Grain Reduction in Scanned Image Sequences under Time Constraints

Examensarbete utfört i Bildbehandling vid Tekniska högskolan i Linköping

av

Lina Stuhr

LiTH-ISY-EX--09/4203--SE

Linköping 2009
Grain Reduction in Scanned Image Sequences
under Time Constraints

Examensarbete utfört i Bildbehandling
vid Tekniska högskolan i Linköping
av

Lina Stuhr

LiTH-ISY-EX--09/4203--SE

Handledare: Klas Nordberg
ISY, Linköpings universitet

Thomas Hansson
Image Systems AB

Examinator: Klas Nordberg
ISY, Linköpings universitet

Linköping, 23 March, 2009
**Titel**

Grain Reduction in Scanned Image Sequences under Time Constraints

**Författare**

Lina Stuhr

**Abstract**

This thesis is about improving the image quality of image sequences scanned by the film scanner GoldenEye. Film grain is often seen as an artistic effect in film sequences but scanned images can be more grainy or noisy than the intention. To remove the grain and noise as well as sharpen the images a few known image enhancement methods have been implemented, tested and evaluated. An own idea of a thresholding method using the dyadic wavelet transform has also been tested. As benchmark has MATLAB been used but one method has also been implemented in C/C++.

Some of the methods works satisfactory when it comes to the image result but none of the methods works satisfactory when it comes to time consumption. To solve that a few speed up ideas are suggested in the end of the thesis. A method to correct the color of the sequences has also been suggested.

**Keywords**

image enhancement, digitalization, film grain, scanner
Abstract

This thesis is about improving the image quality of image sequences scanned by the film scanner GoldenEye. Film grain is often seen as an artistic effect in film sequences but scanned images can be more grainy or noisy than the intention. To remove the grain and noise as well as sharpen the images a few known image enhancement methods have been implemented, tested and evaluated. An own idea of a thresholding method using the dyadic wavelet transform has also been tested. As benchmark has MATLAB been used but one method has also been implemented in C/C++.

Some of the methods works satisfactory when it comes to the image result but none of the methods works satisfactory when it comes to time consumption. To solve that a few speed up ideas are suggested in the end of the thesis. A method to correct the color of the sequences has also been suggested.
Acknowledgments

First of all I would like to thank Image Systems AB for the opportunity to write this master thesis. At Image Systems AB I would especially like to thank my supervisor Thomas Hansson for his guidance. Also special thanks to my supervisor Klas Nordberg at Linköpings University for his many useful ideas and remarks on my report.

I am also grateful to Erika Emilsson and Elin Stuhr for proofreading my report. Finally I would like to thank all my friends and family for their love and support.
Contents

1 Introduction ................................................................. 1
  1.1 Background ............................................................ 1
  1.2 GoldenEye .............................................................. 1
    1.2.1 Software ......................................................... 2
    1.2.2 Lamp ............................................................. 2
    1.2.3 Optics ............................................................ 2
    1.2.4 Sensor ............................................................ 3
    1.2.5 Film .............................................................. 4
  1.3 Aim ................................................................. 4
  1.4 Outline ............................................................. 4

2 Theory ........................................................................... 7
  2.1 Noise, Grain and Sharpening ........................................... 7
    2.1.1 Low-pass Filter ................................................... 7
    2.1.2 Median Filter ..................................................... 8
    2.1.3 Wiener Filter ..................................................... 9
    2.1.4 Adaptive Filtering ............................................. 10
    2.1.5 Nonlinear Diffusion ........................................... 13
    2.1.6 Dyadic Wavelet Transform ................................... 14
  2.2 Color Correction ...................................................... 18
    2.2.1 Todays Color Correction in GoldenEye ....................... 19
    2.2.2 Another Color Correction Method ............................ 19

3 Results ........................................................................... 25
  3.1 Noise, Grain and Sharpening ........................................... 25
    3.1.1 Low-Pass Filter ................................................... 25
    3.1.2 Median Filter ..................................................... 32
    3.1.3 Wiener Filter ..................................................... 33
    3.1.4 Adaptive Filtering ............................................. 37
    3.1.5 Nonlinear Diffusion ........................................... 38
    3.1.6 Dyadic Wavelet Transform ................................... 40
  3.2 Color Correction ...................................................... 40
    3.2.1 Result ............................................................ 43
## 4 Further Development of Dyadic Wavelet Transform

4.1 Result ........................................................................ 45
4.2 Time Consumption ..................................................... 45

## 5 Conclusion and Future Work

5.1 Conclusion ................................................................. 49
5.2 Future Work .............................................................. 49

Bibliography ..................................................................... 51
Chapter 1

Introduction

This is a master thesis made at Image Systems AB in Linköping. This first chapter contains some background information, some information about the film scanner GoldenEye and the aim with the work and an outline for the thesis.

1.1 Background

For a couple of years Image System AB has developed scanners for digitalization of 16 and 35 mm analogue film mainly for the military and car industry. Now they have developed a new generation of scanners, GoldenEye, for the film industry.

Today the film industry needs a scanner since they shoot almost all films analogue, converts it to digital to work with it and then converts it back to analogue and sends it out to the cinemas all around the world. The reason for this procedure is that analogue film cameras still are cheaper than digitals and also since the industry already have a lot of analogue cameras which makes it to expensive too reinvest in only digital ones.

The scanner is also needed to convert old stored analogue film to digital. This to be able to save the film for the future since the stored film often decomposes when it gets old.

The film industry cares naturally more about the image quality than the military and car industry and Image Systems AB is therefore in need of an image enhancement algorithm.

1.2 GoldenEye

The GoldenEye is a film scanner for high-resolution digitalization of various film formats. It handles all major film types like Color print/negative, B/W print/negative and intermediate. The scanner is a line scanner which means that it scans one line of the film at a time. The sensor is a triple CCD, Charged Couple Device, sensor with 12 bit/color in dynamic range. The resolution of the scanned image can be either 2k or 4k, hence they can consist of either 2000 or 4000 lines. With
Introduction

the resolution 2k the frame rate is approximately 12 frames per second, fps, and for 4k about 3 fps. The system consists of a scanner and belonging software. A schematic image of the scanner can be seen in Figure 1.1. [3]

![Figure 1.1. GoldenEye, the film scanner developed by Image Systems for the film industry.](image)

1.2.1 Software

The software is written in C++ and the development environment is Visual Studio. GoldenEye already perform some automatic image corrections, see Table 1.1 for those. [3]

1.2.2 Lamp

The lamp is a halogen light bulb that shines through a line fiber and optical filters. Line illumination through optical fiber provides an even illumination across the image and a combination with an asymmetric diffuser makes the film illumination really smooth. [3] A halogen lamp provides light of a higher color temperature compared to a non-halogen incandescent lamp and has often also a longer life compared to a non-halogen incandescent lamp. [15]

1.2.3 Optics

GoldenEye uses a high quality apochromatic camera lens [3]. An apochromatic lens is a lens which is designed to bring three wavelengths into focus in the same plane. That give the apochromatic lens a better color correction than a regular achromatic lens which only brings two wavelengths into focus. Apochromatic lenses are also corrected for spherical aberration at two wavelengths unlike the achromatic lens that only correct at one wavelength. [15]
1.2 GoldenEye

<table>
<thead>
<tr>
<th>Correction:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gain</td>
<td>A basic light adjustment for each color are made.</td>
</tr>
<tr>
<td>Shading</td>
<td>A correction of the difference in light level across the film are made. Shading from the light source, fix pattern noise and nonlinearities in the camera are removed.</td>
</tr>
<tr>
<td>Film Adjustment</td>
<td>Calibrations to compensate for different film characteristics like film brightness, contrasts and gamma are done.</td>
</tr>
<tr>
<td>Color</td>
<td>Finally all colors are corrected towards a reference color matrix. This is normally done for the different film types used.</td>
</tr>
</tbody>
</table>

Table 1.1. The corrections already performed for GoldenEye

1.2.4 Sensor

The sensor in GoldenEye is a triple CCD sensor with a 12 bit/color in dynamic range [3]. A CCD sensor consist of sensor cells which consists of Si-photodiodes, that detects light (photons), accumulate an electric charge proportional to the light intensity and finally converts the charge into voltage. By sampling and digitizing the voltage an image can be stored, printed or displayed. The CCD sensor can give rise to many types of noise

**Thermal noise** or **dark current** means that the CCD sensor response in dark is not zero like it should be. That is due to the ability of the CCD sensors to generate charge in each pixel on its own with time and depending on the temperature. Thermal noise can be modeled as additive white, zero mean Gaussian noise [10] and is removed either by cooling the detector to extremely low temperatures or, in the software, by subtracting a dark frame. A dark frame is an image taken at a totally black object e.g. with the camera shutter closed. [4]

**Read out noise** is the noise that can emerge during sensor readout. It is produced by the A/D converter and it is easiest removed by subtraction of a bias frame. A bias frame is a readout image, read out without any exposure time. [4]

**Photon noise** or **Shot-noise** is unavoidable in an imaging system and is due to the fact that the total number of photons emitted by a steady source over any time interval varies and therefore also the charge collected by the CCD sensor varies. The variation is in both cases Poisson distributed. [10]
1.2.5 Film

GoldenEye can scan many different types of analogue film. Film contains tiny crystals of Silver Halide, which are sensitive to light. When the film crystals have received enough photons they are turned into tiny filaments of metallic silver in black-white images, and apart from the silver filaments also color dye clouds in color images, which forms the image. When the silver filaments or the dye clouds is clumped together it develop so called film grain. Film grain is often considered as an artistic effect but it can lead to problems like grain aliasing, which makes the scanned image even more grainy or noisy than the intention. [9] Film grain can be modeled in its limit as a Poisson process or as a Gaussian process [10]. Another problem with analogue film is that the color of the film is not always correct; especially old film can become a bit reddish.

1.3 Aim

Today GoldenEye is one of the fastest scanners at the market but it does not always give the best image result; the customers mainly complain about to much film grain and the colors. The main goal with this thesis is therefore to improve the image quality of the by GoldenEye scanned image sequences. The images need removing of unwanted grain as well as sharpening and color correction. The task is to

- Develop fast methods for image enhancement. The methods should not slower the scanner and must therefore have a speed of at least 12 fps for 2k respectively 3 fps for 4k images.
- Evaluate the methods regarding image result and computer effort or/and time consumption.
- Implement a well-functioning method in C/C++.

The first priority is to find a method to remove the grain and noise in the images as well as sharpen the images and the second priority is to correct the colors of the images.

1.4 Outline

Here follows the outline for the thesis.

Chapter 2 contains theory about known methods for image enhancement.

Chapter 3 contains the results for the known methods used at some different images.

Chapter 4 is about an idea of another thresholding method using the Dyadic Wavelet Transform.
Chapter 5 contains a conclusion of the results and ideas for further developments.
Chapter 2

Theory

This section contains theory of some known image enhancement methods that were tested.

2.1 Noise, Grain and Sharpening

The first steps was to remove grain and noise as well as sharpen the image. To do this a few filters were tested. All testing was performed in MATLAB. This section contains apart from theory about the methods tested also some comments on the MATLAB functions used.

2.1.1 Low-pass Filter

The most basic filter to start with is the low-pass filter. The low-pass filter is a spatial frequency filter which preserves low spatial frequencies and suppresses high frequencies. Noise represent a high-frequency image component. By decreasing the magnitudes of image frequencies of the noise, thus the high frequencies, the low-pass filter suppresses the noise. [12] The main problem with a low-pass filter is that it suppresses high frequencies phenomena like noise, lines, edges and the filter will hence apart from removing the noise also e.g. blur sharp edges. To filter involve a convolution with a kernel where the kernel is the 2-dimensional impulse response of the filter, see Figure 2.1.

![Figure 2.1. The moments of low-pass filtering.](image-url)

Image In → Convolution with Impulse Response → Image Out

Figure 2.1. The moments of low-pass filtering.
The filter is hence defined by its impulse response, \( h(x,y) \), or by its frequency transfer function, \( H(u,v) \). The impulse response for an ideal low-pass filter is

\[
h(x,y) = \sin \left( \frac{\omega_c (x,y)}{x,y} \pi \right).
\]  

(2.1)

where \( \omega_c \) is the cutoff frequency. The frequency transfer function for a low-pass filter in the Laplace domain is

\[
H(s) = \frac{K}{1 + \tau \cdot s},
\]

(2.2)

where \( K \) is a constant corresponding to the gain in the passband and \( \tau \) is a time constant corresponding to the cutoff or break frequency \( (\tau = 1/\omega_c) \) of the filter.

### 2.1.2 Median Filter

An often more useful filter than the low-pass filter, since it does not blur edges as much as the low-pass filter, is the median filter. A median filter is a non-linear rank-value filter that is useful when reducing salt-and-pepper noise, but less useful when reducing Gaussian noise [5]. The idea behind the filter is to replace a point in the image by the median of its neighborhood points. But to sort the neighbors for every pixel in a fairly large image could be expensive; to use a brute-force-solution leads to a complexity of \( O(r \log r) \), where \( r \) is the kernel radius. A more efficient approach has been developed by Huang [12] and that method leads to an order of \( O(r) \). The MATLAB function medfilt2 is based on Huang’s algorithm [14]. Recently even more efficient methods have been developed, specially the method developed by Perrault and Heberts [11] that have a complexity of \( O(\log r) \).

The algorithm for computing the efficient median filter by Huang can be seen in Algorithm 1 and the one by Perrault and Heberts can be seen in Algorithm 2.

---

**Algorithm 1** Huang’s median filtering algorithm

**Require:** Input Image \( X \) of size \( m \times n \)

**Require:** Output Image \( Y \) of the same size as \( X \)

**Require:** Initialize Kernel histogram \( H \)

for \( i = 1 \) to \( m \) do

for \( j = 1 \) to \( n \) do

for \( k = -r \) to \( -r \) do

Remove \( X_{i+k,j-r-1} \) from \( H \)

Add \( X_{i+k,j+r} \) to \( H \)

end for

\( Y_{i,j} \leftarrow \text{median}(H) \)

end for

end for

---

The main disadvantage with the median filter, apart from it being time consuming, is that the rectangular neighborhood of the kernel leads to damaging of
2.1 Noise, Grain and Sharpening

Algorithm 2: Perrault and Hebert’s median algorithm

Require: Input Image $X$ of size $m \times n$
Require: Output Image $Y$ of the same size as $X$
Require: Initialize Kernel histogram $H$ and column histograms $h_{1...n}$

for $i = 1$ to $m$ do
    for $j = 1$ to $n$ do
        Remove $X_{i-r-1,j+r}$ from $h_{j+r}$
        Add $X_{i+r,j+r}$ to $h_{j+r}$
        $H \leftarrow H + h_{j+r} - h_{j-r-1}$
        $Y_{i,j} \leftarrow \text{median}(H)$
    end for
end for

thin lines and sharp corners in the image. To avoid this another shape of neighborhood can be used, e.g. a cross with equal arms if there is vertical/horizontal lines in the image that needs to be preserved.

2.1.3 Wiener Filter

A Wiener filter or a minimum mean square error filter is based on the assumption that images and noise are random variables and the objective is to minimize the mean square error between the estimate of the image and the uncorrupted image. The filter can be used to filter out noise that has corrupted a stationary signal. The main problem with the Wiener filter is that it works unsatisfactory on images or image sequences that are non-stationary, which images often are [7]. Knutsson and Granlund [7] therefore suggest a local Wiener filter that is optimal for each neighborhood, but to compute such a Wiener filter for each position would need a large amount of computational effort.

The MATLAB function $\text{wiener2}$ [14] uses a low-pass filter to filter a grayscale image that has been degraded by constant power additive noise. The method uses a pixel-wise adaptive Wiener method based on statistics estimated from a local neighborhood of each pixel. The function estimates the local mean and variance around each pixel of the image $A$ as

$$\mu = \frac{1}{NM} \sum_{n_1,n_2 \in \eta} a(n_1, n_2) \tag{2.3}$$

and

$$\sigma^2 = \frac{1}{NM} \sum_{n_1,n_2 \in \eta} a^2(n_1, n_2) - \mu^2. \tag{2.4}$$

where $(n_1, n_2)$ are variables of spatial locations and $N$ and $M$ represent the number of local pixel neighbors that is taken into consideration. The function uses the estimates to create a pixel-wise Wiener filter as
\[ b(n_1, n_2) = \mu + \frac{\sigma^2 - \nu^2}{\sigma^2} (a(n_1, n_2) - \mu), \] 

where \( \nu^2 \) is the noise variance. If the noise variance is unknown the function uses an average of all the local estimated variances. [14]

### 2.1.4 Adaptive Filtering

Instead of a local Wiener filter Knutsson and Granlund [7] presents a more computationally efficient technique for obtaining an adaptive filter, with many of the wanted features of the local Wiener filter present. This filter is synthesized as a weighted summation of shift-invariant filter outputs by using adaptive weighting coefficients. The basis for control of the adaptive filter is the orientation tensor which is processed to produce a suitable control tensor. The main steps in the process can be seen in Figure 2.2. This method will work unsatisfactorily if performed at each channel of the RGB image, but the human vision has a low resolution for colors, it is instead the changes in intensity that controls our apprehension of how sharp a color image is. By transforming the image from the red-green-blue color space to for example the hue-saturation-value color space, eg. using the MATLAB function `rgb2hsv`, we do not lose any information for our apprehension of the image and the filtering method works more satisfactory [6].

![Figure 2.2. The moments of the adaptive filter.](image-url)

### Filter Optimization

The first step is to produce an optimal quadrature filter-set and an optimal enhancement filter-set. When it comes to filter optimization it is always a question of reconciling different demands. This is done by using the MATLAB function, `krnopt`, created at Linköpings University to be used for example at a laboration in medical image analysis.

The function `krnopt` takes the frequency ideal function, spatial ideal function, frequency weight function and spatial weight function as input variables and solves a weighted least square problem. The four variables must be into line with the characteristic of the image to get optimal filters. The frequency ideal function describes the ideal frequency function in the Fourier domain. It is chosen with a bandwidth and a center frequency that preserves interesting regions, eg. edges and lines, in the image. The frequency weight function is closely related to the spectrum and the signal-to-noise ratio of the image since it is important that the filter is good for the frequencies most common in the image. The purpose of the spatial weight function is to introduce a requirement of the spatial locality which...
improve the resolution as well as reduces the risk for interference of adjacent events in the image. Finally the spatial ideal function has the purpose to enhance the spatial resolution by the optimization and is the most compact spatial function there is. [6]

The quadrature filters are used to produce a orientation tensor and the enhancement filters are used to enhance the image. The number of quadrature filters needed are at least three in the 2D-case but four or more filters often gives an even better result. The filters should be evenly distributed over a half circle in the Fourier Domain. The eigenvalues in this directions need to be calculated to be able to later produce a tensor. [7] In MATLAB it can be expressed as

\[
\text{nr\_of\_filters=}4;
\text{filter\_directions}=[0 \, \text{pi/4} \, \text{pi/2} \, 3*\text{pi/4}];
\]

for k=1:length(filter\_directions)
    \[\text{eigen}\{k\} = [\sin(\text{filter\_directions}(k)); \cos(\text{filter\_directions}(k))];\]
end

The set of enhancement filters consists of one isotropic low-pass filter, LP, and anisotropic bandpass filters, BP, in the same directions as the Quadrature filters. The enhancement filters need to have the same frequency weight functions and directions as the quadrature filters but can have a different size in the spatial domain [6].

**Representation of Local Structure**

By using the filter responses of quadrature filters designed above, \(q_k\), the local structure in the image can be represented by a orientation tensor,

\[
\mathbf{T} = \begin{pmatrix} t_1 & t_2 \\ t_2 & t_3 \end{pmatrix} = \sum_k |q_k| \mathbf{M}_k,
\]  

(2.6)

where \(\mathbf{M}_k\) are the projection tensors in each direction

\[
\mathbf{M}_k = \frac{4}{3} \hat{\mathbf{n}}_k \hat{\mathbf{n}}_k^T - \frac{1}{3} \mathbf{I},
\]

(2.7)

where \(\hat{\mathbf{n}}_k\) are the filter vectors and \(\mathbf{I}\) is the identity matrix. [7]

The orientation tensor can be rewritten in terms of eigenvalues, \(\lambda_n\), and eigenvectors, \(e_n\), as

\[
\mathbf{T} = \begin{pmatrix} t_1 & t_2 \\ t_2 & t_3 \end{pmatrix} = \lambda_1 \mathbf{e}_1 \mathbf{e}_1^T + \lambda_2 \mathbf{e}_2 \mathbf{e}_2^T, \quad \lambda_1 > \lambda_2 > 0.
\]

(2.8)

The eigenvalues, \(\lambda_1\) and \(\lambda_2\), are for a second order 2D tensor calculated as

\[
\lambda_1 = \frac{1}{2} \left( t_1 + t_2 + \sqrt{(t_1 - t_3)^2 + 4t_2^2} \right)
\]

(2.9)

and
\[ \lambda_2 = \frac{1}{2} \left( t_1 + t_2 - \sqrt{(t_1 - t_3)^2 + 4t_2^2} \right). \]  
\[ (2.10) \]

The sum of \( \lambda_1 \) and \( \lambda_2 \) relates to how much energy the quadrature filters have detected and the relation between \( \lambda_1 \) and \( \lambda_2 \) relates to how much anisotropic energy that exist in the image [7].

**Local Adaptive Image Enhancement**

By using the orientation tensor, \( T \), a control tensor, \( C \), with the same eigensystem as the orientation tensor, can be calculated as

\[ C = \begin{pmatrix} c_1 & c_2 \\ c_2 & c_3 \end{pmatrix} = \gamma_1 e_1 e_1^T + \gamma_2 e_2 e_2^T, \]

where \( \gamma_1 \) and \( \gamma_2 \) are mappings of \( \lambda_1 \) and \( \lambda_2 \). The mapping is performed to provide the degrees of freedom that has been found useful in practice. The two functions that will be used for the point-by-point mapping of the orientation tensor are

\[ m(x_m, \sigma, \alpha, \beta, j) = \left( \frac{x^\beta}{x^{\beta+\alpha} + \sigma^\beta} \right)^{1/j} \]

and

\[ \mu(x_\mu, \alpha, \beta, j) = \left( \frac{(x (1 - \alpha))^{\beta}}{(x (1 - \alpha))^{\beta} + (\alpha (1 - x))^{\beta}} \right), \]

where \( x_m \) and \( x_\mu \) are the position-dependent variables which are \( x_m = \sqrt{\lambda_1^2 + \lambda_2^2} \) and \( x_\mu = \frac{\lambda_1}{\lambda_2} \). The variables \( \alpha \), \( \beta \) and \( n \) are constant parameters and \( \sigma \) is an estimate of the noise and can be constant or vary with position. The constant parameters \( \alpha \) and \( \beta \) controls the position and steepness of the passband of the filters. [6] The variables \( \gamma_1 \) and \( \gamma_2 \) are calculated as

\[ \gamma_1 = m(x, \sigma, \alpha, \beta, j) \]

and

\[ \gamma_2 = \gamma_1 \mu_2. \]

The \( m \)-function defines how much of the contents in the broadband of the bandpass filter that will be let through. The \( \mu \)-function defines how the contents of the bandpass filter will be mapped to each direction in the resulted image, thus how anisotropic the adaptive filter will be [7].

By using the control tensor, \( C \), and the projection tensor, \( M_k \), it is now possible to calculated the filter weights for the enhancement filters

\[ w_k = C \cdot M_k. \]

\[ (2.16) \]
The result image, $I_{Enh}$ can thereafter be calculated from the filter responses of the enhancements filter low-pass, $LP$, and bandpass, $BP_k$, parts as

$$I_{Enh} = LP + \alpha_{hp} \cdot \sum_k w_k \cdot BP_k,$$

(2.17)

where $\alpha_{hp}$ is a high-pass amplification factor [7].

### 2.1.5 Nonlinear Diffusion

The first nonlinear diffusion technique was described by Perona and Malik [8]. Their method encourages intraregion smoothing while inhibiting interregion smoothing. The main disadvantage with the Perona-Malik model is that it gives poor results for very noisy images.

**Definition**

The diffusion process by Perona and Malik is mathematically described as

$$\frac{\partial}{\partial t} I(x, y, t) = \nabla \cdot (c(x, y, t) \nabla I)$$

(2.18)

where $I(x, y, t)$ is the image, $(x, y)$ are variables of spatial locations, $t$ iteration steps and $c(x, y, t)$ is the so called diffusion function. The diffusion function can be chosen either as

$$c_1(x, y, t) = \exp \left( - \left( \left| \nabla I(x, y, t) \right| \kappa \right)^2 \right)$$

(2.19)

or as

$$c_2(x, y, t) = \frac{1}{1 + \left( \left| \nabla I(x, y, t) \right| \kappa \right)^2},$$

(2.20)

where $\kappa$ is referred to as the diffusion constant or the flow constant. The first diffusion function, equation (2.19), privileges high-contrast edges over low-contrast ones and the second, equation (2.20), privileges wide regions over smaller ones [1].

**Discrete Implementation**

The discrete implementation of the diffusion is unexpectedly straight forward [1]. First of all we rewrite equation (2.18) as

$$\frac{\partial}{\partial t} I(x, y, t) = \nabla \cdot (\Phi(x, y, t)),$$

(2.21)

where $\Phi_1$ and $\Phi_2$ are the so called flow functions which corresponds to the diffusion functions, $c_1$ and $c_2$. By using that

1. A gradient or derivate can be approximated as the difference in intensity between pixel neighbors in an image.
2. The flow function in equation (2.21) can be calculated independently for each of the pixel neighbors.

3. The filter is iterative, thus the right hand side of equation (2.18) describes the image intensity change by one iteration of the filter.

equation (2.18) can be rewritten as a discrete equation

\[
\frac{\partial}{\partial t} I(x,y,t) = \frac{\partial}{\partial x} \left[ c(x,y,t) \cdot \frac{\partial}{\partial x} I(x,y,t) \right] + \frac{\partial}{\partial y} \left[ c(x,y,t) \cdot \frac{\partial}{\partial y} I(x,y,t) \right]
\]

\[\approx \frac{1}{(\Delta x)^2} \left[ c \left( x + \frac{\Delta x}{2} , y, t \right) \cdot (I(x + \Delta x, y, t) - I(x, y, t)) \right]
\]

\[-\frac{1}{(\Delta x)^2} \left[ c \left( x - \frac{\Delta x}{2} , y, t \right) \cdot (I(x, y, t) - I(x - \Delta x, y, t)) \right], \quad (2.22)
\]

\[+ \frac{1}{(\Delta y)^2} \left[ c \left( x, y + \frac{\Delta y}{2} , t \right) \cdot (I(x, y + \Delta y, t) - I(x, y, t)) \right]
\]

\[-\frac{1}{(\Delta y)^2} \left[ c \left( x, y - \frac{\Delta y}{2} , t \right) \cdot (I(x, y, t) - I(x, y - \Delta y, t)) \right],
\]

\[= \Phi_{East} + \Phi_{West} + \Phi_{South} + \Phi_{North},
\]

with \(\Delta x = 1\) and \(\Delta y = 1\). The filtering is performed by updating each pixel in the image by an amount equal to the flow from four of its nearest neighbors

\[I(x,y,t + \Delta t) \approx I(x,y,t) + \Delta t \cdot (\Phi_{East} + \Phi_{West} + \Phi_{South} + \Phi_{North}). \quad (2.23)
\]

It is also possible to use the flow from eight of its nearest neighbors when updating each pixel

\[I(x,y,t + \Delta t) \approx I(x,y,t) + \Delta t \cdot [\Phi_E + \Phi_W + \Phi_S + \Phi_N
\]

\[+ \frac{1}{\Delta d^2} (\Phi_{NE} (\Delta d) + \Phi_{NW} (\Delta d) + \Phi_{SE} (\Delta d) + \Phi_{SW} (\Delta d))], \quad (2.24)
\]

where \(\Delta d\) is the distance to the pixel of interest, hence \(\Delta d = \sqrt{\Delta x + \Delta y} = \sqrt{2}\). A visualization of the process around a pixel, \(f(x,y,t)\), can be seen in Figure 2.3 [1].

### 2.1.6 Dyadic Wavelet Transform

Discrete Dyadic Wavelet Transform, discrete DWT, was first used by Mallat and Zhong and has been used for edge detection, texture analysis, noise reduction and image enhancement. Zong et. al. [2] have developed a method to reduce speckle and enhance contrast of echocardiograms using DWT. The idea is to decompose the image using a discrete DWT, threshold it and then reconstruct it again according to Figure 2.4.
2.1 Noise, Grain and Sharpening

Figure 2.3. Diffusion around a pixel, \(f(x, y, t)\).

Figure 2.4. The moments of the dyadic wavelet transform

DWT and IDWT

A finite level-level discrete DWT of a 2-dimensional discrete function \(f(m, n) \in \ell^2(Z^2)\) is represented as

\[
W[f(m, n)] = \left\{ (W_d^j[f(m, n)])_{d=1,2,1\leq j \leq J}, S_J[f(m, n)] \right\},
\]

(2.25)

where \(W_d^j[f(m, n)]\) is a wavelet coefficient at level \(j\) (or scale \(2^j\)), position \((m, n)\) and spatial orientation \(d\) (\(d = 1\) means horizontal orientation and \(d = 2\) means vertical orientation). The term \(S_J[f(m, n)]\) is a coarse scale approximation at the final level \(J\).

Mallat and Zhong have shown that for a particular class of 2-dimensional dyadic wavelets the finite-level direct and inverse discrete DWT can be represented by four filters, \(H, G, K\) and \(L\) [2]. The four filters should satisfy

\[
|H(\omega)|^2 + G(\omega)K(\omega) = 1
\]

(2.26)

and

\[
L(\omega) = \frac{1 + |H(\omega)|^2}{2}.
\]

(2.27)

The impulse responses of the filters \(H(\omega), G(\omega), K(\omega)\) and \(L(\omega)\) can be seen in Table 2.1.
Table 2.1. Impulse responses of the filters $H(\omega), G(\omega), K(\omega)$ and $L(\omega)$ which are used in the DWT decomposition and reconstruction.

<table>
<thead>
<tr>
<th>n</th>
<th>$h(n)$</th>
<th>$g(n)$</th>
<th>$k(n)$</th>
<th>$l(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td></td>
<td>0.001953125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td>-0.00390625</td>
<td>0.015625</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>0.0625</td>
<td>-0.03515625</td>
<td>0.0546875</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0.25</td>
<td>0.14453125</td>
<td>0.109375</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0.375</td>
<td>-1.0</td>
<td>-0.36328125</td>
<td>0.109375</td>
</tr>
<tr>
<td>1</td>
<td>0.25</td>
<td>0.375</td>
<td>0.109375</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.0625</td>
<td>0.14453125</td>
<td>0.0546875</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.03515625</td>
<td>0.015625</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.00390625</td>
<td>0.001953125</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In Figure 2.5 a 3-level DWT decomposition and reconstruction of a 2-dimensional function can be seen [2].

Figure 2.5. A 3-level DWT decomposition and reconstruction of a 2-dimensional function.

Noise Model, Log

The general model for speckle noise is

$$ f(x, y) = g(x, y) \eta_m(x, y) + \eta_a(x, y), \quad \text{(2.28)} $$

where $f(x, y)$ is the noisy observation of the noise-free image to be recovered ($g(x, y)$), $\eta_m$ and $\eta_a$ is the multiplicative respectively the additive noise and $(x, y)$
are variables of spatial locations, \((x, y) \in \mathbb{R}^2\). By assuming that the effect of the additive noise is considerably smaller than effect of the multiplicative noise, i.e.
\[
||\eta_a(x, y)||^2 \ll ||g(x, y) \eta_m(x, y)||^2,
\]
the model can be approximated to
\[
f(x, y) = g(x, y) \eta_m(x, y).
\] (2.29)

By taking a logarithmic transform on both sides of equation 2.29 the noise will be separated from the original noise free image, hence
\[
\log(f(x, y)) = \log(g(x, y)) + \log(\eta_m(x, y)),
\] (2.30)
which can be rewritten as
\[
f_l(x, y) = g_l(x, y) + \eta_l^m(x, y).
\] (2.31)

With uniform sampling and by using that a DWT is a linear transform the discrete version is obtained as
\[
W[f_l(m, n)] = W[g_l(m, n)] + W[\eta_l^m(x, y)],
\] (2.32)
where \((m, n) \in \mathbb{Z}^2\), \(f_l(m, n) = f_l(mT_x + s_x, nT_y + s_y)\). The variables \(T_x\) and \(s_x\) are the sampling period and the sampling shift along the horizontal direction and \(T_y\) and \(s_y\) the same along the vertical direction [2].

**Soft Thresholding, DeN**

Soft thresholding is performed to reduce noise. The operation can be expressed as
\[
u = T(v, t) = \text{sign}(v)(|v| - t)_+,
\] (2.33)
where \(u\) is the result of the soft threshold and has the same sign as the input \(v\), and the threshold parameter \(t\) is proportional to the noise level, \(\sigma\), in the image. The expression \((|v| - t)_+\) is defined as
\[
(|v| - t)_+ = \begin{cases} 
|v| - t, & \text{if } |v| > t, \\
0, & \text{otherwise.}
\end{cases}
\] (2.34)

The soft thresholded wavelet coefficients can be expressed as
\[
W^{d, *}_j[f(m, n)] = T(W^d_j[f(m, n)], t^d_j)
\] (2.35)
where \(d = 1, 2, j = 1, \ldots, k, k \leq J\) and \(t^d_j\) is a threshold related to the noise level orientation and scale which can be computed through a linear decreasing factor
\[
t^d_j = \begin{cases} 
(T_{\max} - \alpha(j - 1)) \sigma^d_j, & \text{if } T_{\max} - \alpha(j - 1) > T_{\min}, \\
T_{\min} \sigma^d_j, & \text{otherwise},
\end{cases}
\] (2.36)
where \(\sigma^d_j\) is the standard deviation (which if unknown can be the estimated noise level), \(\alpha\) is a decreasing factor between two levels, \(T_{\max}\) and \(T_{\min}\) the maximum respectively minimum factor for \(\sigma^d_j\), when \(1 \leq j \leq J\) and \(d \in \{1, 2\}\).
Hard Thresholding, Enh

To enhance edges in the images a generalized adaptive gain (GAG) factor is used. The enhancement-operator, $E_{GAG}$, is defined as

$$E_{GAG}(v) = \begin{cases} 0, & \text{if } |v| < T_1, \\ \text{sign}(v)T_2 + \bar{u}, & \text{if } T_2 \leq |v| < T_3, \\ v, & \text{otherwise}, \end{cases}$$  \hspace{1cm} (2.37)

where

$$\bar{u} = a(T_3 - T_2)\text{sign}(c(u - b)) - \text{sign}(-c(u + b)), \quad v \in [-1, 1],$$

$$u = \text{sign}(v)\frac{|v| - T_2}{T_3 - T_2}, \quad b \in (0, 1),$$

$$0 \leq T_1 \leq T_2 \leq T_3 \leq 1,$$

$$a = \frac{1}{\text{sign}(c(1 - b)) - \text{sign}(-c(1 + b))},$$

$$\text{sign}(x) = \frac{1}{1 + \exp(-x)}$$

and $c$ is a gain factor. The interval $[T_2, T_3]$ can be used to choose which feature in the image to enhance. The DWT coefficients after the hard thresholding is expressed as

$$W_{j,*}^{d,*}[f(m, n)] = M_j^d E_{GAG}\left(\frac{W_j^d[f(m, n)]}{M_j^d}\right)$$  \hspace{1cm} (2.38)

and

$$M_j^d = \max_{m,n}(|W_j^d[f(m, n)]|),$$

where position $(m, n) \in D$, $d = 1, 2$, $j \in \{k, \ldots, J\}$ and $1 \leq k \leq J$ [2].

2.2 Color Correction

Color is the perception of the electromagnetic radiation, waves between 400-700 nm, that reaches the human eye. If the colors of an image for some reason are not true it is said to have a color cast. Compare for example Figure 2.2 (a) and Figure 2.2 (b); Figure (a) is an image with true colors, the sky is blue and the snow is white, while Figure (b) is greenish and has hence not true colors. One reason for an image to get a color cast can for example be wrong illumination. A common way to describe a color cast is by stating the particular color predominant in the image, eg. “The sky appears to have a green color cast”. The process of eliminating unwanted color cast is called color correction.
2.2 Color Correction

Figure 2.6. An image, (a), with true colors and an image, (b), with a green color cast

2.2.1 Today's Color Correction in GoldenEye

Today GoldenEye carry out a color correction algorithm by first performing a gamma correction for each color channel and then use the method of least squares to calculate a correct color output from a known color input. Both steps are performed by using a look up table. This correction works satisfactory but the question is if there are any better methods to correct the color in the image. This section contains an idea of another color correction algorithm suggested by Strachen et. al. [13]

2.2.2 Another Color Correction Method

Strachen et. al. [13] have suggested a method of calibrating a video digitizing system using a Macbeth color chart in the CIE L*u*v* color space which can be useful for GoldenEye. The main points of the method are black and white levels correction, transformation from RGB to CIE L*u*v*, gamma correction and finally minimizing an overall average difference between the reference and the test values, see Figure 2.7.
MacBeth Color Chart

The MacBeth color chart is often used to calibrate systems. It contains 24 squares of color, each about 55mm. Six of the 24 squares are neutral patches ranging from white to black, see image 2.8. Eastman Kodak Company supplies a so called TAF, Telecine Analysis Film, that can be used in calibration purpose. The TAF consists of a neutral density scale and an eight-bar color test pattern with a LAD gray surround, see image 2.9.

Figure 2.8. Example of a MacBeth Color chart often used to calibrate systems.

Figure 2.9. Example of a Telecine Analysis Film supplied by Eastman Kodak Company.
2.2 Color Correction

Black and White Levels

The first step is to correct the black, $R(b), G(b)$ and $B(b)$, and white, $R(w), G(w)$ and $B(w)$, levels of the camera which can be obtained by lens capping and reference white tile. Knowing the black and white levels of the camera the red, green and blue, $R(A_n), G(A_n)$ and $B(A_n)$, channels for each of the $n$ colors of the chart can be corrected for the white and black levels through the following equations

$$R_n = \frac{R(A_n) - R(b)}{R(w) - R(b)} \times 100\%, \quad (2.39)$$

$$G_n = \frac{G(A_n) - G(b)}{G(w) - G(b)} \times 100\% \quad (2.40)$$

and

$$B_n = \frac{B(A_n) - B(b)}{B(w) - B(b)} \times 100\%. \quad (2.41)$$

RGB to CIE XYZ

The transformation from RGB to CIE $L^*u^*v^*$ is done in two steps. The first step is to transform the image from the RGB color space to the CIE XYZ color space. The chromaticity values for the primaries and the white point for a three tube color camera are given in Table 2.2.

<table>
<thead>
<tr>
<th>Color</th>
<th>Chromaticity Coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>$x_1 \ y_1 \ z_1$</td>
</tr>
<tr>
<td>Red</td>
<td>$x_2 \ y_2 \ z_2$</td>
</tr>
<tr>
<td>Blue</td>
<td>$x_3 \ y_3 \ z_3$</td>
</tr>
<tr>
<td>White</td>
<td>$x_0 \ y_0 \ z_0$</td>
</tr>
</tbody>
</table>

Table 2.2. The Primary color points and the white point

The white point values can be converted into tristimulus values with unity for the $Y$ value by dividing all three of the color chromaticity coordinates by $y_0$

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} x_0/y_0 \\ 1 \\ z_0/y_0 \end{bmatrix}, \quad (2.42)$$

where $a_1$, $a_2$ and $a_3$ are scaling factors. By adding $R$ units of red, $G$ units of green and $B$ units of blue the resulting $X$, $Y$ and $Z$ tristimulus values can be obtained from

$$\begin{bmatrix} a_1 x_1 & a_2 x_2 & a_3 x_3 \\ a_1 y_1 & a_2 y_2 & a_3 y_3 \\ a_1 z_1 & a_2 z_2 & a_3 z_3 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}. \quad (2.43)$$
The tristimulus value of Y is correlated to the intensity of the image.

**Gamma Correction**

It is necessary to perform a gamma correction of the output image to maintain a linear brightness relationship over the whole intensity range. Measuring the Y tristimulus values for the neutral patches of the test chart and plot them against their corresponding R, G and B values it is found that

\[
Y = R^{1/\gamma_R},
\]

(2.44)

where \(\gamma_R\) is the gamma correction for the red channel of the image

\[
\gamma_R = \frac{\log R}{\log Y}.
\]

(2.45)

The gamma correction factors for the green and the blue channels are calculated in the same way.

**CIE XYZ to CIE L\(u^*\)v\(*\)**

The second step in transforming the image from the RGB color space to the CIE \(L^*u^*v^*\) color space is the transform from the CIE XYZ to the CIE \(L^*u^*v^*\). To be able to do this transformation a transformation from the CIE XYZ to the CIE 1976 chromaticity coordinates \(u'\) and \(v'\) has to be done first

\[
u' = \frac{4X}{X + 15Y + 3Z}
\]

(2.46)

and

\[
v' = \frac{9Y}{X + 15Y + 3Z}.
\]

(2.47)

Thereafter the CIE \(L^*u^*v^*\) color space parameters can be calculated as

\[
L^* = \begin{cases} 
116.0 \frac{Y}{Y_0} - 16.0 & \text{if} \frac{Y}{Y_0} > 0.008856 \\
903.3 \left( \frac{Y}{Y_0} \right)^{1/3} & \text{if} \frac{Y}{Y_0} \leq 0.008856
\end{cases},
\]

(2.48)

\[
u^* = 13.0L^* (u' - u'_0)
\]

(2.49)

and

\[
v^* = 13.0L^* (v' - v'_0),
\]

(2.50)

where \(Y'_0, u'_0\) and \(v'_0\) are the coordinates for the reference white.
2.2 Color Correction

Average Difference

The CIE L*u*v color difference formula is defined as

\[
\Delta E_n^* = \sqrt{\left(\Delta L_n^*\right)^2 + \left(\Delta u_n^*\right)^2 + \left(\Delta v_n^*\right)^2},
\]

(2.51)

where

\[
\Delta L_n^* = L_n^* - L_{n0}^*,
\]

(2.52)

\[
\Delta u_n^* = u_n^* - u_{n0}^*,
\]

(2.53)

and

\[
\Delta v_n^* = v_n^* - v_{n0}^*,
\]

(2.54)

where \((L_{n0}^*, u_{n0}^*, v_{n0}^*)\) are the reference values which be seen in Table 2.3. The reference values have been measured under the CIE standard tungsten illuminant A [13].

<table>
<thead>
<tr>
<th>No</th>
<th>Color</th>
<th>(L_{n0}^*)</th>
<th>(u_{n0}^*)</th>
<th>(v_{n0}^*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yellow</td>
<td>80.25</td>
<td>22.20</td>
<td>29.45</td>
</tr>
<tr>
<td>2</td>
<td>Cyan</td>
<td>52.26</td>
<td>-48.74</td>
<td>-10.16</td>
</tr>
<tr>
<td>3</td>
<td>Green</td>
<td>54.45</td>
<td>-48.00</td>
<td>22.85</td>
</tr>
<tr>
<td>4</td>
<td>Magenta</td>
<td>52.03</td>
<td>76.75</td>
<td>-20.11</td>
</tr>
<tr>
<td>5</td>
<td>Red</td>
<td>41.69</td>
<td>97.94</td>
<td>-2.61</td>
</tr>
<tr>
<td>6</td>
<td>Blue</td>
<td>31.80</td>
<td>3.45</td>
<td>-34.16</td>
</tr>
</tbody>
</table>

Table 2.3. The reference values for the MacBeth colors measured under illuminant A

Using equation 2.51 the overall average difference can be calculated as

\[
\Delta E_{avg}^* = \frac{1}{N} \sum_{n=1}^{N} \Delta E_n^*,
\]

(2.55)

where \(N\) is the number of non-neutral colors.

Minimization of the Average Difference

The optimizing procedure to minimize \(\Delta E_{avg}^*\) is done by deriving the elements of the \(3 \times 3\) in equation (2.43). The nine elements of equation (2.43) is reduced to six independent and three dependent elements using equation (2.42), thus

\[
\begin{bmatrix}
  a_1 x_1 & a_2 x_2 & a_3 x_3 \\
  a_1 y_1 & a_2 y_2 & a_3 y_3 \\
  a_1 z_1 & a_2 z_2 & a_3 z_3
\end{bmatrix}
\begin{bmatrix}
  a_1 x_1 & a_2 x_2 & \frac{x_0}{y_0} - a_1 x_1 - a_2 x_2 \\
  a_1 y_1 & a_2 y_2 & 1 - a_1 y_1 - a_2 y_2 \\
  a_1 z_1 & a_2 z_2 & \frac{z_0}{y_0} - a_1 z_1 - a_2 z_2
\end{bmatrix}.
\]

(2.56)
The white point of the xyz chromaticity points is given by $x_0 = 0.44757$, $y_0 = 0.40745$ and $z_0 = 0.14498$ for illuminant A. The six independent variables can now be used in a standard optimization algorithm, for example the simplex algorithm, to minimize $\Delta E_{avg}^*$. 
Chapter 3

Results

This chapter contains the results of the different methods described in chapter 2. The result for each method is divided into a section about the result in the images and a section about how time consuming the method is. The time has been measured by using MATLAB’s stopwatch timer tic, toc or by using MATLAB’s function profile that profiles execution time for functions. The images used as test images can be seen in Figure 3.1, 3.3, 3.5 and 3.7. The first image, Figure 3.1, contains almost no grain, the second, Figure 3.3, contains a little bit more grain and the third, Figure 3.5, is really grainy. The fourth image, Figure 3.7, is a so called TAF-image with different color patches which will be used in the color correction part. In Figure 3.2, 3.4 and 3.6 enlargements of a part of the original images in Figure 3.1, 3.3 and 3.5 can be seen.

To be able to compare how time efficient a method is the time has been measured using the same image and the same computer. The image used is the 1080 × 1920 pixels large image that depicts a clown, see Figure 3.3. The computer is a PC with Microsoft Windows XP and 2.0 GB in RAM. The CPU is a 2.67 GHz Intel Core 2. All implementation in MATLAB and C/C++ has been done using standard functions and standard libraries.

3.1 Noise, Grain and Sharpening

The first priority was to remove grain as well as sharpen the image. The results of the methods tested for grain removal and sharpening follows here.

3.1.1 Low-Pass Filter

The first filter to implement was the low-pass filter. The filter was implemented both by using MATLAB and by using an own filter kernel. MATLAB’s function fspecial creates a Gaussian low-pass filter of size hsize with standard deviation sigma

\[
\text{filter} = \text{fspecial}('gaussian', \text{hsize}, \text{sigma});
\]
Figure 3.1. Original image
An example of a chosen kernel of size $3 \times 3$ is instead

\[
\text{filter\_kernel}=[1 \ 1 \ 1; \ 1 \ 2 \ 1; \ 1 \ 1 \ 1]*(1/10);
\]

**Image Result**

The result when using MATLAB’s kernel with a size of $11 \times 11$ and sigma of 0.7 can be seen in Figure 3.8. A little bit grain has been removed but the low-pass filtered images have as expected also lost in sharpness.

A smaller size of the kernel leads to a sharper image than the large kernel but a smaller kernel does not remove as much grain as the larger one does. The result with a smaller kernel size, $3 \times 3$, with the same sigma, $\sigma = 0.7$, can be seen in Figure 3.9.

The chosen kernel above leads to a result which can be seen in Figure 3.10. It can be seen that the gaussian low-pass filter of size $3 \times 3$ created with MATLAB gives the same result as the low-pass filter chosen above.

**Time Consumption**

The elapsed time for the low-pass filter, using both MATLAB’s kernel and an own predefined kernel, can be seen in Table 3.1. It is established that the larger the kernel size is the slower the method is.
Figure 3.3. Original image

Figure 3.4. A part of the Original image in Figure 3.3 enlarged.
3.1 Noise, Grain and Sharpening

Figure 3.5. Original image

Figure 3.6. A part of the Original image in Figure 3.5 enlarged, note the graininess.
Figure 3.7. Original image

Figure 3.8. Low-pass filtered image by using the MATLAB’s kernel with kernelsize $11 \times 11$
3.1 Noise, Grain and Sharpening

Figure 3.9. Low-pass filtered image by using the MATLAB’s kernel with kernel size $3 \times 3$.

Figure 3.10. Low-pass filtered image by using a chosen kernel of size $3 \times 3$.

<table>
<thead>
<tr>
<th>Kernel size</th>
<th>Method</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3 \times 3$</td>
<td>fspecial</td>
<td>0.322 s</td>
</tr>
<tr>
<td>$11 \times 11$</td>
<td>fspecial</td>
<td>1.200 s</td>
</tr>
<tr>
<td>$3 \times 3$</td>
<td>predefined kernel</td>
<td>0.345 s</td>
</tr>
<tr>
<td>$11 \times 11$</td>
<td>predefined kernel</td>
<td>1.706 s</td>
</tr>
</tbody>
</table>

Table 3.1. Time consumption for the low-pass filter, using both a kernel created by `fspecial` in MATLAB and an own predefined kernel.
3.1.2 Median Filter

To see what the median filter could do for the image quality the MATLAB function `medfilt2` was tested first.

**Image Result**

The result of a $3 \times 3$ median filter can be seen in Figure 3.11. The median filter removes some grain but it also, as expected, blurs the image. This is even clearer in Figure 3.12, which is a median filtered image with the larger kernel size $11 \times 11$.

![Figure 3.11: Median filtered image with a filter size of $3 \times 3$.](image1)

![Figure 3.12: Median filtered image with a filter size of $11 \times 11$.](image2)
3.1 Noise, Grain and Sharpening

Time Consumption

The time consumption for the median filter using the function medfilt2 and different kernel sizes can be seen in Table 3.2. Using the brute-force method instead takes a couple of minutes when the filter size is $3 \times 3$.

<table>
<thead>
<tr>
<th>Kernel size</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3 \times 3$</td>
<td>0.735 s</td>
</tr>
<tr>
<td>$11 \times 11$</td>
<td>0.822 s</td>
</tr>
<tr>
<td>$27 \times 27$</td>
<td>1.519 s</td>
</tr>
</tbody>
</table>

Table 3.2. Time consumption for the median filter using MATLAB’s medfilt2.

3.1.3 Wiener Filter

The wiener filter was tested using MATLAB’s wiener2. As said in section 2.1.3 the function can take a known noise variance as well as an unknown noise variance as in parameter.

\[
\text{Image\_Out} = \text{wiener2(Image\_in, [size\_m size\_n], est\_noise\_variance)};
\]

\[
[\text{Image\_Out, noise\_variance}] = \text{wiener2(Image\_in, [size\_m size\_n])};
\]

The noise variance in this case is unknown for the image sequences. But by using the wiener2-function without any noise variance on a fairly large homogen colored square in an image, an estimate of the noise variance is obtained as the average of all the local estimated variances in the square. This estimate can then be used as input to wiener2. The idea by using a homogen colored square to estimate the noise is that the square then almost only contains grain, hence the estimated noise variance correspond to the grain characteristic.

Image Result

In Figures 3.13 - 3.15 the result of the Wiener filtered “clown-image” is shown. Figure 3.13 (a) and 3.13 (b) show Wiener-filtered images with the kernel sizes $3 \times 3$ both using the method with the square to estimate noise variance, Subfigure (a) and an unknown noise variance, Subfigure (b). Figure 3.14 (a) and 3.14 (b) show the same wiener-filtered images but now with a larger kernel size of $11 \times 11$ and finally Figure 3.15 (a) and 3.15 (b) also the same images but with the filter size $27 \times 27$. It is clearly that by using a square to estimate the noise variance more grain is removed and the image get less blurred.

The estimated noise variance can be seen in Table 3.3. The blue channel always has the highest noise variance and is also the channel that contains the most grain, which is due to a chemical process when making the film.
Table 3.3. The by the function \texttt{wiener2} estimated noise variance, $\nu^2$, for the Red, Green and Blue Channel.
3.1 Noise, Grain and Sharpening

(a) Noise variance estimated from square

(b) Unknown noise variance

Figure 3.14. Wiener filtered image with kernel size $11 \times 11$ with estimated noise variance and unknown noise variance.
(a) Noise variance estimated from square

(b) Unknown noise variance

**Figure 3.15.** Wiener filtered image with kernel size $27 \times 27$ with estimated noise variance and unknown noise variance.
3.1 Noise, Grain and Sharpening

Time Consumption

The time it takes for the wiener-filter can be seen in Table 3.4. Since it takes time to first estimate the noise variance, the method with the square is a little bit slower than the method with unknown noise variance.

<table>
<thead>
<tr>
<th>Kernel size</th>
<th>Method</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 3</td>
<td>Unknown Noise</td>
<td>2.204 s</td>
</tr>
<tr>
<td>3 × 3</td>
<td>Estimated Noise</td>
<td>2.299 s</td>
</tr>
<tr>
<td>11 × 11</td>
<td>Unknown Noise</td>
<td>3.804 s</td>
</tr>
<tr>
<td>11 × 11</td>
<td>Estimated Noise</td>
<td>3.942 s</td>
</tr>
<tr>
<td>27 × 27</td>
<td>Unknown Noise</td>
<td>7.145 s</td>
</tr>
<tr>
<td>27 × 27</td>
<td>Estimated Noise</td>
<td>7.240 s</td>
</tr>
</tbody>
</table>

**Table 3.4.** Time consumption for the Wiener filter.

3.1.4 Adaptive Filtering

The adaptive filtering was performed after a laboration instruction performed at IMT at Linköpings University [6].

Image Result

The result of the adaptive filtering can be seen in Figure 3.16. The edges are obviously enhanced but some of the grain is also enhanced which is not desirable.

**Figure 3.16.** The result of the adaptive filtering
The main problem with two dimensional adaptive filtering and image sequences is that a moving object in the sequence, e.g. a ball, can look different with different backgrounds.

### Time Consumption

The time consumption for the adaptive filtering can be seen in Table 3.5

<table>
<thead>
<tr>
<th>Filter size</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 × 7</td>
<td>20.993 s</td>
</tr>
<tr>
<td>9 × 9</td>
<td>23.702 s</td>
</tr>
<tr>
<td>11 × 11</td>
<td>25.480 s</td>
</tr>
</tbody>
</table>

**Table 3.5.** Time consumption for adaptive filtering

#### 3.1.5 Nonlinear Diffusion

Nonlinear diffusion can be done either at each color channel or on the whole image by using combined diffusion coefficients. The combined diffusion coefficients is calculated by replacing the gradient magnitude, $|\nabla I(x, y, t)|$, in equation 2.19 and 2.20 with $\sqrt{I_{\text{Red}}^2(x, y, t) + I_{\text{Green}}^2(x, y, t) + I_{\text{Blue}}^2(x, y, t)}$ [1].

### Image Result

The results from making the nonlinear diffusion by Perona and Malik, channel by channel, can be seen in Figure 3.17. Subfigure 3.17 (a) is after one iteration, Subfigure 3.17 (b) is after five iterations and Subfigure 3.17(c) is after ten iterations. It can be seen that the more iterations the more grain is removed but the image also becomes more unsharp and some small lines are also removed.

The result when taking the diffusion on the whole image at once is very alike the one taking it channel by channel. The main problem with diffusion is the same as mention with adaptive filtering; at image sequences it can lead to that a moving object in the sequence, e.g. a bird or a ball, looks different with different backgrounds.

### Time Consumption

The time consumption per iteration by doing the Perona-Malik diffusion channel by channel on the “clown-image” can be seen in Table 3.6.

The time consumption per iteration when making the Perona-Malik diffusion at the whole image can be seen in Table 3.7.

It is thus faster to do the diffusion channel by channel.
3.1 Noise, Grain and Sharpening

(a) One iteration of the Perona-Malik diffusion

(b) Five iterations of the Perona-Malik diffusion

(c) Ten iterations of the Perona-Malik diffusion

Figure 3.17. Result of the Perona-Malik diffusion after one, five and ten iterations.
Table 3.6. Time consumption for nonlinear diffusion by Perona and Malik, per iteration

<table>
<thead>
<tr>
<th>Number of Neighbors</th>
<th>Diffusion function</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>equation 2.19</td>
<td>2.295 s</td>
</tr>
<tr>
<td>4</td>
<td>equation 2.20</td>
<td>2.170 s</td>
</tr>
<tr>
<td>8</td>
<td>equation 2.19</td>
<td>4.722 s</td>
</tr>
<tr>
<td>8</td>
<td>equation 2.20</td>
<td>4.440 s</td>
</tr>
</tbody>
</table>

Table 3.7. Time consumption for multichannel nonlinear diffusion by Perona and Malik, per iteration

<table>
<thead>
<tr>
<th>Number of Neighbors</th>
<th>Diffusion function</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>equation 2.19</td>
<td>4.364 s</td>
</tr>
<tr>
<td>4</td>
<td>equation 2.20</td>
<td>4.118 s</td>
</tr>
<tr>
<td>8</td>
<td>equation 2.19</td>
<td>6.467 s</td>
</tr>
<tr>
<td>8</td>
<td>equation 2.20</td>
<td>6.228 s</td>
</tr>
</tbody>
</table>

3.1.6 Dyadic Wavelet Transform

The DWT was tested both by using an algorithm developed in MATLAB as well as one in C using a mex-file in MATLAB.

Image Result

The results from the DWT can be seen in Figure 3.18 and 3.19 for one-level. In Figure 3.18 it is clearly that the image is shaper than the original in Figure 3.1, look at the text on the bottle for example. Figure 3.19 shows that the method also removes grain pretty well, look for example on the backside of the white road sign. The problem with the method is however that the thresholds contains a lot of parameters and therefore it is hard to find an optimal setting for each image; it is a fine balance between grain removing and sharpening, see for example Figure 3.20.

Time Consumption

The time consumption for the DWT and the thresholding in MATLAB can be seen in Table 3.8 for one, two and three levels of decomposition and reconstruction. The times in the table can be compared with C version developed that takes 3.919 seconds for one level of decomposition and reconstruction.

3.2 Color Correction

Since the work with removing grain and sharpening took longer time than calculated the color correction part has been suffering. Parts of the algorithm by Strachen et. al. has been tested but not the whole algorithm.
3.2 Color Correction

**Figure 3.18.** Result after the dyadic wavelet transform, compare with Figure 3.2 (Look for example at the text).

**Figure 3.19.** Result after the dyadic wavelet transform, compare with Figure 3.6.
Figure 3.20. Result after the dyadic wavelet transform.

Table 3.8. Time consumption for the Dyadic Wavelet Transform in MATLAB.

<table>
<thead>
<tr>
<th>Levels</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.772 s</td>
</tr>
<tr>
<td>2</td>
<td>18.890 s</td>
</tr>
<tr>
<td>3</td>
<td>28.106 s</td>
</tr>
</tbody>
</table>
3.2 Color Correction

3.2.1 Result

The result of the correction of black- and white-levels can be seen in Figure 3.21. It can be seen in the figure that the correction of the black and white levels makes the images brighter; the white box is now white and not gray as it was before the correction.

Figure 3.21. Black and white levels of the image corrected
Chapter 4

Further Development of Dyadic Wavelet Transform

Since the parameters in the Dyadic Wavelet Transform were hard to optimize we got an idea of making an easier threshold. By looking at the histogram of the 1-level wavelet transformed image we tried to localize which value coefficients that contained the grain and which did not. In Figure 4.1 three histograms can be seen; the two above of a square with grain and the lower of the whole image in Figure 3.3. The two grain histograms clearly have a special characteristic. Figure 4.2 represent the same three histograms after a try to remove most of the frequencies containing grain. Figure 4.3 contains the same histogram after removing the grain frequencies and doing the hard thresholding as described in chapter 2.1.6.

4.1 Result

The result of the idea can be seen in Figure 4.4. The new method has removed some grain but the image still gets a bit blurred.

4.2 Time Consumption

Since this method needs more preprocessing than the ordinary DWT suggested in chapter 2.1.5, because it is necessary to know where the grain is, it takes more time; but without the preprocessing the time consumption is about the same.
Figure 4.1. Histogram of two squares of grain and the whole image before any thresholding.
Figure 4.2. Histogram of two squares of grain and the whole image after trying to remove the grain frequencies.
- **Figure 4.3.** Histogram of two squares of grain and the whole image after the hard threshold.

- **Figure 4.4.** Image after the DWT with a different threshold.
Chapter 5

Conclusion and Future Work

This chapter contains a conclusion of the results and proposals for some improvements to think about in the future.

5.1 Conclusion

The goal with this thesis was to test and implement a fast method to improve the image quality of the in GoldenEye scanned image sequences. A few methods to remove grain as well as sharpen the image sequences has been tested and one method, the DWT, has been implemented in C/C++. A method to correct the colors of the images has also been suggested but this method has not been thoroughly tested due to lack of time.

Some of the methods tested removes grain satisfactory as well as sharpen the image. But none of the methods manage to work up 12 images per second in MATLAB. The method that is closest is the small low-pass filter which do not give a sufficient image result, see Figure 3.9 and Figure 3.10; the images become less grainy but as expected they also become blurred.

The method with the most satisfactory image result was the DWT, see Figure 3.18 and 3.19, but this method does neither in MATLAB nor in C/C++ follow the time demand, see Table 3.8 compared with GoldensEye’s 12 fps for 2k. When comparing Figure 3.20 and Figure 4.4 it seems like the own threshold idea suggested in chapter 4 is better than the original DWT. That is a bit misleading since Figure 3.20 shows the result of a badly adjusted DWT. The true image is that the results are comparable.

5.2 Future Work

The most important future work when it comes to grain removing and sharpening is to speed up an otherwise good method. The Dyadic Wavelet transform gives a good image result but its time consumption must be reduced. A speed up of the DWT can be done by e.g.
• Parallelizing the code for multiprocessor hardware. Today most PC’s have at least two cores, to utilize them in our case there are some obvious opportunities for parallelism. Since the three RGB channels of the image are independent of each other, each channel can be calculated in parallel on different cores. Looking at Figure 2.5 it can be seen that the filters are separable in x- and y-direction and therefore the filtrations in the decomposition and reconstruction can be done in parallel. If the hardware being used supports simd-instructions, single instruction multiple data, it can be a possible way to exploit parallelism.

• Implement the code in hardware. Since hardware often is faster than software an idea is to implement the DWT on an FPGA, field-programmable gate array, for example. To do that the code has to be transformed to a hardware description language, for example VHDL. In MATLAB there is a special toolbox for transforming digital filters to VHDL [14], which could be used in this case.

• Try to optimize the code. The code today is implemented using standard libraries in C/C++. An idea is to research if there is any special libraries for fast implementations.

To avoid the problem mentioned in Chapter 3.1.5 and 3.1.4 with the adaptive filtering and nonlinear diffusion the three dimensional versions of the two methods can be tested.

The color correction method needs more work. The method suggested has to be fully implemented and tested.
Bibliography


