## Abstract

The thesis work is done in the form of a Windows application that allows for automatic capture and generation of HDR images. The basic method is to take a sequence of standard images, also called LDR (Low Dynamic Range) images, and then construct an HDR (High Dynamic Range) image. This is achieved by processing the sequence of images and accumulating the brightness values in the separate LDR images into an HDR image that contains the true radiance values. By installing this application, and connecting a Canon system camera via the supplied USB cable, the user is able to acquire and create HDR images in the field or on location with minimal effort. The software controls the camera and lets the user choose settings via an easy to use interface. When all settings are applied, the user can then take a sequence of images. When satisfied with the images in the sequence, all that is needed is to create the HDR image with the click of a single button. The computer then does all the necessary computation of the HDR image automatically and lets the user choose output destination and file format of the HDR image.

## Table of contents

# 1 Background

There has been a substantial amount of research in the area of High Dynamic Range Imaging. There are also many applications available for creating HDR images. But there has never been an application that can both automatically control a standard digital camera to acquire a suitable sequence of standard images, and that can create an HDR image with multiple preview and saving options. That is the underlying reason for this thesis work, which consists of an application that aims to both fill this gap in the software market, and this report which describes the work that has been done to create it.

## 1.1 High Dynamic Range Imaging

Digital photography has made the photographer's life easier when it comes to capturing a scene. But capturing the true light in a scene with great contrast is difficult, and in some cases, impossible. A good example would be to try and take a picture of a window from inside a room, with the sun shining brightly outside. In this scenario, the photographer has to make a choice. Either he or she has to adjust the exposure of the image show the interior of the room. In this case the window will be overexposed and seen as just a white rectangle. Or the photographer can adjust the image settings so that the world outside the room is clearly visible – but in this case the interior of the room will be black and impossible to make out.

All this is due to the limitations of the sensor in the common digital cameras of today. Although the contrast ratio of an outside scene can be up to 1:100.000 or even 1:1.000.000, the only measuring instrument available that comes close to see the details in this vast range of contrast is our eyes. A camera sensor has normally about 256 levels of contrast in each color channel, red, green and blue - which are not enough (Reinhard, Ward, Pattanaik, & Debevec, 2006). The solution to this problem is to create a new type of image that contains more radiance information than before, and this is where High Dynamic Range imaging comes in. The basic idea of an HDR image is to take a set of images, instead of just one, of the exact same scene.  Each image in the set has a different exposure setting, from light (where details in the dark areas can be seen), to dark (where details in bright areas of the picture can be seen and distinguished). This set of images ranging from light to dark can, using the right methods, then is combined into a single image with far greater contrast ratio than a normal digital photograph. The brightness levels in the pixels no longer range from 0-256 for each channel, but as described earlier, they are significantly more. Since much more of the radiance of the scene is stored in such an image, it allows the photographer to adjust parameters like exposure time in the picture - after it was taken. It also means that applications that need true light information in an image can benefit from HDR images.  For instance, HDR images can be used to light virtual scenes in 3D production software, using what is called Image Based Lighting. In order to make the synthetic objects in the scene correctly to the image based lighting, the image used has to have high dynamic range (Watt, 2000).

There are some problems with HDR images, however. As of yet, there is no way to view an HDR image in its true form directly on a screen, i.e. to show the whole dynamic range. The contrast ratio of a normal video monitor is very narrow, but some research has been done to create displays that can show more of the contrast ratio than previous technologies. A company called BrightSide Inc. demonstrated in 2005 an HDR monitor aimed at the consumer market.  They were acquired in 2007 by Dolby, who now seems to be aiming at bringing better visual solutions to the market using HDR technology, as well as their famous sound enhancing technology.

For those who can't afford an HDR display, there is other ways to use and to enjoy HDR images. One solution is to tone map the image, by simply compressing the available range in the HDR image down to that of what an ordinary display device can handle, such as a computer monitor. This can create an overly colorful and unnatural (although very artistic) image depending of the dynamic range in the HDR image. Another way of displaying a HDR image on an ordinary display device is to choose the exposure level. If this is done with a slider in the software, the viewer can drag the handle to view the image with different exposure levels. In this way it is possible to simulate the situation that the photographer was in when the image was acquired, in that he or she had to choose the exposure setting at the scene if a standard low dynamic range image was taken.

# 2 Method

The application communicates with any Canon SLR (digital system) camera. The user can set a range of parameters on the camera. When the parameters are set to the user's satisfaction, a sequence of low dynamic range pictures is taken. These are stored on the computer for manipulation, along with a description file containing information about the images taken. This allows the user to load a previously taken sequence, and to create HDR images without a camera.

The application communicates with the SDK created by Canon (Canon Europa N.V., 2008) which allows for programming their range of digital cameras. In turn, the SDK then communicates with the driver for the camera connected to the computer. As this application is developed for Microsoft Windows, the SDK is connected to the Windows driver for the connected camera. It is an advantage that the application does not communicate with the camera directly, as it allows for efficient error handling. This means for example, that if the camera were disconnected during operation, the application gets a message about this via the driver and it can then display an error message without the application crashing.

## 2.1 Overview

The user interface of the software is divided into sections. These correspond to the different steps required to acquire a sequence of images, and to convert these into an HDR image. The reason for dividing the user interface was to increase usability and to improve workflow. Also, even though the application is compact, it gives an overview as well as detailed information about the work that is done for the moment. The three different sections are camera control, capturing or loading a sequence, and HDR creation.

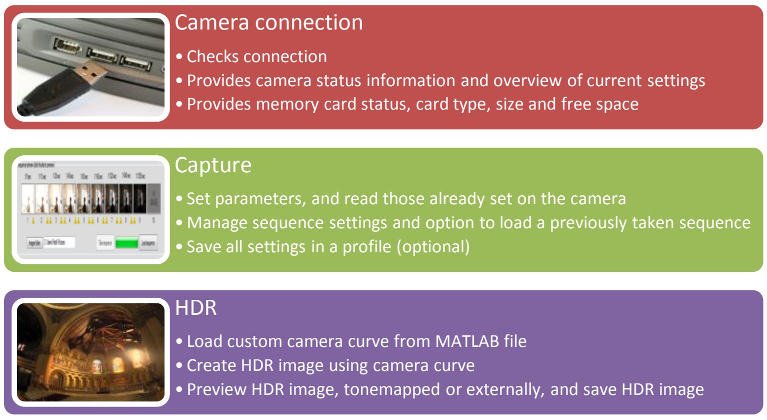
Camera connection
• Checks connection
• Provides camera status information and overview of current settings
• Provides memory card status, card type, size and free space
Capture
• Set parameters, and read those already set on the camera
• Manage sequence settings and option to load a previously taken sequence
• Save all settings in a profile (optional)
HDR
• Load custom camera curve from MATLAB file
• Create HDR image using camera curve
• Preview HDR image, tonemapped or externally, and save HDR image

Figure 2-1: Basic components of the application

The application was created in the Microsoft Visual Studio 2008 environment. It uses the Canon EDSDK framework to communicate with the camera, and the FreeImage framework for image and pixel manipulations (SourceForge, 2009).

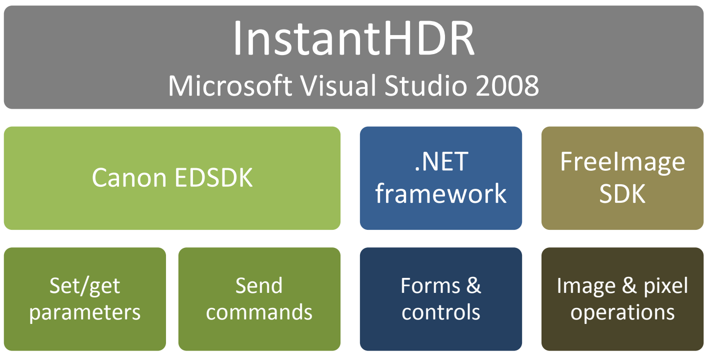

Figure 2-2: The building blocks

## 2.2 HDR creation pipeline

Creating an HDR image from standard, low dynamic range images, involves a series of actions needed. First, and most obviously, the images need to be captured. Normally, the sequence of images is taken with the camera, while being careful not to touch it between exposures. If the camera moves during capture of the sequence, blurring and/or ghosting might appear. Blurring is spatial inconsistency between images, when shooting a sequence of a tree with moving branches and leaves, for instance. Ghosting occurs when there is residual image information in the image. When there is an object moving through the scene during the shooting of a sequence, there may be faint but noticeable copies of the object along its path.

These problems might also arise if the camera moves during the shoot. The best way to avoid this earlier was to mount the camera on a tripod, and use a remote control. This would allow for a steady and good sequence. Using this software, no handling of the camera is needed. Once it is connected to the computer, and the application is started up, there is no more need to touch the camera. This is apart from repositioning it for new and interesting HDR motifs, of course.

Another part of the first section is the general camera settings. ISO values, aperture and resolution settings are found here, and these should be set prior to making adjustments to the sequence settings, according to the preferred settings of the particular camera model used in shooting the current scene.

When it is time to start adjusting the sequence settings, the one major thing to think about is which exposures to choose when shooting a sequence. Changing exposure times between shots is the most common method, and is implemented in the software. Here, we consider the middle shot to be the one the sequence originates from. So when shooting a sequence of 5 images, the median shutter speed corresponds to the image in the middle of the sequence once it is shot.

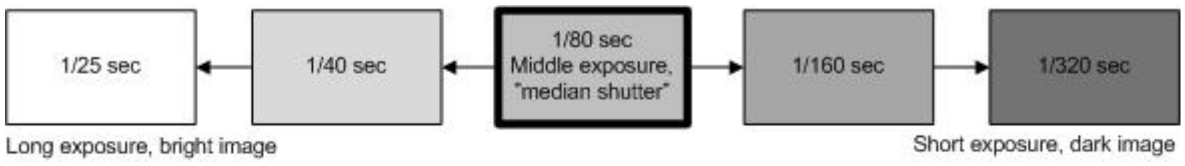

Figure 2-3: Shutter time selection

After choosing a median shutter speed, the next thing to consider is the spacing between the shots. Basically, in the software there is a list of all the shutter speeds that a particular camera can handle. So, choosing a series of shutter speeds is simply a matter of looking them up in the shutter speed table. This means that a sequence of 5 images consists of the median shot, two shots below and two shots above. As mentioned, there is a spacing function available, which allows for the shutter speed above and under the median, can be one or more spaces apart in the lookup table. Below are some sample values that can be selected from the list of shutter times in the EDSDK framework. The actual selection is quite larger, of course. The downside is that it is not certain that some cameras can take the value it is asked to set from this list. In those cases, the camera sets automatically the nearest value. While it is good that the software doesn't crash or the camera locks up while trying to handle the wrong values, the user can't really know which value is actually set when this happens. This applies also to the other properties in the camera that can be read or set, such as the aperture value or the ISO value.

| Value | Shutter speed |
|---|---|
| •0x10 | •30" |
| •0x5D | •1/25 sec |
| •0x65 | •1/50 sec |
| •0x6D | •1/100 sec |

Figure 2-4: Value to description conversion table (example)

When these values are returned by the function in CameraInterface, they need to be translated to a format that can be used when the software need to display the selected value. This is just an example of the different types of code that needs to be synchronized, especially since the first example (the table) is written in C++ and the translation is written in C#. As mentioned before, all these function calls are parsed through the wrapper function, to translate the C++ commands into C# and back.

One feature of this software is that all these settings can be saved in a file for later use. When all the settings are made, it is time to shoot the sequence. When the user clicks the "Acquire sequence" button, commands are sent to the camera to shoot the number of images required for the sequence, altering the shutter speed for each exposure according to the sequence settings set in the software. Once finished, the images in the sequence are presented as thumbnails.

On each image, a histogram function is done on each captured image. By analyzing the histograms for each image, the software can alert the user that there may be over- and/or underexposed pixels in the images in the sequence. Important here is to make sure that there are pixels that are ok in at least one image in the sequence. If an image is shot with long exposure time, chances are that the image contains overexposed pixels. If the scene is very bright, there is even a chance that some pixels are overexposed in all images in the sequence. In this case it is preferred that the image sequence is reshot, with values selected so that more valid pixels are acquired. This can be achieved by either shifting the shutter times so that the shortest exposure is even darker, to try to avoid the overexposed pixels, or to alter the scope of the shutter times in the sequence so that the overexposed pixels can be avoided.

The captured images can be individually inspected by clicking on their thumbnails. Some extra information about each image is shown under the preview windows at the bottom center of the application. When the user is satisfied, the sequence can be saved for future manipulation.

The HDR image created with this application can be saved in two different formats, OpenEXR and PFM. The OpenEXR format was developed by Industrial Light & Magic for use in computer imaging applications. Since such movies as Harry Potter and the Sorcerer's Stone and others that ILM worked on at the time, OpenEXR has become their main image file format (Lucas Digital LLC, 2009).

The PPM format is very basic in its construction, by just having a header that contains information about what type of image it is and the dimensions of the image, and then the pixel data after that. The PFM image format (SourceForge, 2003) is a lot like PPM (Portable Pixel Map), but uses floating point numbers with no maximum value to achieve a HDR format. In PPM the image data might be compressed in some way, but for HDR storage, uncompressed floating point pixel values are stored like in PFM (Portable Float Map).

## 2.3 Practicals

Before using the application, the user should make sure that the driver for the camera is installed properly on the computer used to run the application. On most Windows systems, the camera is automatically detected and the driver is installed. In the event that the driver is not automatically installed, the user can use the driver on the CD supplied with their camera. Canon states that the OS standard driver should be used for cameras that support PTP (I3A, 2005) if the operating system also supports PTP (Picture Transfer Protocol). If the camera, or the operating system, does not support PTP, the driver provided by Canon should be used. Also make sure that the batteries are fully charged before taking the computer and the camera out on the field.

## 2.4 Implementation

The application consists of a variety of functions, but they belong to a specific set of categories. These can be summarized as follows:

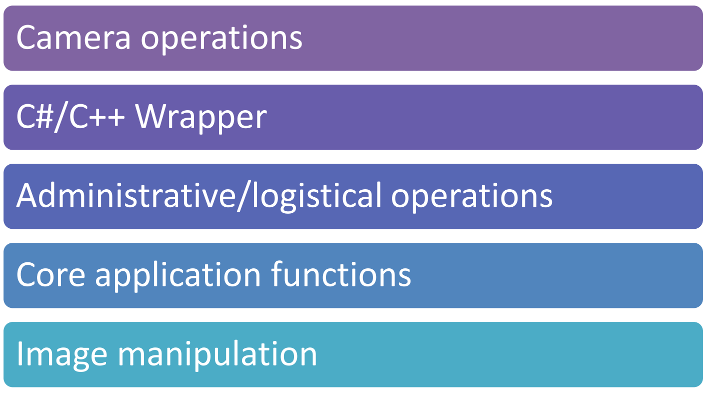
Camera operations
C#/C++ Wrapper
Administrative/logistical operations
Core application functions
Image manipulation

Figure 2-5: Program function categories

Some of these different operations are addressed within the same actual functions, but the behavior and the way that they are implemented may differ.

### 2.4.1 Camera operations

The control and manipulation of the connected camera are managed in a source file called CameraInterface. This file is the interface that connects to the camera, and does so by utilizing the functions available in the Canon SDK toolkit. By calling on the functions in the SDK, parameters in the camera can be set or retrieved. Commands can also be sent to the camera, such as commands to take pictures.

The Canon SDK requires code written in C++ so CameraInterface is written in this language as well. But the application itself and the GUI is written in C# for efficiency and convenience, as well as superior handling of windows forms. This means that when parameters and commands are sent to the camera, they have to be translated via a wrapper.

### 2.4.2 C++/C# Wrapper

As CameraInterface can't receive or send commands and parameters directly to or from the application GUI, they have to pass a specially written wrapper file for translation. It's relatively simple; it takes the commands and parameters that the application needs and returns the reply from the target source. This file, CanonHDRLibNET, is compiled as a dynamic library link file or DLL for short. This makes it easy to include in the installation folder of the target system, but by not including it into the main application source file, it can be recompiled for other operating systems and thus much of the applications functionality can be ported. As the name suggests, it uses the .NET framework for integration on Microsoft Windows systems.

### 2.4.3 Administrative and logistical operations

Pictures in a sequence taken by the camera are downloaded when the sequence is complete. This is an operation which involves sending commands to the camera as described above, but the handling of acquired sequences involve more than just downloading the images onto the computer. Which each sequence, a file containing information about the sequence is also created. This logistical function is there to make management of acquired sequences easier and to add functionality to the application. By creating a file with data about the sequence, old sequences can be read into the application at a later time without the need of a camera, as described earlier. The result is an XML-file containing information about the camera user to acquire the sequence, the name and the shutter speeds of each image in the sequence. As this file is always created for each sequence, and each sequence is in its own folder, each sequence can be transferred to another computer or backed up for storage. But they can always be read by the software in any computer. The added bonus of this approach is that the sequence can be acquired in the field by small light-weight netbooks or laptops, and then later be transferred to a more powerful desktop computer for HDR creation.

Other functions involve file operations, such as the ability to read the camera curve files that correspond to the different cameras used with this software. The application cannot create the camera curve needed for HDR creation by itself, but has the ability to read camera curve files created by an application called (HDRShop) in the MATLAB .m format. Also, once an HDR image has been created, it can be saved to the computers storage facilities. The format of the saved image is OpenEXR, an HDR format created by Industrial Light & Magic (Lucas Digital LLC, 2009).

### 2.4.4 Core application functions

Spread throughout the application is labels, drop down lists, picture boxes, text fields and file dialogs. The code for managing these controls and to control their behavior, is for the most part facilitated by the built-in functionality of Microsoft Visual Studio using the .NET framework. This makes the development of windows functions easy. To create a file dialog with .NET all that is needed is to call the built-in functionality for it, while the amount of code needed is symbolic. Creating a file dialog by hand, managing files and data streams etc. is both difficult and time consuming.

### 2.4.5 Image manipulation

The control of the camera and sending and receiving commands for it is done in the CameraInterface file described above, with the help of the wrapper CanonHDRLibNET. This was because of the inconsistency between the code written in C++ for the camera and the code written in C# written for the application GUI and all its helper functions.

For managing the images and manipulating them, yet another framework was needed. For all image and pixel operations, the FreeImage framework is used. The code for interacting with this open source library is also written in C#, making it susceptible for compilation together with the GUI application code. Once the source files for FreeImage is included in the project, the FreeImage functionality can easily be accessed. The source file in the project dealing with image manipulation is called ImageOperations.

Once an image is read into ImageOperations, it is then stored into a proprietary bitmap format supplied by FreeImage. The bitmap formats provided by FreeImage offers great flexibility, and can consist of a number of bit depths, such as 1-, 4-, 8-, 16-, 24-, 32-bit per pixel for standard bitmaps. In this case we store the images in the acquired or loaded sequence in standard bitmaps, and save the created HDR image in an RGB Float bitmap. The bit depths are 32-, 96-, 128-bit per pixel for FLOAT, RGBF and RGBAF image types, providing great flexibility and high accuracy. In this application, RGBF is used for storing the images and manipulating the pixels.

As the software currently only supports JPEG images acquired by the camera, FreeImage was a suitable choice since it supports a number of graphics image formats such as PNG, BMP, JPEG, TIFF among others. Implementing FreeImage was quite straight forward. According to the development team behind FreeImage, it is fast, multithreading safe and compatible with all 32-bit versions of Windows. It also supports multiple platforms such as Linux and Macintosh OSX. This is another reason why FreeImage was chosen, since the Canon SDK also supports Mac OSX. This should minimize the effort needed to port the application to Mac OSX and possibly Linux (since Mac OSX and Linux both are built using similar technology).

## 2.5 Implementation of HDR techniques

As this application employs the technique of converting a sequence of low dynamic range images into an HDR image, the required functions have to be implemented into code. The HDR creation algorithm by combining LDR images used by this software uses the techniques established by Debevec et.al. (Debevec & Malik, 1997). Also, the methods of combining LDR images into HDR images have been repeatedly implemented in a variety of HDR applications over the last few years, such as HDRShop, Photogenics HDR, and EasyHDR to name but a few (Panotools, 2008). In recent versions, Adobe Photoshop has also gained support for rudimentary HDR image creation.

The implemented functionality in this software is mainly concerned with gathering images, and the fundamental calculations and operations required to assemble an HDR image. The application also has the ability to generate an image with compressed dynamic range for display on screen, by providing a selection of tone mapping algorithms.

### 2.5.1 HDR algorithm

Even if the resulting HDR image contains intensity data from all the images in the image sequence, it is not just a matter of adding the intensities of each pixel position together into a high dynamic range intensity map. First, the images in the sequence have to be calibrated according to the response function of the cameras image sensor. If the intensity data in the final image is to be regarded as fairly accurate, it is important to consider how the camera reacts to light in a scene, rather than just accept the images in the camera as fact. Basically all cameras are sensitive to noise in the dark regions of a picture, i.e. at lower intensity values, as this is a limitation common to most CCD sensors.

The sensors also usually have problems with high intensity values, and there is a risk that the pixels in a high intensity area are overexposed. This means that the intensities in those pixels might have the maximum value, and this in turn might mean that the value really was higher but the camera just couldn’t acquire the correct value. The same could be true to underexposed pixel values, where the intensity was so low that the camera just set the value to 0. To summarize, the color values near the extremes are not to be trusted (Unger & Löw, 2009).

An exposure $X_{ij}$ is defined as the irradiance $E_i$ at the sensor, times the exposure time $t_j$, where $i$ denotes the particular pixel location and $j$ denotes the exposure, as follows:

$$X_{ij} = E_i \Delta t_j$$

Although the relative brightness in the acquired sequence of images usually is not a true measurement of the radiance in the scene that is being captured, it is possible to approximate the radiance. This is done by mapping the radiance values seen by the cameras sensor to pixel colors $Y_{ij}$ via a camera curve, called $f(X)$. The camera curve, which is assumed to be monotonic as it needs to be inverted later on in the calculations, is unique for each camera model and translates what the camera sees to what pixel values it outputs.

The resulting image contains pixel colors as per the following expression:

$$Y_{ij} = f(X_{ij}) = f(E_i \Delta t_j)$$

As the camera curve is assumed to be invertible, we therefore have a $f^{-1}$ function that can give us the calibrated values of the captured images. This in turn yields the irradiance $E$:

$$f^{-1}(Y_{ij}) = E_i \Delta t_j$$

As mentioned before, small pixel values (or dark pixels) are sensitive to noise. Also, cameras are generally prone to saturate at a lower value than the maximum, which means that an image with saturated pixels might actually have been saturated long before the theoretical limit of 255 has been reached. In order to avoid these values we have to use a weighting function that filters out such undesirable values from the calculations. In the mapping of irradiance values $E_i$ to A/D converted pixel values $Y_{ij}$, some values might appear also are considered unreliable, so the weighting function is applied to these calculations as well. The weighting function will produce 0 for the values not desired, and 1 for the reliable ones.

The cameras sensor interprets the irradiance values in each pixel $(i)$ as per the following expression:

$$E_{iweighted} = g(Y_{ij}) \frac{f^{-1}(Y_{ij})}{\Delta t_j}$$

This weighting function $g(Y_{ij})$ would produce artifacts in the resulting image in the form of banding, if it was used without modification, as it would basically behave like a thresholding function. To avoid this, we have to introduce ramps at the start and the end of the weighting function, creating a somewhat smoother curve.

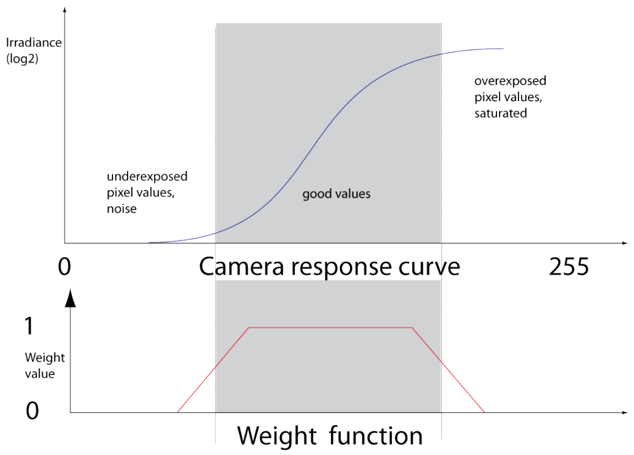

Figure 2-6: Camera response curve and the weight function

To calculate the final HDR image, we combine the weighted mean values from the exposures in the acquired sequence. The pixels $Z_i$ in the high dynamic range image are calculated by averaging the weighted pixel values $Y_{ij}$ in the $N$ low dynamic range images in the acquired sequence with the exposure time $\Delta t_j$ in each image.

The final expression to obtain the HDR image is given as follows:

$$Z_i = \frac{\sum_{j=0}^{N}\left(g\left(Y_{ij}\right)\frac{f^{-1}\left(Y_{ij}\right)}{\Delta t_j}\right)}{\sum_{j=0}^{N} g\left(Y_{ij}\right)}$$

### 2.5.2 HDR creation process overview

The HDR calculations are done in the ImageOperations file. Basically, the above functions and algorithms are implemented in C#. Since the language in MATLAB is related to the C# language, it was convenient to first pseudo-program the solutions in MATLAB first for experimentation and reference, check the results, and then implement them in C#.

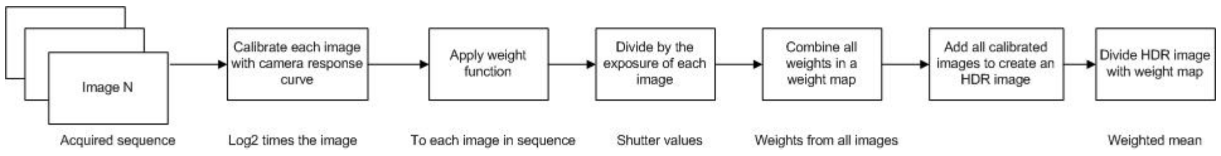
Image N
Acquired sequence
Calibrate each image with camera response curve
Log2 times the image
Apply weight function
To each image in sequence
Divide by the exposure of each image
Shutter values
Combine all weights in a weight map
Weights from all images
Add all calibrated images to create an HDR image
Divide HDR image with weight map
Weighted mean

Figure 2-7: HDR process flowchart

The actual HDR creation takes place in a function called ProcessImages, as described in the figure above. First, the images in the sequence are collected using the upper part of the application's interface utilizing the Canon SDK framework. Then the functions in ProcessImages are called:

1. Read the current calibration curve into a vector with ReadCalibrationCurve()
2. Create weight matrix with the function CreateWeightMatrix(). This matrix is empty initially but will later contain all the combined weights calculated from the pixel values in the images
3. Check the weight matrix for anomalies and unwanted values with CheckWeightMatrix()
4. The algorithm processes the images in the sequence in shortest-to-longest order
5. Set the current exposure
6. Calibrate incoming image using CalibrateHDRImage()
7. Calculate the HDR image with CalculateHDRImage() by adding together the calibrated and weighted images in the sequence.
8. Show progress meter in the user interface. This needs to increment instead of decrement (reverse to the loop).
9. After all images have been processed, adjust the HDR image by dividing it with the weight matrix by calling AdjustHDRImage().
10. Tone map the resulting HDR bitmap for display on the monitor using one of the avaliable tone mapping algorithms.

Steps 5 to 8 are repeated for each image in the sequence. When the HDR image is complete, it can be viewed directly in the application's preview window by using one of the available tone mapping algorithms. It can also be previewed in an external HDR viewing application that is bundled with the software. If the HDR image looks good, it can be saved to disk via a file dialog. If adjustments are needed, a new sequence can be taken immediately and a new HDR image can be calculated.

### 2.5.3 Tone mapping

HDR images cannot be viewed directly on a computer screen or any other standard viewing device, as they only can cope with images that are considered to have a low dynamic range. Some devices might have 10 bits, 12 bits or even slightly more bits of information per pixel and per color channel. But nothing that can yield the range that we refer to when it comes to high dynamic range imaging. So, for the standard monitors and projectors available today, we have to compress the contrast range to 8-bits per color channel. This is called tone mapping.

This application uses a couple of tone mapping operators, the main one being developed by Sumanta Pattanaik presented in the paper A Simple Spatial Tone Mapping Operator for High Dynamic Range Images. This algorithm is set up as follows: Displaypixel = HDRpixel / (HDRpixel + c*HDRaverage) where "c" is a "contrast" variable, giving some control over the total contrast of the HDR tone mapped image. A very simple second tone mapping generator is also available, which just compresses the whole dynamic range of all the pixel values down to the range 0-255 which Is viewable on today's display devices:  Displaypixel = HDRpixel / (HDRpixel + 1).

The tone mapping algorithms used in this application is of the type global tone mapping filters. They are fast and simple, but works on the whole image and does not take into account effects that might occur in neighborhoods between and around specific pixels.

Local tone mapping algorithms are more sophisticated, as they preserve contrast between neighboring regions rather than absolute values. The algorithms are based on contrast or gradient domains, but these calculations are more complicated and take more time to compute, which is why they are not implemented in this application yet.

## 2.6 Discussion

The software was created in several iterations. The first version of the software was simple but competent enough to communicate with the camera. The Canon SDK was implemented in a simple command-line based application. The software could send commands to the camera, such as to take a picture, set aperture or exposure time, and order it to download an image to the computer. It could also read settings from the camera, such as camera model, which lens was fitted, memory card information and so on.

The logical next step was to convert this simple command-line application into a Windows application. The first version used MFC for Windows creation, although it was soon apparent that this approach was cumbersome and difficult. The MFC framework is old and complicated to use, and as such was abandoned.

It is easier and more efficient to create windows applications using C# and the .NET framework. As mentioned, the downside of this approach is that the Canon SDK requires code in C++. The solution was to create two separate applications, one in C# for Windows forms creation, and one for camera communication written in C++. Obviously, the trick was to make these two different applications work together. This is accomplished by using a layer of software that effectively translates commands to and from the two applications.

## 2.7 Segmentation and portability

The solution to this problem was to write the software as two separate programs. The first part contains the code for creating the user interface, and is written in C#. The second part contains the code for communicating with the camera and is written in C++. The first part is the main application containing the user interface, and the second is in the current iteration compiled as a DLL for easy access by the main application.

This approach has another benefit as well. Since all the code for communicating with the camera was written in a separate program, using C++, it can be used for exporting these functions to other platforms and operating systems. All that is needed is for the developer to write a new user interface for each operating system, and connect the program to a recompiled version of the camera communication software. As such, the software can be ported to Linux or OSX, for instance.

# 3 Results

The aim of this project was to create a complete but simple solution for photographers interested in High Dynamic Range imagery. The idea was that the user only needed a camera and a laptop to acquire HDR images, with the ability to do all the steps needed in the field with minimum effort. This is achieved by this easy-to-use application. The software achieves this by allowing the user to set all the necessary parameters in the camera, specify the settings for the sequence, capture the sequence onto the computer, and then create the HDR image. The exported HDR image could then be used for further manipulation in other software.

The user interface was easy to construct using Visual Studio and the .NET framework. Implementing the Canon SDK in C++ was not too complicated, and using FreeImage for image and pixel manipulation was agreeable once the basics were learned. Making the C++ camera communication code work together with the user interface code written in C#, however, was a challenge. Through the creation of the C++ / C# wrapper, it was possible to solve in the end.

The HDR images that can be created with this software are useful in 3D and animation packages where they can be used for Image Based Lighting of a synthetic scene. Another area where HDR images are useful is image editing, where manipulating images with great accuracy and correct color and radiance information is important. Other applications of HDR images include satellite imagery, scientific visualization and medical visualization.

The software is light-weight and comes with its own installer, and does not require a high-end computer to run. The sequences are stored on the computer’s hard drive in separate folders. After a while when there are many sequences stored they tend to take up a sizeable amount of space, especially if the sequences are taken using anything lager than the S (Small) quality setting. Added to this is amount of space that the created HDR images take up. If the user is not careful, and plays with settings and shoots many sequences, it is possible to fill up a hard drive within a short period of time. Also, creating HDR images from very large images consumes a lot of computing power, relative to the amount of pixels that are analyzed and calculated.

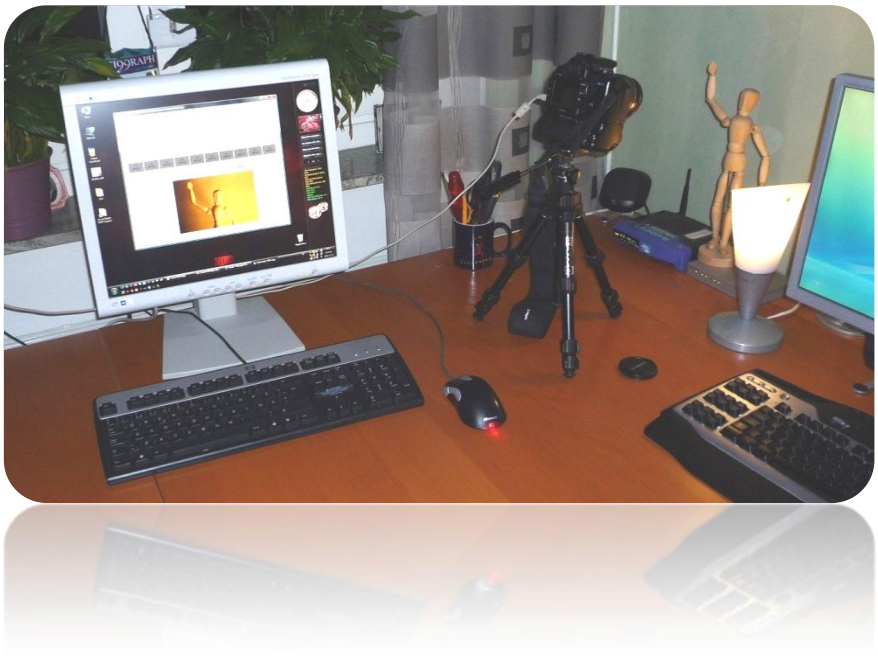

Figure 3-1: HDR image creation on the desktop

# 4 Future work

The application could benefit from supporting a wider range of file formats. Also it would benefit from a function where the camera response curve is calculated automatically from the camera or sequence. In its current form, the user have to use an external software solution do obtain a camera curve for their camera. HDRShop has this capability, and creates the curve vector in the form of a MATLAB m-file. This software utilizes this, and can read m-files.

There is a bug regarding which exposure values the camera allows, resulting in that some values are not quite correct. This is due to a mismatch in the table containing the exposure values for a specific camera model. A solution to this would be the ability to read these values directly from the camera instead of having them in a table, hard coded into the program. This also applies to aperture values.

The application basically does what is needed for it to work properly. But there is room for many improvements, such as support for more image formats. A better and more interactive HDR viewer would be a nice addition. Faster and more efficient handling of the captured images is also welcome in future releases.

The program could be extended by adding a function for creating a dome. By converting the spherical mapping of an image taken with an extreme-angled lens (fisheye) into a latitude-longitude mapping, it would be useful for image based lighting in 3D applications.

Another obvious extension, which would greatly improve the usability of the application, would be to include support for cameras other than just those manufactured by Canon.

# 5 References

Canon Europa N.V. (2008). *EDSDK 2.4 API Programming Reference.* Retrieved 06 10, 2009, from Digital Imaging Developer Programme: http://www.didp.canon-europa.com/

Debevec, P. E., & Malik, J. (1997). *Recovering High Dynamic Range Radiance Maps from Photographs.* Berkeley: University of California at Berkeley, ACM Siggraph.

HDRShop. (n.d.). Retrieved 06 10, 2009, from HDRShop: http://www.hdrshop.com

I3A. (2005). *PTP*. Retrieved 06 10, 2009, from International Imaging Indusrty Association: http://www.i3a.org/tehnologies/image-formats/ptp/

Lucas Digital LLC. (2009). *www.openexr.com*. Retrieved 06 10, 2009, from OpenEXR: http://www.openexr.com/about.html

Panotools. (2008). *HDR Software overview*. Retrieved 06 10, 2009, from Panotools.org Wiki: http://wiki.panotools.org/HDR_Software_overview

Reinhard, E., Ward, G., Pattanaik, S., & Debevec, P. (2006). *High Dynamic Range Imaging - Acquisition, display and image-based lighting.* San Francisco: Morgan Kaufmann Publishers.

SourceForge. (2003). *PPM Format Specification*. Retrieved 06 10, 2009, from SourceForge.net: http://netpbm.sourceforge.net/doc/ppm.html

SourceForge. (2009). *The FreeImage Project*. Retrieved 06 10, 2009, from SourceForge.net: http://freeimage.sourceforge.net/

Unger, J., & Löw, J. (2009). *High Dynamic Range Images TNM083 - Image Based Rendering.* Norrköping: Joakim.Löw@itn.liu.se.

Watt, A. (2000). *3D Computer Graphics.* Harlow: Pearson Education Limited.

## 5.1 Figures

# 6 Appendix 1: Code example

The following example code shows the actual HDR creation algorithm, implemented in C#. There are many more lines of code in the software, but to include more examples than this would not be sensible. This particular example, ProcessImages(), is central to the project.

```
internal static void ProcessImages(System.Drawing.Image[]
ImageArray, int nSize, string filename, float[] expotimes, string
calibrationCurveFile, out System.Drawing.Image ResultImage)
{
    if (nSize == 0)
    {
        ResultImage = new System.Drawing.Bitmap(0, 0);
        return;
    }

    // Make a bitmap array
    System.Drawing.Bitmap[] BitmapArray = new
    System.Drawing.Bitmap[nSize];
    for (int i = 0; i < nSize; i++)
    {
        BitmapArray[i] = new System.Drawing.Bitmap(ImageArray[i]);
    }

    // Go through the pixels...
    Height = BitmapArray[0].Height;
    Width = BitmapArray[0].Width;

    // Create a result bitmap (FreeImageBitmap) based on the size
    of the first bitmap (they should all be the same)
    System.Drawing.Bitmap ResultBitmap = new
    System.Drawing.Bitmap(Width, Height);
    fib_result = FreeImage.AllocateT( FREE_IMAGE_TYPE.FIT_RGBF,
    Width, Height, 96, 0, 0, 0 );
    if ( fib_result == null )
    {
        // No bitmap
        ResultImage = new System.Drawing.Bitmap(0, 0);
        return;
    }

    // Create a bitmap holding the calibrated image (reused for
    each image)
    FIBITMAP fib_calibrated =
    FreeImage.AllocateT(FREE_IMAGE_TYPE.FIT_RGBF, Width, Height,
    96, 0, 0, 0);
    if (fib_calibrated == null)
    {
        // No bitmap
        ResultImage = new System.Drawing.Bitmap(0, 0);
        return;
    }

    // Create weight matrix and clear it
    // Read the current calibration curve
    CreateWeightMatrix(Height, Width);
    ReadCalibrationCurve(calibrationCurveFile);
```

```
        // **** Calculate the HDR over all images ****
        // Go top-down, from shortest shutter time to longest
        // Set the current exponentiation rate (below)
        // Calibrate incoming image (below)
        // Render the HDR image (below)
        // Show nice progress meter (below) This needs to increment
        // instead of decrement (reverse to i).
        // After all images have been processed, adjust HDR by dividing
        // with weight matrix (below)
        // And then render the resulting bitmap (for display) (below)

        for (int i = nSize - 1; i >= 0; i--)
        {
            currentexp = expotimes[i];
            CalibrateHDRImage(BitmapArray[i], Height, Width,
            fib_calibrated);
            CalculateHDRImage(BitmapArray[i], Height, Width,
            fib_result, fib_calibrated);
            HDRProgressEvent(nSize-i-1);
        }

        // Check the weight matrix for anomalies (undesired values)
        CheckWeightMatrix(Height, Width, nSize);

        // Apply the weighting function
        AdjustHDRImage(Height, Width, nSize, ResultBitmap, fib_result);

        // Save the HDR image using FreeImage’s save routine
        if (!FreeImage.Save(FREE_IMAGE_FORMAT.FIF_EXR, fib_result,
        filename, FREE_IMAGE_SAVE_FLAGS.DEFAULT))
        {
            // Save failed
            MessageBox.Show(filename, "File saving failed." );
        }

        System.Drawing.Image resImage = ResultBitmap as
        System.Drawing.Image;

        // Tone map the finished HDR iamge
        GenerateTone2(ref resImage, 0.25F);
        ResultImage = resImage;
    }
```

# 7 Appendix 2: Tutorial

A prerequisite for this tutorial is that the camera is set to manual [M] mode before connecting it to the computer or launching the application. Otherwise, the computer can't control some of the parameters in the camera.

Connect your camera to the computer using the USB cable. Set your basic parameters, such as ISO and aperture values. Now choose the settings for your sequence. Set the median exposure time for your sequence. The median exposure time corresponds to the exposure time of the image in the middle of the sequence, hence the images before it will have longer exposure times and be brighter, and the images after it will have shorter exposure times and will be darker.

If you want, you can add information of your sequence in the text field on the left. And now it's time to shoot the sequence! Press the ”Take sequence” button and wait for the camera to finish taking the images, and for the application to download the images to the hard drive. If you are satisfied with the result, you can save all the settings for this shooting session for later use. Use the ”Save settings” button for this.

Now you can look closer on the different images in your sequence. If something is wrong, you can adjust your settings for the sequence, recapture it and review the results again. If you see an improvement, you can resave the settings for later use. Remember that all sequences are stored on the hard drive separately, and are accessible for later use.

Next, load the appropriate camera response curve. This is needed for a correct calculation of the HDR image. When you are satisfied with the sequence of images, you can create the HDR image. Simply click the ”Create HDR image” button to do this.

When the calculations of the HDR image is finished, a tone mapped version of it is displayed in the center of the application. The default setting is the “Pattanaik” tone mapping algorithm, with the parameter set at 0.25. You can now adjust the parameter, or choose another tone mapping algorithm if you like.

It is also possible to preview the HDR image in an external viewer application, EXRdisplay by Industrial Light & Magic. Simply click the button for this, and your image will appear in a new window. EXRdisplay has the ability to adjust the exposure of the image in (almost) real time, showing you the full dynamic range in the HDR image.

When satisfied with the resulting HDR image, it can be saved to the computer in Open EXR floating point format. Future versions of the software will let you save the image in PFM format too.