Saarland University

Faculty of Natural Sciences and Technology I

Department of Computer Science

Bachelorthesis

A PDE formalization of Retinex theory

submitted by

Mykola Byelytskyy

submitted

16.09.2014

Reviewers:

Dr. Joachim Weickert

Dr. Karol Myszkowski

Contents

1. Introduction

1.1 The Retinex theory: motivation and applications

If we take a camera and make a picture of illuminated street in the evening then, even if this photo is automatically corrected by the camera electronics, the result will look different from what we can see around us. The most appreciable difference on such photos is that the street illumination looks totally unnatural, mostly with strong orange or green tone. This effect can be explained with spectral properties of lamps used in street lightning. This lamps contain peaks in orange or green part of the spectrum. But as mentioned above, if we look around us the colors appear much more natural without strong biases in orange or green bands. Besides, the street lightning looks normally much whiter than on the pictures. Thus, we can observe the first strong difference between the human visual system (HVS) and an electronic device that can also detect light in visible spectrum. The HVS has much better abilities to adapt itself to the properties of environment illumination, for instance, to a non-white spectrum of the street lightning.

Further if we take a picture of some scene one part of which is good illuminated and the other part lies in shadow, then the shadows on the photo will look much darker and the light areas much brighter than we can see it. Despite the fact that the camera can make some sort of brightness or contrast correction, it is normally very difficult to make the details in shadows and within bright areas to be equally good visible. At the same time our eyes seem to have no problem with it. We can much better distinguish details in shadows and on the light simultaneously. Thus, we have shown the other feature of our visual system – the much wider dynamic range that our eyes are capable to perceive. We can simultaneously distinguish details in dark areas as well as in bright areas.

We can also deliberately alter the external illumination in such a way that the same object will have different colors in the sense of RGB values. But often, despite the fact of significantly different RGB numbers, we can still define at least the name of the original color of this object. Furthermore, in some other scene another object with the same RGB value can appear for the HVS as an object with totally different color. So our visual system defines a color not only on the basis of RGB values of its points, but in the context of the whole scene. The HVS always tries to define the original color of an object eliminating the effects of non-uniform or non white external illumination. This feature is called the Color Constancy.

All these features of the human visual system are very important and could be used in many different areas of digital image analysis. For instance, in the field of object recognition the software would be very helpful to remove all environmental effects such as reflections and non-uniform illumination to keep only the original visual characteristics of the objects. Or it would be possible to find more effective solutions for color balancing and dynamic range extension problems in the field of digital photography. Texture processing software could be also improved with good algorithms that can clear a texture from shadows, reflexes and other effects of external illumination.

Thus, there is a strong motivation to develop such algorithms that would be able to remove the effects of environment and restore the original color properties of image objects, similar to the HVS. One of the first set of papers in that direction were papers by Land and McCann [1]-[3]. They have developed an approach that simulates in certain way some properties of our HVS. This algorithm was called “Retinex”. This word was derived from “retina” and “cortex”, highlighting the assumption that our HVS mainly consists of these two important parts.

1.2 Overview of related work and different Retinex implementations.

As was mentioned above one of the first attempts to understand and mathematically simulate the way in which the HVS processes visual information was made by E. Land. In cooperation with McCann he has defined the starting points for developing a mathematical model of the human vision system [1]-[3] called Retinex. This basis includes the idea of logarithmical intensity ratios computation, reset mechanism and general observations about the properties of the HVS. I will describe these ideas more detailed below.

A set of Retinex variations was developed on the basis of the Land's and McCann ideas and a few further approaches. This set can be separated on several subsets. Each of these subsets uses the one of those approaches or their compilation.

The first subset consists of algorithms that use the original Land's and McCann approach to the lightness calculation, averaging it over a set of paths. But in the original papers of Land and McCann these paths are not precisely defined so there is a huge possibility for different variations. The reference [4] can be mentioned as an example of path based Retinex algorithms where brownian particle motion adaptated for using as Retinex paths. Or paper [7] from Provenzi *et al.*, where the authors are trying to predict the qualitative behavior of the original Retinex algorithm with respect to parameters used (number of paths, used threshold, number of iterations etc.). The paper from Morel [5] considered below is some sort of path-based algorithms as well.

Another subset of Retinex algorithms consists of physic-based approaches. One of the first papers in this group of algorithms is the paper by Horn [8] where the author uses a poisson equation with hard thresholded Laplace operator of logarithmic image intensity as the right side of the equation. The idea of such an approach is based on the fact that the environmental multiplicative effect such as shadows, have much softer gradients, opposing to the scene objects with contrast edges. So the hard thresholding operator can remove these shadows. In [5] the authors also derive a form of the Poisson equation but with different thresholding approach.

There is also a set of algorithms called “center-surround” Retinex. The main idea of this approach is to calculate a difference between the light intensity in a current point and averaged intensity of its neighbors [9]. In such way the low frequency components that should define shadows will be suppressed. An extension of this idea is so called single- and multi-scale center-surround Retinex [10]. The single-scale Retinex in this context is a modification of above idea defining the averaging method as a gaussian blur. The multi-scale Retinex is a combination of the single-scale processions with different scaling factors.

Besides of these approaches there are a sort of frameworks for Retinex algorithms. A good example is the paper by Dominique Zosso [16] where many different Retinex approaches were reformulated as a variational problems and presented within one framework.

1.3 Outline.

The main paper that I use in this work is the article by Morell [5]. I will start trying to understand a little bit deeper the theoretical basis that was used to construct described in [5] PDEs for extrema and resetless Retinex because in the original article some details were reduced (Chapter 3). Furthermore, I will implement the numerical solution of the both equations with two different methods (Chapter 5). For the resetless equation fast cosine transformation will be used and for the extrema Retinex – the Gauss-Seidel method. It should be mentioned, that the idea of using digital Fourier transformation and Gauss-Seidel method was proposed by the authors of the paper [5]. Besides, their own version of the numerical resetless Retinex PDE solution can be downloaded from the web site mentioned in the article [5]. In this software authors use FFTW library for discrete cosine transformation. In my implementation I use explicit realization of a fast cosine transformation algorithm instead. I have not found any implementation of the extrema Retinex algorithm made by the authors of the article [5] thus I have only used the authors proposition to solve the equation with Gauss-Seidel method. Furthermore, I have added Neumann boundary conditions in the extrema Retinex PDE to make it comparable with the resetless version.

Besides of just implementing the both approaches I have tried to analyze the geometrical meaning of the PDEs from the variational point of view (Chapter 4). The equivalence of the resetless PDE from [5] to a minimization problem was mentioned in [16] with references on [17]. Based on this point of view I will try to explain some disadvantages and weaknesses of the original approach by working with real images. Generally this disadvantages could be expressed as a lack of locality. In chapter 4 I will try also to extend the original PDE to make it more local and to improve its behavior when working with high detailed textures and generally with the real world images. I will also show a difference between the extended and original version of PDE with concrete examples (Chapter 6).

2. A short mathematical background. Used terms.

Illumination (B).
All external light falling on an object.

Reflectance (R).
Characteristic of an object to reflect external light falling on it.

Light intensity (I).
Reflected portion of light, that was received by a camera matrix, human eye or another type of light detector. I suppose, that $I = B \cdot R$.

Poisson equation.
Elliptical partial differential equation (PDE), that can be defined in 2D case as $\Delta\phi(x, y) = f(x, y)$

Dirichlet, Neumann and mixed boundary conditions.
Dirichlet conditions are a type of PDE boundary conditions, that specifies a solution function value on the domain boundary.
Neumann conditions opposite to Dirichlet conditions define a value of the solution derivative on the domain boundary.
Mixed boundary conditions are the combination of Dirichlet and Neumann boundary conditions.

2D random walk.
Is a mathematical model of a random path built in our case upon the following principle: each of the 4 adjacent points left, right, top and bottom can be chosen with the same probability 0.25 as a next step in every point.

Markov chain.
Is a memoryless random process on a state space wherein the next state depends only on the current state and not on previous states. 2D Random walk is a Markov chain.

Derivative discretization.
An approximation of n-th order derivative with finite differences. I will use the following forms of this differences:

$$\frac{\partial}{\partial x_i} f(x_1, .., x_n) = f(x_1, .., x_i, .., x_n) - f(x_1, .., x_{i-1}, .., x_n)$$

and

$$\frac{\partial^2}{\partial x_i^2} f(x_1, .., x_n) = f(x_1, .., x_{i+1}, .., x_n) - 2f(x_1, .., x_i, .., x_n) + f(x_1, .., x_{i-1}, .., x_n) \quad .$$

Thus

$$\Delta f(x, y) = f(x_{i+1}, y_i) + f(x_{i-1}, y_i) + f(x_i, y_{i+1}) + f(x, y_{i-1}) - 4f(x_i, y_i)$$

Normal derivative.
This is a directional derivative taken along the normal to the domain boundary. For our square domain it's just a derivative along negative and positive direction of the x-axis (positive is from left to right) in case of left and right boundary accordingly and along negative and positive direction (top-bottom is positive) of y-axis in case of top and bottom border.

Greens formula (the second Greens identity)
A relation that connects an integral of a function within domain and on the domain boundary. It will be used in the form of

$$\iint_D \Delta\phi \, dS = \int_{\partial D} \frac{\partial \phi}{\partial n} dl$$

where n is the normal to the domain boundary ∂D and $\frac{\partial \phi}{\partial n}$ is the normal derivative.

Gershgorin circle theorem.
A theorem that allows us to bound eigenvalues of a matrix $A = a_{i,j} \in R^{n\,x\,n}$.
The theorem says [19] the following

let $K_i = \left\{ \mu \in C, |\mu - a_{i,i}| \le \sum_{j=1, i \neq j}^{n} |a_{i,j}| \right\}$, than all eigenvalues of A lie in $\underset{i}{U} K_i$.

Euler's formula.
A relation between the complex exponent and trigonometrical functions:

$$e^{ix} = \cos(x) + i \sin(x) \quad .$$

Thus

$$\cos(x) = \Re(e^{ix}) = \frac{e^{ix} + e^{-ix}}{2}$$

Gaussian blur.
A blurring method based on convolution of the original function with gaussian kernel. Suppose that the kernel has the size 2n+1 x 2n+1, than for each pixel i, j of image I the following calculation should be made:

$$I'_{i,j} = \frac{1}{C} \sum_{l=-n}^{n} \sum_{k=-n}^{n} G_{l,k} \cdot I_{i+l, j+k}$$

where

$$G_{l,k} = \frac{1}{2 \cdot \pi \cdot \sigma^2} e^{\frac{-(l^2+k^2)}{2 \cdot \sigma^2}} \quad \text{and} \quad C = \sum_{l} \sum_{k} G_{l,k}$$

σ is the standard deviation.

Discrete Fourier transformation and its properties.
The 1-D forward transformation:

$$\hat{F}(k)=\frac{1}{N}\sum_{n=0}^{N-1}F(n)\mathrm{e}^{\frac{-i2\pi kn}{N}}$$

The 1-D backward transformation:

$$F(n)=\sum_{k=0}^{N-1}\hat{F}(k)\mathrm{e}^{\frac{i2\pi kn}{N}}$$

1-D Shift theorem:

$$F(n-n_0)=\sum_{k=0}^{N-1}\hat{F}(k)\mathrm{e}^{\frac{i2\pi kn}{N}}\mathrm{e}^{\frac{-i2\pi kn_0}{N}}$$

3. The PDE Formalization.

3.1 Basics of Land's Retinex theory.

As mentioned above the main purpose of the Retinex theory is to describe how the human visual system (HVS) can detect colors under conditions of non-uniform illumination. Edwin Land described in his article [1] an experiment, that showed the difference between mathematical description of a color using for example the RGB basis and perception of this color by a human (see Fig. 1). Depending on the environment, the same RGB value of a point can be sensed as two different colors by the HVS. Our vision always tries to recognize the original reflectance value of a point even if the object is illuminated with a non-white light. So how does the system do it?

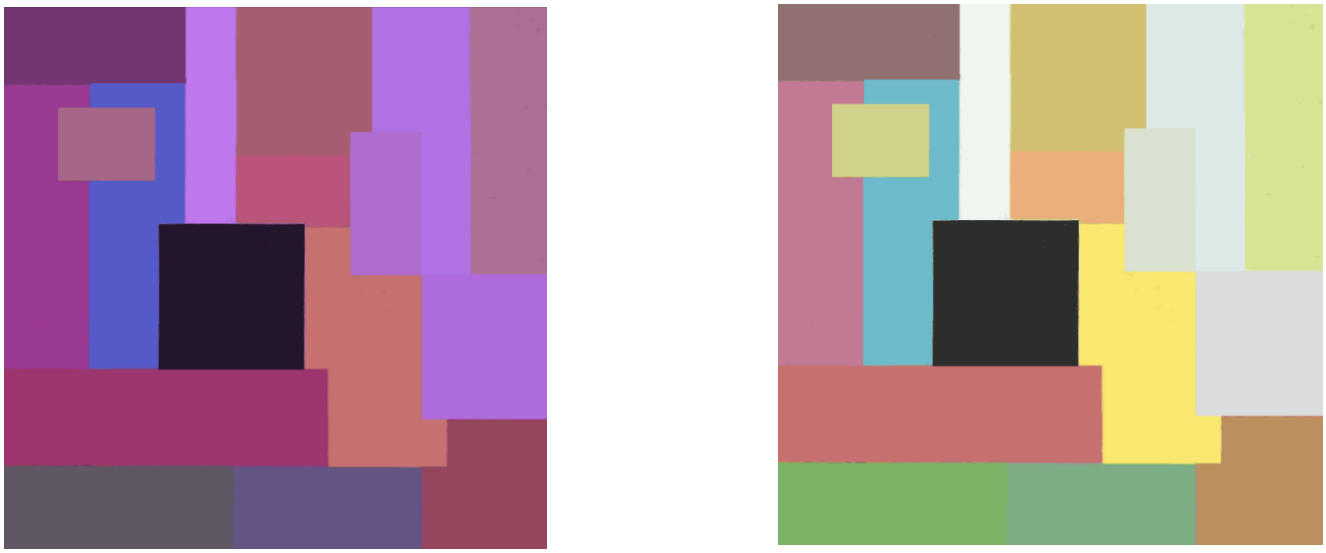


Fig 1. RGB value of the orange zone on the left side is the same as RGB value of the red zone (bottom left) on the right side. Here the left image was created from the right image by changing the spectrum of illumination. Yellow zone on the left looks now more like orange and the red looks more like lila. The source of the original image: [1]

The main idea of Lands Retinex theory is that edges within the image play a major role by the human's color perception mechanism. We know that the light intensity in each point of a picture consists of a product of reflectance and illumination.

$$I(x_1) = R(x_1) \cdot B(x_1) \qquad (3.1)$$

where $R(x_1)$ is a reflectance and $B(x_1)$ is an illumination at the point x_1 .

Suppose we have an edge between two homogenous areas with different reflectance values. In this case, if the external illumination changes slowly (what is normally the case) the color intensity ratio between two adjacent points on both sides of the edge is almost equal to the ratio of its reflectances:

$$\frac{I(x_1)}{I(x_2)} = \frac{R(x_1)\cdot B(x_1)}{R(x_2)\cdot B(x_2)} \approx \frac{R(x_1)}{R(x_2)} \quad (3.2)$$

If we take two remote points and calculate in such way the product of intensity ratios for all adjacent homogenous zones along some path between this two points, we will get approximately the ratio between illuminations of this two remote points. Indeed:

$$\begin{aligned} \frac{I(x_1)}{I(x_N)} &= \frac{I(x_1^1)}{I(x_1^2)}\cdot\frac{I(x_2^2)}{I(x_2^3)}\cdot\ldots\cdot\frac{I(x_a^{N-2})}{I(x_a^{N-1})} \\ &= \frac{R(x_1^1)\cdot B(x_1^1)}{R(x_1^2)\cdot B(x_1^2)}\cdot\frac{R(x_2^2)\cdot B(x_2^2)}{R(x_2^3)\cdot B(x_2^3)}\cdot\ldots\cdot\frac{R(x_a^{N-2})\cdot B(x_a^{N-2})}{R(x_a^{N-1})\cdot B(x_a^{N-1})} \quad (3.3) \\ &\approx \frac{R(x_1)}{R(x_N)} \end{aligned}$$

where x_1^1 is the adjacent point in the zone 1 to the point x_1^2 in the zone 2 and so on. Top index identifies a zone and the bottom index identifies a boundary. The zones are homogenous in the sense that the reflectance value changes slowly within such a zone, so that $R(x_1^2)\approx R(x_2^2)$. Under our assumptions the illumination also changes slowly within the image, so $B(x_2^1)\approx B(x_2^2)$. An example of such calculation is shown on the image below.

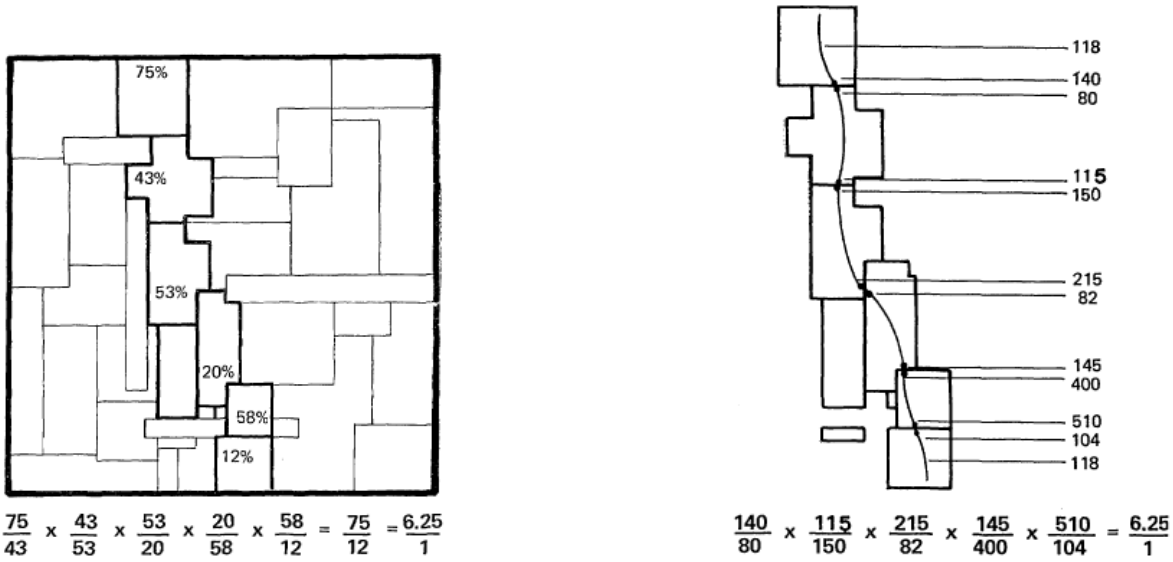


Fig. 2. On the left side: an example of calculating the ratio between reflectances of two Mondrian zones from top to bottom along some path without any illumination. On the right side: the same path, but this time there is an illumination. Each point is taken on the border of the Mondrian zone. Image source: [2].

To reduce inaccuracy that arises due to the slightly different illumination values and roundness error, we can calculate an average value of ratios along different paths between this two points (as originally proposed by Land).

If we take a logarithm of the average ratio $\frac{I_i}{I_k}$ and calculate for each point I_i the ratio to all other points of image $I_k, k=1..\mathrm{N}$, add this numbers together and divide through the number of points, we will obtain a ratio logarithm between the reflectance of a current point and a sort of an average reflectance of all other points I_{avr} (see 3.4).

$$\begin{aligned} L(I_i) &= \frac{1}{N}\sum_{k=1}^{N}\log\left(\frac{I_i}{I_k}\right) &&= \frac{1}{N}\sum_{k=1}^{N}\log(I_i)-\log(I_k) \\ &&&= \log(I_i)-\frac{1}{N}\sum_{k=1}^{N}\log(I_k) \quad (3.4) \\ &&&= \log\left(\frac{I_i}{I_{avr}}\right) \end{aligned}$$

The value $L(I_i)$ should represent the original reflectance or at least the effects of non-uniform illumination should be weakened. This data can be then normalized for each color channel to get a color image. This approach is called “resetless Retinex”.

Another way to remove effects of non-uniform illumination is to calculate for each point the ratio between the light intensity of the current point and some image extremum S, that represents the maximum value of light intensity for this color channel. We will than obtain some $\tilde{L}(I_i)$, that should be also normalized for each channel. This approach is called “extrema Retinex”.

$$\tilde{L}(I_i) = \frac{1}{N(S)}\sum_{k\in S}\log\left(\frac{I_i}{I_k}\right) \quad \text{where s are set of those extrema.}$$

In real world images the environment effects such shadows or reflexes have normally no edges and change slowly. There is a sense to add a thresholding function to the equation (3.4) to remove such effects:

$$\frac{1}{N}\sum_{k=1}^{N}\log\left(\frac{I}{I_k}\right) \rightarrow \frac{1}{N}\sum_{k=1}^{N}\delta\left(\log\left(\frac{I}{I_k}\right)\right) \quad (3.5)$$

where

$$\delta(x)=\begin{cases} 0, & for \quad |x|<threshold \\ x, & otherwise \end{cases} \quad (3.6) \quad .$$

Now we can define a value, that called a relative lightness between two pixels x and y on a path p from x to y

$$L(x, y, p) = \sum_{k=1}^{N(p)}\delta\left(\log\left(\frac{I(x_k)}{I(x_{k+1})}\right)\right), \quad where\ x_k \in p \quad \forall k=1..\mathrm{N}(p) \quad (3.7)$$

But as was said above, to reduce inaccuracy we can calculate an average of the ratios between x and y along different paths:

$$L(x, y) = \underset{p}{avg}\, L(x, y, p) \quad (3.8)$$

Expression (3.8) defines a relative lightness. The lightness of a pixel x is defined as an average of all relative lightnesses between x and the other pixels of image:

$$L(x) = \frac{1}{N} \sum_{y \in Image} L(x, y) \quad (3.9)$$

The lightness of a pixel in this case has the same meaning as described in (3.4) this is the ratio between reflectance of a current point and a sort of average reflectance of all other points.

In the case of extrema Retinex the lightness of a pixel is defined as an average of relative lightness of a current pixel and an image extrema S:

$$L(x) = \frac{1}{N(S)} \sum_{y \in S} L(x, y) \quad (3.10)$$

3.2 Interpretation of the original Land and McCann Retinex paths

The choice of path in all path based Retinex algorithms is a critical point of every such an approach. Results and computation complexity significantly depend on the form and on the number of the paths. There are many different approaches that use different sets of paths. Threshold is also important, making low detailed areas less relevant.

In the main work of Land the paths are not precisely defined. Most general approach were to generate random path. But it is complex and the result depends on the length and the number of this paths.

There is a work where the authors try to predict the result of Retinex algorithm to analyse the used paths and parameters [7]

There is also some improvement of direct paths calculation such as Brownian motion model [4] that needs less paths as direct approach to generate acceptable result and can greatly improve the computation.

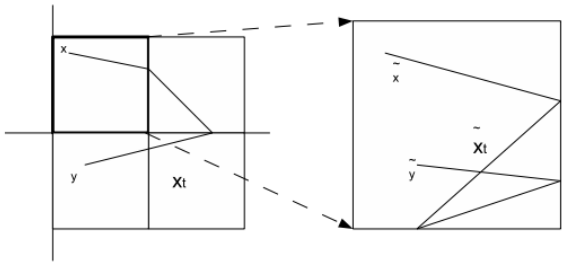
But the authors [5] do not reduce the set of paths, they try directly the opposite. The consider a general class of paths, the 2-D standard random walks.

3.3 Image domain symmetrization

Let's say we have an image and are trying to compute a standard random walk on the image domain. Then there is a question on what should be done, when we have reached an image border. The authors [5] are first consider a random walk on a whole plane which is covered with reflected copies of the original image. The reflection is made in the following manner: first the original image will be reflected through its four boundaries. Then, each reflected version will be again reflected through all its boundaries except the boundary with the previous image. Due to this process the whole plane will be covered with the 2N x 2M periodic 2x2 squares of reflected image copies.

Let's say the current step of the random path is in the point at the right border and the next step

will go through this border. We can [5] interpret this process as bouncing the random path back on this border and move in a congruent point within the same domain. The authors define a congruent points as a points obtained from the same pixel due to described above reflection procedure. The example of such process is shown on fig. 2.1



Bouncing the random path back on image borders. Image source: [5]

Figure 2.1

So we can consider the random path on the whole plane as a bounced random path within the original image because the image intensity in all congruent points is the same.

The authors then propose to consider the random walk as an irreducible Markov chain with the finite number of states. Thus, any pixel can be reached by the random walk from any other pixel in finite number of steps (our random walk is isotropic with respect to the 4 possible directions). And with the reference on a theory of Markov chains authors say also that for any pixel the expectation of any other pixel is uniformly bounded.

With all these assumptions we can define our averaged relative lightness as an expectation of a pixel y.

$$L(x,y) = \underset{p}{E}\, L(x,y,p) \qquad (3.11)$$

3.4 Equivalence of the random path model and the Poisson PDE.

The relative lightness equation. The following represents one of the main ideas in [5]. Let's say we have a pixel x with coordinates i, j and we want to compute the relative lightness between this pixel and a point y. As described above the lightness value in this case is an expectation of the relative lightness computed on all possible random paths from x to y:

$$L(x,y) = \underset{p}{E}\, L(x,y,p) = \sum_{p} P(p(x,y)) \cdot L(x,y,p(x,y))$$

where $P(p(x,y))$ is probability of a path $p(x,y)$ between x and y.

It is clear, that

$$P(p(x,y)) = \prod_{i=1}^{N(p(x,y))} P(p(x,y)_i)$$

where $p(x,y)_i$ is a point i within the path and $p(x,y)_1 = x$.

When x makes the first step, there are four possibilities to move: up, down, left and right. Each possibility has probability ¼. Four other sets of random paths start at each neighbor point, that is one step shorter than the correspondent path from original set of paths. So, we can decompose our

expectation as follows [5] (note, that $\sum_p P(p(x,y)) = 1$):

$$\begin{aligned} L(x,y) &= \frac{1}{4} \sum_{A=\text{up},\text{down},\text{left},\text{right}} \sum_p P(p(A,y)) \cdot \left(\delta\left(\log\left(\frac{I(x)}{I(x_A)} \right) \right) + L(x,y,p(A,y)) \right) \\ &= \frac{1}{4} \sum_{A=\text{up},\text{down},\text{left},\text{right}} \sum_p P(p(A,y)) \cdot \delta\left(\log\left(\frac{I(x)}{I(x_A)} \right) \right) + \sum_p P(p(A,y)) \cdot L(x,y,p(A,y)) \\ &= \frac{1}{4} \sum_{A=\text{up},\text{down},\text{left},\text{right}} \left(\delta\left(\log\left(\frac{I(x)}{I(x_A)} \right) \right) + \sum_p P(p(A,y)) \cdot L(A,y,p(A,y)) \right) \end{aligned}$$

where “up”, “down”, “left”, “right” means the corresponding adjacent point to x. Simple arithmetic:

$$4 \cdot L(x,y) - \sum_{A=\text{up},\text{down},\text{left},\text{right}} \sum_p P(p(A,y)) \cdot L(x,y,p(A,y)) = \sum_{A=\text{up},\text{down},\text{left},\text{right}} \delta\left(\log\left(\frac{I(x)}{I(x_A)} \right) \right)$$

The correspondent expectations are from the left side:

$$4 \cdot L(x,y) - \sum_{A=\text{up},\text{down},\text{left},\text{right}} E_p L(x_A,y,p(A,y)) = \sum_{A=\text{up},\text{down},\text{left},\text{right}} \delta\left(\log\left(\frac{I(x)}{I(x_A)} \right) \right)$$

Per definition (3.11):

$$4 \cdot L(x,y) - \sum_{A=\text{up},\text{down},\text{left},\text{right}} L(x_A,y) = \sum_{A=\text{up},\text{down},\text{left},\text{right}} \delta\left(\log\left(\frac{I(x)}{I(x_A)} \right) \right)$$

Now there is a discrete Laplace-operator from the left side;

$$-\Delta L(x,y) = \sum_{A=\text{up},\text{down},\text{left},\text{right}} \delta\left(\log\left(\frac{I(x)}{I(x_A)} \right) \right) = F(x) \quad (3.12)$$

It is also clear, that

$$L(x,y) = 0, \quad \text{if } x = y \quad (3.13)$$

Border conditions. From our image domain symmetrization it follows that for each border pixel of the original image the opposite pixel exists from the other side of the border with the same

intensity and with the same lightness value. For example, for the border pixel x (Fig. 3) from the right

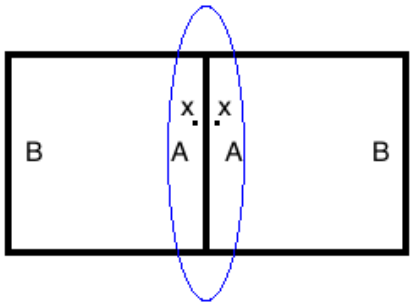


Figure 3

side of image there is an opposite pixel x from the left side with the same lightness value. Mathematically this can be formulated as following:

$$L(x_{i,0}, y) = L(x^{rl}_{i,n-1}, y), \quad L(x_{i,n-1}, y) = L(x^{rr}_{i,0}, y) \quad , for\ i = 0..n-1$$

$$L(x_{0,j}, y) = L(x^{rt}_{m-1,j}, y), \quad L(x_{m-1,j}, y) = L(x^{rb}_{0,j}, y) \quad , for\ j = 0..m-1$$

where x^{rl} pixels of the left reflected copy of the image, x^{rr} - right, x^{rt} - top, x^{rb} - bottom. This can be rewritten as

$$L(x_{i,0}, y) - L(x^{rl}_{i,n-1}, y) = 0, \quad L(x_{i,n-1}, y) - L(x^{rr}_{i,0}, y) = 0 \quad , for\ i = 0..n-1 \quad (3.14)$$

$$L(x_{0,j}) - L(x^{rt}_{m-1,j}) = 0, \quad L(x_{m-1,j}) - L(x^{rb}_{0,j}) = 0 \quad , for\ j = 0..m-1 \quad (3.15)$$

3.14 is the discrete analog of the normal derivative at pixels on the left and right borders of the original image. 3.15 is the discrete normal derivative at the top and bottom borders. So

$$\frac{\partial L(x, y)}{\partial n} = 0, \quad \forall x \in \partial D \quad (3.16)$$

where D is original image domain.

The resetless lightness PDE.

We have now defined the equation for the relative lightness. But eventually we need an equation for computing the lightness that can be normalized and converted in standard RGB image. This equation can be derived from (3.12) using the discrete version of Greens law and Neumann boundary condition (3.16) [5].

We want to compute a sum of (3.12) by all $y \in D$ and transform the right side of the correspondent equation. But our equation for the relative lightness doesn't define the $\Delta L(x, y)$ for x = y. Our equation only defines that $L(x, y) = 0$ in this case. So we want to find an expression for $\Delta L(y, y)$.

Using discrete version of the Greens formula:

$$\sum_{x\in D}\Delta L(x,y) \;=\; \sum_{x\in D,\,x\neq y}\Delta L(x,y)+\Delta L(y,y) \;=\; \sum_{x\in \partial D}\frac{\partial L(x,y)}{\partial n} \;=\; 0$$

from this expression we can derive:

$$\Delta L(y,y) \;=\; -\sum_{x\in D,\,x\neq y}\Delta L(x,y) \;=\; \sum_{x\in D,\,x\neq y}F(x) \;=\; -F(y)$$

The last step of the above transformation can be derived as follows:

$$\sum_{x\in D,\,x\neq y}F(x) \;=\; \sum_{x\in D}F(x)-F(y) \;=\; \sum_{x\in D}\left(\sum_{A=\text{up},\,\text{down},\,\text{left},\,\text{right}}\delta\left(\log\left(\frac{I(x)}{I(x_A)}\right)\right)\right)-F(y) \;=\; -F(y)$$

Here $\sum_{x\in D}F(x)=0$ because for each ratio $\log(\frac{I(x)}{I(x_i)})$ there is a antisymmetric ratio for the correspondent adjacent point $\log(\frac{I(x_i)}{I(x)}) \;=\; -\log(\frac{I(x)}{I(x_i)})$.

So we can now write using (3.12)

$$\begin{aligned}\Delta L(x) \;&=\; \frac{1}{M\cdot N}\Big(\sum_{y\in D,\,x\neq y}\Delta L(x,y)+\Delta L(y,y)\Big)\\ &=\; -\frac{1}{M\cdot N}\big((M\cdot N-1)\cdot F(x)+F(x)\big)\\ &=\; -F(x)\end{aligned} \qquad (3.17)$$

with boundary conditions derived from (3.16)

$$\frac{\partial L(x)}{\partial n} \;=\; 0, \quad \forall\, x\in\partial D \quad (3.18)$$

The case of extrema Retinex.

As for equation (3.17), in the case of extrema Retinex all above argumentation can be used to derive an equation for the lightness [5]. The only difference is the fact that instead of computing the average of (3.12) on the whole image domain, only the set of image extrema is used. As a result we will get the following system of equations

$$\begin{aligned}-\Delta L(x) \;&=\; \sum_{A=\text{up},\,\text{down},\,\text{left},\,\text{right}}\delta\log\left(\frac{I(x)}{I(x_A)}\right) \quad \forall\, x\notin Y\\ L(x) \;&=\; 0 \qquad\qquad\qquad\qquad\qquad\qquad \forall\, x\in Y\end{aligned} \qquad (3.19)$$

where Y is the set of image extrema.

4. Geometrical interpretation.

4.1 Geometrical interpretation of the resetless PDE.

Poisson equation similar to those derived in [5] appears in other applications of image analysis, such as texture flattening, poisson image edition, shadow removing etc.

In this chapter I will try to analyze the geometrical meaning of the Retinex equations derived in [5] and described above. I will only consider the case with non-zero threshold and the influence of this threshold.

In the paper by Dominique Zosso [16] with references on works of A.Blake (for example [17]) it was mentioned, that the resetless Retinex equation derived in Morel's paper [5] is equivalent to the following optimization problem

$$L(x,y)=\underset{L}{argmin}(\|\nabla L(x,y)-\delta(\nabla \tilde{I}(x,y))\|_2^2)$$

where $\tilde{I}(x,y)$ is the logarithmic intensity. That means that solution of equation (3.17) represents a function whose gradient lies close (optimal close with respect to L2 norm) to the thresholded gradient of the function $\tilde{I}(x,y)$. I will use this point of view to interpret the results of the resetless PDE solution and to describe the geometrical meaning of this PDE.

The gradient of a function is a vector, that points on the direction of the fastest function changing. Mathematically this differential operator described as a vector, which components are partial derivatives of the function along the corresponding coordinate axes. In 2D case it looks like this:

$$\nabla L(x,y)=\begin{pmatrix}\frac{\partial L(x,y)}{\partial x}\\ \frac{\partial L(x,y)}{\partial y}\end{pmatrix} .$$

If we compute a gradient of an image, it will show us in each point in which direction the most significant changes of the light intensity are with respect to the current point. The amplitude of the gradient represents the speed of this intensity changing in the direction of the gradient.

To understand the geometrical meaning of the equation (3.17) let us consider an example on the figure 5 left. This is the object with circle symmetry and with the varying gray filling. Its diametral gray value profile is shown on the figure 4 left and the gradient profile along the same line is shown on the figure 4 right.

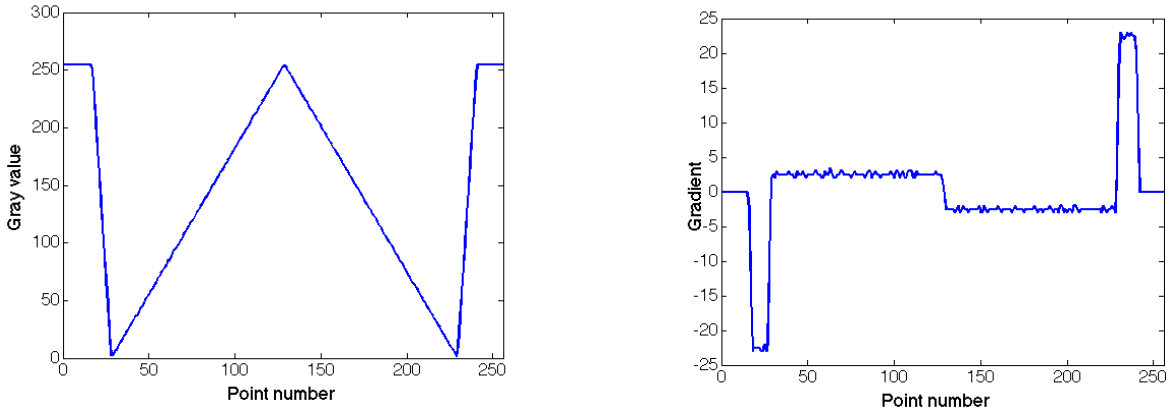


Figure 4.

In the center of the Fig. 5 left one can see the maximum of the light intensity, being the white color . This corresponds to the central maximum on the Fig. 4 left. From the center to the border the light intensity linearly sinks and the border looks partially black. Then the intensity grows fast an becomes white, that corresponds to the blurred outer side of the ball. On the Fig. 4 right one can see the speed profile of described changes. Two peaks left and right correspond to the blurred border and the central two plateaus correspond to the inner gradient of the ball. On the Fig. 5 right a vector field of the image gradient on both sides of the balls border is showed.

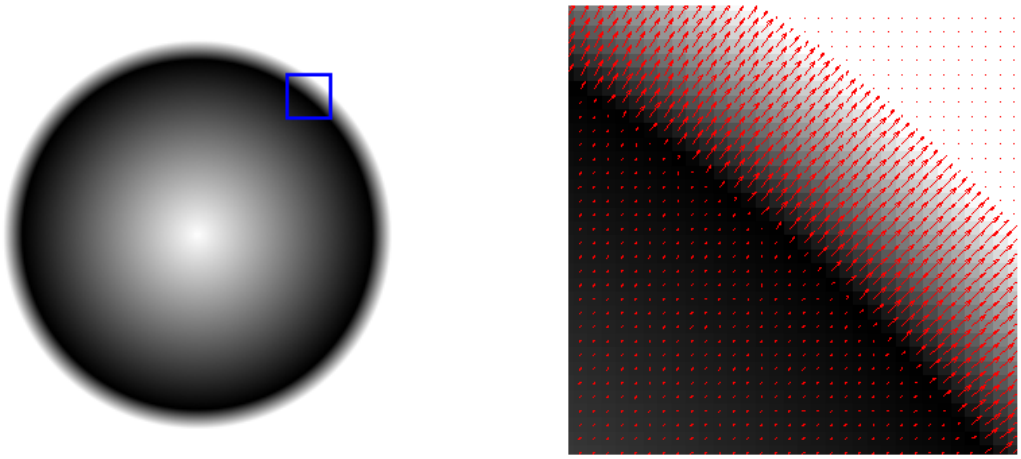


Figure 5.

Let us apply our resetless Retinex equation to this example with a threshold, that is higher than the inner part of the ball's gradient. What will we get? The inner part of the gradient should be deleted and the outer two peaks should remain intact. So the gradient must be zero within the processed version of the ball, that corresponds to the constant filling, and the border must remain blurred with the same gradient. The result of the processing is shown on the Fig. 6. It's exactly the same as what was

expected. The light inhomogeneity within the ball can be explained as an artifact of discretization (we use a cross like pattern).

Figure 6.

This example shows a trend on the processing with the reseltless PDE. If some region of image has a border with a strong gradient and the filling with a soft gradient, that is lower than used threshold, than the inner part after the processing is only defined by the border gradient. This gradient shows virtually for how much the light intensity around the ball should be increased or decreased during the transition over the balls border.

Let us consider another example, that is almost similar to the previous one with one important difference. Instead of the soft gradient we will put a sharp object in the center of the ball. Let it be a square (Fig. 7 left). We will put another square with the same gray value outside the ball to see the difference in the processing of this two squares.

Figure 7. Left: the original image. Right: the processed image

One can see some difference in the squares color already without any processing,. The outer one looks a little bit darker than the inner one. It was our visual system processing. And the result of applying the resetless PDE is on the right side of the Fig. 7. The threshold was set this time higher than the border gradient. The algorithm has firstly removed the ball and secondly it repeated contrastly exactly the same trend as our visual system has. It made the inner square much brighter and the outer square much darker. Let's try to understand why it has happened?

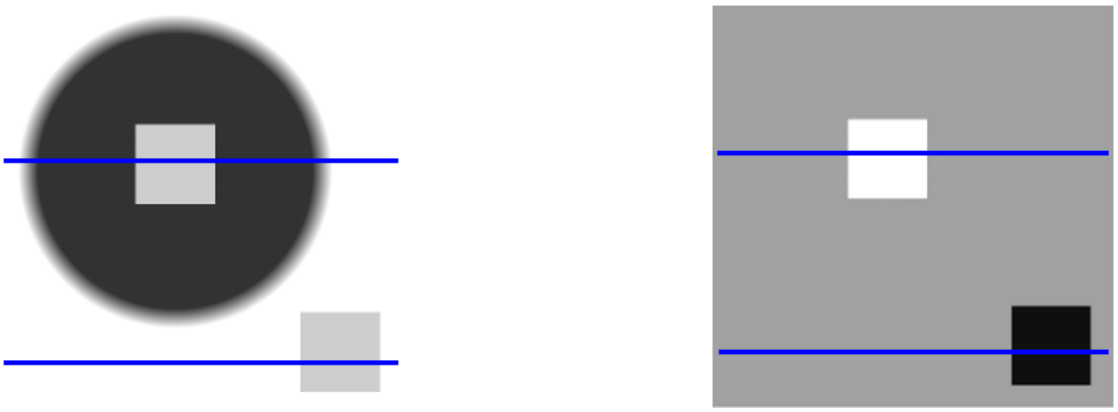


Figure 8.

If we compute a gradient along the blue lines showed on the Fig. 8 left, we will get the result showed on the left images above (upper left for the top line, bottom left for the bottom line). The red lines on the left upper part of the Fig. 9 mark used threshold, that is higher than both ball border peaks. So this both peaks was set to zero.

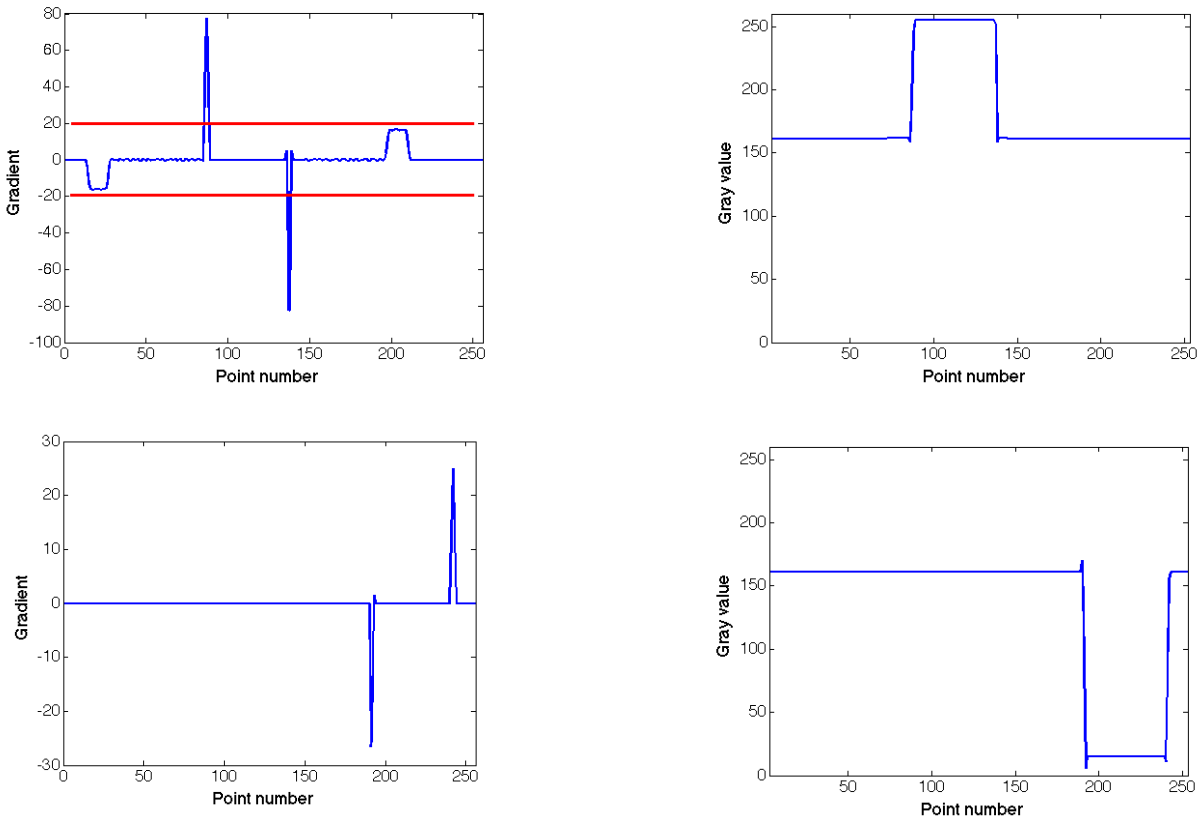


Figure 9.

If we use this processed version of the gradient to restore the image, that corresponds to this gradient, we will get something like a white region on the homogenous background for the upper gradient and the black region on the same background for the lower gradient. This is practically the result of the spatial integration both these gradients. Exactly this result we have obtained after applying the Retinex PDE (Fig. 8 right - the result, Fig. 9 right - the gray value profiles). There is no ball, because the gradient contained the information about the region was removed. The square within the ball becomes much lighter because its gradient was retained while the balls gradient was removed. So the "starting point" for the squares color reconstruction from the gradient is located now higher on the gray value axis and the whole square thus become lighter. The outer square become darker, because its "starting point" (the former 100% white point) was sinked after normalization to the middle gray.

The result of applying the resetless PDE to the Adelsons checker shadow illusion (Fig. 15 left and middle) can be explained exactly in the same way. The square that lies in the shadow becomes lighter after the Retinex processing if the threshold corresponds to the shadow gradient. The shadow was not completely deleted because its border is not homogeneous. It contains parts with gradients, that are higher than selected threshold. And if the threshold is too high, the other details that lie out of the shadow can be also deleted.

4.2 Color constancy.

Consider one more synthetical example, that will show us the ability of this algorithm to restore original colors in case of non-uniform colored illumination.

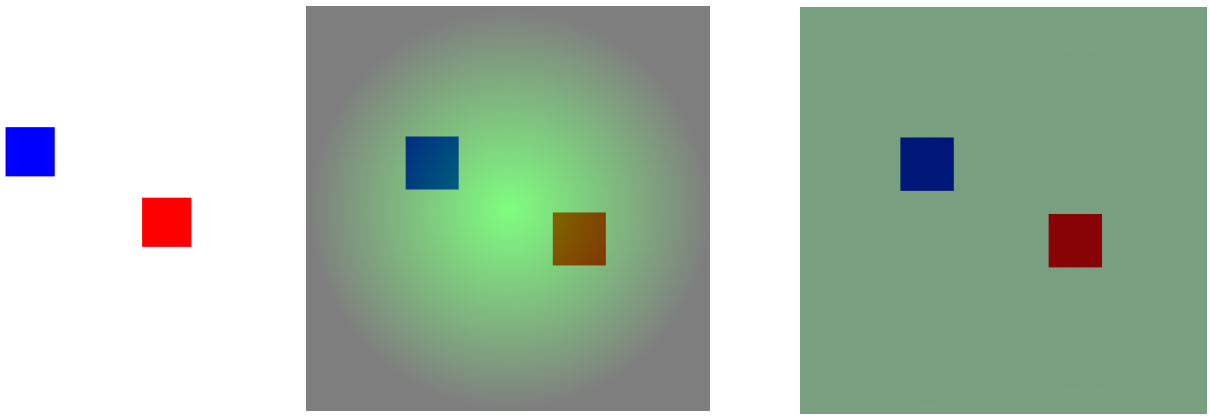


Figure 10.

On the Fig. 10 left the blue and the red square regions are illuminated with 100% white light. Further in the middle of the Fig. 10 are the same regions illuminated with the green gradient light mixed with 50% of homogenous white. This is our synthetic case of non-uniform non-white illumination. For our HVS the blue square remains blue with a light mix of green. The red square becomes something like brown or dirty red, however we can say that it is more likely red in this environment. The RGB values of this two squares are (0,63,127) for the blue one and (127,81,0) for the red one. The color (127,81,0) for example, can be sensed in other environment as brown.

Let us apply our Retinex PDE to this image with the threshold higher than the green lights gradient. The result is shown on the Fig.10 right. Both colors are restored and now the blue is blue and the red is red.

The mechanics in this case is the same as in the examples, considered above. In the red and blue channels there are no gradient lower than the selected threshold, so all gradients in these two channels remain without changes. In the green channel of the illuminated image (Fig. 10 middle) there is a gradient, that lies under the selected threshold, so it was deleted. There were two dark squares within this gradient (Fig. 11 left), as the result of multiplying of illumination with the squares reflectance. After removing of the gradient in the green channel, the two dark regions became darker due to the same reason, as on the Fig 7 right. Thus, during the Retinex processing the green color was removed from these two regions and they became again almost blue and almost red.

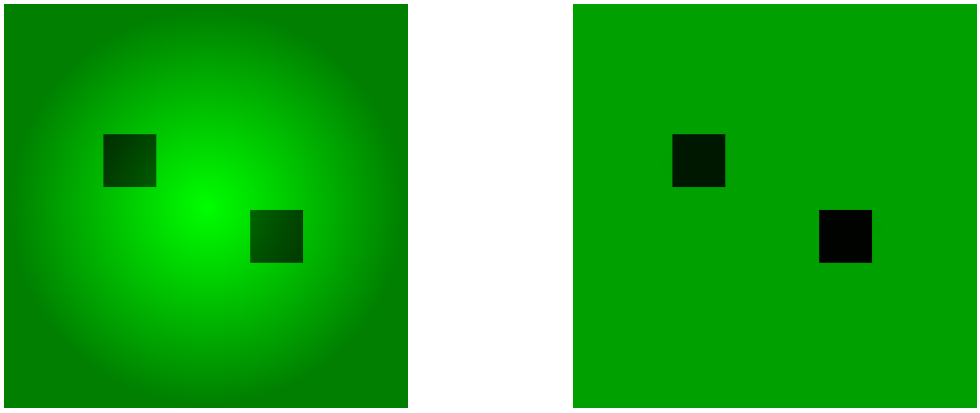


Figure 11.

It follows from the considerations above that the algorithm is capable to remove soft gradients such as shadows, color reflexes and non-uniform illumination, in case of relative homogenous (in sense of reflectance) surfaces and correctly selected threshold. Moreover, due to the separated processing of color channels, certain reflectance restoration effect can be achieved in case of non-uniform colored illumination.

I would like to say a few words about the white balance. The normalization method used by authors of the article [5] and in my implementation of their Retinex PDE preserve the median and the standard deviation in each color channel. Thus, the white balance of the whole image changes weakly and can be separately corrected, if needed. Described algorithm can only restore colors on the parts of the image, but not the white balance of the whole image.

4.3 Textured surface and importance of the shadows border.

After synthetical tests I would like to consider the behavior of the algorithm in the “natural conditions”. While experimenting with the real world images, I have noticed that the more homogeneous the surfaces on the image, the better the performance of the algorithm. For example, it perfectly removes the shadow in the Adelson's checker shadow illusion (Fig. 15) but poorly in case of a natural textures, such those presented on the Fig. 18, 19. In latter case, in order to remove the shadow one should make the threshold so high, that almost all the details of the texture will be also removed. This effect can be explained by the fact that shadows are multiplicative and the more sharp details lie within the shadow, the higher will be the resulting gradient amplitude in these points. The higher will be the gradients, the higher must be the threshold to delete them. The higher is the threshold, the more details will be deleted within the whole image.

While carrying out experiments with synthetical images I have noticed also that the texture under the shadow border plays a major role in the processing with our Retinex PDE. If the texture in

this region is flat and contains no sharp details than the shadow can be relatively easy removed, despite the fact that the texture within the shadow can be more complex. If the texture is complex under the shadow border, than the algorithm works much worse. I suppose the reason for this is that the shadow border gradient, containing the information about the inner flat part of the shadow, will be preserved due to the presence of the sharp high gradient details under the border. Only if the threshold is high enough to remove the gradients of these details, the shadow will be also removed.

At the top of the Figure 12 a shadow falling on the surfaces with different textures is shown. The left surface is completely flat. The middle surface contains a texture under the shadow border. The right surface contains a texture with the same pattern as the middle one, but the texture lies under the inner part of the shadow. The processing results of these three shadows are at the bottom of the Figure 12. The threshold in this three cases is the same and it is higher than the shadow gradient. In the first example (left) the shadow was completely removed. In the middle example the inner part of the shadow and the texture were preserved. Finally, in the third case the shadow was removed but the texture was preserved. This illustrates the effect described above.

On the Figure 13 the textured shadow border is shown in more details. One can see that as a result of the texture pattern and the shadow border multiplication, the texture acquires a gradient and preserves the information about the shadow. On the upper right side of the border the texture pattern is shown apparent and there is no variation of the gray value in it, only the black one.

It is clear that in the case when the whole shadow is falling on the highly textured surface, the same effect will appear. The shadow will be preserved till the threshold acquires the values of those details gradient, that lies under the shadow border (see figures 18-21 top right).

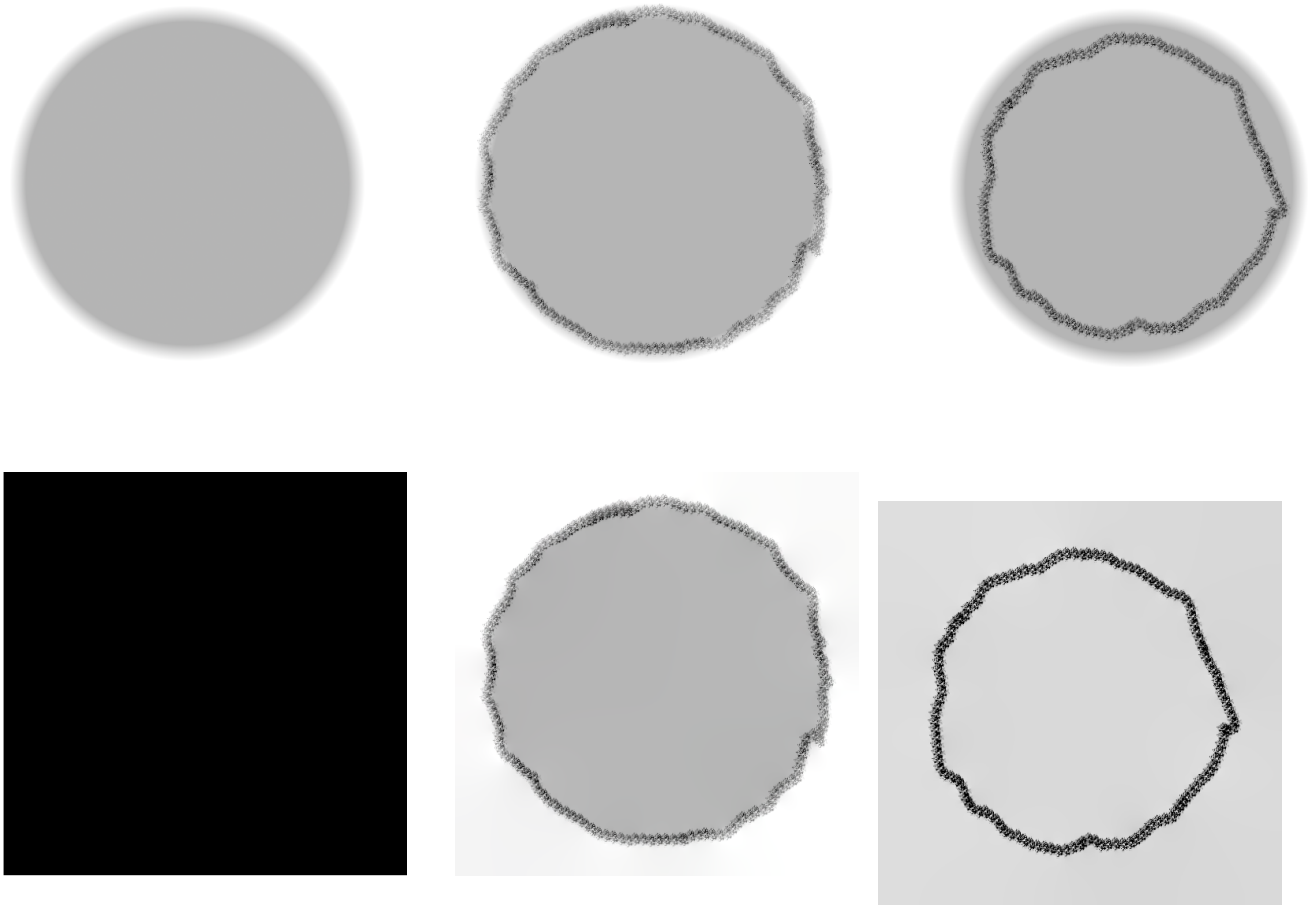


Figure 12.

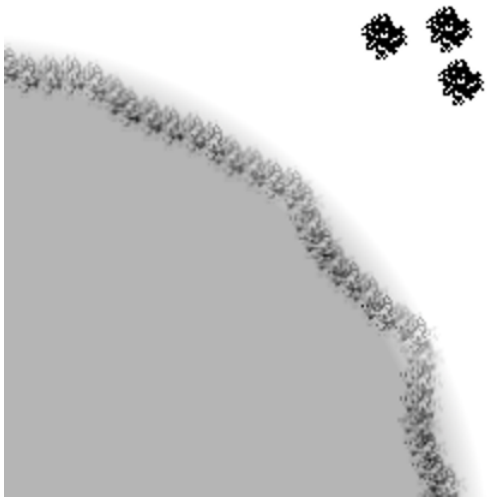


Figure 13.

All above considerations are valid in case of reset Retinex PDE with mixed border conditions when the Dirichlet part of the conditions is set in a point for each color channel. With this one point we can eliminate the constant, that appears due to the second order derivative in our Poisson equation and due to the first order derivative in the Neumann part of conditions.

4.4 Extended PDE.

Above I have described one of the problems of mentioned lightness PDE. This problem appears in cases when the image contains multiplicative effects, such as shadows or illuminance changes, that lie in the areas of the complex textures. Because of these textures the gradients in these areas are greater than the acceptable threshold value and will be preserved. Together with these gradients the information about shadows will be saved and this shadows will not be removed. To avoid this effect I have tried to improve the equation. For this purpose the hard thresholding operator was replaced with another expression. Let us call the equation with the new filtering method the "extended PDE". The extended thresholding is:

$$\delta_{ext}(\nabla \tilde{I}(x,y)) = \nabla \tilde{I}(x,y) - \alpha \hat{\delta}(\nabla \tilde{I}_b(x,y))$$

where

$$\hat{\delta}(x) = \begin{cases} 0, & for \quad |x| \geq threshold \\ x, & otherwise \end{cases}$$

and $\tilde{I}_b(x,y)$ is blurred version of the logarithmic intensity $\tilde{I}(x,y)$. $\alpha \in [0..1]$ is a model parameter, which meaning will be described later. In my experiments I have used gaussian blurring.

The sense of this replacement is relatively simple. The Gaussian blur destroys small sharp details but preserves those ones, which radius is big enough comparing to the blurring radius. Thus varying the blurring parameters we can achieve an optimal blurring when the contribution of texture details in the gradient will be suppressed and we will get mainly the shadows gradient. Since the gradients of shadows and other multiplicative effects should be relatively low, the inverted hard thresholding operator $\hat{\delta}(x)$ will help us to remove gradients that more likely belong to blurred

texture details. So by subtracting in such way isolated law gradients from the original gradient field we will suppress shadows and other multiplicative effects. The coefficient α in this case controls degree of this suppression.

Further relying on the Blake's paper [17] and replace $\delta_{ext}(\nabla \tilde{I}(x,y))=E$ we can say , that the equation $\Delta L=\nabla E$ with the same Neumann boundary conditions as for our symmetrized image gives us the solution of the following optimization problem:

$$L=\underset{L}{argmin}(\|\nabla L-E\|_2^2) \quad .$$

Thus, we will find a function L with the closest gradient to the gradient of E. This is exactly what I wanted.

We need also to compute a discrete version of $\nabla\cdot E$. Note, that

$$\begin{aligned}\nabla\cdot E &= \nabla\cdot(\nabla \tilde{I}(x,y)-\alpha\hat{\delta}(\nabla \tilde{I}_b(x,y)))\\ &= \Delta \tilde{I}(x,y)-\alpha\nabla\cdot\hat{\delta}(\nabla \tilde{I}_b(x,y))\\ &= \Delta \tilde{I}(x,y)-\alpha\frac{\partial}{\partial x}\hat{\delta}\left(\frac{\partial}{\partial x}\tilde{I}_b(x,y)\right)-\alpha\frac{\partial}{\partial y}\hat{\delta}\left(\frac{\partial}{\partial y}\tilde{I}_b(x,y)\right)\end{aligned} \quad .$$

And using the partial derivatives discretization:

$$\nabla E = -\sum_{A=\text{up, down, left, right}} \log\left(\frac{I(x)}{I(x_A)}\right)-\alpha\hat{\hat{\delta}}\left(\log\left(\frac{I_b(x)}{I_b(x_A)}\right)\right) \quad .$$

Correspondingly, the extended version of our Retinex PDE has the following form:

$$\Delta L_{ext}=\sum_{A=\text{up, down, left, right}} \log\left(\frac{I(x)}{I(x_A)}\right)-\alpha\hat{\hat{\delta}}\left(\log\left(\frac{I_b(x)}{I_b(x_A)}\right)\right)=-F_{ext} \quad (4.4.1)$$

Using the same boundary conditions as for (3.17), this equation can be solved with Fourier transformation. The details are described below.

Chapter 6 contains a set of examples for this approach.

5. Implementation.

5.1 Calculation of the right side of the PDEs.

Before we can solve the equation (3.17) we must calculate the right side of the equation. In case of raw matrix data the function F is defined as a sum of intensity ratios logarithms between the current pixel and its neighbors. But in case of gamma corrected images (this is the largest part of modern digital photos) the authors in [5] propose another approach. I will quote it:

"The gamma-correction consists of applying a concave function to the raw image, in practice logarithm or a s^{γ} with $0 < \gamma < 1$. Assuming that the gamma-correction is logarithmic is not restrictive: s^{γ} and log have a very similar shape over the usual image range. As a rewarding consequence of this assumption, instead of working with differences between logarithms of intensities like in Retinex, we can deal directly with intensity differences when working with gamma-corrected images. "

Thus in case of gamma corrected images the authors propose the following expression to calculate the F:

$$F = \sum_{A=\text{up, down, left, right}} \delta\left(I(x) - I(x_A)\right)$$

I will use this argument to simplify computations and solution of the equation. Additionally, I would like to note that because of the Neuemann conditions (3.18) following from the image domain symmetrization, the boundary points of two reflected image copies are equal along the same boundary. Thus, the ratio logarithm in case of RAW data or intensity difference in case of gamma correction for such points are always zero. This terms can be simply dropped out. On the basis of above considerations the following algorithm can be constructed (see Listing 1).

Listing 1 Compute F

Input: input - array of points in spatial domain
w – width of image
h – height of the image
threshold – the Retinex threshold
Output: **output – F**

```
computeF(input: array of double, w: int, h:int, threshold: int)
{
   // empty output array
   output = new array(w,h);

   for(int i = 0; i < h-1; i++){
      for(int j = 0; j < w-1; j++){
         int temp = 0;

         // detecting if it is a top border point
         if(!topBorder){
            temp+= abs(input[i][j] - input[i-1][j]) > threshold ? input[i][j] - input[i-1][j] : 0;
         }

         // detecting if it is a left border point
         if(!leftBorder){
            temp+= abs(input[i][j] - input[i][j-1]) > threshold ? input[i][j] - input[i][j-1] : 0;
         }

         // detecting if it is a right border point
         if(!rightBorder){
            temp+= abs(input[i][j] - input[i][j+1]) > threshold ? input[i][j] - input[i][j+1] : 0;
         }

         // detecting if it is a bottom border point
         if(!bottomBorder){
            temp+= abs(input[i][j] - input[i+1][j]) > threshold ? input[i][j] - input[i+1][j] : 0;
         }

         output[i,j] = tmp;
      }
   }

   return output;
}
```

5.2 Calculation of the extended right side of the PDEs.

To solve the extended version of (3.17) we must calculate the extended version of F (the right side of eq. 4.4.1). All other computations in the case of extended version will be the same as for the original version (3.17).

At first we will compute a blurred version of an image. Then the same function as on the previous listing but with inverted threshold will be used. And finally the difference between F and "blurred version" of F will be calculated.

Listing 2 contains the main function that calculates our F_{ext}. The subroutine computeFR is the same as above computeF with the only difference – the inverted threshold function ("<" instead of ">").

Listing 2 Compute F_ext

Input: input - array of points in spatial domain
w – image width
h – image height
threshold – the extended Retinex threshold
a – the parameter alpha from (4.4.1)
blur_box – blurring kernel size
blur_dev – gaussian blur deviation parameter

Output: **output – F_ext**

```
computeF_ext(input: array of double, w: int, h:int, threshold: int, a: int, blur_box: int, blur_dev:
int)
{
   // empty output array
   output = new array(w,h);

   inputb = gauss_blur(input, w, h, blur_box, blur_dev);
   inputb = computeFR(inputb, w, h, threshold);
   input = computeF(input, w, h, 0);

   for(int i = 0; i < h-1; i++){
      for(int j = 0; j < w-1; j++){
          output[i,j] = input[i,j] – a*inputb[i,j];
      }
   }

   return output;
}
```

Listing 3 gauss_blur

Input: input - array of numbers in spatial domain
w – image width
h – image height
blur_box – blurring kernel size
blur_dev – gaussian blur deviation parameter

Output: **output – blurred input**

```
gauss_blur(input: array of double, w:int, h:int, blur_box: int, blur_dev: int)
{
   // empty output array
   output = new array(w,h);

   //generate gauss blurring kernel
   kernel = new array(2* blur_box + 1, 2*blur_box+1)
   for (int i = 0;  i < 2* blur_box + 1;  i++){
      int x = i - size;
      for (int j = 0;  j < 2* blur_box + 1;  j++){
         int y = j - blur_box;
         kernel[i,j] = exp(-(y*y + x*x) / (2.0*blur_dev*blur_dev)) / (M_PI * 2.0 * blur_dev *
blur_dev);
      }
   }
   //calculate convolution with the kernel
   for(int i = 0; i < h; i++){
      for(int j = 0; j < w; j++){

      double average = 0;
      double norm = 0;
      //correct boundaries
      int ibegin = (i - blur_box ) > 0 ? - blur_box : -i;
      int iend = (h - (i + blur_box ) > 0 ? blur_box  : h - i - 1);
      int jbegin = (j - blur_box ) > 0 ? - blur_box : -j;
      int jend = (w - (j + blur_box ) > 0 ? blur_box : w - j - 1);

      for(int ii = ibegin; ii < iend + 1; ii++){
         for(int jj = jbegin ; jj < jend + 1; jj++){
            average += kernel[blur_box + ii][blur_box + jj]*input[ ii + i, jj + j];
            norm += kernel[blur_box + ii][blur_box + jj];
         }
      }
      average = average / norm;
      output[i,j] =  average;
   }
}
return output;
}
```

5.3 Discrete Fourier transformation.

As proposed in [5] equations (3.17) can be solved using discrete Fourier transformation. Discrete derivatives in Fourier domain become simple multiplications and our PDE reduces to ordinary arithmetical operations. We can than convert the Fourier domain solution in spatial domain and thus solve the original equation.

Using the properties of Fourier coefficients we can express the result of applying the Laplace operator in Fourier domain as follows:

$$\begin{aligned} -\Delta \hat{L}(l,m) &= 4\hat{L}(l,m) - \hat{L}(l,m)\cdot e^{\frac{-i\cdot 2\cdot \pi}{N}} - \hat{L}(l,m)\cdot e^{\frac{i\cdot 2\cdot \pi}{N}} - \hat{L}(l,m)\cdot e^{\frac{-i\cdot 2\cdot \pi}{M}} - \hat{L}(l,m)\cdot e^{\frac{i\cdot 2\cdot \pi}{M}} \\ &= 4\hat{L}(l,m) - \hat{L}(l,m)\cdot (e^{\frac{-i\cdot 2\cdot \pi}{N}} + e^{\frac{i\cdot 2\cdot \pi}{N}} + e^{\frac{-i\cdot 2\cdot \pi}{M}} + e^{\frac{i\cdot 2\cdot \pi}{M}}) \end{aligned}$$

where N and M is height and width of the image domain.

Using then the Euler's formula for cosine, we will get

$$-\Delta \hat{L}(l,m) = \hat{L}(l,m)(4 - 2\cos(\frac{2\cdot \pi}{N}) - 2\cos(\frac{2\cdot \pi}{M}))$$

Thus, the whole equation (3.17) becomes in Fourier domain

$$\hat{L}(l,m)(4 - 2\cos(\frac{2\cdot \pi}{N}) - 2\cos(\frac{2\cdot \pi}{M})) = \hat{F}(l,m)$$

where $\hat{F}(l,m)$ is the Fourier transformation of the right side of (3.17). Then, with simple arithmetic operations:

$$\hat{L}(l,m) = \frac{\hat{F}(l,m)}{4 - 2\cos(\frac{2\cdot \pi}{N}) - 2\cos(\frac{2\cdot \pi}{M})} \quad (5.3.1)$$

Thus, as said above, to solve the equation (3.17) we need to make a forward Fourier transformation and find $\hat{F}(l,m)$. Then using (5.3.1) we can find $\hat{L}(l,m)$ and transform it back to spatial domain to find the original $L(x,y)$. Periodicity of the back Fourier transformation ensure automatic fulfillment of our Neuemann boundary conditions (3.18) for the symmetrized image.

5.4 Discrete cosine transformation

As mentioned in [5] due to the symmetrization of our image the complex part of Fourier transformation will be always zero. This can simplify the calculations, if we reduce the transformation to the real numbers field and will calculate only cosine part of Fourier transformation. I will show a simple geometrical explanation of the above possibility.

Let's take a row of 2N pixels from a symmetrized image. Each point is symmetrically reflected with respect to the center of the row, the point N/2-1+1/2. Let's consider the point B on the Figure 14 to be the reflected version of the point A on the other side of the the center line. Clearly, that these two points have the same lightness intensity value and the same distance to the centre of the row. The function $\sin(\frac{2\pi jk}{2N})$ will be always antisymmetrical with respect to the center point. Thus, such pairs of symmetrically reflected points being multiplied by the sine functions will always have the same absolute value and the opposite sign. Then, being added together, such pairs will result in zero.

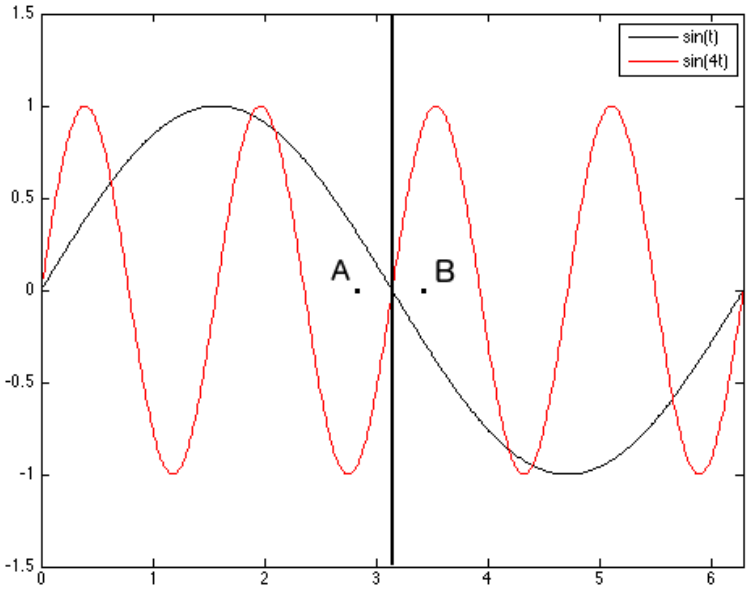


Figure 14.

Thus, the complex part of Fourier transformation in our case is also always zero. The above considerations will be valid also for the columns of points because they are symmetrized in the same manner.

Another case is the real part of Fourier transformation with the basis function $\cos(\frac{2\pi jk}{2N})$. The function is symmetric with respect to the center of each image row or column. The sum of products of light intensities and cosine basis functions will be thus the same for the left and the right part of each row or for the top and the bottom part of each column. This feature of cosine basis makes it possible to compute the transformation only for the upper left quadrant of the symmetrized image.

As was mentioned above the rows and the columns of the symmetrized image are symmetrical with respect to the point $\frac{N}{2}-1+\frac{1}{2}$. Basis cosine functions must be also symmetrical with respect to this point and the transformation result must be mirrored with respect to the borders , so we should use

the version of fast cosine transform, that called DCT-2. Explicit expression for this type of cosine transformation is showed below.

$$\hat{L}(l,k)=\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} C_N(l)C_M(k)\cdot L(i,j)\cdot\cos\left(\frac{\pi}{N}\cdot\left(i+\frac{1}{2}\right)l\right)\cdot\cos\left(\frac{\pi}{N}\cdot\left(j+\frac{1}{2}\right)k\right) \quad (5.4.1)$$

where

$$C_A(n)=\begin{cases}\sqrt{\frac{1}{A}}\ , & if\ x=0\\ \sqrt{\frac{2}{A}}\ , & if\ x>0\end{cases}$$

An exact proof that DCT 2 for the part of image corresponds to the Fourier transformation of the whole symmetrized image can be found for example in the solution of Homework Assignment H3 Problem 2 Lecture 6 [12].

Back cosine transformation.

The back cosine transformation can be defined as an analog to the forward cosine transformation. An explicit expression of this transformation is shown below (further details can be found, for example, in [12]) .

$$L(i,j)=\sum_{l=0}^{N-1}\sum_{k=0}^{M-1} C_N(l)C_M(k)\cdot\hat{L}(l,k)\cdot\cos\left(\frac{\pi}{N}\cdot\left(i+\frac{1}{2}\right)l\right)\cdot\cos\left(\frac{\pi}{N}\cdot\left(j+\frac{1}{2}\right)k\right)$$

5.5 1D fast Fourier and cosine transformation.

If the algorithms are implemented directly, the complexity of the forward and back Fourier transformation for the 2D image belongs to the class $O(N^2\cdot M^2)$. This is relatively easy to see, because for each point in Fourier domaine all points in spatial domain have to be iterated. Even for a 512 x 512 image it will make some problem, especially considering the fact that both back and forward transformations must be done for each color channel separately. Thus, we should try to optimize these computations to find a faster algorithm. There is a set of algorithms, called Fast Fourier Transformation, that can accelerate the Fourier transformation significantly. One of the most popular and simple algorithms from this set is Cooley-Tookey algorithm. In case when image size can be represented as a power of two, its complexity belongs to the class $O(N\cdot\log(N))$ in 1D case. This is much better than $O(N^2)$ for the direct implementation.

This algorithm uses some properties of the complex exponent, such as periodicity and antisymmetrie relative to the phase: $e^{ix}=e^{i(x+2\cdot\pi)}$ and $e^{ix}=-e^{i(x+\pi)}$. I will describe the algorithm briefly.

The whole set of input points is separated on the odd and even parts with respect to the index:

$$\hat{L}(k)=\sum_{j=0}^{N/2-1} L(2\cdot j)\cdot \mathrm{e}^{-\frac{2\cdot\pi\cdot i}{N}\cdot j\cdot 2\mathrm{k}}+\sum_{j=0}^{N/2-1} L(2\cdot j+1)\cdot \mathrm{e}^{-\frac{2\cdot\pi\cdot i}{N}\cdot j\cdot(2\mathrm{k}+1)}$$

this expression can be transformed as the following

$$\hat{L}(k)=\sum_{j=0}^{N/2-1} L(2\cdot j)\cdot \mathrm{e}^{-\frac{2\cdot\pi\cdot i}{N/2}\cdot j\cdot k}+\mathrm{e}^{-\frac{2\cdot\pi\cdot i}{N}\cdot k}\cdot\sum_{j=0}^{N/2-1} L(2\cdot j+1)\cdot \mathrm{e}^{-\frac{2\cdot\pi\cdot i}{N/2}\cdot j\cdot k}=E(k)+\mathrm{e}^{-\frac{2\cdot\pi\cdot i}{N}\cdot k}\cdot O(k)$$

where E is the even subset and O is the odd subset. Then using the properties of the complex exponent described above the amount of calculations can be reduced:

$$\hat{L}(k)=\begin{cases} E(k)+\mathrm{e}^{-\frac{2\cdot\pi\cdot i}{N}\cdot k}\cdot O(k)\ , & if\ k=0..N/2-1 \\ E(k-N/2)-\mathrm{e}^{-\frac{2\cdot\pi\cdot i}{N}\cdot k}\cdot O(k-N/2)\ , & if\ k=N/2..N-1 \end{cases} \qquad (5.5.1)$$

So we have to calculate the subsets E and O only for $k=0..\mathrm{N}/2$, to restore the rest of the $L(k)$ values . Further, if the size of the original set can be expressed as a power of two, the steps described above can be made recursively since the subsets E and O are ordinal Fourier transformations for N/2 elements.

As was mentioned above, in our case due to the image domain symmetrization, instead of Fourier transformation on the whole symmetrized image the discrete cosine transformation can be calculated only for a quarter of the symmetrized image. But direct implementation of cosine transformation, despite the fact that it needs fewer operations needed for the whole Fourier transformation, belongs to the same complexity class as the direct implementation of Fourier transformation, so it should be also accelerated. A some sort of direct using of Cooley-Tookey algorithm is not possible because the cosine function does not have all properties of the complex exponent needed for this algorithm. But there are different adaptations of the Cooley-Tookey idea for the case of cosine transformation. In some of them such functions appear as secant, that is an inversion of cosine: $sec\,x=\frac{1}{\cos x}$. This function has singularities in zero points of cosine. The closer we approaching to these points, the higher will be amplified errors caused by discretization and rounding inaccuracies. Therefore I have chosen an algorithm, that requires no singular functions by its implementation.

The idea and theoretical justification of this algorithm described in [14]. But I will explain it here a little bit deeper than it was made in the original page.

We will use the periodicity and symmetry of cosine function. Consider

$$C_N^{n,k} = \cos\left(\frac{\pi(2\mathrm{n}+1)k}{2\mathrm{N}}\right)$$

then

$$C_{N/2}^{n,k} = C_{N/2}^{N-1-n,k},\quad C_N^{n,2k+1} = -C_N^{N-1-n,2k+1}\ .$$

This can be justified with the following considerations:

$$
\begin{aligned}
C_{N/2}^{N-1-n,2k} &= \cos\left(\frac{\pi(2(N-1-n)+1)k}{N}\right) \\
&= \cos\left(-\frac{\pi(2n+1)k}{N}+2\pi k\right) \\
&= \cos\left(\frac{\pi(2n+1)k}{N}\right) \\
&= C_{N/2}^{n,k}
\end{aligned}
$$

and

$$
\begin{aligned}
C_{N}^{N-1-n,2k+1} &= \cos\left(\frac{\pi(2(N-1-n)+1)(2k+1)}{2N}\right) \\
&= \cos\left(-\frac{\pi(2n+1)(2k+1)}{2N}+\pi(2k+1)\right) \\
&= -\cos\left(\frac{\pi(2n+1)(2k+1)}{N}\right) \\
&= -C_{N}^{n,2k+1}
\end{aligned}
$$

Thus, the amount of computations in cosine transformation can be reduced twice because of the cosine symmetry. Furthermore, to construct an recursive algorithm let us again separate the original set of $\hat{L}(k)$ on two subsets – the odd and the even:

$$\hat{L}(2k)=\sum_{n=0}^{N/2-1}(L(n)+L(N-n-1))C_{N/2}^{n,k}, k=0..N/2-1 \quad (5.5.2)$$

$$\hat{L}(2k+1)=\sum_{n=0}^{N/2-1}(L(n)-L(N-n-1))C_{N}^{n,2k+1}, k=0..N/2-1 \quad (5.5.3)$$

For our recursive function it would be good to transform the coefficients C in (5.5.3) to the same form as (5.5.2). To achieve this we will first use the famous trigonometric formula, that replaces a product of two cosines with the sum cosines:

$$
\begin{aligned}
C_{N}^{n,1}\cdot C_{N/2}^{n,k} &= \cos\left(\frac{\pi(2n+1)}{2N}\right)\cdot\cos\left(\frac{\pi(2n+1)k}{N}\right) \\
&= \frac{1}{2}\cdot\cos\left(\frac{\pi(2n+1)(2k+1)}{2N}\right)+\frac{1}{2}\cdot\cos\left(\frac{\pi(2n+1)(2k-1)}{2N}\right) \quad (5.5.4) \\
&= \frac{1}{2}(C_{N}^{n,2k+1}+C_{N}^{n,2k-1})
\end{aligned}
$$

This means, that if we make a N/2-point discrete cosine transformation of the following expression

with coefficients from (5.5.2):

$$B(n)=(L(n)-L(N-n-1))\cdot C_N^{n,1} \quad (5.5.5)$$

we will get, using (5.5.4) and (5.5.3), the following expression:

$$\begin{aligned} \hat{B}(k) &= \sum_{n=0}^{N/2-1} B(n)\cdot C_{N/2}^{n,k} \\ &= \frac{1}{2}\cdot(\hat{L}(2k+1)+\hat{L}(2k-1)),\ k = 0 \,..\, \frac{N}{2}-1 \end{aligned} \quad (5.5.6)$$

So for $\hat{B}(k)$ we use the same transformation as in (5.5.2). The initial coefficients $\hat{L}(2k+1)$ can be restored using 5.5.6 and considering the fact, that $\hat{B}(0)=\hat{L}(1)$:

$$\hat{L}(2k+1) = 2\cdot B(k)-L(2k-1),\ k = 1 \,..\, \frac{N}{2} \quad (5.5.7)$$

Since the transformation (5.5.2) and (5.5.6) are ordinal discrete cosine transformations for N/2 elements we can use the recursion in this case too. For the recursive processing using the above scheme it is also required that the size of the initial set can be expressed as a power of two.

So using all above considerations the much more effective algorithm, than the direct cosine transformation, can be developed for forward cosine transformation. The optimization is achieved due to the reduction the sum and product operations while iterating through n and due to the usage of the recursive computation of the halved cosine transformation (5.5.2).

Listing 4 contains a pseudocode of such recursive function used in my experiments.

The algorithm divides the input set in odd and even elements. Then for odd subset $B(n)$ is calculated using (5.5.6). After that for the even subset and for the set $B(n)$ the whole function will be called recursively. The recursion stops if there are only two elements in the input set. For these two elements the ordinary discrete cosine transformation will be performed. From the results of this recursive call for both subsets an output array will be then composed using (5.5.7) and (5.5.2).

Algorithm complexity estimation.

At each recursive call for the input set of N elements there are two recursive sub calls for N/2 elements required. Moreover, at each call there are $O(N)$ additional operations performed. This corresponds to the following recursive scheme:

$$f(N) = 2\cdot f(N/2)+O(N) \quad (5.5.8).$$

This type of scheme belongs according to [15] to the linear-logarithmic complexity class $O(N\cdot\log(N))$.

Listing 4 Fast cosine forward transformation algorithm

Input: input - array of numbers in spatial domain
n – number of elements in input
Output: output – array of numbers in cosine domain

```
fastCosineTransform(input: array of double, n: int)
{
   // empty output array
   output = new array(n);

   // cosine forward transformation for 2 points.
  // note, that  C_N^{0,0} = C_N^{1,0} = 1  and  C_N^{0,1} = -C_N^{1,1} = 1/sqrt(2)
   if(n == 2){
      output[0] = input[0] + input[1];
      output[1] = (input[0] – input[1])*(1/sqrt(2));
      return output;
   }

   // splitting the input set on the odd and even subsets and computing B(n) for the odd part
   even = array(n/2);
   odd = array(n/2);

   for(int i = 0; i < n/2; i++){
      even[i] = input[i] + input[n-1-i];
      odd[i] = (input[i] - input[n-1-i])*cos(PI*(2*i + 1) / (2*n));
   }

   // recursive call of the forward transformation
   even = fastCosineTransform(even, n/2);
   odd = fastCosineTransform(odd, n/2);

   //restoring output set from the even and odd subsets
   for(int i = 0; i < n/2; i++){
      output[2*i] = even[i];

      //restore L from B as noted in (5.5.7)
      if(i == 0){
         output[1] = odd[0];
      }
      else{
         output[2*i+1] = 2*odd[i] - output[2*i-1];
      }
   }

   return output;
}
```

5.6 1-D fast cosine backward transformation.

Instead of a schema for the back transformation proposed in [14] I have simply inverted the above algorithm for the forward transformation. It was made to make the backward transformation simpler, as the original version contains additional optimizations and therefore is more complex.

Just as was described above I divide again the input set of elements in cosine domain on even and odd subsets. Using the odd elements I restore $\hat{B}(k)$ with the following formula:

$$B(k) = \frac{1}{2}\cdot(\hat{L}(2k+1)+L(2k-1)),\quad k = 1 \;..\; \frac{N}{2}$$

where the first element can be found in the same way as described above $\hat{B}(0)=\hat{L}(1)$. After that the backward cosine transformation routine is recursively called for the both N / 2 subsets. The recursion stops when there are only two elements in the input set. For these two elements the inverted cosine transformation is performed.

After the recursive call both resulting subset are mixed in one set using the following expressions:

$$L(n)=E(n)+O(n)/C_N^{n,1} \quad (5.6.1)$$

$$L(N-n-1)=E(n)-O(n)/C_N^{n,1} \quad (5.6.2)$$

where $E(n)$ and $O(n)$ are the even and odd subsets contained result of the N/2 back transformation. These two expressions (5.6.1) and (5.6.2) are the inverted versions of $L(n)+L(N-n-1)$ from (5.5.2) and $(L(n)-L(N-n-1))\cdot C_N^{n,1}$ from (5.5.5) .

The fast cosine back transformation algorithm is on Listing 5.

For this algorithm, analogues to the forward transformation, the same recursive scheme (5.5.8) can be applied and therefore this algorithm has also linear-logarithmic complexity $O(N\cdot\log(N))$.

Listing 5 Fast cosine backward transformation algorithm

Input: input - array of points in cosine domain
n – number of elements in input
Output: output – array of points in spatial domain

```
fastCosineBackTransform(input: array of double, n: int)
{
   // empty output array
   output = new array(n);

   // inverted forward transformation
   if(n == 2){
      output[0] = input[0]*0.5 + input[1]*(1/sqrt(2));
      output[1] = input[0]*0.5 – input[1]*(1/sqrt(2));
      return output;
   }

   // splitting the input set on the odd and even subsets
   //and computing L(2*n+1) for the odd part
   even = array(n/2);
   odd = array(n/2);

   for(int i = 0; i < n/2; i++){
      even[i] = input[2*i];
      if(i == 0){
         odd[0] = input[1];
      }
      else{
         odd[i] = (input[2*i+1] + input[2*i-1])/2;
      }
   }

   // recursive call of the forward transformation
   even = fastCosineBackTransform(even, n/2);
   odd = fastCosineBackTransform(odd, n/2);

   //restoring output set from the even and odd subsets using 5.6.1 and 5.6.2
   for(int i = 0; i < n/2; i++){
      output[i] = (even[i] + odd[i] / cos(PI*(2*i + 1) / (2*n))) / 2;
      output[n - i - 1] = (even[i] - odd[i] / cos(PI*(2*i + 1) / (2*n))) / 2;
   }

   return output;
}
```

5.7 2D fast cosine forward and backward transformation

To develop a 2D fast cosine transformation algorithm consider the following properties of the 1D transformation. Let us transform the expression (5.4.1) with simple arithmetic operations to get the following formula:

$$\hat{L}(l,k)=\sum_{i=0}^{N-1} C_N(l)\cdot\tilde{L}(i,k)\cdot\cos\left(\frac{\pi}{N}\cdot\left(i+\frac{1}{2}\right)l\right)$$

where

$$\tilde{L}(i,k)=\sum_{j=0}^{M-1} C_M(k)\cdot L(i,j)\cdot\cos\left(\frac{\pi}{N}\cdot\left(j+\frac{1}{2}\right)k\right)$$

Practically this means that we can apply the 1-D forward cosine transformation to all rows of image matrix and after that to all columns. The result of this operation will be a matrix in the cosine domaine. Such approach together with the fast cosine transformation reduces complexity of the transformation from $O(M^2\cdot N^2)$ to $O(N\cdot M\log(M)+M\cdot N\log(N))$ in case of the fast cosine transformation. This algorithm is implemented on Listing 6.

Listing 6 2D fast cosine transformation algorithm

Input: input – NxM array of points in spatial domain
n, m – height and with of input
Output: output – array of points in cosine domain

```
fastCosineTransform2D(input: array of array of double, m:int, n: int)
{
   // empty output array
   output = new array(n, m)

   for(int i = 0; i < n; i++){
       fastCosineTransform(input[i], m);
   }

   // buffer array to save
   buff = new array(n);

   for(int j = 0; i < m; i++){
      //copy column j in the buffer
      for(int i = 0; i < n; i++){
          buff[i] = output[i, j];
      }
       // rewrite column j in output array
       column(j, output) = fastCosineTransform(buff, n);
   }

   return output;
}
```

For the back transformation we can use the same approach with the same considerations. The only difference is that instead of forward 1D transformation we must use the backward 1D transformation.

5.8 Normalization of the results.

The authors of the paper [5] use in their software for resetless Retinex the following affine transformation for the normalization of the results:

$$\bar{I} = \left(I - M_{data} \right) \cdot \frac{\sigma_{ref}}{\sigma_{data}} + M_{ref}$$

where σ_{ref} and M_{ref} are the standard deviation and the mean of the original image, σ_{data} and M_{data} are those for the processed image.

This method brings practically the mean and the standard deviation of the processed image to the mean and standard deviation of the original image. Thus, this method does not automatically correct the white balance. Some sort of white balance correction could be performed if we set, for example, the M_{ref} for all color channels to the middle of intensity diapason (127 in our case).

I will also use the authors approach in my implementation because it is simple and preserves the average intensity value for each channel. The pseudocode of my implementation is on Listing 7.

Listing 7 Normalization of results

Input: inputData – NxM array of processed data
inputRef - NxM array of reference data
n, m – height and with of input
Output: **output – array of normalized data**

```
// empty output array
output = new array(n, m)

//mean and standard deviation of reference image
double meanRef = 0;
double devRef = 0;
//mean and standard deviation of processed image
double meanData = 0;
double devData = 0;

for(int i = 0; i < n; i++){
   for(int j = 0; j < m; j++){
      meanRef += inputRef[i,j];
      devRef += inputRef[i,j]*inputRef[i,j];

      meanData += inputData[i,j];
      devData += inputData[i,j]*inputData[i,j];
   }
}
meanRef /= n*m;
meanData /= n*m;
//compute the standard deviation
devRef  = sqrt(devRef  / n*m – meanRef *meanRef);
devData = sqrt(devData  / n*m – meanData *meanData);


for(int i = 0; i < n; i++){
   for(int j = 0; j < m; j++){
      output[i,j] =  (inputData[i,j] – meanData)*(devRef/devData) + meanRef;
   }
}
```

5.9 Extrema Retinex and Gauss-Seidel Method

There are only a few words about the solution of extrema Retinex equation in the paper [5]. The authors have mentioned that they have used a Gauss-Seidel method to solve the equation and that they have used a simple scaling method for the data normalization. So I have made all the following considerations and calculations with the help of literature and the Internet.

As was also mentioned in the paper [5] in case of Dirichlet border conditions within the image domain the Fourier transformation method of solving the equation will not work. Thus, some other method should be used.

Gauss-Seidel approach solves a system of finite differences equation obtained as a result of discretization of the original PDE. This is a simple and easy-to-understand iterative method. There are other methods, more complex and more effective, such as "Conjugate Gradients" or "Successive Over-Relaxation". But I don't try to find the best method to solve this PDE, I just want to make a few experiments with this model and the Gauss-Seidel method is suitable for this purpose.

Before solving this equation I would like to prove that the solution exits and the method converges. Each linear system of equations $Ax=b$ with symmetric matrix A can be considered as a gradient of the following quadratic form (more details in [20])

$$\frac{1}{2}x^T Ax - b^T x + c \quad .$$

The solution of the system lies in a point, where the gradient is equal to zero, so this should be an extremum. This quadratic form has an unique extremum, if the matrix A is positive or negative definite. Positive definiteness of the matrix A is also a sufficient criteria for the convergence of the Gauss-Seidel method. So I will show that the matrix of our PDE is positive definite in case of Dirichlet conditions (the idea of this prove was found in [11]).

In construction of the following equations I have used the Neuemann conditions, that follow from the image domain symmetrization and Dirichlet conditions of the reset Retinex.

For the four corner points top left, top right, bottom left and bottom right the finite differences equations have the following form:

$$\begin{aligned}
f(x_{0,0}) &= 4\cdot L(x_{0,0}) - L(x_{-1,0}) - L(x_{0,-1}) - L(x_{1,0}) - L(x_{0,1}) \\
f(x_{0,N-1}) &= 4\cdot L(x_{0,N-1}) - L(x_{-1,N-1}) - L(x_{0,N-2}) - L(x_{1,N-1}) - L(x_{0,N}) \\
f(x_{M-1,0}) &= 4\cdot L(x_{M-1,0}) - L(x_{M-2,0}) - L(x_{M-1,-1}) - L(x_{M,0}) - L(x_{M-1,1}) \\
f(x_{M-1,N-1}) &= 4\cdot L(x_{M-1,N-1}) - L(x_{M-2,N-1}) - L(x_{M-1,N-2}) - L(x_{M,N-1}) - L(x_{M-1,N})
\end{aligned} \qquad (5.9.1)$$

From the Neumann boundary conditions (3.18) it follows also

$$\begin{aligned}
L(x_{i,0}) &= L(x_{i,-1}) \\
L(x_{0,j}) &= L(x_{-1,j}) \\
L(x_{i,N-1}) &= L(x_{i,N}) \\
L(x_{M-1,j}) &= L(x_{M,j})
\end{aligned} \qquad i=0..M-1,\ j=0..N-1 \qquad (5.9.2) \quad .$$

(5.9.1) and (5.9.2) give as 2 in the corners of matrix A (see below for 4 x 4 image). The same

considerations are valid to other border points, that marked with 3 in the matrixes A and B.

Suppose that we have a 4 x 4 image. We can now construct the matrix of the equation system of our PDE for this image. I will write it down in the form of a block matrix.

$$A=\begin{matrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{matrix} \qquad B=\begin{matrix} 3 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 3 \end{matrix} \qquad I=\begin{matrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{matrix}$$

The whole block matrix of our system has the following form:

$$M=\begin{matrix} A & I & 0 & 0 \\ I & B & I & 0 \\ 0 & I & B & I \\ 0 & 0 & I & A \end{matrix}$$

and the whole system can be written as $M \cdot x = f$.

So the solution for our PDE is the solution of this equation system (without the Dirichlet conditions, that have to be added extra). A direct solution of this equations system does not make sense because for real images this system will be very large. Thus, we will apply the Gauss-Seidel method. This method has the following iteration formula:

$$x_i^{n+1}=\frac{1}{m_{i,i}}\left(f_i-\sum_{k=1}^{i-1} m_{i,k}x_k^{n+1}-\sum_{k=i+1}^{m} m_{i,k}x_k^{n}\right) \quad .$$

As was mentioned above it converges if the matrix M is symmetric and positive definite. The matrix is clearly symmetric. Let's show the positive definiteness.

We know that a matrix is positive definite if all its eigenvalues are positive. We can show with the Gershgorin theorem that all eigenvalues of our matrix are not negative. Indeed, all Gershgorin circles in our case have the following form:

$$|2-\lambda|\le 2 \ \cup \ |3-\lambda|\le 3 \ \cup \ |4-\lambda|\le 4$$

and λ can not be negative.

Further proof that all eigenvalues strictly lager than zero I will show analogous to the proof in [11] for the case of Dirichlet condition set on the domain boundary and without the Neumann conditions.

Suppose that $\lambda=0$. In this case $M \cdot x = 0$ also. Let's say further that there is no Dirichlet conditions set . Find such index i, that $|x_i|=max(|x|)$. $m_{i,i}$ can be 2,3 or 4. The amount of non-diagonal non-zero elements in the row i is also 2,3 or 4 and all these elements are equal to -1. If further x_i is the minimal by the absolute value component of the vector x, then for $x_i>0$ all components x_j for $a_{i,j}\neq 0, j\neq i$ have to be also positive and for $x_i<0$ - negative because of $a_{i,i}x_{i,i}+\sum_{j=1}^{N-1} a_{i,j}x_{i,j}=0$. Furthermore, all the components x_j and the component x_i have to be equal because the x_i is the maximum and all $a_{i,j}\neq 0, j\neq i$ are equal to -1. Than we can replace i

with j and make again all above considerations . In this way we will show that all components of the vector x have to be equal.

If we now set a Dirichlet condition at least at one point, then the correspondent column can be moved from the matrix to the right side of the equation system. In this case if we repeat all above operations and considerations we will see that at least in one column, that had non-zero elements on the places of removed column, the condition $a_{i,i}x_{i,i}+\sum_{j=1}^{N-1} a_{i,j}x_{i,j}=0$ can not be satisfied. We have thus a contradiction. This proves that $\lambda>0$.

The algorithm is shown on Listing 8. As a normalization method for this algorithmm I have used a simple linear scaling to the interval 0..255. As a set of extrema I use one point with the maximal light intensity per channel.

The Gauss-Seidel method is less efficient then the cosine transformation.

Listing 8 Gauss-Seidel method

Input: input - array of original image points
F – the right side of the PDE
delta – tolerance
n – number of elements in input
Output: output – array of processed points

```
gaussSeidel(input: array of double, n: int, delta: float, f: array of double)
{
   // empty output array
   output = new array(n);
   do{
      float dmax = 0;
      for(int i = 0; i < n; i++){
         for(int j = 0; j < m; j++){
            //calculate the matrix coefficients
            if(extremum){
               output[i,j] = 0;
            }
            if(j == 0){
               left = output[i, j];
            }else{
               left = output[i, j - 1];
            }
            if(i == 0){
               top = output[i, j];
            }else{
               top = output[i - 1, j];
            }
            if(j == w - 1){
               right = output[i, j];
            }else{
               right = output[i, j + 1];
            }
            if(i == h - 1){
               left = output[i, j];
            }else{
               left = output[i + 1, j];
            }
         }
      }
      temp = output[i,j];
      output[i,j] = 0.25(right + left + top +bottom + f[i,j]);
      //difference between the current and the previois step
      dmax += abs(output[i,j] – temp);
   }while(dmax > delta);
   return output;
}
```

6. Experimental results.

The results of Retinex algorithms application are shown below. There are some tests with synthetic images and experiments with real world textures and scenes. Differences between various implementations or between the same implementation with different parameters will be commented.

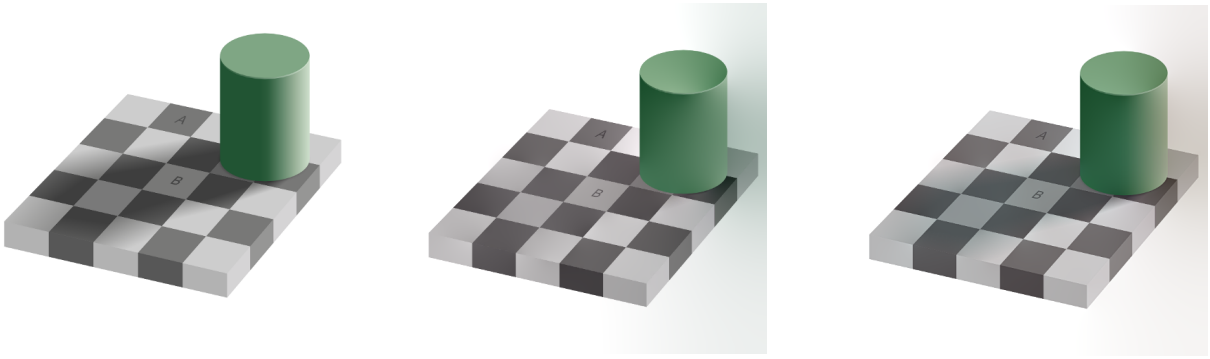


Figure 15. Adelson's checker shadow illusion (512 x 512). From left to right: the original image. Resetless Retinex PDE (RRPDE) processing result with threshold 5. The extended resetless Retinex PDE (eRRPDE) processing result with 11x11 blurring kernel, $\sigma = 1$, $\hat{\delta} = 5$ and $\alpha = 0.8$. In this case there are no complex textures, all surfaces are flat. So the original RRPDE works good. Original image source: [5]

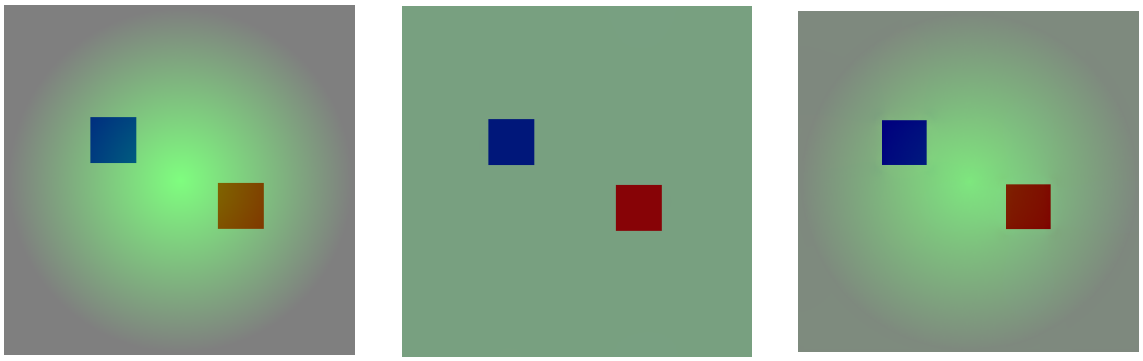


Figure 16. Color constancy test (512 x 512).. From left to right: the original image. RRPDE with threshold 2. ERPDE with the 21x21 kernel $\sigma = 10$, $\hat{\delta} = 3$, $\alpha = 0.5$. The RRPDE tends to clear the colors but it removes totally the gradient. eRRPDE preserves the gradient and clears colors, looks more natural.

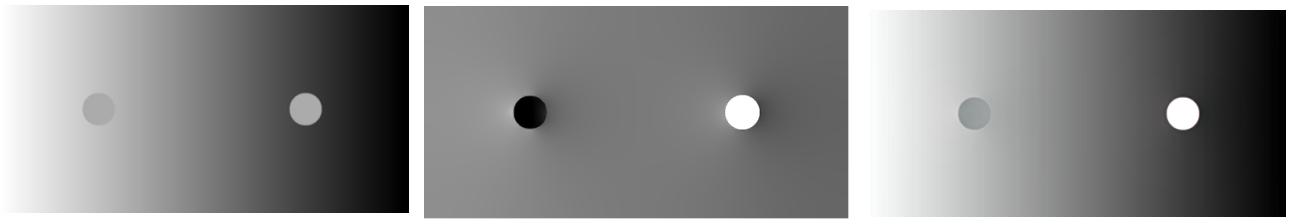


Figure 17. Simultaneous contrast (512 x 256). From left to right: the original image. RRPDE with threshold 2. eRRPDE with the 11x11 kernel $\sigma=2$, $\hat{\delta}=3$, $\alpha=0.5$. The same as above. RRPDE has the right tendency but eRRPDE looks more natural because it simultaneously preserves the background and makes the circles brighter or darker. Original image source: [5]

Figure 18. Grass (512 x 512).. Top left: original image. Top right: RRPDE with t=20. Bottom left: eRRPDE 21x21, $\sigma = 10$, $\hat{\delta} = 10$, $\alpha = 0.5$. Bottom right: eRRPDE 21x21, $\sigma = 10$, $\hat{\delta} = 5$, $\alpha = 1$. As we can see the RRPDE can not remove the shadow even with t = 20. Higher threshold will destroy texture details. Whereas eRRPDE makes the shadow brighter and increase contrast in small details in light area, it does not destroy any of those details. Thus, the eRRPDE looks more natural and tends to what I have really seen with my eyes.

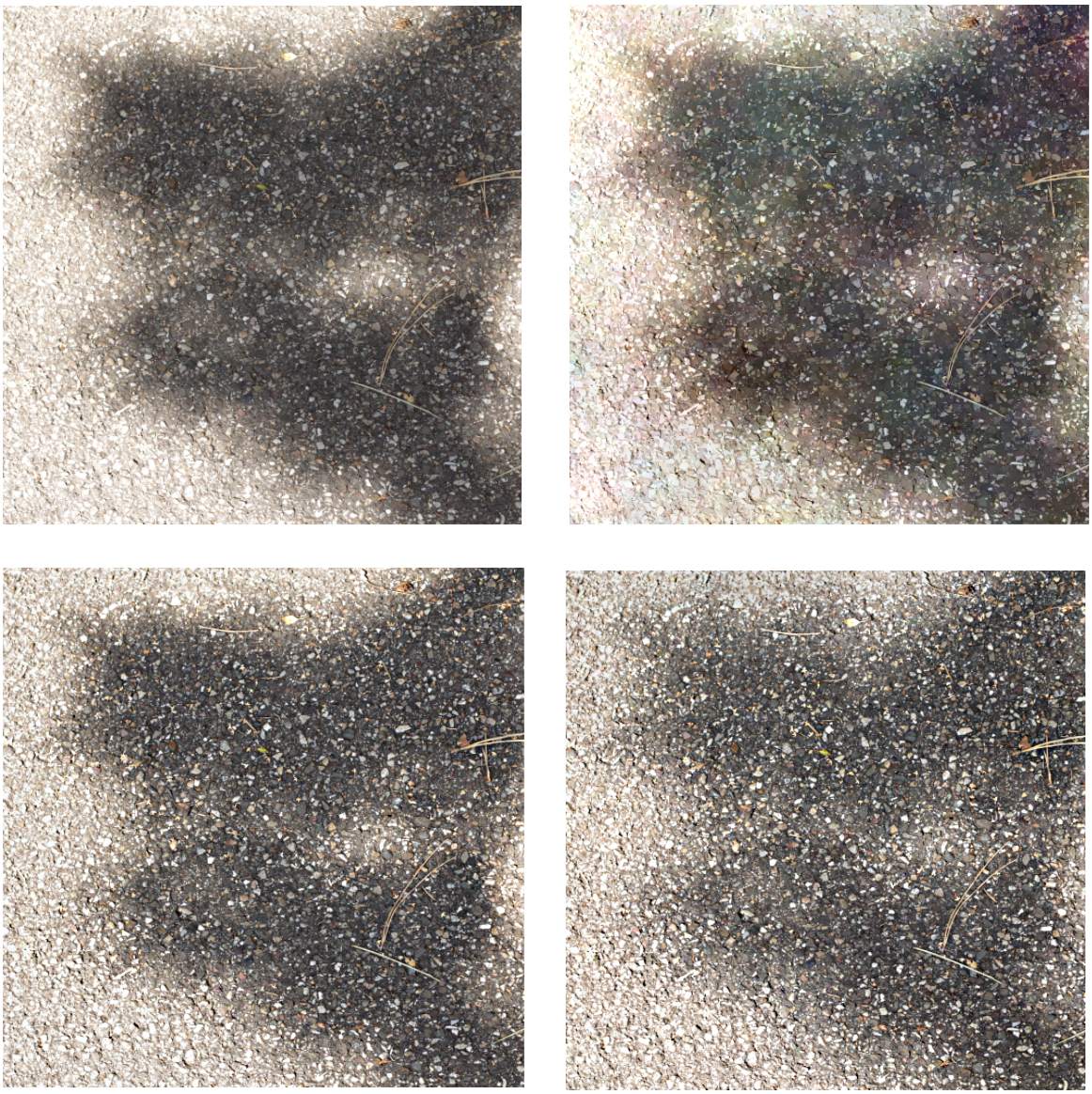


Figure 19. Asphalt (512 x 512).. Top left: original image. Top right: RRPDE with t = 20. Bottom left: eRRPDE 21x21, $\sigma = 10$, $\hat{\delta} = 10$, $\alpha = 0.5$. Bottom right: eRRPDE 21x21, $\sigma = 10$, $\hat{\delta} = 5$, $\alpha = 1$. The same as above. The RRPDE can not remove the shadow and it destroy texture details within the shadow with t = 20. Higher threshold will destroy texture completely. eRRPDE makes the bright detail within the shadow brighter and increase contrast of small details in general. It does not destroy any details. Here the eRRPDE looks also more natural and tends to what I have really seen.

Figure 20. Needles (512 x 512).. Top left: original image. Top right: RRPDE with t = 20. Bottom left: eRRPDE 21x21, $\sigma = 10$, $\hat{\delta} = 10$, $\alpha = 0.5$. The RRPDE can not remove the shadow without destroying its details. The eRRPDE makes the shadow more brighter and increases the contrast of texture details in lights, that looks much more natural.

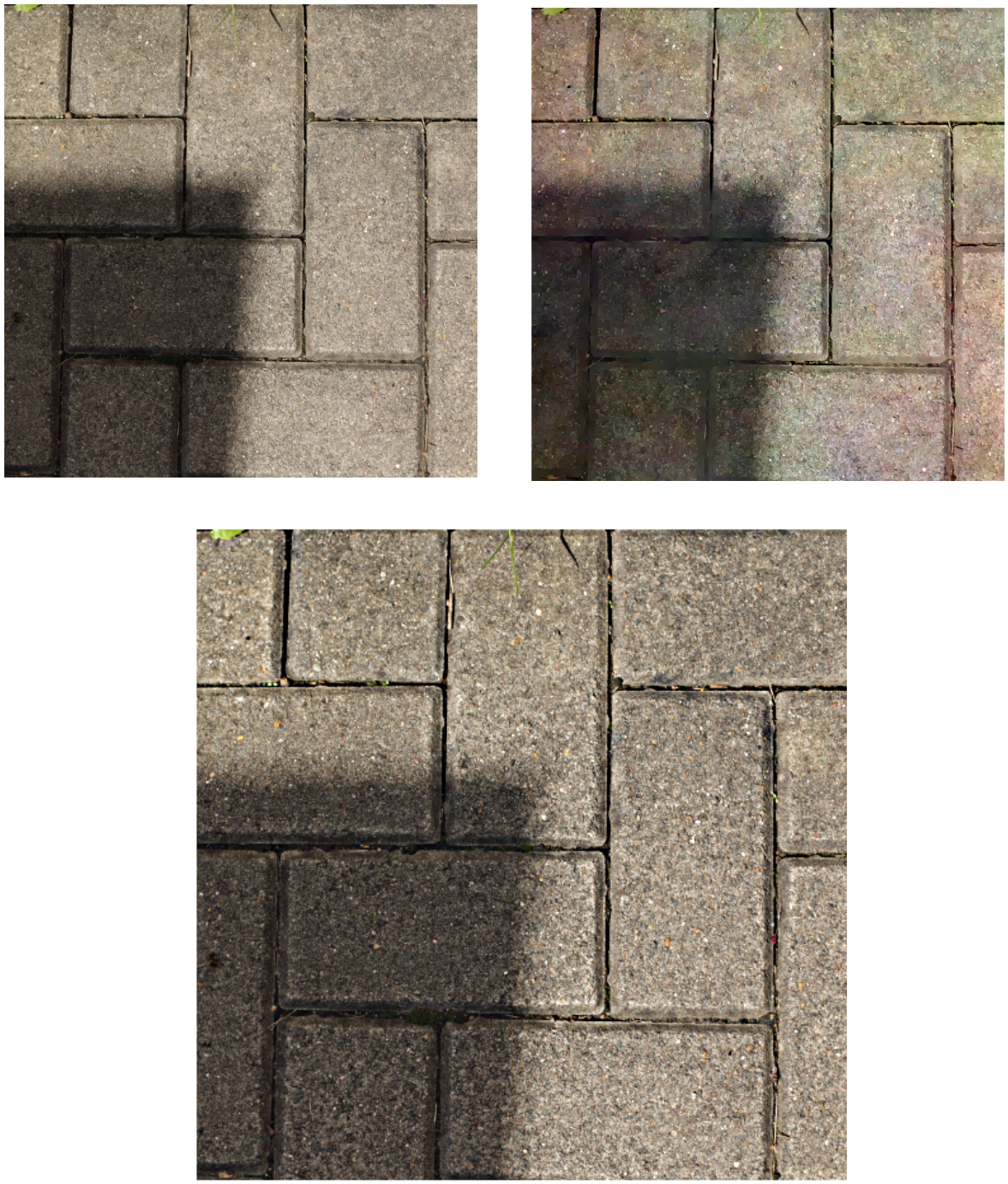


Figure 21. Pavement (512 x 512).. Top right: RRPDE with t = 20. Bottom: eRRPDE 21x21, $\sigma = 10$, $\hat{\delta} = 10$, $\alpha = 0.5$. The same as above. eRPDE makes the shadow brighter and increases the contrast of texture details.

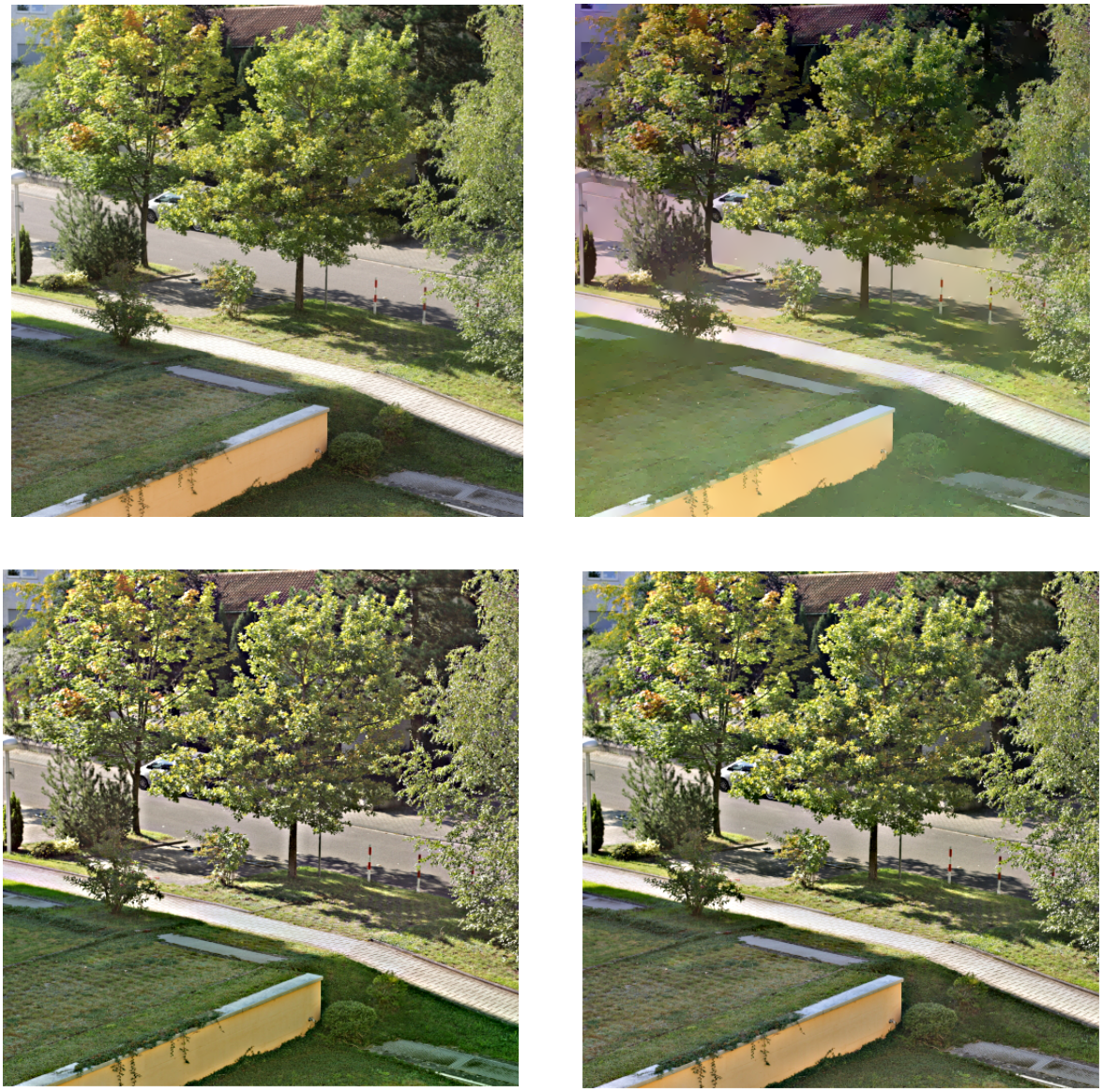


Figure 21. The street (512 x 512). Top right: RRPDE with t = 20. Bottom left: eRRPDE 21x21, $\sigma = 10$, $\hat{\delta} = 10$, $\alpha = 0.5$. Bottom right eRRPDE 41 x 41, $\sigma = 15$, $\hat{\delta} = 20$, $\alpha = 0.5$. In this case RRPDE works better as before. The algorithm makes shadows brighter but parallel it destroys the grass texture. The better results could be explained by the fact, that textures here are more flat as on the images above. The eRRPDE makes also parts of shadow brighter and increases the contrast in small details such as leaves on the trees. It preserves also all textures, that looks better. The problem here is that there are details with different scale on the same image and some details have the same size as shadows. It is difficult to find such a kernel, that will not change significantly all details and will remove only the shadows.

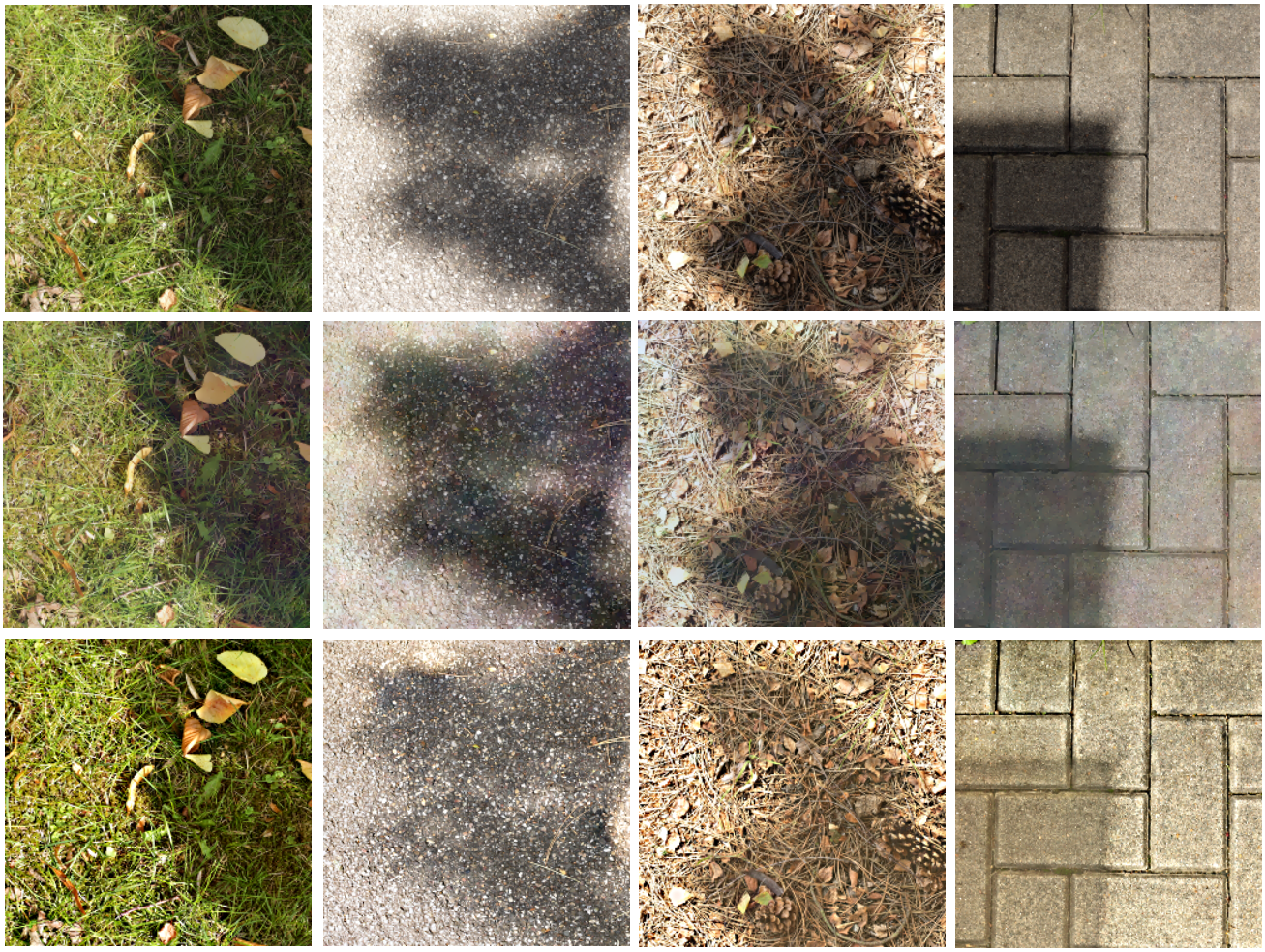


Figure 22. Extrema Retinex. The first row: original images. The second row: extrema Retinex with the maximum acceptable threshold (higher threshold will remove texture details). The third row: the extended extrema Retinex with manually increased contrast (+50%). Here one can see the same tendency: the extended version of the PDE can better remove soft gradients in case of complex textures as the original one.

7. Conclusions.

The both resetless and reset Retinex PDEs were implemented in this work as proposed in [5]. For this purpose I have used a fast cosine transformation and Gauss-Seidel algorithms (the possibility of using a fast Fourier transformation and Gauss-Seidel method was also mentioned in [5]). My implementation of the fast cosine forward transformation is based on the idea described in [14] and the back transformation was constructed by me as an inversion of the forward algorithm.

To understand the geometrical meaning of the both equations from the paper [5] a variational interpretation the PDEs was used. The equivalence of such interpretation with the original PDEs is mentioned in [16] and proved in [17]. From variational point of view the Retinex PDEs from [5] are the Euler-Lagrange equations for some optimization problem. In this interpretation the expected "Retinex" effects can be described as removing those gradient components, that lie under the used threshold and the following restoring of an image from the processed gradient field. The "restoring" in this context means a calculation of such an image, that has a gradient passing best to the processed gradient field.

I have tested the both Retinex algorithms on real images and found that in case of high detailed textures the original PDEs work not so good. The shadows can only be removed with the most texture details. Besides of that, numerous color artifacts appear. I have analyzed this behavior using variational interpretation of both PDEs. As a result of this analysis an extension of the original resetless PDE was proposed.

This extension uses another approach to remove soft gradients instead of the hard thresholding operator. A weighted difference between the gradient field of the original image and thresholded from the top gradient field of a blurred version of the image is used instead. My experimental results show a better ability of the extended PDE (for both versions: resetless and extrema) to suppress shadows in case of textures with lots of contrast details including the textured shadow borders. This extended approach makes the original PDE more local.

A problem with the extended version is that there can be details with different scale on the same image and some details might have the same size as shadows. It is difficult to find such a kernel, that will not change significantly all details and will remove only the shadows.

Further improvement of this PDEs is possible, for instance due to improvement of the "right side " of PDE. This could be a some sort of an edge detection algorithm that can separate small high contrast details and larger low contrast shadows or other multiplicative environmental effects. Also some sort of multi-scale Retinex [21] can be used for filtering the right side of the equation.

A white balance correction could be also added.

8. Bibliography.

1. E. H. Land, “The retinex theory of color vision,” Scientific American, vol. 237, no. 6, pp. 108–128, Dec 1977.

2. E. Land and J. McCann, “Lightness and retinex theory,” J. Opt. Soc. Amer., vol. 61, no. 1, pp. 1–11, Jan 1971.

3. E. H. Land, “Recent advances in retinex theory and some implications for cortical computations: color vision and the natural image” Proceedings of the National Academy of Sciences of the United States of America 80, 5163-9 (Aug. 1983)

4. D. Marini, “A computational approach to color adaptation effects,” Image and Vision Computing, vol. 18, no. 13, pp. 1005–1014, Oct 2000.

5. J. Morel, A.B. Petro, and C. Sbert, "A PDE Formalization of Retinex Theory", ;presented at IEEE Transactions on Image Processing, 2010, pp.2825-2837.

6. D. Marini, “A computational approach to color adaptation effects,” Image and Vision Computing, vol. 18, no. 13, pp. 1005–1014, Oct 2000.

7. E. Provenzi, L. D. Carli, A. Rizzi, and D. Marini, “Mathematical definition and analysis of the retinex algorithm,” J. Opt. Soc. Amer. A, vol. 22, pp. 2613–2621, 2005.

8. Horn, B.K.P., “Determining Lightness from an Image,” Computer Graphics and Image Processing, Vol. 3, No. 1, December 1974, pp. 277–299.

9. E. Land, “An alternative technique for the computation of the designator in the retinex theory of color vision.” Proceedings of the National Academy of Sciences of the United States of America 83, 3078-80 (May 1986).

10. Jobson, D. J., Rahman, Z.-u., and Woodell, G.A., “A Multiscale Retinex for Bridging the Gap Between Color Images and the Human Observation of Scenes,” IEEE Transactions on Image Processing 6, 965-76 (Jan. 1997).

11. Carola-Bibiane Schönelib. “Numerical Analysis Part 2.. Lecture 2.” University of Cambridge. Online: http://www.damtp.cam.ac.uk/user/cbs31/Teaching_files/c02.pdf

12. Joachim Weickert. “Image Processing and Computer Vision. Lecture slides.” Lecture slides. University of Saarland. Winter term 2013/2014. Online: http://www.mia.uni-saarland.de/Teaching/ipcv13.shtml

13. Samarsky A. A. “Vvedeniye v chislennye metody” (Introduction to Numerical Methods) M.: Nauka, 1982 (rus.: Самарский А. А. “Введение в численные методы” М.: Наука, 1982).

14. M. N. Yatsimirskiy, “Bystryi algoritm diskretnogo kosinusnogo preobrazovaniya” (“Fast cosine transformation algorithm”) J. vychisl. matem. i matem. fiz., 33:2 (1993), 303-305 (rus.: М. Н. Яцимирский, “Быстрые алгоритмы дискретного косинусного преобразования”, Ж. вычисл. матем. и матем. Физ., 33:2 (1993), 303–305).

15. Gert Smolka, “Programmierung – eine Einführung in die Informatik mit Standard ML.” Oldenbourg, München 2008.

16. Dominique Zosso, Giang Tran and Stanley Osher "A unifying retinex model based on non-local differential operators ", Proc. SPIE 8657, Computational Imaging XI, 865702 (February 14, 2013);

17. A. Blake, "Boundary conditions for lightness computation in Mondrian World", ;presented at Computer Vision, Graphics, and Image Processing, 1985, pp.314-327.

18. V. A. Ilyin, E. G. Poznyak “Osnovy matematicheskogo analiza” (“Fundamentals of mathematical analysis”) - M.: Nauka.Fiz.-mat.lit, 1973 (rus.: Ильин В. А., Позняк Э. Г. Основы математического анализа . Ч. 2 : Учебник для вузов / В.А. Ильин, Э.Г. Позняк .- М. : Наука.Физ.-мат.лит. , 1973).

19. Oliver Labs, Frank–Olaf Schreyer. “Mathematik für Informatiker Teil 1,2 und 3” Lecture script. University of Saarland. Winter Term 2012/2013. Online: http://www.math.uni-sb.de/ag/schreyer/LEHRE/1213_MfI3/0910_MfI123_book.pdf

20. J. R. Shewchuck: “An Introduction to the Conjugate Gradient Method Without the Agonizing Pain” School of Computer Science Carnegie Mellon University. Online: http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf

21. D. J. Jobson, Z. Rahman, and G. A. Woodell, "A multi-scale Retinex for bridging the gap between color images and the human observation of scenes, " IEEE Transactions on Image Processing, 6(7), 1997.