Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studiijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

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

Master's Thesis

Intelligent Algorithms for Image Inpainting

Bc. Jakub Fišer

Supervisor: Ing. Daniel Sýkora, Ph.D.

Study Programme: Open Informatics

Field of Study: Computer Graphics and Interaction

May 2012

Aknowledgements

I would like to thank to my supervisor, Daniel Sýkora, for patient guidance throughout the whole process of creation of this thesis. Also, I would like to thank to my relatives and especially to my girlfriend, who all were there for me when I needed them most.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on May 10, 2012 .....

Abstract

The problem of synthesis of missing image parts represents an interesting area of image processing with significant potential. This thesis focuses on methods addressing the image inpainting problem using the information contained in the rest of the image. Selected methods are discussed in more detail, implemented and tested on different data sets.

Abstrakt

Problém syntézy chybějících částí obrazu představuje zajímavou oblast na poli zpracování obrazu se značným potenciálem. Tato práce se zabývá popisem situace na poli metod pro řešení zmíněné problematiky s využitím informace obsažené ve viditelné části obrazu. Vybrané metody jsou podrobněji diskutovány, implementovány a otestovány na různých datových sadách.

Contents

1 Introduction **1**
1.1 Problem description . . . 1
1.2 State-of-the-art . . . 1
1.3 Applications . . . 2
1.4 Terminology . . . 2

2 Algorithms for fast NN search **5**
2.1 Distance metrics . . . 5
2.1.1 Handling unknown pixels . . . 6
2.2 Early termination . . . 7
2.3 SSE . . . 7
2.4 Hierarchical approach . . . 8
2.4.1 Anti-aliasing filters . . . 8
2.4.2 Disadvantages . . . 8
2.4.3 Gaussian pyramid . . . 9
2.4.4 Hierarchical approaches on images with unknown pixels . . . 9
2.5 Phase correlation . . . 10
2.6 Sequential overlap exploitation . . . 12
2.6.1 Single column processing . . . 13
2.6.2 Extensions . . . 14
2.6.3 Limitations . . . 14
2.7 PatchMatch . . . 14
2.7.1 Approximate nearest-neighbor algorithm . . . 15

3 Selected algorithms **19**
3.1 Method of Efros and Leung . . . 19
3.1.1 Algorithm . . . 19
3.1.2 Summary . . . 20
3.2 Method of Averbuch et al. . . . 21
3.2.1 Algorithm . . . 21
3.2.2 Distance metric . . . 21
3.2.3 Search structure . . . 21
3.2.4 Summary . . . 23
3.3 Method of Criminisi et. al . . . 23
3.3.1 Key observations . . . 24

3.3.2 Algorithm . . . 24
3.3.3 Summary . . . 27
3.4 Method of Kwok et al. . . . 27
3.4.1 Theoretical background . . . 27
3.4.2 Search structure . . . 30
3.4.3 Algorithm . . . 31
3.4.4 Parallelization on GPU . . . 31
3.4.5 Summary . . . 32
3.5 Method of Simakov et al. . . . 32
3.5.1 Summarizing visual data using bidirectional similarity . . . 32
3.5.2 Incorporating PatchMatch . . . 35

4 Implementation **37**
4.1 OpenCV . . . 37
4.2 Framework overview . . . 37
4.2.1 Image class . . . 38
4.2.2 Input/output class . . . 39
4.2.3 IPaper interface . . . 39
4.2.4 ISampler interface . . . 40
4.2.5 ISearcher interface . . . 40
4.2.6 ISolver interface . . . 41
4.2.7 Matrix operations . . . 41
4.3 Samplers . . . 41
4.3.1 Scanline sampler . . . 41
4.3.2 Onion peel sampler . . . 42
4.3.3 Fill ratio sampler . . . 43
4.3.4 Priority sampler . . . 43
4.4 Searchers . . . 46
4.4.1 Exhaustive searcher . . . 47
4.4.2 Patch vector searcher . . . 47
4.4.3 Phase correlation searcher . . . 48
4.4.4 Fast query searcher . . . 49
4.5 PatchMatch . . . 49
4.6 Papers . . . 50
4.6.1 Simakov . . . 50

5 Results **53**
5.1 Algorithm settings . . . 53
5.2 Real-world images . . . 54
5.2.1 Scratch-like holes . . . 54
5.2.2 Large holes . . . 55
5.3 Cartoon graphics . . . 58
5.4 PatchMatch . . . 59
5.5 Performance comparison . . . 59
5.5.1 SSE instruction set . . . 60
5.5.2 Phase correlation . . . 60

5.6 Time performance/visual quality 60
5.6.1 Patch size 60
5.6.2 Constrained source region 62
5.6.3 Hierarchical approach 62
5.7 Comparison with consumer applications 63

6 Conclusion **75**
6.1 Future work 75

A Note on color spaces **81**

B List of used abbreviations **83**

C Installation manual **85**

D Content of the accompanying CD **87**

List of Figures

2.1 The example of target and source zones . . . 6
2.2 Illustration of Lanczos filter properties . . . 9
2.3 The example of Gaussian pyramid . . . 10
2.4 The unwanted blur effect in both image and its mask caused by direct application of Gaussian pyramid approach . . . 11
2.5 Fast exact nearest patch matching using the sequential overlap . . . 13
2.6 The principle of propagating correspondences in PatchMatch algorithm . . . 14
2.7 Phases of the PatchMatch algorithm . . . 15

3.1 Overview of algorithm of Efros and Leung . . . 20
3.2 Problems of concentric-layer filling order . . . 24
3.3 Structure propagation along isophote . . . 25
3.4 Relationship of patch and its normal and isophote . . . 26
3.5 Comparison of different transformation methods used on highly textured image samples . . . 29
3.6 The bidirectional similarity measure . . . 33
3.7 Notations for the update rule of the bidirectional similarity summarization algorithm . . . 34

4.1 Illustration of the image handling within the framework . . . 39
4.2 Flowchart of simple solver scheme . . . 41
4.3 Illustration of sampling in spiral manner . . . 42
4.4 Contour normal estimation using contour pixels only . . . 44
4.5 Contour normal estimation using distance transform and robust gradient operator . . . 44
4.6 . . . 46
4.7 Illustration of the unrolled linked list . . . 48
4.8 The example of patch offset mask . . . 50
4.9 Examples of possible patch-weighting functions . . . 51

5.1 Test image “Interview” . . . 54
5.2 Test image “Beach Text” . . . 55
5.3 Number of elements stored in lists of proposed search structure . . . 55
5.4 Test image “Bungee Jumper” . . . 56
5.5 Test image “Palace” . . . 57
5.6 Test image “Horse” . . . 57

5.7 Test image “Fruits” 58
5.8 Test image “Bear” 58
5.9 Test image “Fairy Tale” 59
5.10 Utilizing SSE instruction set 61
5.11 Comparison of exhaustive patch search vs. phase correlation based approach 62
5.12 Test image”Elephant” 63
5.13 Test image”Pumpkin” 64
5.14 Test image”Battlements” 65
5.15 Test image”Sign” 67
5.16 Test image”Sign” 68
5.17 Hierarchical approach, test image “Baby” 69
5.18 Hierarchical approach, test image “Monkey” 69
5.19 Hierarchical approach, test image “Car” 70
5.20 Hierarchical approach, test image “Ship” 70
5.21 Comparison with reference output, test image “Chairs” 71
5.22 Comparison with reference output, test image “Elephant” 71
5.23 Comparison with reference output, test image “Sign” 72
5.24 Comparison with reference output, test image “Pumpkin” 72
5.25 Comparison with reference output, test image “Bungee Jumper” 73

A.1 Example of misleading interpretation of color distance 81

List of Tables

3.1 Initialized search data structure 22
3.2 Query examples on the search data structure 22

5.1 Performance of different methods of NN search. 60
5.2 Time performance of selected methods with changing patch size 66
5.3 Time performance of selected methods with changing size of source region . . 66

Chapter 1

Introduction

1.1 Problem description

Completion of missing parts in various images has become quite an interesting area and challenging problem in computer graphics as well as in computer vision. While human eyes can often see the visually plausible solution of missing image part using a „global insight", finding an algorithm capable of real-time performance with minimal user-given guidance remains the task to be solved. In this thesis, some of the algorithms that aim to solve this problem were implemented.

Image completion (often referred to as image inpainting) is a process of filling specified parts in image in a visually plausible way. This, perhaps little vague, definition does not imply any other specific requirements or conditions on the resulting image than it should be visually coherent. However, it does not determine, e.g., which part of the image the samples for inpainting shall be taken from or any other algorithm-dependent specification. To formally define the problem, intuitive scheme may arise - the following scheme or notation is widely used, e.g., in [11, 25]: The input image $\mathcal{I}$ is given. The user then specifies the *target region* Ω, i.e., the missing, corrupted or unknown part of the image. Also a *source region* may be specified (if the algorithm uses the known part of the image) as $\Phi = \mathcal{I} - \Omega$. However, the rest of the image might not be used as source at all like, e.g., in [17].

1.2 State-of-the-art

Current approaches can be roughly divided into two groups. The first one can be referred to as *exemplar-based* techniques. Such algorithms search for patches in known area of the image and match them locally to find best match which is then copied into the unknown region of the image. This search of best match is performed iteratively until no missing pixels remain.

Starting with work of Efros [13], many algorithms have been proposed to improve the visual appearance of the resulting image or/and the time complexity required to achieve the solution. Criminisi et al. [11] proposed a priority order to emphasize the propagation of linear structures. Later, Ting et al. [28] and Komodakis [24] and Wexler [29] proposed a global-optimized approach to overcome the possible visual inconsistencies that may come

from use of greedy algorithm. Drori et al. [12] introduced an iterative method using adaptive example fragments, however it is relatively slow. Yang et al. [31] extended the Priority Belief Propagation approach used by Komodakis by formulating the algorithm called Structural Priority Belief Propagation and improved the usability of large displacement view (LDV) completion algorithms. Averbuch [3] presented a vector-based search structure for fast comparison of incomplete patches. Barnes et al. [4] introduced new randomized algorithm for finding correspondences that dramatically reduced the computation time and could be used in various image manipulation techniques. Kwok et al. [25] proposed new method to search faster over the set of candidates and sped up the Criminisi's scheme.

The latter group can be called *inpainting*. In most cases these algorithms use partial differential equations. Inpainting itself was first introduced by Bertalmio et al. [7]. Their method propagates image Laplacians from the known areas of the image inwards, in the isophote direction and also treats the inpainting process as the fluid dynamics problem [6]. Chan [10] proposed the use of Euler's elastica as a hint of processing of curved structures. These methods usually work well for small, scratch-like holes, however, for larger missing regions, these algorithms may tend to generate blurry artifacts.

1.3 Applications

Practical usage of image completion brings useful and interesting applications to the end-user. Inpainting can be used in various areas, such as:

- Texture synthesis - to fill the target region with visually plausible texture from given (usually smaller) sample
- Image restoration - to remove scratches or corrupted areas on old photos or, e.g., scanned paintings of old masters
- Image retouching - to erase unwanted parts from the image or even to switch positions of certain objects in the image or their aspect ratios (image reshuffling and retargeting)

Some of these algorithms are used in commercial widely spread software applications. Probably the most known example is the Adobe Photoshop CS5, which utilizes the work of Barnes [4], and its content-aware fill function. Prior to the CS5 version, older versions of Photoshop (first introduced in version 7.0) implemented tool known as "healing brush" [14] which used iterative algorithm based on partial differential equation (PDE). Also the most popular non-commercial software - the GNU Image Manipulation Program, known as GIMP - provides its own implementation of image inpainting or texture generation via the plug-in based on PhD. thesis of Harrison [16].

1.4 Terminology

Since often slightly different terminology is used in various works, the list of unified terminology is presented here, to be more consistent throughout the description of theoretical background of described algorithms and methods:

- Ω - Target/masked region/zone - area of the image that shall be replaced; the content of its pixels must not affect the resulting image in any way
- Φ - Source/allowed region/zone - are of the image that shall serve as the source of patches to fill the missing part of the image
- Ψ_p- Patch - square, centrally symmetric square neighborhood of a pixel p with this pixel placed in the center
- Best exemplar or best matching patch - patch with minimal distance (computed using given metric) to another - query - patch
- Patch size - size of the side of the patch; since the probed pixel is placed in the center of the patch, the size is typically odd number
- Patch radius - half of the patch size or more precisely $patchRadius = \frac{patchSize-1}{2}$, since the patch size is odd number

Chapter 2

Algorithms for fast NN search

In exemplar based methods, that are in particular focus of this thesis, patch-based, respectively pixel-based, comparison is needed to determine the nearest neighbor (NN) of given patch. Lets denote the patch, for which the nearest neighbor shall be found, as Ψ_p (the subscript p denotes the center pixel of the patch). Its NN patch Ψ_q is then obtained as follows:

$$\Psi_q = \arg\min_{\Psi \in \Phi} d(\Psi_p, \Psi),$$

where Φ denotes the set of all potential candidates (usually the whole image minus the area marked as to-be-synthesized) and d some perceptual distance metric.

Since the time-complexity of exhaustive search over the complete search space grows with increasing sizes of the image and the patch window, several techniques may be employed to speed the searching process up. Doing so can usually greatly improve the performance of given algorithm because the patch-to-patch comparisons are, specially in exemplar-based methods, the most frequent operations. This techniques are discussed later in this chapter.

2.1 Distance metrics

Before the nearest neighbor search methods will be discussed, let us first make a note about the distance metrics used for patch-to-patch comparison. As already stated, patch-to-patch comparison lie usually at the "core" of every exemplar-based approach for image synthesis. Therefore the distance metric, evaluating the distance between two given patches, should be fast enough yet precise to give satisfactory results. In most cases one of these metrics/norms is used:

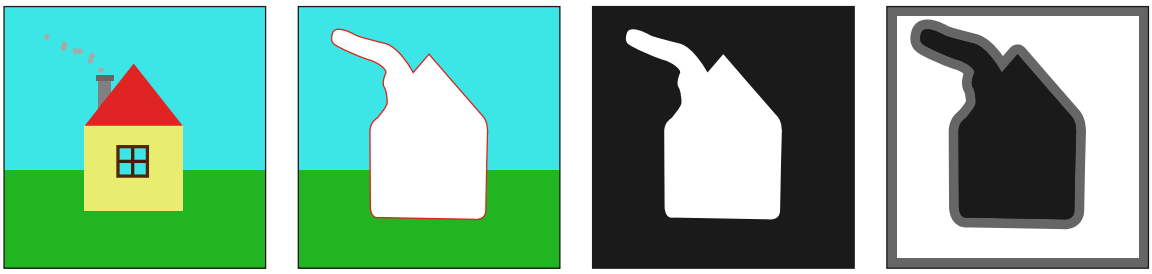
- Sum of Absolute Differences (SAD) - the distance of two patches/pixels is given as a sum of absolute differences of their corresponding pixels/channels
- Sum of Squared Distances (SSD, L_2) - the distance of two patches/pixels is given as a sum of squared differences of their corresponding pixels/channels
- Maximum Norm (L_∞) - the distance of two patches/pixels is given as a maximal differences of two of their corresponding pixels/channels

2.1.1 Handling unknown pixels

When dealing with incomplete images, i.e., images containing unknown pixels, the target patch, and possibly also the source patch that are to be compared, can contain some of these missing pixels and simply comparing pixels on corresponding positions within both patches would compromise the error distance computation. To overcome this obstacle, two approaches can be chosen:

By constraining the source region to only those pixels with fully known patch neighborhood, one can guarantee that only the same number of pixels would be compared. In fact, only valid (known) pixels from query patch are used and there is an assurance they will always have corresponding pixel in such a source patch. The patches in certain distance (at least equal to half of the patch size) from boundaries and from missing regions are simply not taken into account - see fig. 2.1. While removing a little portion of search space, this approach simplifies and speeds up the comparison process, since only pixels in target region patches must be checked whether they are sources of valid information or are unknown.

The latter option is to normalize the patch distance per valid pixel couples. This method does not restrict the source region at all because it allows even incomplete patches to be source of information. However, there are some disadvantages. The first and most significant is the fact that the error distance between patches must be computed as relative value (i.e., a per-pixel error). This situation comes from the observation that different numbers of valid pixel-pairs (one pixel from the source patch and the other one from the target patch) are being compared. The second disadvantage is the possibility of comparing two non-overlapping patches (meaning there are no valid pixel-to-pixel correspondences) but this can be avoided simply by setting and checking the minimal required number of valid pixel-to-pixel correspondences within both patches.



a b c d

Figure 2.1: The example of target (masked) zone and source (allowed) zone. (a) The original image to be processed. (b) The area to be removed and re-synthesized is selected. (c) The binary mask is obtained. White color denotes masked area of the image. (d) The allowed zone is obtained as a negation of mask and possibly (depending on the algorithm) the dilated bounds may also be removed from being considered as valid sources of information (gray band). White pixels are the centers of fully known patches.

2.2 Early termination

Early termination is a method applicable in a great number of computer science problems. This simple methods allows us for given set of candidates to terminate the computation of some value, that is at the center of our interest, if the current value, based on some features of currently processed candidate, loses at some point of the process of its computation the chance to become the new best candidate. To be able to terminate such a computation, two requirements must be met. First, we must keep track of the best score achieved so far. Secondly, the process of computing the score for given candidate must guarantee, that once the computed value gets over or under (depending on whether we seek for maximum or minimum value) the so-far best value, it can never reach it again. This can be formally written down as suggested in alg. 1.

If the early termination is to be used on source patches, which may contain unknown pixels (and the minimal error value ε_{min} found so far is therefore stored as a relative per-pixel value), the estimate must be first computed as $\varepsilon_{min} \cdot |\Psi|$, where $|\Psi|$ is the area of the patch, i.e., simply the square of patch size. Hence, the fact, that currently examined patch gives worse result than the so-far-minimum, can sometimes be found only at the end of the comparison, when the absolute error is divided by number of valid pixels to get the per-pixel value.

Algorithm 1 The scheme of early termination technique. The input parameter C denotes the candidate and ε_{best} the best value found so far examining the candidates prior to C.

ComputeValue$\varepsilon(C,\varepsilon_{best})$

```
ε ←initialization value
repeat
    Update required variables
    if the value of ε can no longer be the best value then
        return Some specific value to notify that candidate C did not succeed
    end if
until computation of ε is done
return ε
```

2.3 SSE

SSE, Streaming SIMD[1] Extensions, is a instructions set to x86 architecture designed to make use of *data level parallelism*. It is very helpful when the same operations shall be performed over and over on the different data. SSE vectors are 128-bits wide, thus allowing to process 16 8-bit chars at once (only for some operations). Since the patch-to-patch comparisons is such a data-parallel operation (large amount of single pixel values is processed the very same way), it seems to be a good choice of use. In particular interest stands the instruction (and its intrinsic equivalent)

```
PSADWB __128i _mm_sad_epu8(__m128i a, __m128i b),
```

[1]Single Instruction, Multiple Data - different data are processed simultaneously on multiple processing elements performing the same operations.

Packed Sum of Absolute Differences, which computes the absolute difference of the 16 unsigned 8-bit integers from the first operand a and the 16 unsigned 8-bit integers from the second operand b. Hence the available distance metric is the sum of absolute differences (SAD).

To be able to make use of the intrinsic functions, the data must be placed on16-byte-aligned memory address. Thus, all the fully patches are copied into one consecutive memory chunk.

2.4 Hierarchical approach

Multi-resolution schemes have certainly proved useful in image processing. The principle is simple. Instead of searching over the whole image (in possibly large resolution), the search space is repeatedly downsampled. The lowest level - the smallest image in the so called pyramid - is then examined and the solution is propagated up serving as an initial guess or hint on the next level of the pyramid. This approach is used in many image-processing-related fields of interest [2], especially in texture synthesis [18].

The downsampling process, i.e., the (possibly repetitive) reduction of the image, aims to reduce the higher image frequencies to diminish the number of samples needed to represent the original signal. Normally, when a signal is downsampled, the high-frequency part of the signal is aliased with the low-frequency one. However, the desired outcome is to keep only the low-frequency part and so the image must be preprocessed (alias-filtered) to prevent the remove the high-frequency part and therefore to avert the occurrence of aliasing.

2.4.1 Anti-aliasing filters

The optimal filter to remove the high-frequency part of the image would be the *sinc filter* with sinc function (standing for "sine cardinal"),

$$\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x},$$

as its impulse response in the time domain. Such filter removes all frequency components above a given bandwidth and preserves all low-frequency ones. However, since the this idealized filter has infinite impulse response in both positive and negative time directions, it is not directly usable and must be approximated for real-world applications and processes.

A windowed form of the sinc filter is the *Lanczos filter*, which attenuates the sinc coefficients and truncates them as the values drop to insignificance. Its impulse response is the normalized Lanczos window (also called sinc window). The Lanczos window is the central lobe of a horizontally-stretched sinc, $\operatorname{sinc}\left(\frac{x}{a}\right)$ for $-a \leq x \leq a$. Examples of two and three-lobed Lanczos-windowed sinc function are presented in fig. 2.2.

2.4.2 Disadvantages

While the hierarchical approaches favor speed, there are obvious drawbacks of these methods. Depending of how many downscaling iterations are performed, the significant amount of

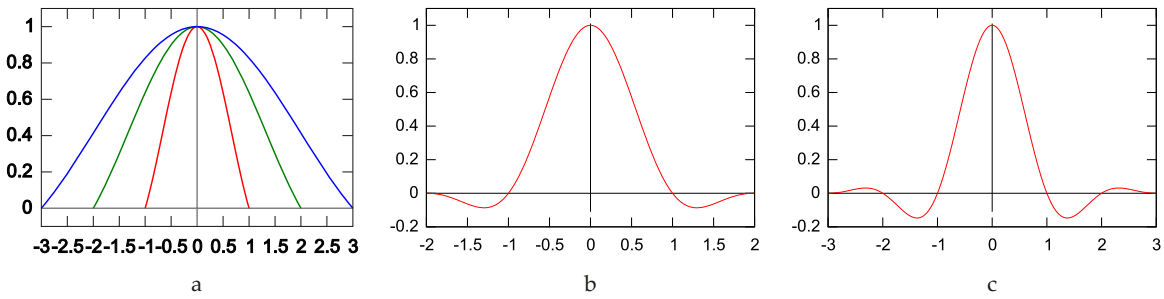


Figure 2.2: Illustration of Lanczos filter properties; image taken from <`http://en.wikipedia.org`>. (a) Extent of Lanzcos windows for $a = 1, 2, 3$. (b) Two-lobed Lanczos-windowed sinc function ($a = 2$). (c) Three-lobed Lanczos-windowed sinc function ($a = 3$).

details can be lost and thus the search process can be trapped in local minima and the global optimum may never be found. Therefore, both the number of downsampling steps and the downscale-ratio should be chosen carefully.

2.4.3 Gaussian pyramid

The Gaussian pyramid consists of set of images that are scaled down using a gaussian filter. Starting on the top level with the original image, the next image on lower level is created as a low-pass filtered and downsampled version of the previous image. More formally, the Gaussian pyramid consisting of L levels $l \in \{0, \ldots, L-1\}$ for given image $I(x, y)$ is defined recursively as follows:

$$G_0(x, y) = I(x, y) \text{ on the highest level, } l = 0, \text{ and}$$

$$G_l(x, y) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m, n)\, G_{l-1}(2x + m, 2y + n) \text{ on levels } l \in \{1, \ldots, L-1\},$$

where $w(m, n)$ is the weighting function, *generating kernel*, that remains the same on all levels of the pyramid and must be separable and symmetric. This kernel is an approximation of Gaussian function (thus the name of the pyramid). An example of usable weighting function is a 5-tap filter $\frac{1}{16}\begin{pmatrix} 1 & 4 & 6 & 4 & 1 \end{pmatrix}$. Fig. 2.3 shows example of Gaussian pyramid obtained using such a kernel.

2.4.4 Hierarchical approaches on images with unknown pixels

When used on image with unknown pixels, image pyramid approaches suffer particularly from one difficulty. Say that the unknown pixels of an input images are first, immediately after loading, cleared with some value to assure that the context in this image area does not affect the image synthesis in any way. Let this value be 0, i.e., the hole in the image will appear black, see fig. 2.4a. When filtering and downsampling the image the "standard way" as described in previous section, the edges of the unknown area both in the original image and its mask become more and more blurry as shown on fig. 2.4f,g,h.

Figure 2.3: The example of Gaussian pyramid. From (a) to (f), the original image (the 0th level of the pyramid) of size 600×400 is downscaled up to the size 19×13.

Of course, one could easily consider every pixel in the downsampled mask that is not zero (i.e., black thus unmasked) as masked one. However, this would lead to slight yet unwanted expansion of the mask during the pyramid creation. To overcome this obstacle, only known pixels must be convolved and the ratio of known/unknown pixels must be maintained during the process. When it reaches certain ratio, the pixel is accepted as valid and the mask is reset (thus signaling the pixel is known). Otherwise, the mask is set and the pixel is considered to be unknown.

Considering the masked pixels as a special case and excluding them from the convolution process guarantees that boundaries between the valid and masked areas of the image stay sharp as shown on fig. 2.4j,k,l. In fact, careful setting of the minimal ratio of valid pixels within the convolution mask (25% is the value used in implementation) causes the gradual ingrowth of the mask.

2.5 Phase correlation

Phase correlation is a method of image processing to check the similarity of two images (of equal proportions) that makes use of fast frequency-domain processing. It can be used to solve various problems such as motion estimation, object tracking or template matching. Therefore, the idea is to try to use it in image completion process as a variant of nearest neighbor search.

The method is based on Fourier Shift Theorem [8] and was originally proposed for the registration of translated images. The theorem states that the Fourier transform of a convolution is the pointwise product of Fourier transforms:

$$\mathcal{F}\{a * b\} = \mathcal{F}\{a\} \cdot \mathcal{F}\{b\}.$$

Moreover, the convolution of a function f with the Dirac δ function leads to replication of

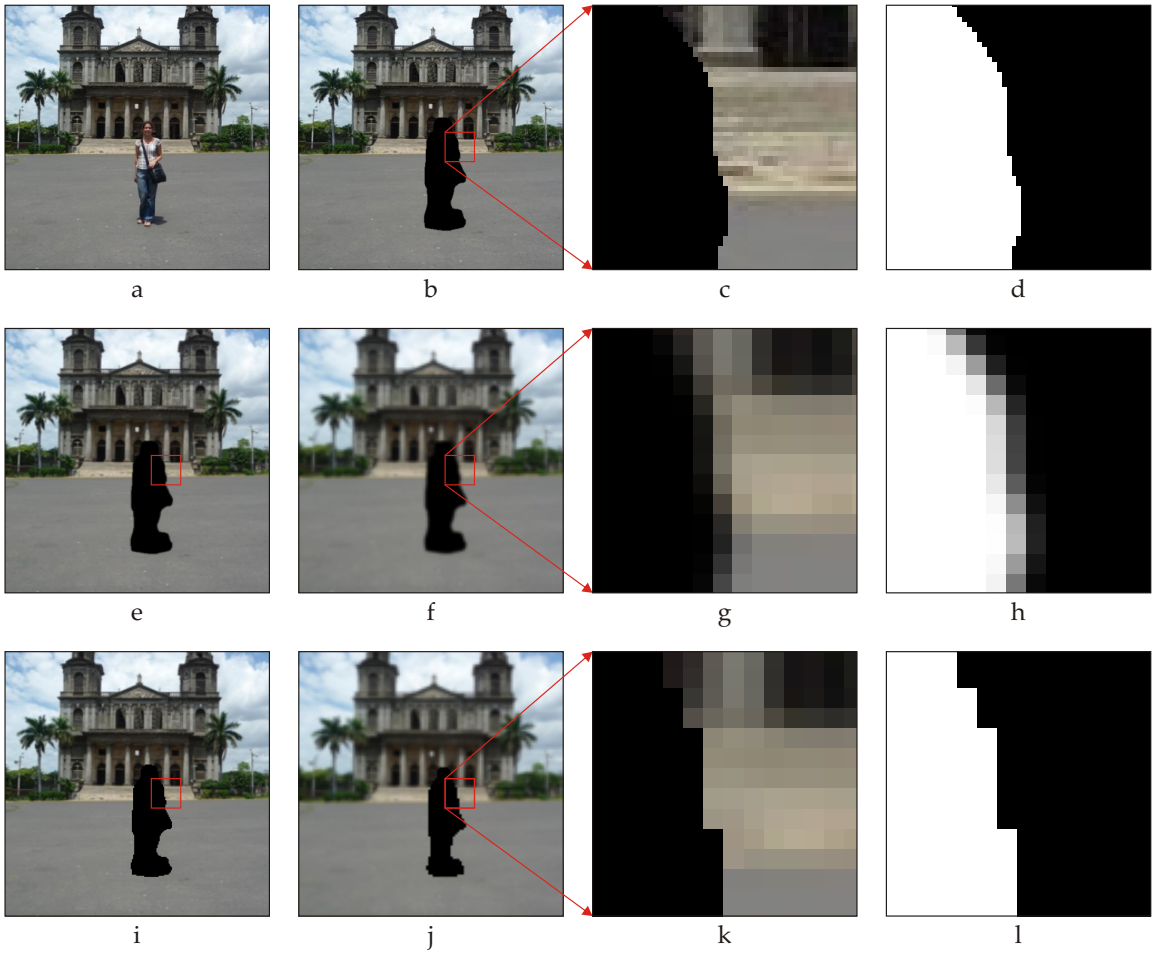


Figure 2.4: The unwanted blur effect in both image and its mask caused by direct application of Gaussian pyramid approach. The original image (a) is assigned zeros (i.e., black color) on pixels corresponding to masked area (b). The cut of this image is shown in (c) and corresponding cut of mask in (d). First (e) and second (f) levels of Gaussian pyramid are created without focusing to preserve the sharp contour of the mask. This is highly unwanted situation, as one can not consider the blurred pixels near the hole as valid, since they are "polluted" by the color assigned to masked area (and which can be of any arbitrary color). To resolve this, one must, when convolving, consider only valid pixels and decide (based on the ratio of known/unknown pixels), whether the synthesized pixel in next level's image is valid or not. This approach is demonstrated on the same data - the first (i) and second (j) levels of the pyramid. Both the zoomed cut of the second level's image (k) and its corresponding mask cut (l) show no blur between the known and unknown area of the image.

the function on the position of the function:

$$\int_{-\infty}^{\infty} f(t)\,\delta(t-T)\,dt = f(T)\,.$$

This is known as the "sifting" property, since it "sifts" out the value of the integrand at the point of its occurrence. To summarize it in other words, the convolution with Dirac δ shifts the image on the position of the Dirac pulse. By applying deconvolution on the shifted image, removing the unshifted image, will get us the position of Dirac δ, which's offset from the original pulse determines the shift we were looking for.

Therefore, the algorithm utilizing the aforementioned observations proceeds as follows (without considering incompleteness of input image for now):

1. Two images are given: a, the input image to be searched in, and b, which is blank except the patch that we search for.
2. Both images are transformed into frequency domain: $\mathbf{A} = \mathcal{F}\{a\}$, $\mathbf{B} = \mathcal{F}\{b\}$.
3. The cross-power spectrum in frequency domain is computed: $\mathbf{R} = \frac{\mathbf{G}_a \mathbf{G}_b^*}{|\mathbf{G}_a \mathbf{G}_b|}$, where $*$ denotes the complex conjugate.
4. The result is transformed back by applying inverse Fourier transform and obtain the normalized cross-correlation: $r = \mathcal{F}^{-1}\{\mathbf{R}\}$.
5. The offset vector is found by determining the location of the peak in r: $(\triangle x, \triangle y) = \arg\max_{x,y} r$.

However, to be able to use this approach on image completion, few problems must be previously solved. First, the queried image a must not contain any missing values. To use another approach to synthesize the missing part and then only try to improve it, would be ineffective and time-consuming. Thus, the holes in input image are filled only with some average value or gradually computed as average value of known neighboring pixels.

The second problem is, that the correlation very likely points to the original (but useless) patch. To overcome this, more than just one maximum must be found and from these candidates, the final pixel value must be chosen by using some patch-to-patch distance metric.

Although not intended for image completion, phase correlation, might prove useful. The advantage over the non-parametric sampling is the fact that time and computational complexity does not rely on size of the patch. Thus with increasing patch size, this approach might show better results on scratch-like holes.

2.6 Sequential overlap exploitation

Recently, Xiao et al . [30] proposed an algorithm that makes use of exploring the sequential overlap between patches neighboring patches. The algorithm is based on following observation: when performing the nearest patch matching (using sum of squared differences as the distance metric) in sequential order using an exhaustive search method, the adjacent corresponding patch-pairs overlap to a considerable extent. By using this sequential overlap, redundant computations may be eliminated and, therefore, the time complexity can be reduced greatly, since the algorithm reduces the time complexity of a patch-pair of size r (thus having r^2 pixels) from $\mathcal{O}(r^2)$ to $\mathcal{O}(r)$.

2.6.1 Single column processing

Let Z and X denote the source and target image, respectively, and S and P the number of patches in each column of Z and X, respectively. Fig. 2.5 illustrates how to find the nearest neighbor patch in the first column in the target X. Each patch slides only by one pixel at the time. X_0 and Z_0 are first compared. Based on the overlap of this result, X_1 and Z_1 are compared and the similarity of adjacent patch pairs is sequentially computed until the end of first iteration, when X_{P-1} and $Z_{(P-1)mod(S)}$ are compared. Then, similarly to the first iteration, X_0 is compared with Z_1 and its subsequent patches are compared with the corresponding subsequent patches in Z until the end of the second iteration. In the final iteration, X_0 is compared with Z_{S-1}, and the subsequent corresponding patch pairs are compared until X_{P-1} finishes the comparison with $Z_{(P+S-2)mod(S)}$. Using this method reduces, as mentioned, the complexity of 2D patch comparison from $\mathcal{O}(r^2)$ to $\mathcal{O}(r)$ for each patch. This speed up becomes quite significant with increasing size of the patch.

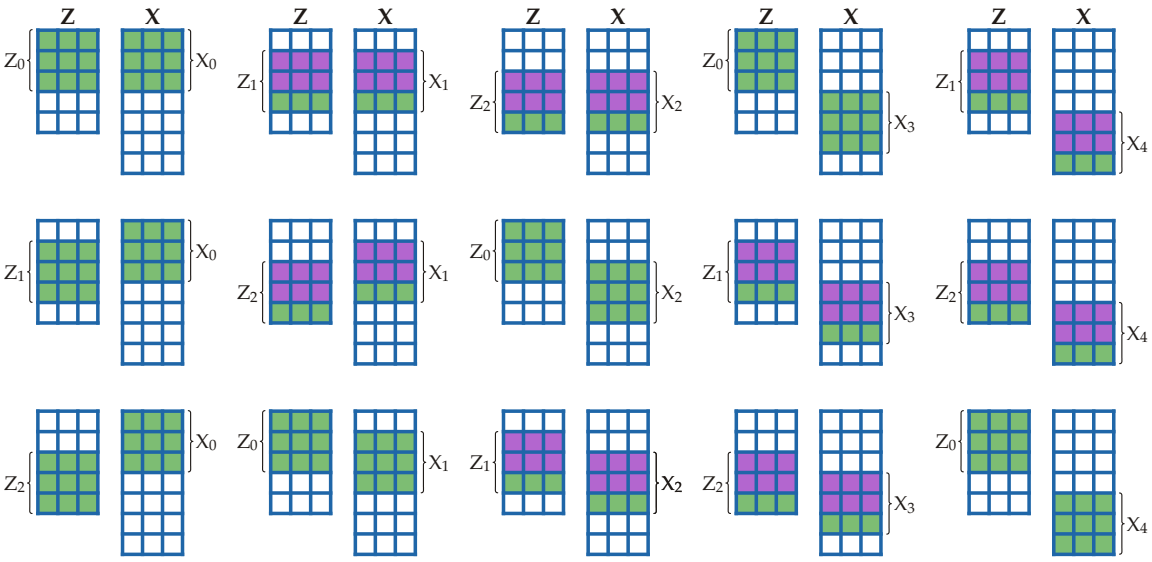


Figure 2.5: Nearest patch matching for target image $X\,(P=5)$, source image $Z\,(S=3)$, and the patch size $r=3$; image taken from [30]. The first row is the first iteration $(L=0)$, the $d\,(X_0, Z_0)$ is first computed, then its consequent patches (X_1, X_2, X_3, X_4) are compared with the corresponding consequent patches (Z_1, Z_2, Z_0, Z_1) in Z. Specially, based on the overlap between X_0 and Z_0, $d\,(X_1, Z_1)$ is computed, similarly, based on the overlap between X_1 and Z_1, $d\,(X_2, Z_2)$ is computed, then we compute $d\,(X_3, Z_0)$, based on the overlap, $d\,(X_4, Z_1)$ is computed. The second row is the second iteration $(L=1)$, the $d\,(X_0, Z_1)$ is first computed, its consequent patches (X_1, X_2, X_3, X_4) are compared with the corresponding consequent patches(Z_2, Z_0, Z_1, Z_2) in Z. Note that the overlaps are used in the distance computation. The third row is the third iteration $(L=2)$, similarly, the $d\,(X_0, Z_2)$ is first computed, its consequent patches (X_1, X_2, X_3, X_4) are compared with the corresponding patches (Z_0, Z_1, Z_2, Z_0) in Z.

2.6.2 Extensions

To further improve the performance of proposed algorithm, the authors presented the extension to single column processing routine. The nearest neighbor is being found for N columns at once by comparing N adjacent columns of patches in source image Z. Also, the presented approach can be extended to higher dimensions, namely to 3D for 3D "patch" matching.

2.6.3 Limitations

As reported in [30], the results of the fast nearest patch search highly depend on the input data. When the data shows no major sequential overlap (like, e.g., when sampling from target region by priority as proposed in [11]), the data can not work at maximum efficiency.

2.7 PatchMatch

PatchMatch, introduced in [4] and extended in [5], represents another approach to search the image space. In contrary to deterministic methods, it is based on random sampling and propagation of good guesses. The idea behind PatchMatch is the approximate nearest neighbor (NN) algorithm. Even though it does not guarantee to find the NN for a given patch, it converges very fast and the found approximate NN almost certainly corresponds to the NN found by the exhaustive search. However, the speed of PatchMatch outperforms the exhaustive search approaches in one or two orders of magnitude. PatchMatch makes great use of natural structures of the images. In real-world images, single pixel is rarely a feature on its own but usually a part of a larger coherent area. Thus, neighboring pixels will likely have their nearest neighbors placed abreast. This approach dramatically reduces the required number of iterations as well as the time of computation.

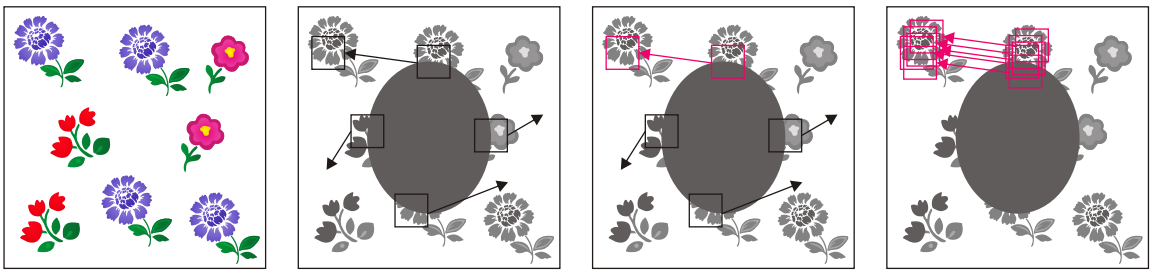


Figure 2.6: The principle of propagating correspondences; image based on video associated with [4]. In the input image (a), the target region is selected (the dark gray oval) (b). Random guesses for the correspondence are likely to be wrong most of the time. However, in sufficiently large region, a few lucky guesses will be almost the correct correspondence (c).Once a good guess for some patch is found, it is likely that many nearby patches have similar correspondences (d).

As the NN-search between image regions is the core issue of many image manipulation algorithms, the PatchMatch has vast ways of use such as, e.g., image retargeting, image reshuffling, texture synthesis or image completion.

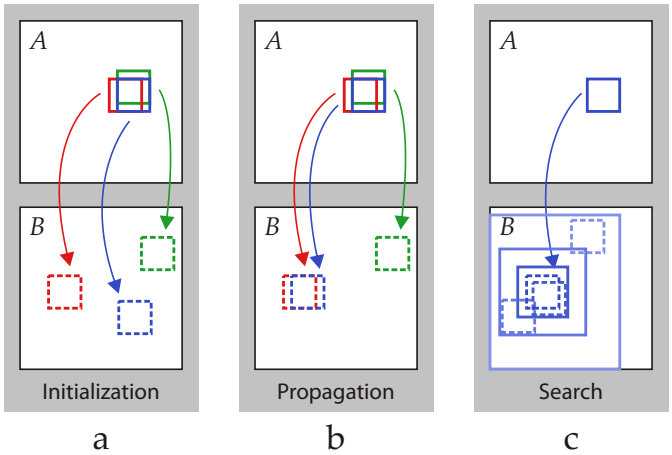


Figure 2.7: Phases of the PatchMatch algorithm; image taken from [4]. (a) The values of $f(\mathbf{x})$ for red, green and blue patches is initialized with random values. (b) The blue patch checks above/green and left/red neighbors to see if they will improve the blue mapping, hence propagating good matches. (c) The patch searches randomly for improvements in concentric neighborhoods.

2.7.1 Approximate nearest-neighbor algorithm

Nearest-neighbor field The core element of the PatchMatch algorithm is the *nearest-neighbor-field* (NNF) defined as a function $f : A \mapsto \mathbb{R}^2$. This function is defined over all patches within the source zone of image A, for some distance function of two patches d. Given a patch coordinate $\mathbf{a}$ in image A and its corresponding nearest neighbor $\mathbf{b}$ in image B, $f(\mathbf{a})$ is defined simply as $\mathbf{b}$ [2] and values of f are stored in an array that has the same dimensions as image A. The algorithm then proceeds in two steps.

Initialization The nearest-neighbor field can be initialized either by assigning the random values to the field, or by using information obtained before. Prior information can be utilized, e.g., when combined with Gaussian pyramid (see sec. 2.4.3) or any other *coarse-to-fine* gradual resizing process.

Iteration After initialization, NNF is iteratively improved. There are two types of iterations that differ only by the direction of processing the image. In odd iterations, offsets are examined in scanline order (from top-left to bottom-right corner), and in even iterations the order is reversed. Every iteration consists of two operations, *propagation* and *random search*, which are interleaved at patch level: if P_i and S_i denote propagation and random search phase, respectively, the algorithm proceeds in following order: $P_1, S_1, P_2, S_2, \ldots, P_n, S_n$.

[2] Using notation in absolute coordinates, as used in [5], in contrary to relative coordinates used in [4].

Propagation When trying to improve the value at $f(\mathbf{x})$ during odd iteration, new candidates are values at $f(\mathbf{x} - \triangle_{\mathbf{p}}) + \triangle_p$, where $\triangle_p$ is a unit vector $(1, 0)$ or $(0, 1)$. The value of $f(\mathbf{x})$ is improved if any of the candidates has smaller distance to patch located at $\mathbf{x}$ with respect to given distance metric d.

As the effect of the propagation phase, the whole coherent image areas can be propagated in a single iteration, thus it converges very quickly. However, if used alone, propagation could end trapped in a local minimum. Therefore, it is followed by random search.

Random search In random search phase, to improve the value at $f(\mathbf{x})$, the sequence of new candidates is sampled from an exponential distribution and $f(\mathbf{x})$ is improved if any of the candidates has smaller distance to $\mathbf{x}$ with respect to given distance metric d. Let $\mathbf{v}_0$ denotes the current nearest neighbor of $\mathbf{x}$, i.e., $f(\mathbf{x}) = \mathbf{v}_0$. The candidates $\mathbf{u}_i$ are then constructed by sampling around $\mathbf{v}_0$ at an exponentially decreasing distance, i.e., $\mathbf{u}_i = \mathbf{v}_0 + \omega\alpha^i\mathbf{R}_i$, where $\mathbf{R}_i$ is a uniform random in $[-1, 1] \times [-1, 1]$, ω is the maximum search "radius" (usually defined as maximum image dimension), and α is a fixed ratio between search window sizes (usually $\alpha = \frac{1}{2}$). Patches for $i = 0, 1, 2, \ldots$ until the current search radius $\omega\alpha^i$ is below 1 pixel. The search window must be clamped to the bounds of image B before it can be sampled from.

Halting criteria As reported in [4], fixed number of iterations had been found to work well. The results showed that after approximately 4 to 5 iterations, the NNF had almost always converged.

Generalization Even though the original algorithm works well, in [5] the authors presented a generalized form of PatchMatch to extend its functionality in three ways:

1. Finding k nearest neighbors instead of just one.
2. Arbitrary descriptors and/or distance metrics are allowed to be used in opposite to using only sum of squared differences on colors in [4].
3. Searching across rotations and scales, not only the translations.

Finding k nearest neighbors can be used for tasks such as denoising, symmetry detection or object/clone detection. However, for image synthesis, finding only the best matching patch is sufficient and practice shows that extending the PatchMatch in this way has almost no influence on the result image for this type of application. Also point (3) is not in particular concern of this thesis. However, extending the search space on more patch transformations might improve the result image.

Rotations and scale To search a range of rotations $\theta \in [\theta_1, \theta_2]$ and a range of scales $s \in [s_1, s_2]$, the search space of the original PatchMatch algorithm must be extended from (x, y) to (x, y, θ, s), extending the definition of the NNF to a mapping $f : \mathbb{R}^2 \mapsto \mathbb{R}^4$. Assignment f is initialized by uniformly sampling from a range of possible positions, orientations and scales. In the propagation phase, adjacent patches are no longer related

by a simple translation, thus the relative offsets must by transformed. Let $T(f(\mathbf{x}))$ be the full transformation defined by (x, y, θ, s). The candidates are then defined as $f(\mathbf{x} - \triangle_{\mathbf{p}}) + T^{'}(f(\mathbf{x} - \triangle_{\mathbf{p}}))\triangle_p$. In random search phase, the search window is extended to all 4 dimensions. As documented in [5], in spite of searching over 4 dimensions instead of just one, the combination of propagation and random search successfully samples the search space and efficiently propagates good matches between patches.

Chapter 3

Selected algorithms

In this section, more detailed description of theoretical base of selected algorithms will be given.

3.1 Method of Efros and Leung

Efros and Leung [13] proposed an algorithm for texture generation based on model of the texture as a Markov Random Field (MRF). To be more specific, they assume that the probability distribution of brightness (or color) values for a pixel given the brightness values of its spatial neighborhood is independent of the rest of the image. The only adjustable parameter to control the degree of the randomness is the size of the pixel's neighborhood, i.e., the *patch size*.

The missing part of the image is generated in scanline order pixel by pixel. When pixel p is chosen to be the next one to synthesize, all known pixels in the patch around pixel p are take as context. No probability distribution table or model is being constructed and instead, for each new context the image (or more specifically, the source region of the image) is queried and the distribution of p is constructed as a histogram of all possible values that occurred in the image.

3.1.1 Algorithm

As stated previously, the texture (image) is modeled as MRF, thus the probability distribution of color values for a pixel given the color values of its spatial neighborhood is assumed to be independent of the rest of the image. The neighborhood of the pixel is represented by the patch around it. The size of this patch is recommended to be on the scale of the biggest regular image feature.

We will explain, how the algorithm works but first, let me declare some definitions. Let p be the aforementioned pixel, $\omega(p)$ its patch neighborhood and $d(\omega_1, \omega_2)$ some perceptual distance metric between two patches. Moreover, suppose that the image, from which the samples are taken, I_{sample} is a part of the real infinite texture I_{real}. Then if a set of all occurrences of $\omega(p)$ within I_{real} was known,

$$\Omega(p) = \{\omega' \subset I_{real} : d(\omega', \omega(p)) = 0\},$$

then the conditional probability distribution function of p, $P(p|\omega(p))$, could be estimated with a histogram of all center pixels' color values in it.

Since only the I_{sample} is known, the approximate candidate set $\Omega'(p) \approx \Omega(p)$ must be found. The authors propose k-nearest neighbor technique when first the closest sample ω_{best} from I_{sample} is found as $\omega_{best} = \arg\min_{\omega} d(\omega, \omega(p)) \subset I_{sample}$ and then set Ω' of all patches ω', that satisfy the patch-to-patch distance condition $d(\omega_{best}, \omega') < \epsilon$, where ϵ is a threshold value, is obtained. Values of center pixels of patches in Ω' give the histogram of values for pixel p, from which the values can be sample either uniformly or weighted by the distance metric d.

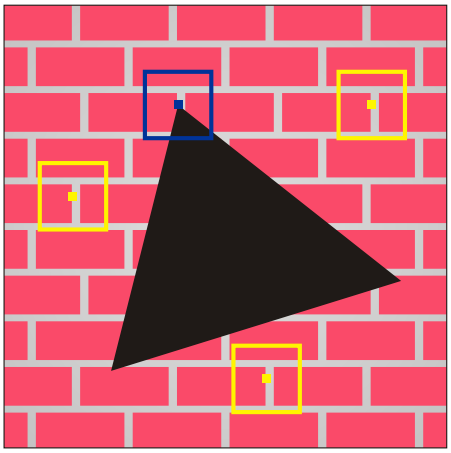


Figure 3.1: Algorithm overview: to synthesize color of the examined pixel (blue), the rest of the image is scanned and then, the center pixel value of one the best matching sample patches (yellow) is used to be the newly synthesized pixel.

The proposed distance metric to be used to sample is sum of squared differences (SSD) convolved with two-dimensional Gaussian kernel to emphasize the local structure of the texture, i.e., $d = d_{SSD} * G$.

When using this approach to synthesize missing image parts, for each processed pixel, not all of its neighboring pixels are known. To overcome this, distance metric must be slightly modified. One of the methods mentioned in sec. 2.1.1 is used.

3.1.2 Summary

Despite its simplicity, this algorithm proved to produce quite plausible results. Its main disadvantage, however resides in the need of exhaustive search. Scanning for best patch correspondences over the entire image is simply quite time-consuming process even when early termination is incorporated. The other considerable drawback is that it does not handle the propagation of linear structures in any way and the results may vary by the selected sampling method. This is also the key observation and main contribution of the work of Criminisi et. al [11].

3.2 Method of Averbuch et al.

The slowest part of many exemplar-base image completion algorithms is the exhaustive search over the whole image again and again. To speed this process up, some search structure could be used and this is also the contribution of presented paper.

3.2.1 Algorithm

The algorithm proceeds in two main steps: First, the image is scanned and the information about pixel color values is inserted into the search data structure. In the latter phase, for each pixel marked as unknown, the search data structure is queried and the best corresponding pixel (or set of pixel candidates) is obtained.

Learning phase In this phase, the search data structure is created. The patch $\omega(p)$ of size α around each pixel p is represented by a vector $v(p)$ of integer numbers in the range from 0 to 255. Thus this vector contains α^2 entries for grayscale image and $3\alpha^2$ entries for color image. The authors advise to use the YUV color space. The pixel values are taken in row-major order in a consecutive manner.

Synthesis phase In this step, the search data structure is repeatedly queried and the image hole is gradually filled. The missing area is traversed in a spiral order (also called onion-peel order) and new pixel values for unknown pixels are assigned. The patch around each processed pixel $\widetilde{p}$ is again modeled as a vector $v(\widetilde{p})$. However, one or more pixels around processed pixel (at least the the point being processed itself) is unknown. Therefore, another vector $m(\widetilde{p})$ is used as a mask for vector v. This mask vector contains ones on places of valid pixels and zeros on places of unknown pixels.

After the query vector is compared to the vectors stored in the search data structure, on or more best candidates are retrieved. The final pixel value to be assigned to unknown pixel $\widetilde{p}$ is then selected among them.

3.2.2 Distance metric

The distance metric the authors used in the search data structure is L_∞. The distance between two vectors v and u of size α^2 is given by $\|v-u\|_\infty = max_{1\leq i\leq \alpha^2}|v(i)-u(i)|$ where $v(i)$ and $u(i)$ denote the ith coordinate of v and u, respectively.

3.2.3 Search structure

For clarity, the principle of search data structure will be shown on the case of monochromatic image. The size α^2 of vector v will be denoted as d from now on.

Let $V = \{v_i\}_{i=1}^K \subseteq [0..\beta]^d \subseteq \mathbb{N}^d$ be a set of K vectors to be inserted into the data structure, where d is the dimension of the vectors and i is the index of the vector in the data structure. The jth element of vector v_i will be denoted $v_i(j)$.

The search data structure consists of d arrays $\{A_i\}_{i=1}^d$ of $\beta+1$ elements, where $A_i(j)$ denotes the jth element of the array A_i. Each of this elements contains a set of vector indices $I \subseteq \{1,\ldots,K\}$.

Insertion into the structure In the learning phase, the vectors of pixels from allowed zone (i.e., those, whose patch neighborhood is fully known) are inserted into the data structure. When inserting vector v_l, its index l is inserted in the sets in $A_i(v_l(i))$ for every $1 \leq i \leq d$. Following example demonstrates the insertion process.

Let the set V contain vectors $v_1 = (2, 0, 1, 3)$, $v_2 = (3, 0, 4, 4,)$ and $v_3 = (3, 0, 1, 3)$. Clearly $K = 3$, $d = 4$ and $\beta = 4$. Table 3.1 shows the data structure after insertion of vectors from set V.

	A_1	A_2	A_3	A_4
0		1,2,3		
1			1,3	
2	1			
3	2,3			1,3
4			2	2

Table 3.1: Search data structure constructed for parameters $K = 3$, $d = 3$ and $\beta = 4$ after insertions of vectors vectors $v_1 = (2, 0, 1, 3)$, $v_2 = (3, 0, 4, 4,)$ and $v_3 = (3, 0, 1, 3)$.

When a vector is added into the data structure, its number i is inserted element once in each search array A. For example, since $v_1(1) = 2$, we insert the number 1 to the set $A_1(2)$. Next element of $v_1(2) = 0$, thus $A_2(0) = 2$ and so on.

Querying the data structure Queries to the data structure return a set of candidate vectors and are parameterizable by parameters E and C, where $0 \leq E \leq \beta$ is the upper bound of maximal error distance between the vectors in result set and the query vector, and C is the number of candidate vectors that shall be found. This means that every vector in the result set $R = \{r_i\}_{i=1}^{C} \subseteq V$ must satisfy $\|r_i - q\|_\infty \leq E$ for the given query vector q. The run of one query is depicted in algorithm 2.

	A_1	A_2	A_3	A_4
0		**1,2,3**		
1			**1,3**	
2	**1**			
3	2,3			**1,3**
4			2	2

a

	A_1	A_2	A_3	A_4
0		**1,2,3**	**Ø**	
1		**Ø**	**1,3**	
2	**1**		**Ø**	**Ø**
3	**2,3**			**1,3**
4	**Ø**		2	**2**

b

	A_1	A_2	A_3	A_4
0		**1,2,3**		
1			1,3	
2	1			
3	**2,3**			1,3
4			2	**2**

c

Table 3.2: Query examples on the data structure shown in tab. 3.1. Visited array elements are highlighted in bold typeface. (a) The data structure is queried for an exact match ($E = 0$) for the vector $q_1 = (2, 0, 1, 3)$. Query mask $m_1 = (1, 1, 1,)$ indicates that the query vector contains no missing elements. The only matching vector stored in data structure is v_1. (b) An approximate match ($E = 1$) for query vector $q_2 = (2, 0, 1, 3)$; $m_2 = (1, 1, 1,)$. Vectors v_1 and v_3 are within the L_∞ distance 1 from query vector q_2. (c) Querying the data structure for an exact match for partially incomplete vector $q_3 = (3, 0, ?, 4)$; $m = (1, 1, 0, 1)$. Only vector v_2 matches the given query.

Algorithm 2 The query algorithm to the search data structure as presented in [3].

Query({A_i},q,m,E,C)

1: $R \leftarrow \phi$, $N \leftarrow$Number of zero elements in q
2: **for** $e = 0 \rightarrow E$ **do**
3: **for** $i = 1 \rightarrow d$ **do**
4: **if** $m(i) \neq 0$ **then**
5: $R \leftarrow R \cup A_i(q(i) - e) \cup A_i(q(i) + e)$
6: **end if**
7: **end for**
8: **if** there are C elements in R that each appear at least $d - N$ times **then**
9: **return** R
10: **end if**
11: **end for**
12: **if** $e \geq E$ **then**
13: **return** all elements that appear $d - N$ times and indicate that $|R| < C$
14: **end if**

3.2.4 Summary

The proposed data structure can speed up the synthesis process, however the major drawback of used distance metric is that for some images, that hardly contain good guesses for missing pixels, no solution within the given error bound E might be found. Such situations can be handled by either extending the error bound and query the data structure until some results are obtained, or by selecting C best candidates and test them using another distance metric.

Also the memory-complexity of the search data structure grows rapidly with increasing image and patch sizes which further increases the time needed to perform the union operation (see alg. 2, line 5). This is the subject to study and the result will be presented in further sections.

3.3 Method of Criminisi et. al

By comparing and studying large amount of image-completion-related work, Criminisi et. al proposed a new algorithm that combined the advantages of both exemplar-based and inpainting-based methods. The contribution of inpainting-driven is the ability to encourage the propagation of linear structures prior to the textures. In inpainting, linear structures (called also isophotes) are propagated via diffusion and the process resembles the heat flow spreading, which is actually true due to use of partial differential equations. This works well for smaller holes but for larger ones, blurry artifacts.

The exemplar-based techniques, on the other hand, do not suffer from these problems, since they cheaply copy existing parts of the image from one area to another. On consistent textures, these methods work quite effectively. However, real-world scenes often consist of mixture of linear structures and multiple textures interacting spatially [32].

As will be explained, the described algorithm solves the problem of texture by priority sampling, i. e. the linear structures are given greater priority and they are propagated

sooner.

3.3.1 Key observations

The presented algorithm is based on two observations.

Exemplar-based synthesis suffices - both structure and structure can be propagated via exemplar-based approaches, i.e., there is no need for separate mechanism to handle the isophotes.

The filling order is crucial - as observed, the quality of the output image is highly dependent on the order in which the filling process proceeds. As shown in fig. 3.2, the filling in spiral order (onion peel) may introduce undesired artifacts and the horizontal edge between two regions might be reconstructed as a curve in a case of filling concave regions. The very similar situation may occur when scanline order is used.

Thus an ideal algorithm must give higher priority to those areas of the image that lie on the continuation of image's linear structures. The goal is to find the balance, i.e., when to propagate the structure and when the texture.

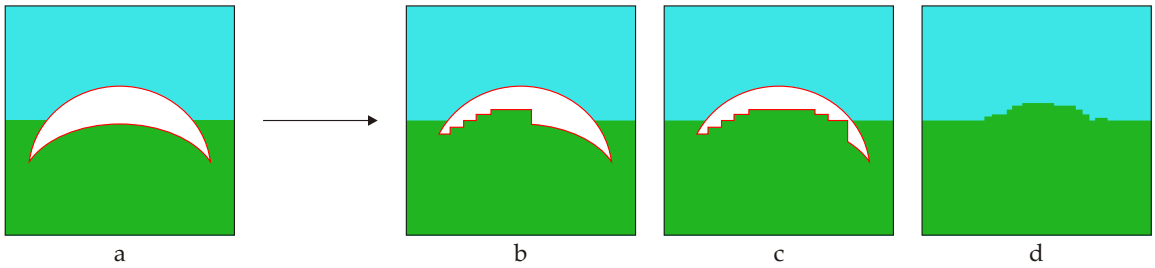


Figure 3.2: Problems of concentric-layer filling order (anti-clockwise); image taken from [11]. (a) Image with selected target region. Exemplar patches are taken from the rest of the image. (b,c) Different stages during the filling process. (d) Result image - the horizontal line from the original image was reconstructed as a curved structure.

3.3.2 Algorithm

The notation is as follows: Given an input image, target region Ω is selected. The source region Φ may be either specifically selected or may be taken as the rest of the image $\Phi = \mathcal{I} - \Omega$ (note that a band of at least size of *patch radius* pixels must be stripped from source zone to consider only patches entirely contained in Φ; for illustration see fig. 4.1).

The size of the patch window Ψ is then specified. Results presented by authors were obtained using mostly patches of size 9×9 pixels. Fig. 3.3a illustrates the used notation. After determining the size of patch window, the algorithm proceeds automatically, i.e., no user guidance is needed. Pixel on the contour line of the target region (called also *fill front*) with highest priority is selected and best candidate to fill its unknown pixels is found. This process repeats until the image is complete.

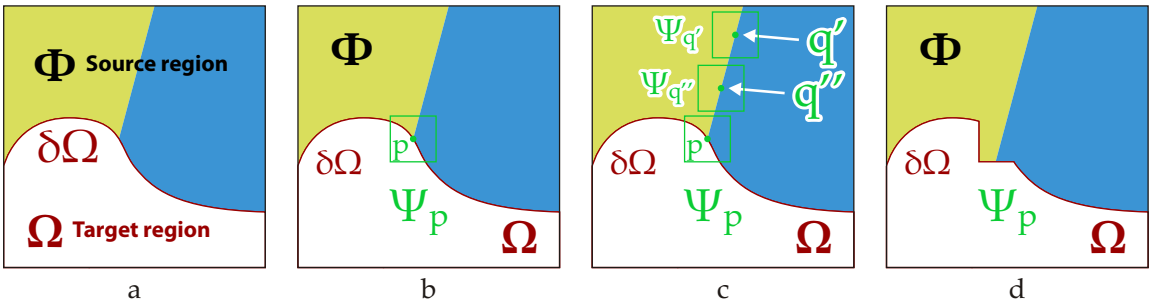


Figure 3.3: Structure propagation along isophote; image taken from [11]. (a) The input image with labeled target region Ω, its contour $\delta\Omega$ and the source region Φ. (b) Patch Ψ_p around pixel $p \in \delta\Omega$ is selected to be synthesized in the next iteration. Selection of this contour point is based on its priority. (c) The best matching candidates lie along the boundary between the yellow and blue texture, e.g., $\Psi_{q'}$ or $\Psi_{q''}$. (d) Best matching candidate is selected and pixels on positions corresponding to unknown pixels within patch Ψ_p are copied. Both texture and structure have been propagated.

Each pixel has assigned a property called *confidence*. This non-zero number, as the name suggests, indicates how much we are sure about the value the pixel has. This value is initialized when the algorithm starts and when the pixel is filled (i.e., only pixels from target region change their confidence value during the course of the algorithm).

Selecting patch with highest priority In each iteration the algorithm performs selection of a patch-to-be-processed based solely on its priority. High-priority pixels are:

- pixels on the continuation of linear structures (edges between textures)
- pixels that are surrounded by other pixels with high confidence

Priority is computed for all patches centered on pixels on current $\delta\Omega$. The priority $P(p)$ of the patch Ψ_p centered at some point $p \in \delta\Omega$ is defined as follows:

$$P(p) = C(p)D(p),$$

where $C(p)$ is the *confidence term* and $D(p)$ is the *data term* and they are defined as follows:

$$C(p) = \frac{\sum_{q \in \Psi_p \cap (\mathcal{I}-\Omega)} C(q)}{|\Psi_p|}$$

$$D(p) = \frac{|\nabla I_p^{\perp} \cdot n_p|}{\alpha}.$$

The meaning of used notation is this: α is a normalization factor (e.g. $\alpha = 255$ for grayscale image, but can be used even for color ones), $|\Psi_p|$ is the area of patch Ψ_p, n_p is a unit vector orthogonal to the fill front $\delta\Omega$ at point p, ∇I_p is the gradient (computed as the maximum value of gradient in $\Psi_p \cap (\mathcal{I} - \Omega)$) and $\perp$ denotes the orthogonal operator (i.e., the isophote is simple the gradient rotated by 90 degrees). The notation is shown in fig. 3.4.

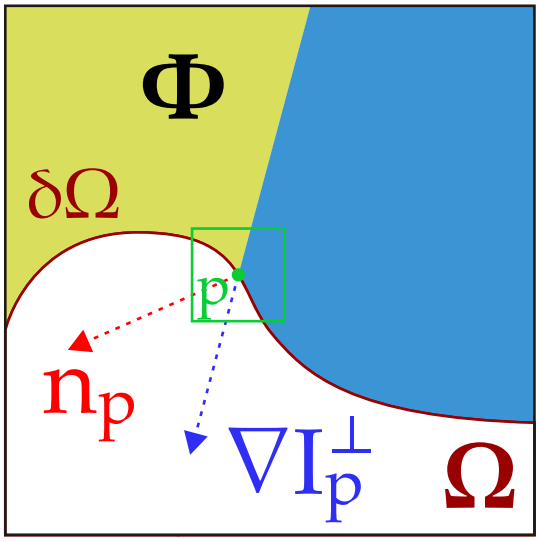


Figure 3.4: Relationship of patch and its normal and isophote; image taken from [11]. For patch Ψ_p, n_p denotes the unit normal vector to the contour line $\delta\Omega$ and $\nabla I_p^{\perp}$ the isophote.

As presented, the priority $P(p)$ is computed for every patch that have its center pixel placed on the contour line of the target region $\delta\Omega$.

Before the iterative process, the confidence must be set. $C(p) = 0\ \forall p \in \Omega$ and $C(p) = 1\ \forall p \in \mathcal{I} - \Omega$.

Confidence term The confidence term, denoted $C(p)$, provides information about the surrounding pixels of pixel p. It gives higher priority to those patches that have already filled more of their pixels. As the synthesis progresses, the confidence term values begin to degrade, as less and less information from the source region is propagated to the pixels nearer to the center of target region. Roughly speaking, the confidence term favors the patches with more pixels already filled in, since these patches provide more valid information to match against. Using only confidence term to determine the priority of the pixel $P(p)$, the fill order would roughly resemble the concentric onion-peel order.

Data term The data term reflects the strength of isophotes that cross the contour line $\delta\Omega$ in given iteration. It gives higher priority to patches that contain "stronger" isophotes (since isophote is not a unit vector like the normal vector n_p). In fact, it is the data term that takes care the linear structures are propagated first.

Finding best exemplar After the priorities of all patches centered along the fill front $\delta\Omega$ have been computed, the patch $\Psi_{\hat{p}}$ with highest priority is selected to be filled in current iteration. As in [13], the source region is then scanned for patch that is most similar to $\Psi_{\hat{p}}$, i.e., $\Psi_{\hat{q}} = \arg\min_{\Psi_q \in \Phi} d(\Psi_{\hat{p}}, \Psi_q)$. The used distance metric is sum of squared differences and the color comparison is done in CIE Lab color space. Pixels from the best candidate Ψ_q are than copied on corresponding positions into patch $\Psi_{\hat{p}}$ and the algorithm proceeds to the next step which is the update of confidence values of pixels within processed patch.

Updating confidence values After the best exemplar patch Ψ_q has been found, confidence term of pixels within the patch $\Psi_{\hat{p}}$ are updated as follows:

$$C(p) = C(\hat{p})\, \forall p \in \Psi_{\hat{p}} \cap \Omega.$$

Algorithm 3 Pseudocode of exemplar-based image inpainting algorithm as proposed in [11]. The superscript t denotes current iteration.

```
Extract the manually selected initial front δΩ^0
repeat
    Identify the fill front δΩ^t
    if δΩ^t = ∅ then
        return
    end if
    Compute priorities P(p) ∀p ∈ δΩ^t
    Find the patch Ψ_p̂ with the maximum priority
    Find the exemplar Ψ_q̂ that minimizes d(Ψ_p̂, Ψ_q)
    Copy image data from Ψ_q̂ to Ψ_p̂ ∀p ∈ Ψ_p̂ ∩ Ω
    Update C(p) ∀p ∈ Ψ_p̂ ∩ Ω
until done
```

3.3.3 Summary

A new approach for exemplar-based-driven algorithms has been proposed in [11]. As observed and demonstrated, the fill order is critical to successfully propagate linear structures. The proposed algorithm provides a way to control it by introducing confidence and data terms that both control the priority by which the next patch to be processed is chosen.

The major drawback of this algorithm is the use of exhaustive search to determine the best exemplar $\Psi_{\hat{q}}$.

3.4 Method of Kwok et al.

Since the method of Criminisi et al. [11] proved itself quite useful, attempts to further improve it had been made. The bottleneck of mentioned approach is clearly the exhaustive search over the whole image space. Therefore, a fast query to find the best matching exemplar is essential to speed the scheme of [11] up. Kwok et al. [25] developed such a method based on application of Discrete Cosine Transformation (DCT) and the reduction of compared coefficients.

3.4.1 Theoretical background

Trying to improve the exhaustive search, the authors first try to employ the *Approximate Nearest Neighbor* (ANN) library [26] which is implemented by KD-tree. However, the dimension of such a tree can not be fixed, since the ratio of known and unknown pixels within the to-be-completed patches is not fixed as well. Therefore, the initial attempt was to fill these missing pixels with average color so that the dimension of KD-tree could be fixed. Surprisingly, same results were obtained with similar lengths of time. Further study showed that when extending the dimension of KD-tree from 20 to more than 200 (e.g. a trichromatic patch of size 9 is a point in the space with a dimension of $9 \times 9 \times 3 = 243$), the performance of KD-tree drops significantly. Thus, to improve the speed end efficiency, the method to approximate the patch with fewer coefficients than all its pixel colors is needed.

To reduce the dimensionality of the search query, different methods are taken into account in [25], namely: Principal Component Analysis (PCA), Fast Fourier Transform (FFT), Discrete Cosine Transformation (DCT) and standard and non-standard Haar wavelet decomposition. The target is to keep 10% or fewer coefficients of the whole patch, while keeping the accuracy of the backward transformation (with respect to the given input) as high as possible (around 90%).

Therefore, PCA is discarded due to the larger fraction of coefficients needed to keep the desired accuracy. FFT is not considered as well because the imaginary part is not useful in image querying. Thus the remaining transformations - two-dimensional type-II DCT and standard and non-standard Haar wavelet decomposition - are taken as candidates. Since they are all linear and their basic functions are orthonormal, the distance can between two image patches can be measured by the difference in their transformed coefficients (see the following lemma).

Lemma If the basic functions of transformation are orthonormal, squared L^2 error of the transformed coefficient differences between two image blocks Ψ_P and Ψ_q are the same as the squared L^2-norm difference between two images on pixel values.

Selection of coefficients to be kept Two approaches to which coefficients of the transformed image patch keep and which to discard is discussed (as shown in fig. 3.5):

1. First, keeping m coefficients at the top-left corner of the transformed image patch is proposed. The idea behind this is based on the reason that human eyes are more sensitive to noises in low frequency components than in high frequency ones [20]. The remaining coefficients are truncated to zero. However, as shown in fig. 3.5b,c,d, the image patches reconstructed from these kept coefficients using Inverse DCT and the standard and non-standard Haar wavelet reconstruction are dramatically different from the input, especially on highly textured images.

2. Second suggested method is to keep m significant coefficients (i.e., the m coefficients with largest magnitude). Using this approach, the result of the backward reconstruction from the m retained coefficients gives clearly better results, as shown in fig. 3.5b',c',d', with DCT being the best candidate as a patch transformation routine (fig. 3.5b').

Gradient-based filling Before the DCT can be applied to image query patches, all unknown pixels must be assigned new value. As observed in [25], using average color value is not a good option. For a smooth image, the gradient at pixels will be approximately equal to zero. Thus, a gradient-based filling method is suggested to determine the values of unknown pixels before the computation of its DCT.

Each unknown pixel $p_{i,j}$ within the image patch is bound to its left/right and top/bottom neighbors using the forward or backward difference, respectively. Therefore, for l unknown pixels, at least k linear equations is given, with $k > l$, which is actually an overdetermined system. The optimal values of unknown pixels are then compute by using the least squares

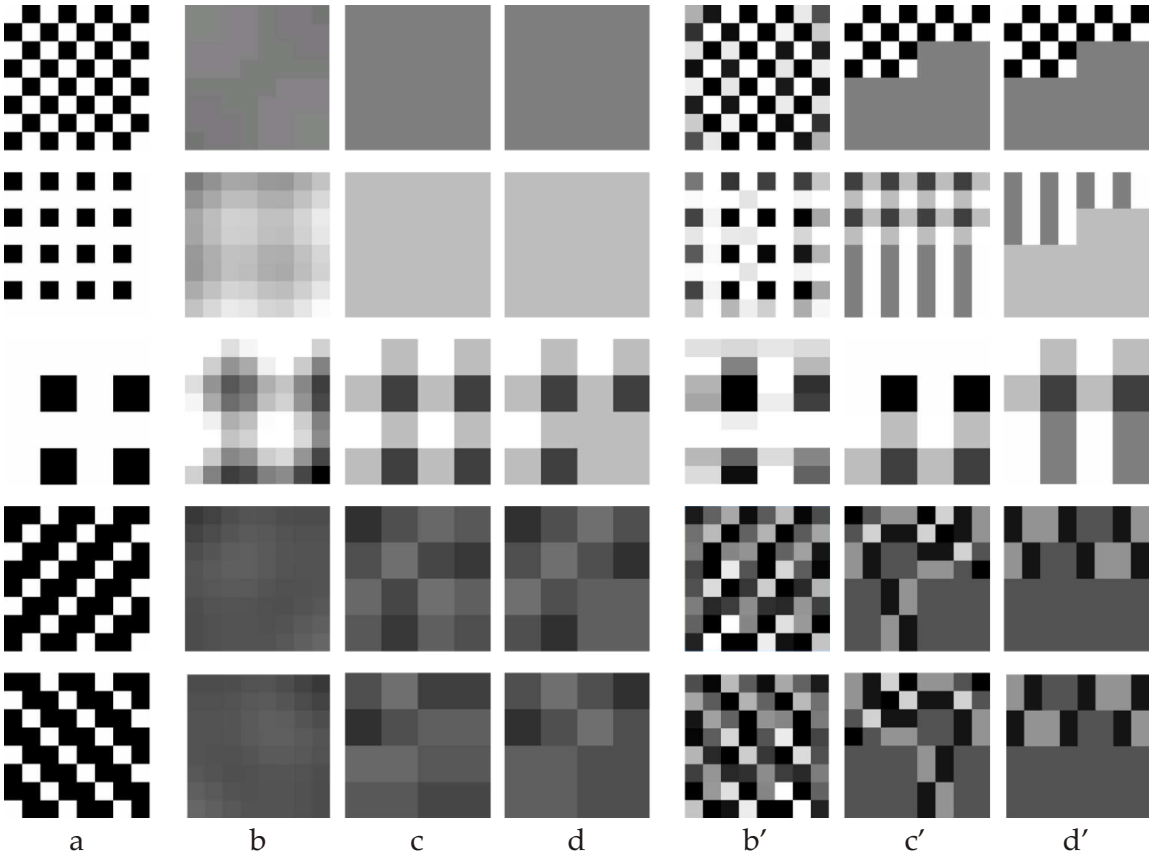


Figure 3.5: Comparison of different transformation methods on differently chosen m coefficients on highly textured image samples; image taken from [25]. The better the original image is reconstructed, the more accurate the image difference error is reflected by these m coefficients. (a) The input patch. (b,c,d) The performance of DCT, standard and non-standard Haar wavelet transformation, respectively, using m coefficients at top-left corner. (b',c',d') The performance of DCT, standard and non-standard Haar wavelet transformation, respectively, using m significant coefficients. One can clearly confirm that using m significant coefficients of DCT (b') gives the best results.

method. Then, the nearest neighbor is then found using the SSD distance metric on truncated m-dominant DCT coefficients.

However, as a drawback reported by authors, the gradient filling may generate smoother images at unknown pixels while the known patch area is highly textured. This complication is overcome by selecting more than just one best matching candidate using the proposed method. Instead, best 0.1% of total exemplars is selected and the final best exemplar is selected among them using the SSD distance metric on known pixels.

3.4.2 Search structure

As suggested in previous sections, before the inpainting process can start, the source patches (i.e., the patches with all pixels known) must be transformed and inserted into a search structure. This search structure must provide fast way to compare stored patches with query patches (i.e., the patches for which the best matching exemplar shall be found). This search structure must take care of storing and accessing those m retained coefficients of each source patch.

Since the positions of m significant coefficients is not fixed, no structure based on KD-trees can be a good option (it would still need to have as many dimension as if no coefficients were truncated). Instead, a structure similar to one proposed in [3] is used. Let $\hat{p}^t_{i,j}$ denotes the DCT coefficient at (i,j) in the to-be-filled patch $\Psi_{\hat{p}}$, where t denotes the color channel of CIE Lab space (i.e., $t = L, a, b$). Best exemplar$\Psi_{\hat{q}}$ minimizing the Euclidean distance from the query patch $\Psi_{\hat{p}}$ is then given as:

$$\tilde{d}\left(\Psi_{\hat{p}}, \Psi_{\hat{q}}\right) = \sum_t \sum_{(i,j)} \left(\hat{p}^t_{i,j}, \hat{q}^t_{i,j}\right)^2 \ .$$

By expanding and eliminating the zero terms, the previous equation can be formulated as:

$$\tilde{d}\left(\Psi_{\hat{p}}, \Psi_{\hat{q}}\right) = \tilde{d}_{\hat{p}} + \tilde{d}_{\hat{q}} - 2\tilde{d}_{\hat{p}\hat{q}} \ , \text{ where}$$

$$\tilde{d}_{\hat{p}} = \sum_t \sum_{\hat{p}^t_{i,j} \neq 0} \left(\hat{p}^t_{i,j}\right)^2 ,$$

$$\tilde{d}_{\hat{q}} = \sum_t \sum_{\hat{q}^t_{i,j} \neq 0} \left(\hat{q}^t_{i,j}\right)^2 \text{ and}$$

$$\tilde{d}_{\hat{p}\hat{q}} = \sum_t \sum_{\hat{p}^t_{i,j} \neq 0, \hat{q}^t_{i,j} \neq 0} \hat{p}^t_{i,j} \hat{q}^t_{i,j}.$$

Furthermore, the value of $\tilde{d}_{\hat{p}}$ for a given query patch does not change during the computation and can be omitted, so the scoring metric for comparing similarity between patches $\Psi_{\hat{p}}$ and $\Psi_{\hat{q}}$ can be further simplified to

$$S\left(\Psi_{\hat{p}}, \Psi_{\hat{q}}\right) = \tilde{d}_{\hat{q}} - \tilde{d}_{\hat{p}\hat{q}}.$$

The smaller the value of S for given patch pair is, the more they are similar one to each other.

Given a query patch of size $n \times n$ pixels with m significant DCT coefficients, to evaluate $S\left(\Psi_{\hat{p}}, \Psi_{\hat{q}}\right)$ only $2m + 1$ multiplications plus one subtractions for each color channel are needed in the worst case scenario, while up to n^2 multiplications plus n^2 subtractions are needed to obtain the value of SSD on all corresponding pixels of given patch pair. In fact, the aforementioned count of operations $(2m+1)$ is only performed, when the nonzero coefficients of both patches $\Psi_{\hat{p}}$ and $\Psi_{\hat{q}}$are placed on the same patch coordinates. In general, however, there are fewer overlapped coefficients. Besides, the values of $\tilde{d}_{\hat{q}}$ can be precomputed and so only $\tilde{d}_{\hat{p}\hat{q}}$ must be evaluated on the fly.

While computing $\tilde{d}_{\hat{p}\hat{q}}$, one first has to check if neither $\hat{p}^t_{i,j}$ nor $\hat{q}^t_{i,j}$ are zero. To do this, the authors propose a search array data structure similar to ones used in [22] or [3]. For each color channel t, separate search array D^t is constructed. Each point of this array, $D^t[i,j]$ then contains a list pointing to all source patches that have the DCT coefficient at (i,j) in color channel t nonzero. More specifically, the record stored in $D^t[i,j][k]$ (with k representing the k-th source patch having nonzero DCT coefficient at (i,j)) contains two values:

- $D^t[i,j][k].ID$ - the global index of associated source patch
- $D^t[i,j][k].\alpha$ - the actual value of nonzero DCT coefficient

3.4.3 Algorithm

Search arrays for all channels are constructed before the actual image synthesis starts - in the initialization phase. All source patches from the source region Φ (see notation in section 3.3.2) are decomposed into the frequency domain by using the DCT. Obtained DCT coefficients are truncated with only m most significant coefficients left. These coefficients are then inserted into the search arrays and during the scoring process (the selection of best exemplar for given query patch) serve as the values of $\tilde{d}_{\hat{p}\hat{q}}$ term. Also, during the initialization step, another array, denoted as B (meaning *base-score array*), is created to store the precomputed values of the $\tilde{d}_{\hat{q}}$ term. The element $B[k]$ stores the value of $\tilde{d}_{\hat{q}}$ of the k-th source patch. Both arrays B and D are then used to score the stored exemplars and to choose the best matching one. More detailed description of the algorithm is given in alg. *FindBestMatch* 4.

Algorithm 4 Pseudocode of selection of the best matching exemplar as proposed in [25].

FindBestMatch

```
scores[k] ← B[k]
for each color channel t do
    for each nonzero DCT coefficient p̂^t_{i*,j*} do
        for each element e of D^t[i*, j*] do
            K ← D^t[i*, j*][e].ID
            scores[k] ← scores[k] − 2 (D^t[i*, j*][e].α · p̂^t_{i*,j*})
        end for
    end for
end for
k_min ← arg min_k scores[k]
```

3.4.5 Summary

The approach presented in [25] improves the bottleneck of [11] - the exhaustive search of best matching exemplars for given queries. This is done by transforming the patches into the frequency domain (using DCT) and truncating larger fraction of resulting coefficients, thus reducing the search space. A search array-like structure is presented to efficiently score the examined stored exemplars during the synthesize phase. By this approach, the algorithm significantly reduces the time complexity of image completion of given image.

3.5 Method of Simakov et al.

The idea behind randomized correspondence algorithm (PatchMatch) was already described in sec. 2.7. As noted, PatchMatch itself can not be used for image completion (or other image maniuplation technique) task directly but rather serve as a fast method to search for nearest neighbors for the particular method. More specifically, an image completion algorithm that relies on search of dense correspondences (i.e., finding nearest neighboring patches for all patches) would be an ideal candidate for incorporation of PatchMatch. The one, also mentioned by authors of [4], is presented in [27].

3.5.1 Summarizing visual data using bidirectional similarity

Simakov et al. proposed a method of summarization of visual data based on a bi-directional similarity measure that aims to provide a metric for classifying the similarity between two images or videos (of possibly different sizes).

The Bidirectional Similarity Measure The measure is given by the following definition: two visual signals S and T are considered "visually similar" if as many as possible patches of S (at multiple scales) are contained in T, and vice versa. The signals S and T are of the same type (i.e., images in case of inpainting) and do not need to be of the same size. The measure is formally defined as follows:

$$d\left(S,T\right) = d_{complete}\left(S,T\right) + d_{cohere}\left(S,T\right), \text{ where} \tag{3.1}$$

$$d_{complete}\left(S,T\right) = \frac{1}{N_S}\sum_{P\subset S}\min_{Q\subset T} D\left(P,Q\right),$$

$$d_{cohere}\left(S,T\right) = \frac{1}{N_T}\sum_{Q\subset T}\min_{P\subset S} D\left(Q,P\right) \text{ and}$$

P and Q denote the patches in S and T, respectively, N_S and N_T denote the number of patches in S and T, respectively, and $D\left(,\right)$ denotes is any distance metric between two patches (SSD measured in CIE Lab color space is used in [27]) and which is the better the smaller. Illustration of completeness and coherence terms is presented in fig. 3.6. The similarity measure defined above captures two requirements and suggests how "good" visual summary of source image S is the target image T. The *completeness term* indicates how much is T visually complete w.r.t. S (i.e., how much visual data of S is represented in T

) and the *coherence term* whether T is visually coherent w.r.t. S (i.e., whether T does not introduce new visual artifacts that were not observed in S).

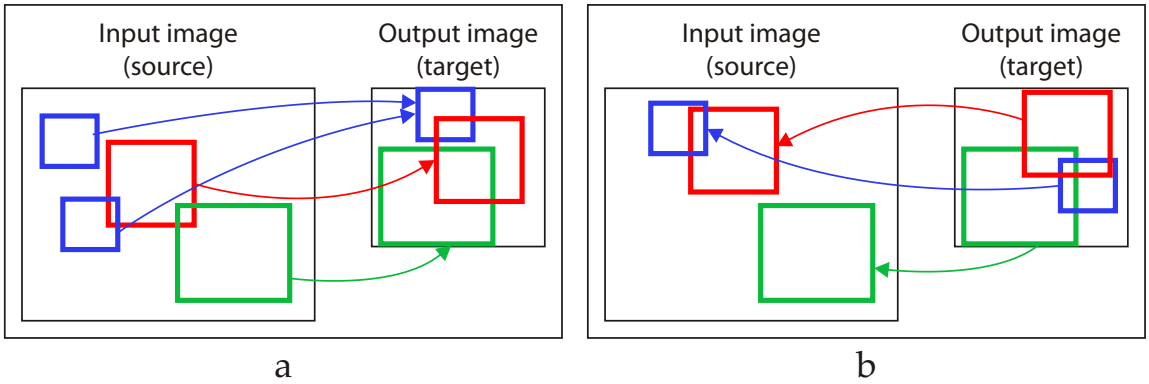


Figure 3.6: The bidirectional similarity measure; image taken from [27]. Two signals are considered visually similar if all patches of one signal are contained in the other signal, and vice versa. The patches are taken at multiple spatial scales. (a) The illustration of the completeness term ($d_{complete}$). (b) The depiction of the coherence term (d_{cohere}).

From the definition of $d(S,T)$, it is obvious that dense patch correspondence is needed, since for every patch $Q \subset T$, the most similar patch $P \subset S$ has to be found. Thus, the spatial geometric relations are implicitly captured by treating images as unordered sets of all their overlapping patches.

Weighted similarity measure The similarity metric can further be modified so the relative weight of both terms may be used: $d(S,T) = \alpha d_{complete}(S,T) + (1-\alpha) d_{cohere}(S,T)$, where α ranges from 0 to 1. This adjustment can be used to emphasize one of the two incorporated terms to better fit needs of desired purpose (as, e.g., image completion).

Summarization (Retargeting) Algorithm With bi-directional (dis)similarity measure defined, the iterative algorithm is proposed to iteratively minimize it. I.e., given a source image S, the task is to reconstruct a target image T that optimizes the similarity measure. Such a output, denoted as T_{output}, is defined as follows:

$$T_{output} = \arg\min_{T} d(S,T).$$

Two key points of this optimization problem must be stressed out. The iterative update rule and the use of iterative (coarse-to-fine) gradual resizing algorithm.

Iterative update rule The algorithm that minimizes the metric is an iterative process. All pixels in T contribute some error value both to d_{cohere} and $d_{complete}$. Let $q \in T$ denote a pixel in T and $T(q)$ its color. Then these contributions are defined as follows:

Contribution of pixel q to d_{cohere} term: Let $Q_{i=1...m}$ denote all patches in T that contain pixel q (see illustration in fig. 3.7) and $P_{i=1...m}$ the nearest neighbors (most similar)

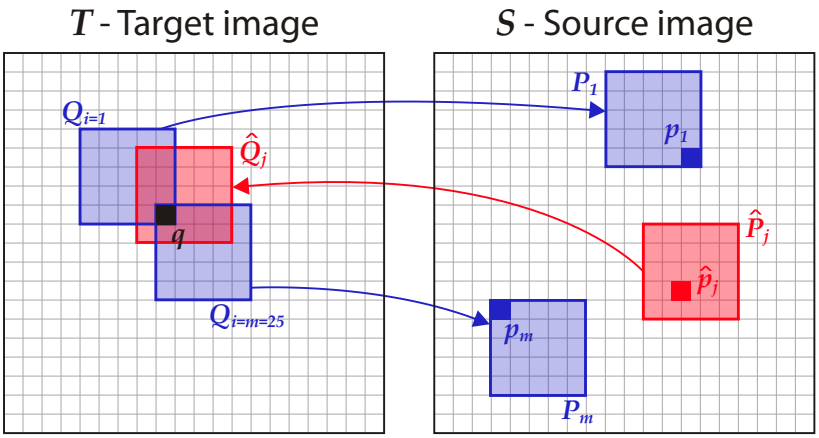


Figure 3.7: Notations for the update rule; image based on image taken from [27]. For all patches $Q_{1...m}$ (blue) from target image T containing processed pixel q (black), their nearest neighboring patches $P_{1...m}$ are found in the source image S and pixels $p_{1...m}$ corresponding to q are processed to evaluate the value d_{cohere}. Also, for all patches $\hat{P}_j$ (red) from source image S, whose nearest neighbor patches $\hat{Q}_j$ contain pixel q, pixels $\hat{p}_j$ are processed to obtain the contribution of processed pixel q to $d_{complete}$ term.

patches in S. Let $p_{i=1...m}$ be the pixels in patches $P_{i=1...m}$ corresponding to the positions of pixel q within $Q_{i=1...m}$.Then $\frac{1}{N_T}\sum_{i=1}^{m}\left(S\left(p_i\right)-T\left(q\right)\right)^2$ is the contribution of pixel q to the coherence term.

Contribution of pixel q to $d_{complete}$ term: Let $\hat{Q}_{j=1...n}$ denote all patches in T that contain pixel q and are nearest neighbors for some patches in S. Unlike m, which is fixed for all pixels, the value of n varies from pixel to pixel and may also be zero. Furthermore, let $\hat{p}_{j=1...n}$ be the pixels in patches $\hat{P}_{j=1...n}$ corresponding to the positions of pixel q within $\hat{Q}_{j=1...n}$. Then the contribution of pixel q to the completeness term is defined as $\frac{1}{N_S}\sum_{j=1}^{n}\left(S\left(\hat{p}_j\right)-T\left(q\right)\right)^2$.

Having the contribution of a single pixel to both terms, the update rule (for a pixel q) may be defined as follows:

$$T\left(q\right)=\frac{\frac{1}{N_S}\sum_{j=1}^{n}S\left(\hat{p}_j\right)+\frac{1}{N_T}\sum_{i=1}^{m}S\left(p_i\right)}{\frac{n}{N_S}+\frac{m}{N_T}}.$$

The iterative algorithm that minimizes the global error can be then summarized in the following way (also see fig. 3.7):

1. For each target patch $Q\subset T^l$ (where T^l denotes the target image refined in l-th iteration) find its nearest neighboring patch $P\subset S$. Colors of pixels in P are votes for pixels in Q with weight $\frac{1}{N_T}$.

2. For each source patch $\hat{P}\subset S$ find its nearest neighboring patch $Q\subset T^l$. Colors of pixels in $\hat{P}$ are votes for pixels in $\hat{Q}$ with weight $\frac{1}{N_S}$.

3. For each target pixel q update the color and set the new value in the next target T^{l+1} as a weighted average of all its votes obtained in steps 1 and 2.

Gradual resizing In [27], a gradual resizing algorithm is suggested. However, since the purpose of the original algorithm is slightly different than just image inpainting, it is not necessary to describe it here in full extent. Briefly, it is used to overcome the problem of possibly large differences in size between the source image S and the target image T, since the local refinement process converges to a good solution only if the initial guess is "close enough" to the solution. Therefore, if the initial target was obtained simply as a resized source, the update algorithm could get trapped in local minima and the output could be unsound. Instead, to avoid this problem, the initial target is resized gradually and on each level of the pyramid the intermediate target is first iteratively improved before the process may continue.

3.5.2 Incorporating PatchMatch

In section 3.5.1, the algorithm proposed in [27] was described. To be able to use it for purpose of image completion, few changes of the default scheme must be made.

Non-uniform importance The (dis)similarity measure, given by eq. 3.1, supposes that all patches are equally important. In case of image inpainting, however, this could be a wrong assumption. In fact, the patch-to-patch distance (i.e., the distance between the patch an its nearest neighbor) should be incorporated in some way. Therefore, a weighting parameter is introduced to eq. 3.1:

$$d\left(S,T\right) = \frac{\sum_{P\subset S} \omega_P \min_{Q\subset T} D\left(P,Q\right)}{\sum_{P\subset S} \omega_P} + \frac{\sum_{Q\subset T} \omega_{\hat{P}} \min_{P\subset S} D\left(Q,P\right)}{\sum_{Q\subset T} \omega_{\hat{P}}}$$

where ω_P is the patch importance and $\hat{P} = \arg\min_{P\subset S} D\left(Q,P\right)$. The weights are defined over the source image.

Omitting the completeness term So far, the bidirectional similarity measurement was based on two terms - completeness term $d_{complete}$ and coherence term d_{cohere}. As defined, the completeness term makes the requirements on the target image T in the way that it should contain as many patches from source image S as possible. However, when using the scheme, where the rest of the image except the hole (the unknown pixels) serves as the source region, this condition is automatically fully satisfied (since the desired target T_{output} is the exact copy of source plus some new pixel values on positions marked as unknown). Therefore, the completeness term can be omitted by setting the parameter α to 0. In fact, this operation reduces the similarity measure to the very same algorithm as used in [29].

Gradual resizing To avoid getting stuck in bad local minima, a multi-scale gradual scaling process is used. The image is repeatedly downscaled (using the Gaussian pyramid technique as described in sec. 2.4.3). Then at the lowest resolution, several iterations of the bidirectional similarity algorithm are run using PatchMatch as the "provider" of nearest neighbor pairing

(with random initial assignment). The final target output T_{output} is then upsampled and is used as a initial guess or hint in the next level. The number of iterations of both bidirectional similarity measure refinement and the PatchMatch varies from level to level.

Chapter 4

Implementation

The implementation section covers the interesting points of implementation parts of algorithms described in chapter 3. The full source code listing can be reviewed on accompanied CD.

4.1 OpenCV

Prior the description of selected parts, the chosen framework must be briefly introduced. OpenCV (Open Source Computer Vision Library) [9] is a multiplatform open source library mainly aimed at computer vision and image processing. It is originally written in C with wrappers in many others programming languages. Moreover, since version 2.0 the C++ interface is also provided.

Thus, the OpenCV library was used during implementation extensively since it already has written many both essential and non-essential image processing algorithms as well as, e.g., matrix handling and is optimized at high-level.

4.2 Framework overview

Since the implemented algorithms are all exemplar-based, they share many common properties and concepts. E.g., they all use the known part of the image (or its subset) as the source zone, they all use similar distance metric (with the exception of [3], all the algorithms use SSD as the distance metric) and some others. Thus, some of these common properties could (and should) be extracted to exploit the refactorization of code. The core of the implementation concept tends to be interface-based (even though there is no such thing like in C++). The following overview of important class will be briefly introduced before describing some of them in more detail:

- `IPaper` - an interface encapsulating the methods of a paper implementation. All implemented image completion algorithms must extend this class. Since slightly different parameters for each algorithm may be needed, the common input/output class is required.

- `IO` - an input/output class. Provides method to store and pass variables of different primitive data types such as numbers, strings or pointers.
- `ISampler` - an interface providing the basic sampling operations to sample from target zone. Not all papers may take advantage of this (e.g. the PatchMatch since it aims to find dense correspondences) but nevertheless it has proven useful.
- `ISearcher` - the other similar concept to sampler is the searcher. Object implementing the searcher interface finds for given point or patch from target zone the best matching point or patch, respectively, in source zone.
- `ISolver` - since several algorithms use a repetitive scheme "sample from target → search for best match" until the missing part is re-synthesized, this interface and its default implementation `SimpleSolver` provide methods to encapsulate it.
- `DistanceMetrics` - the class providing various distance metric to compare two pixels or, more precisely, their patches.
- `SimpleImage` - possibly the very heart of the framework. Encapsulates the basic image instance handling, pixel access, target and source zone management and others as well as some image-processing-related methods such as the creation of Gaussian pyramid with clever mask handling as described in sec. 2.4.3. and illustrated in fig. 2.4.
- Matrix operations - set of functions (not occupying one common class) providing useful tools (like, e.g., gradient based filling described in sec. 3.4.1).

The aforementioned classes form the basis of the whole implementation, wrapped in namespace `hfl` (Hole Filling Library). More detailed description of some of these classes will be given in the following sections of this chapter.

4.2.1 Image class

The image class is the common and connecting element of the rest classes and functions in the `hfl` namespace. Almost all other instances like searchers, samplers or input/output processing have something to do with the `SimpleImage` class. The input image might be created either by passing image filenames or the `cv::Mat` objects (which is the 2D matrix class in OpenCV library). The image instance takes care of the image data, stores the information about source and target zones, patch size etc. It can be basically imagined as a 3-layered image as shown in fig. 4.1b. These layers are of the same size and, as can be seen in fig. 4.1a, the image, as well as the target and source zones, is padded by certain distance, usually set as *patch radius* (see sec. 1.4). Thanks to this simple modification of original data, no boundary checking is required while comparing the patches near the boundaries, which gives the algorithms some extra performance.

The image is by default stored as set of triplets of 8-bit unsigned characters defining the colors in BGR space (which is the same as RGB except the order of channels is reversed) to make use of memory coherence. However, some of the algorithms may need to process the image in different precision. Therefore, all three layers are accessible to the user without no restrictions. On the other hand, the framework is not responsible for any incompatibilities

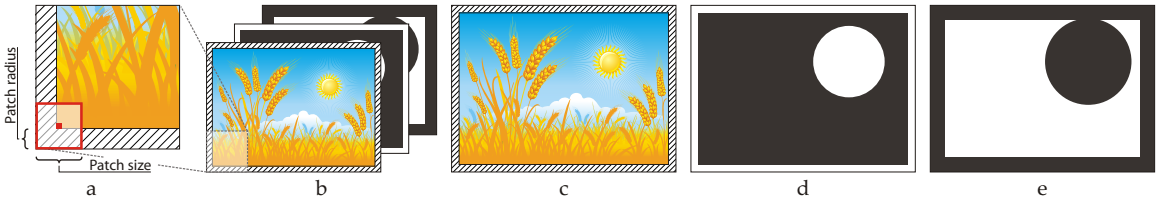


Figure 4.1: The illustration of how the image is being handled in the described framework. (a) The explanation of the patch size (which is odd number) and patch radius. By setting the padding at least to the value of patch radius, the boundary checking (i.e., testing whether the patch does not span out of the image) can be omitted. (b) The depiction of the 3-layered concept. (c) The input image plus the padding area (shaded). Only the area corresponding to the original image (without padding) is processed. (d) The mask corresponding to the input image. The white areas mark the pixels where the mask is set. The padding is by default also set to `true`. (e) The allowed zone (source region). The source region is by default set as the inverse of the mask minus a dilated band near the boundaries between masked/known regions.

incurred by inappropriate use. The target (also referred to as masked) and the source (allowed) zones are by default binary images with ones placed on pixels where the specific property is `true` (e.g. where the the pixel is masked) and zeros where the property is set to `false`. The image instance also stores the data related to patch size such as patch size or patch radius.

To summarize, the `SimpleImage` class encapsulates all the needed data and methods to efficiently handle the image data during image completion process.

4.2.2 Input/output class

Despite all being based on exemplar-based approaches, the different implemented methods may require slightly different parameters. Thus, a unifying point for passing information to every single one of them is highly required. The `IO` class represents such a class for exchanging information via one encapsulating instance. The class provides methods to:

- store the information of different primitive data types
- loading stored information - the popped information can be converted to any supported data type if it is possible (e.g. one numerical data type can be converted to another)
- checking whether the parameter exists and is stored within the class instance

4.2.3 IPaper interface

The `IPaper` interface serves as an entry point for a custom paper image-completion-related implementation. As all classes or functions of this project, it is contained within the `hfl` namespace. The interface is particularly simple and besides the methods for image synthesis,

it provides the basic info about the implemented paper. In more detail, there are methods for:

- providing the basic information about the implemented algorithm, namely the title, abstract text and the names of the authors
- synthesis of the image - the parameters are passed via an instance of the `IO` class
- time measuring, more specifically the method that returns the elapsed time of the last image completion

No additional constraints are placed on any implemented algorithm, thus making their use by the end-user substantially easy. Also, adding newly implemented papers into the framework would be quite simple.

4.2.4 ISampler interface

The sampler is a common concept among the implemented algorithms. With exception of the PatchMatch, the rest of the algorithms uses some kind of sampling from the target zone. An example would be basic scanline sampling, when the samples are obtained by iterating through the image in row-major order, or the spiral (onion) sampling, when the unknown area is traversed in concentric spiral manner. To separate the sampling from the rest of the algorithm, and to provide a way to quickly change the sampling method to compare results, the `ISampler` interface is important. It separates the implementation and thus gives the possibility of design rapid changes that would otherwise mean replacing large parts of code. Giving an example, if one had wanted to change the sampling method in the algorithm stated in [13], the whole blocks of code responsible for sampling would have had to rewrite almost the whole part. In contrary, by using a sampler instance implementing `ISampler` interface, this can be done instantly. Thus, the required functionality is:

- sampling from the target zone of given SimpleImage instance or from another, specifically passed, binary image (respecting the notation of masked/known pixels as stated in sec. 4.2.1)
- updating the the image after a best match is found (using the searches, see sec. 4.2.5) - usually, some post-processing must be done before next sample can be generated and, therefore, the interface provides a method to do it

4.2.5 ISearcher interface

As with the sampler, the searcher is another common concept. Given a masked pixel with partially unknown patch, the task is to find its nearest neighbor, i.e., the most similar patch, with respect to some predefined criteria. Moreover. although the methods to achieve this do differ significantly, they all share the input/output point and are, therefore, the candidates for another encapsulating interface. The `ISearcher` interface provides the desired functionality plus contains methods for possible parallel processing within each searcher.

4.2.6 ISolver interface

As mentioned, the very core concept of several exemplar-based image synthesis algorithms can be summarized as the repetitive loop of sampling and searching, as illustrated in fig. 4.2. Therefore, a solver instance utilizing both given sampler and solver is implemented with the `SimpleSolver` class as its basic implementation.

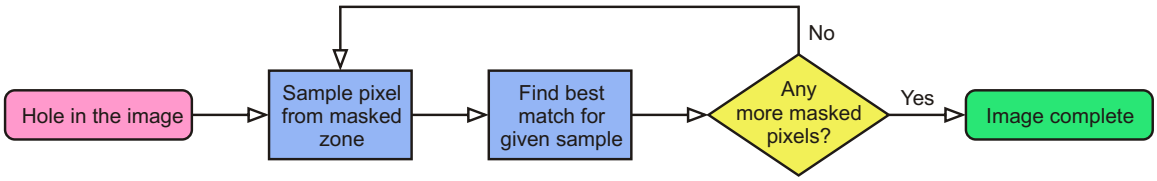


Figure 4.2: Flowchart of simple solver scheme.

4.2.7 Matrix operations

Although the majority of implemented work is sorted in different class, few common functionality have been separated and are implemented as global within the `hfl` namespace scope.

Selected parts

Describing the whole implementation, class by class, is surely unnecessary. Therefore, only a selection of interesting classes will be given in this section, with more detailed description.

4.3 Samplers

4.3.1 Scanline sampler

Starting with basic sampler, the instance `ScanlineSampler` proceeds as follows: Given an input `SimpleImage` class instance, the sampler first examines its masked zone (ignoring the optional padding) and initializes the internal pointer to the first element of the list. When a request for next sample is made by calling the method

$$\texttt{boolsample}(\texttt{cv} :: \texttt{Pointcenter}),$$

the sampler first checks whether the internal pointer points to a valid element. If so, the given point parameter is assigned the sample value and a boolean `true` is returned to signalize that the sample was successfully passed. Otherwise, if all points from masked zone were already sampled, no value is assigned to the parameter `center` and the sampler return `false` to signalize that the sampling has finished.

If `true` is returned, the last generated sample is stored to be processed in the update phase of the iteration, and after its nearest neighbor is obtained (outside the sampler), method

`voidupdate(cv :: PointbestExemplar)`

must be called to update the image instance for which the sampler was created.

Settings For its simplicity, this sampler implementation provides only a little space for any adjustable behavior:

- **Sampling order** - the order of sampling can be set either to scanline (from top-left corner to the bottom-right one) or reverse scanline
- **Fill mode** - whether to fill the whole patch around the last sample with the data from patch around the best exemplar or only the pixel value itself

4.3.2 Onion peel sampler

As demanded in [3] or mentioned in [11], the spiral order is used in many hole filling approaches to produce smoother and more coherent result. The implementation of spiral iterator, class `OnionPeelSampler`, makes use of the `cv::findContours` function provided by the OpenCV library. As in case of `ScanlineSampler`, the list of points is initialized before the sampling starts. This is possible due to the fact that the order of sampled points can be predetermined in advance (in contrary to, e.g., the `ImportanceSampler`). Note that in the case of more separated holes present in the target region, the concentric layers are taken across all holes before processing the next contour (see fig. 4.3 for clearer idea).

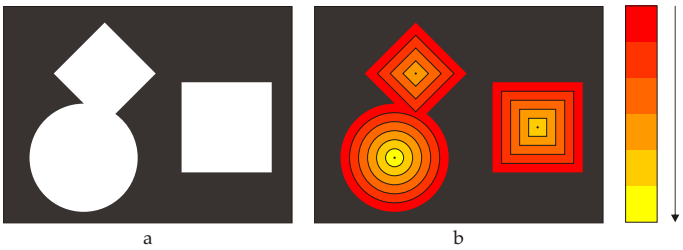


Figure 4.3: Illustration of sampling in spiral manner. (a) The mask of the image. White areas are marked as to-be-synthesized pixels. (b) Order of sampling. Each color band represents a one-pixel thin contour. The samples are generated from the outermost contour of the target region and proceed inwards in a process resembling the peeling of onions, hence the name. The order of processed contours is also emphasized by the color scale ranging from red (outermost contour) to yellow.

Settings Similar to the previously described sampler, the onion peel sampler only have two changeable parameters:

- **Sampling order** - the default order is clockwise and can be changed to counter-clockwise
- **Fill mode** - whether to fill the whole patch around the last sample with the data from patch around the best exemplar or only the pixel value itself

4.3.3 Fill ratio sampler

Inspired by the confidence term introduced in [11] and described in sec. 3.3.2, the idea behind the `FillRatioSampler` class is to fill first those unknown pixels, whose patches contain more valid pixels, since these pixels contain more pixels against which the candidates for best exemplar can be matched. As showed in [11], this assumption itself does not guarantee the sufficient propagation of linear structure, however, represents more reasonable approach than the basic scanline sampling that does not reflect the shape of the masked areas at all. In contrary to previously described samplers, the `FillRatioSampler` can not precompute and store the order of outgoing samples since the order is determined by the found best exemplars and which is determined outside the sampler.

Settings

- **Fill mode** - whether to fill the whole patch around the last sample with the data from patch around the best exemplar or only the pixel value itself

4.3.4 Priority sampler

The priority sampling, as it is presented in [11], represents an advanced method to generate samples. Unlike `ScanlineSampler` or `OnionPeelSampler`, the algorithm implemented in `PrioritySampler` class does reflect the actual state of the image and generates next sample by using it. According to the algorithm described in 3.3.2, the priority of the patch is determined by evaluating its confidence and data terms. While computation of the confidence term is relatively simple, the evaluation of the data term will be described in more detail.

Boundary normal estimation In the original paper, the estimation of normal vector $\mathbf{n}_p$ to the contour line is obtained as follows: Once the contour line $\delta\Omega$ has been found, for currently processed point $\mathbf{p} \in \delta\Omega$ the normal is estimated as the unit vector orthogonal to the line through the preceding and the successive points in the contour. However, this only gives a rough approximation since there are only a fixed number of combinations resulting only in limited set of possible angles (see fig. 4.4). Therefore, the accuracy of the normal estimation should be increased.

To improve it, different method must be incorporated. More specifically, at first, the distance transform of the image mask must be obtained. Note, that the distance map must be recreated every iteration, since the mask of the image changes during the course of image synthesis. The example of such distance transform is illustrated in fig. . Then, the boundary normals are computed using the distance map as a gradient. To improve the results, robust gradient operator should be used.

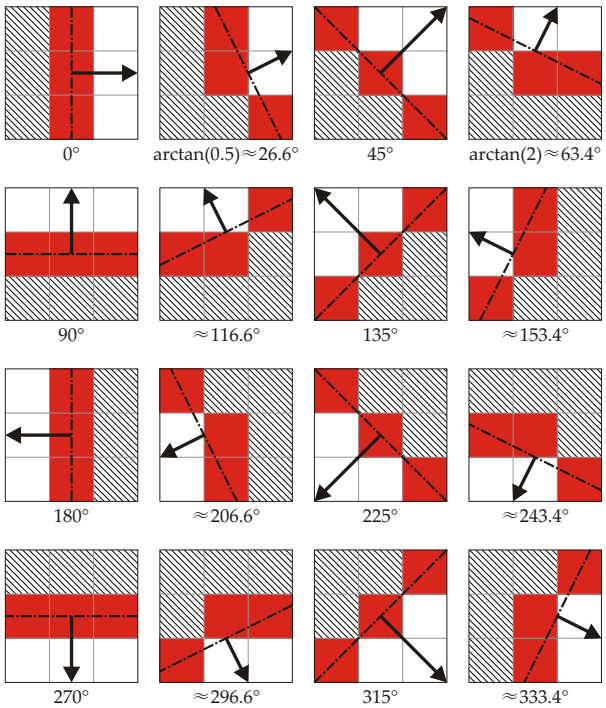


Figure 4.4: Contour normal estimation using contour pixels only. The unique angle values that can be obtained when using the contour normal estimation method mentioned in [11]. The shaded area determines the known part of the image Φ while the white one represents the image hole Ω and red pixels denote its $\delta\Omega$. The symmetry is clearly visible.

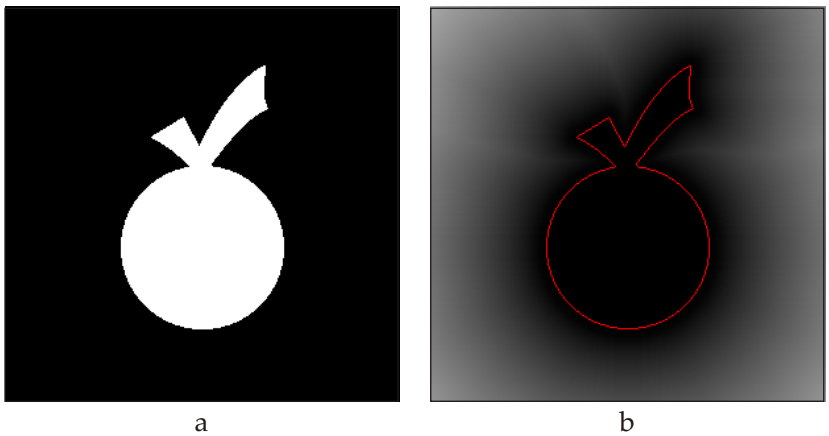


Figure 4.5: Contour normal estimation using distance transform (L_2-norm). (a) The image mask. (b) The distance transform with the contour $\delta\Omega$ marked in red. The contour is a 8-connected line.

Isophote estimation According to the proposed algorithm, the isophote is computed as the as the maximum value of the gradient $\nabla I_{\mathbf{P}}$ in $\Psi_{\mathbf{p}} \cap \Phi$ rotated by 90°. The gradient is computed on the grayscale version of the processed image (which must also be updated

after retrieval of the best exemplar). Since the simple numerical gradient is not sufficient, another robust operator, such as *Scharr* operator or the ones proposed in [19], should be used. However, the pixels marked as unknown could obviously affect the resulting gradient. In fact, since the target region is filled with zeros (thus appearing as black color), there would likely be a significant gradient magnitude between the blackened holes Ω and the rest of the image Φ as illustrated in fig. 4.6c.

To overcome this, similar approach to the one described in fig. 2.4 could be used. However, since the OpenCV image routines are highly optimized, rewriting the convolution algorithm could lead to potential loss of performance (which does not matter in case of creating the image pyramid since there are only several iterations in contrary to approximately two orders of magnitude larger count of gradient computation during the course of the inpainting process).

Instead, since the gradient is computed in a floating point format, one can make use of incorporating the `NaN` (Not A Number) value, introduced by the IEEE 754 floating-point standard [21]. `NaN` is a numeric data type value that represents an undefined or unrepresentable value, especially in floating point calculations (note that `NaN` is not the same as the infinity value, although both are typically handled as special cases).

There are three kinds of operations that can return `NaN`: [15]

1. Operations with a NaN as at least one operand.
2. Indeterminate forms
   - The divisions $\frac{0}{0}$ and $\frac{\pm\infty}{\pm\infty}$.
   - The multiplications $0 \cdot \pm\infty$ and $\pm\infty \cdot 0$.
   - The additions $\infty + (-\infty)$, $(-\infty) + \infty$ and equivalent subtractions.
   - The standard has alternative functions for powers:
     - The standard `pow` function and the integer exponent `pown` function define 0^0, 1^∞, and ∞^0 as 1.
     - The `powr` function defines all three indeterminate forms as invalid operations and so returns `NaN`.
3. Real operations with complex results, e.g:
   - The square root of a negative number.
   - The logarithm of a negative number.
   - The inverse sine or cosine of a number that is less than -1 or greater than $+1$.

There are also two kind of `NaN`s:

1. Signaling `NaN` (`sNaN`) - the type of `NaN` that, being processed by most operations, should raise an invalid exception and then, if appropriate, be "quieted" into a `qNaN`.
2. Quiet `NaN` (`qNaN`) - the type of `NaN` that does not raise any additional exceptions as they propagate through most operations.

a b c d

Figure 4.6: The example of using qNaN value to remove the unwanted raise of gradient magnitude at boundaries between the source and target region. (a) The original image. (b) The portion of the image that is to be removed is selected by the user. The red line denotes the initial contour line $\delta\Omega$ and the hole Ω is by default filled with zeros. (c) The gradient magnitude on the untreated image hole (and padding) creates undesired artifacts thus compromises the evaluation of the data term. (d) The gradient magnitude on the image in which the zero values on missing pixels were substituted by `qNaN` values.

In particular interest is the fact that the `NaN` is returned when at least one of the operands is `NaN`. Therefore, if we fill the holes with `qNaN` values, they will not be treated as if they have any valid color at all and the gradient near the boundaries will not be affected by the content of the missing part of the image. There is, however, one minor disadvantage - some values within certain distance from outside of the image holes will be lost. Fortunately, since the Scharr operators in the implementation use kernels of size 3×3, namely

$$\begin{pmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{pmatrix} \text{ and } \begin{pmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{pmatrix},$$

this distance will only be one or two pixels at maximum and since the single pixel is rarely a feature on itself, this operation can be processed without a significant loss of the quality of the output image.

4.4 Searchers

The searcher classes, as the name suggests, are responsible for finding the nearest neighbor (or best exemplar) of given pixel, respectively its surrounding patch. They examine the given part of the image and find the solution using of the algorithms described in chap. 3.By implementing the `ISearcher` interface, the searcher can be used (as in case of samplers implementing the `ISampler` interface) as a modular "box" that can be easily changed in the synthesis process. In this section, the implemented searchers will be described.

4.4.1 Exhaustive searcher

`ExhaustiveSearcher` is the basic searcher class designed mainly for purpose of [13]. For a given image part specified by target region matrix (see fig. 4.1e), the area is first preprocessed and structures that are used repeatedly (e.g., the matrix headers or patch centers) are stored to spare some processing time. Since `ExhaustiveSearcher` supports the parallel processing, the image may be tiled into several horizontal regions based on the number of required threads. I such case, the pre-prepared structures are initialized per image tile.

The rest of the processing is simple. When the searcher is requested to find the best exemplar for given image point, it iterates through the precomputed data structures and selects the best matching one based using the comparison of a distance metric, which is, by default, sum of squared differences (SSD).

Modes The exhaustive searcher can be run in three modes:

- **Linear** - the basic single-core mode
- **Parallel** - the multi-core routine; note, however, that the speedup is not linear since the threads must by synchronized just before the final nearest neighbor value for the queried pixel can be assigned.
- **SSD** - single-core implementation using the `PSADBW` instruction; the distance metric is thus changed to sum of absolute differences (SAD) and the results may vary from results obtained using the other modes

Sometimes (like in [25] when the set of 0.1% best candidates is probed to determine the overall nearest neighbor), there is a need for selection the best exemplar from the set that changes every iteration. Preprocessing the data would be useless in such occasion and, therefore, another method (besides the `ISearcher` interface) is implemented in the exhaustive searcher class for this particular case.

4.4.2 Patch vector searcher

The search data structure described in [3] is implemented within the `PatchVectorSearcher` class. Having the image instance, maximal error distance and minimal candidates count as the input parameters, it initializes the search data structure.

The structure consists of several arrays. For each image channel the `data` array is created. This array has dimensions of $256 \times |\Psi_p|$, where 256 is to represent all values of 8-bit color channel and $|\Psi_p|$ denotes the area of the patch, i.e., the square of patch size. In each cell, a list to store the identifiers of patches having the specific value on given patch position is needed. The choice of this list gets more and more important with the increasing size of the search space. Using `std::vector` class satisfies good cache performance, but with increasing image size, the re-allocation of the data will consume significant time.Using a `std::list` is also not a good option. Although it lacks the need of re-allocating the data, it breaks the principle of locality and impose additional memory overhead associated with storing list metadata such as references.

Therefore, an optimal structure would be a combination of both aforementioned ones - the unrolled linked list. An unrolled linked list consists of nodes that all contain fixed number n of elements, thus exploiting the spatial locality. Also the memory overhead is reduced since there is only one pointer needed per n elements (see fig. 4.7). Setting the value of n must be then made considering the proportions of both the given image and the patch size.

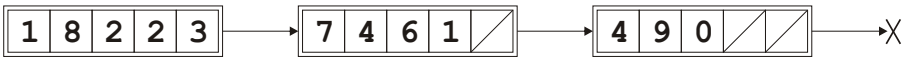


Figure 4.7: Illustration of the unrolled linked list consisting of three nodes capable to contain up to five elements per node.

The last array used within the searcher is the `occurrence` array. As its name suggests, it simply counts the occurrence of patch identifiers. On every iteration, the array must be first cleared. Then, as the patch around the query pixel is being examined and the lists in all channels' `data` arrays are being traversed, the occurrence counter is incremented on corresponding position. After all lists are processed, the occurrence array is examined and if it contains enough candidates to criteria the rule described in alg. 2, the set of potential candidates is returned. Otherwise, next iteration is performed testing the data structure with increased error value (see alg. 2 for details). If not enoughcandidates is found after the last iteration, the set of so-far-best candidates is outputted.

Among the returned candidate, the overall best match is selected by using the instance of `ExhaustiveSearcher`.

4.4.3 Phase correlation searcher

The `PhaseCorrelationSearcher` class utilizes the phase correlation to find the nearest neighbors of the given query pixel/patch as described in sec. 2.5. After the input image is provided, it must first be preprocessed - the missing part must "pre-synthesized", i.e., filled with some initial information. This operation must be as less time-consuming as possible. Therefore, the missing pixels in the holes are assigned the values obtained by averaging the known pixels from their neighborhood, resulting in blurred area. The iteration over the missing pixels can be done in scanline or spiral manner. After this operation is done, the grayscale copy of the image is made for the purpose of Fourier transform.

The searcher then proceeds in simple iterations until the content of the missing area is re-synthesized. First, the grayscale query patch is placed in the center of otherwise black image A (thus creating the impulse) and the whole input image (also grayscale) is placed into the image B. Note that both images have the proportions of the input `SimpleImage` instance minus the padding. Then, the phase correlation is performed on mentioned images and the locations of peaks are examined. The position of n highest values placed within the source zone form a candidate set of exemplars, from which the best match is selected by an exhaustive search routine.

After the best exemplar is found the grayscale image must be updated (note that for updating the original image is responsible the sampler instance).

4.4.4 Fast query searcher

Similar to `PatchVectorSearcher`, the `FastQuerySearcher` class utilizes a precomputed search structure to speed up the search process. The main difference, comparing to the one proposed in [3] is the fact that the number of stored coefficients (thus the number an length of lists that must be traversed) is significantly reduced while the structure is being created. The arrays themselves are also very similar. The `data` array has dimensions of a patch (which is few orders of magnitude less than in case of the structure suggested in [3]) having a list, storing identifiers of patches that have the non-truncated DCT coefficient on corresponding patch position, in each cell. The `data` array is again created per image channel. Another array, the `baseScore` array, is initialized to contain the precomputed values of one of the $\tilde{d}_{\hat{q}}$ term described in [25]. This array serves during every iteration as an initial score for every patch. Note, however, that the method of [25] was implemented only on CPU and thus the result run times of the algorithm might be more or less slower than the result documented in the original paper.

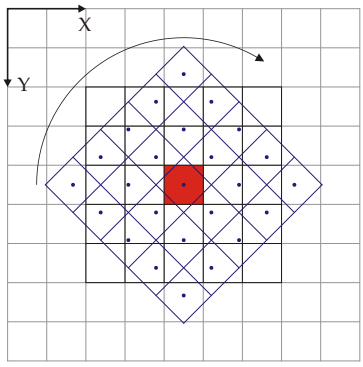
Gradient-based filling The gradient fill method that performs the gradient-based filling of examined incomplete patches (see description in sec. 3.4.1) proceeds in two iterations. In first iteration, the number of variables and the number of equations of the overdetermined linear system is found. In the second pass, the matrix A and the right side vector b of the equation $Ax = b$ are assembled. Each row of the matrix represents a single equation that bounds the single unknown pixel from the patch to one of its top or right neighbor assuming the discrete gradient to be zero. After the data structures are assembled, the solution of the system is found by using the OpenCV built-in least squares method.

4.5 PatchMatch

The PatchMatch, proposed in [4] and described in sec. 2.7, can not be simply assigned to the searchers since it proceeds in a different way. Instead of finding a single nearest neighbor per iteration, it searches for best matching across the whole image and iteratively improves the initial guess during the course of next iterations. Thus, it can not be implemented as a derived class of `ISearcher` interface.

The PatchMatch class provides convenient methods for initialization (either by randomly assigned values or by another, possibly downscaled, nearest neighbor field, NNF) and iterations. It must also be pointed out that since the PatchMatch is applied in the gradual resizing process, the target region is constructed ass the whole image minus the target zone, i.e., the source patches may contain missing pixels as well (as long as as the central pixel of the patch is known). This modification must be done because in smaller resolutions, it is not desired to further reduce the search space.

Handling rotations The rotations, introduced to PatchMatch scheme in [5], extend the search space to the another dimension examining the matching also across the predefined range of angles. To efficiently handle the rotations, the patch offset mask must be precomputed for all possible angle values. This matrix of size of the patch stores the values of the relative offset of each patch element as illustrated in fig. 4.8.



(2,-1)	(2,0)	(1,1)	(1,1)	(1,2)
(1,-1)	(1,0)	(1,0)	(0,1)	(0,2)
(1,-1)	(0,-1)	(0,0)	(0,1)	(-1,1)
(0,-2)	(0,-1)	(-1,0)	(-1,0)	(-1,1)
(-1,-2)	(-1,-1)	(-1,-1)	(-2,0)	(-2,1)

Figure 4.8: The example of patch offset mask for angle of 45°. The original pixel (red) and its patch of size 5 (black) is rotated and the relative offsets (blue table on the right) are taken as using nearest-neighbor interpolation method.

4.6 Papers

Having the image synthesis logic disassembled into the sampling and searching units, the papers themselves consist usually only from these two blocks and therefore, there is no need to specify them in more detail. There is, however, one exception - the PatchMatch and the Bidirectional similarity measure introduced by Simakov et. al [27] and described in sec. 3.5.1.

4.6.1 Simakov

The code is implemented in class `Simakov` implementing the `IPaper` interface. After extracting the input data from parameter of input/output class `IO`, the algorithm proceeds as follows:

1. The gaussian pyramid, utilizing technique mentioned in sec. 2.4.3 and illustrated in fig. 2.4, is created. The number of levels depends on the specific image and the size of its missing region since new levels of the pyramid are constructed until either the size of the downscaled image is larger than a single patch size or the hole "grows in" as the downsampling continues.

2. The initial target (the smallest image in the pyramid) is then unmasked, i.e. all pixels are explicitly declared as known (the initial source, which is actually the same image, remains untouched). The iterative processing of the levels of the pyramid from the smallest image up to the original one then proceeds as follows:

   (a) The nearest neighbor field of the `PatchMatch` class is initialized, either randomly (if the currently processed image is the initial one) or using previous iteration's solution as the hint.

   (b) Several iterations of the summarization algorithm are run. In each iteration, another several iterations of PatchMatch algorithm are performed (with more iterations on the coarse level) followed by the application of iterative update rule (see sec. 3.5.1). The PatchMatch is used as a tool to find dense correspondences before the update rule can be applied. Contribution of pixels is weighted to emphasize the effect of correctly mapped image areas.

Weighting function As suggested in [27], the optional weighting function can be incorporated to control the update process. In [1], the weight of the pixels is fixedly computed as the inverse of the patch distance between the given patch and its nearest neighbor. However, for better modularity, it would be optimal if the patch distances were first mapped into a certain range of values from 0 to N. Then, a weighting function can be created for the same range, thus providing an easy way to test various patch weighting. Currently, three weighting functions are implemented as the lookup table generating methods in the framework: the distance inverse, linear decrease and the hyperbolic tangent, as illustrated in fig. 4.9.

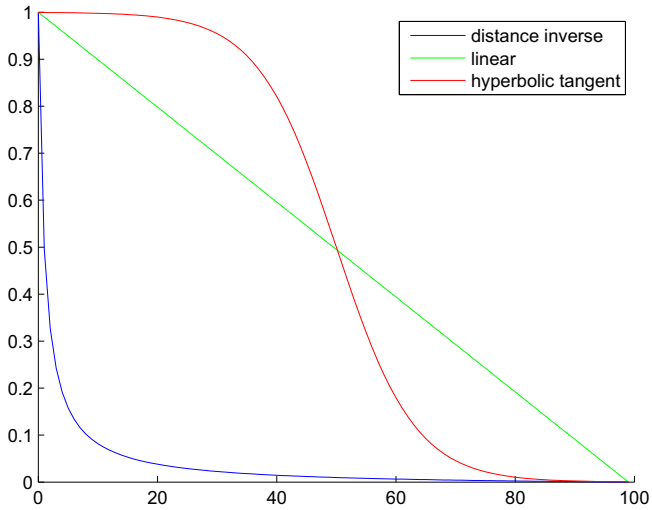


Figure 4.9: Examples of possible patch-weighting functions. The patch distance (to its nearest neighbor) is first scaled in the given range (in this example $0 \ldots 99$). Then the weight of the patch is selected from the discretely sampled weighting function. The values can be also possibly scaled along X-axis moving the zero towards the origin

Chapter 5

Results

In this chapter, the achieved results will be presented. Although the image inpainting algorithms described in chap. 3 share the property of being patch-based, the can not be simply compared in the sense of computation time since some of them are pixel-based (one pixel at the time is synthesized) while the others are patch-based (all uknown pixels in the patch around currently processed pixel are filled at once). Therefore, a visual comparision will be given as well as some performance comparisons where the make sense.

5.1 Algorithm settings

Unless stated otherwise, the presented results were obtained running the algorithms with following settings:

- **The method of Averbuch et al. [3]** - size of the patch 7, maximal error distance 10, minimal number of candidates 10
- **The method of Criminisi et al. [11]** - size of the patch 9
- **The method of Efros and Leung [13]** - size of the patch 7
- **The method of Kwok et al. [25]** - size of the patch 9, number of kept coefficients 9, number candidates 0.1% of total exemplars
- **The method of Simakov et al. [27] using work of Barnes et al. [4]** - size of the patch 7

All tests were run on an Intel Core i5 CPU, 2.53 GHz and with 64-bit Windows 7 as an operating system. Also, an early termination 2.2 method was used on SSD calculation wherever possible.

5.2 Real-world images

5.2.1 Scratch-like holes

Scratch-like holes represent usually an easier task for an image inpainting algorithm than the situation when larger part of the image has to be re-synthesized. However, this type of image synthesis is also required in many cases, when, e.g., an overlaying text must be removed or scratches in an old photography or painting is to be repaired.

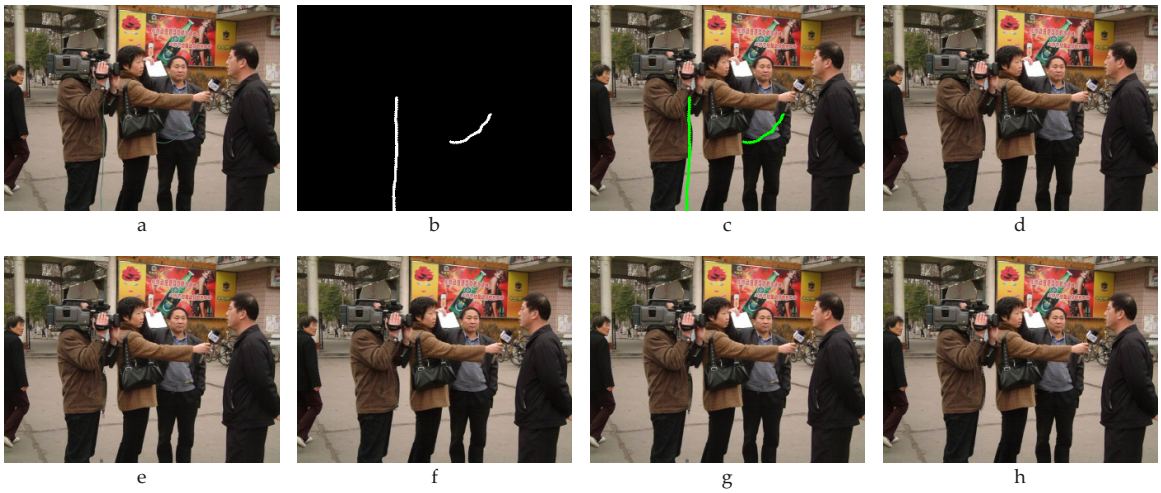


Figure 5.1: Test image "Interview". (a) The original image of size 334×250. (b) The mask covering 0.85% of the image. (c) The input image with emphasized masked areas. (d) The output of [3], 20.049 sec. (e) The output of [11], 0.891 sec. (f) The output of [13], 2.763 sec. (g) The output of [25], 4.021 sec. (h) The output of [27, 4], 0.477 sec.

As can be clearly deduced from the running times, building a search structure for images with only little missing pixels is not efficient. Especially the search structure of [3] does not seem to work very well. To further investigate this observation, the structure must be further examined. Its major drawback is clearly the significant memory footprint which makes the process of its iteration very time-consuming. With increasing patch size and image proportions, the size needed to store the search structure grows rapidly. Fig. 5.3 shows as the number of stored data in lists of search structure grows with increasing size of patch window. In addition, the even more time-consuming operation than the creation of the structure is its iteration. Without possible parallel processing, this operation is simple the bottleneck of the approach of [3].

When synthesizing only few pixels wide holes in rapidly changing area (like in fig. 5.1), the careful setting of the size of patch in case of patch-filling methods (methods that fills the whole patch instead of setting only the single pixel) becomes more important because small artifacts can easily occur (see fig. 5.1e,g).

Comparing all the aforementioned in matter of computational time, the PatchMatch-supported [4] method of Simakov et. al. [27] outperforms the rest of the algorithms by one to two orders of magnitude, as documented in [4].

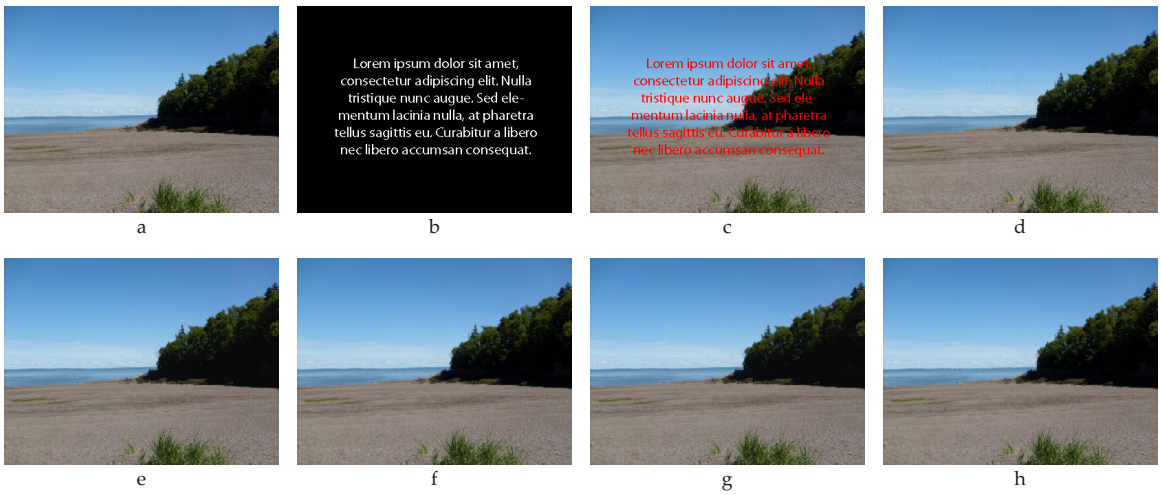


Figure 5.2: Test image "Beach Text". (a) The original image of size 400×300. (b) The mask covering 5.05% of the image. (c) The input image with emphasized masked areas. (d) The output of [3], 85.433 sec. (e) The output of [11], 42.336 sec. (f) The output of [13], 49.693 sec. (g) The output of [25], 27.093 sec. (h) The output of [27, 4], 0.714 sec.

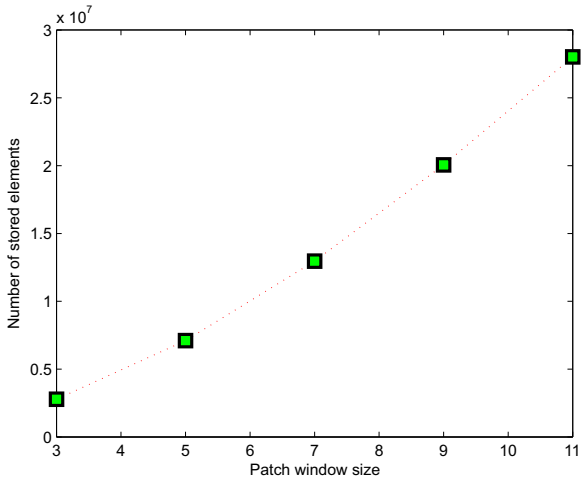


Figure 5.3: The number of elements stored in lists of search structure proposed in [3] for image in fig. 5.2. For example, having the patch of size 9, every patch contributes with $9 \times 9 \times 3$ integer values, giving (assuming 4 bytes per integer variable) 243×4 bytes per single valid patch center. In case of fig. 5.2 and patch size 9, there are 82528 such patches centers (pixels).

5.2.2 Large holes

Completing larger holes is a more difficult task then completing the scratch-like ones. While to synthesize the few pixels wide area, the sufficient information is often located within the

immediate proximity of the holes, in case of reconstructing larger missing region, the best fitting exemplars may be located almost anywhere in the image.

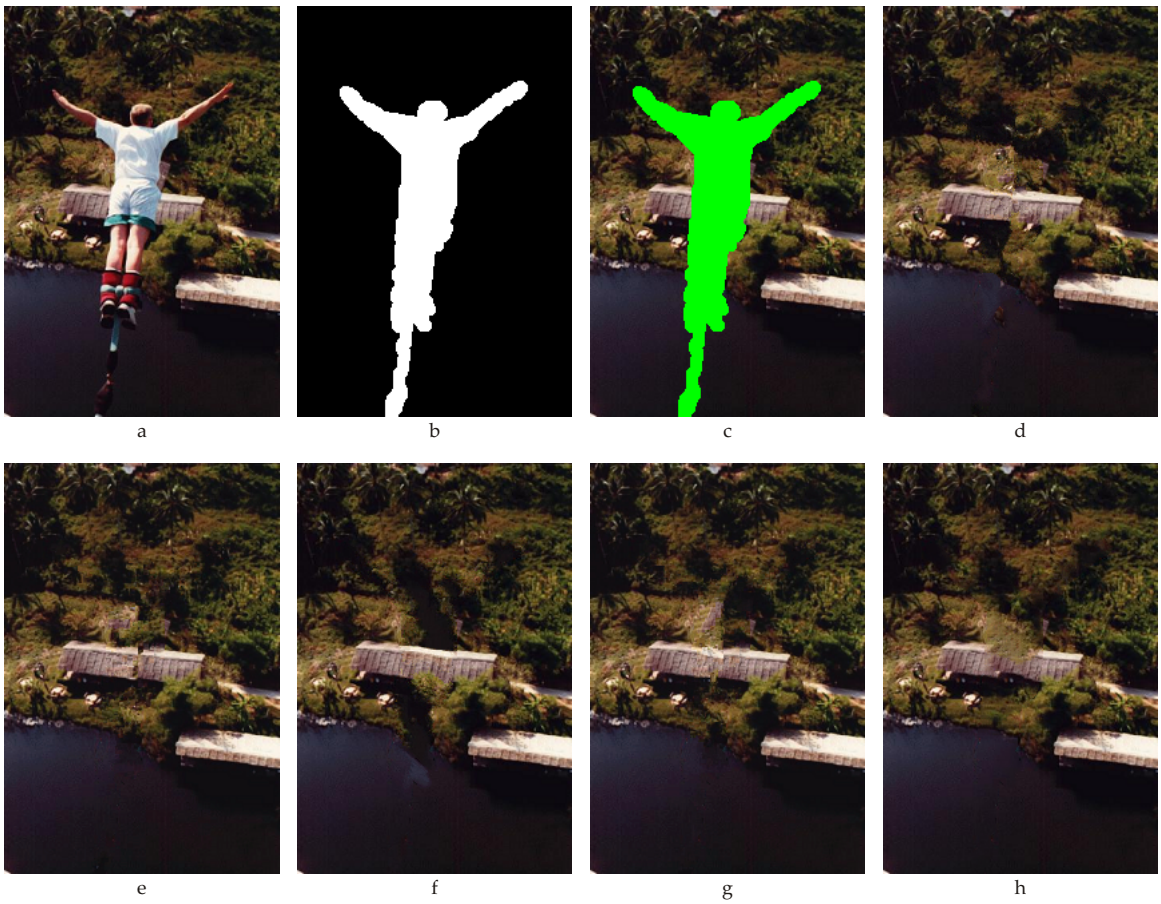


Figure 5.4: Test image "Bungee Jumper". (a) The original image of size 206×308. (b) The mask covering 13.86% of the image. (c) The input image with emphasized masked areas. (d) The output of [3], 65.754 sec. (e) The output of [11], size of the patch 7, 7.647 sec. (f) The output of [13], 21.585 sec. (g) The output of [25], size of the patch 7, 6.404 sec. (h) The output of [27, 4], size of the patch 5, 0.431 sec.

As can be seen, the gradual resizing process used in [27] leads to the loss of details as they get lost when the image is iteratively downsampled. See the fig. 5.4h. In contrary to the borderline between the ground and the river, which is reconstructed well within the missing part of the image, the roof part is synthesized in an inaccurate manner. This fact is the tradeoff between the outstanding speed of the algorithm and its accuracy.

With increasing size of the image, the time of computation of [3] (fig. 5.5d) becomes very unsatisfactory. Also, as greedy-based method, the method of [13], processing the missing pixels in scanline order, may get trapped into a loop when certain patch is repeatedly identified as the best exemplar (fig. 5.5f).

In highly textured images with many different features, such as in fig. 5.7, all methods

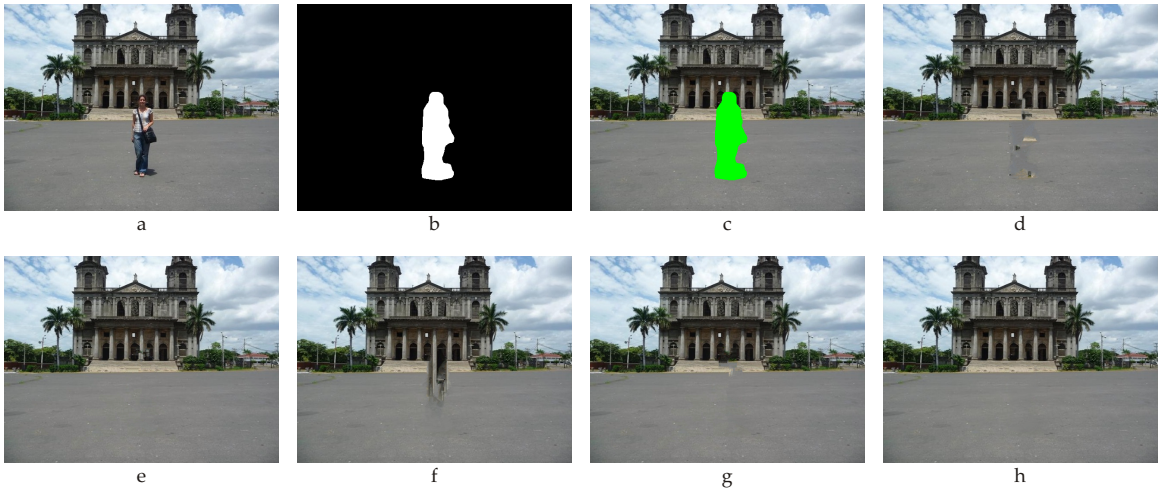


Figure 5.5: Test image "Palace". (a) The original image of size 638 × 478. (b) The mask covering 3.72% of the image. (c) The input image with emphasized masked areas. (d) The output of [3], 630.170 sec. (e) The output of [11], 25.343 sec. (f) The output of [13], 149.663 sec. (g) The output of [25], 31.336 sec. (h) The output of [27, 4], 2.328 sec.

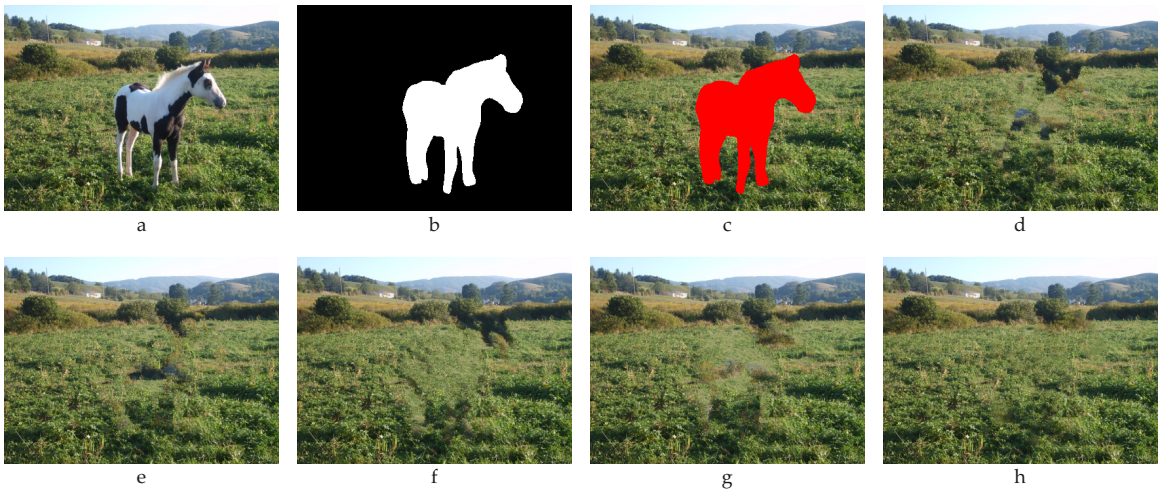


Figure 5.6: Test image "Horse". (a) The original image of size 420 × 315. (b) The mask covering 14.58% of the image. (c) The input image with emphasized masked areas. (d) The output of [3], 227.459 sec. (e) The output of [11], 25.442 sec. (f) The output of [13], 106.167 sec. (g) The output of [25], 19.176 sec. (h) The output of [27, 4], 1.044 sec.

give visually more or less unpleasant output.

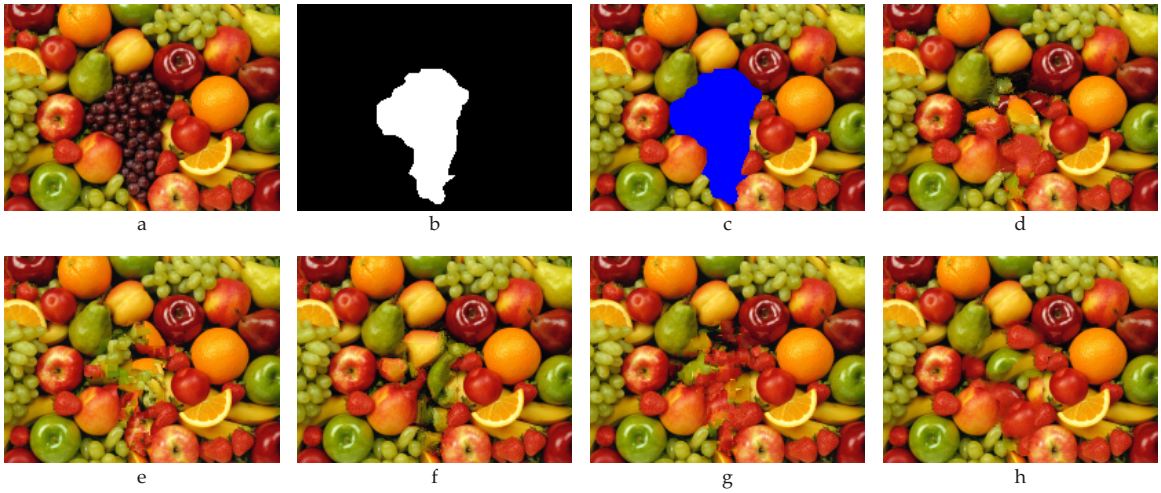


Figure 5.7: Test image "Fruits". (a) The original image of size 200×150. (b) The mask covering 12.55% of the image. (c) The input image with emphasized masked areas. (d) The output of [3], 12.316 sec. (e) The output of [11], 1.204 sec. (f) The output of [13], 3.905 sec. (g) The output of [25], 2.269 sec. (h) The output of [27, 4], 0.241 sec.

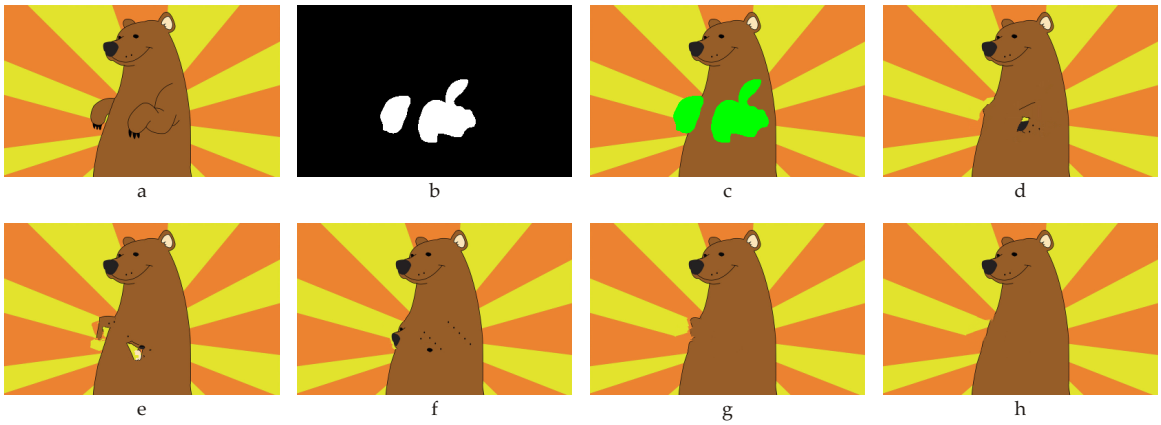


Figure 5.8: Test image "Bear". (a) The original image of size 400×250. (b) The mask covering 6.01% of the image. (c) The input image with emphasized masked areas. (d) The output of [3], 111.977 sec. (e) The output of [11], size of the patch 7, 6.215 sec. (f) The output of [13], 25.855 sec. (g) The output of [25], size of the patch 7, 6.083 sec. (h) The output of [27, 4], 0.721 sec.

5.3 Cartoon graphics

Cartoon graphics represents completely different type of images. With sharp borders between regions, more or less thick contours and uniformly colored image areas, human eyes are much more sensitive to the potential visual inconsistencies. The gradual resizing process may give

better results here, as shown in fig. 5.8, since it can better capture the local context of larger area.

5.4 PatchMatch

As obvious from the running times, the PatchMatch-supported algorithm proved to give good results and incomparable speed. Therefore, we will give some additional test and comparisons. One of the main advantages of the algorithm is the fact that its practically independent of the size of the unknown areas since the correspondences are computed for all image patches. However, the number of comparisons per patch is significantly smaller than in methods that utilize the exhaustive search over the whole set of exemplar candidates per every query patch, such as [13, 11, 3, 25].

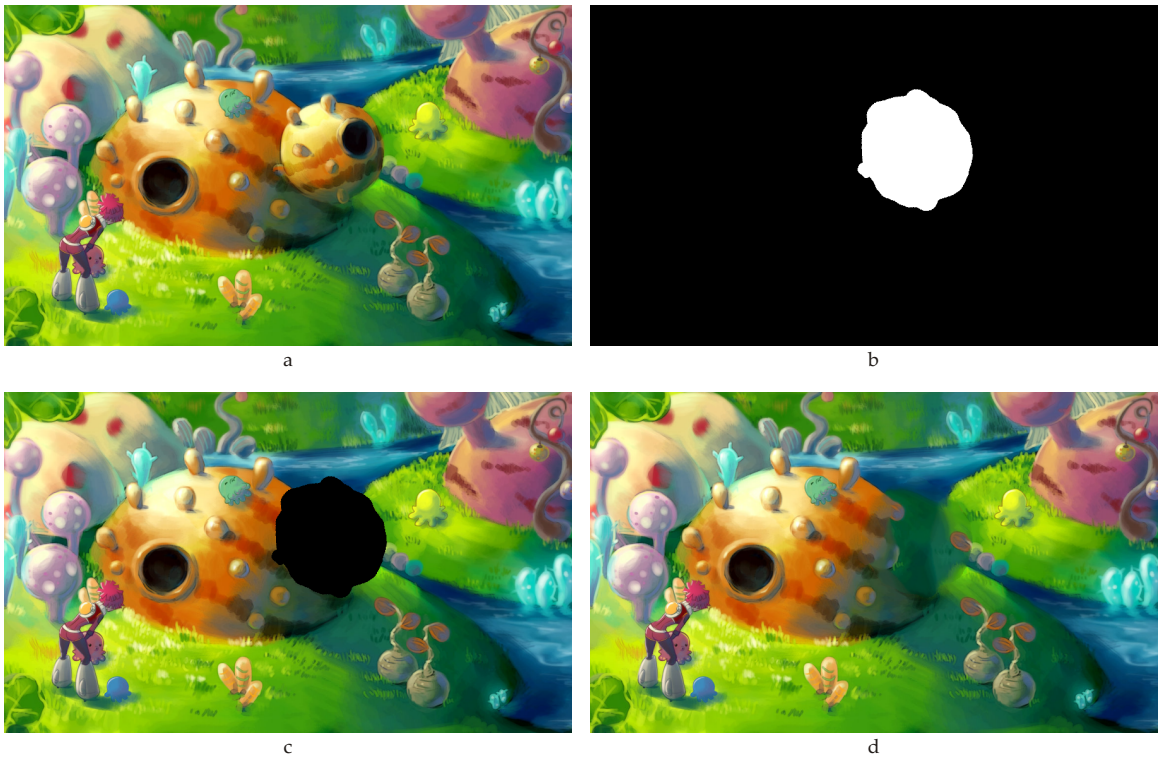


Figure 5.9: Test image "Fairy Tale". (a) The original image of size 900×540. (b) The mask covering 5.25% of the image. (c) The input image with emphasized masked areas. (d) The output of [27, 4], 3.774 sec.

5.5 Performance comparison

Given the comparison of the algorithms in terms of both performance speed and visual quality of the output, we now focus on some points to emphasize certain aspects of image

completion problem.

5.5.1 SSE instruction set

The use of SSE instruction set (more specifically instruction PSADBW) was described in sec. 2.3. Using it, we are able to process multiple pixels at once, although the used metric must be changed sum of absolute differences (SAD) instead of sum of squared differences (SSD). When the early termination is incorporated, usually more comparisons must be performed before the current candidate for optimum may be discarded as unpromising. Also, when the simple brute force method is used, the comparisons of masked pixels within the target patch can be simply skipped by checking a single condition. However, since the adjustment of once pre-prepared SSE data (containing information about all source patches) for every target patch would be inefficient, those pixels must be compared as well, even though they do not contribute to the error some with any value. Still, utilizing SSE provides the speed up as shown in fig. 5.10 and table 5.1.

	Size of the hole				
	20×15 (0.44%)	40×30 (1.77%)	80×60 (7.11%)	120×90 (16.00%)	160×120 (28.44%)
Brute force	3.586	13.738	50.469	97.898	152.919
Early termination	0.973	3.630	13.362	25.823	36.678
SSE brute force	0.966	3.667	13.076	26.506	42.162
SSE with early term.	0.667	2.612	9.723	19.354	29.614

Table 5.1: Performance of different methods of NN search.

5.5.2 Phase correlation

As mentioned in sec. 2.5, the phase correlation, used as method to search for nearest neighbors of given patches, is practically independent on the size of the patch window (the small set of best candidates must be probed in traditional manner). Although giving visually different results on incomplete patch search queries, it can be used when the size of the patch window grows over some value as shown in fig. 5.11.

5.6 Time performance/visual quality

5.6.1 Patch size

All the described and implemented methods (as well as most of the methods mentioned in sec. 1.2) share the patch size property as often one of the few adjustable parameters to control the image synthesis process. Therefore, we tested how this parameter visually influences the output image with respect to the processing time. On selected algorithms ([11], [13] and [27, 4]), the size of the patch was scaled from 5 to 13 pixels and the results are presented in fig. 5.12, 5.13 and 5.14 and summarized in tab. 5.2. As can be deduced, for methods that fill the whole patch in every iteration step, the growth of the patch size parameter actually

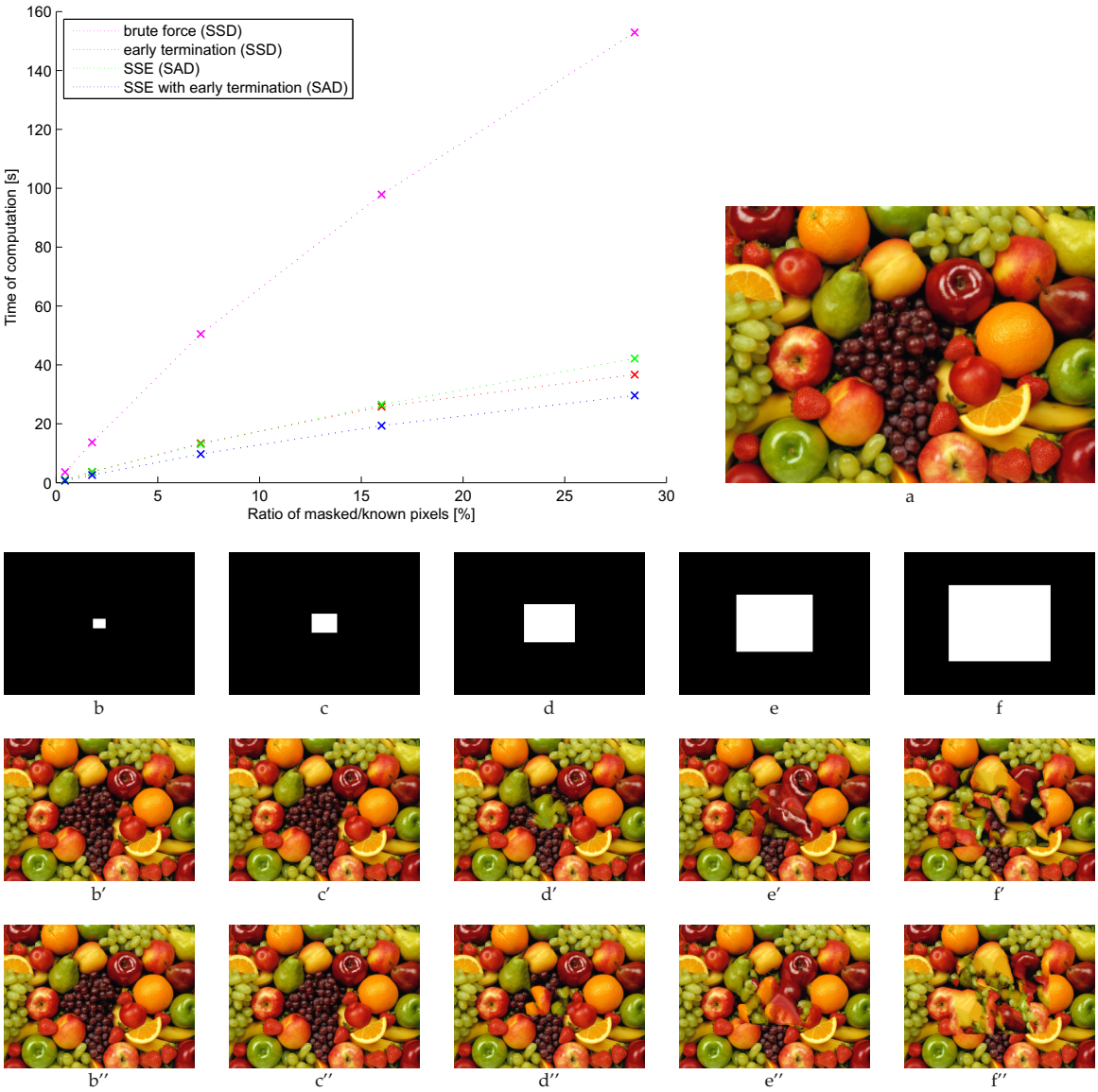


Figure 5.10: Utilizing SSE instruction set. The input image "Fruits" with size 300×225 is scanned with patch window of size 7 to complete missing areas of various sizes from 20×15 to 160×120. (a) The original image. (b,c,d,e,f) Masks of different sizes are placed into center of the image. (b',c',d',e',f') The results obtained using the SSD metric. (b",c",d",e",f"') The results obtained using the SAD metric (together with SSE instruction set). Use of early termination and/or SSE instruction set brings significant speedup in contrary to simple brute force approach.

reduces the computation time. However, more inconsistencies may occur when a bad patch is selected as the nearest neighbor. In contrary, pixel-synthesis-based methods show less tendency to introduce new unintended image edges since a single pixel is rarely a feature

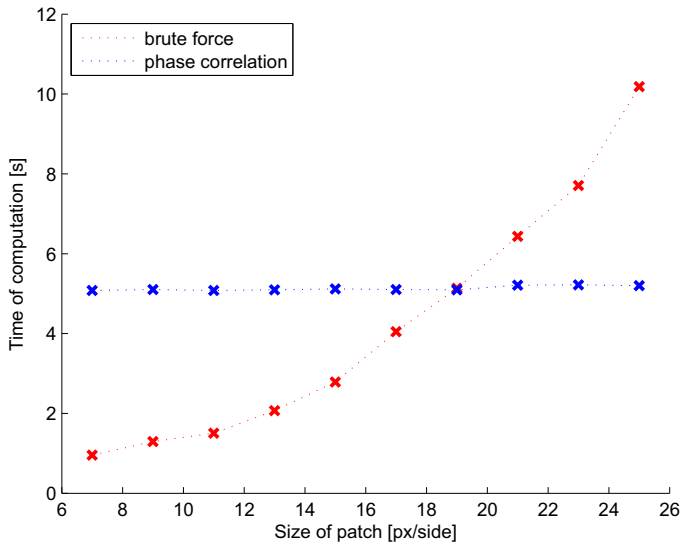


Figure 5.11: Comparison of exhaustive patch search vs. phase correlation based approach. The brute force (blue) uses also the early termination technique. While the computation time of phase correlation search remains stable with increasing patch size, the exhaustive search reflects this change of this parameter by longer computation time. The data was measured on fig. 5.10a using mask of fig. 5.10b.

itself. Nevertheless, the size of patch window dramatically increases the computation time of these methods, especially when a large image part has to be synthesized.

5.6.2 Constrained source region

As mentioned in 2.1.1, the size of the source zone does not always have to correspond to the whole rest of the image. In fact, the sufficient information is often presented within the imminent distance from the hole in a coherent pixel band. Depending of the size of this constrained area, the search of nearest neighbors can be sped up in varying degrees. Therefore, we tested several of the implemented algorithms to reflect this property. Since method of [27, 4] propagates the correspondences through the whole image, methods of [3], [11] and [13] were tested to examine this property. The result images are shown in fig. 5.15 and 5.16, respectively, and the time measurements can be found in tab. 5.3. All tests were run with patch size 7.

The results suggest that in most cases the constrained source zone gives approximately same results as searching over the whole image while the time of computation drops. Therefore, to probe only patches within the near distance from missing region suffices to synthesize the desired image.

5.6.3 Hierarchical approach

When image synthesis in larger image must be performed, or the missing area spans large portion of the input image, great amount of patch-to-patch comparisons must be computed.

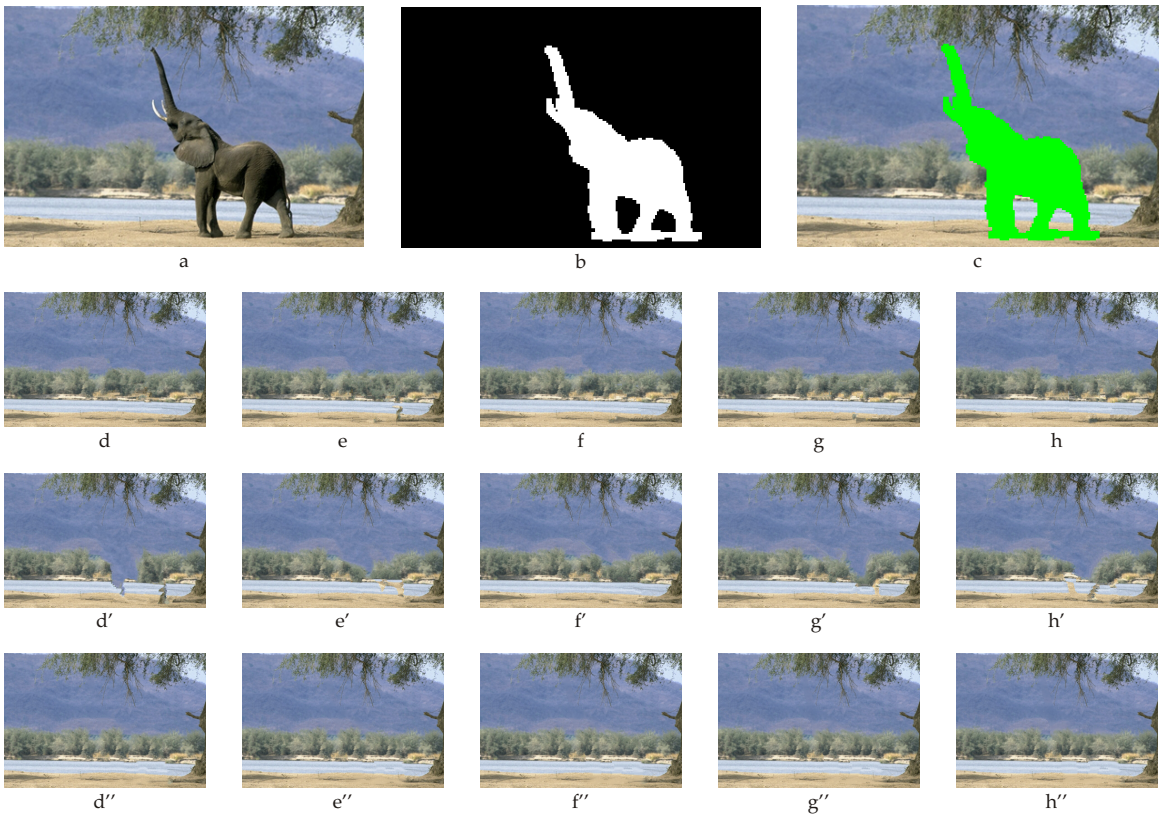


Figure 5.12: Test image"Elephant". (a) The original image of size 384×256. (b) The mask covering 13.27% of the image. (c) The input image with emphasized mask. (d,e,f,g,h) The output of [11] using patch of sizes $5 \ldots 13$. (d',e',f',g',h') The output of [13] using patch of sizes $5 \ldots 13$. (d",e",f",g",h") The output of [27, 4] using patch of sizes $5 \ldots 13$.

By using the brute force alone, the time of computation can easily rise up to few hundreds of seconds. A method that addresses this problem, and deals with it by resampling the image into lower resolution, is the hierarchical approach that makes use of creation of an image pyramid (see sec. 2.4). By downscaling the image in much lower resolution (preferably to half resolution at each step), the performed count of patch comparisons can be significantly lowered. However, the solution (more precisely the solution hint) propagated from the smallest resolution upwards might get trapped in the local minima and the final visual output may differ significantly. To observe and further evaluate the tradeoff between speed and visual quality, fig. 5.17, 5.18, 5.19 and 5.20 are presented along with comparison with results of [27, 4] that also utilize hierarchical approach.

5.7 Comparison with consumer applications

The probably most widespread mainstream image editing tool, the Adobe Photoshop, introduces in version CS5, as mentioned in sec. 1.3, new feature called *Content Aware*

Figure 5.13: Test image"Pumpkin". (a) The original image of size 473×334. (b) The mask covering 4.47% of the image. (c) The input image with emphasized mask. (d,e,f,g,h) The output of [11] using patch of sizes $5 \ldots 13$. (d',e',f',g',h') The output of [13] using patch of sizes $5 \ldots 13$. (d",e",f",g",h") The output of [27, 4] using patch of sizes $5 \ldots 13$.

Fill. Based on PatchMatch-supported [4] method of Wexler. et al. [29], it is able to quickly retouch the desired portion of the image. The utilization of PatchMatch algorithm reduces the computation time of [29] by several degrees of magnitude. Therefore, we would like to compare the results of implemented algorithms with the visual output of both the Adobe Photoshop CS5 and the original algorithm of Wexler et al. [29]. Since both algorithms are not implemented within the same framework, we can not give the precise time measurement but note that in practice, the time needed by the first mentioned method (Content Aware Fill) tends to be only few milliseconds, while the latter one [29] usually requires several tens of seconds. The comparison results are shown from fig. 5.21 to fig. 5.25.

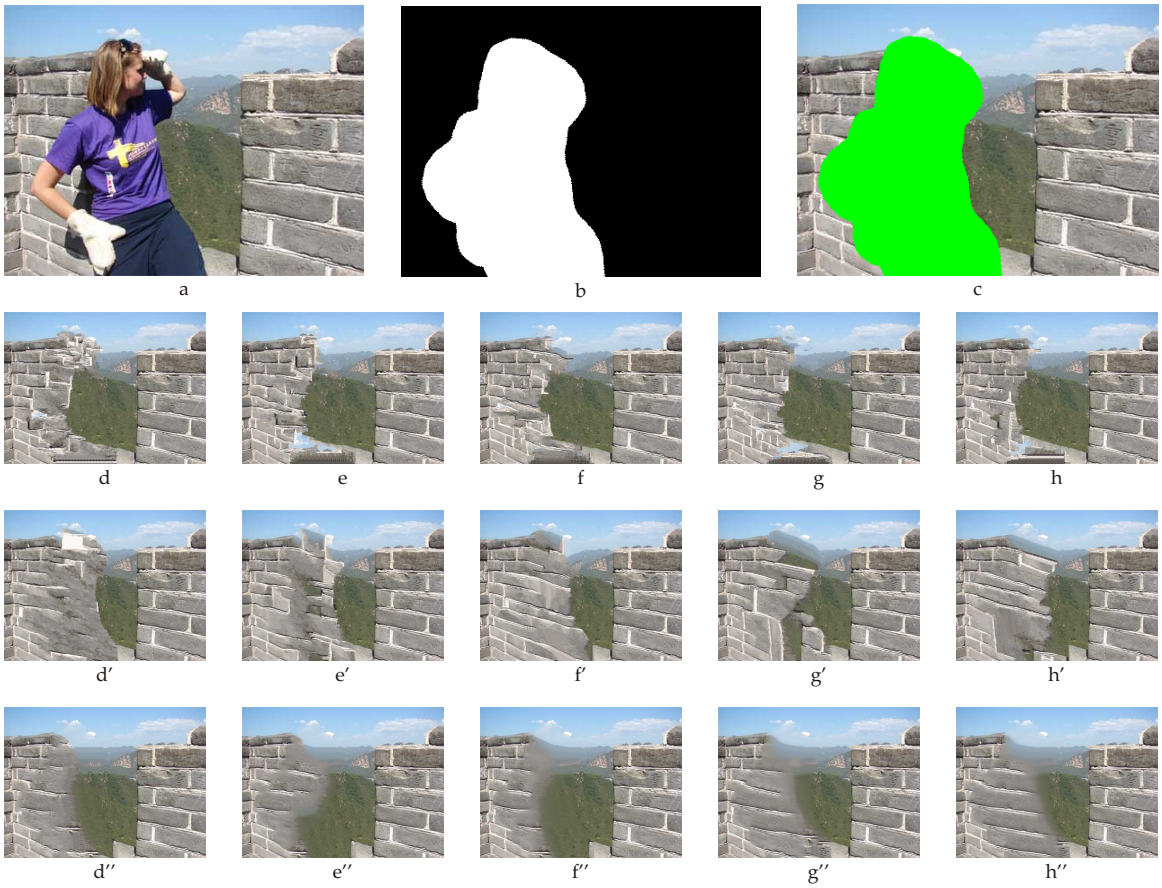


Figure 5.14: Test image"Battlements". (a) The original image of size 400×300. (b) The mask covering 4.47% of the image. (c) The input image with emphasized mask. (d,e,f,g,h) The output of [11] using patch of sizes 5 . . . 13. (d',e',f',g',h') The output of [13] using patch of sizes 5 . . . 13. (d",e",f",g",h") The output of [27, 4] using patch of sizes 5 . . . 13.

	Criminisi [11]				
	5	7	9	11	13
Elephant	19.053	14.716	13.762	13.603	13.453
Pumpkin	12.591	9.517	8.618	8.297	8.313
Battlements	61.042	47.706	41.567	37.008	35.317

	Efros [13]				
	5	7	9	11	13
Elephant	38.345	52.315	71.227	101.009	136.591
Pumpkin	36.376	47.119	59.386	77.029	97.570
Battlements	112.816	157.171	216.792	282.388	373.309

	Simakov/Barnes [27, 4]				
	5	7	9	11	13
Elephant	0.673	0.783	0.942	1.180	1.495
Pumpkin	0.988	1.105	1.257	1.447	1.723
Battlements	0.940	1.122	1.409	1.830	2.304

Table 5.2: Time performance of selected methods with changing patch size from 5 to 13. Images “Elephant”, “Pumpkin” and “Battlements” are shown in fig. 5.12, 5.13 and 5.14, respectively.

	Averbuch [3]					
	10	20	30	40	50	full
Sign	7.314	12.095	17.983	25.651	35.187	92.238
Crossroads	12.975	26.457	39.711	49.729	57.769	82.957

	Criminisi [11]					
	10	20	30	40	50	full
Sign	3.597	4.569	4.951	5.516	5.986	20.216
Crossroads	8.541	11.504	13.279	14.518	15.394	16.796

	Efros [13]					
	10	20	30	40	50	full
Sign	2.811	5.956	8.660	10.934	12.641	40.407
Crossroads	9.540	22.140	31.791	38.279	42.955	54.663

Table 5.3: Time performance of selected methods with changing size of source region. Images “Sign” and “Crossroads” are shown in fig. 5.15, 5.16 and 5.14, respectively.

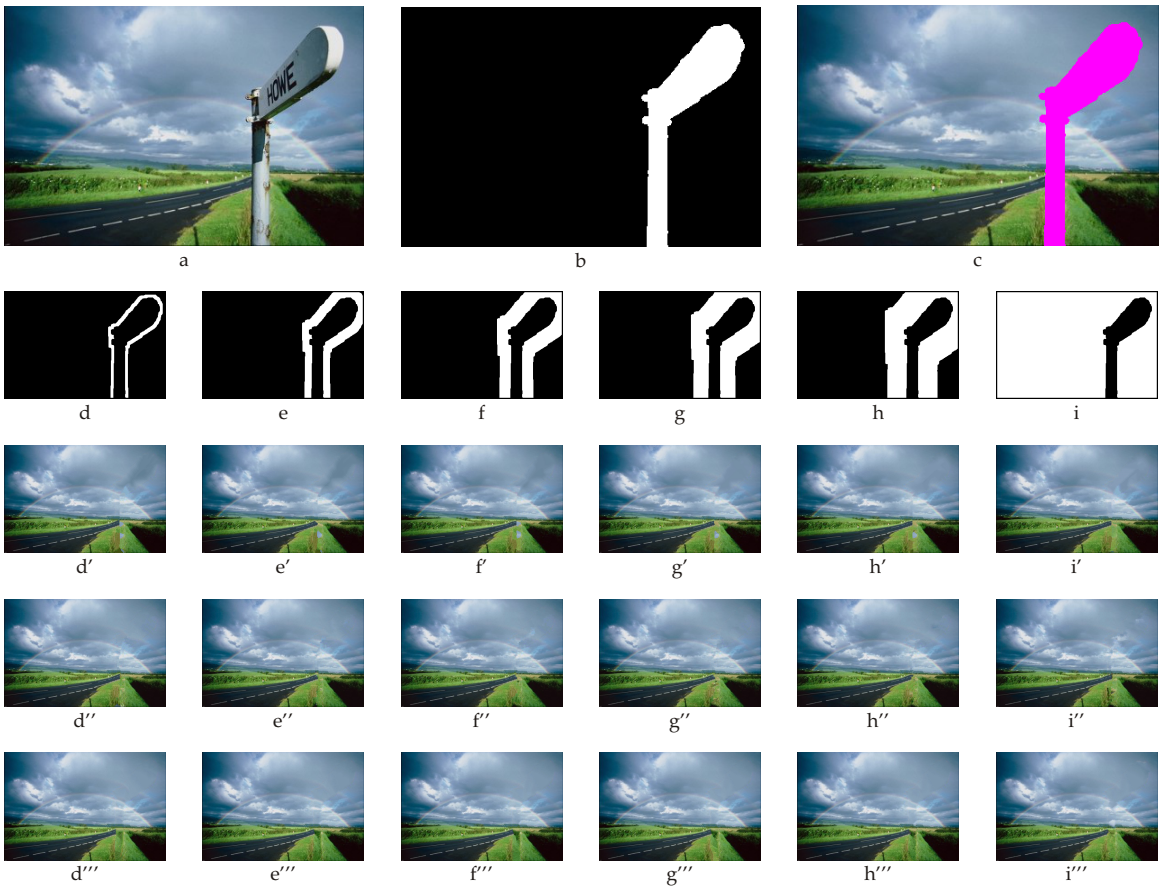


Figure 5.15: Test image"Sign". (a) The original image of size 400×266. (b) The mask covering 8.40% of the image. (c) The input image with emphasized mask. (d,e,f,g,h) The source region (containing only pixels with full patches) constructed as a dilated band of $n = (10, 20, 30, 40, 50)$ pixels. (i) The maximal possible source region (discarding only incomplete source patches). (d',e',f',g',h',i') The output of [3]. (d",e",f",g",h",i") The output of [11]. (d"',e"',f"',g"',h"',i"') The output of [13]. All results were obtained using patch size of 7.

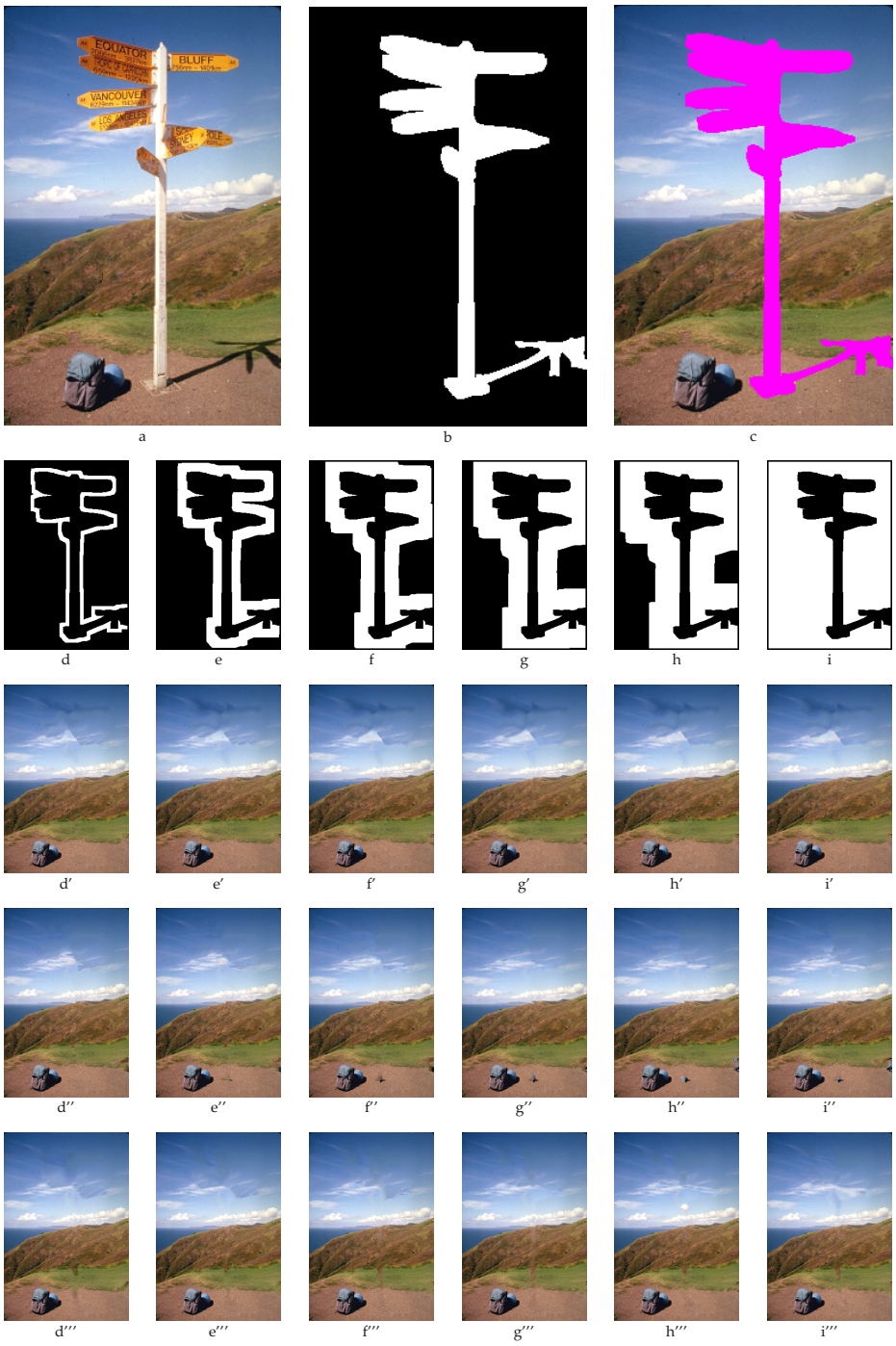


Figure 5.16: Test image"Sign". (a) The original image of size 243×368. (b) The mask covering 15.05% of the image. (c) The input image with emphasized mask. (d,e,f,g,h) The source region (containing only pixels with full patches) constructed as a dilated band of $n = (10, 20, 30, 40, 50)$ pixels. (i) The maximal possible source region (discarding only incomplete source patches). (d',e',f',g',h',i') The output of [3]. (d",e",f",g",h",i") The output of [11]. (d"',e"',f"',g"',h"',i"') The output of [13]. All results were obtained using patch size of 7.

Figure 5.17: Hierarchical approach, test image "Baby". (a) The original image of size 350×265. (b) The mask covering 26.01% of the image. (c) The input image with emphasized mask. (d) The result of exhaustive search over the original image, 77.066 sec. (e) Using 1 pyramid step, 19.759 sec. (f) Using 2 pyramid steps, 5.025 sec. (g) Using 3 pyramid steps, 3.760 sec. (h) The result of [27, 4], 0.854 sec.

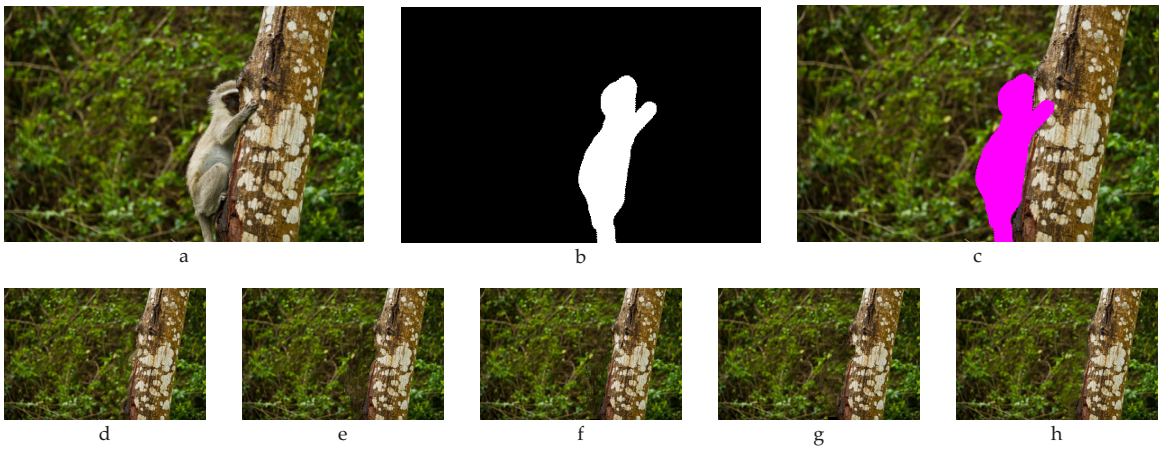


Figure 5.18: Hierarchical approach, test image "Monkey". (a) The original image of size 400×262. (b) The mask covering 7.33% of the image. (c) The input image with emphasized mask. (d) The result of exhaustive search over the original image, 32.686 sec. (e) Using 1 pyramid step, 7.449 sec. (f) Using 2 pyramid steps, 1.486 sec. (g) Using 3 pyramid steps, 1.132 sec. (h) The result of [27, 4], 0.771 sec.

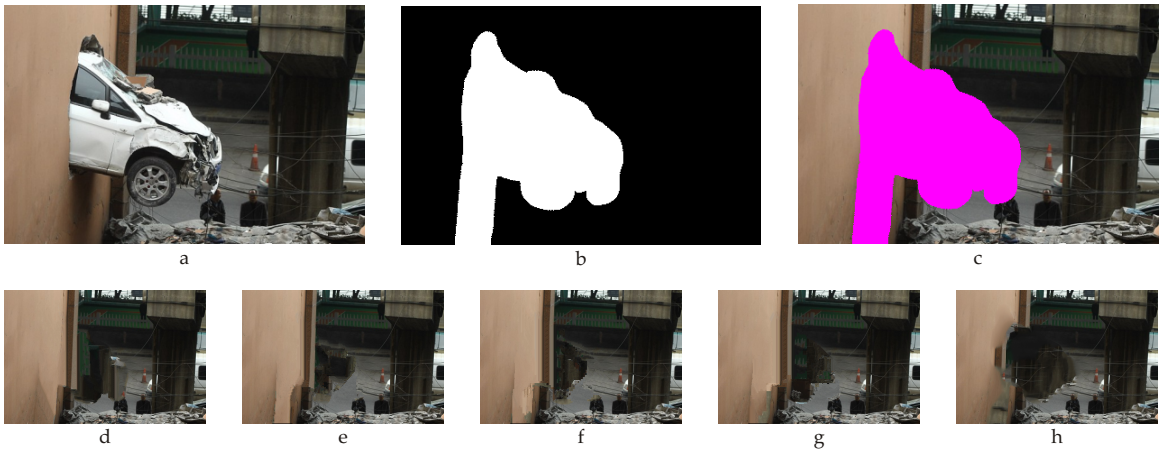


Figure 5.19: Hierarchical approach, test image "Car". (a) The original image of size 400×265. (b) The mask covering 23.78% of the image. (c) The input image with emphasized mask. (d) The result of exhaustive search over the original image, 81.405 sec. (e) Using 1 pyramid step, 17.148 sec. (f) Using 2 pyramid steps, 4.421 sec. (g) Using 3 pyramid steps, 3.588 sec. (h) The result of [27, 4], 0.921 sec.

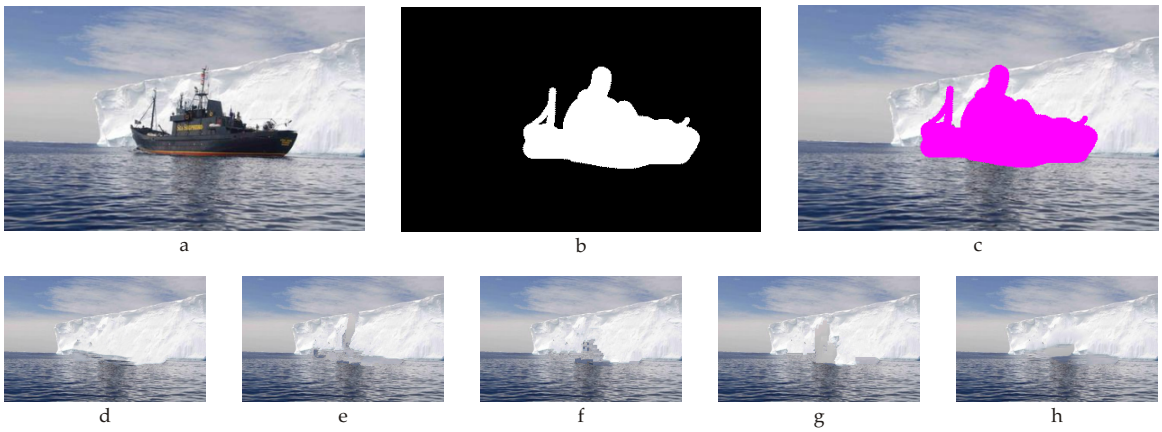


Figure 5.20: Hierarchical approach, test image "Ship". (a) The original image of size 400 × 249. (b) The mask covering 11.95% of the image. (c) The input image with emphasized mask. (d) The result of exhaustive search over the original image, 49.031 sec. (e) Using 1 pyramid step, 11.446 sec. (f) Using 2 pyramid steps, 2.613 sec. (g) Using 3 pyramid steps, 1.694 sec. (h) The result of [27, 4], 0.799 sec.

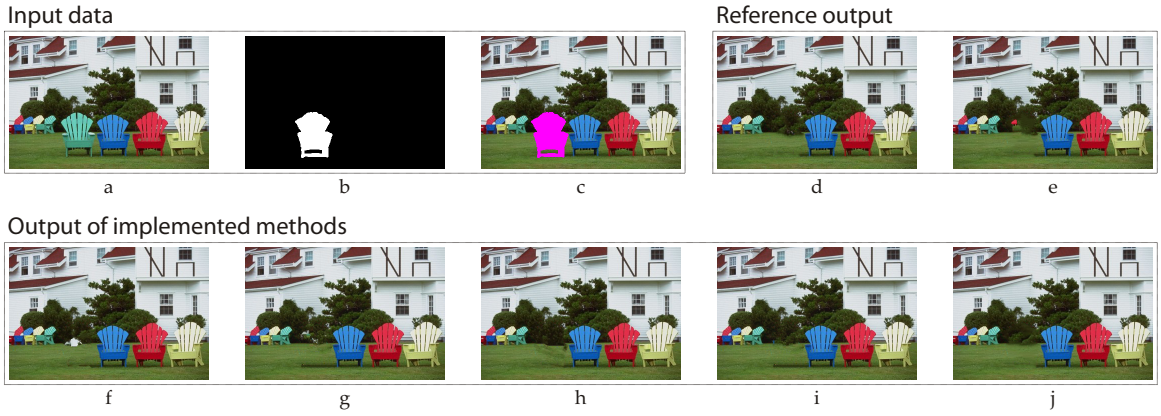


Figure 5.21: Comparison with reference output, test image "Chairs". (a) The original image of size 400 × 266. (b) The mask covering 4.60% of the image. (c) The input image with emphasized mask. (d) The result from Photoshop. (e) The result of [29]. (f) The result of [3]. (g) The result of [11]. (h) The result of [13]. (i) The result of [25]. (j) The result of [27, 4].

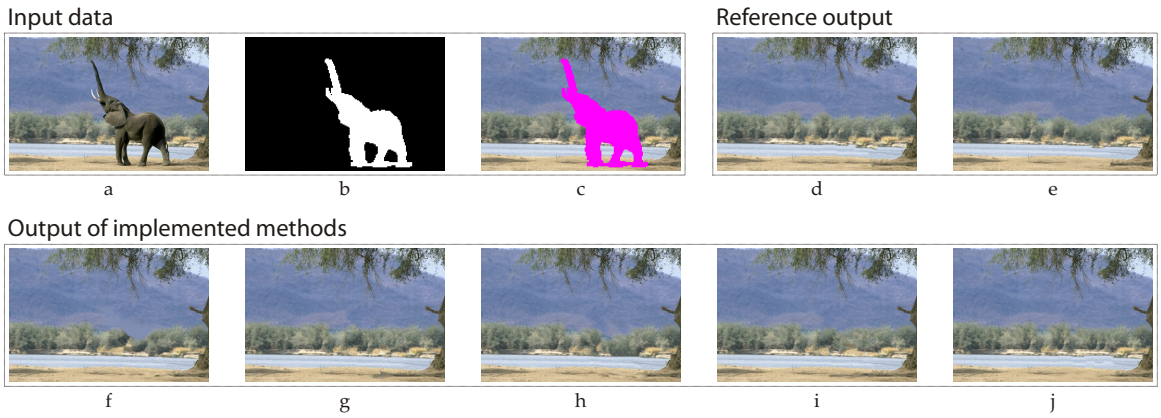


Figure 5.22: Comparison with reference output, test image "Elephant". (a) The original image of size 384 × 256. (b) The mask covering 13.27% of the image. (c) The input image with emphasized mask. (d) The result from Photoshop. (e) The result of [29]. (f) The result of [3]. (g) The result of [11]. (h) The result of [13]. (i) The result of [25]. (j) The result of [27, 4].

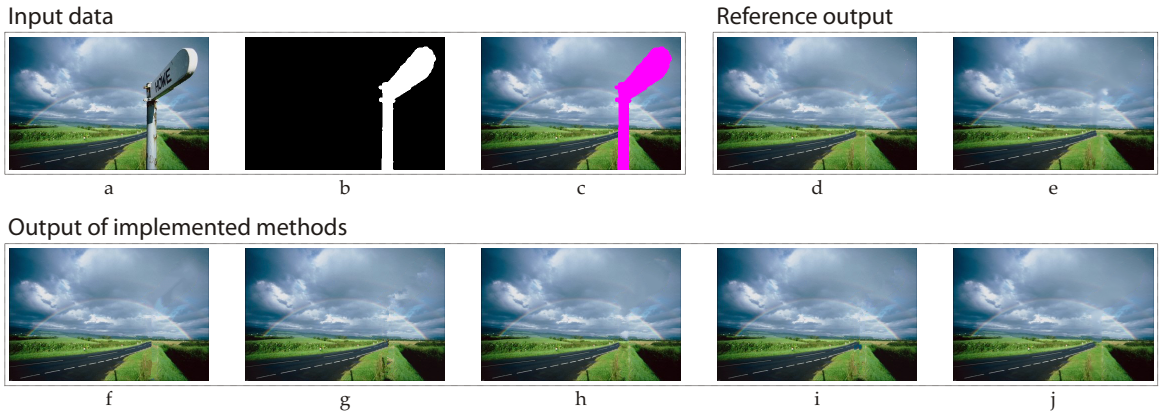


Figure 5.23: Comparison with reference output, test image "Sign". (a) The original image of size 400 × 266. (b) The mask covering 8.40% of the image. (c) The input image with emphasized mask. (d) The result from Photoshop. (e) The result of [29]. (f) The result of [3]. (g) The result of [11]. (h) The result of [13]. (i) The result of [25]. (j) The result of [27, 4].

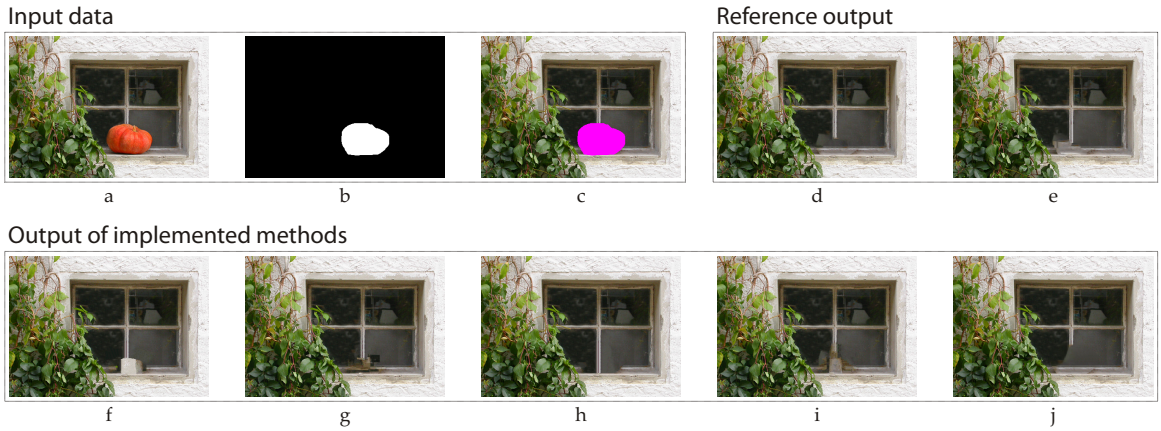


Figure 5.24: Comparison with reference output, test image "Pumpkin". (a) The original image of size 473 × 334. (b) The mask covering 4.47% of the image. (c) The input image with emphasized mask. (d) The result from Photoshop. (e) The result of [29]. (f) The result of [3]. (g) The result of [11]. (h) The result of [13]. (i) The result of [25]. (j) The result of [27, 4].

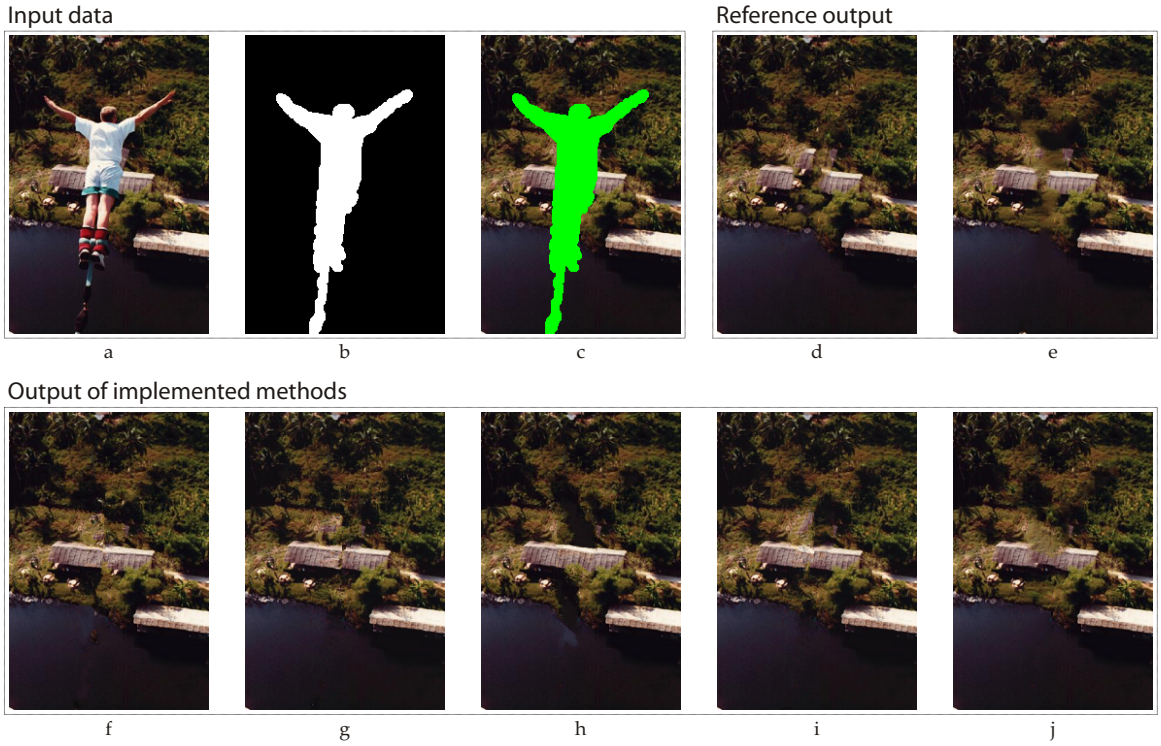


Figure 5.25: Comparison with reference output, test image "Bungee Jumper". (a) The original image of size 206×308. (b) The mask covering 13.86% of the image. (c) The input image with emphasized mask. (d) The result from Photoshop. (e) The result of [29]. (f) The result of [3]. (g) The result of [11]. (h) The result of [13]. (i) The result of [25]. (j) The result of [27, 4].

Chapter 6

Conclusion

The goal of this thesis was to examine, describe and implement several image inpainting algorithms constituted on examplar-based approach using the rest of the image as the source of information to perform given task. The brief introduction was given with emphasis on both state-of-the-art methods and examples of practical usage. Also, the short theoretic introduction into the exemplar-based synthesis was presented.

The problem of finding nearest neighbors of given patches was described in more detail in chap. 2 since the problem of finding nearest neighbors plays one of the key roles in exemplar-based image completion area. Also, the description of selected image completion methods was given in chap. 3.

In the second part of this thesis, the implementation of selected methods was proposed, providing the unified framework to reuse the parts which the implemented algorithms share in common, thus making it easy to possibly add more image inpainting algorithms in the future. The algorithms of [3], [11], [13], [25] and [27, 4] were implemented in C++ language using the features of the advanced image processing library OpenCV [9].

Finally, the implemented algorithms and selected NN-search methods were tested and compared in terms of time complexity and visual quality of the output. Furthermore, the comparison with currently well-established consumer image editing software, the Adobe Photoshop, and its underlying algorithm [29] was given to compare the quality of output images.

6.1 Future work

More methods targeting the image completion problem could be included into the implemented framework in future, making use of pre-prepared structures shared also by majority of already implemented algorithms. Also, porting the presented methods to GPU could be a important task for future development, since most of them can be parallelized. Lastly, a simple graphical user interface could be created to simplify the manipulation with the image as well as bring the option to step the algorithms and see the "live" updates, as the missing portion of the image is being synthesized. This feature could also serve as an instructional or educative tool to explain the mechanism of implemented algorithms.

Bibliography

[1] Andrew Adams. ImageStack - a command-line stack calculator for images. <http://code.google.com/p/imagestack/>, 2012. Accessed: 01/05/2012.

[2] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. Pyramid methods in image processing. *RCA Engineer*, 29(6):33–41, 1984.

[3] A. Averbuch, G. Gelles, and A. Schclar. Fast hole-filling in images via fast comparison of incomplete patches. In *Proceedings of the International conference on Multimedia Content Representation, Classification and Security*, pages 738–744, 2006.

[4] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics*, 28(3):24, 2009.

[5] Connelly Barnes, Eli Shechtman, Dan B. Goldman, and Adam Finkelstein. The generalized patchmatch correspondence algorithm. In *Proceedings of the 11th European Conference on Computer Vision*, pages 29–43, 2010.

[6] M. Bertalmio, A. L. Bertozzi, and G. Sapiro. Navier-stokes, fluid dynamics, and image and video inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 355–362, 2001.

[7] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424, 2000.

[8] Ronald N. Bracewell. *The Fourier transform and its applications*. 2d ed. edition, 1978.

[9] G. Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[10] Tony F. Chan and Jianhong Shen. Non-texture inpainting by curvature-driven diffusions (cdd). *Journal of Visual Communication and Image Representation*, 12: 436–449, 2001.

[11] A. Criminisi, P. Perez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing*, 13(9):1200–1212, 2004.

[12] Iddo Drori, Daniel Cohen-Or, and Hezy Yeshurun. Fragment-based image completion. *ACM Transactions on Graphics*, 22(3):303–312, 2003.

[13] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the International Conference on Computer Vision*, pages 1033–1038, 1999.

[14] Todor Georgiev. Photoshop healing brush: a tool for seamless cloning, 2004.

[15] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[16] Paul Harrison. *Image Texture Tools*. PhD thesis, Monash University, 2005.

[17] James Hays and Alexei A. Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics*, 26(3):4, 2007.

[18] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 229–238, 1995.

[19] Pavel Holoborodko. Noise robust gradient operators. <`http://www.holoborodko.com/pavel/image-processing/edge-detection/`>, 2009.

[20] Chiou-Ting Hsu and Ja-Ling Wu. Hidden digital watermarks in images. *IEEE Transactions on Image Processing*, 8(1):58–68, 1999.

[21] IEEE. IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–58, 2008.

[22] Charles E. Jacobs, Adam Finkelstein, and David H. Salesin. Fast multiresolution image querying. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 277–286, 1995.

[23] James M. Kasson and Wil Plouffe. An analysis of selected computer interchange color spaces. *ACM Transactions on Graphics*, 11(4):373–405, 1992.

[24] Nikos Komodakis. Image completion using global optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 442–452, 2006.

[25] Tsz-Ho Kwok, Hoi Sheung, and Charlie C. L. Wang. Fast query for exemplar-based image completion. *IEEE Transactions on Image Processing*, 19(12):3106–3115, 2010.

[26] David Mount and Sunil Arya. ANN: A library for approximate nearest neighbor searching. 1997.

[27] Denis Simakov, Yaron Caspi, Eli Shechtman, and Michal Irani. Summarizing visual data using bidirectional similarity. 2008.

[28] Huang Ting, Shifeng Chen, Jianzhuang Liu, and Xiaoou Tang. Image inpainting by global structure and texture propagation. In *Proceedings of the 15th international conference on Multimedia*, pages 517–520, 2007.

[29] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3):463–476, 2007.

[30] Chunxia Xiao, Meng Liu, Nie Yongwei, and Zhao Dong. Fast exact nearest patch matching for patch-based image editing and processing. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1122–1134, 2011.

[31] Yingzhen Yang, Yin Zhu, and Qunsheng Peng. Image completion using structural priority belief propagation. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 717–720, 2009.

[32] A. Zalesny, V. Ferrari, G. Caenen, and L. Van Gool. Parallel composite texture synthesis. In *Texture 2002 Workshop in conjunction with ECCV 2002*, pages 151–155, 2002.

Appendix A

Note on color spaces

Comparing grayscale images, using on of the metrics mentioned before, is an easy job, since the pixel is represented by only a single value. However, extending the comparison into non-monochromatic color spaces brings new questions. In most color spaces the computer images are represented as triplet of values.

The most common color space to store color images is the RGB (or BGR) model. Although this model is based on model of human eye and its possession of three types of cone cells (trichromacy), measuring the distances between two colors from RGB space can produce ambiguous results. Consider this simple example. Lets have three RGB colors with 8-bit per channel: $a = (135, 20, 20)$, $b = (255, 0, 65)$ and $c = (122, 149, 20)$. The SSD difference between colors a and b is 16825 and the difference between colors a and c is 16810, thus they should be slightly more similar one to each other than a to b. However, as shown in fig. A.1, the first two couple appears to be much more visually coherent to human eyes.

Figure A.1: Example of misleading interpretation of color distance in RGB color space.

Therefore, as suggested in number of image-completion-related works, CIE Lab [23, 11, 25] or YUV [3] color space are more meaningful choice, since these models take the human perception of color more into account and separate the information about lightness from the color. For example, in Lab color space the colors a and b would have distance 1965, while between colors a and c the distance would be 6131.

Appendix B

List of used abbreviations

ANN Approximate Nearest Neighbor

CIE Commission internationale de l'éclairage (International Commission on Illumination)

DCT Discrete Cosine Transform

FFT Fast Fourier Transform

GIMP GNU Image Manipulation Program

GNU GNU's Not Unix!

LDV Large Displacement View

MRF Markov Random Field

NNF Nearest Neighbor Field

PCA Principal Component Analysis

PDE Partial Differential Equation

SAD Sum of Absolute Differences

SIMD Single Instruction, Multiple Data

SSD Sum of Squared Differences

SSE Streaming SIMD Extensions

Appendix C

Installation manual

The thesis was programmed on Microsoft Windows 7 64-bit using Microsoft Visual Studio 2010 with OpenCV 2.3 and FFTW libraries. In order to be able to run the code, the followings requirements must be met:

1. The OpenCV library must be present on the host computer. It can be downloaded from <`http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.4.0/`>. Also download the last release of FFTW library from <`http://www.fftw.org/download.html`>.

2. The paths in Visual Studio must be correctly set. Go to *Project* > *Properties* > *Configuration Properties* > *Linker* > *General* and change the *Additional Library Directories* input field to the correct value.

3. Go to *Project* > *Properties* > *Configuration Properties* > *Linker* > *Input* and check that includes for OpenCV and FFTW are set.

4. Go to *Project* > *Properties* > *Configuration Properties* > *C++* > *General* and change the directories to correct value.

5. Make sure that the aforementioned changes are made both in *Debug* and *Release* modes.

Appendix D

Content of the accompanying CD

```
CD
 |-- demo                        - Executable dmonstration binaries
 |-- doc                         - Doxygen documentation
 |-- img                         - Test images
 |   \--results                  - Result images presented
 |                                 in the chapter "Results"
 |-- src                         - Source files of the implementation
 |-- text
 |   |--latex                    - Source files of this thesis' text
 |   \--thesis.pdf               - PDF version of this thesis' text
 \-- README.txt                  - The short resume information
```