage of source patches, a method closely based on that of Kaspar et al. [21] (see section 2.5) is used.

Merely using a sequence of occurrence maps, one for each source frame, is not sufficient for preventing washout in our setting. This is due to the combination of two factors. First, the count of available source pixels is typically much larger than the count of synthesized pixels in a single target frame. Second, the contents of consecutive source frames are very similar to each other. Therefore, penalizing repeated usage of individual source pixels would not prevent the usage of many very similar source patches from different frames. Instead, a slightly modified approach is taken, where besides penalizing the usage of source pixels lying in an assigned window, usage of collocated pixels in several neighboring frames gets penalized too. The count of neighboring frames is configured with the parameter *occ_radius*.

Additionally, rotations of source patches have to be considered. Therefore, the formula for the occurrence map becomes

$$\Omega(p,i) = \left|\{q \in T^{rgb} \,|\, (p_1,p_2,i) \in w^*(NNF(q))\}\right|, \quad \forall p \in S_i^{rgb},$$

where $w^*(x,y,k,\theta)$ represents a three-dimensional box with dimensions $n \times n \times (2*occ\_radius+1)$, centered on the point (x,y,k), and rotated by θ radians in the x,y-plane. The new formula for ω_{best} is set to

$$\omega_{best} = n^2(2*occ\_radius+1)|T^{mask}|/|S^{mask}|.$$

3.4 Edge effects

In a wide range of artistic media, including watercolor, the appearance of a painted area is substantially different between the area's edge and its interior. Also, in regions close to the edge, the directionality of the texture is typically much more pronounced. An approach very similar to that in LazyFluids [23] is taken to address that. A signed distance field is used to enable the above effects in the synthesized result. For a boolean mask I^{mask}, it is an image D_I, for which the following holds:

$$D_I(p) = \begin{cases} \min_{q \notin I^{mask}} \|p-q\| & \text{if } p \in I^{mask} \\ -\min_{q \in I^{mask}} \|p-q\| & \text{otherwise.} \end{cases}$$

Its computation was implemented using the Meijster's algorithm [40].

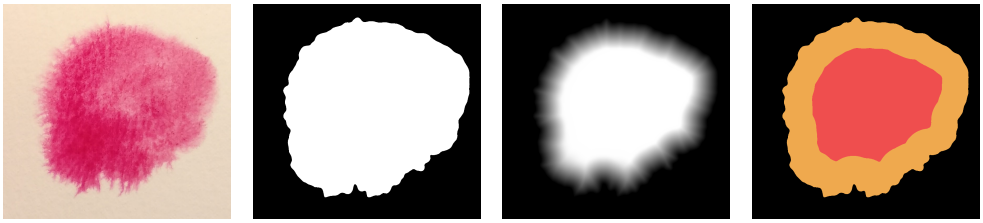
A guidance channel is derived from the distance field (see figure 3.2c). Let $S^{guide} = \{S_i^{guide}\}_{i=1}^{l_S}$ and $T^{guide} = \{T_i^{guide}\}_{i=1}^{l_T}$ be the sequences of source and target guides, respectively. Both S^{guide} and T_i^{guide} become additional parameters of the previously defined functions OPTIMIZE and FINDNNF. The edge guidance channel is then defined as

$$I_i^{df}(p) = \min(D_I(p)/border\_width, 1).$$

The *border_width* parameter is used to set the distance range, where the distinct edge effects take place in the exemplar images, partitioning the image into a border segment and an interior segment, where $I^{df} < 1$ and $I^{df} = 1$, respectively (see figure 3.2d).

The ratio between sizes of the two segments is dependent on the shape of the mask. If the size ratios of the segments differ substantially between the source and the target, the current formulation of spatial uniformity can force source's edge patches to the target's interior, or vice-versa. To mitigate this problem, the NNF mapping is modified so that it only matches pixels in corresponding segments, and the ω_{best} variable is evaluated for each segment separately:

$$\omega_{best} = \begin{cases} n^2(2*occ\_radius+1)|T^{df}<1|/|S^{df}<1|, & \text{for border pixels,} \\ n^2(2*occ\_radius+1)|T^{df}=1|/|S^{df}=1|, & \text{for interior pixels.} \end{cases}$$



(a) Color channels of an exemplar

(b) The exemplar's mask

(c) The border guidance channel

(d) The border and interior segments derived from the guidance channel

Figure 3.2: An example of the border guidance channel and the border/interior segmentation

3.5 Flow effects

Another source of a media's distinctive look is its flow over the surface. Parts of an image, where the material is subject to movement, typically have different qualities of the texture than those mostly stationary (see figure 3.3a). To construct guidance channels facilitating this effect, we need to take the material's motion into account. Let us assume that I_i^{motion} is an accurate two-dimensional flow field between frames i and $i+1$ of a sequence I. The first of the two flow-related guidance, I^{fwd}, is then constructed by accumulating the amount of motion at each pixel since the start of the sequence. Formally, it is defined as $I_i^{fwd}(p) = \|acc_i(p)\|$, where

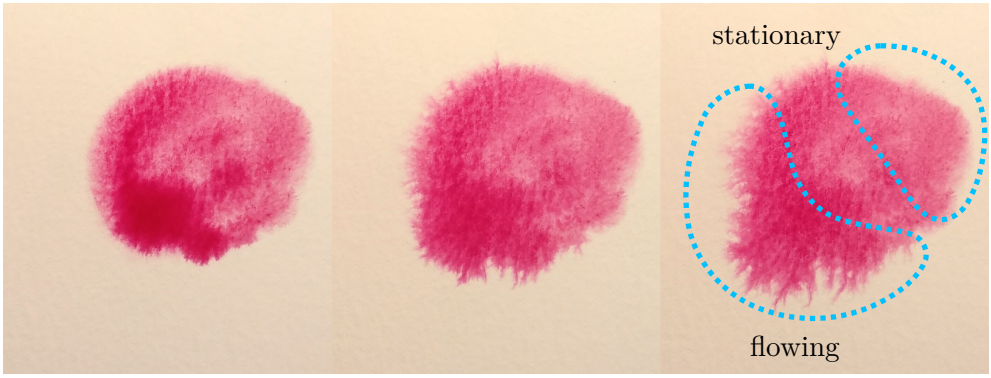
$$acc_i(p) = \begin{cases} acc_{i-1}(p) + I_{i-1}^{motion}(p), & \text{if } i > 1 \text{ and } I_i^{mask}(p) \\ (0,0), & \text{otherwise.} \end{cases}$$

The channel is illustrated in figure 3.3c. By zeroing the accumulator outside the mask we make sure that the accumulation is restarted at coordinates where the material appears repeatedly. Note that although the guidance is a single value, the accumulation is done in two dimensions. This goes hand in hand with our assumption of reversibility: a backward movement following an equivalent forward one should reduce the accumulated value back to zero, as if the process running backward got the material to its original state.

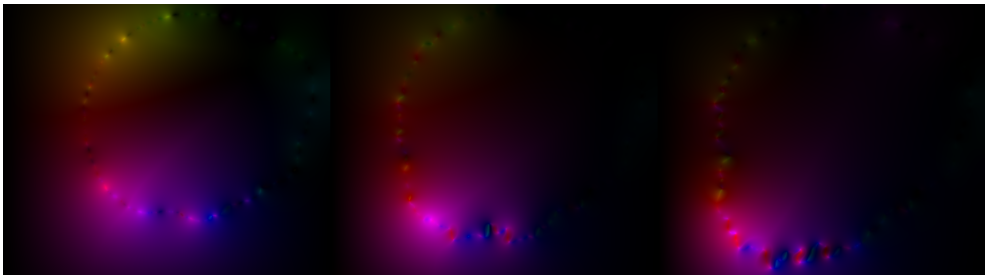
The second, complementary, guidance channel (illustrated in figure 3.3d) is calculated in the opposite time direction, i.e., $I_i^{bwd}(p) = \|acc'_i(p)\|$, where

$$acc'_i(p) = \begin{cases} acc'_{i+1}(p) + I_i^{motion}(p), & \text{if } i < l_I \text{ and } I_i^{mask}(p) \\ (0,0), & \text{otherwise.} \end{cases}$$

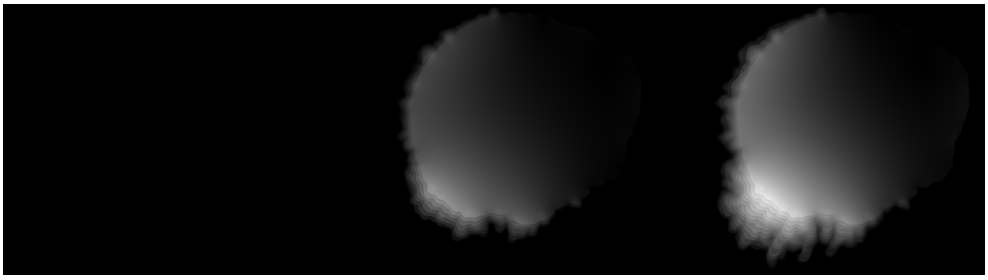
The final step towards fulfilling the assumption of reversibility is that in the distance measure d, when the source and target directions are roughly opposite, the S^{fwd} and S^{bwd} guidance channels in the source are swapped before



(a) A splotch of watercolor manifesting different looks at flowing and stationary parts



(b) Corresponding flow field. Angle is encoded as hue and magnitude as intensity.



(c) Derived guidance channel fwd

(d) Derived guidance channel bwd

Figure 3.3: Flow consistency guidance illustrated on three frames picked from a longer sequence

comparing them to the target ones. Formally, an assignment $q \to (p_1, p_2, k, \theta)$ is said to be in roughly opposite direction if $T_i^{motion}(q) \cdot R_\theta S_k^{motion}(p) < 0$, where R_θ is the operator of rotation by θ radians, Thus, reversing the order of the source frames should have no impact on the resulting sequence.

3.6 Flow orientation alignment

Another guidance channel, I^{dir}, is used to keep the flow direction in a source patch aligned with the direction in the area of the target, where this patch is copied onto. This channel holds the orientation of the motion field, i.e.,

$$I_i^{dir}(p) = \text{atan2}(I_i^{motion}(p)).$$

The orientation alignment term in the distance measure d is an exception to the SSD metric. When evaluating distance between a single source pixel p in frame i and a target pixel q in frame j with a rotation of θ radians, the contribution of the term is set to the tangent of the angle difference after the rotation, i.e., $|\tan(S_i^{dir}(p) + \theta - T_j^{dir}(q))|$. This way, similarly aligned flow orientations (possibly in the opposite direction) are preferred, and, conversely, mappings resulting in an orientation close to being perpendicular to the direction of the target flow are strictly avoided.

3.7 Temporal coherence

The core idea of this thesis is in the way the source movement is transferred to the resulting sequence. In a fashion similar to other video synthesis methods, additional guidance channels are introduced to guide the synthesis of a frame based on the previously synthesized one. Contrary to most of those methods, the guidance is not obtained by transforming the previous frame.

Instead, the guidance channels are synthesized using the final NNF of the previous frame. A new mapping, NNF_{prev}, is produced by shifting the time components of the assigned coordinates to match the desired amount of flow in each target pixel.

Let us assume, for now, that flow directions at the mapped source coordinates match the target ones (i.e., they are not allowed to point in opposite directions, see section 3.6). The amount of movement $\Delta T^{fwd}(q)$ at a target pixel q between frames $j-1$ and j is equal to $T_j^{fwd}(q) - T_{j-1}^{fwd}(q)$. Let the nearest neighbor coordinates of q from the previous frame $j-1$ be (p_1, p_2, i, θ), and $\Delta S_{\Delta i}^{fwd} = S_{i+\Delta i}^{fwd}(p) - S_i^{fwd}(p)$ be the amount of movement at the source pixel p between the frames i and $i + \Delta i$. Now, finding the optimal time offset Δi is a question of setting it such that

$$\Delta i = \underset{\Delta i' \geq 0}{\arg\min} |\Delta T^{fwd}(q) - \Delta S_{\Delta i'}^{fwd}(p)|.$$

$NNF_{prev}(q)$ is then set to $(p_1, p_2, i + \Delta i, \theta)$. This way, the mapping NNF_{prev} maps each target pixel to the same x, y-location in the source sequence as the previous frame's NNF does, but to a different point in time, at which the material at that location has moved by an amount corresponding with the desired one. And since the flow directions at the target pixels and the assigned source pixels are roughly aligned, this process essentially amounts to pushing the material from the previous frame into areas it should be located at in the current frame.

When the target and rotated source flow directions at pixels q and p are roughly opposite (see section 3.5), S^{bwd} is used instead of S^{fwd}, and the time component is shifted backward, i.e., $\Delta i \leq 0$. This enables, for example, a sequence of an expanding splotch of watercolor to serve as an exemplar for a sideways moving target animation: in the target areas where the desired flow is directed inside the mask, the new patches for the temporal coherence guidance are taken from earlier source frames, facilitating the effect of the paint "de-expanding" in the direction of the motion.

The target guidance T^{prev} is then simply obtained by running the voting step with NNF_{prev}. In the source sequence, S^{prev} is equal to S^{rgb}; that is, the target guidance channels are used to directly guide the appearance of the currently synthesized frame. Also, NNF_{prev} is used as the initial NNF to encourage stability of the mapping.

Clearly, this guidance channel is only meaningful in areas where the previous mask overlaps with the current one. Therefore, the weight of this temporal coherence term in the distance measure is set to zero outside of those areas. In the first frame, it is set to zero over the whole image, since no previous frame is present.

A fairly common scenario is that the movement in a target animation is way stronger than in an exemplar sequence. Therefore, the amount of movement between source frames i and $i + \Delta i$ may not be sufficient to match the desired movement between target frames $j - 1$ and j and the synthesized sequence would thus lag behind the target animation. Thankfully, there is a way to detect this case and correct for it. A sign of the material in T^{prev} at pixel q not being moved by a sufficient amount is that the assigned source pixel p is too far or too near to the mask's edge in comparison to the target pixel q. Taking inspiration in LazyFluids [23], we introduce a spatially varying temporal modulation factor $m(q)$ in the distance function. It determines the temporal coherence weight based on the difference between the targets distance $T_j^{df}(q)$ and the sources distance $S_{i+\Delta i}^{df}(p)$ using the formula

$$m(q) = \begin{cases} 0, & \text{if } q \notin T_{j-1}^{mask}, \\ \text{smoothstep}(|T_j^{df}(q) - S_{i+\Delta i}^{df}(p)|, m_u, m_l), & \text{otherwise,} \end{cases}$$

where smoothstep is the common utility function defined as

$$\text{smoothstep}(x,l,u) = \begin{cases} 0, & \text{if } x' \leq 0, \\ 1, & \text{if } x' \geq 1, \\ 3x'2 - 2x'3, & \text{otherwise,} \end{cases} \quad ; x' = (x-l)/(u-l),$$

and m_u and m_l are configurable upper and lower thresholds.

This concludes the introduction of all guidance channels used in the algorithm. The guidance channels I^{guide} are therefore formed by concatenating individual values in I^{df}, I^{fwd}, I^{bwd}, I^{prev} and I^{dir}. The distance measure formula thus becomes

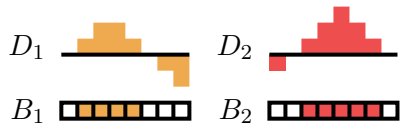
$$\begin{aligned} d(p,i,\theta,q) = {} & \|\boldsymbol{w}_\theta^{rgb}(p) - \boldsymbol{w}^{rgb}(q)\|^2 \\ & + weight_{df}\|\boldsymbol{w}_\theta^{df}(p) - \boldsymbol{w}^{df}(q)\|^2 \\ & + weight_{fwd}\|\boldsymbol{w}_\theta^{fwd}(p) - \boldsymbol{w}^{fwd}(q)\|^2 \\ & + weight_{fwd}\|\boldsymbol{w}_\theta^{bwd}(p) - \boldsymbol{w}^{bwd}(q)\|^2 \\ & + m(q)weight_{prev}\|\boldsymbol{w}_\theta^{prev}(p) - \boldsymbol{w}^{prev}(q)\|^2 \\ & + weight_{dir}|\tan(S^{dir}(p) + \theta - T^{dir}(q))|, \end{aligned}$$

where w_θ is a square window cropped from an image rotated by θ radians around the window's center.

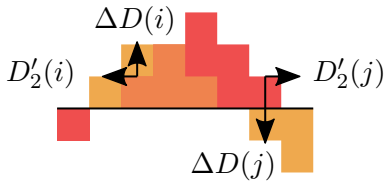
3.8 Flow field construction

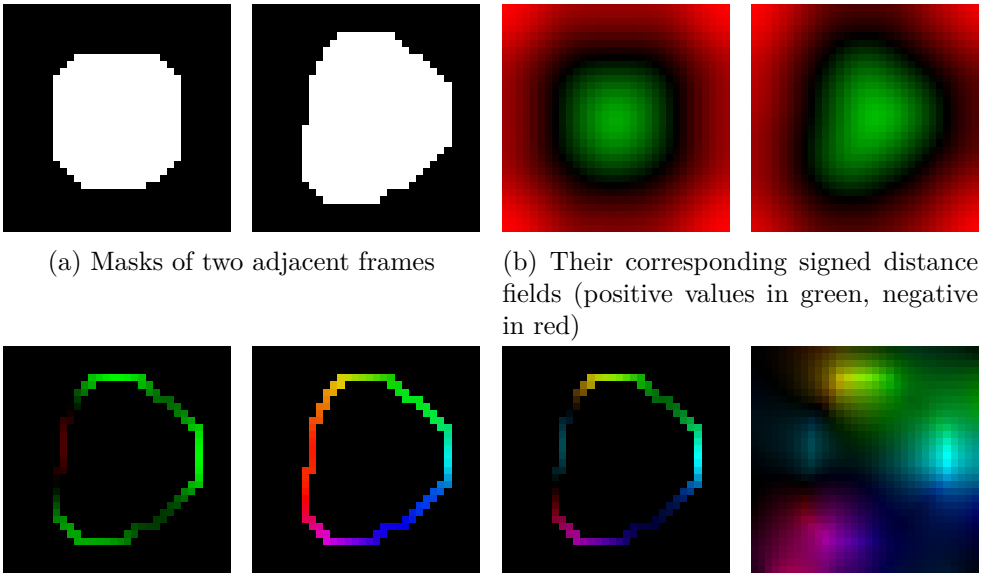
The overall algorithm is designed to allow for any source of the motion information, as long as it adheres to the assumptions posed in section 3.1.

The model implemented for our purposes is very simplifying, yet sufficiently general. Its idea is best demonstrated in a one-dimensional case first. Let B_1, B_2 be boolean masks of two adjacent frames and D_1, D_2 their corresponding signed distance fields.



Let $\Delta D = D_2 - D_1$ and D_2' be the discrete derivative of D_2. The amount of movement at border pixels of B_2, i and j, is then approximated as the product of ΔD and D'.





(a) Masks of two adjacent frames

(b) Their corresponding signed distance fields (positive values in green, negative in red)

(c) The difference of the distance fields at the second mask's border, i.e., the amount of flow

(d) The gradient of the second distance field, i.e., the flow direction (hue is direction, intensity is magnitude)

(e) The difference and gradient multiplied, i.e., the final flow at the border

(f) The final flow field obtained by diffusing the border pixels' values

Figure 3.4: Flow field construction in 2D

The smooth motion field over all interior pixels is then obtained by interpolating the values calculated at the border pixels.

In the two-dimensional case, the distance field's discrete gradient is used in place of the derivative. The interpolation of the border values is replaced with diffusion over the interior pixels. All the steps are illustrated in figure 3.4. The diffusion is calculated by solving a Poisson equation with zero right-hand-side: $\nabla^2 I_i^{motion} = \mathbf{0}$. The computed values at the border pixels are used as the boundary conditions. Computation of the diffusion was implemented using a simplified V-cycle multigrid method with no pre-smoothing and several iterations of the Gauss-Seidel algorithm for post-smoothing.

Note that even in the two-dimensional case, it is still assumed that the motion direction is aligned with the gradient of the distance field, or, in other words, that it is perpendicular to the edge of the mask. While this is a significant simplification, it usually corresponds well with most natural exemplars, where the movement is caused by diffusion.

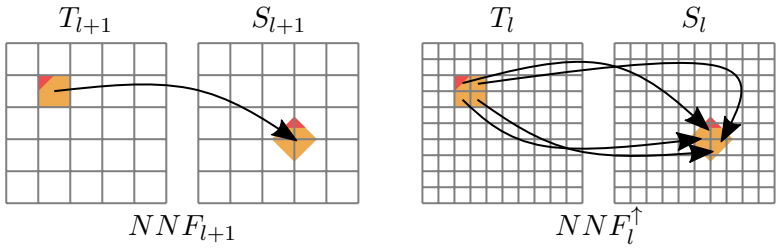


Figure 3.5: Upscaling an NNF mapping with a target window mapped to a rotated source window

3.9 Multi-scale scheme

The process described above is plugged into a multi-scale scheme described in section 2.4. The number of pyramid levels is set to $\lfloor \log_2 n_{min}/n \rfloor$, where n_{min} is the minimum dimension of the full-resolution source and target images and n is the window size. This ensures that the window size gets close to, but does not exceed, the size of the images in the coarsest level.

Since we are dealing with arbitrarily rotated mappings, the rotations have to be taken into account when calculating the upscaled x, y-coordinates. In the process of upscaling an NNF, a 2-by-2 block of pixels is generated from a single pixel in the coarser mapping (see section 2.4). Each of the four coordinates in the upscaled NNF is spatially offset by some amount from the doubled original x, y-coordinates. With transformations in place, these offsets have to be transformed by the Jacobian of the window transformation (similarly to how propagation is handled in PatchMatch), which, in our case, is simply the opposite rotation about the center of the source pixel (see figure 3.5). Thus, the final formula for the x, y-coordinates in the upscaled NNF becomes

$$NNF_l^{\uparrow}(q) = 2NNF_{l+1}(\lfloor q/2 \rfloor) + \mathbf{0.5} + R_{-\theta}(q \bmod 2 - \mathbf{0.5}),$$

where $\mathbf{0.5}$ is the vector $(0.5, 0.5)$ and R_θ is the operator of rotation by θ radians.

As noted in section 3.7, the NNF_{prev} is used to initialize the NNF mapping for each frame except the first one. To implement this principle on multiple levels, the initial NNF on level l is obtained by merging two mappings: the already described upscaled $NNF_l^{\uparrow}$ and a downscaled $NNF_{prev}^{\downarrow}$. The NNF downscaling process consists of the following steps done per each pixel $p \in NNF_{prev}^{\downarrow}$:

1. nearest neighbor coordinates are gathered from a square window from NNF_{prev} of side length 2^l with the top-left corner at $2^l p$ (figure 3.6a),

2. each (x, y) pair of these coordinates is transformed by the inverse of the upscaling transformation (figure 3.6b),

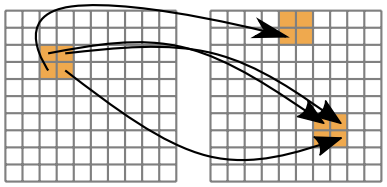
3. the mode of transformed (x, y, i, θ) coordinates is assigned as the new value of $NNF^{\downarrow}_{prev}(p)$ (figure 3.6c).

These two mappings are then merged on a per-pixel basis based on which of the two mappings has the smaller error.

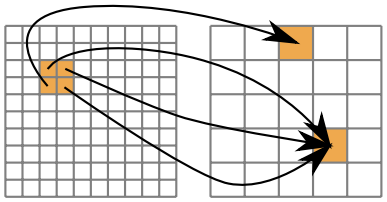
The complete algorithm per frame i thus can be described with the following pseudocode:

Algorithm 2 Multi-scale optimization

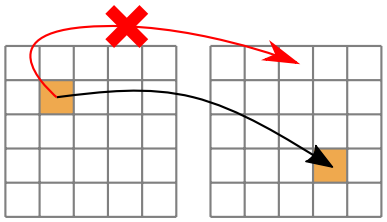
$NNF_{prev} \leftarrow$ empty mapping
$T_i \leftarrow$ empty image
for each level l **do**
  if l is the coarsest level **then**
    if first frame **then**
      $NNF_{init} \leftarrow$ random mapping
    else
      $NNF_{init} \leftarrow$ downscaleNNF(NNF_{prev})
    end if
  else
    if first frame **then**
      $NNF_{init} \leftarrow$ upscaleNNF(NNF_{l+1})
    else
      $NNF^{\uparrow}_{l} \leftarrow$ upscaleNNF(NNF_{l+1})
      $NNF^{\downarrow}_{prev} \leftarrow$ downscaleNNF(NNF_{prev})
      $NNF_{init} \leftarrow$ mergeNNF$(NNF^{\uparrow}_{l}, NNF^{\downarrow}_{prev})$
    end if
  end if
  $NNF, T^{rgb}_i \leftarrow$ optimize$(S^{rgb}, S^{guide}, S^{mask}, T^{guide}_i, T^{mask}_i, NNF_{init})$
end for
output T^{rgb}_i
$NNF_{prev} \leftarrow$ shiftNNF(NNF)



(a) A 2-by-2 block of pixels mapped to different source pixels



(b) Their coordinates transformed to the coarser level



(c) Voting on the final coordinates by majority

Figure 3.6: Downscaling an NNF mapping

Chapter 4

Implementation details

The method was implemented for CPU in C++17. Performance-critical parts were parallelized using the OpenMP API.

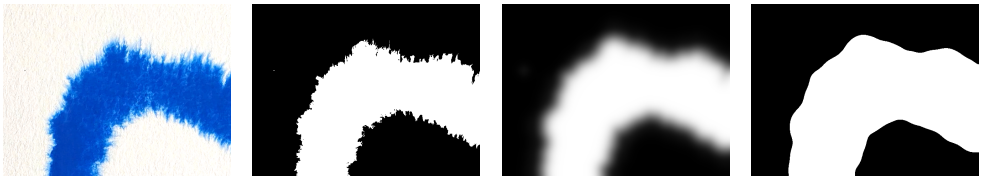
Single-precision (32-bit) floating-point values were used in image representation. Although this poses significantly larger memory size requirements than using the traditional 8-bit unsigned chars, it greatly improves flexibility when interpolating and blending the values, without significant loss of precision.

In this chapter, we will go over some of the most important implementation details and describe the usage of the implemented program.

4.1 Spatio-temporal smoothing

How the source masks are obtained is heavily dependent on the nature of the source imagery. Therefore it is left up to the user to provide them. However, in practice, several steps turned out to be common to the mask extraction process regardless of the nature of the exemplar, and have been therefore implemented as part of the program. It is desirable for a mask to have smooth edges and also to include a little of the unpainted surface surrounding the painted area. To achieve both these goals, the provided masks are convolved with a smoothing Gaussian kernel of tunable size and thresholded with a value less than 0.5 (see figure 4.1).

Boolean masks inherently are not able to capture sub-pixel movement between two consecutive frames. When such slow movement is present in the source or target sequences, our method for flow-field approximation produces highly inconsistent results. To alleviate this, an artificial smoothing in the temporal domain is introduced: prior to feeding the distance fields into the flow-field generation procedure (see section 3.8), they are first convolved with a small one-dimensional Gaussian kernel along the time axis. The difference between flow-fields with and without the temporal smoothing applied can be seen in figure 4.2.



(a) An RGB source image

(b) A corresponding user-provided mask obtained by thresholding the red channel of the RGB image

(c) The blurred mask

(d) The final mask obtained by thresholding the blurred version

Figure 4.1: Common pre-processing of boolean masks

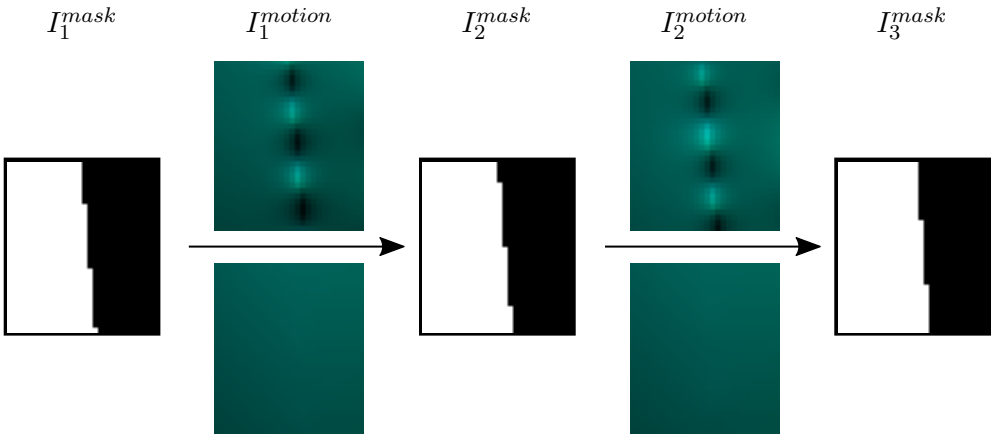


Figure 4.2: Enlarged part of three successive frames' masks with corresponding motion fields between them, without (top) and with (bottom) the distance-field temporal smoothing applied

The complete pipeline for obtaining all guidance channels is illustrated in figure 4.3.

4.2 Adaptive initialization

A random guess of both the NNF and the synthesized image is only done in the coarsest pyramid level of the first frame, as shown in section 3.9. In the following frames and pyramid levels, the initialization is obtained from previous results (through $NNF_{prev}^{\downarrow}$ and $NNF_l^{\uparrow}$). Since the whole synthesis process thus essentially lies in successive refinement of a single NNF mapping, it is desirable for the initial guess to be as good as possible. Therefore, in the final implementation, the initialization (outlined at the end of section 2.1.2) is followed by a single run of PatchMatch with a zero weight on the color

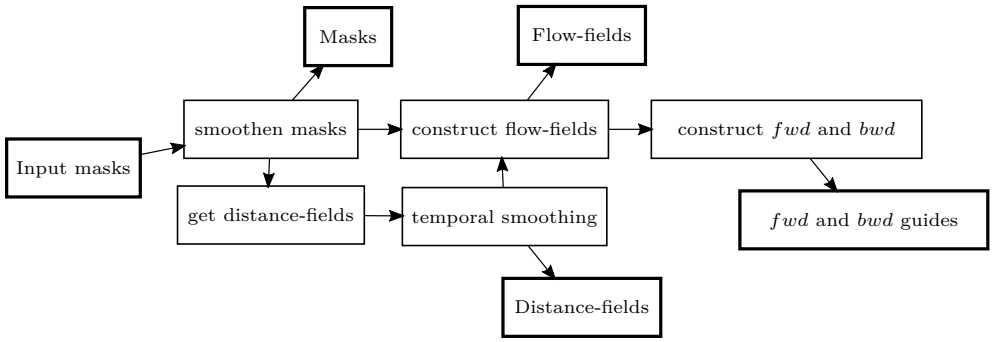


Figure 4.3: Diagram of all steps in the guidance channels' construction

channels. This way, the initial guess does not only reflect the global color distribution, but also takes into account differences of looks of different parts of an image based on information in the guidance channels.

4.3 Adaptation of the PatchMatch algorithm

The PatchMatch algorithm (see section 2.2) was used to implement the NNF retrieval in the function `findNNF`. In its propagation phase, the x and y source coordinates get offset from coordinates at a neighboring target pixel by certain amount. In our setting, there are two additional coordinates in the NNF: k for the source frame number, and θ for rotation (described in section 3.2). Both these two values get copied between neighbors without any transformation, i.e., the propagation is completely equivalent to a basic 2D case.

The four-dimensional parameter space is searched over only in the random search phase. Each candidate (x, y, k) integer triplet is randomly sampled from a cuboid of dimensions $0.75^i(x_{max}, y_{max}, k_{max})$, where i is the candidate number, x_{max} and y_{max} are the width and height of the source images, and k_{max} is the length of the source sequence. As for the rotation coordinate θ, it is actually not a part of the random search. Instead, two values are tried for each (x, y, k) triplet when attempting to improve a mapping at target pixel q in frame j: in the direction of the target flow and in the opposite direction, i.e., $\theta_T - \theta_S$ and $\theta_T - \theta_S + \pi$, respectively, where $\theta_S = \text{atan2}(S_k^{motion}(x, y))$ and $\theta_T = \text{atan2}(T_j^{motion}(q))$. Treating the rotation parameter in this way reduces the search space by one dimension, which means substantial speedup of the search. The particular choice of the two values is justified by the principles described in sections 3.6 and 3.7.

4.4 Run-time optimizations

There are many opportunities to make the program run faster than a naive sequential implementation. In general, all computation-heavy loops without

loop-carried data dependencies were parallelized. Furthermore, several steps have been made to save work, some of which slightly reduce the quality of produced results.

4.4.1 Optimization of the distance measure function

The function `patchError` evaluating the distance measure between two windows gets called in the tightest inner loops of the algorithm and runs billions of times in a typical run of the program. This makes it the single most obvious candidate for optimization and even micro-optimization.

It loops over pixels in the source and target windows and accumulates the individual per-pixel errors (see section 2.3). Most of the time it is called from the `findNNF` procedure, where its value is used for comparison with the error of the current nearest neighbor assignment. If the accumulated sum exceeds the current error during the iteration, it is already clear that the mapping will be discarded, and therefore there is no need to continue with the summation. This is implemented with an additional parameter of the `patchError` function, `errorBest`, through which the function takes the current error, and prematurely breaks from the loop once the accumulated sum of errors exceeds it. Where the exact error value is desired, `errorBest` is set to `INFINITY`.

From observation, our treatment of rotation in the random search phase of the PatchMatch algorithm (see section 4.3) is enough to produce mappings with the orientations of the flow-fields aligned (see section 3.6 for the alignment). Therefore, the orientation alignment term was intentionally left out of the final implementation of the distance computation in `patchError`. As this term was the only exception to the distance measure being an SSD, the only arithmetic operations taking place in the adjusted `patchError` function are addition and multiplication. These operations are orders of magnitude faster than computation of the tangent function present in the alignment term and are typically fused and/or vectorized by the compiler.

Another micro-optimization with a perceivable effect on the run-time was hard-coding the value of the window size. This opens up possibilities for more aggressive optimizations by the compiler, such as loop unrolling and vectorization. Thanks to the multi-level approach, smaller window sizes are sufficient (see section 2.4). The hard-coded window size in our implementation is 5×5 pixels.

4.4.2 PatchMatch parallelization

A parallelized version of the PatchMatch algorithm proposed in [35] was implemented. Despite the propagation part of the algorithm being inherently sequential, the parallel algorithm is able to produce results comparable to the sequential version with speedup nearly linear in the number of utilized CPU

cores. In each iteration of the outermost loop, each thread works on a separate vertical tile of the NNF. The propagation across the tiles' boundaries is thus delayed by one iteration. To prevent data races in accesses to values in adjacent tiles, the last row of each tile is not stored until the end of an iteration when other threads no longer read from it. The synchronization is implemented with a barrier. Following is the commented C++ code of the outer loops in our final implementation of the PatchMatch algorithm with some minor implementation details, such as bound checking, left out.

```
for(int i = 0; i < pm_iters; ++i)
#pragma omp parallel default(shared)
{
  int num_threads = omp_get_num_threads();
  int thread_num = omp_get_thread_num();

  // tile_height = ceil(target_height / num_threads)
  int tile_height = (target_height + num_threads - 1) /
     num_threads;

  int dir, begin_x, begin_y, end_x, end_y;

  // reverse propagation direction in odd iterations
  if(i % 2) {
    dir = -1;
    begin_x = target_width - 1;
    end_x = -1;
    begin_y = (thread_num + 1) * tile_height - 1;
    end_y = thread_num * tile_height - 1;
  } else {
    dir = 1;
    begin_x = 0;
    end_x = target_width;
    begin_y = thread_num * tile_height;
    end_y = (thread_num + 1) * tile_height;
  }

  // copy the last row into a thread-local storage
  // signature: slice(left, top, width, height)
  NNF last_row = nnf.slice(0, end_y-dir, target_width, 1);

  for (int y = begin_y; y != end_y; y += dir) {
    for (int x = begin_x; x != end_x; x += dir) {
      // get reference to the coordinates mapped at x, y
      Coords &coords =
        (y == end_y - dir) ? last_row(x) : nnf(x, y);

      // propagation...
      // random search...
    }
```

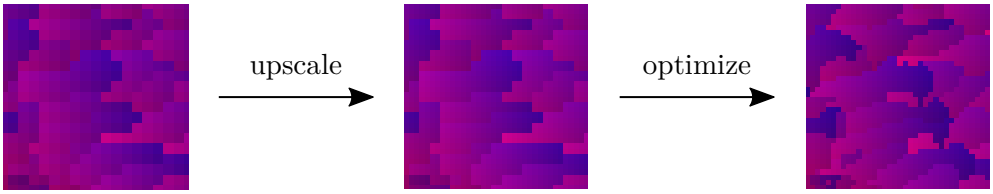


Figure 4.4: Refinement of patch boundaries through optimization on finer pyramid levels. The x any y source coordinates are encoded in the red and blue channels.

```
  }

  // store the last row back into the shared storage
  #pragma omp barrier
  nnf.slice(0, end_y - dir, target_width, 1) = last_row;
}
```

4.4.3 Tunable iteration counts

The run-time of the program is almost directly proportional to the number of iterations in the `optimize` and `findNNF` functions. Therefore, tuning these parameters and finding their lowest sufficient values is key to achieving quicker computation. Moreover, it makes sense to set different iteration counts on different pyramid levels (see section 2.4). Since the computation of coarse levels is faster, one can set a higher iteration count to provide as good an initial guess as possible for the finer levels. On the other hand, optimization on the finer levels typically tends to mostly optimize the patches' boundaries from the initial upscaled mapping through propagation with only a few improvements via the random search (see figure 4.4). Therefore, the iteration counts on finer levels can be set to lower values without a major negative effect on the result, while saving time in the most expensive parts of the computation. From experimentation, the following values of the parameters `opt_iters` and `pm_iters` were determined as the lowest sensible ones:

pyramid level	`opt_iters`	`pm_iters`
l	4	6
$l-1$	2	4
$l-2$ to 1	1	4

Furthermore, optimization on the finest pyramid level has been completely left out. Instead, the final image is obtained by voting with the upscaled NNF from the second finest level. In other words, only the initial guess of T_i^{rgb} at the finest level is output.

4.5 Program usage

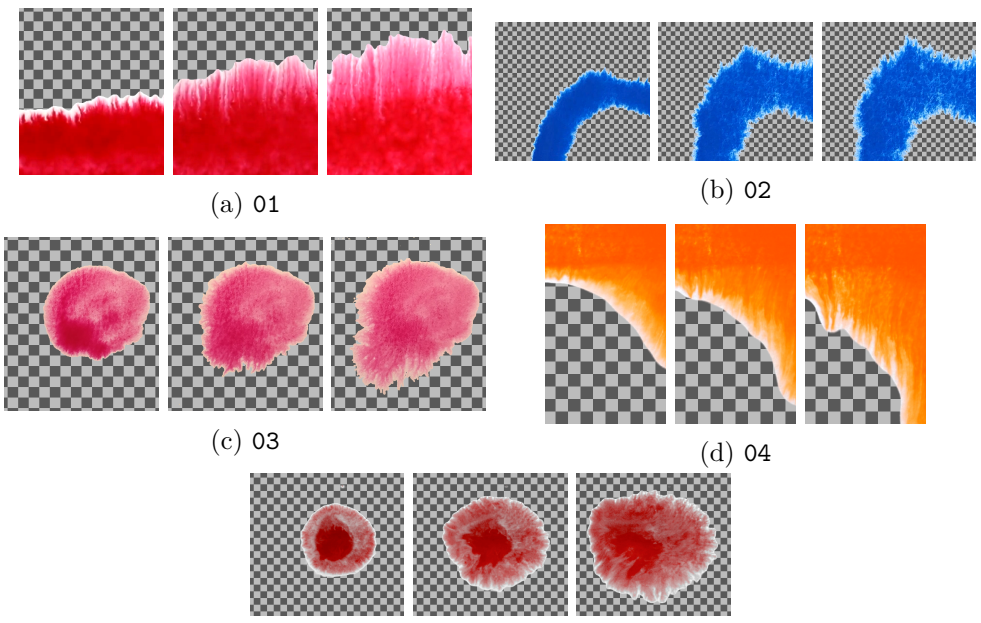
The program is run from the command line and takes all arguments in the form `--option_value`. Both the inputs and outputs are sequences of PNG images. The possible arguments are listed in the following table.

option name	default value	description
`--target-path`	*required*	`printf`-formatted paths to images, e.g., `output/%05d.png`
`--src-path`	*required*	
`--src-mask-path`	*required*	
`--out-path`	*required*	
`--src-start`	1	index of the first input frame
`--target-start`	1	
`--src-frames`	50	count of input frames
`--target-frames`	50	
`--src-speed`	1	index increment between two frames
`--target-speed`	1	
`--src-scale`	1.0	scaling factor of input frames
`--target-scale`	1.0	
`--blur-sigma`	8	standard deviation of the Gaussian filter for spatial smoothing (see section 4.1)
`--time-blur-sigma`	2	as above for temporal smoothing
`--dist-weight`	5.0	weights of the edge, flow and temporal coherence guidance (see sections 3.4, 3.5, and 3.7)
`--flow-weight`	0.1	
`--prev-weight`	1.0	
`--lambda`	2	see section 2.5
`--occ-radius`	10	see section 3.3
`--patchmatch-alpha`	0.75	see section 2.2
`--border-width`	10	in pixels; see section 3.4
`--prev-modulation-bounds`	10,2	comma-separated list of exactly two values m_u and m_l in pixels (see section 3.7)
`--patchmatch-iterations`	6,4	comma-separated list of iteration counts of PatchMatch for different pyramid levels, starting with the coarsest one; the last value gets used for all unspecified finer levels
`--opt-iterations`	4,3,2	as above for the OPTIMIZE function

Chapter 5

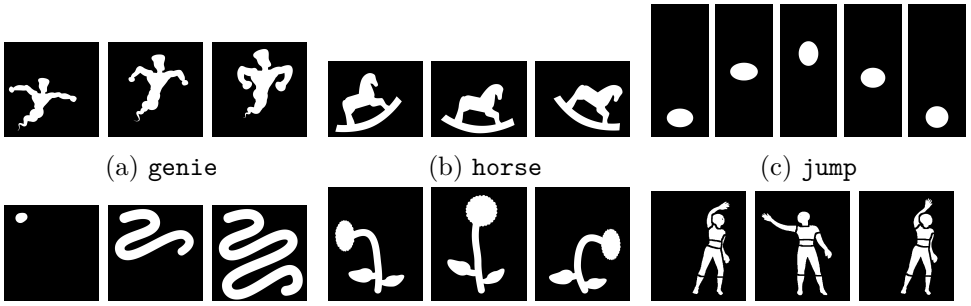
Evaluation

Five style exemplars with different characteristics (see figure 5.1) and six target animations manifesting various kinds of movement (see figure 5.2) were used to evaluate the method's capabilities. Results of all their combinations are available in the supplementary material. The outputs differ in scale, speed, and frame range of the style exemplars used. For specific values of these parameters, we refer readers to the scripts used to generate the results, which are also attached. Previews of some of the outputs are in figure 5.3.



(a) 01 (b) 02 (c) 03 (d) 04 (e) 05

Figure 5.1: Previews of the source exemplars used for evaluation. The checkerboard pattern indicates areas outside the mask.



(a) `genie` (b) `horse` (c) `jump`

(d) `squiggle` (e) `sunflower` (f) `waving`. Segmented mask provided by the authors of [41].

Figure 5.2: Previews of the target animations used for evaluation

The results show the method's ability to generalize to complex shapes not present in the exemplars and transfer finely detailed texture. Despite the interpolation of source pixels' values due to arbitrary rotation and blending of the rotated patches, the outputs generally do not suffer from loss of detail or washout. The computation times are feasible: generating the 750×600-pixel versions of the `horse` animation took approximately 2.7 seconds per frame on an Intel® Core™ i5-10400F CPU.

When dealing with a small scale movement, e.g., in the first frames of a longer animation, the temporal coherence is very good. The synthesized animation tends to locally preserve the exemplar's appearance exactly. However, our goal is to extend the limited movement in an exemplar over a whole target sequence. How well this is achievable heavily depends on the style exemplar used. For example, the exemplar `03` has only a small area with substantial movement, and thus the results tend to slightly flicker at some moving edges because no appropriate movement pattern is available in the source. Also, areas where the movement is directed inward the mask are slightly more problematic compared to regions where the target mask advances outward. This is given by the fact that the temporal coherence guidance is propagated only forward in time and therefore the exemplar's movement is not able to keep up with the desired movement in the target. Using lower values of the `prev-weight` parameter governing the temporal coherence guidance's weight results in less frequent but more obvious flickering. A fast motion that is inherently incoherent is generally handled well (see results with the `jump` animation).

Note that in the case of the style exemplar `04`, our simple flow-field approximation algorithm fails to determine the real direction of the motion. This manifests in the results as the material moving diagonally to the mask's edge.

(a) The `genie` animation stylized with the exemplar `01`

(b) The `horse` animation stylized with the exemplar `02`

(c) The `sunflower` animation stylized with the exemplar `03`

(d) The `jump` animation stylized with the exemplar `04`

(e) The `waving` animation stylized with the exemplar `05`. Segments were generated separately, colorized and composited together with a solid color background.

Figure 5.3: Previews of the results

(a) Normal result (b) Target animation reversed

Figure 5.4: The `squiggle` animation displaying unrealistic appearance of a material being added onto a canvas and a more realistic result produced when the target animation was reversed

5.1 Limitations and future work

An undesirable effect of working with complete source and target sequences is that the whole source sequence and the yet unprocessed part of the target sequence, along with all computed guidance channels, need to be stored in memory. The maximum virtual memory size utilized when synthesizing 400 frames of the `horse` animation reached up to approximately 9 GiB. This limits the dimensions and length of a sequence that could be synthesized in a single program run.

Another shortcoming of our method is that parameter selection can be quite unintuitive for a user. For example, setting too high a value of the `lambda` parameter leads to loss of spatial coherence, as the NNF retrieval tends to favor pixels with zero occurrence count over the coherent ones. Another instance of this problem is the weight of the temporal coherence guidance, which, when lowered, typically leads to increased flickering in other video synthesis methods. As mentioned above, the effect is often almost the opposite in our setting: low values of the `prev-weight` parameter can reduce high-frequency flickering at the cost of having sudden "jumps" in the resulting animation caused by replacing large areas with new content at once. Also, the `border-width` parameter needs to be set by hand for each style exemplar separately. During the research, we experimented with automatic segmentation of the source imagery into the border and the interior segments based on differing texture characteristics. However, we have not found any sufficiently reliable method that could be used with a general style exemplar without user input.

A case not addressed in this work is when a new material should be added onto the surface during the animation. It can be observed in the results obtained with the `squiggle` animation. The area under the imaginary brush looks like it has already bled into the paper (see figure 5.4a), which is the exact

opposite of what one would expect in an animation of a line being painted. Somewhat more realistic results can be obtained by generating the sequence in reverse order (see figure 5.4b). This, however, is not an intended behavior but rather a side-effect of our approach to temporal coherence enforcement. Our method could be modified to correctly handle the addition of a material onto a canvas in possible future work.

Conclusion

In this thesis, we have presented a general overview of various approaches to automated artistic stylization and style transfer for still images and video. We then focused on patch-based texture synthesis and looked at some of the most important related concepts in detail. Among those concepts were the formulation of the problem as a global optimization one, guiding the synthesis through additional image channels, different ways of preventing undesirable washout in the synthesized results, maintaining temporal coherence of an animation, and a coarse-to-fine approach to the synthesis.

Based on those concepts, we then formulated our method for example-based stylization of animation with the appearance of natural artistic media. The presented method aims to transfer a media's dynamic behavior convincingly. This was achieved with a novel approach to temporal coherence enforcement, which utilizes the movement already present in the exemplar sequence. We described all parts of the algorithm in detail, including the computation of all necessary guidance channels, washout prevention in our specific setting, and utilizing the multi-scale approach to synthesis.

The method was implemented in C++, and the results evaluated with several target animations and source styles. Implementation details were also presented as part of this thesis, with the main focus on improving the computation speed.

The results confirm that our method is able to achieve the set goals. Still, there is room for further improvement. The author believes that, despite several limitations and shortcomings, the method can serve as a basis for future research in the field.

Bibliography

[1] Haeberli, P. Paint by numbers: Abstract image representations. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 1990, pp. 207–214.

[2] Meier, B. J. Painterly Rendering for Animation. In *SIGGRAPH Conference Proceedings*, edited by J. Fujii, 1996, pp. 477–484.

[3] Hays, J.; Essa, I. A. Image and video based painterly animation. In *Proceedings of International Symposium on Non-Photorealistic Animation and Rendering*, edited by A. Hertzmann; C. S. Kaplan; S. N. Spencer, 2004, pp. 113–120.

[4] Hertzmann, A. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In *SIGGRAPH Conference Proceedings*, edited by S. Cunningham; W. Bransford; M. F. Cohen, 1998, pp. 453–460.

[5] Hertzmann, A. Paint By Relaxation. In *Proceedings of Computer Graphics International*, edited by H. H.-S. Ip; N. Magnenat-Thalmann; R. W. H. Lau; T.-S. Chua, 2001, pp. 47–54.

[6] Montesdeoca, S. E.; Seah, H. S.; et al. MNPR: a framework for real-time expressive non-photorealistic rendering of 3D computer graphics. In *Expressive '18: Proceedings of the Joint Symposium on Computational Aesthetics and Sketch-Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*, edited by B. Wyvill; H. Fu, 2018, pp. 9:1–9:11.

[7] Bousseau, A.; Kaplan, M.; et al. Interactive watercolor rendering with temporal coherence and abstraction. In *Proceedings of International Symposium on Non-Photorealistic Animation and Rendering*, edited by D. DeCarlo; L. Markosian, 2006, pp. 141–149.

[8] Bousseau, A.; Neyret, F.; et al. Video watercolorization using bidirectional texture advection. *ACM Transactions on Graphics*, volume 26, no. 3, 2007: pp. 104:1–104:7.

[9] Breslav, S.; Szerszen, K.; et al. Dynamic 2D patterns for shading 3D scenes. *ACM Transactions on Graphics*, volume 26, no. 3, 2007: pp. 20:1–20:5.

[10] Curtis, C. J.; Anderson, S. E.; et al. Computer-generated Watercolor. In *SIGGRAPH Conference Proceedings*, volume 31, 1997, pp. 421–430.

[11] Adobe Inc. Fresco. 2019. Available from: `https://www.adobe.com/products/fresco.html`

[12] Escape Motions, s.r.o. Rebelle. 2015. Available from: `https://www.escapemotions.com/products/rebelle/about`

[13] Hertzmann, A.; Jacobs, C. E.; et al. Image analogies. In *SIGGRAPH Conference Proceedings*, 2001, pp. 327–340.

[14] Efros, A. A.; Leung, T. K. Texture Synthesis by Non-parametric Sampling. In *Proceedings of IEEE International Conference on Computer Vision*, 1999, pp. 1033–1038.

[15] Barnes, C.; Zhang, F.-L. A survey of the state-of-the-art in patch-based synthesis. *Computational Visual Media*, volume 3, no. 1, 2017: pp. 3–20.

[16] Efros, A. A.; Freeman, W. T. Image quilting for texture synthesis and transfer. In *SIGGRAPH Conference Proceedings*, edited by L. Pocock, 2001, pp. 341–346.

[17] Kwatra, V.; Schödl, A.; et al. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics*, volume 22, no. 3, 2003: pp. 277–286.

[18] Sýkora, D.; Jamriška, O.; et al. StyleBlit: Fast Example-Based Stylization with Local Guidance. *Computer Graphics Forum*, volume 38, no. 2, 2019: pp. 83–91.

[19] Wexler, Y.; Shechtman, E.; et al. Space-Time Video Completion. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2004, pp. 120–127.

[20] Kwatra, V.; Essa, I. A.; et al. Texture optimization for example-based synthesis. *ACM Transactions on Graphics*, volume 24, no. 3, 2005: pp. 795–802.

[21] Kaspar, A.; Neubert, B.; et al. Self Tuning Texture Optimization. *Computer Graphics Forum*, volume 34, no. 2, 2015: pp. 349–359.

[22] Fišer, J.; Jamriška, O.; et al. StyLit: illumination-guided example-based stylization of 3D renderings. *ACM Transactions on Graphics*, volume 35, no. 4, 2016: pp. 92:1–92:11.

[23] Jamriška, O.; Fišer, J.; et al. LazyFluids: appearance transfer for fluid animations. *ACM Transactions on Graphics*, volume 34, no. 4, 2015: pp. 92:1–92:10.

[24] Bénard, P.; Cole, F.; et al. Stylizing animation by example. *ACM Transactions on Graphics*, volume 32, no. 4, 2013: pp. 119:1–119:12.

[25] Fišer, J.; Jamriška, O.; et al. Example-based synthesis of stylized facial animations. *ACM Transactions on Graphics*, volume 36, no. 4, 2017: pp. 155:1–155:11.

[26] Jamriška, O.; Sochorová, Š.; et al. Stylizing video by example. *ACM Transactions on Graphics*, volume 38, no. 4, 2019: pp. 107:1–107:11.

[27] Kwatra, V.; Adalsteinsson, D.; et al. Texturing Fluids. *IEEE Transactions on Visualization and Computer Graphics*, volume 13, no. 5, 2007: pp. 939–952.

[28] Gatys, L. A.; Ecker, A. S.; et al. Image Style Transfer Using Convolutional Neural Networks. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016: pp. 2414–2423.

[29] Texler, O.; Futschik, D.; et al. Arbitrary style transfer using neurally-guided patch-based synthesis. *Computers & Graphics*, volume 87, 2020: pp. 62–71.

[30] Wang, T.-C.; Liu, M.-Y.; et al. Video-to-Video Synthesis. In *Proceedings of International Conference on Neural Information Processing Systems*, edited by S. Bengio; H. M. Wallach; H. Larochelle; K. Grauman; N. Cesa-Bianchi; R. Garnett, 2018, pp. 1152–1164.

[31] Ashikhmin, M. Synthesizing natural textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, 2001, pp. 217–226.

[32] Bentley, J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, volume 18, no. 9, 1975: pp. 509–517.

[33] Chen, Y.-S.; Hung, Y.-P.; et al. Winner-Update Algorithm for Nearest Neighbor Search. In *Proceedings of International Conference on Pattern Recognition*, 2000, pp. 2704–2707.

[34] Barnes, C.; Shechtman, E.; et al. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics*, volume 28, 2009: pp. 24:1–24:11.

[35] Barnes, C.; Shechtman, E.; et al. The generalized patchmatch correspondence algorithm. In *Proceedings of European Conference on Computer Vision*, 2010, pp. 29–43.

[36] Darabi, S.; Shechtman, E.; et al. Image melding: combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics*, volume 31, no. 4, 2012: pp. 82:1–82:10.

[37] Kopf, J.; Fu, C.-W.; et al. Solid texture synthesis from 2D exemplars. *ACM Transactions on Graphics*, volume 26, no. 3, 2007: pp. 2:1–2:9.

[38] Simakov, D.; Caspi, Y.; et al. Summarizing visual data using bidirectional similarity. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 2008, pp. 1–8.

[39] Fišer, J.; Lukáč, M.; et al. Color Me Noisy: Example-based Rendering of Hand-colored Animations with Temporal Noise Control. *Computer Graphics Forum*, volume 33, no. 4, 2014: pp. 1–10.

[40] Meijster, A.; Roerdink, J. B.; et al. A general algorithm for computing distance transforms in linear time. In *Mathematical Morphology and its applications to image and signal processing*, Springer, 2002, pp. 331–340.

[41] Dvorožňák, M.; Li, W.; et al. Toonsynth: example-based synthesis of hand-colored cartoon animations. *ACM Transactions on Graphics*, volume 37, no. 4, 2018: pp. 167:1–167:11.

Appendix A

Contents of enclosed CD

- `code` ..... the directory of source codes
- `samples` ..... sample style exemplars and target animations
  - `source` ..... the style exemplars
  - `target-mask` ..... the target animations' masks
  - `generate_all.sh` ..... the script generating all sample combinations
- `thesis` ..... the directory of LaTeX source codes of the thesis
  - `DP_Adam_Platkevic_2020.pdf` ..... the thesis text in PDF format
- `video` ..... the directory with video previews of the results