Utrecht University

Master Thesis

Image Inpainting

Author:
Sander van de Ven

Student Number:
3369137

Supervisor:
Dr. Robby T. Tan

Thesis Number:
ICA-3369137

May 18, 2012

Abstract

Image inpainting is the process of filling an unknown region of an image with visual plausible information. Filling this region with information that could have been in the image. There is nothing like the 'perfect' inpainting algorithm, each method has it own advantages and drawbacks. In this thesis we want to investigate if we can improve both the speed and the visual appeal of two popular algorithms. First we look at the Bertalmio method, second at the Criminisi method. The two algorithms were implemented and analyzed by looking at the advantages and drawbacks of these methods. From these drawbacks we came up with contributions to increase the visual appeal or the speed of these algorithms.

For the Bertalmio paper three contribution are found, these are using multi-resolution images, inpainting the unknown region inwards and estimating the amount of necessary iterations of the algorithm. These contributions all increase the speed of the algorithm. Three contributions are also found for the Criminisi paper. These are local searching which increases the speed but should decrease the visual appeal, although in the experiments it increased the visual quality. Patch estimation increases the quality of the resulting images at the cost of a small speed decrease. The use of a lookup data structure significantly increases the speed of the algorithm at the cost of a small quality drop.

Observations showed that the current numerical quality estimation is not always ideal. Therefore an user study is conducted to investigate if this quantitative estimation matches the quality of the images as perceived by human beings. This user study showed that three observation were verified.

The main conclusions of the project are that contributions were found of two important inpainting algorithms and the observations from the project were verified by the user study.

Contents

Chapter 1

Introduction

As early as the Renaissance, people try to restore damaged paintings in a way that the paintings will properly look like the original for an observer who is unfamiliar with the original. This process is called restoration, conservation, inpainting or retouching. This is done to preserve the paintings and other fine art for future generations. An example of a restored painting and frame is shown in Figure 1.1.

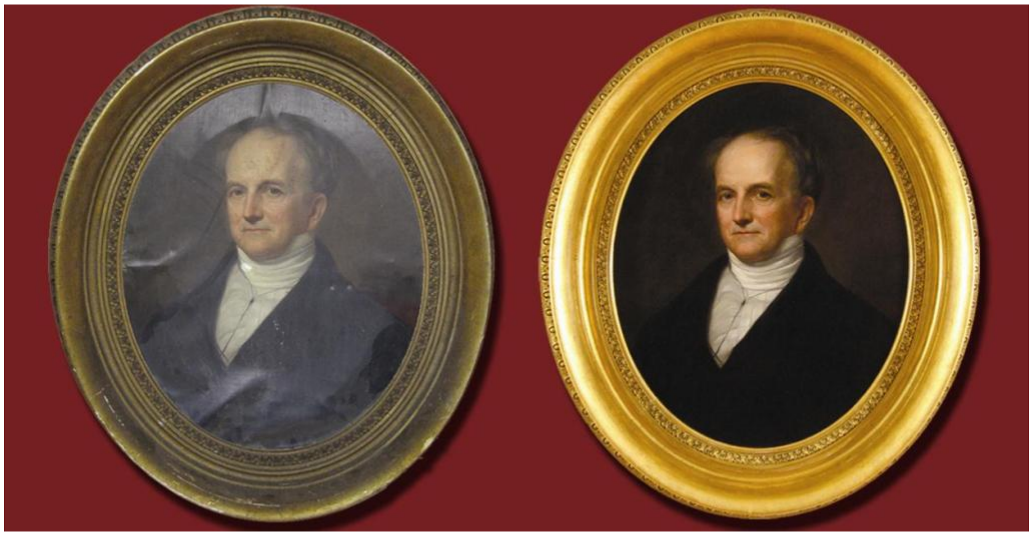


Figure 1.1: A restored painting and frame, photo by Oliver Brothers Fine Art Restoration.

In image processing and computer vision we try to do the same with digital images, this is called image completion, image disocclusion or (digital) image inpainting. The goal of image inpainting is filling the target region in the image with visual plausible information. Filling this region with information that could have been in the image. There are several applications for image inpainting, one of these is the restoration of old images and movies by removing cracks from these images and movies. The removal of objects from images, like time stamps or a person. Another application can be image inpainting as the pre-process of other computer vision or image processing applications, an example of this is the use of image inpainting in image-based material editing. It can also be used to fill missing parts in image communication, for example image compression, lost packets retrieval and zooming. An example of image inpainting is shown in Figure 1.2.

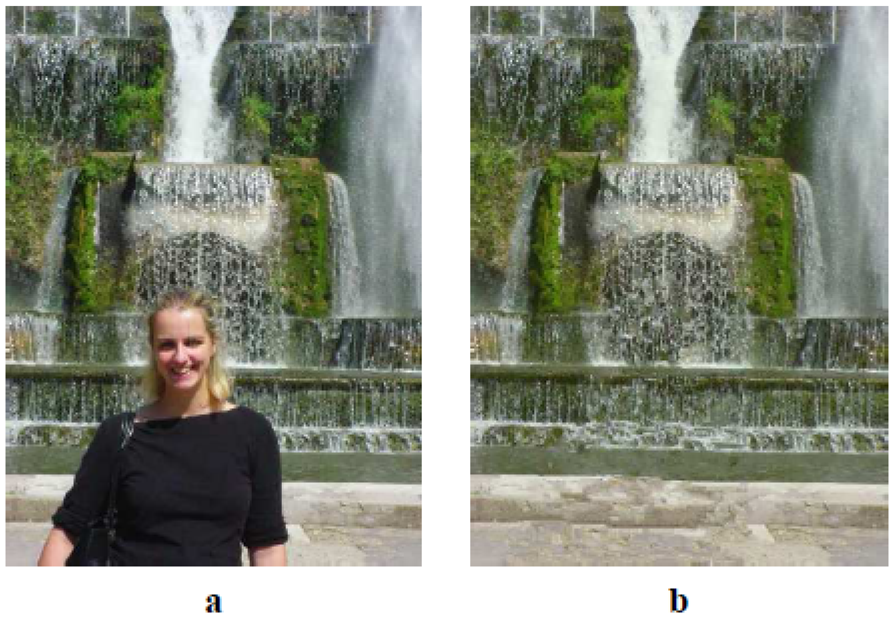


Figure 1.2: Example of image inpainting from [1], the original image is shown on the left and the inpainted image on the right.

This field consists of three main topics: the restoration of images by texture synthesis, restoration of pictures by propagating linear structures and the restoration of films. This master thesis primarily focuses on the second technique: the restoration of images by propagating linear structures.

In this thesis we want to investigate if we can improve both the speed and the visual appeal of two popular algorithms. We also investigate if the current estimation method of the visual appeal of the result of image inpainting matches the visual appeal as defined by humans. We look at two popular methods and see if we can improve them by implementing the algorithms, looking at the advantages and drawbacks and making contributions based on these advantages and drawbacks. First we look at the Bertalmio[2] method, second at the Criminisi[1] method. We also want to see how good an estimation method for the quality of image inpainting works. This is done by comparing the results of estimation method with a user study and verifying some observations made throughout the project.

The main challenges in image inpainting are the continuation of structure, finding the correct information and the speed of inpainting. An example of the continuation challenge is an image with a fence; when a user wants to inpaint an unknown part of a fence, the fence should continue. In inpainting algorithms we have to find the perfect information to fill in a gap. That information can come from multiple places, adjacent pixels, other parts of the image or other images, it can be challenging to find the best match for each situation. Current research of image inpainting techniques consists of some different approaches. Some approaches are based on pixel by pixel updating by computing partial differential equations. Others are based on copying parts of the input image to the unknown region or even information from a dataset of images.

The thesis starts with related work in chapter 2, followed by the analysis of the image inpainting method of Bertalmio *et al.*[2] in chapter 3. In chapter 4 the analysis of the method, exemplar based image inpainting by Criminisi *et al.*[1] is described. An analysis of quality estimation of image inpainting is shown in chapter 5 and the conclusion and future work follow in chapter 6.

Chapter 2

Related Work

Image and video completion is discussed in a lot of papers and different approaches are used for this. This chapter describes important methods of image and video completion and some examples are shown.

2.1 Image inpainting techniques

Bertalmio *et al.*[2] introduce the term image inpainting to computer science. In the algorithm the region that has to be inpainted will be filled-in by information of the region surrounding the gap. The curves of equal intensity (isophotes) arriving at the boundary are propagated inwards. Because this is the first and one of the most important papers in image inpainting, we will look more closely at it in chapter 3.

Criminisi *et al.*[1] propose an algorithm inspired by the algorithm by Bertalmio *et al.*[2] and texture synthesis [3]. In contrast to the paper of Bertalmio et al. the unknown region is inpainted patch by patch. The sequence of which patch should be inpainted is based on the isophote information and the amount of known information surrounding the patch. This popular method increased the result of image inpainting at lot. Because of that this method is further explained in chapter 4.

In the algorithm of Oliveira *et al.*[4], the unknown region of the image is convolved with a Gaussian kernel. To prevent edges to be blurred the user manually specifies barriers for the diffusion. This algorithm is much faster than the other algorithms but it can only be used with very small unknown regions and is less accurate. An example of this algorithm is shown in figure

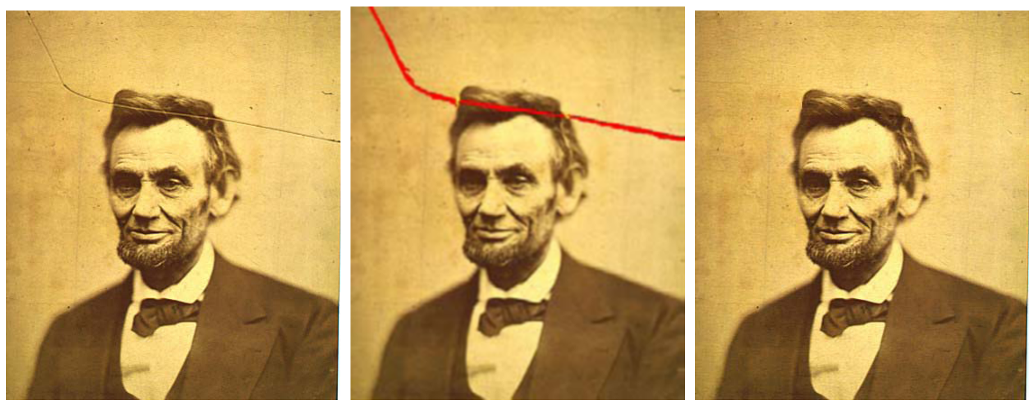


Figure 2.1: Olivera method [4], input, mask and result, note the diffusion barriers near the boundaries of the hair of Abraham Lincoln.

2.1.

The algorithm of Perez *et al.*[5] showed how gradient domain reconstruction can be used in image editing applications. The actual pixel values for the unknown pixel values are computed by solving a Poisson equation that locally matches the gradients while obeying the fixed Dirichlet (exact matching) conditions at the seam boundary. Poisson Image Editing can be used best for seamless inserting and local illumination changes but can also be used for image inpainting.

The author of [6] proposed a new inpainting algorithm based on propagating an image smoothness estimator along the image gradient. Similar to the algorithm of Bertalmio *et al.*[2]. The image smoothness is estimated as the weighted average of the known image neighborhood of the pixel to inpaint. The fast marching method (FMM) is used to create a distance function to the initial boundary. The pixels of the unknown region are inpainted in the order of the distance to the boundary, proceeding from the smallest to the largest. This method is fast but creates blurry effects with larger unknown regions. An example of this algorithm is shown in Figure 2.2.

2.2 Texture syntheses

Texture synthesis is the process of algorithmically constructing a large digital image from a small digital sample image by taking advantage of its structural content [3][7]. Texture synthesis can be also used to fill in unknown regions

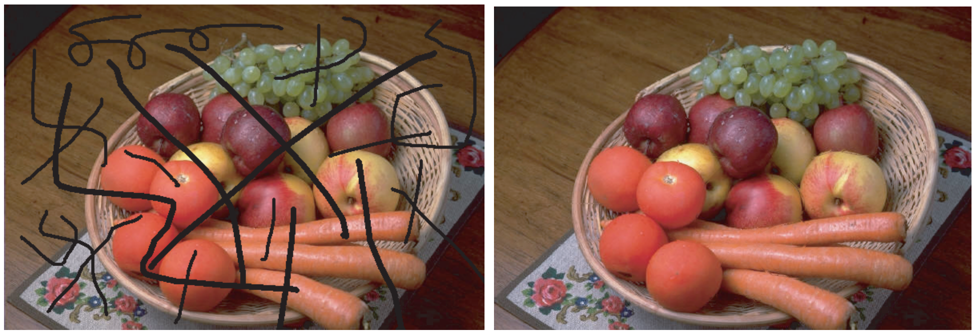


Figure 2.2: Telea method [6], input plus mask and result.

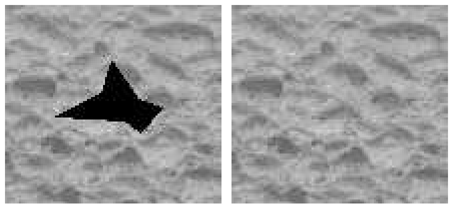


Figure 2.3: Texture Syntheses by Efros *et al.*[3], input plus mask and result.

in images, the algorithm of Criminisi et al.[1] is partially based on these algorithms. An example of the algorithm by Efros *et al.*[3] is shown in Figure 2.3

2.3 Guided method

Sun *et al.*[8] introduces a new direction in image inpainting. In the authors algorithm the user is also able to specify support lines. These support lines specify where important lines in an image should be continued. This algorithm first fills in the unknown information around the support lines by dynamic programming or belief propagation, depending on the structure of the support lines. After that the rest of the image is filled in by texture propagation. The result of this algorithm is good but we can not compare these with others because this algorithm requires extra input. An example of this algorithm is shown in Figure 2.4.

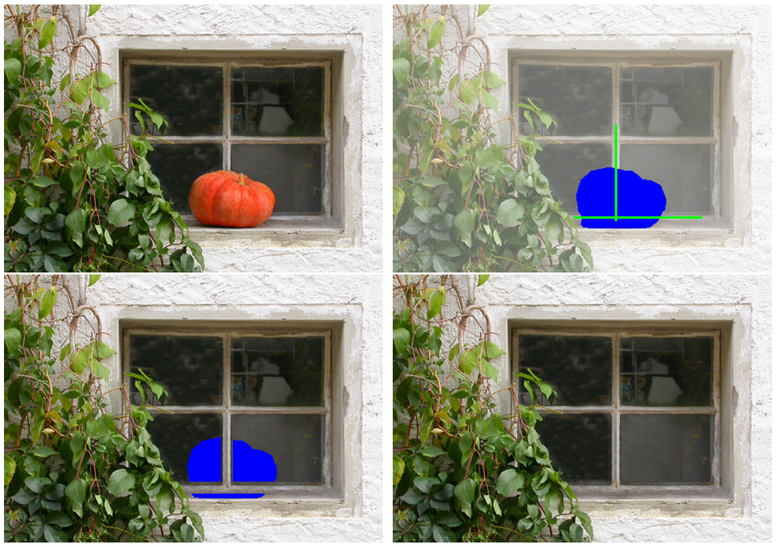


Figure 2.4: Sun method [8], input, mask, first stage and result.

2.4 Multiple images

Hays *et al.*[9] developed a total different way of image completion, instead of searching in the input image, the algorithm searches throughout a whole database of images to find information to fill the missing region. The new area is pasted in using Poisson blending [5]. The authors state that their results look better then the algorithm of Criminisi *et al.*[1] but the algorithm needs a database of two million images for only three scenes. This algorithm gives visual pleasing results but is very slow and is unpractical because it needs the large dataset. An example of this algorithm is shown in Figure 2.5.

2.5 Video inpainting

Image completing methods can also be used for videos, because a video is a set of multiple images. Naively inpainting each frame will not result in the best result important information about the current frame can be found in the adjacent frames. Patwardhan *et al.*[10] presented an algorithm for video inpainting of a scene taken from a static camera. This method is an extension of the paper of Criminisi *et al.*[1]. It extends the idea from a single image

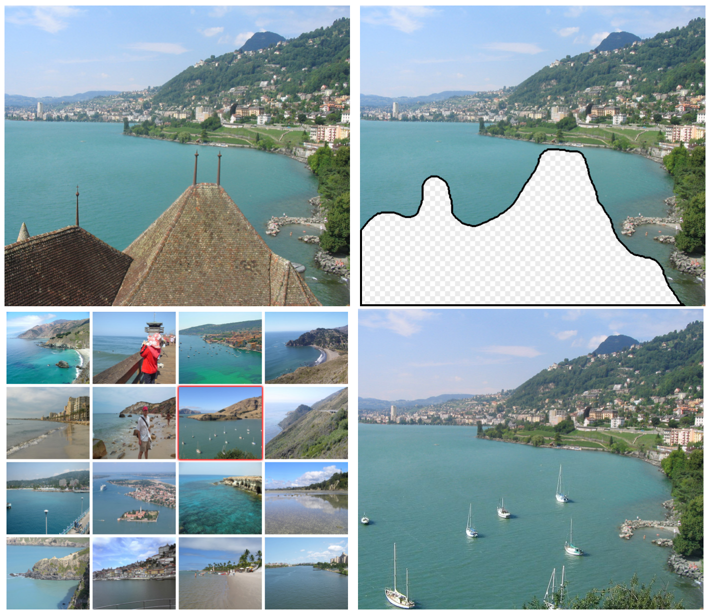


Figure 2.5: Hays method [9], input, mask, first stage and result

to a set of images but also taken in to account the adjacent frames. In this algorithm the frame is separated into a background and a foreground in which the foreground is first inpainted. After this step the background is inpainted, each patch is copied to every frame to get a consistent background. This technique has some nice results but it has some restrictions, it requires a fixed camera position and a stationary background with some moving foreground.

Wexler *et al.*[11] have proposed a method for space-time completion of large damaged areas in a video sequence. They pose the problem of video completion as a global optimization problem with a well-defined objective function. In the algorithm every local patch should be found in the remaining part of the video and globally all these patches must be consistent with each other spatially and temporally. An example of this algorithm is shown in Figure 2.6.

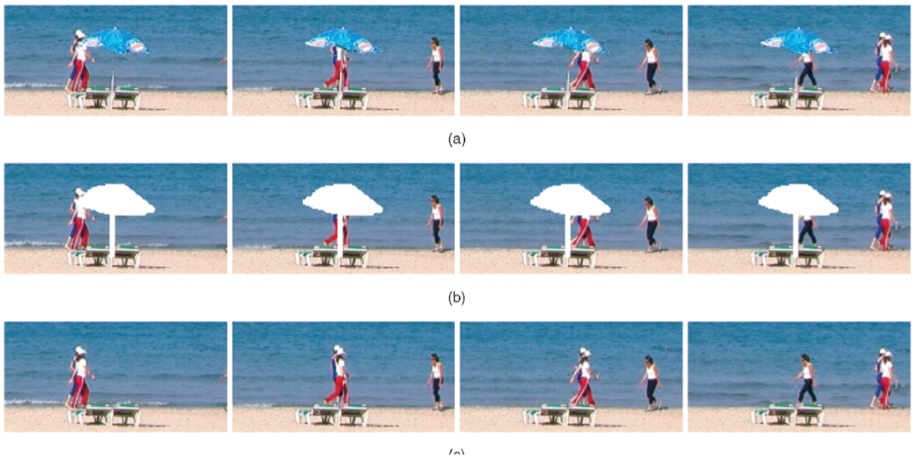


Figure 2.6: Wexler method [11], input, mask and result sequence.

Chapter 3

Image Inpainting

This chapter discusses the paper Image Inpainting by Bertalmio *et al.*[2]. It is an important paper in this field because it is the first paper that uses a different approach. The algorithm presented in this paper is based on partial differential equations (PDE's). The ideas in this paper are inspired by the basic techniques used by professional art restorators. By analysis the advantages and drawbacks of this algorithm, we try to reach one of the goals of the project, to improve this algorithm.

3.1 Theory

The technique proposed in this paper does not require any user intervention after the user specifies the image that has to be restored and the mask that represents which part of the image should be inpainted. The original image is described as I_0, the area that has to be inpainted is denoted as Ω, the border of this region $\delta\Omega$ and the known region also known as source region is indicated by Φ. This is shown in Figure 3.1

The algorithm works at an iterative way, by creating a series of image $I_1, I_2 ... I_n$ in a way that the new image is an improved version of the previous image. This process will continue until a certain amount of iterations or until the algorithm converges. The process is described as shown in Equation (3.1). The new pixel value is calculated as the old pixel value of a pixel inside Ω plus Δt times the newly calculated value for this iteration.

$$I^{n+1}(i,j) = I^n(i,j) + \Delta t I_t^n(i,j), \forall (i,j) \in \Omega \tag{3.1}$$

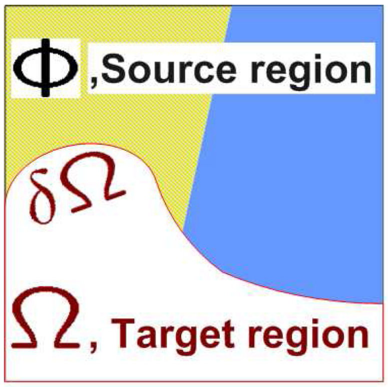


Figure 3.1: Definitions image inpainting.

The values for each iteration can be calculated. This is shown in Equations (3.2), in which $\overrightarrow{\delta L^n}(i,j)$ is an estimation of the smoothness of the image at pixel i,j. $\left|\overrightarrow{N}(i,j,n)\right|$ is the direction of the isophote. And $|\nabla I^n(i,j)|$ is the slope-limited version of the norm of the gradient of the image, a factor to make the algorithm more stable. The equations for the terms are Equations (3.3), (3.5) and (3.6).

$$I_t^n(i,j) = (\overrightarrow{\delta L^n}(i,j) \cdot \frac{\overrightarrow{N}(i,j,n)}{\left|\overrightarrow{N}(i,j,n)\right|}\ ) \left|\nabla I^n(i,j)\right|, \tag{3.2}$$

$\overrightarrow{\delta L^n}(i,j)$ is an estimation of the smoothness of the image at pixel i,j. It is calculated as the neighbors of the discrete implementation of the Laplacian of the pixel i,j. The neighborhood is calculated as described in Equation (3.3) and the discrete implementation of the Laplacian is described as Equation (3.4), in which the subscripts represent the derivatives, in this case the second derivatives in the x and y direction.

$$\overrightarrow{\delta L^n}(i,j) := (L^n(i+1,j) - L^n(i-1,j), L^n(i,j+1) - L^n(i,j-1)), \tag{3.3}$$

$$L^n(i,j) = I_{xx}^n(i,j) + I_{yy}^n(i,j), \tag{3.4}$$

The next part of Equation (3.2) is the direction of the isophote meaning the direction of the curves of equal intensity. This is done by calculating the direction of the least change of color, by calculating the gradient

$\nabla I^n(i,j)$ as $(I^n_x(i,j), I^n_y(i,j))$ and taking the vector orthogonal to this vector $(-I^n_y(i,j), I^n_x(i,j))$. The normalized version of the equation can be calculated as shown in Equation (3.5).

$$\frac{\overrightarrow{N}(i,j,n)}{\left|\overrightarrow{N}(i,j,n)\right|} := \frac{(-I^n_y(i,j), I^n_x(i,j))}{\sqrt{(I^n_x(i,j))^2 + (I^n_y(i,j)^2)}} \tag{3.5}$$

The final part of Equation (3.2) can be calculated as the slope limited version of the norm of the gradient of the image. This equation is shown in Equation (3.7), this equation has as parameter β, the projection of $\overrightarrow{\delta L}$ onto the vector $\overrightarrow{N}$, which is calculated as shown in Equation (3.6), which is the first part of Equation (3.2).

The terms of Equation (3.7) are as followed, the sub indexes b and f denote backward and forward differences respectively, while the sub indexes m and M denote the minimum or maximum, respectively, between the derivative and zero. This equation is used to prevent too small and too big numbers.

$$\beta^n(i,j) = \overrightarrow{\delta L^n}(i,j) \cdot \frac{\overrightarrow{N}(i,j,n)}{\left|\overrightarrow{N}(i,j,n)\right|}, \tag{3.6}$$

$$|\nabla I^n(i,j)| = \begin{cases} \sqrt{(I^n_{xbm})^2 + (I^n_{xfM})^2 + (I^n_{ybm})^2 + (I^n_{yfM})^2}, \\ when \beta^n > 0 \\ \sqrt{(I^n_{xbM})^2 + (I^n_{xfm})^2 + (I^n_{ybM})^2 + (I^n_{yfm})^2}, \\ when \beta^n < 0 \end{cases} \tag{3.7}$$

The algorithm will first run one step of anisotropic diffusion on the whole image. After this step the algorithm will run A steps of the algorithm followed by B steps of anisotropic diffusion. A and B are parameters, the authors suggest the values A is 15 and B is 2 for the best result. Anisotropic diffusion will be explained in the next section.

3.2 Anisotropic diffusion

Anisotropic diffusion or Perona- Malik diffusion[12] is a process that removes noise off an image without removing the 'edges' of the image. The process is inspired by the technique of applying a Gaussian kernel to image to generate smoother image. When applying a Gaussian kernel to an image the set of images get smoother but it does not keep edges. In the paper of Perona and Malik, the authors give the example of an image of a tree with a blue sky. The result what the authors want is that the leaves of the tree get merged together in the first iterations of the algorithm when applying more steps of the algorithm the branches of the tree should merge until there is only a single 'Blop' left. When applying a Gaussian kernel to the image the leaves already get merged with the blue sky before the branches of the tree get merged. In this algorithm the anisotropic diffusion helps to smooth the filled in regions. It also helps to join edges across the unknown region, so they we be continued.

3.3 Quality of inpainting results

The quality of a result of image inpainting is very subjective, sometimes it is very clear if an image looks "good" but most of the times "not that clear". Therefore we want to define a formal measurement to see how "good" an image looks. A numerical measurement would be practical to compare different inpainting methods with each other. This is theoretically not possible because image inpainting is filling the gap with visual plausible information. The information that *could* have been there. The information that is in the original image is not always known. Therefore there is no ground truth to compare the result with. For example Figure 3.2, an image of a bungee jumper. After removing the bungee jumper there is no information of what would be behind the bungee jumper, so there is no ground truth to compare the result with.

A solution for this problem is only using a custom dataset to compare the results of different methods. This dataset is created by taking a picture with the object that you want to remove and taking one without while the rest of the picture has to be exactly the same. This is almost impossible, keeping the rest of the image exactly the same. For example the camera moved just a little, the reflection of an object is on another object etc. See Figure 3.3. Therefore we use a different method to estimate the quality of an inpainting result. On an image a mask is drawn of an object that could have been on the image. The use of this virtual mask will allow us to compare the end result with a ground truth, which is the input image.

To measure the quality of an image we calculate the Mean Square Error(MSE) between the inpainted image and the original image. This tech-

Figure 3.2: Bungee jumper, no ground truth known.

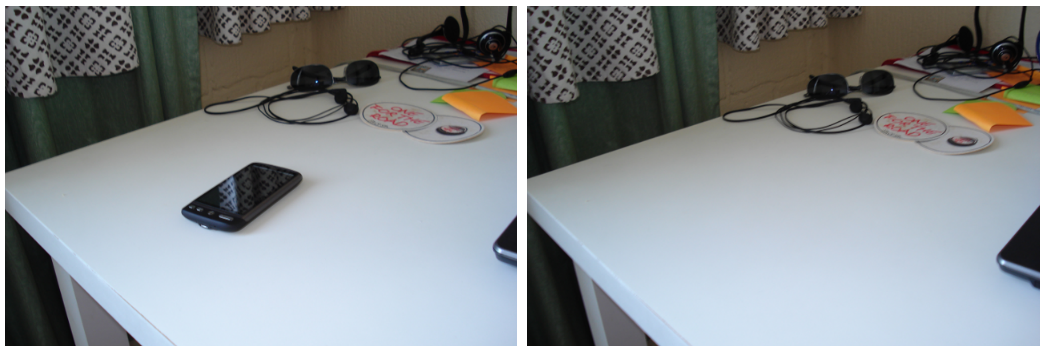


Figure 3.3: Ground Truth problem, left original image, right incorrect ground truth.

nique has also been used in the paper of Oliveira *et al.*[4] and other papers. This is not the optimal way of comparing images because human perception does not work the same way as color values in RGB color space.

The MSE is defined as:

$$MSE = \frac{1}{mn} \sum_{y=1}^{M} \sum_{x=1}^{N} [I(i,j) - I'(i,j)]^2, \tag{3.8}$$

In which m and n are the dimensions of the image, I(i,j) is the original image at pixel (i,j) and I'(i,j) is the inpainted images at the same position. Note that a higher MSE is lower quality in the resulting image.

The maximum value of the MSE is 65536 (256^2). To get a better overview between the values we normalized the results resulting in a value between 0 and 1. This normalized Mean Squared Error can be calculated as:

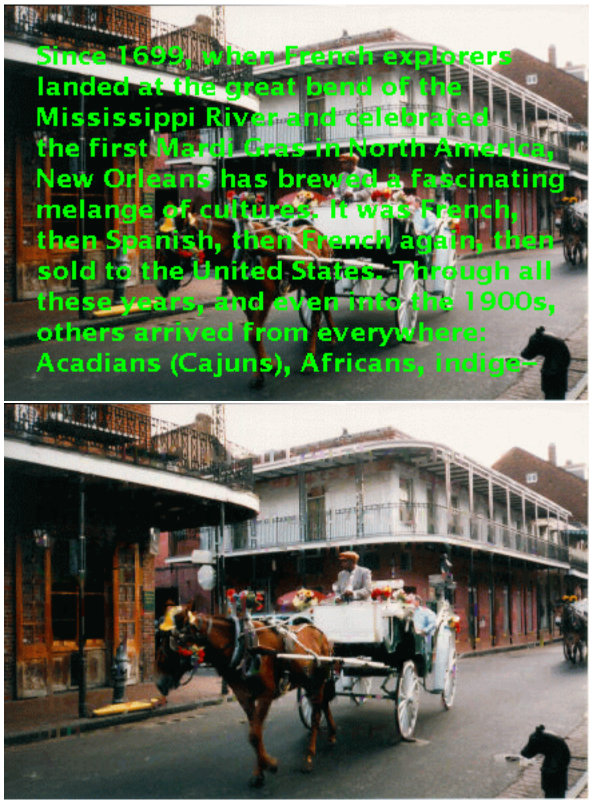
$$nMSE = \frac{1}{mn * 255^2} \sum_{y=1}^{M} \sum_{x=1}^{N} [I(i,j) - I'(i,j)]^2, \tag{3.9}$$

3.4 Experiments

One of the goals of this project is to see if it is possible to make some improvements to the Bertalmio paper. This is done by implementing the paper and looking at the advantages and drawbacks of the algorithm. The algorithm was implemented in C++ with the Open Source Computer Vision library version 2.2.

The algorithm gives acceptable results in specific settings. The algorithm will produce good results if the unknown regions are thin, if the unknown region gets to wide the results of the algorithm will decrease which is further explained at the drawbacks. The algorithm does not need any user interaction after the input image and the masked region is specified. This in contrast to the papers published before this paper. No limitations are made on the topology of the unknown region.

The first drawback is that the algorithm does not perform well on big unknown regions. The result will be blurry. This is because the algorithm is not able to reproduce texture. This can be seen in Figure 3.5, when the mask is not wide the algorithm works a lot better as can be seen in Figure 3.4.



Since 1699, when French explorers
landed at the great bend of the
Mississippi River and celebrated
the first Mardi Gras in North America,
New Orleans has brewed a fascinating
melange of cultures. It was French,
then Spanish, then French again, then
sold to the United States. Through all
these years, and even into the 1900s,
others arrived from everywhere:
Acadians (Cajuns), Africans, indige-

Figure 3.4: Result small region width of 3 pixels.

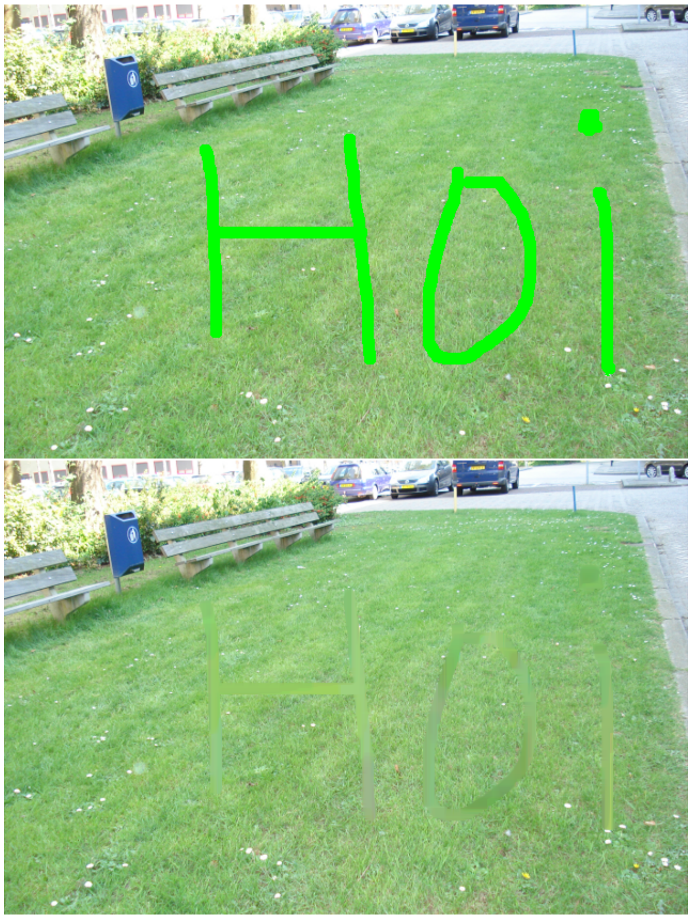


Figure 3.5: Result big region width of 12 pixels.

This drawback is conquered in other papers, for example the paper of Criminisi and texture synthesis. In these algorithms the information is being copied in patch by patch, not pixel by pixel. Th3is will result in texture in the resulting image in images with larger gaps instead of blur. It is not possible to incorporate this techniques in this algorithm because it would change the core ideas of the algorithm.

3.4.1 Multi resolution images

The algorithm is relatively slow because it needs a lot of iterations of complex calculations for all unknown pixels of the algorithm. This can be partially solved by using multi resolution images. With this idea the image is scaled down a few times, the smallest scale is inpainted first of which the inpainted pixels are copied to a larger scale. Knowing the information from the previous scale will reduce the amount of iteration the algorithm needs to converge. With this idea the unknown region can be faster inpainted knowing the color value of the previous scale. This process increases the speed of the algorithm significantly.

3.4.2 Inpainting inwards

Another way of improving the speed of the algorithm is changing the sequence of the to be inpainted pixels. Our hypothesis is that we can improve rate of change and by doing that, the speed of the algorithm by working from the border of the unknown region $\delta\Omega$ inwards. The idea of this approach is that the pixels at the border of the unknown region get inpainted with the information from outside the unknown region first. The pixels that lay one pixel inwards now get inpainted by the information of the border values. With this inpainted sequence the pixels that lay one pixel inwards will be based on the already filled border pixels not on the possible not yet filled border pixels.

To see if our hypothesis is confirmed we evaluate the rate of change if the algorithm is not yet finished. By doing this evaluation with a not yet finished results we can see if the algorithm is 'further' at a certain amount of iterations of the algorithm than the default algorithm would be.

For this evaluation we use four images which are shown in Figure 3.6, these image are used because they will result in visual pleasing end results. We evaluate the current pixel values if the algorithm is approximately 75%

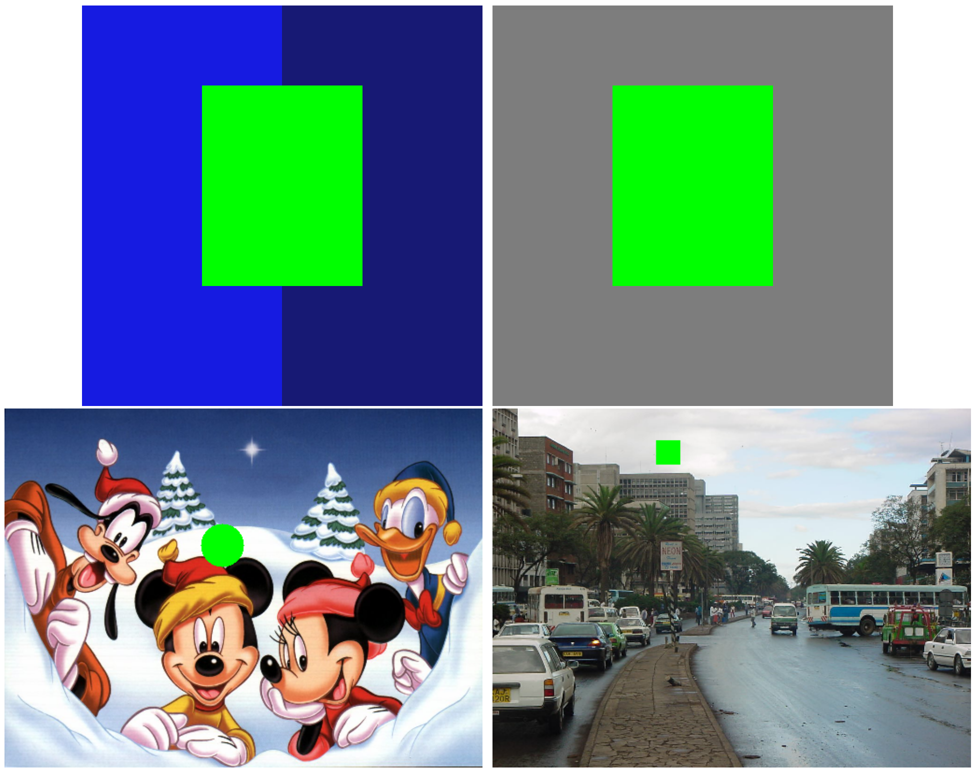


Figure 3.6: Input images plus masks, input14, input14a, disney006, city004.

finished. We compare these images of the regular and enhanced algorithm with the ground truth. By calculating the MSE between the images and the ground truth we can evaluate how far the image is inpainted.

This experiment shows no improvement as can be seen in Table 3.1. This can be explained by the small value of the update value Δt. The border values are only marginally updated. The one pixel inwards of border values get calculated with marginally update values, not enough to improve the value. Increasing the value of Δt to 0.2 will result in the expected end result. All error values of the enhanced method will be equal or decreased, so the hypothesis is confirmed. The results are shown in Table 3.1 and 3.2. Table 3.1 shows the error values when $\Delta t = 0.1$ and table 3.2 shows the error values when $\Delta t = 0.2$.

image	amount of iterations	MSE regular	MSE inwards
input14	400	78.2	78.2
input14a	400	8.9	8.9
disney006	3200	5.7	5.7
city004	400	0.6	0.6

Table 3.1: Error comparison regular versus inwards $\Delta t = 0.1$.

image	amount of iterations	MSE regular	MSE inwards
input14	200	418.7	396.0
input14a	200	167.2	164.7
disney006	1600	12.7	7.0
city004	400	0.6	0.6

Table 3.2: Error comparison regular versus inwards $\Delta t = 0.2$.

3.4.3 Iterations estimation

Multiple parameters of the algorithm can be set, each set of different parameters resulting in different results. These parameters are the amount of iterations before the algorithm finishes. The numbers of inpainting steps after which an amount of diffusion steps is applied. The margin of convergence can also be set. All the possible values of the parameters in combination with the relatively slow algorithm makes the algorithm less practical to use.

The usability of the algorithm will be improved if we would be able to correctly specify how many iterations the algorithm should run given a certain image. Running the algorithm always with the same amount of iterations is not user friendly and will not always give the best results. Using a margin for the convergence will not be effective too because of the possible slow rate of improvement. We want to estimate the amount of needed iterations based on the size of the unknown region.

The amount of iterations is based on the width of the masked region. The width of the mask is calculated as how many times you can erode the mask before it is empty. We base the amount of iterations per width layer on the amount of iterations the algorithm needs to change a masked region from total white to total black. We notice that the algorithm needs 400 iterations to calculate one layer. The algorithm is used to calculate the end results for the previous mentioned images, which are shown in Figure 3.6 because the end results of these images are visual pleasing.

In the regular algorithm the user has to specify a fixed amount of iterations. In this case the amount of 12800 is used based on experimentation. The end results of the test images are the same after the estimated amount of iterations as after 12800 iterations. We can state that estimation of the amount of needed iterations works. Table 3.3 shows the amount of iteration needed for four test images. We also analyzed how many iterations the algorithm really needed. This is shown in Table 3.4. There are slightly more iterations estimated than really needed to make sure the result is correct.

image	estimated iterations	regular iterations	percentage
input14	800	12800	6.25 %
input14a	800	12800	6.25 %
disney006	9600	12800	75.00 %
city004	5200	12800	40.63 %

Table 3.3: The amount of estimated iterations compared to the regular amount of iterations.

image	estimated iterations	needed iterations
input14	800	750-800
input14a	800	500-550
disney006	9600	8700-8750
city004	5200	800-850

Table 3.4: The amount of estimated iterations compared to the amount of needed iterations.

3.4.4 Conclusion

We can conclude that we found three possible contributions to the paper. The first using multi-resolution images is already mentioned in the paper but not explained. The experimentation shows that this contribution significantly decreases the time needed to finish the algorithm. The second contribution, inpainting inwards significantly reduces the needed amount of iterations. By doing so it also reduces the time needed for the algorithm. The third contribution estimating the amount of needed iterations will reduce the amount of unnecessary iterations. By doing so it will also increase the speed of the algorithm.

Chapter 4

Exemplar Based Image Inpainting

This chapter discusses and analyzes the paper of Criminisi *et al.*[1]. The authors proposed an algorithm inspired by the algorithm by Bertalmio *et al.* [2] and texture synthesis [3]. In contrast to the paper of Bertalmio et al. the isophote information is not directly used to update the target region of the image but it is used to determine which patch should be updated first.

4.1 Theory

The goal of the algorithm is to fill in a region in an image. This is done by first selecting which patch of the unknown region of the image has to be inpainted. After that step the best matching patch for this region is searched. The following definitions are used in the paper, see Figure 4.1. The region to be filled is indicated by Ω and its border is denoted by $\delta\Omega$. The known region also known as source region is indicated by Φ. First the pseudo code of the algorithm is defined, after that each step is looked into more closely.

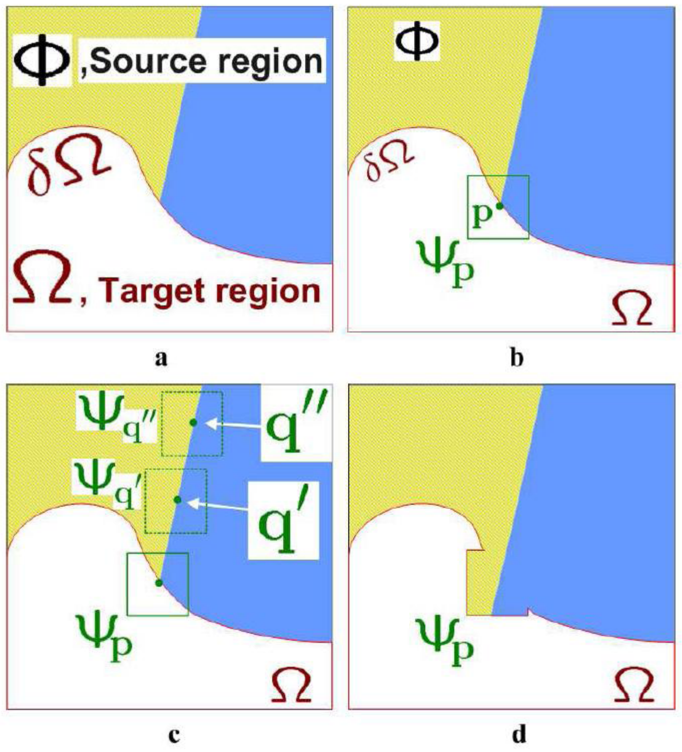


Figure 4.1: Exemplar based image inpainting definitions part 1.

The pseudo code of the algorithm:

1. Calculate the initial boundary $\delta\Omega$ and confidence values C(p).
2. Repeat until the boundary is null.
   (a) identify boundary.
   (b) compute the priority P(p) for every point on this boundary.
   (c) find the patch with the maximal priority $\Psi(p)$.
   (d) find the best exemplar $\Psi(\hat{q})$.
   (e) copy the pixel information from the exemplar to the patch with the highest priority.
   (f) update the confidence values of the patch with the highest priority with the calculated values.

1. Calculate the initial boundary $\delta\Omega$ and confidence values C(p).
   The first step of the algorithm is to calculate the boundary also known as the fill front of the masked region. This boundary is needed because we want to select a point on the boundary if there are still masked values left.
   To be able to calculate the confidence value of a patch, the initial confidence value C(p) has to be set before the iterative steps of the algorithm. The pixels that are on the mask are set to 0, the other pixels are set to 1.

2. Repeat until the boundary is null
   Repeat the following steps of the algorithm until there is no more boundary, are pixels are inpainted.

   (a) Identify boundary.
   If the image has been changed in the previous iteration, the boundary has to be re-calculated. See figure 4.1 a.

   (b) Compute the priority P(p) for every point on this boundary. For each point on this boundary, the priority has to be calculated. This priority is the product between the Confidence term C(p) and the Data term D(p), see Equation (4.1.)

   $$P(p) = C(p)D(p) \tag{4.1}$$

   The Confidence term represents how much information is known from the patch around the point p. At the first iteration this is the amount of known pixels in the patch divided by the amount of pixels in the patch. This is because the known pixels are initialized with the value of 1 and the not known pixels with 0. After the first iteration, the values calculated with this equation are also taken in account. The equation for this confidence term is Equation(4.2).

   $$C(p) = \frac{\sum_{q \in \Psi p \cap (I-\Omega)} C(q)}{|\Psi_p|}, \tag{4.2}$$

   The data term represents the edge propagation. See Equation (4.3).

   $$D(p) = \frac{\left|\nabla I_{p^\perp} \cdot \mathbf{n}_p\right|}{\alpha}, \tag{4.3}$$

In which $\nabla I_{p^\perp}$ is the vector orthogonal to the gradient vector. $\mathbf{n}_p$ is the normal vector of the boundary and α is a normalization factor, typically 255 for a grey-scale image, see Figure 4.2.

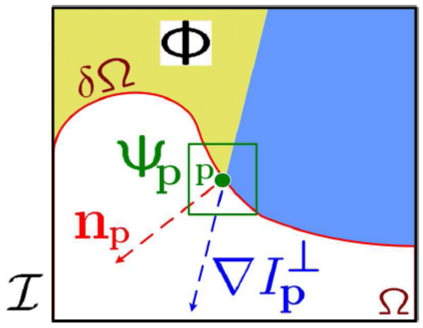


Figure 4.2: Exemplar based image inpainting definitions part 2.

(c) Find the patch with the maximal priority $\Psi(p)$.
$\widehat{p}$ = arg max $_{p \in \delta\Omega^t}$ P(p)
Find the patch with the highest priority of all points on the boundary. See Figure 4.1 b.

(d) Find the best exemplar $\Psi(\widehat{q})$.
$\Psi(\widehat{q})$ = arg min $d(\Psi(\widehat{p}), \Psi(q))$
Search in Φ for a patch that has the least Sum of the Squared distance(SSD) in CIE LAB color space with $\Psi(\widehat{p})$ See Figure 4.1 c.

(e) Copy the color information from the exemplar to the unknown pixels of the patch with the highest priority.
See figure 4.1 d.

(f) update the confidence values of the patch with the highest priority with the calculated value of equation 4.2.

4.2 Experiments

To analyze the algorithm, it was implemented in C++ with the Open Source Computer Vision library version 2.2. The experiments were done to improve the visual quality of the results of the algorithm and to improve the speed of the algorithm. The first experiment was meant to improve the visual appeal of the end result. The second set of experiments were meant to increase the speed of the algorithm.

4.2.1 Finding the exemplar

In the paper of Criminisi *et al.* the image is searched for the patch that is most similar to the patch around the point with the highest priority p. Where most similar is the patch that has the lowest Sum of Squared Differences(SSD) of the color values of the known pixels between the two patches. Nothing is explained about the case with multiple different possible patches with the same SSD. Naive searching for the best exemplar from the left top to the right bottom corner and selecting the first patch that has the lowest SSD, might not result in the best result. The algorithm searches for a patch that looks like $\Phi(p)$, shown by the blue square in Figure 4.3. The close-up is shown in Figure 4.4. Searching the image from left to right and from top to bottom will result in a set of possible patches. The first patch the algorithm encounters is the red square. Other possible candidates are on the same row to the right of the first patch, for example the yellow patch. Because the SSD of the first patch is equal to the other good patches this one is selected. Although the other patches might be perceived as better patches.

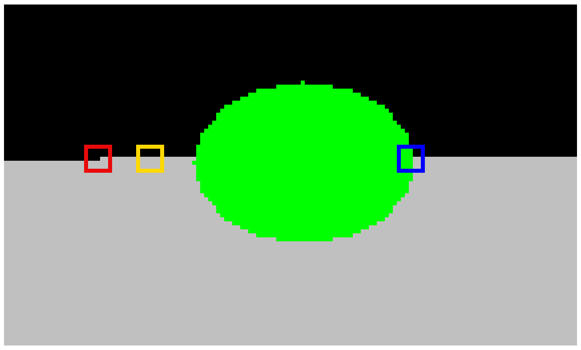


Figure 4.3: Patch selection sequence. Blue square is $\Psi(p)$, red and yellow are two possible patches.

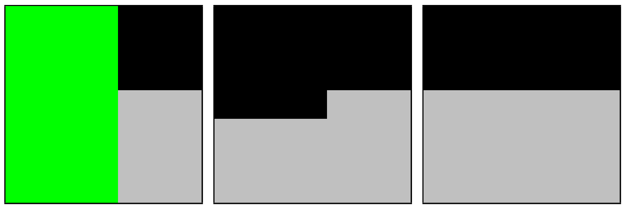


Figure 4.4: Patch selection close-up, from left to right: the blue, red, yellow patch.

There can be more than one patch with the lowest SSD and not all possible patches are the same. Selecting the best solution would be the patch that would 'fit' the mask the best. For example in Figure 4.3, in this image the yellow patch is perceived as a better fit then the red patch, because the human observer would think that the edge would continue instead of going down a pixel as shown in the yellow and red square. See close up in figure 4.4. Modeling what humans perceive as better patches would result in higher quality images. This is done by estimation the masked values of the patch and taking them into account when searching for the best patch.

The unknown pixel values are estimated by the same method as in paper of Kwok *et al.*[13]. In their algorithm the estimated pixels are needed to search trough a search array data structure. The authors suggest two ways of extrapolating the patch with the highest priority, in the paper also called the query patch. The first way is filling the empty pixels with the average color values of the known pixels, the second idea is filling the patch gradient based.

In the average based estimation the unknown pixel values are assigned the value of the average value of the known pixels values. The gradient based filling method is based on the known pixel values surrounding a not known pixel. In detail, for each unknown pixel $p(i,j)$ letting the gradient be zero in relation to the known left/right $p(i \pm 1, j)$ and top/bottom $p(i, j \pm 1)$ neighbors. This will result in an over-determined linear system. The optimal value will be calculated by computing the Least-Square solution which minimizes the norm of the gradient. This means calculating the different pixel values of the known pixel values surrounding the unknown pixel. And taking the value that has the least difference between all of them, which is the average of the known surrounding pixels.

The set of possible best matches is first selected using the regular selection procedure. After this step the best one is selected by calculating the SSD of all the pixels taking in account the estimated values. Figure 4.5 shows the different ways of filling a simple example patch. The first patch is the regular patch, with the masked values displayed green, the second patch is the patch average filled and the third patch is gradient based filled.

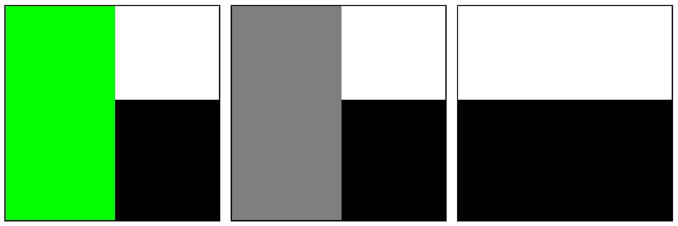


Figure 4.5: Example of different query patches, from left to right: original patch, average and gradient based estimated.

The results of the patch estimation are shown in Table 4.1, note that a lower value is a lower error which means a better result. The values shown in the table are the average normalized MSE or the whole image category.

image category	nMSE average	nMSE gradient	nMSE regular
City	0.003772	0.003737	0.003796
Dieren	0.002838	0.002835	0.002910
Disney	0.003664	0.003771	0.003742
Nature	0.003834	0.003834	0.003834
Simple	0.017309	0.011678	0.020300
Real	0.003527	0.003544	0.003571
Total	0.006284	0.005171	0.006916

Table 4.1: Error comparison patch estimation.

In Table 4.1 the results are shown of five categories: the first four categories are datasets with 10 images of different real images. These are images from cities, animals, nature in general and Disney cartoons. In the fifth category there are 10 images of synthetic images.

The average nMSE of the average and gradient estimation of most of the categories is only marginally less than the regular method. In the synthetic images the nMSE differs a lot. This can be explained because of the fact that the synthetic images were chosen to prove some more complicated scenes and the images and masks of the other categories are randomly chosen. In the real images categories there are no images in which the regular algorithm really fails. So for the further analysis an image is used that will result in a visual unpleasing end result.

In the synthetic images the gradient based fill has always a better or equal result than the regular algorithm, the average based fill performs sometimes better and sometimes worse. This is shown in Figure 4.6.

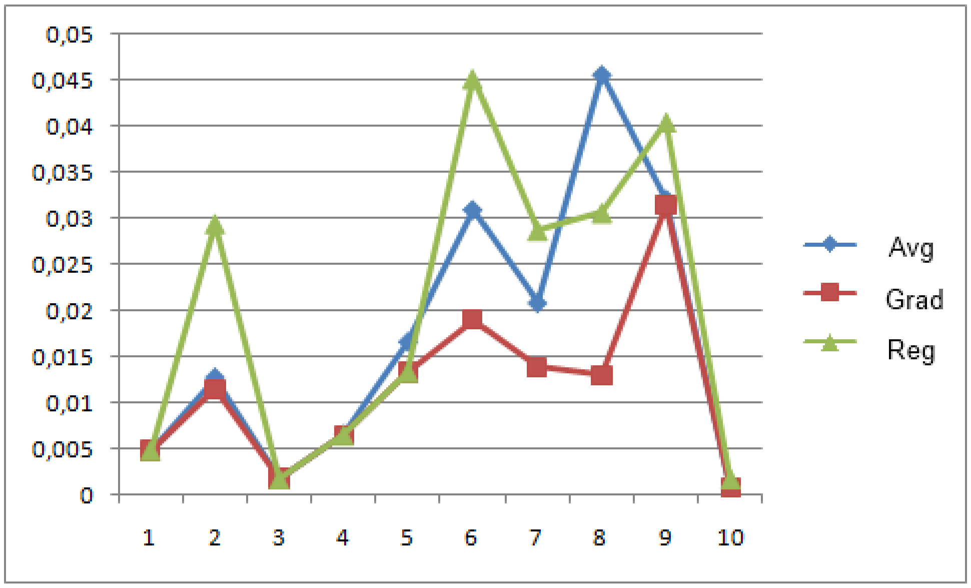


Figure 4.6: Comparison error rating of the different estimation methods (image 1-10).

We can conclude that gradient-based patch estimation significantly increases the results of synthetic images. The result of the other images of the previous mentioned dataset is not increased significant.

The next set of images shows the scenario that the use of patch estimation increases the result in real images. Figure 4.7 shows the image that has to be inpainted, the green values indicate the mask on the images. Figure 4.8 shows the result of regular inpainting, Figure 4.9 shows the result of gradient based inpainting. The iteration in which the problem with the regular inpainting occurs is shown in Figure 4.10. In this image the mask is indicated with red pixels, the pixel that will be inpainted in this iteration is shown green. The blue pixels indicate the patch that has the lowest SSD with the query patch. Note that is on the brown area near the top border.

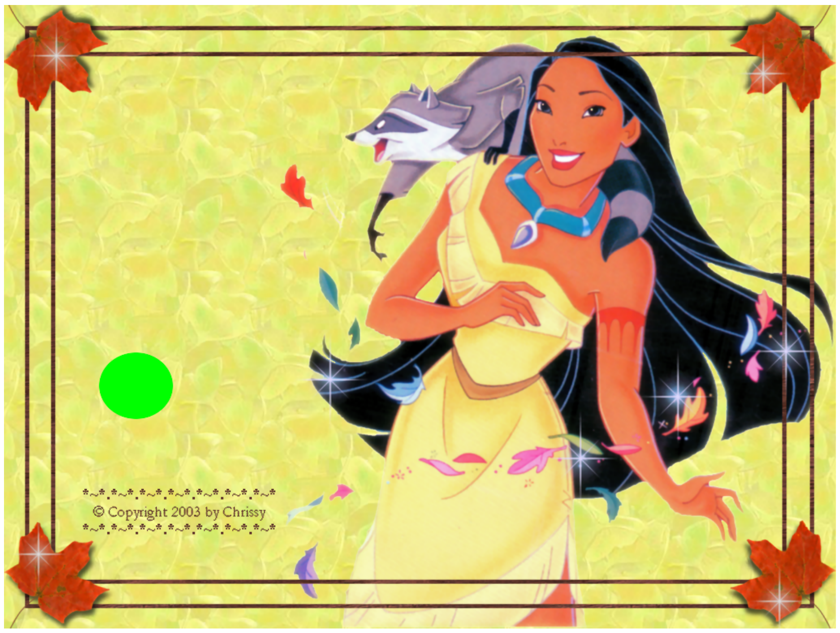


Figure 4.7: The to be inpainted image.

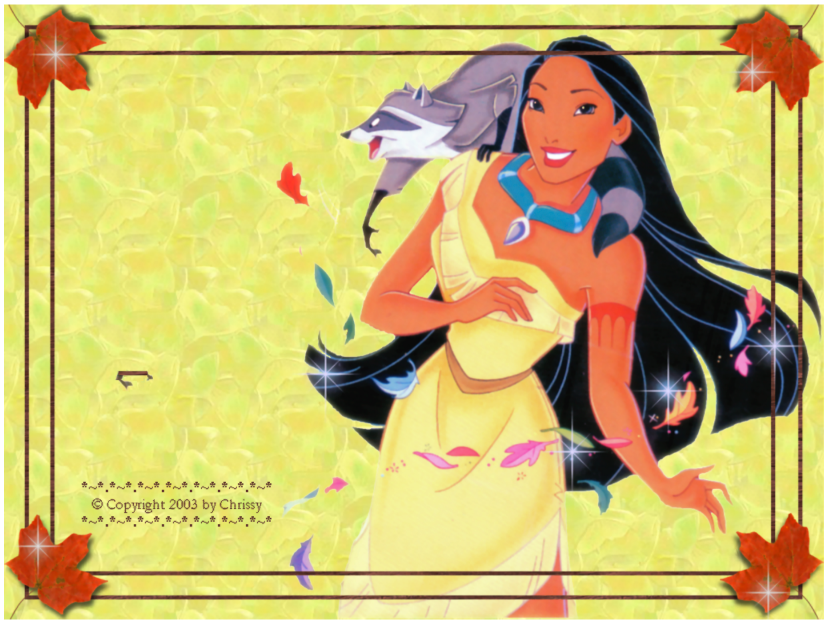


Figure 4.8: result of regular inpainting.

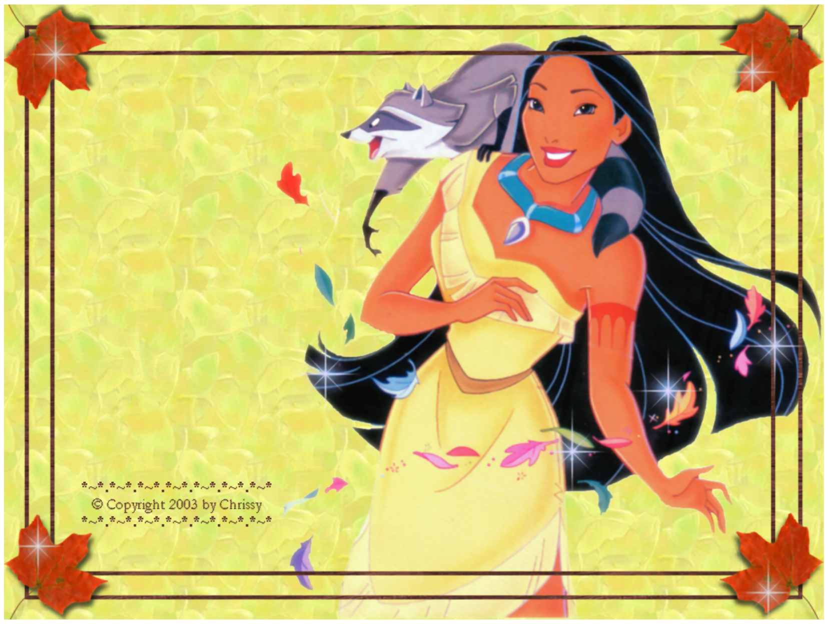


Figure 4.9: Gradient based inpainting.

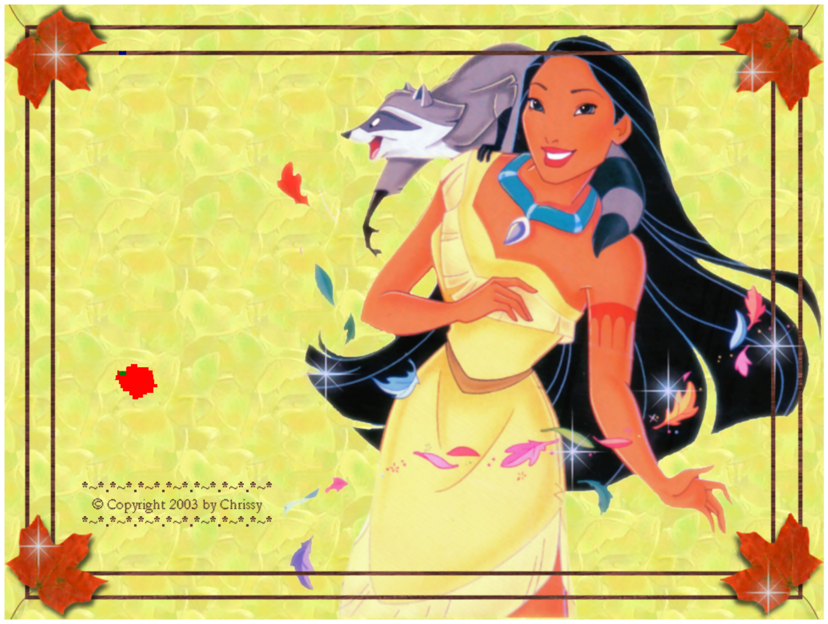


Figure 4.10: Iteration in which the problem occurs.

An idea would be to use the extrapolated patch as the query patch. This is less intuitive as the normal algorithm because we calculate the SSD with all patches in the image with estimated values. The amount of error is shown in Table 4.2.

image category	nMSE gradient	nMSE first gradient est.	nMSE regular
City	0.003737	0.003676	0.003796
Dieren	0.002835	0.002770	0.002910
Disney	0.003771	0.003971	0.003742
Nature	0.003834	0.003783	0.003835
Simple	0.011678	0.013861	0.020300
Real	0.003544	0.003550	0.003571
Total	0.005171	0.005612	0.006916

Table 4.2: Error comparison first selection vs. patch first estimation.

The previous mentioned method does not seem to improve the results compared to the regular gradient based: the average error values on the real images are almost equal and the result of the synthetic images are worse.

4.2.2 Local search

The speed of the algorithm is depended on two factors, the amount of pixels that have to be filled and the search area. When there are a lot of pixels in the masked region: it takes a lot of iterations before the algorithm finishes. If the search area is very big one iteration takes a lot of time. The search area can be reduced by only searching in an area around the point with the highest priority. This solution is also suggested in different variations in other papers, two of those papers are the papers by Chen *et al.*[14] and Anupam *et al.*[15]. In this technique there is only searched in the area around the patch with the highest priority. In which the search area is an input parameter. This method does speed up the process significantly but it does not guarantee that the correct patch is selected. We want to investigate the quality and the speed of this local search method. If the quality does not significantly decreases and the speed significantly increases this is a valuable improvement. This local searching is shown in Figure 4.11. The patch with the highest priority is shown with the red square, multiple local search areas are shown in yellow. The algorithm will only search in the yellow area instead of the whole image, which is a lot faster.

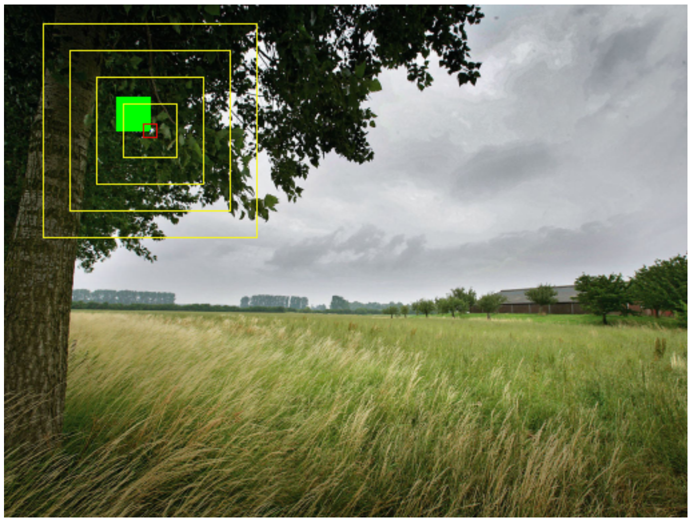


Figure 4.11: Different areas of local searching. The red patch is $\Psi(p)$. The yellow squares indicate the search regions.

In the case of nature001.png, it is hard to say which result looks better.

The result of the original algorithm or the result of the algorithm with the local search. This is shown in Figure 4.12 and 4.13.

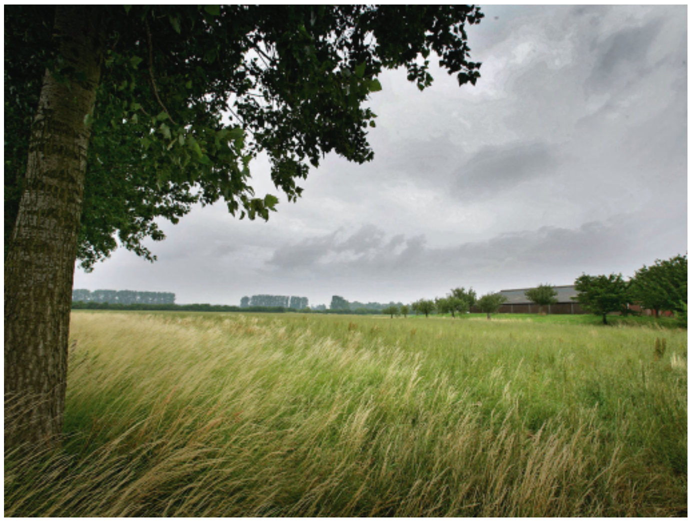


Figure 4.12: Result local search.

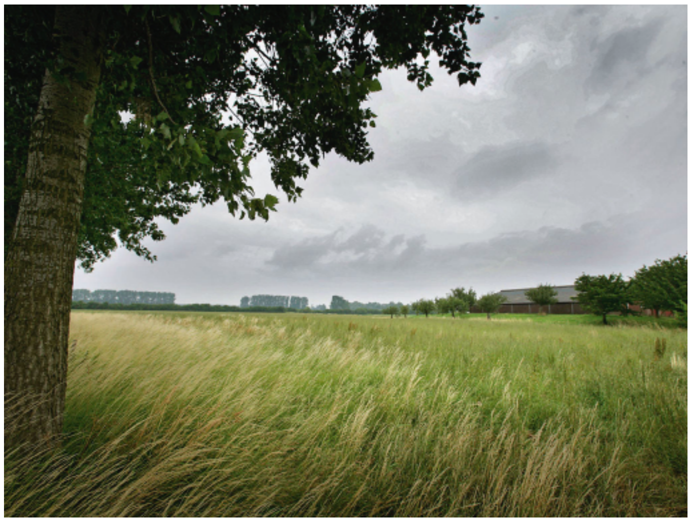


Figure 4.13: Result regular search.

We have two hypotheses for the local searching of a complete dataset. The first is that the different local searching methods speed will be at most

50 % of the original algorithm on average on the whole dataset. The second is that the error margin of the local searching methods will be less than 105 % of the original algorithm. We want to compare a dataset consisting of 5 categories, images of cities, animals, Disney cartoons, images of nature in general and synthetic images. In these synthetic images we want to prove some easy and some challenging cases. The size of the query patch is the same as in the original paper. The different search areas that we want to research are:

- method 1 the area of the search area is 41 x 41: roughly 5 times as large as the query patch.
- method 2 the area of the search area is 81 x 81: roughly 10 times as large as the query patch.
- method 3 the area of the search area is 121 x 121: roughly 15 times as large as the query patch.
- method 4 the area of the search area is 161 x 161: roughly 20 times as large as the query patch.

The average running times of the different methods are:

- Average run time method 1 = 7,1 s
- Average run time method 2 = 7,2 s
- Average run time method 3 = 8,2 s
- Average run time method 4 = 9,4 s
- Average run time for the original method = 27,4 s

The comparison between the different methods is shown in the next graph. The City dataset is chosen because it represents the rest of the datasets and showing all datasets will result in a less clear graph.

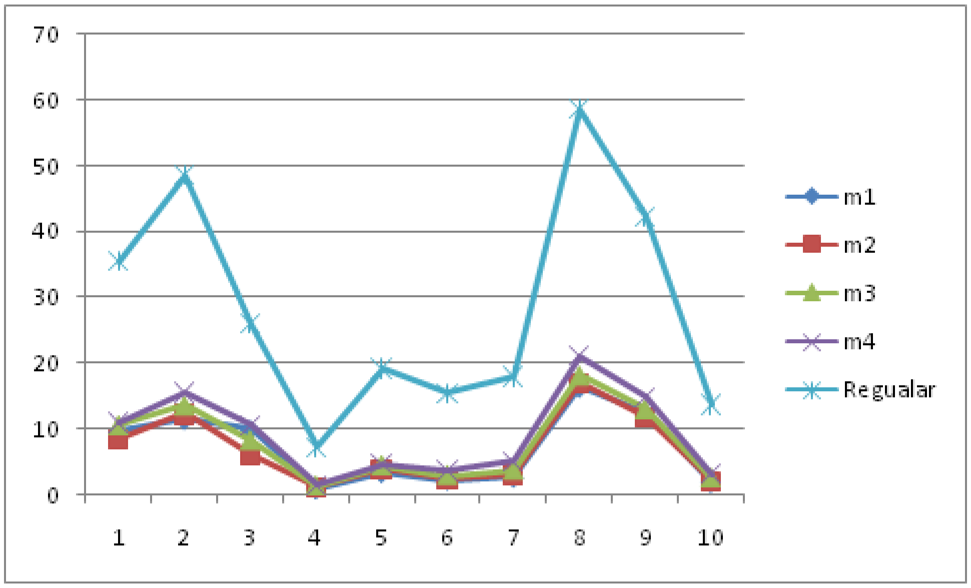


Figure 4.14: Comparison running time (s) different methods city dataset (image 1-10)

The average running time of method 1-4 is 27.2, 27.6, 31.6, 36.1 % of the original algorithm. We can conclude that hypothesis 1 is true: the time needed for the algorithm to finish is decreased, it is less than 50 % on average on the whole dataset. Now look at the quality of the local search, the average error of the whole dataset should not be more than 105 % of the original algorithm. The next table will show the average nMSE of the different types of images in the dataset.

method	m1	m2	m3	m4	Regular
city	0.003867931	0.0037856	0.003846555	0.003852556	0.003795645
animals	0.002903525	0.00304247	0.002954053	0.002852621	0.002910101
Disney	0.003538927	0.003627958	0.003737955	0.003778746	0.003742715
nature	0.003815098	0.004174904	0.004016146	0.003807364	0.003834601
synthetic	0.012937862	0.016778802	0.019923678	0.01978139	0.020300476
all images	0.005412669	0.006281947	0.006895677	0.006814535	0.006916708
real images	0.00353137	0.003657733	0.003638677	0.003572822	0.003570766

Table 4.3: Error comparison different methods and types of images.

Method 1 should have a higher error rate than method 2, etc. This is true for none of the datasets, when the local search very close to current patch (method 1). This will sometimes give even better results. The average error rating are for method one until four, 0.00541, 00628, 0.00689, 0.00681 respectively. This is 85.2 %, 98.9 %, 108.5 %, 107.2 % respectively. This results in the fact that hypothesis 2 is assumed false. We expected that the MSE would be bigger if we search only near the query patch. The experiments show that this is not always the case. The average error of the smallest local searches is even less than the regular algorithm, 0.005412669 and versus 0.006916708.

Some of the synthetic images are chosen to prove some challenging cases, this means that the result of these cases can be very extreme and might influence the average error of the whole dataset a lot. The average error value of method 1 of the real image dataset is 0.00353137 compared with 0.003570766, so the local search has less effect on the real image dataset. The other local search methods have a higher average error value, these are 0.003657733, 0.003638677 and 0.003572822.

Further analysis local search

We want to explain why the error in the closest local search sometimes gives better results. For this analysis we look at image Disney006.png which is shown in image 4.15, in this images the result of the local search witch the size of 41x 41 is compared with the result of the local regular algorithm with the size of 81x81. This is because the problem is more obvious in this method then in the regular algorithm.

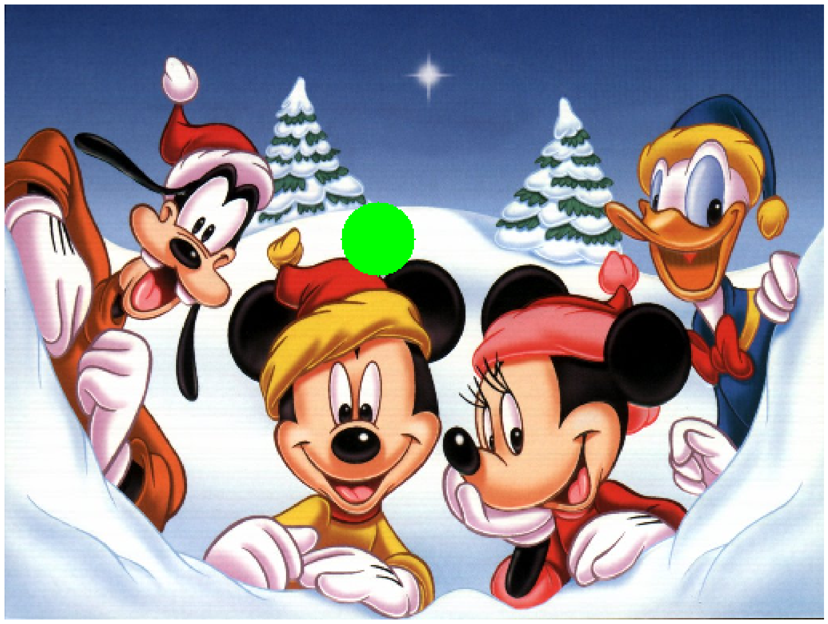


Figure 4.15: Disney006.png, the mask is shown with green pixels

In the iteration 1457 pixels left, the problem is the most obvious. In method 1 the patch is selected that is part of the red part of the cap. In method 2 the patch is selected that is part of the yellow/ brown part of the cap. This selection will result in a better end result for method 1 compared to method 2. Also the patch of method 1 looks like it would fit the query patch the best.

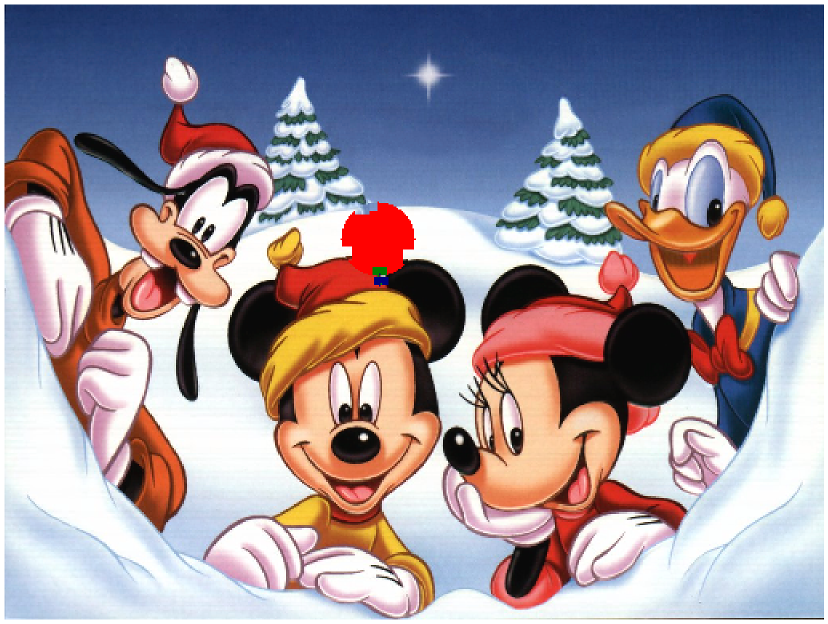


Figure 4.16: Disney006.png, method 1, at iteration 1457 pixels left.

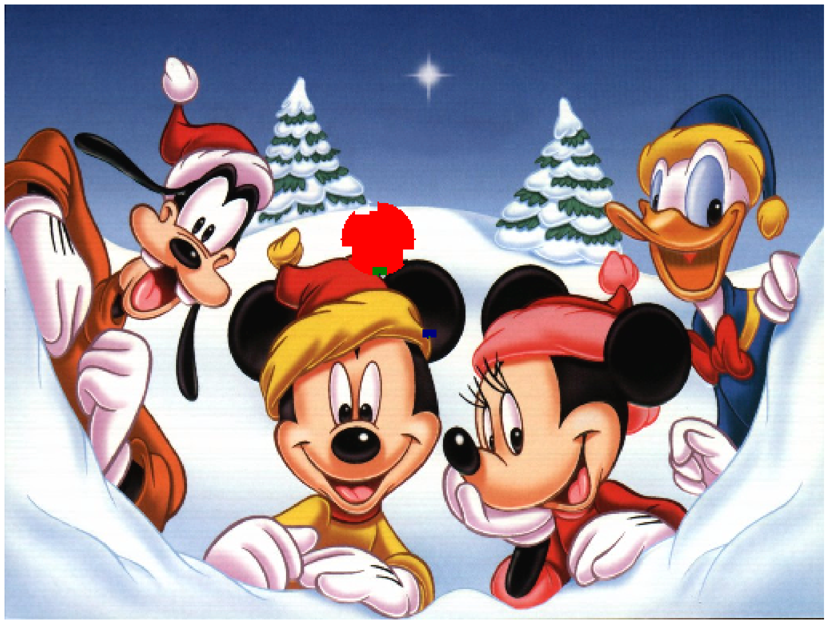


Figure 4.17: Disney006.png, method 2, at iteration 1457 pixels left.

Now we have to analyze why the patch of method 2 is chosen instead of the patch of method 1. This is possible because the area of method 2 is bigger than method 1, also including the area of method 1. In Figure 4.18, the query patch is shown. In Figure 4.19 the exemplar patch of method 1, in Figure 4.20 the exemplar patch of method 2. First the patch is displayed with the mask. Second without the mask and third the difference of the known pixels with the query patch, in the last patch the difference is displayed from white, no difference to red, a difference of 255 on average in 3 color channels.

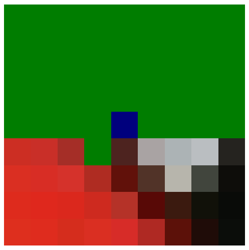


Figure 4.18: patch with the highest priority.

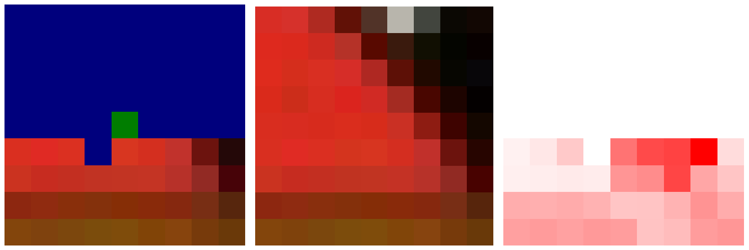


Figure 4.19: exemplar patch 1, the patch with the mask, without the mask and the difference.

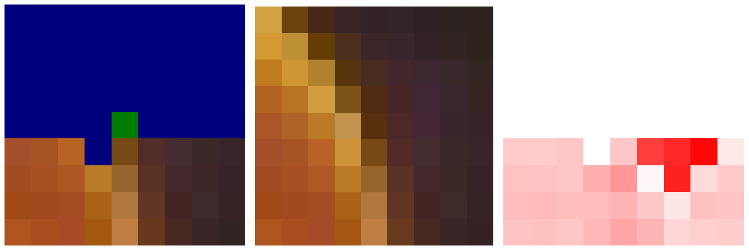


Figure 4.20: exemplar patch 2, the patch with the mask, without the mask and the difference.

Although at first it looked like the patch of method 1 would fit a lot better but the error is bigger than the patch of method 2. The SSD between the patches and the query patch is 131323 versus 115889, this is a difference of 15434. This is 190,5 per pixel, this is 63 per color channel, this results in a difference of 7,9 per pixel per color channel because the previous mentioned value is squared.

The SSD of method one compared to method two in CIE LAB color space is bigger, is this also the case in other color spaces? In RGB space the SSD of method 1 is: 336389 versus 309455 with method 2, this is a difference of 26934. This difference is even larger. In the HSV color space the difference is even bigger, method 1 has an SSD of 546567, compared with 340497 of method 2. We can conclude that although the patch of method 1 looks better in a SSD calculation is not lower. The right part of the patch influences the result a lot.

In some cases, the patch that looks like a better match is not a better matching patch as calculated by the mean square error. The two resulting images are shown in Figure 4.21 and Figure 4.22.

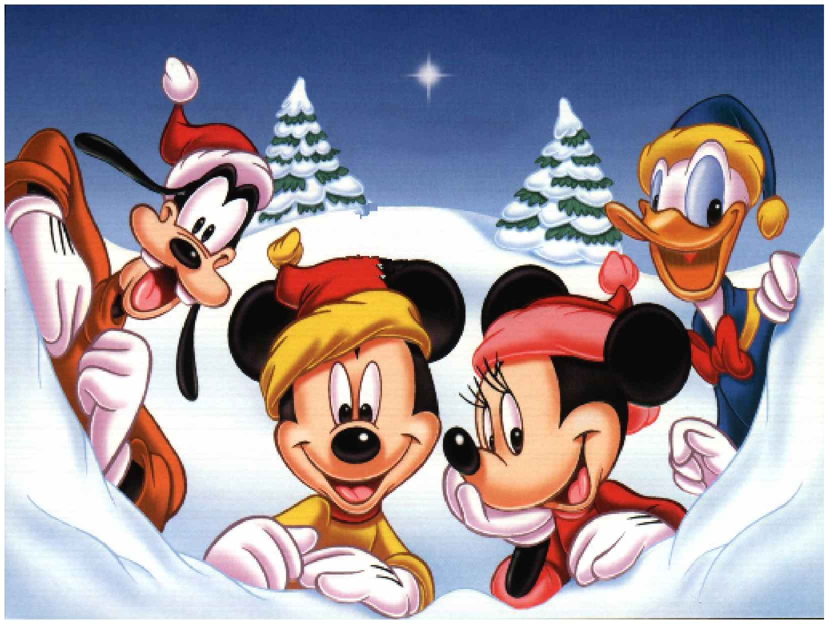


Figure 4.21: Disney006.png, result method 1.

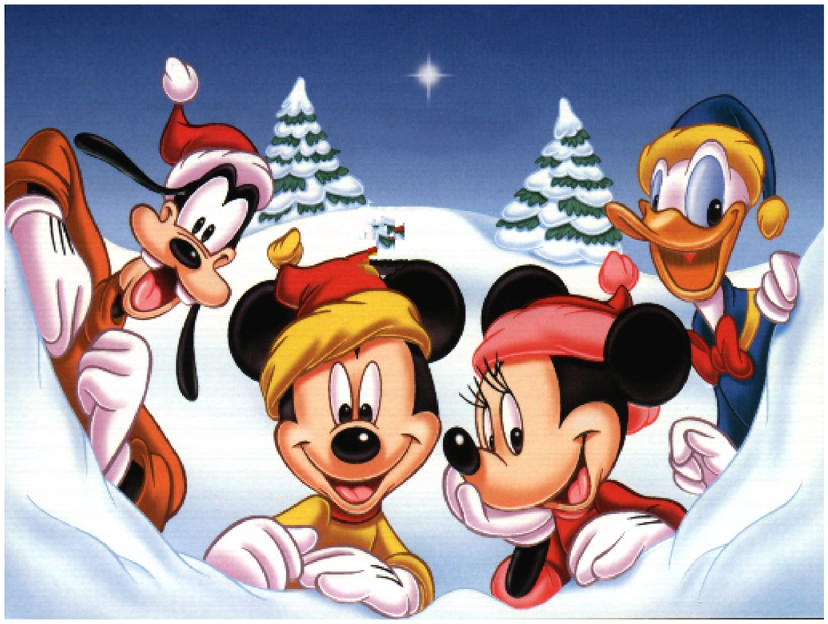


Figure 4.22: Disney006.png, result method 2.

From this analysis we can conclude two things. First, a patch with a lower SSD will not always result in a better end result. Second, in some cases the search with the smallest search region will result in a better end result. This might be explained because the exemplar patch belongs to the same object as the query patch.

4.2.3 Lookup data structures

Another way to reduce the search time is creating a lookup data structure to speed up the process. Instead of calculating the Sum of Squared Differences(SSD) of the query patch with all the possible patches we only have to calculate it for the amount of leaves at each level of the data structure. A tree structure can be used for this, in which a node is a patch of the image. This idea is shown in Figure 4.23. This tree can be created by using different methods, k-means and k-medoids are used for this, which are explained in the next sections.

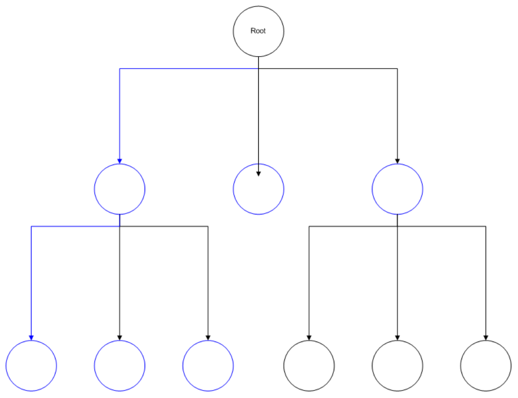


Figure 4.23: Idea of a tree structure.

K-means

K-means is a clustering algorithm, which in this algorithm is used to cluster groups of patches together. The algorithm is able to create k clusters, depending on the user input. This clustering process is visualized in the next four figures, for visualizing proposes the patches are visualized as 2dimential points. The steps of the k-means algorithm on a dataset of points are:

1. random select the initial means see Figure 4.24.
2. associate points to the initial means see Figure 4.25.
3. shift the means to the centroid of the initial cluster see Figure 4.26.
4. repeat the step 2 and 3 until the algorithm converges see Figure 4.27.

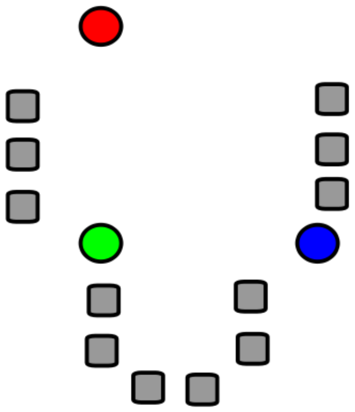


Figure 4.24: K-means step 1, random select the initial means.

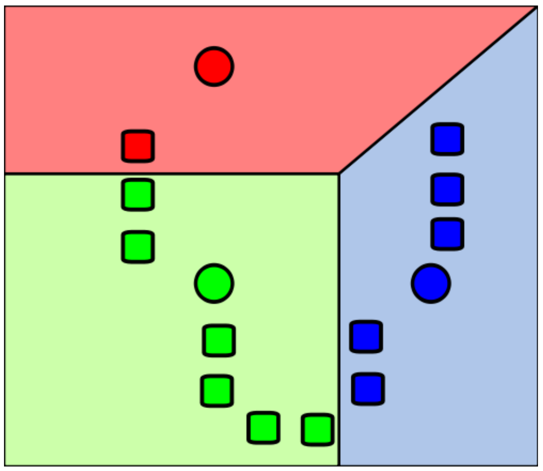


Figure 4.25: K-means step 2, associate points to the initial means.

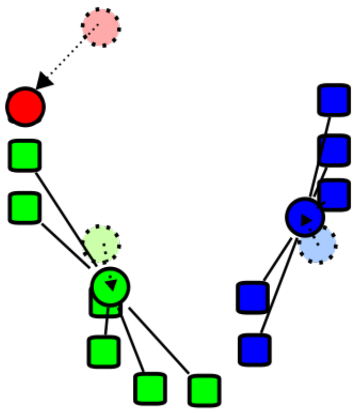


Figure 4.26: K-means step 3, shift the means to the centroid of the initial cluster.

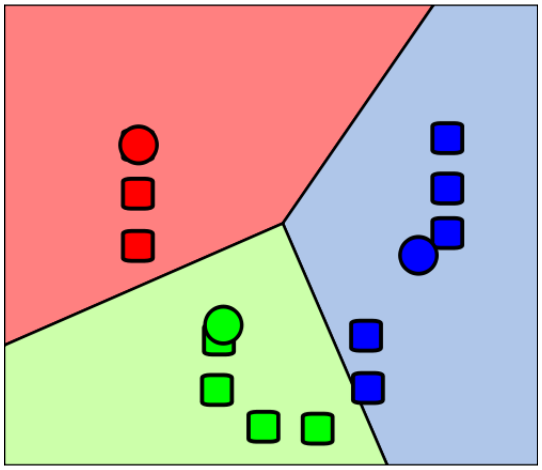


Figure 4.27: K-means step 4, repeat the step 2 and 3 until the algorithm converges.

K-medoids

K-medoids is another clustering algorithm, instead of K-means, the algorithm allows the user to specify the distance metric. It is more robust to noise and outliers as compared to k-means because it can calculate the difference between the data points in different ways instead of the sum of squared Euclidean distances. In this case the patches are also simplified into points.

1. Randomly select k data points as medoids.
2. Associate each data point to the closest medoid. (”closest” here is defined using any valid distance metric, most commonly Euclidean distance or Manhattan distance)
3. Calculate the total cost, using the previous mentioned distance metric
4. For each medoid
   For all associated data points
   Swap m and o' and compute the total cost of the configuration
5. If the cost of the old configuration is more than the new configuration, change it to the new configuration.
6. Repeat steps 2 to 5 until there is no change in the medoid.

The following figures will demonstrate K-medoids for a data set of 2-d points. Figure 4.28 demonstrates the original dataset. Figure 4.29 shows the two random selected medoid points (c1 and c2) and the points that are close to those medoid points. Figure 4.30 shows the swapping of the points, changing c2 in o'. This will not result in a better result so another new medoid point o' is selected.

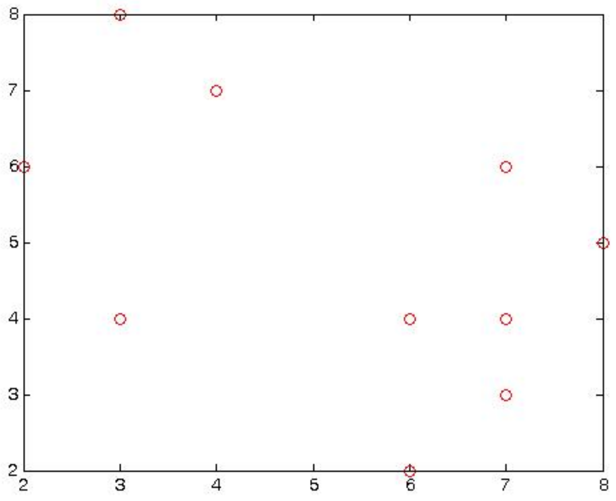


Figure 4.28: K-medoids, points.

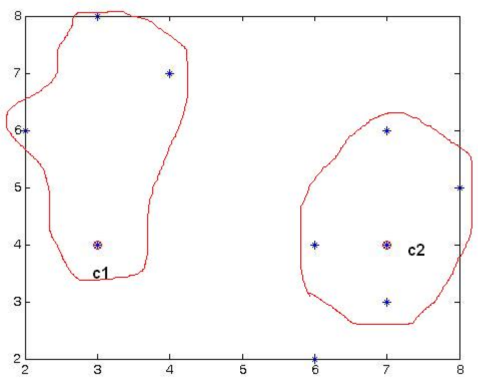


Figure 4.29: K-medoids, random selected medoid points c1 and c2.

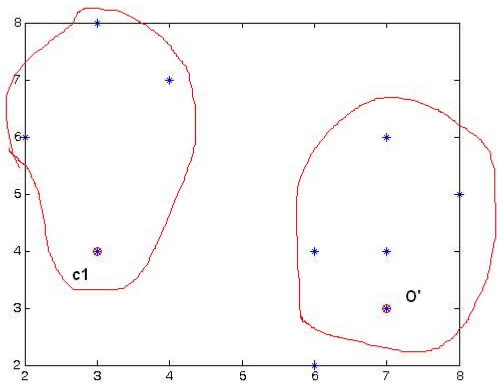


Figure 4.30: K-medoids, swapped o' with c2.

Estimated values

The problem with this approach is that at the creation step of the tree, all pixel values of a patch are known. When searching for the best exemplar only the non-masked values are used. This will result in selecting a possible wrong patch. This is shown in Figure 4.31. The patch with the highest priority $\Psi(p)$ is shown on the right, with the masked values in green. When we traverse the tree, we first go left because the white values of $\Psi(p)$ have the least color difference with the left patch. After this step it is impossible to access the left sub patch of the right patch. So the wrong patch will be selected.

Extrapolating the patch as suggested in the paper of Kwok *et al.*[13] would solve this problem according to the authors of the paper. The authors suggest two ways of extrapolating the patch with the highest priority, in the paper also called the query patch. The first way is filling the empty pixels with the average color values of the known pixels, the second idea is filling the patch gradient based. This is also used in chapter 4.2.1.

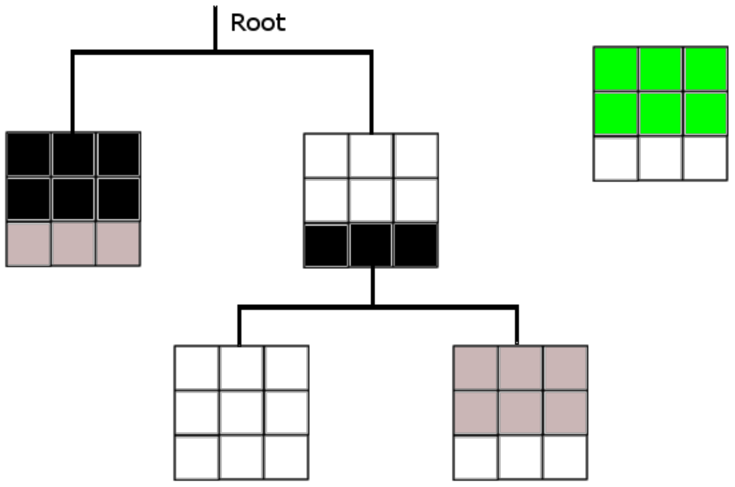


Figure 4.31: Problem using tree.

First Experiment

My hypothesis is that the algorithm will become 25 % faster using the lookup data structures, with an error that may be 5 % higher calculated as the nMSE. For this experiment a dataset is used containing random selected images and random selected masks. The first experiment is designed to notice the difference between the two data structures, estimate the correct parameters and notice other problems. This experiment only consists out of ten images. The second experiment is designed to validate the in the experiment made conclusions and consists out of 40 images.

Speed analysis

During the test runs of the experimentation, the runs with the k-means data structures were not always faster than the regular Criminisi algorithm. This is because the masked region has to be large enough. When the masked region is too small the creation of the tree takes too much time.

The tests showed that the masked region of the images has to cover at least 1.5% of the image for the k-means method to be faster. The average running time of the test dataset using the k-means data structure decreased with 9.95 %. In this dataset there were also images with a small masked region. The average time of the K- medoids method decrease with 48.37 %.

Quality analysis

The decrease of the quality of the result image estimated by the nMSE has to be less than 5%. The average nMSE of the test dataset of the K-means data structure is 1.5% more than the nMSE of the regular Criminisi algorithm. The average nMSE using the K-medoids data structure is 10.8 % more than the regular algorithm. This shows that for this dataset we can not use the K-medoids data structure.

Second Experiment

In the second experiment we try to confirm the results from the first experiment. The average nMSE of the K-means and K-medoids data structure compared to the regular algorithm should be less than 5% for a bigger dataset. The parameters of the algorithm are k = 2 and max dept level of the tree = 2. A dataset of 40 random selected images is used. For these images random created masks are used. The size of these masks is increased to be larger than 1.5% of the whole image but it seems that the mask were smaller because of the random creation of the masks.

Speed analysis

The speed of the algorithm increases with 7.8% using the k-means data structure. This value is lower than expected because not all masks cover 1.5%, because of the randomness of the masks. Removing the images with a mask smaller than 1.5% results in an increasement of 26.3%. The speed of the algorithm using the k-medoids increases with 34.4%.

Quality analysis

The quality as estimated by the nMSE decreases with 0.8 % of the k-means data structure compared with the regular algorithm. The results of the algorithm using the k-medoids decrease with 6.6%.

Conclusion

My hypothesis is false for the k-medoids data structure. The speed increases with 34.4%, as predicted with the hypothesis. The quality however does decreases too much. The quality as estimated by the nMSE decreases with 6.6%. The hypothesis of the k-means is confirmed if the masked region in the image is at least 1.5% of the whole image. For the whole dataset the algorithm becomes 7.8 % faster and the error becomes 0.8% higher. If we calculate the error and speed from the images with a masked region that is big enough. The speed increases with 26.3% and the quality only decreases with 0.4%.

Chapter 5

Quantitative and Qualitative Comparison

As described in chapter 3.3 the quality of an inpainted image is subjective. In the previous chapters the result of the inpainting method is compared with the ground-truth using a virtual mask. The result comparison is done by using the (n)MSE to estimate the quality of the resulting images. During the project we noticed the (n)MSE is not always an ideal error estimation. An example of this is that a low error value will result in a visual pleasing result, but a large error value does not necessarily has to result in a poor result. This observation is showed in Figure 5.1. We want to investigate if this quantitative estimation matches the quality of the images as perceived by human beings. This is done by creating an experiment to analyze some of these observations.

5.1 Experimental setup

To do this comparison, we want to do a user study to compare the quantitative estimation matches the quality of the images as perceived by persons. To compare these two methods we look at the rating of the MSE of the different methods and compare this with the rating of the different methods in the user study. By doing this we can compare both methods.

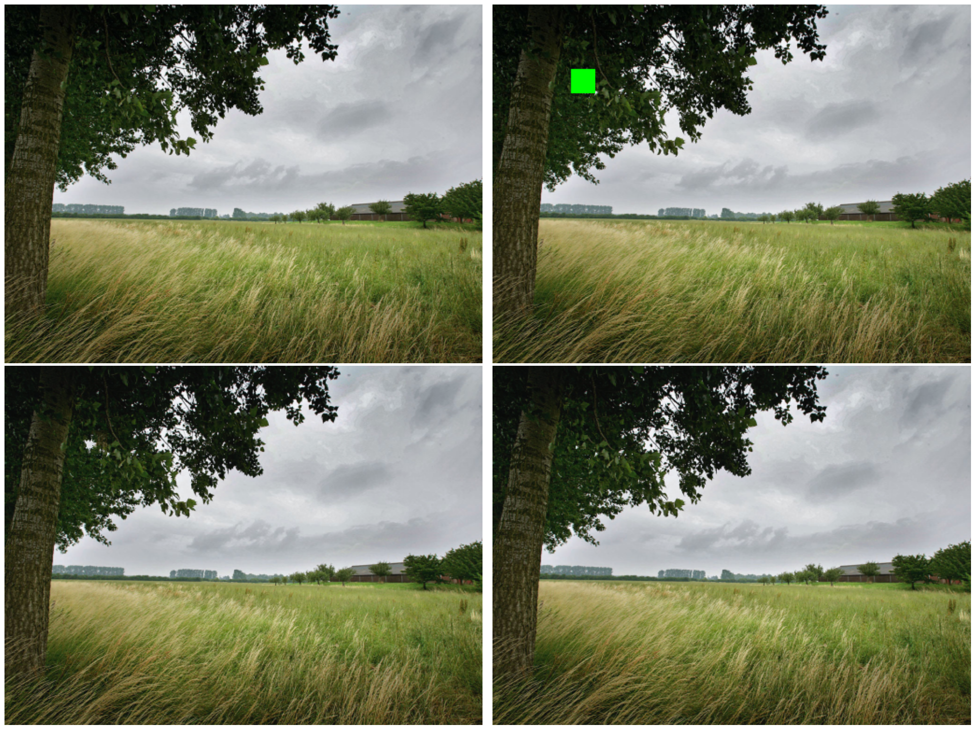


Figure 5.1: Observation 2, input, mask, first result (high error) and second result (low error).

5.1.1 Dataset

In this experiment we verify some important observations made during the project. The dataset of the experiment is based on the following observations:

1. A low error rating will result in a visual pleasing result; a high error rating will result most of the times in a bad result.
2. A higher error rating does not necessarily has to result in a less pleasing result.
3. The error calculation of the Criminisi algorithm can be higher as the Bertalmio algorithm although the result will be equal or more visual pleasing.

The dataset of the rest of the experiments consisted out of random selected images and masks, in this experiment a dataset is used that consist of images supporting the previous mentioned observation. This experiment is based on a small dataset of 14 images. In each image an observation is verified.

5.1.2 Implementation

The user study is created with the idea that a lot of people can cooperate in the experiment. The experiment is implemented as a web page that can be viewed on different kinds of devices. In this way people are able to do the experiment from their home pc or do the user survey on a smart phone or tablet. The experiment starts with an introduction and an example. After that the real experiment starts: first the original image and mask are shown in one image, second the result is shown and the participants can grade the result of an image. Participants can grade the resulting images from 1 to 5, in which 1 is visual very unpleasing result and 5 is visual very pleasing result.

5.2 Results

25 persons participated in the experiment with an average age of 31. The average rating of all images of all participants was calculated. With these average rating the observations were validated. The results were as followed:

Observation 1: a low error rating will result in a visual pleasing result, a high error rating will result most of the times in a bad result. This observation is validated by two times three images. Three low error rated images which should result in visual pleasing results and three high error rated image which should result in visual unpleasing results. The three low error rated images result in an average rating of 3.6, 3.9 and 4.8. The three high error rated images result in an average rating of 1.5, 1.7 and 1.8. We can state that the user study verified this observation.

Observation 2: a higher error rating does not necessarily has to result in a less pleasing result. For this observation we tested two times two images. In which the image with the higher error rate as calculated by the MSE should result in equal rating by the participants. This is true for the first set of two images, the rating differs less than one point between the two images. The ratings are 4.1 and 4.6. In the second set of images the difference is larger, 3.1 and 4.3. This can be explained by the large difference in rating in image 14, the image with the rating of 3.0. Some participants graded this image with the grade 1, a very poor result, while other people graded this image with the grade 5, a perfect result. Because of the big spread we can state that the result is acceptable. Observation two is verified taking into account the big spread in the ratings of image 14.

Observation 3: the error calculation of the Criminisi algorithm can be higher as the Bertalmio algorithm although the result will be equal or more visual pleasing. The observation is tested with two times two images. In the first set of images this observation is confirmed, the Criminisi algorithm result image is graded with 4.4 versus the 1.9 of the result of the Bertalmio algorithm. In the second set of images the both methods are graded equal, with the average grade of 3.1. This observation is also confirmed.

We can conclude that all of observations were confirmed in the user study. Only observation 2 is less concrete because of the big spread of ratings in images 14.

Chapter 6

Conclusions and Future Work

In this thesis project we have implemented and analyzed two popular image inpainting algorithms and possible contributions. For the Bertalmio paper three contributions are found, these are using multi-resolution images, inpainting the unknown region inwards and estimating the amount of necessary iterations of the algorithm. These contributions all increase the speed of the algorithm.

Three contributions are also found for the Criminisi paper. These are local searching which increases the speed but should decrease the visual appeal, although in the experiments it increased the visual quality. Patch estimation increases the quality of the resulting images at the cost of a small speed decrease. The use of a lookup data structure increases the speed of the algorithm at the cost of a small quality drop.

Calculating MSE of the resulting images with the input image using a virtual mask is the only way of calculating the visual appeal of the resulting image found in literature. The user study verified the observation made throughout the project. The technique has it flaws, a low error value means a visual pleasing result but a high error value can result in both a visual pleasing and an unpleasing result. Another conclusion that can be made from the observation is that the MSE is not a correct way of comparing different image inpainting methods.

6.1 Future work

Future projects may consist out of extending the ideas from this thesis project from images to videos. Another idea for future work is the creation of a better quantitative evaluation method. The MSE is no good quality estimation in all cases. Creating the perfect method for this is impossible because it would be the same as image inpainting. During the project there were a lot of ideas that could not be done because of time limitations. Most of these ideas are on improving the Criminisi paper. These were creating patches if there is no exemplar patch under a certain error value. Using the information of image segmentation. For example in an image of the beach and the ocean, when searching for a patch on the beach, searching on the ocean would be unnecessary. Another idea that was investigated was patch shifting. Shifting the patch in the Criminisi paper in such a way that a certain amount of the patch is known. This will decrease the speed of the algorithm but will probably increase the quality. There was also no time to look into the vantage point data structure to improve the performance of the lookup data structures. Future work for the Sun paper may be the automatic creating support lines based on extrapolated lines which are recovered by using line segment detection. The analysis of other image inpainting algorithms and programs to come up with some other ideas, for example Photoshop and Remove. Remove is an commercial smart phone application designed to fast remove an object from an image based on a sequence of adjacent recorded images. The advertisement video of this application shows very promising results.

Chapter 7

Acknowledgments

First of all I would like to thank my supervisor dr. R.T. Tan for his important ideas and suggestions and for keeping me innovative and motivated throughout the whole master thesis project. I would also like to thank my family, my girlfriend and my friends for keeping me smiling and motivated. My thanks also goes to the other students who were also working on their Game and Media Technology master thesis project. Especially W. Saaltink, W.J. Spoel and M. Tomin, not only for the valuable discussions but also for the pleasant working environment.

Bibliography

[1] A. Criminisi, P. Pérez, and K. Toyama, "Region filling and object removal by exemplar-based image inpainting," *IEEE Transactions on Image Processing*, vol. 13, pp. 1200–1212, 2004.

[2] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, (New York, NY, USA), pp. 417–424, ACM Press/Addison-Wesley Publishing Co., 2000.

[3] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, (Washington, DC, USA), pp. 1033–, IEEE Computer Society, 1999.

[4] M. M. Oliveira, B. Bowen, R. Mckenna, and Y. sung Chang, "Fast digital image inpainting," in *Proceedings of the International Conference on Visualization, Imaging and Image Processing (VIIP 2001)*, pp. 261–266, ACTA Press, 2001.

[5] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," in *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, (New York, NY, USA), pp. 313–318, ACM, 2003.

[6] A. Telea, "An image inpainting technique based on the fast marching method," *Journal of Graphics, GPU, and Game Tools*, vol. 9, no. 1, pp. 23–34, 2004.

[7] L.-Y. Wei and M. Levoy, "Fast texture synthesis using tree-structured vector quantization," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, (New

York, NY, USA), pp. 479–488, ACM Press/Addison-Wesley Publishing Co., 2000.

[8] J. Sun, L. Yuan, J. Jia, and H.-Y. Shum, "Image completion with structure propagation," *ACM Trans. Graph.*, vol. 24, pp. 861–868, July 2005.

[9] J. Hays and A. A. Efros, "Scene completion using millions of photographs," *ACM Transactions on Graphics (SIGGRAPH 2007)*, vol. 26, no. 3, 2007.

[10] K. A. Patwardhan, G. Sapiro, and M. Bertalmo, "Video inpainting of occluding and occluded objects," in *International Conference on Image Processing*, pp. 69–72, 2005.

[11] Y. Wexler, E. Shechtman, and M. Irani, "Space-time video completion," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 1, pp. 120–127, 2004.

[12] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, pp. 629–639, July 1990.

[13] T.-H. Kwok, H. Sheung, and C. C. L. Wang, "Fast query for exemplar-based image completion," *IEEE Transactions on Image Processing.*, vol. 19, pp. 3106–3115, December 2010.

[14] Q. Chen, Y. Zhang, and Y. Liu, "Image inpainting with improved exemplar-based approach," in *Proceedings of the 2007 international conference on Multimedia content analysis and mining*, MCAM'07, (Berlin, Heidelberg), pp. 242–251, Springer-Verlag, 2007.

[15] Anupam, P. Goyal, and S. Diwakar, "Fast and enhanced algorithm for exemplar based image inpainting," *Image and Video Technology, Pacific-Rim Symposium on Image and Video Technology*, vol. 0, pp. 325–330, 2010.