

# **DEEP DIVE INTO HIGH DYNAMIC RANGE IMAGING (A MATLAB TUTORIAL)**

**Konstantinos Monachopoulos**, Software Engineer.

## ABSTRACT

High dynamic range imaging (HDRI) is an emerging field that has the potential to cause a great scientific and technological impact. This field is large and complex, with non-trivial relations to many different areas, such as image synthesis, computer vision, video and image processing, digital photography, special effects among others. For the above reasons, HDRI has been extensively researched over the past years and, consequently, the related scientific literature is vast. As an indication that the field is reaching maturity, tutorials and books on HDRI appeared. Moreover, this new resource has already reached interested practitioners in various application areas. In this whitepaper, we do not aim at covering the whole field of high dynamic range imaging and its applications, instead, we aim to cover the basic principles behind HDRI using **debevec** HDR approach and focus on resolving the HDR problem arising, both theoretical and practical. That is, the reconstruction of high dynamic range images from regular low dynamic range images using Matlab.

## TABLE OF CONTENTS

ABSTRACT .....	1
LIST OF FIGURES.....	3
LIST OF TECHNICAL DETAILS.....	4
CHAPTER 1: INTRODUCTION.....	5
CHAPTER 2: HDR BACKGROUND THEORY.....	6
2.1 COMBINING DIFFERENTLY EXPOSED IMAGES .....	6
2.2 IMAGES AND SAMPLE PIXELS PROPORTION .....	7
2.3 ESTIMATE THE CAMERA RESPONSE FUNCTION .....	8
2.3.1 ESTIMATE THE SMOOTHING FUNCTION .....	10
2.3.2 CONSTRUCT THE HIGH DYNAMIC RANGE RADIANCE MAP .....	10
CHAPTER 3: HDR IMAGING IMPLEMENTATION .....	11
3.1 IMAGE REGISTRATION .....	11
3.2 SMOOTHING FUNCTION .....	12
3.3 LOADING AND SAMPLING THE IMAGES .....	12
3.4 FINDING THE RESPONSE CURVE.....	13
3.5 CONSTRUCTING THE HIGH DYNAMIC RANGE RADIANCE MAP .....	15
CHAPTER 4: TONE MAPPING .....	20

## LIST OF FIGURES

Figure 1. Camera vs human eye .....	5
Figure 2. Human eye vs low-end device dynamic range.....	5
Figure 3. HDR imaging pipeline.....	6
Figure 4. Nonlinear blocks pipeline recovering radiance map.....	8
Figure 5. Exposure matrix .....	11
Figure 6. Image data set and their corresponding exposures.....	12
Figure 7. Weighted (hat) function. ....	12
Figure 8. Random pixels in exposure sequence.....	14
Figure 9. Response function coefficients (Matrix A, first part) .....	15
Figure 10. Positive and Negative Weighted coefficients .....	16
Figure 11. Smoothing coefficients (Matrix A, second part).....	17
Figure 12. Weighted Smoothing irradiance coefficients.....	17
Figure 13. Measured exposures array.....	18
Figure 14. Weighted measured exposures array (b array) .....	18
Figure 15. Camera response function in every channel.....	19
Figure 16. Irradiance HDR map.....	20
Figure 17. Reinhard local and global tonemapping operator.....	21

## LIST OF TECHNICAL DETAILS

Listing 1. HDR_main.m_matlab language technical details.....	22
Listing 2. ReadImagesMetaData.m_matlab language technical details.....	24
Listing 3. weight.m_matlab language technical details.....	25
Listing 4. makeImageMatrix.m_matlab language technical details.....	26
Listing 5. sample.m_matlab language technical details.....	27
Listing 6. gsolve.m_matlab language technical details.....	28
Listing 7. hdr.m_matlab language technical details.....	29
Listing 8. reinhardLocal.m_matlab language technical details.....	31
Listing 9. reinhardGlobal.m_matlab language technical details.....	33
Listing 10. makeLuminanceMap.m_matlab language technical details.....	34

## CHAPTER 1: INTRODUCTION

The dream of electronic vision is to mimic the capabilities of the human eye and possibly to go beyond in certain aspects. The eye and human vision overall are so versatile and powerful that any strategy to realize similar features with electronics and information technology have to be focused on certain aspects of man's powerful sense to collect the most comprehensive information on our physical world. Human eyes have extremely high visual range that can easily differentiate between very bright and dark regions in the same scene. Computer screens can't display nearly the dynamic range (ratio between dark and bright regions) as what is present in the real world, and as a result, cameras are not designed to capture even close to such a range.



Figure 1. Camera vs human eye.

When we photograph a scene, either with film or an electronic imaging array, and digitize the photograph to obtain a two-dimensional array of "brightness" values, these values are rarely true measurements of relative radiance in the scene. For example, if one pixel has twice the value of another, it is unlikely that it observed twice the radiance.

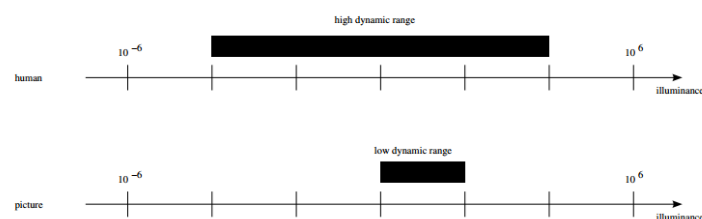


Figure 2. Human eye vs low-end device dynamic range

Instead, there is usually an unknown, nonlinear mapping that determines how radiance in the scene becomes pixel values in the image. This nonlinear mapping is hard to know beforehand because it is actually the composition of several nonlinear mappings that occur in the

photographic process. High Dynamic Range Imaging is a technique to emulate this behavior with the low end commercial cameras available in the market. By taking a series of pictures with different exposure settings the range can be covered. With this technique such a series of images can be combined into a single high dynamic range image called irradiance map. Irradiance maps are useful for representing true illumination values in image-based rendering applications and are useful for recording incident illumination and using this light to illuminate objects for realistic compositing.

## CHAPTER 2: HDR BACKGROUND THEORY

In a simple static setting, the process of HDR starts from image acquisition at various exposures of the same scene at quick succession so that effect of changing light conditions can be ignored. Despite using tripods for greater control, some perturbation is inevitable, therefore, image alignment is often necessary. In this whitepaper, we have implemented **Debevec** algorithms to calculate the radiance map. Tone mapping goes in reverse direction, the HDR values that are calculated, cannot be used on devices which has limited dynamic range. The idea of tone mapping is to reduce the dynamic range in a way that preserves the features in the HDR images and present colors which are pleasing and appreciable to human eyes.

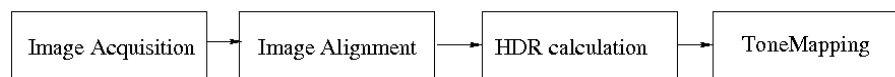


Figure 3. HDR imaging pipeline

### 2.1 COMBINING DIFFERENTLY EXPOSED IMAGES

HDR images may be created in two fundamentally different ways. The first method employs rendering algorithms and other computer graphics techniques. The second method employs conventional (LDR) photo cameras to capture HDR data. This may be achieved by photographing a static scene multiple times, varying the exposure time for each frame. Due to the limitations inherent in most digital image sensors, and to a lesser degree in film emulsions, it is not possible to capture the full dynamic range of an image in a single exposure.

However, by recording multiple exposures, a standard camera with the right software can create a single HDR image. These exposures are usually captured by the camera itself, although in the case of recovering HDR information from a single negative the same technique may be

applied during the film-scanning phase. By capturing multiple exposures, each image in the sequence will have different pixels properly exposed (under or over exposed). This way, each pixel will be properly exposed in at least one or more images in the sequence. It is therefore possible and desirable to ignore very dark and very bright pixels in the subsequent computations.

This approach generally requires the subject in front of the camera to remain still between shots, and toward this end the camera should be placed on a tripod avoiding ghost effect. This limits the range of photographs that may be taken. Fortunately, several techniques exist that align images, remove ghost effects, and reduce the effect of lens flare, thus expanding the range of HDR photographs that may be created. These techniques are not discussed here, we will only focus on the process of creating an HDR image in ideal conditions (no moving objects or light change in the image sequence).

## 2.2 IMAGES AND SAMPLE PIXELS PROPORTION

The minimum number of photographs that are required to extract a good quality high dynamic range image is two photographs, but whether two photographs are enough, can be understood in terms of the heuristic explanation of the process of film response curve recovery. If the scene has sufficiently many different radiance values, the entire curve can, in principle, be assembled by sliding together the sampled curve segments, each with only two samples. Note that the photos must be similar enough in their exposure amounts that some pixels fall into the working range of the film in both images, otherwise there is no information to relate the exposures to each other. Obviously, using more than two images with differing exposure times, improves performance with respect to noise sensitivity. The minimum number of images is computed by using the inequality:

$$M(N-1) > (B_{\max} - B_{\min})$$

Where  $M$  is the number of sample pixels,  $N$  is the number of images and  $B_{\max}$ ,  $B_{\min}$  are the maximum and minimum intensity values that pixels holds. Because the image data type is unsigned integer with eight bits values, the maximum and minimum values are 255 and 0. For optimization purposes we won't use all the pixels that describes every image, but we will sample a proportion of them. Based on the inequality above we can find the optimum number of sample pixels that will be used for recovering the camera response curve function. Since we already have a steady number of images, we can find the correct number of sample pixels. Therefore, we can retype the inequality equation, isolating the number of sample pixels.

$$M > (B_{\max} - B_{\min}) / (N - 1)$$



## 2.3 ESTIMATE THE CAMERA RESPONSE FUNCTION

Since we want to compose a high dynamic range image from differently exposed images of the same scene, we must know what the camera configuration. Generally, pixel values are proportional to the scene radiance, transformed by a nonlinear mapping called the response function. This nonlinearity is induced by different parts of the imaging process.

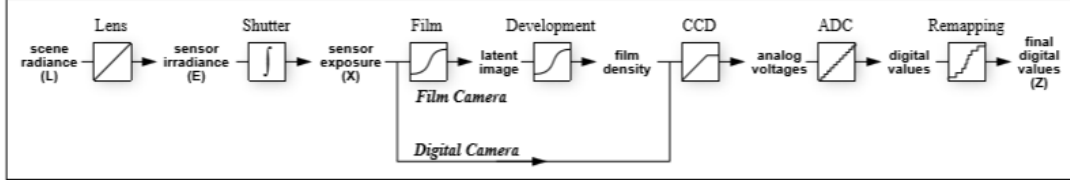


Figure 4. Nonlinear blocks pipeline recovering radiance map

The response of an image, to variations in exposure, is summarized by the characteristic curve (or Hurter - Drifted curve). This is a graph of the optical density  $D$  of the processed film, against the logarithm of the exposure  $X$  to which it has been subjected. The exposure  $X$  is defined as the product of the irradiance  $E$  and exposure time  $\Delta t$ , so that its units are  $\text{Jm}^{-2}$ .

Key on the concept of the characteristic curve, is the assumption that only the product  $E\Delta t$  is important, and that halving  $E$  and doubling  $\Delta t$  will not change the resulting optical density  $D$ . After the image acquisition, we obtain a digital number  $Z$ , which is a nonlinear function of the original exposure  $X$  at pixel level. Let us call this function  $f$ , which is the composition of the characteristic curve of the film as well as all the nonlinearities introduced by the later processing steps. Our first goal will be to recover this function  $f$ . Once we have that, we can compute the exposure  $X$  at each pixel, as  $X = f^{-1}(Z)$ . We make the reasonable assumption that the function  $f$  is monotonically increasing, so its inverse  $f^{-1}$  is well defined. Knowing the exposure  $X$  and the exposure time  $\Delta t$ , the irradiance  $E$  is recovered as  $E = X/\Delta t$ , which is proportional to the radiance  $L$  in the scene.

The input to our algorithm is a number of digitized photographs taken from the same vantage point with different known exposure durations  $\Delta t_j$ . We will assume that the scene is static and that this process is completed quickly enough that lighting changes can be safely ignored. It can then be assumed that the film irradiance values  $E_i$  for each pixel  $i$  are constant. We will denote pixel values by  $Z_{ij}$  where  $i$  is a spatial index over pixels and  $j$  indexes over exposure times  $\Delta t_j$ .

We may now write down the film reciprocity equation as:

$$Z_{ij} = f(E_i \Delta_{ij})$$

Since we assume  $f$  is monotonic and invertible, and we can rewrite it as:

$$f^{-1}(Z_{ij}) = E_i \Delta_{ij}$$

Taking the natural logarithm of both sides:

$$\ln(f^{-1}(Z_{ij})) = \ln(E_i) + \ln(\Delta_{ij})$$

We can notate  $g = \ln(f^{-1})$  and now define the equation as:

$$g(Z_{ij}) = \ln(E_i) + \ln(\Delta_{ij})$$

To make the results more robust, first, we expect  $g$  to be smooth. Debevec adds a constraint to the linear system which penalizes  $g$  according to the magnitude of its second derivative. Since  $g$  is discrete (defined only at integer values from  $g(0)$  to  $g(255)$ ) we can approximate the second derivative with finite differences, e.g.  $g''(x) = (g(x-1) - g(x)) - (g(x) - g(x+1)) = g(x-1) + g(x+1) - 2g(x)$ . We will have one such equation for each integer in the domain of  $g$ , except for  $g(0)$  and  $g(255)$  where the second derivative would be undefined.

We wish to recover the function  $g$  and the irradiances  $E_i$  that best satisfy the set of equations, arising from the above equation in a least-squared error sense. We note that recovering  $g$  only requires recovering the finite number of values that  $g(z)$  can take since the domain of  $Z$ , pixel brightness values, is finite. Letting  $Z_{\min}$  and  $Z_{\max}$  be the least and greatest pixel values (integers),  $N$  be the number of pixel locations and  $P$  be the number of photographs, we formulate the problem as one of finding the min / max values of  $g(Z)$  and the  $N$  values of  $\ln(E_i)$  that minimize the following quadratic objective function.

$$O = \sum_{i=1}^N \sum_{j=1}^P \left[ g(Z_{ij}) - \ln(E_i) - \ln(\Delta_{ij}) \right]^2 + \lambda \sum_{Z=Z_{\min}+1}^{Z=Z_{\max}-1} g''(Z)^2 \Rightarrow$$

$$O = \sum_{i=1}^N \sum_{j=1}^P \left[ g(Z_{ij}) - \ln(E_i) - \ln(\Delta_{ij}) \right]^2 + \lambda \sum_{Z=Z_{\min}+1}^{Z=Z_{\max}-1} [g(Z-1) - 2g(Z) + g(Z+1)]^2$$

Above, the first term is used for minimization of the objective function, and the second term is used as a smoothness term on the sum of squared values of the second derivative of  $g$ , in order to ensure that the function  $g$  is smooth.

### 2.3.1 ESTIMATE THE SMOOTHING FUNCTION

Each exposure only gives us trustworthy information about certain pixels (i.e. the well exposed pixels for that image). For dark pixels the relative contribution of noise is high and for bright pixels the sensor may have been saturated. To make our  $E_i$  estimated values more accurate we need to weight the contribution of each pixel.

An example of a weighting function  $w$  is a triangle function that peaks at  $Z=127.5$  and is zero at  $Z=0$  and  $Z=255$ . This weighting function should be used both when solving  $g$  and when using  $g$  to build the HDR radiance map for all pixels.

To establish a scale factor, we introduce the additional constraint  $g(Z_{\text{mid}}) = 0$ , where  $Z_{\text{mid}} = \frac{1}{2}(Z_{\text{min}} + Z_{\text{max}})$ , simply by adding this as an equation to the linear system. The meaning of this constraint is that a pixel with value midway between  $Z_{\text{min}}$  and  $Z_{\text{max}}$  will be assumed to have unit exposure. The solution can be made to have a much better fit by anticipating the basic shape of the response function. Since  $g(z)$  will typically have a steep slope near  $Z_{\text{min}}$  and  $Z_{\text{max}}$ , we should expect that  $g(z)$  will be less smooth and will fit the data more poorly near these extremes. To recognize this, we can introduce a weighting function  $w(z)$  to emphasize the smoothness and fit in  $g$  terms toward the middle of the curve. A sensible choice of  $w$  is a simple hat function:

$$W(Z) = Z - Z_{\text{min}} \quad \text{for pixel values } \geq Z_{\text{mid}}$$

$$W(Z) = Z_{\text{max}} - Z \quad \text{for pixel values } < Z_{\text{mid}}$$

After that assumption the minimization equation now becomes:

$$O = \sum_{i=1}^N \sum_{j=1}^P \left\{ W(Z_{ij}) \left[ g(Z_{ij}) - \ln(E_i) - \ln(\Delta_{ij}) \right] \right\}^2 + \lambda \sum_{Z=Z_{\text{min}}+1}^{Z=Z_{\text{max}}-1} [W(Z) g'(Z)]^2$$

### 2.3.2 CONSTRUCT THE HIGH DYNAMIC RANGE RADIANCE MAP

Denote the measured value of the pixel  $i$  observed in the image  $j$  by  $Z_{ij}$ , the sensor irradiance at this point is computed by  $E_{ij}$ . The corresponding pixel  $i$  of the high dynamic range map is computed by  $E_i$  and the integration time of image  $j$  by  $\Delta_{tj}$ . Once the inverse response  $g$  is recovered, we can quickly convert pixel values  $Z_{ij}$  to irradiance values  $E_{ij}$ , assuming the integration time  $\Delta_{tj}$  is known. Note that the curve can be used to determine radiance values in any image that acquired by the imaging process associated with  $g$ , not just the images used to recover the response function.

$$\begin{aligned}
f^{-1}(Z_{ij}) &= E_i \Delta_{ij} \Rightarrow \\
E_i &= \frac{f^{-1}(Z_{ij})}{\Delta_{ij}} \Rightarrow \\
\ln(E_i) &= \ln(f^{-1}(Z_{ij})) - \ln(\Delta_{ij}) \Rightarrow \\
g(Z_{ij}) &= \ln(E_i) + \ln(\Delta_{ij}) \Rightarrow \ln(E_i) = g(Z_{ij}) - \ln(\Delta_{ij})
\end{aligned}$$

Having images registered and properly aligned, we can fuse the reconstructed irradiance values  $E_{ij}$  to a high dynamic range map. For robustness, we should use all the available exposures for a particular pixel to compute its radiance. Consequently, we reuse the weighting function to give higher weight to exposures in which the pixel's value is closer to the middle of the response function.

$$\ln(E_i) = \frac{\sum_{j=1}^P W(Z_{ij})(g(Z_{ij}) - \ln(\Delta_{ij}))}{\sum_{j=1}^P W(Z_{ij})}$$

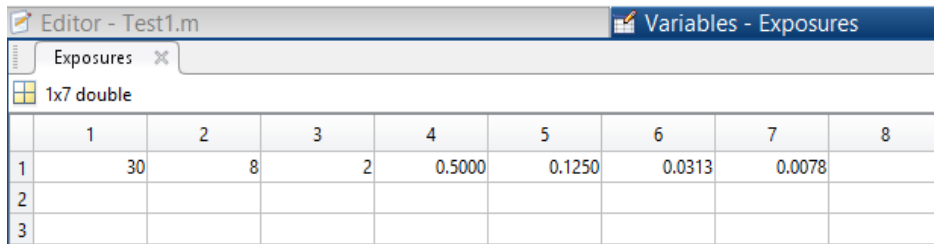
Reconstructing values  $E_i$  using irradiance values, computed from all exposures in the set, helps reducing noise in the final data. Combining the multiple exposures has the effect of reducing noise in the recovered radiance values.

## CHAPTER 3: HDR IMAGING IMPLEMENTATION

In this section we will explain the implementation design, following the background theory that we described above on Matlab.

### 3.1 IMAGE REGISTRATION

The main algorithm implementation is described in *HDR\_main.m* [Listing.1]. First, we must register the data images and extract the exposure of each sample image using *ReadImagesMetaData* [Listing.2]. The exposure of each image is described in the filename. For example, an image named 'church\_30\_1.jpg' means that the exposure time of the image is 30 / 1 = 30 ( $\text{Jm}^{-2}$ ).



The screenshot shows a MATLAB window titled 'Editor - Test1.m' with a 'Variables - Exposures' pane. The variable 'Exposures' is a 1x7 double matrix. The matrix is displayed as follows:

	1	2	3	4	5	6	7	8
1	30	8	2	0.5000	0.1250	0.0313	0.0078	
2								
3								

Figure 5. Exposure matrix

In descent exposure order the corresponding exposure images looks as follows:

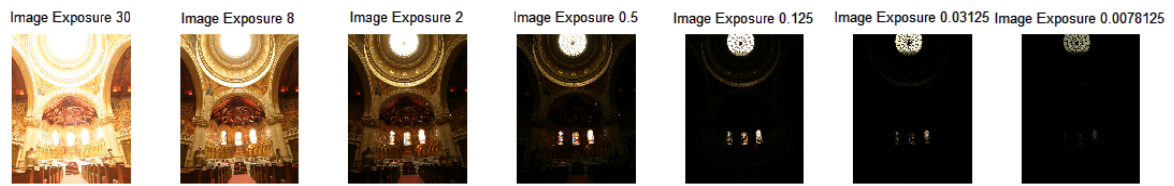


Figure 6. Image data set and their corresponding exposures

### 3.2 SMOOTHING FUNCTION

As we discussed in the background theory, a weighting function is used to avoid saturation and reduce the pixel noise. For pixel smoothing, we use a hat function as the most suitable. This function is described in [Listing.3].

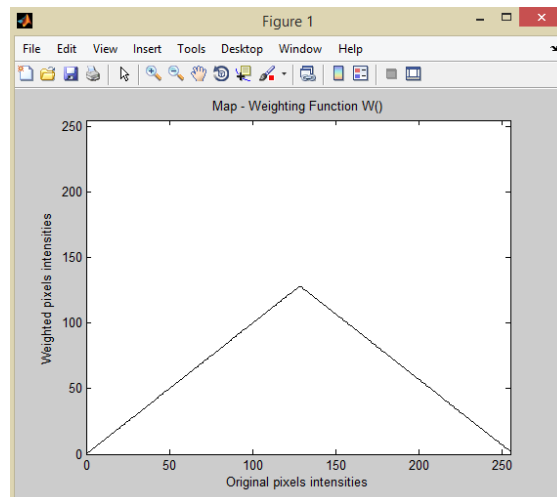


Figure 7. Weighted (hat) function.

### 3.3 LOADING AND SAMPLING THE IMAGES

To find the response curve we must first sample a number of pixels from every channel of each image. We implement the inequality  $M(N-1) > (B_{\min} - B_{\max})$  in *makeImageMatrix* [Listing.4] to calculate the minimum number of sample pixels.

$$\begin{aligned}
NumOfPx(NumOfImg - 1) &> (MaxBright - MinBright) \Rightarrow \\
NumOfPx &> (MaxBright - MinBright) / (NumOfImg - 1) \Rightarrow \\
NumOfPx &> (255 - 0) / (7 - 1) \Rightarrow \\
NumOfPx &> 42.5
\end{aligned}$$

To be sure that we are within the equation limits, we define the number of sample pixels to be four times the minimum (42.5000), that clearly fulfills this requirement. The number of sample pixels calculated to be 170 px. That means that from every exposure we will sample 170 pixels from every RGB channel of every image. To sample the image pixels a random sampling matrix is used, that tell us which pixels of the original image we will sample [Listing.5]. To determine this, a step variable is used, that includes the proportion of the number of sample pixels that fits to all the pixels of every image, ending up to an index array that points to the coordinate of every sample pixel.

### 3.4 FINDING THE RESPONSE CURVE

As the basic equation indicates, the logarithm of the *Exposure* array is calculated and saved in matrix B. We find the coefficients of the response function through *gsolve* [Listing.6] which has as input arguments the sample pixels of the corresponding channel (*zRed*, *zGreen* or *zBlue*), the logarithm of the exposures matrix B, the smoothing term *l* and the weighting hat function weights. As an output, the function returns the coefficients of the linear system that we want to solve, both the logarithmic of the exposures corresponding to pixel value and the logarithmic of the film irradiance at the proportional pixel location.

To find the response function of the camera, we must solve  $g = \ln(f^{-1})$  which maps pixel values (0 - 255) to the log of exposure values  $g(Z_{ij}) = \ln(E_i) + \ln(t_j)$ . Solving *g* is the difficult part (indeed, we only recover *g* up to a scale factor) because we know neither *g* nor *E<sub>i</sub>*. The key observation is that the scene is static, and while we might not know the absolute value of *E<sub>i</sub>* at each pixel *i*, we do know that the value remains constant across the image sequence.

Solving  $g(Z_{ij})$  will be used to map the observed pixels values and shutter times to radiance by the equation  $\ln(E_i) = g(Z_{ij}) - \ln(\Delta_{tj})$ . In the following figure we have randomly chosen 5 pixels and plot their values to all their exposures of the red channel, which is part of the sample matrix of red pixels. Every spot in the scatter plot that has the same color represents the same pixel in a different exposure.

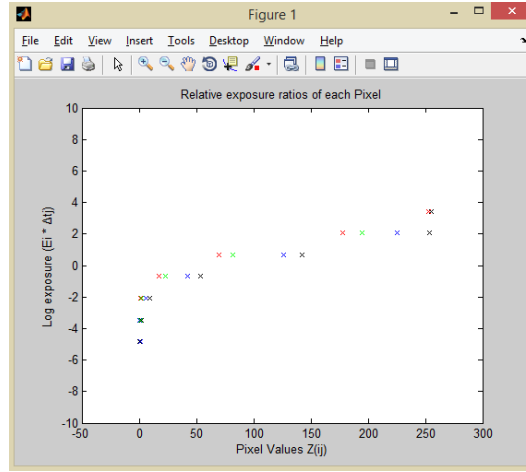


Figure 8. Random pixels in exposure sequence

As we observe, every pixel sequence of all the exposures is creating its own response function. The goal is to find a general response function of every sample pixel for every of the three channels. That constitutes a regression problem and since we hold a matrix of unknowns we can solve that using singular value decomposition, minimizing this way the objective function using the mean square error approach. The function that we want to solve takes the form  $Ax=b$  where  $x$  array is representing the response function and the irradiance values.

So far, the known variables are the pixel values and the exposure values. Decomposing the minimization equation, we know that it consists of two parts. Let us first develop the minimization equation that we are trying to solve.

$$O = \sum_{i=1}^N \sum_{j=1}^P \left\{ W(Z_{ij}) \left[ g(Z_{ij}) - \ln(E_i) - \ln(\Delta t_i) \right] \right\}^2 + \lambda \sum_{Z=z_{\min}+1}^{Z=z_{\max}-1} \{ W(Z) g''(Z) \}^2 \Rightarrow$$

$$O = \sum_{i=1}^N \sum_{j=1}^P \left\{ W(Z_{ij}) \left[ g(Z_{ij}) - \ln(E_i) - \ln(\Delta t_i) \right] \right\}^2 + \lambda \sum_{Z=z_{\min}+1}^{Z=z_{\max}-1} \{ W(Z) g(Z-1) - W(Z) 2g(Z) + W(Z) g(Z+1) \}^2$$

Fitting straight lines and higher-order polynomials to the data, is part of the linear algebra. Therefore, finding the  $(n+1)$  unknown  $A_{mn}$  coefficients is a linear problem.

$$\begin{aligned} A_{11}x_{11} + A_{12}x_{12} + \dots + \dots + A_{1n}x_{1n} &= b(1) \\ A_{21}x_{21} + A_{22}x_{22} + \dots + \dots + A_{2n}x_{2n} &= b(2) \\ &\vdots \\ A_{m1}x_{m1} + A_{m2}x_{m2} + \dots + \dots + A_{mn}x_{mn} &= b(mn) \end{aligned}$$

Each equation (row) corresponds to one experiment measuring the pair,  $A_{mn}$ ,  $b_{(mn)}$ . Remember that all the A's and b's constitute the weighted sample pixels and the corresponding logarithm of exposures. The  $X_{mn}$  values are the unknowns that we want to determine. In matrix notation the system is,  $Ax = b$ . Where A is the  $m$ -row- $n$ -column matrix of measured values, X is the column vector of unknown coefficient values to be determined, and b is the vector of measured values. We know that if the rank of [A, b] matrices are equal, there is a unique solution given by  $x=A \backslash b$ .

The first part of the equation consists the least squares implementation and the second part consists the smoothing process of the values. In more detail the first part takes the following form.

$$\sum_{i=1}^N \sum_{j=1}^P \left\{ W(Z_{ij}) \left[ g(Z_{ij}) - \ln(E_i) - \ln(\Delta_{ij}) \right] \right\}^2$$

Multiplying the common factor of the equation at the end, this equation turns into  $W(Z_{ij})g(Z_{ij}) - W(Z_{ij}) \ln(E_i) = W(Z_{ij})\ln(\Delta_{ij})$ . The unknown part from this equation are  $g(Z_{ij})$  and  $\ln(E_i)$ , and the known part is  $W(Z_{ij})\ln(\Delta_{ij})$ . This is exactly like  $A1 \cdot \overline{x1} + (\sim A1) \cdot \overline{x2} = b$ , with A1 representing the mapped pixels,  $\sim A1$  representing the negative mapped pixels (as it does not contain complex numbers) and b representing the exposure values multiplied with the corresponding weights of the weighting function. We can understand that A matrix has a set of linear equations that has to be solved in every row. As the minimization equations contains two parts, so the A matrix will contain two parts as well.

The first part, must have size equals to  $[N \times P, n+N]$  where N are the number of sample pixels, P the number of exposures and n the maximum intensity value that a pixel can hold (plus one). The reason of that size is that the rows must have all the linear equations of every exposure in every sample pixel and the number of columns is representing the positive and the negative weighted pixels. The first part of the matrix is shown in the figure below.

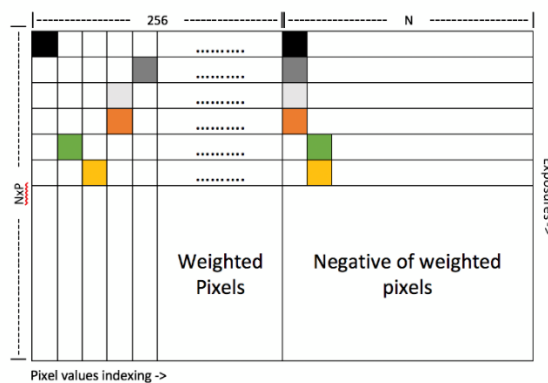


Figure 9. Response function coefficients (Matrix A, first part)



Observing the matrix above, we can see that the color squares represent the weighted pixel values that are stored in rows and represent the exposures (1 to 7 in our case) and the columns represent the initial pixel. For example, if  $(i)$  had a pixel in the first column of the sample pixel matrix that holds the intensity sample value 187, then the mapped value will be 2. This value will be stored to the first row (which means first exposure) in column 187 (which means original intensity). This process represents the term  $W(Z_{ij})$  and the output is stored to the proportional coordinate of the first sub-part of matrix A. Furthermore, we will store the negative value of the output to the first row and column in the second sub-part of matrix A that represents  $(-W(Z_{ij}))$  term. Here, we attach the coefficients of  $g(Z)$  that yields in the first sub-part of matrix A. As we can see these coefficients has already a hat shape because of the weighting function.

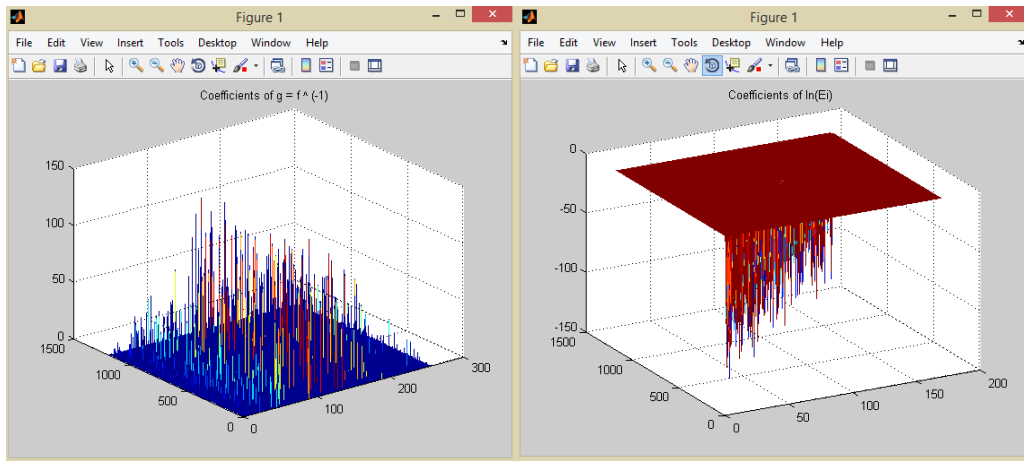


Figure 10. Positive and Negative Weighted coefficients

Before we proceed to the second part of the objective function we must fix the curve by setting the middle value to zero. The second part of the objective function is the smoothing term and takes the form below.

$$\lambda \sum_{Z=z_{\min}+1}^{Z=z_{\max}-1} \{W(Z)g(Z-1) - W(Z)2g(Z) + W(Z)g(Z+1)\}^2$$

This is nothing else but the multiplication of the weighted outputs of all the possible pixel intensity values with each term of the second derivative of the smoothing term placed in the first sub – part of the second part of matrix A. The second sub - part represents null space (all zeros) and as a result this area takes the form in the following attached matrix.

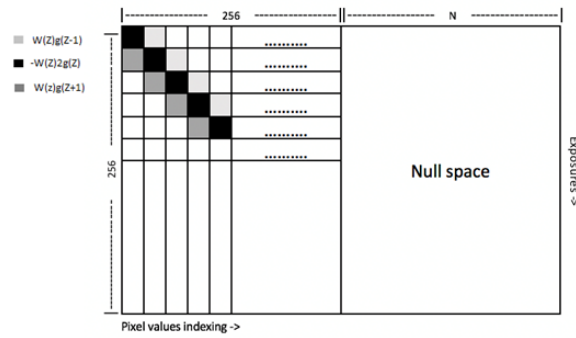


Figure 11. Smoothing coefficients (Matrix A, second part)

Here, a diagonal matrix with non - zero values is created, corresponding to the coefficients that will be used for reaching the  $\ln(E_i)$  values. These coefficients will be used to find the irradiance values.

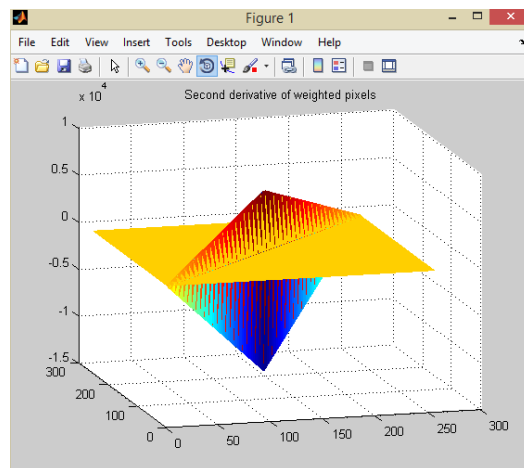


Figure 12. Weighted Smoothing irradiance coefficients

Now that we finished describing matrix A, we will see how array b is developed. As we already know, one of our known values are the measured exposures of the images. That means that b array must be filled with multiplications of the weighted pixel with the logarithm of the exposure of the specific image, creating the term  $W(Z_{ij})\ln(\Delta_{tj})$ .

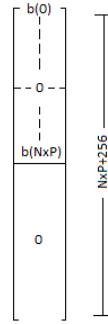


Figure 13. Measured exposures array

Based on equation  $Ax=b$ ,  $b$  matrix will be filled with the corresponding log of exposures in the first  $N \times P$  rows ( $\exp 1\_exp 2 \dots \exp 7$ ,  $\exp 1\_exp 2 \dots \exp 7$  etc.) and zeros to the rest of it. This way, we give to the system the results that must be exported, in order to find the unknown  $x$  ( $1 - N \times P$ ) values that fulfils our linear systems.

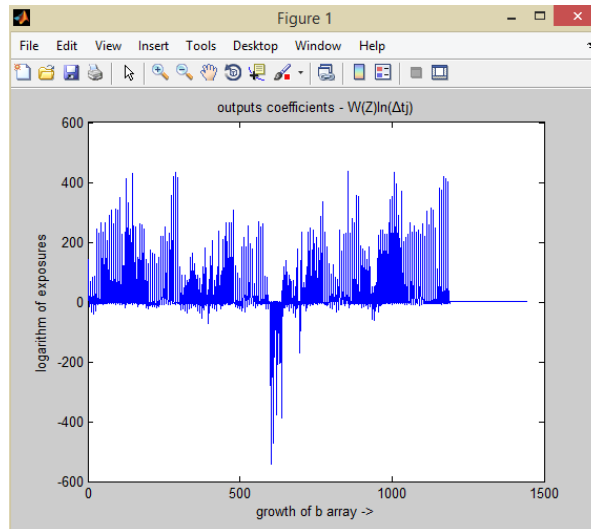


Figure 14. Weighted measured exposures array ( $b$  array)

The size of matrix  $A$  that holds the coefficients and the second derivative of the weighting function has size  $[N \times P + n + 1, n + N]$  and array  $b$  has size  $[N \times P + n + 1, 1]$ . As a result array  $x$ , that represents the response function and the irradiance values must have size  $[N + n, 1]$ . This array, consists of both, the response function coefficients and the irradiance values. The final response function for all channels of the RGB format, is attached below.

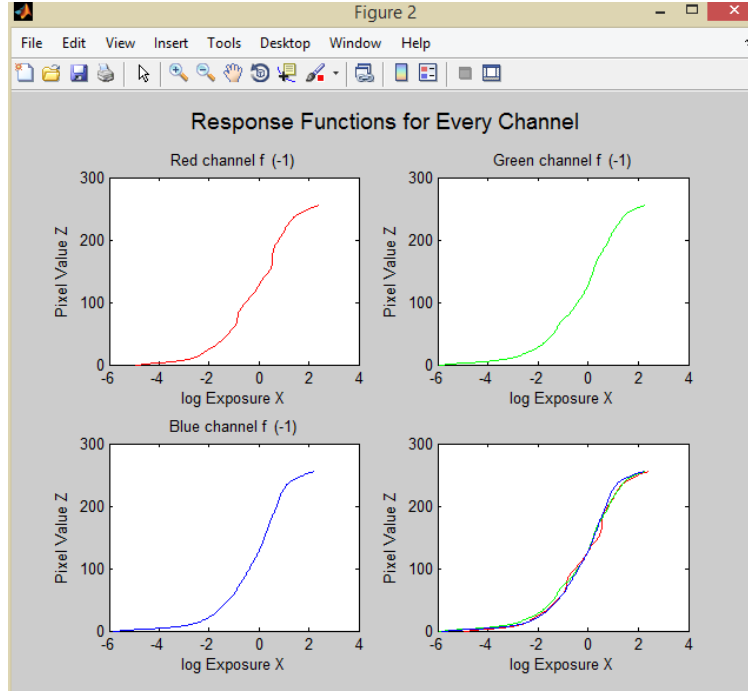


Figure 15. Camera response function in every channel

### 3.5 CONSTRUCTING THE HIGH DYNAMIC RANGE RADIANCE MAP

To generate the high dynamic range radiance map, the original pixels of every image in the database should be mapped to the radiance values through the response function that we found earlier. To achieve that, we feed *hdr* [Listing.7] function with (gRed, gGreen, gBlue) response function values, the matrix *w* that is the weighting hat function as well as the matrix *d<sub>t</sub>* that holds all the exposures repeated by row.

$$\ln(E_i) = \frac{\sum_{j=1}^P W(Z_{ij}) (g(Z_{ij}) - \ln(\Delta_{ij}))}{\sum_{j=1}^P W(Z_{ij})}$$

In this function, we must first weight the original pixels of every image through the weighting function and find  $W(Z_{ij})$ , accumulate the results to matrix *sum* and find  $\Delta(t_i)$ , and finally map the original pixels through the proportional response curve function  $g(Z_{ij})$ . Note that if a pixel is saturated, its influence is ignored in the weighted sum. In the equation above the numerator is represented by the *HDR* matrix and the denominator by the matrix *sum*. To complete the operation, we divide these terms and as a result we end-up with the logarithmic mapped values of the HDR image. After passing the results through an exponential function to eliminate the logarithmic term, we end up with the high dynamic range map.



Figure 16. Irradiance HDR map

The high dynamic range image is usually represented by a floating point map, where each pixel consumes 12 bytes – when represented by a single precision floating point number – or even 24 bytes – in the case of double precision representation. A six-megapixel image would consume as much as 432MB of memory in the double precision format. Additionally, the numeric calculations with floating point numbers are often more time consuming than the fixed point calculations. Unlike the high dynamic range images, the tone-mapped images may be represented by integers without significant loss of precision.

## CHAPTER 4: TONE MAPPING

As we explained, the high dynamic range image needs to be tone mapped first. More formally, the global contrast of the high dynamic range image has to be reduced while the local contrast should be retained as it carries the most valuable information. Image size and processing speed are also some other factors, advancing the use of tone-mapped images over the high dynamic range images. In this implementation we decided to implement Reinhard local and global tone-mapping operator to extract the high dynamic range image, on our low dynamic range display.

The local and global operator algorithm is attached in [Listing.8] and [Listing.9] represented by the function files *reinhardLocal* and *reinhardGlobal* respectively. In addition, we attach the

internal function *makeLuminanceMap* [Listing.10] that is included both in local and global operation. The purpose of this function is to implement a formula extracting a 2D luminance map from a 3D High dynamic range map.



Figure 17. Reinhard local and global tonemapping operator.

### Listing 1. HDR\_main.m matlab language technical details

```
% -----
% Implements a complete hdr and tonemapping cycle
%
% 1.: Computes the camera response curve according to "Recovering High
% Dynamic Range Radiance Maps from Photographs" by P. Debevec.
% You need a wide range of differently exposed pictures from the same scene
% to get good results.
%
% 2.: Recovers a hdr radiance map from the range of ldr pictures
%
% 3.: Tonemaps the resulting hdr radiance map according to "Photographic
% Tone Reproduction for Digital Images" by Reinhard et al.
% Both the simple global and the more complex local tonemapping operator
% are applied to the hdr radiance map.
%
% Some code taken from Paul Debevec's implementation of his SIGGRAPH'97
% paper "Recovering High Dynamic Range Radiance Maps from Photographs"
% -----

clc;clear all;close all;

% Specify the directory that contains your range of differently exposed
% pictures. Needs to have a '/' at the end.
% The images need to have exposure information encoded in the filename,
% i.e. the filename 'window_exp_1_60.jpg' would indicate that this image
% has been exposed for 1/60 second. See readDir.m for details.

dirName = ('../Database_Images/church_exposure_images/');

[filenames, exposures, numExposures] = ReadImagesMetaData(dirName);

tmp = imread(filenames{1});
numPixels = size(tmp,1) * size(tmp,2);
numExposures = size(filenames,2);

% Show the image sequence and the corresponding exposures
% fprintf('Opening test images\n');
% figure('units','normalized','outerposition',[0 0 1 1])
% for i=1:size(filenames,2)
% subplot(1,size(filenames,2),i),imshow(filenames{i});
% title(['Image Exposure ' num2str(exposures(i))])
% end

% define lamda smoothing factor
l = 50;

fprintf('Computing weighting function\n');
% precompute the weighting function value
% for each pixel
weights = [];
for i=1:256
    weights(i) = weight(i,l,256);
end

% load and sample the images
[zRed, zGreen, zBlue, sampleIndices] = makeImageMatrix(filenames,
numPixels);

B = zeros(size(zRed,1)*size(zRed,2), numExposures);

fprintf('Creating exposures matrix B\n')
for i = 1:numExposures
    B(:,i) = log(exposures(i));
end

% solve the system for each color channel
```

```

fprintf('Solving for red channel\n')
[gRed,lERed]=gsolve(zRed, B, l, weights);
fprintf('Solving for green channel\n')
[gGreen,lEGreen]=gsolve(zGreen, B, l, weights);
fprintf('Solving for blue channel\n')
[gBlue,lEBlue]=gsolve(zBlue, B, l, weights);
save('gMatrix.mat','gRed','gGreen','gBlue');

% Plot the response function for every colour channel
% showG;

% compute the hdr radiance map
fprintf('Computing hdr image\n')
hdrMap = hdr(filenamees, gRed, gGreen, gBlue, weights, B);
figure,imshow(hdrMap);title('Irradiance HDR map');

% compute the hdr luminance map from the hdr radiance map. It is needed as
% an input for the Reinhard tonemapping operators.
% fprintf('Computing luminance map\n');
% luminance = 0.2125 * hdrMap(:, :,1) + 0.7154 * hdrMap(:, :,2) + 0.0721 *
hdrMap(:, :,3);

% apply Reinhard local tonemapping operator to the hdr radiance map
fprintf('Tonemapping - Reinhard local operator\n');
saturation = 0.6;
eps = 0.05;
phi = 8;
[ldrLocal, luminanceLocal, v, v1Final, sm ] = reinhardLocal(hdrMap,
saturation, eps, phi);

% apply Reinhard global tonemapping operator to the hdr radiance map
fprintf('Tonemapping - Reinhard global operator\n');

% specify resulting brightness of the tonapped image. See reinhardGlobal.m
% for details
a = 0.72;

% specify saturation of the resulting tonemapped image. See reinhardGlobal.m
% for details
saturation = 0.6;

[ldrGlobal, luminanceGlobal ] = reinhardGlobal( hdrMap, a, saturation );

figure,imshow(ldrGlobal);
title('Reinhard global operator');

figure,imshow(ldrLocal);
title('Reinhard local operator');

fprintf('Finished!\n');

```



## Listing 2. ReadImagesMetaData.m matlab language technical details

```
% Creates a list of all pictures and their exposure values in a certain
directory.
%
% Note that the directory must only contain images which are named according
to the
% naming conventions, otherwise this function fails.
%
% Filename naming conventions:
% The filename of a picture must contain two numbers specifying the
% exposure time of that picture. The first number specifies the nominator,
% the second one the denominator. E.g. "image_1_15.jpg" specifies that this
% image has been taken with an exposure of 1/15 second.
function [filenames, exposures, numExposures] = readDir(dirName)

    filelist = dir(dirName);
    for i = 3:size(filelist,1)
        filenames{i-2} = strcat(dirName,filelist(i).name);
    end

    i = 1;
    for filename = filenames
        filename = filename{1};
        [s f] = regexp(filename, '(\d+)');
        nominator = filename(s(1):f(1));
        denominator = filename(s(2):f(2));
        exposure = str2num(nominator) / str2num(denominator);
        exposures(i) = exposure;
        i = i + 1;
    end

    % sort ascending by exposure
    [exposures indices] = sort(exposures);
    filenames = filenames(indices);

    % then inverse to get descending sort order
    exposures = exposures(end:-1:1);
    filenames = filenames(end:-1:1);

    numExposures = size(filenames,2);
```

Listing 3. weight.m\_matlab language technical details

```
function w = weight(z, zmin, zmax)
    if z <= 0.5 * (zmin + zmax)
        w = ((z - zmin) + 1); % never let the weights be zero because that
        would influence the equation system!!!
    else
        w = ((zmax - z) + 1);
    end
```

#### Listing 4. makeImageMatrix.m\_matlab language technical details

```
% Takes relevant samples from the images for use in gsolve.m
%
%
function [ zRed, zGreen, zBlue, sampleIndices ] = makeImageMatrix(
filenames, numPixels )

    % determine the number of differently exposed images
    numExposures = size(filenames,2);

    % Create the vector of sample indices
    % We need N(P-1) > (Zmax - Zmin)
    % Assuming the maximum (Zmax - Zmin) = 255,
    % N = (255 * 2) / (P-1) clearly fulfills this requirement
    numSamples = ceil(255*2 / (numExposures - 1)) * 2;

    % create a random sampling matrix, telling us which
    % pixels of the original image we want to sample
    % using ceil fits the indices into the range [1,numPixels+1],
    % i.e. exactly the range of indices of zInput
    step = numPixels / numSamples;
    sampleIndices = floor((1:step:numPixels));
    sampleIndices = sampleIndices';

    % allocate resulting matrices
    zRed = zeros(numSamples, numExposures);
    zGreen = zeros(numSamples, numExposures);
    zBlue = zeros(numSamples, numExposures);

    for i=1:numExposures

        % read the nth image
        fprintf('Reading image number %i...', i);
        image = imread(filenames{i});

        fprintf('sampling.\n');
        % sample the image for each color channel
        [zRedTemp, zGreenTemp, zBlueTemp] = sample(image, sampleIndices);

        % build the resulting, small image consisting
        % of samples of the original image
        zRed(:,i) = zRedTemp;
        zGreen(:,i) = zGreenTemp;
        zBlue(:,i) = zBlueTemp;
    end
end
```

Listing 5. sample.m matlab language technical details

```
function [ red, green, blue ] = sample( image, sampleIndices )
    % Takes relevant samples of the input image

    redChannel = image(:,:,1);
    red = redChannel(sampleIndices);

    greenChannel = image(:,:,2);
    green = greenChannel(sampleIndices);

    blueChannel = image(:,:,3);
    blue = blueChannel(sampleIndices);
end
```

### Listing 6. gsolve.m matlab language technical details

```
% gsolve.m - Solve for imaging system response function
%
% Code taken from Paul Debevec's SIGGRAPH'97 paper "Recovering High Dynamic
Range
Radiance Maps from Photographs"
%
%
% Given a set of pixel values observed for several pixels in several
% images with different exposure times, this function returns the
% imaging system's response function g as well as the log film irradiance
% values for the observed pixels.
%
% Assumes:
%
% Zmin = 0
% Zmax = 255
%
% Arguments:
%
% Z(i,j) is the pixel values of pixel location number i in image j
% B(j) is the log delta t, or log shutter speed, for image j
% l is lamdba, the constant that determines the amount of smoothness
% w(z) is the weighting function value for pixel value z
%
% Returns:
%
% g(z) is the log exposure corresponding to pixel value z
% lE(i) is the log film irradiance at pixel location i
%
function [g,lE]=gsolve(Z,B,l,w)
    n = 256;
    A = zeros(size(Z,1)*size(Z,2)+n+1,n+size(Z,1));
    b = zeros(size(A,1),1);

    %% Include the data-fitting equations
    k = 1;
    for i=1:size(Z,1)
        for j=1:size(Z,2)
            wij = w(Z(i,j)+1);
            A(k,Z(i,j)+1) = wij;
            A(k,n+i) = -wij;
            b(k,1) = wij * B(i,j);
            k=k+1;
        end
    end

    %% Fix the curve by setting its middle value to 0
    A(k,129) = 1;
    k=k+1;

    %% Include the smoothness equations
    for i=1:n-2
        A(k,i)=l*w(i+1); A(k,i+1)=-2*l*w(i+1); A(k,i+2)=l*w(i+1);
        k=k+1;
    end

    %% Solve the system using SVD
    x = A\b;
    g = x(1:n);
    lE = x(n+1:size(x,1));
```

### Listing 7. hdr.m matlab language technical details

```
% Generates a hdr radiance map from a set of pictures
%
% parameters:
% filenames: a list of filenames containing the differently exposed
% pictures used to make a hdr from
% gRed: camera response function for the red color channel
% gGreen: camera response function for the green color channel
% gBlue: camera response function for the blue color channel
% dt : log of exposures matrix

function [ hdr ] = hdr( filenames, gRed, gGreen, gBlue, w, dt )

    numExposures = size(filenames,2);

    % read the first image to get the width and height information
    image = imread(filenames{1});

    % pre-allocate resulting hdr image
    hdr = zeros(size(image));
    sum = zeros(size(image));

    for i=1:numExposures

        fprintf('Adding picture %i of %i \n', i, numExposures);

        image = double(imread(filenames{i}));

        wij = w(image + 1);
        sum = sum + wij;

        m(:, :, 1) = (gRed(image(:, :, 1) + 1) - dt(1,i));
        m(:, :, 2) = (gGreen(image(:, :, 2) + 1) - dt(1,i));
        m(:, :, 3) = (gBlue(image(:, :, 3) + 1) - dt(1,i));

        % If a pixel is saturated, its information and
        % that gathered from all prior pictures with longer exposure times
        % is unreliable. Thus
        % we ignore its influence on the weighted sum (influence of the
        % same pixel from prior pics with longer exposure time ignored as
        % well)

        saturatedPixels = ones(size(image));

        saturatedPixelsRed = find(image(:, :, 1) == 255);
        saturatedPixelsGreen = find(image(:, :, 2) == 255);
        saturatedPixelsBlue = find(image(:, :, 3) == 255);

        % Mark the saturated pixels from a certain channel in *all three*
        % channels
        dim = size(image,1) * size(image,2);

        saturatedPixels(saturatedPixelsRed) = 0;
        saturatedPixels(saturatedPixelsRed + dim) = 0;
        saturatedPixels(saturatedPixelsRed + 2*dim) = 0;

        saturatedPixels(saturatedPixelsGreen) = 0;
        saturatedPixels(saturatedPixelsGreen + dim) = 0;
        saturatedPixels(saturatedPixelsGreen + 2*dim) = 0;

        saturatedPixels(saturatedPixelsBlue) = 0;
        saturatedPixels(saturatedPixelsBlue + dim) = 0;
        saturatedPixels(saturatedPixelsBlue + 2*dim) = 0;

        % add the weighted sum of the current pic to the resulting hdr
        % radiance map
        hdr = hdr + (wij .* m);
```

```

        % remove saturated pixels from the radiance map and the sum
(saturated pixels
    % are zero in the saturatedPixels matrix, all others are one)
    hdr = hdr .* saturatedPixels;
    sum = sum .* saturatedPixels;
end

    % For those pixels that even in the picture with the smallest exposure
time still are
    % saturated we approximate the radiance only from that picture instead
    % of taking the weighted sum
    saturatedPixelIndices = find(hdr == 0);

    % Don't multiply with the weights since they are zero for saturated
    % pixels. m contains the logRadiance value from the last pic, that one
    % with the longest exposure time.
    hdr(saturatedPixelIndices) = m(saturatedPixelIndices);

    % Fix the sum for those pixels to avoid division by zero
    sum(saturatedPixelIndices) = 1;

    % normalize
    hdr = hdr ./ sum;
    hdr = exp(hdr);

```

### Listing 8. reinhardLocal.m\_matlab language technical details

```
% Implements the Reinhard local tonemapping operator
%
% parameters:
% hdr: high dynamic range radiance map, a matrix of size rows * columns * 3
% luminance map: the corresponding lumiance map of the hdr image

function [ ldrPic, luminanceCompressed, v, v1Final, sm ] = reinhardLocal(
hdr, saturation, eps, phi )

    fprintf('Computing luminance map\n');
    luminanceMap = makeLuminanceMap(hdr);

    alpha = 1 / (2*sqrt(2));
    key = 0.18;

    v1 = zeros(size(luminanceMap,1), size(luminanceMap,2), 8);
    v = zeros(size(luminanceMap,1), size(luminanceMap,2), 8);

    % compute nine gaussian filtered version of the hdr luminance map, such
    % that we can compute eight differences. Each image gets filtered by a
    % standard gaussian filter, each time with sigma 1.6 times higher than
    % the sigma of the predecessor.
    for scale=1:(8+1)

        %s = exp(sigma0 + ((scale) / range) * (sigma1 - sigma0)) * 8
        s = 1.6 ^ (scale-1);

        sigma = alpha * s;

        % discretize gaussian filter to a fixed size kernel.
        % a radius of 2*sigma should keep the error low enough...
        kernelRadius = ceil(2*sigma);
        kernelSize = 2*kernelRadius+1;

        gaussKernelHorizontal = fspecial('gaussian', [kernelSize 1], sigma);
        v1(:, :, scale) = conv2(luminanceMap, gaussKernelHorizontal, 'same');
        gaussKernelVertical = fspecial('gaussian', [1 kernelSize], sigma);
        v1(:, :, scale) = conv2(v1(:, :, scale), gaussKernelVertical, 'same');
    end

    for i = 1:8
        v(:, :, i) = abs((v1(:, :, i)) - v1(:, :, i+1)) ./ ((2^phi)*key / (s^2) +
v1(:, :, i));
    end

    sm = zeros(size(v,1), size(v,2));

    for i=1:size(v,1)
        for j=1:size(v,2)
            for scale=1:size(v,3)

                % choose the biggest possible neighbourhood where
                % is still smaller than a certain epsilon.
                % Note that we need to choose that neighbourhood which is
                % as big as possible but all smaller neighbourhoods also
                % fulfill v(i,j,scale) < eps !!!
                if v(i,j,scale) > eps

                    % if we already have a high contrast change in the
```



```

        % first scale we can only use that one
        if (scale == 1)
            sm(i,j) = 1;
        end

        % if we have a contrast change bigger than epsilon, we
        % know that in scale scale-1 the contrast change was
        % smaller than epsilon and use that one
        if (scale > 1)
            sm(i,j) = scale-1;
        end

        break;
    end
end
end

% all areas in the pic that have very small variations and therefore in
% any scale no contrast change > epsilon will not have been found in
% the loop above.
% We manually need to assign them the biggest possible scale.
idx = find(sm == 0);
sm(idx) = 8;

v1Final = zeros(size(v,1), size(v,2));

% build the local luminance map with luminance values taken
% from the neighbourhoods with appropriate scale
for x=1:size(v1,1)
    for y=1:size(v1,2)
        v1Final(x,y) = v1(x,y,sm(x,y));
    end
end

% TODO: try local scaling with a/key as in the global operator.
% But compute key for each chosen neighbourhood!
numPixels = size(hdr,1) * size(hdr,2);
delta = 0.0001;
key = exp((1/numPixels)*(sum(sum(log(v1Final + delta)))));
scaledLuminance = v1Final * (a/key);
luminanceCompressed = (luminanceMap* (a/key)) ./ (1 + scaledLuminance);

% Do the actual tonemapping
luminanceCompressed = luminanceMap ./ (1 + v1Final);

ldrPic = zeros(size(hdr));

% re-apply color according to Fattals paper "Gradient Domain High
Dynamic
% Range Compression"

for i=1:3

    % (hdr(:,:,i) ./ luminance) MUST be between 0 and 1!!!!
    % ...but hdr often contains bigger values than luminance!!!??
    % so the resulting ldr pic needs to be clamped
    ldrPic(:,:,i) = ((hdr(:,:,i) ./ luminanceMap) .^ saturation) .*
luminanceCompressed;
end

% clamp ldrPic to 1
indices = find(ldrPic > 1);
ldrPic(indices) = 1;

```

### Listing 9. reinhardGlobal.m\_matlab language technical details

```
% Implements the Reinhard global tonemapping operator.
%
% parameters:
% hdr: a rows * cols * 3 matrix representing a hdr radiance map
%
% a: defines the desired brightness level of the resulting tonemapped
% picture. Lower values generate darker pictures, higher values brighter
% pictures. Use values like 0.18, 0.36 or 0.72 for bright pictures, 0.09,
% 0.045, ... for darker pictures.
%
% saturation: a value between 0 and 1 defining the desired saturation of
% the resulting tonemapped image
%
function [ ldrPic, ldrLuminanceMap ] = reinhardGlobal( hdr, a, saturation)

fprintf('Computing luminance map\n');
luminanceMap = makeLuminanceMap(hdr);

numPixels = size(hdr,1) * size(hdr,2);

% small delta to avoid taking log(0) when encountering black pixels in the
% luminance map
delta = 0.0001;

% compute the key of the image, a measure of the
% average logarithmic luminance, i.e. the subjective brightness of the image
% a human
% would approximateley perceive
key = exp((1/numPixels)*(sum(sum(log(luminanceMap + delta)))));

% scale to desired brightness level as defined by the user
scaledLuminance = luminanceMap * (a/key);

% all values are now mapped to the range [0,1]
ldrLuminanceMap = scaledLuminance ./ (scaledLuminance + 1);

ldrPic = zeros(size(hdr));

% re-apply color according to Fattals paper "Gradient Domain High Dynamic
% Range Compression"
for i=1:3
    % (hdr(:,:,i) ./ luminance) MUST be between 0 and 1!!!!
    % ...but hdr often contains bigger values than luminance!!!???
    % so the resulting ldr pic needs to be clamped
    ldrPic(:,:,i) = ((hdr(:,:,i) ./ luminanceMap) .^ saturation) .*
    ldrLuminanceMap;
end

% clamp ldrPic to 1
indices = find(ldrPic > 1);
ldrPic(indices) = 1;
```

Listing 10. makeLuminanceMap.m matlab language technical details

```
function [ luminanceMap ] = makeLuminanceMap( image )
%Creates a luminance map from an image
%
% The input image is expected to be a 3d matrix of size rows*columns*3

luminanceMap = 0.2125 * image(:,:,1) + 0.7154 * image(:,:,2) + 0.0721 *
image(:,:,3);
```