HDR Image Construction from Multi-exposed Stereo LDR Images

by

Ning Sun

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

January 2012

Abstract

The vast majority of cameras in the market nowadays can only capture a limited dynamic range of a scene. To generate high dynamic range (HDR) images, most existing methods use multiple images obtained from a single low dynamic range (LDR) camera at consecutive instances. These methods can obtain good quality HDR images for still or slow motion scenes but not for scenes with fast motion.

In this thesis, we propose the use of two LDR cameras which have different exposures. To generate an HDR image, the two differently exposed LDR images of the same scene are used. The two LDR images should be captured at the same instance, so as to deal with scenes with fast motion. The most challenging step in this approach is to obtain accurate estimates of the disparity maps of the scenes. This will allow us to correctly align the pixels from the two differently exposed pictures when forming the HDR images.

Very few state-of-the-art stereo matching algorithms can deal with the problem of obtaining accurate estimates of the disparity map from two differently exposed images. This is because the input LDR images that are used to construct HDR images have large radiometric changes. In addition, the two input LDR images usually have saturations in different areas. To obtain accurate disparity maps, we present a novel algorithm that obtains an initial estimate of the disparity map. Then a refinement step is used to minimize the edge effect and interpolates the values in the saturated regions.

Compared to other state-of-the-art methods, our algorithm has a simpler set up with only two standard commercial LDR cameras. The offline processing of the LDR images has a simpler cost function, especially the cost function we use in the refinement step of the disparity map. This reduces the computational complexity and thus the processing time of the LDR images to form the HDR image. Moreover, the disparity map computed by our algorithm can tolerate greater radiometric changes and saturations. Therefore, the HDR images constructed by our algorithm are smoother and have fewer defects than those constructed by other methods.

Preface

A version of chapter 3 has been published. [Ning Sun], M. Hassan, and R. Ward. (2010) HDR Image Construction from Multi-exposed Stereo LDR Images. 2010 17th IEEE International Conference on Image Processing. 2793 – 2796. I proposed the algorithm and conducted the experiments. I also wrote the manuscript. M. Hassan and R. Ward reviewed the manuscript and R. Ward provided supervision for this project.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ANCC: Adaptive normalized cross correlation

CCD: Charge coupled device

CMOS: Complementary metal–oxide–semiconductor

HDR: High dynamic range

LoG: Laplacian of Gaussian

LDR: Low dynamic range

NCC: Normalized cross correlation.

SAD: Sum of absolute difference

Acknowledgements

I owe my gratitude to my supervisor, Dr. Rabab Ward, who has always provided me good advice and academic guidance where needed. She was also generous on making herself available over the weekends because I started to work full time after attending three semesters at UBC.

I thank Mansour Hassan for sharing his knowledge and ideas. The help and encouragement I received from him helped me to overcome the difficulties I encountered during my research.

I also thank NSERC for funding my research in my first year and the UBC scholarship board for funding my research in my second year.

Tanaya, Di, Anshual, Qiang, Jack, Xinyi and all the people from the Image Processing Lab. I wish to say thank you very much for the peer support and good company during my Master study. The friendly and positive environment kept me motivated in the past few years.

Finally, I would like to thank my family for their support during the hard times of my thesis completion period and for their investment in my education.

Ning Sun

The University of British Columnbia
April 2011

Chapter 1 Introduction

1.1 Background

We now live in a world with overwhelming information. One of the most common medium we use to communicate and store the information in is that of images. It is very important that the images can accurately represent the details in the original scene in order to preserve all the information and avoid miscommunication. The first digital camera, a camera that uses a charged coupled device (CCD) imager and digitizes the captured scene and store the digital info on a standard cassette, was invented in 1975. Since then, the imaging technology has leaped into a new era by significantly extending the dynamic range in capturing real world luminance.

However, the majority of cameras in the market nowadays are still incapable of capturing the entire dynamic range of a scene. For a scene, the dynamic range refers to the ratio between the brightest and darkest parts of the scene. The dynamic range of real-world scene can be quite high. Ratios of 100,000:1 is quite common in the natural world. Human visual system can process a simultaneous dynamic range of 50,000:1 or more and can adapt to a much larger range. The images captured by the cameras in the market now can only have dynamic ranges between 300:1 to 1,000:1. Therefore, these images are considered as low dynamic range (LDR) images.

High dynamic range (HDR) imaging provides the capacity to represent the wider dynamic range of the human visual system (HVS) in digital form. It assigns pixels with floating point values. The use of floating point values gives HDR images several advantages over LDR images. In an LDR image, areas that are too dark are clipped to black and areas that are too bright are clipped to white. In an HDR image, pixel values are normalized between 0.0 and 1.0 with 0.0 representing black and 1.0 representing white. Dark areas and bright areas in HDR images are assigned different values close to 0.0 and 1.0. Therefore, HDR images can preserve details in a scene with large dynamic range. The use of the floating point also gives HDR images perceptual cues which increase the apparent brightness. Another advantage of

HDR images is that they preserve optical phenomena such as reflections, refractions and transparent materials such as glass. In LDR images, the pixels representing all the bright light sources in a scene such as the sun are assigned to have the maximum possible integer value. However, the reflected light should have value less than the light source. In HDR images, the reflected light is assigned with values close but less than 1.0, while the bright source can assume values that equal to 1.0. This allows reflections off surfaces to maintain realistic brightness for bright light source. Therefore, HDR images are able to better represent scenes perceived by human eyes.

In the past, there has been a strong effort to develop high-dynamic-range (HDR) display and camera hardware, as well as the supporting processing algorithms. The idea of using several differently exposed images to capture details in the extreme range of luminance was pioneered by Gustave Le Gray in 1850s. He tried to render images showing both the sky and the sea. The use of high dynamic range imaging (HDRI) in computer graphics was first introduced by Greg Ward in 1985 with his open-source Radiance rendering and lighting simulation software which created the first file format to retain a high-dynamic-range image. However, due to limitations imposed by the computer processing power and displaying technologies in the past, HDR images started to gain wider usage only in the past decade. In 2005, Photoshop CS2 introduced the “Merge to HDR” function which combines LDR images of a still scene taken under different exposures at consecutive time instances to form an HDR image. This year, iphone 4 also introduced HDR photography functionality in iOS version 4.1.

Besides a wider usage of HDR images, there is also an increasing demand for HDR videos Modern movies are often filmed with cameras featuring a higher dynamic range, and legacy movies can be upgraded even if manual intervention would be required for some frames. In addition, special effects, especially those in which real and synthetic footage are seamlessly mixed, require both HDR shooting and rendering. HDR video is also required in all applications in which capturing temporal aspects of changes in the scene is required with high accuracy. This is important in particular in monitoring of some industrial processes such as welding, predictive driver assistance systems in automotive industry, surveillance systems,

to name just a few possible applications. HDR video can also speed up the image acquisition in all applications where a large number of static HDR images are required, as for example in image-based techniques in computer graphics. Finally, with the spread of TV sets featuring enhanced dynamic range, broadcasting HDR video will be important, but may take a long time to actually occur due to standardization issues. For this particular application, enhancing current LDR video signal and transforming them to HDR videos by intelligent TV sets seems to be a more viable near-term solution.

Even though there are a number of products in the market labeled with HDR capabilities, the dynamic range, the accuracy and the tolerance to the movement in the scene of the output images and videos are far from satisfactory. The major difficulty is in developing the hardware for HDR capturing devices that can be widely available in the market. It is very expensive to manufacture sensors that can register contrasts beyond low dynamic range. Few studios have so far managed to develop HDR cameras. However, their solutions are expensive and require a long time to capture the full dynamic range. A few companies such as RED and Arri have been developing digital sensors capable of a higher dynamic range, but have yet to be released or made affordable. Therefore, there is a need for low cost solutions that can synthesize HDR content using only LDR capturing devices before the hardware obstacle can be overcome.

1.2 Thesis Objective

As mentioned in the previous chapter, the HDR cameras with a single sensor may not be available in the market soon due to the limitation of the hardware. Efforts have been put to derive algorithms that can generate HDR images from LDR images captured by LDR camera. Most of the existing methods perform well for still scenes but fail for scenes with motion. Videos however have become a major source for communicating and storing information nowadays. There is an increasing demand for algorithms to synthesis HDR videos with reasonable computational complexity. In this thesis, we aim to develop a method that generates HDR images from LDR ones and that are better than existing methods in three aspects:

- The algorithm should require a minimum number, namely two, LDR cameras so that the equipment is portable.
- The algorithm should not be computationally intensive. This is one of the key factors for a technology to be available in the market
- The algorithm should be able to construct more accurate images for different cases ranging from stationary scenes to scenes with fast motion. The algorithm can be applied easily to generate HDR videos by combine corresponding frames of the input LDR videos.

Chapter 2 reviews the existing algorithms in constructing HDR images from LDR images. It also reviews the state-to-art technologies in stereo matching. This is because the most challenging and critical part in the algorithm presented by this thesis lies in finding accurate special shifts between corresponding pixels in the input LDR images. In particular, two algorithms, Adaptive Normalized Cross Correlation (ANCC) [9] and Multi-view Multi-exposure algorithm [7], are covered in details. They are the most relevant algorithms which inspired me in developing the algorithm presented by this thesis.

Chapter 3 gives the explanation of our algorithm. It puts special emphasis on the image model and the cost functions used by our algorithm to find accurate spatial displacement between corresponding pixels in the input LDR images.

Chapter 4 compares the experimental results obtained using our algorithm. It also provides the statistics of the HDR images constructed as compared with the HDR images generated using the two state-of-the-art algorithms that have inspired me. The conclusion and proposals for future work are discussed in Chapter 5. The appendix describes a filter used in our algorithm: the Census filter.

Chapter 2 HDR Image Construction

2.1 Introduction

In recent years, several approaches have been developed to produce scenes with expanded dynamic ranges using LDR images. These approaches can be categorized into three types depending on the number of cameras and the number of LDR images used to construct the HDR images:

- Single camera with single LDR image. These approaches are most useful to recover the details in the old low quality images with valuable information
- Single camera with multiple LDR images taken under different exposures at consecutive time instances.
- Multiple cameras with multiple LDR images taken under different exposures at the same time instance.

2.2 HDR Image Construction from a Single LDR Image

For still scenes, one approach is to use a single LDR image and computes the inverse tone mapping curve of the camera used to capture this image. A tone mapping function converts a high dynamic range to a low dynamic range. Various tone mapping curves that compress the pixel values in images with high dynamic ranges to images with low dynamic ranges have been developed. By finding the inverse of these tone mapping curves, the pixel values in an LDR image are converted back to the high dynamic range [1].

2.2.1 Tone Mapping

Tone mapping is a process that converts a set of high dynamic range signals to a set of lower dynamic range signals. It addresses the problem of strong contrast reduction from the scene values to the displayable range while preserving the image details and color appearance. This is important as, in general, viewers appreciate viewing the original scene content. Tone mapping is widely used in image processing and computer graphics to display image signals

on various displays with limited dynamic ranges such as print-outs, CRT or LCD monitors and projectors.

Tone mapping are achieved by applying tone mapping operators on the high dynamic range signals. Various tone mapping operators have been designed to reproduce visibility and the overall impression of brightness, contrast and color of the real world onto a limited dynamic range display. Tone mapping operators can be classified into two categories: global and local. Global tone mapping operators are not computationally intensive. An example of a global tone mapping operator is the exponential function proposed in [REFERECE 21 in FRAMEWO]:

$$L_d(p) = 1 - e^{-\frac{L_w(p)}{\overline{L}_w}}$$

where $L_d(p)$ is the compressed luminance value at pixel p. $L_w(p)$ is the world luminance at pixel p. $\overline{L}_w$ represents the average luminance of the scene. Although global tone mapping operators have the advantage of low computational cost, they do not cope well with huge contract ratios. Local tone mapping operators have mapping functions that vary spatially depending on the neighbourhoods of pixels. Such operators concentrate on preserving contrast between neighbouring regions rather than absolute value. They are motivated by the fact that the human perception is most sensitive to contrast in images rather than absolute intensities. These operators are generally capable of compressing greater dynamic range. They usually also produce very sharp images, which preserve small contrast details very well. An example of a local tone mapping operator is the Gradient domain high dynamic range compression operator [18]. However, local operators often produce artefacts around the edges with high contrast.

2.2.2 Inverse Tone Mapping

One fairly straight way to recover the HDR image from the tone mapped LDR image is to find the inverse tone mapping operators. A framework for such an approach is laid out in [1]. The framework consists of four steps:

- Generate an initial HDR image by applying the calculated inverse tone mapping operator. The inverse tone mapping operators of many monotonically increasing tone mapping operators can be found easily.
- Identify the areas of high luminance using methods associated with importance sampling of light sources such as median cut.
- Create a map from density estimation of the area identified in step 2. This is to levitate the visual artefacts in the final HDR image due to the saturation in the bright areas in the LDR image.
- The final HDR image is composed through weighted linear interpolation of the input LDR image with the inverse tone mapping operator found in step one. The weight is calculated based on the map found in step 3.

Although this approach tries to solve the problem of the lost details in the saturated or coarsely quantized regions of the LDR image, it still cannot recover all the lost information in the areas. As a result, the constructed HDR images have errors and blurred regions, which are unacceptable for commercial use. However, these algorithms are very useful when upgrading old images and videos.

2.3 HDR Image Construction from Multiple LDR Images

2.3.1 Algorithms Using Single Camera

A better approach than that of using inverse tone mapping operators to construct HDR images is to use a single camera to capture multiple LDR images but with different exposures and at consecutive time instances. Each LDR image registers one part of the entire dynamic range of the scene. For example, one LDR image can have very low exposure to capture the details in the bright regions. Another LDR image can have very high exposure to capture the dark areas. A final HDR image that spans the entire dynamic range can be generated by combining these LDR images [5]. Such an approach avoids the loss of information due to saturation. This problem arises when inverse tone mapping operators are used to construct HDR images. Such approaches generally consist of the following four steps:

- Extract the luminance component of the input LDR images. This is because the change in the exposure only affects the luminance channel
- Find the camera response function from the input LDR images. The camera response function is a radiometric model which represents the mapping of the radiance (falling on the camera sensor) to the luminance pixel values in the LDR images.
- Obtain the inverse camera response function. Use the inverse camera response function to obtain the radiance maps of the LDR images. Combine the radiance maps to form the HDR image radiance map.
- Construct the final HDR image by integrating the HDR image radiance map with the chrominance components in the input LDR images. The resultant integration forms the HDR image.

The key step in such approaches is to find an accurate camera response function so that the pixel values in the LDR images can be correctly mapped to the radiance falling on the camera sensor. When the image of the scene is captured, the camera processes the amount of light (radiance) fallen on the sensor in two main steps, to convert the radiance to the luminance pixel values in the output image. In the first step, the camera compresses the dynamic range of the scene to the dynamic range that can be handled by the display system. In the second step, the compressed values are quantized to integer values. The integer values form the output as the pixel values in the image. This whole process can be represented by a non-linear function, called the camera response function. It is a function that represents how the radiance arriving on an image film/CCD, after passing through the lens, is transformed to actual pixel "brightness" values. The most popular function used by cameras for compressing the dynamic range of the scene is gamma correction function. Therefore, most algorithms use a gamma correction to model the camera response function.

Various algorithms have been developed in the past to find an accurate camera response function. Most algorithms start with modeling the camera response function as a gamma correction function because most of the cameras use such a function to compress the dynamic range of the scene as explained above. Then they adopt the model suggested by Mitsunaga in

[5]. Mitsunaga models the camera response function as a high order polynomial because it is proved that the gamma correction function can be approximated by a high order polynomial with satisfactory accuracy [5]. We have adopted this model in our algorithm presented in this thesis and will explain it in details in the next chapter.

The algorithms that use a single LDR camera to capture multiple images at consecutive time instances and then construct HDR images work will for still scenes. Some of the algorithms have been implemented for commercial use, such as the HDR function in Photoshop. Many photographers also use this approach to synthesis HDR images. Such approaches however result in defects when there is motion in the scene. This is because the positions of corresponding pixels in different LDR images shift their locations in the area of an image having motion. If the HDR images are constructed by directly summing the pixels with the same coordinated in the left and right LDR images, errors appear at the areas with motion. Such a requirement limits the use of these algorithms because most of the scenes in the world have motion. It also restricts the use of these algorithms in constructing HDR videos from LDR ones.

The static scene (i.e. that with no motion) requirement can be removed by designing special capturing systems. Beam splitters are used to generate multiple copies of the optical image detector whose exposure is pre-set by using an optical attenuator or by changing the exposure time of the detector. This approach is able to produce high dynamic range images in real time. Since the LDR images captured by such method are taken at the same instances, the scene objects are free to move during the capture process. However, such a set up is expensive. This is because it requires multiple image detectors, precision optics for the alignment of all the acquired images. It also requires additional hardware to process the LDR images.

Another approach is to use a different CCD design [19]. In this approach, each detector cell includes two sensing elements (potential wells) of different sizes (and hence sensitivities). When the detector is exposed to the scene, two measurements are made within each cell. The camera also has a special chip with an algorithm that processes the signals and outputs the final HDR image directly. However, this technique is expensive as it requires a sophisticated

detector to be fabricated. In addition, the spatial resolution is reduced by a factor of two since the two potential wells take up the same space as two pixels in a conventional image detector. Moreover, the technique is forced to process the outputs of the two wells on the chip. This forces the algorithm to be simple because the chip power is limited. Therefore, it limits the accuracy of the results.

A novel idea has been proposed to set up sensors that have spatially varying pixel exposures [2]. In this configuration, pixels in the image are grouped into a square of four. Each of the four pixels has different exposures. This pattern repeats over the entire images. The key feature of this set up is that it simultaneously samples the spatial dimensions as well as the exposure dimensions of image radiance. If one pixel in the image is saturated, its surrounding pixels (which would be captured at different exposures) are unlikely to be saturated. When a pixel produces zero brightness (under-saturation because the area is dark and the exposure time is short), it is likely to have a neighborhood pixel that produces non-zero brightness. Since images are expected to be smooth, except at the edges, the unsaturated surrounding pixels can be used to interpolate the central saturated pixel. The camera response function is calculated from the valid pixel values. The pixel values with zero brightness and maximum brightness are recovered by interpolating its neighboring unsaturated pixels. The advantage of this set up is that it does not require the scene to be still nor it requires any on chip processing of the signals before outputting the image.

The spatially varying exposure can be achieved by directly etching the pattern on the CCDs. It can also be achieved by lower cost solutions such as placing a mask outside the imaging lens next to the detector plane. A primary lens is used to focus the scene onto the mask plane. The light rays that emerge from the mask are received by the imaging lens and focused onto the detector plane. A diffuser may be used to remove the directionality of rays arriving at the mask. Then the imaging lens is focused at the diffuser plane. However, this setup would increase the cost of the LDR cameras as manufacturers have to redesign their production process to include the mask in the camera. In addition, it also reduces the effective resolution of the resulting HDR because some of the pixels with high exposure are expected to be saturated and some of the pixels with very low exposure are expected to produce low and

noisy intensities [2]. Due to the high cost and low effective resolution, this solution is also not suitable for manufacturing commercial cameras.

Recently, another algorithm that uses a single LDR camera to capture multiple LDR images that are differently exposed has been proposed [3]. This algorithm use commercial LDR camera to capture images at consecutive instances and process the LDR images offline to generate HDR image. This algorithm distinguishes itself from other algorithm in following two aspects:

- The capturing system is programmed to automatically vary the exposures under which consecutive LDR images are taken.
- The offline algorithm calculates the motion vectors between consecutive images. The HDR image is formed by combining different LDR images after spatial adjustment [4]–[6] using the motion vectors.

Since the algorithm calculates the motion vectors, it is not limited to still scenes and can be used to generate HDR videos. However, the post processing algorithm is computationally expensive. In addition, the resulting pictures have significant artifacts in those scenes with high motion because computing accurate motion vectors for scenes with fast motions is not easy [3].

2.3.2 Algorithms Using Multiple Cameras

Because constructing the fast motion scene in HDR images is difficult to achieve using a single camera, the use of multiple cameras to obtain multi-view, multi-exposed LDR images have been proposed [7]. Our algorithm also falls into this category. The two major advantages of such an approach are:

- They use standard commercial LDR cameras. They do not require any special designs of CCDs. Therefore, the set up cost using such an approach is low and thus possible for commercial uses.
- This approach takes LDR images at the same instance. No temporal adjustment (calculation of motion vectors) is required to find the corresponding pixels in the

different LDR images. Therefore, they are capable of construct HDR images for scenes with high motion.

However, one challenge faced by such an approach is to find the spatial adjustment for corresponding pixels in the LDR images. This is because the LDR cameras are placed adjacent to each other. The pixels that register the same area in a scene appear at slightly different locations in the LDR images. It is important to correctly find the shifts for each pixel in the LDR images for the following two purposes:

- To accurately calculate the camera response function. The camera response function is calculated by exploiting the difference in the values of corresponding pixels under different exposures.
- To accurately construct the HDR images. The pixel values of the HDR images are usually calculated as a weighted sum of the corresponding pixels in the input LDR images.

2.4 Stereo Matching

Stereo matching is a field in computer vision that has matured over the last few decades. It is mainly used to estimate the distance between the camera and the objects in the scene. Such distance is called the depth of the objects in the scene. In order to find the depth of the objects, more than one camera are aligned and placed next to each other. Images of the same scene are captured by these cameras. Since images are taken at slightly different locations, the positions of pixels representing the same objects shift their locations in these images. The pixels representing objects nearer to the camera have larger shifts than those pixels representing objects further away from the camera.

2.4.1 Disparity Map

A convenient way of representing the depth of the objects is to plot an image, named disparity map. The horizontal (vertical) disparity is defined as a difference between the column (row) coordinates of pixel locations of corresponding pixels in the left and right

image of a stereo pair. In stereo matching, images are usually rectified first before calculating the disparity map so that the search for the corresponding pixels can be performed in one direction (either horizontal or vertical) only. This is because the cameras used to capture the stereo images are usually very hard to be aligned to have the exactly same image plane.

Assume the cameras used to capture the stereo images can be approximated by the pin-hole camera model with positions (exaggerated) shown in Figure 2.1. O_L and O_R represent the focal points of left and right cameras. P_L and P_R represent the image plane of the left and right cameras. P_LE_L and P_RE_R are called the epipolar lines of the left and right images. If the two lines are not in the same plane, the displacement of pixel register the point P in the right image relative to the left images is in both horizontal and vertical direction. This complicates the matching process because the algorithm has to search in both directions for each pixel in the left image to find the corresponding ones in the right images. In order to simplify the matching process, the image coordinates from the cameras can be transformed to emulate having a common image plane so that their epipolar lines coincide with each other. Namely, P_LE_L=P_RE_R. This process is called image rectification. As a result, for each point in one image, its corresponding point in the other image can be found by looking only along a horizontal line.

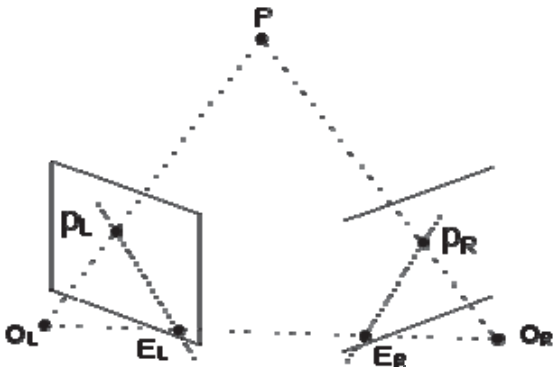


Figure 2.1 Epipolar geometry for the case of using two cameras to capture stereo images

In a disparity map, pixels representing the objects nearer to the camera are assigned greater values, and pixels representing the objects further away from the camera are assigned with

smaller values. Equation (1) shows such a relationship between the disparity value and the depth of the object. The depth is inversely proportional to disparity and is obtained using the property of similarity triangles shown in Figure 2.2.

$$D = \frac{\lambda bf}{d} \tag{1}$$

where

$$\lambda = \frac{pixels}{meter} \tag{2}$$

D represents the radial distance, which is the perpendicular distance between the object and the baseline of the two cameras. f represents the focal length of the camera. b is the distance between the two cameras along the baseline. d represents the displacement in terms of the number of pixels of the corresponding pixel between the two images.

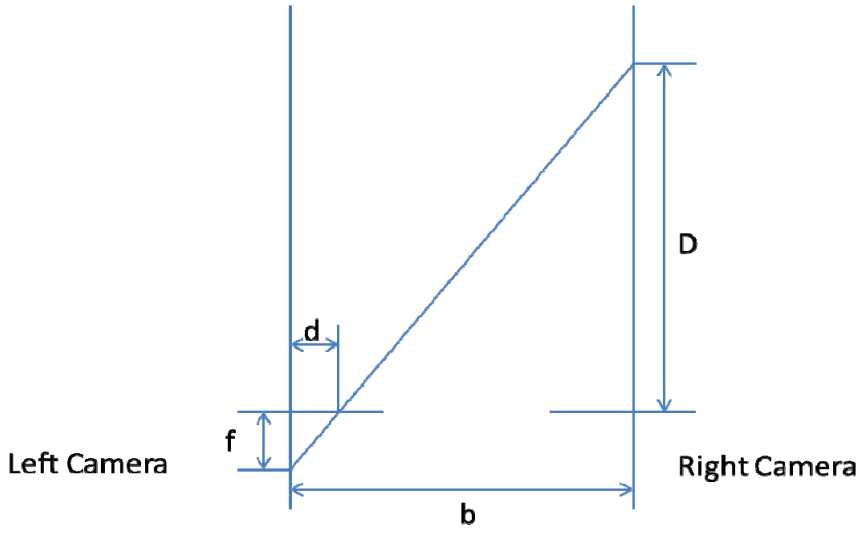


Figure 2.2 Relationship between the disparity in the disparity map and the distance of the object from the image plan of the cameras

As mentioned in the previous section, if we use multiple cameras placed next to one another to capture multiple LDR images under different exposures and at the same time instance to construct HDR images, one of the key step is to find an accurate disparity map for the input LDR images. All the stereo correspondence algorithms have a way of measuring the similarity of image locations in order to construct the disparity map. The function used to measure this similarity is called the matching cost. For each pixel in the image, the matching cost is for all the possible disparities. The disparity value that produces the smallest matching cost is chosen to be the value of this pixel in the disparity map. Depending on how this matching cost function is defined, stereo matching algorithms can be categorized into two categories: local and global.

2.4.2 Local Stereo Matching Algorithm

Local stereo algorithms are window based. They usually aggregate the sum of a certain cost measure over a window and use it as the cost function. Simple cost functions are the sum of absolute differences (SAD), sum of squared differences, normalized cross correlation (NCC) or sampling-insensitive absolute differences [9]. More complicated cost functions involve applying filters such as Laplacian of Gaussian (LoG) filter, Rank filter and Census filter (explained in the Appendix) to the images first and then calculating the absolute differences (for the LoG filter) or Hamming distance (for Rank and Census filters). The latter functions are more robust to noise [9]. This is because of their ability to tolerate small differences between corresponding pixel values in different input images.

The matching decision for a pixel (when using local stereo methods) is made solely based on its surrounding local pixels within the window the method uses. The main problem with such methods is how to determine the optimal size, shape of the window and the weight distribution of the pixels within the window relative to the central pixel. These restrictions cause the local stereo methods to form disparity maps with more errors. In addition, since the local stereo methods lack global information about the image, the disparity maps they generate tend to have less smoothness and less well defined edges.

2.4.3 Global Stereo Matching Algorithm

Global stereo methods, in most cases, rely on Bayesian stereo matching. The relationships among all the pixels in the two input images are encoded into a global objective function which usually consists of a cost term and a smoothness term. Therefore, the global objective function avoids the optimization problems faced by local stereo matching methods, related to the size and shape of the matching window.

Most global stereo methods share the following two common assumptions:

- The stereo images are rectified (explained in the previous chapter) so that the disparity between two pixels in two images is in the horizontal direction.
- The disparity map between two rectified images can be modeled as a Markov Random field (MRF).

MRF is very powerful at modeling spatial relationships. The global objective function usually consists of an observation term and a smoothness term. The observation term usually represents the data in the images, while the smoothness term ensures the continuity in the disparity map. The disparity map is calculated by optimizing this global objective function usually via belief propagation or graph cuts [8].

Most local and global algorithms share the common assumption that corresponding pixels in the stereo images have similar values, i.e. they have small radiometric changes between corresponding pixels. Thus, these methods perform well on images of the same illumination and exposure, which is not the case in our present application. Global stereo matching methods are more robust to noise introduced by a camera during the image capture process. However, when the radiometric variations between the input images are large, very few algorithms (such as the adaptive normalized cross-correlation (ANCC) algorithm proposed in [10]) can successfully estimate accurate disparity maps [9]. Even fewer algorithms can tolerate the presence of saturated regions. ANCC has been shown to fail in saturated image areas [10]. As mentioned earlier, the LDR images in our setup may have large radiometric changes and saturated regions.

2.4.4 Adaptive Normalized Cross Correlation Function

Recently, the adaptive normalized cross-correlation (ANCC) function was proposed as the matching cost function to find the disparity maps of stereo images captured under illumination and camera variations [10]. It uses a modified version of the standard matching cost function NCC and performs matching in the Log space to enable the algorithm to find disparity maps for the stereo pair which has the large radiometric changes between the corresponding pixels.

Before deriving ANCC function, the paper [10], as most of other algorithms, models the camera response function as a gamma correction as discussed in the previous chapter:

$$C(p) = \rho(p)\kappa_c R_c(p)^{\gamma} \tag{3}$$

where $C(p)$ is the value at pixel p in channel c. This value measures the amount of light $R_c(p)$, namely radiance, falling on the sensor of the camera. $\rho(p)$ is a function that depends on the angle between the light direction and the surface normal at the point registered by pixel p.

The motivation of the paper [5] is to propose a cost function that is independent of factors $\rho(p)$ and k. This is achieved by transforming the pixel value in each channel through a series of steps we are going to discussed shortly. As a result, the algorithm can tolerate large radiometric changes in the input images. This is achieved in the following steps.

First, the algorithm transforms the pixel values in R, G and B channels to the Log space. This is for the use of the invariant property of NCC to the affine transformation of pixel values in the images [19] [20]. Assume κ_L, κ_R, v_L and v_R are constants and I_L and I_R are the pixels values corresponding to each other in the left and right images , the invariant property of NCC enables us to use NCC as the cost function to match pixels in the left and right stereo images if the pixels values in images undergo the following transformation:

$$I_L \rightarrow \kappa_L I_L + v_L$$

$$I_R \rightarrow \kappa_R I_R + v_R$$

Next, choromaticity normalization is used to eliminate the effect of light direction, namely the factor $\rho(p)$. This is achieved by subtracting the average value of a pixel over three color channels from its actual value in each of the color channel.

Assume $R_r(p)$, $R_g(p)$ and $R_b(p)$ are the values of the pixel p in each of the channel R, G and B. k_r, k_g and k_b are the factors defined in equation (3) in each of the channel R, G and B. The effect of factor $\rho(p)$ can be eliminated by applying the following equation to the pixel values in each of the R, G and B channel.

$$C''(p) = C'(p) - \overline{C}'(p)$$

where $C'(p)$ is the value of pixel p in the color channel c in the Log space. It is given by

$$C'(p) = \log \rho(p) + \log \kappa_c + \gamma \log R_c(p)$$

and $\overline{C}'(p)$ is given by

$$\overline{C}'(p) = \frac{R'_R(P) + R'_G(p) + R'_B(p)}{3}$$

$$= \log \rho(p) + \frac{\log \kappa_R \kappa_G \kappa_B}{3} + \frac{\gamma\left(\log R_R(p) R_G(p) R_B(p)\right)}{3}$$

After rearranging the log terms, the pixel values in the left image after transformation can be expressed as

$$C''_L(p) = \log \frac{\kappa_R}{\sqrt[3]{\kappa_R \kappa_G \kappa_B}} + \gamma \log \frac{R_L(p)}{\sqrt[3]{R_{L_R}(p) R_{L_G}(p) R_{L_B}(p)}}$$

$$\triangleq \alpha_L + \gamma \Re_L(p)$$

Similarly, the pixel values in the right image after the transformation can be expressed as:

$$C''_R(p + f_p) \triangleq \alpha_R + \gamma \Re_R(p + f_p)$$

Since the corresponding pixels in the left and right images register the same point in the scene, $\Re_L(p)$ is expected to be the same as $\Re_R(p + f_p)$. Using the invariant property of

NCC, we can use NCC as the cost function to match for $C''_L(p)$ and $C''_R(p+f_p)$ as well. This means that after the above transformations of the pixels values, the light direction and radiometric changes do not affect the matching result if NCC is used as the cost function. This modified version of NCC (after transforming the pixels values using above mentioned steps), named ANCC, is able to find disparity maps for the images with large radiometric changes.

However, there is no interpolation process designed in ANCC function. It purely relies on the presence of textures to find the corresponding pixels in the input stereo images. As a result, ANCC cannot handle the stereo image with saturated areas [10] because the texture information is lost in the saturated areas and thus the matching process fails.

2.4.5 Multi-view Multi-exposure Stereo Matching

Another approach that computes the disparity map from multiple images is proposed in [7]. The algorithm can handle large radiometric changes between the input stereo images as well as the presence of saturated regions in the stereo images. The algorithm can be summarized in following steps:

- Compute the initial disparity maps from the unsaturated pixels in the multiple (more than two) input LDR images captured by multiple LDR cameras with the same camera response function.
- To compensate for the difference in exposure among the input LDR images, compute the common camera inverse response function from the initial disparity maps. Then convert each input LDR images to its radiance map using this inverse response function.
- Refine the initial disparity map by running step 1 again, using the radiance maps obtained in step 2.

In order to improve the accuracy of the disparity map, the study in [7] focused on deriving a method that can estimate the camera inverse response function from pixels aligned incorrectly using inaccurate disparity maps.

The algorithm defines the camera response function as:

$$I\left(p\right) = f\left(eR\left(p\right)\right)$$

where f is a non-linear function that maps the scene radiance R at pixel p to the intensity value I when the image is captured under exposure e. It uses the Brightness Transfer Function (BTF) [21] to solve this camera response function. For a pair of stereo images, the BTF is defined as:

$$T_{i,j}\left(I_i\right) = I_j$$

where I_i and I_j are the corresponding pixel brightness in the left and right images. The BTF is usually semi-monotonic. Depending on the exposure ratio $e_{i,j}$ between the two input images, following constrains are applied:

$$\begin{cases} T_{i,j}\left(I_i\right) \le I_j & e_{i,j} \le 1 \\ T_{i,j}\left(I_i\right) \ge I_j & e_{i,j} \ge 1 \end{cases}$$

To compute the BTF between the input images, the algorithm groups all the input images into pairs. For each pair of input images, it plots a two-variable joint histogram J, in which a give entry $J\left(I_i, I_j\right)$ stores the corresponding pixel values in the two input images. The joint histogram is then partitioned into two triangles along the line $I_i = I_j$. To maintain the semi-monotonicity of the BTF, the following conditions are applied:

$$T\left(0\right) = 0$$

$$\begin{cases} T_{i,j}\left(I_i\right) \le I_j & \sum_{uppertriange} J\left(I_i, I_j\right) \ge \sum_{lowertriange} J\left(I_i, I_j\right) \\ T_{i,j}\left(I_i\right) \ge I_j & \sum_{uppertriange} J\left(I_i, I_j\right) \le \sum_{lowertriange} J\left(I_i, I_j\right) \end{cases}$$

The algorithm then uses the dynamic programming technique proposed by Kim and Pollefeys [22] to estimate the BTF that satisfies the above conditions. Once the two BTFs for each pair of the input LDR images, $T_{i,j}\left(I_i\right) = I_j$ and $T_{j,i}\left(I_j\right) = I_i$, are calculated, the algorithm

uses the EMOR model of Grossberg and Nayar [23] to find the inverse camera response function.

When calculating the inverse camera response function, the above algorithm is more robust to large changes in the brightness of the corresponding pixels in the input images. As a result of a more accurate inverse camera response function, the radiance maps recovered in step 2 are expected to have fewer errors than the radiance maps recovered using other techniques. This in turn improves the accuracy of the disparity maps obtained in the refinement step by re-running the matching process used to obtain the initial disparity maps. However, this approach involves a complex capturing system because it uses more than two cameras. This increases the amount of work to align the cameras and to rectify the captured images. In addition, the input images are grouped into pairs and processed individually. This also increases the amount of the computations and thus limits its use in the industry.

Chapter 3 Constructing HDR Image from Two LDR Images

3.1 Overview

In this chapter we present and describe a method that obtains an HDR image from two LDR images. In [7], a method that uses many (more than two) images obtained from different cameras is proposed. The images are all captured with different exposures and at the same instance of time.

Our approach is inspired by the work in [7] where we propose the use of two cameras. Two images are captured using different exposures and at the same instance. The four sets of input LDR images we used to test our algorithm are shown in Fig. 3.1 to Fig. 3.4. The ones with the lower exposure register the information in the dark regions of the scene. The ones with higher exposure register the information in the bright regions of the scene. Each set of two images therefore complement each other in that they contain more information for forming a HDR image with a higher dynamic range. This is because one image has more details about the dark regions of the captured scene and the other contains more information about the bright regions. Thus more information is available than that obtained from one image.

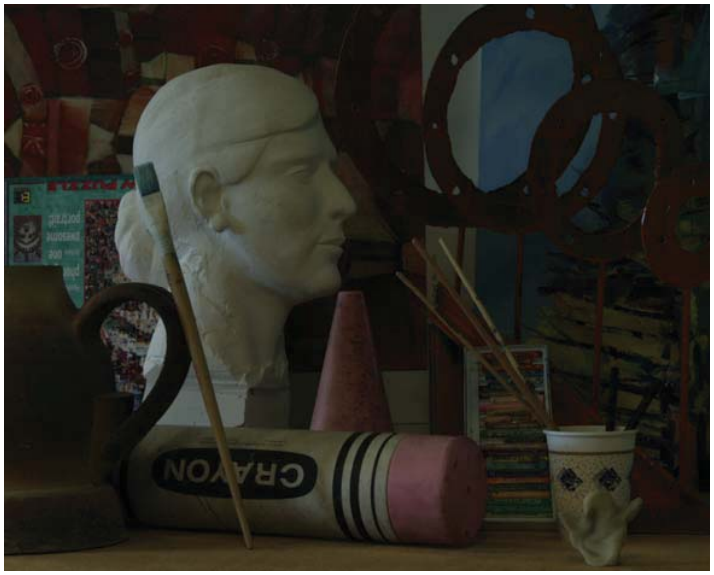
Using two cameras allows us to overcome the temporal adjustment in the fast motion scenes as the two LDR images are captured at the same instance of time .The use of more than one camera however has the problem of image spatial disparity. Thus the key step in such approach is the accurate computation of the disparity map, so that the pixels from the different LDR images can be aligned correctly to form the HDR image. This is a challenging task because the input LDR images have large radiometric changes as well as saturated regions. Not many studies that deal with such cases have been proposed. The four sets of the input LDR images we chose to test the performance of our algorithm impose different degrees of challenges for calculating the disparity map.

Our algorithm is inspired by [7] but faces a greater challenge in computing accurate disparity maps. In [7], many (more than two) input LDR images are used to construct the HDR image.

If one pixel is saturated in one picture, it is unlikely that its corresponding pixels in the rest of the images are saturated. Therefore, the saturated pixels can be ignored. Instead, the pixels values in the rest of images are used calculate the disparity map and construct the HDR image.

Figure 3.1 It shows the input LDR images of the scene Clothes. In the first image has scattered areas of pixels with very low brightness. These patches are of relatively small size. The second image has small patches of saturated pixels in the center and right.

Figure 3.2 This figure shows the input LDR images for the scene Dolls. This pair of images show larger areas of saturated and unsaturated pixels than those in the scene Clothes. In addition, the saturation and unsaturation occurs at the boundaries of the objects in the scene. For example, in the second image, the head of the central doll merged with the right arm of the bear above her.



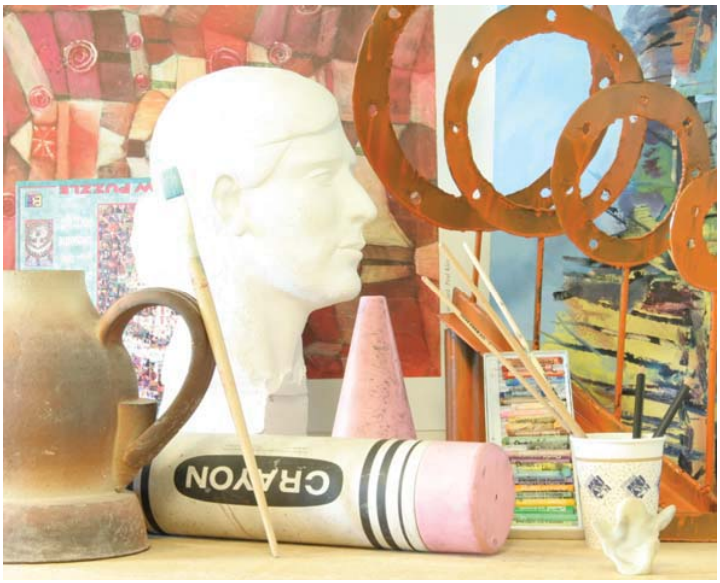


Figure 3.3 This figure shows the input LDR images for the scene Arts. This scene is more difficult to process than Clothes and Dolls. The head of the statue in the second image is saturated. The top part of the pen is also saturated in the second image. The two saturated regions are merged, making it hard to assign correct values at the edges of these two regions in the disparity map. In addition, it has thin objects such as the sticks to the right of the scene. Some parts of the copper kettle to the left of the image are also saturated (nearly white color) in the second image while the corresponding pixels in the first image are brown.

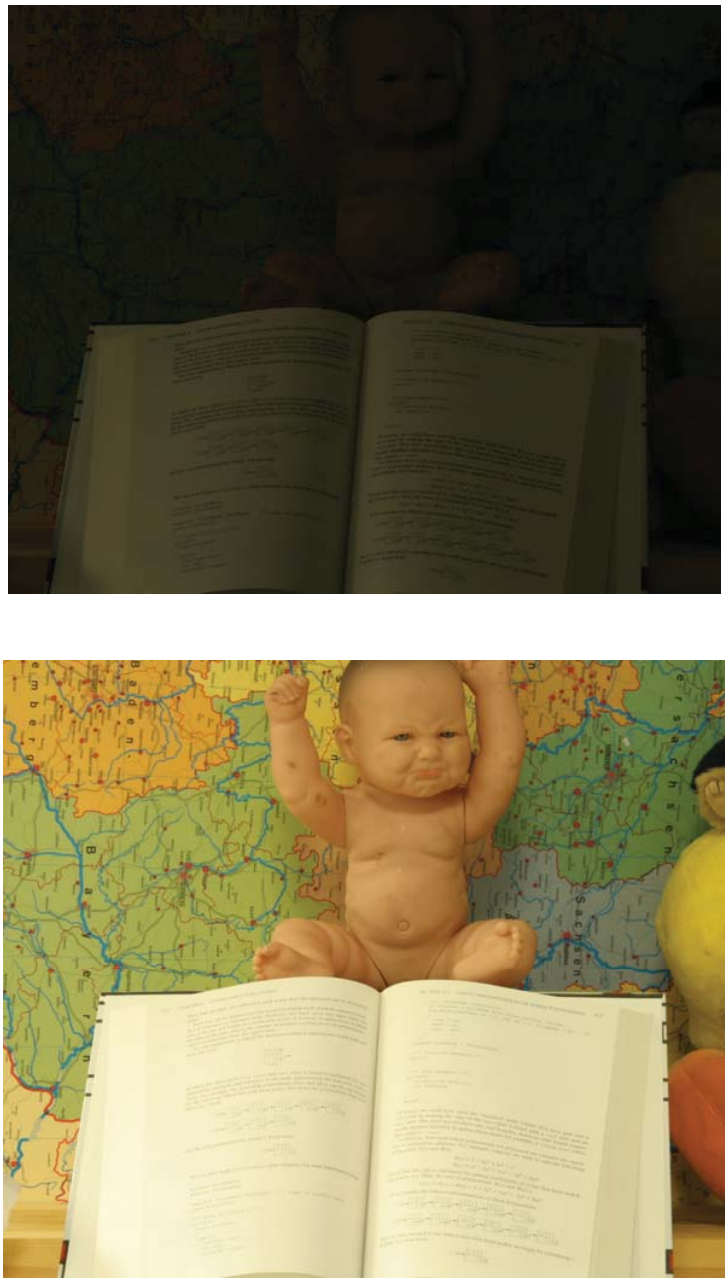


Figure 3.4 This figure shows the input LDR image for the scene Baby. The radiometric difference between the two images is very large. In addition, because the background has high texture, it imposes additional challenge to the matching process because some of the textures shown in the left image is occluded by the baby and book in the right image.

In our set up, we only have two input LDR images. From these images the disparity map is calculated and then used to construct the HDR image. The saturated pixels in one image cannot be ignored when calculating the disparity map and constructing the final HDR image. Therefore, for the saturated regions in one image, we have to derive a method to get the correct disparity value using the unsaturated information from the other image and thus assign correct values to those pixels in the constructed HDR image. In order to increase the accuracy of the disparity map and recover the lost information in the saturated regions, in [7] the focus has been to improve the accuracy of the camera response function. Our algorithm however focuses on improving the initial estimate of the disparity map as well as the refinement of the initial estimate of the disparity map. Figure 3.5 illustrates a block diagram of our proposed scheme. Our algorithm differs from the algorithm in [7] in the several aspects.

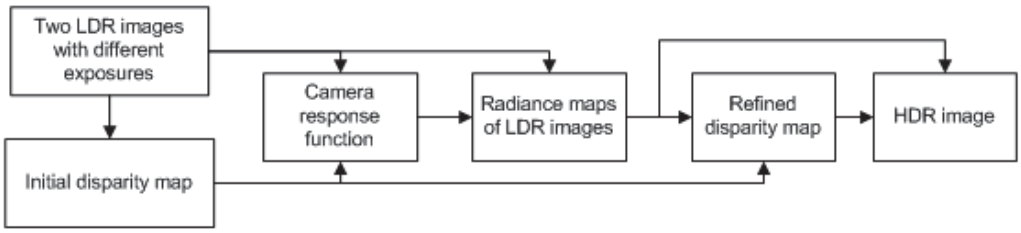


Figure 3.5 Our proposed scheme for HDR construction

First, our method uses only the luminance values of the input LDR images to calculate the initial disparity map. Luminance is an indicator of how bright the surface appears. Usually the value of a pixel in an image is formed by luminance and chrominance information. In [7], the disparity map is calculated using both the luminance and chrominance information.

Second, our method finds an initial estimate of the disparity map using the normalized cross correlation (NCC) function as the matching cost. We use a different weighting function use in [7] to calculate NCC for each pair of pixels to be matched. The weighting function improves the accuracy of the initial disparity maps.

Third, instead of using all the pixels in the initial disparity maps as [7], we use a subset of the pixel values in the LDR images. This subset of pixels consists of pixels are obtained as follows: We construct two initial disparity maps for the two input LDR images. The first uses the left image as the reference and the second uses the right image as the reference. The subset of pixels we are after consists of every pixel whose values in both disparity maps match with each other, i.e. have the same value. Since we exclude the inaccurate disparity values from the initial disparity maps, we can use a less computationally intensive algorithm than the one in [7].

In addition, to refine the initial estimate of the initial disparity map, we apply the census filter (covered in details in Appendix) to the radiance maps of the two input LDR images and use the Hamming distance as the cost function. This is because the radiometric variations (the difference in the luminance values) between the left and right radiance maps are less than the variations between the two input LDR images. Therefore, the stereo matching algorithm in the refinement step can be handled by a less computational expensive cost function than that used in estimating the initial disparity map. The refinement step is also designed to interpolate the values in the saturated regions using those in the unsaturated pixels in the input LDR images.

Our algorithm improves on the algorithm in [7] in three aspects:

- By using two cameras, our method simplifies the set up and reduces the computational complexity. Such setup also helps improve the estimation of the disparity map. The two major factors that reduce the accuracy in computing the disparity map are occlusion and radiometric variation. Our setup minimizes the problem of occlusion as the two cameras are placed in proximity. As a result, the objects captured by one camera are less likely to be blocked in the image captured by the other camera as the angles of views of the two cameras in our setup are almost identical.
- Our algorithm is less computationally intensive. This is because our algorithm uses only the intensity channel, instead of R, G, B channels [7] to in the cost function when estimating the initial disparity maps. In addition, in the refinement

step, we apply Census filter to the radiance maps and then use Hamming distance between two pixels, which is faster to compute than NCC, as the matching cost to calculate the disparity map. In [7], the refinement step still uses NCC between two pixels as the matching cost to calculate the initial disparity map.

- As a result of a better refinement step, our algorithm generates the disparity map with less errors and better smoothness and in turn constructs an HDR image that exhibits fewer artifacts.

3.2 Disparity Map Computation

In this paper, we propose a global stereo matching method to find the disparity map of two input LDR images that have 1) different exposures; 2) large radiometric changes; 3) areas with saturated regions. We model the matching problem via a Bayesian approach. In this approach, we use a vector $\boldsymbol{f}$ to represent the disparity f_p's of every the pixel p in the disparity map. The optimal vector $\boldsymbol{f}$ should contain disparities f_p's that minimize an energy function $\boldsymbol{E(f)}$ defined as equation(4) [11]. The energy function is composed of a pixel dissimilarity term $\boldsymbol{E_d(f)}$ and a pixel smoothness term $\boldsymbol{E_s(f)}$. It measures the degree of difference between corresponding pixels and the discontinuities of the disparity map for the disparity vector $\boldsymbol{f}$.

$$E(f) = E_d(f) + E_s(f) \tag{4}$$

$\boldsymbol{E_d(f)}$ and $\boldsymbol{E_s(f)}$ are

$$E_d(f) = \sum_p D_p(f_p) \tag{5}$$

$$E_s(f) = \sum_{p,q \in N(p)} V_{pq}(f_p, f_q) \tag{6}$$

where $\boldsymbol{E_d(f)}$ measures the summation of dissimilarities of all pixels in the disparity map and $\boldsymbol{E_s(f)}$ controls, i.e. maximizing, the smoothness of the disparity map. The terms $\boldsymbol{f_p}$ and $\boldsymbol{f_q}$ in (3) are the disparity values corresponding to a pixel $\boldsymbol{p}$ and a neighboring pixels $\boldsymbol{q}$ that lies in a

window $N(p)$ centered at pixel p. $D_p(f_p)$ measures the cost associated with matching a pixels p in the left LDR image and $p + f_p$ in the right LDR image. $V_p(f_p, f_q)$ is a function at pixel p in the left LDR image that measures the sum of the differences between the disparity (f_p) at pixel p and the disparities f_q of every neighboring pixel q of p in the window $N(p)$. Thus, minimizing the function $V_p(f_p, f_q)$ will maximize the smoothness in the disparity map at pixel p.

Before obtaining the energy terms $E_d(f)$ and $E_s(f)$, in the rest of this section, we will first discuss how the imaging model and the differences in lighting and exposure affect our choice of these energy terms in the rest of this section.

3.2.1 Imaging Model

To determine the scene radiance from the measured pixel data, imaging models are used. Different imaging models have been presented in the literature. In this paper, we employ two models. The first model is the gamma correction model, equation (8) and (9) below, as many other methods mentioned in the previous chapter. This model approximates the image luminance values in terms of the scene radiance. The second model the polynomial model, equation (10). This model is used to approximate the inverse of first model. It obtains the scene radiance at a pixel in terms of its luminance values in the image.

We use the relationship between the pixel luminance values and the scene radiance, and the first gamma correction model to obtain a suitable cost function for the dissimilarities in the disparity map $E_d(f)$. This cost function is used to find an initial disparity map. Then we use the initial disparity map to compute the coefficients in the second polynomial model [5] so that we can obtain the scene radiance from the image intensities. Finally, we refine our initial disparity estimate of the true disparity map using the scene radiance estimated from the polynomial model.

3.2.1.1 Gamma Correction

Every camera defines and quantizes an estimate of the scene radiance $\boldsymbol{R}$ explained in the previous chapter. In a stereo setup, both images are obtained from the same scene radiance but the recorded intensities have horizontal shifts in the pixel locations of both images. The image intensities recorded by a camera can be modeled as scaled gamma corrections of the scene radiance $\boldsymbol{R}$:

$$I_c(p) = \rho(p) k R_c(p)^{\gamma} \tag{7}$$

This is the same model used in method proposed in [5] to derive ANCC function. $\boldsymbol{I_c(p)}$ is the value at pixel $\boldsymbol{p}$ in the color channel $\boldsymbol{c}$. It is approximated as a scaled gamma correction of the radiance $\boldsymbol{R_c(p)}$, *i.e.*, the amount of light that falls on the sensor of the camera corresponding to pixel $\boldsymbol{p}$. $\boldsymbol{\rho(p)}$ is dependent on the angle between the light direction and the surface normal at the point registered by pixel $\boldsymbol{p}$. $\boldsymbol{\gamma}$ is the correction factor employed by the camera response curve to convert the amount of light $\boldsymbol{R}$ fallen on the camera sensor to image intensities.

In our set up, the two cameras are place very close to each other. As a result, the angles between the light directions and the surfaces normal to the corresponding pixels registered by the two cameras are to be almost the same. This means that the above gamma correction model defined by equation (7) can ignore the factor $\boldsymbol{\rho(p)}$ and k . As a result, the chrominance channels of the input LDR images are expected to have little difference. The only difference is introduced by different exposures under which the images are taken. Since the exposure only affects the luminance values of the pixels in the image, we perform stereo matching using the information in the luminance channels of the input LDR images only. Therefore, the imaging model at pixel $\boldsymbol{p}$ is reduced to the following expressions [6]:

$$I_l = R^{\gamma} \tag{8}$$

$$I_r = (eR)^{\gamma} \tag{9}$$

where I_l and I_r are the left and right image intensities, e is the exposure ratio between the left and right images. The γ factor can be eliminated by converting the luminance value of each pixel to the log space.

3.2.1.2 Polynomial Camera Response

In [3], different camera response functions that model the relationship between the radiance fallen on the senor and the luminance value at a pixel in the image, are compared. It shows that the response curve can be modeled as an nth order polynomial function [7] of the pixel values I in the image.

$$R(I) = \sum_n c_n I^n \tag{10}$$

where $R(I)$ represents the radiance fallen on the sensor at a pixel with intensity value I. The study also showed that it is sufficient to use $n \leq 4$ to build an accurate model to find the values of the radiance from the pixel intensities. To estimate the values of c_n, we obtain two disparity maps $Disp_l$ and $Disp_r$ using the left and the right input LDR images as the reference respectively. For our algorithm, only those pixels in the two disparity maps that match each other correctly are used in obtaining the coefficients c_n, i.e. those pixels that satisfy equation (11):

$$Disp_l(p_l) = Disp_r(p_r) \tag{11}$$

where $Disp_l(p_l)$ is the value of a pixel p_l in the initial disparity map calculated using the left image as the reference. $Disp_r(p_r)$ is the value of the pixel p_r in the initial disparity map calculated using the right image as the reference. p_l and p_l are the corresponding pixels in the input LDR images. Their relationship is given by:

$$p_r = p_l + Disp(p_l)$$

We also exclude the pixels with values equal to 0 so as to minimize the error introduced by under-saturation. The polynomial coefficients c_n are then found by minimizing the cost function:

$$J(c_n) = \sum_{p \in P} \left[\sum_n c_n I_l^n(p) - e \sum_n c_n I_r^n(p) \right]^2 \quad (12)$$

where P is the set of valid pixels defined by equation (11), c_n are the polynomial coefficients, and e is the exposure ratio between the two images.

3.2.2 Computing the Disparity Map

The disparity map represents the integer values that characterize the lateral displacement of pixels of an object in the left image compared to its position in the right image. This map is represented by the vector $\boldsymbol{f}$ and f_p is an element in the vector $\boldsymbol{f}$. To compute the vector $\boldsymbol{f}$, we minimize the energy function $E(\boldsymbol{f})$ defined in equation (4). However, we must first define the dissimilarity term $E_d(\boldsymbol{f})$ and the smoothness term $E_s(\boldsymbol{f}, N)$.

3.2.2.1 Pixel Dissimilarity

We choose the normalized cross correlation (NCC) as the pixel similarity measure. In [7] and [9], it is shown that NCC is the best cost function that copes with exposure variations. Before computing NCC, we convert the image pixel intensities to the log space so that the exposure ratio e does not affect the results

$$I_l^{'} = \log I_l = \gamma \log R \quad (13)$$

$$I_r^{'} = \log I_r = \gamma \log e + \gamma \log R \quad (14)$$

For a pixel p whose corresponding disparity is f_p, NCC is given by the following expression:

$$NCC(p, f_p) = \frac{\sum w_l(p,t)\, w_r\left(p + f_p, t + f_t\right) \tilde{I}_l(p) \tilde{I}_r(p + f_p)}{\sqrt{\left|w_l(p,t)\tilde{I}_l(p)\right|^2} \times \sqrt{\left|w_r\left(p + f_p, t + f_t\right)\tilde{I}_r(p + f_p)\right|^2}} \tag{15}$$

where $f_p \in f$ is the disparity of pixel p, w_l and w_r are bilateral weights defined over a window $W(p,t)$ centered at pixel p in the left image and $(p + f_p)$ in the right image respectively, and $\tilde{I}_l$ and $\tilde{I}_r$ are functions of the pixel values in the left and right images respectively which we will define below. t is a neighbouring pixel of the central pixel p in the window $W(p,t)$. Notice that the effects of γ and e in equation (13) and (14) are cancelled in equation (15).

The fattening effect is one of the major drawbacks of the matching cost function that use windows [12]. The errors caused by the fattening effect occur all over the image, especially at the edges objects with strong depth discontinuity. This is because the central pixel of a window tends to inherit the disparity of the more contrasted pixels in the block [24]. As a result, at the edges of objects that have different distances from the camera, a brighter object tends to appear larger than its actual size.

In our algorithm, we include bilateral weights $w_l(p,t)$ and $w_r(p,t)$ in NCC, equation (15) to reduce the fattening effect. The bilateral weights are functions of the pixel p and its neighbouring pixels t. The weight function is given by the following expression:

$$w(p,t) = \exp\left[-\frac{\|p - t\|^2}{2\sigma_d^2} - \frac{\left\|I'(t) - I'(p)\right\|^2}{2\sigma_s^2} \right] \tag{16}$$

where σ_d and σ_r are the space and range smoothing parameters, respectively. t represents a pixels surrounding the central pixel p in the window $W(p)$. The first term in the exponent measures the spatial difference between pixels p and t. The second term measures the difference in the intensities registered by pixels p and t.

The NCC is effective at finding similarities in highly textured surfaces. Therefore, we subtract the low frequency image components before performing the similarity matching. The functions $\tilde{I}$ are then chosen so that:

$$\tilde{I}_l(p) = I_l'(p) - \frac{\sum_{t \in W(p,t)} w(p,t) I_l'(p)}{\sum_{t \in W(p,t)} w(p,t)} = \gamma \left[\log R(p) - \frac{\sum_{t \in W(p,t)} w(p,t) \log R(p)}{\sum_{t \in W(p,t)} w(p,t)} \right] \quad (17)$$

Similarly,

$$\begin{aligned} \tilde{I}_r(p) &= \gamma \left[(\log e + \log R(p)) - \frac{\sum_{t \in W(p,t)} w(p,t)(\log e + \log R(p))}{\sum_{t \in W(p)} w(p,t)} \right] \\ &= \gamma \left[\log R(p) - \frac{\sum_{t \in W(p,t)} w(p,t) \log R(p)}{\sum_{t \in W(p,t)} w(p,t)} \right] \end{aligned} \quad (18)$$

Note that equation (17) and (18) show that the NCC measure when calculated using $\tilde{I}$ is unaffected by γ and e.

As NCC measures the similarity between pixels $I_l(p)$ and $I_r(p)$, the dissimilarity term can then be expressed as follows:

$$E_d(f) = \sum_p D_p(f_p) = \sum_p \left(1 - NCC(p, f_p)\right) \quad (19)$$

3.2.2.2 Pixel Smoothness

In stereo imaging, the pixels representing the same solid object should have similar disparity values. Therefore, the disparity map should be smooth within area corresponding to solid objects. The potential function V_{pq} in the smoothness term $E_s(f)$ in equation (6) can be a quadratic function, a delta function, a truncated quadratic function or an even more complicated function. We express the smoothness term $E_s(f)$ in terms of a total variation

function V_{pq} regularized by weights $\lambda(p,q)$ which are calculated using the perceptually uniform CIELab color space. Denote a pixel that falls within a neighbourhood window W centered at pixel p by $q \in W(p)$. The variation term V_{pq} is expressed as follows:

$$V_{pq}\left(f_p, f_q\right) = \lambda\left(p,q\right)\min\left(\left|f_p - f_q\right|^2, V_{\max}\right) \tag{20}$$

where $V_{\max}$ is the maximum upper bound on the smoothness. The upper boundary is introduced so that all the discontinuities in the image have the same reasonable potential instead of having different big values that blows up the function. The regularizing parameter $\lambda\left(p,q\right)$ is given by:

$$\lambda\left(p,q\right) = \exp\left[-\frac{\left\|p-q\right\|^2}{2\sigma_s^2} - \frac{\left\|I_L\left(p\right)-I_L\left(q\right)\right\|^2}{2\sigma_r^2} - \frac{\left\|I_a\left(p\right)-I_a\left(q\right)\right\|^2}{2\sigma_r^2} - \frac{\left\|I_b\left(p\right)-I_b\left(q\right)\right\|^2}{2\sigma_r^2}\right] \tag{21}$$

where I_L, I_a, I_b are the CIELab color space components that best represent human perception of colors among all the available color spaces.

Recent studies have shown that by grouping pixels with similar colors before matching can improve the accuracy of the resulting disparity map. Instead of segmenting images using computational intensive algorithms, this grouping can be simply coded by including bilateral weights, equation (21), in the smoothness term [13]. Introducing such bilateral weights in the smoothness term forces the resulting disparity map to agree with the color discontinuities in the reference image. The final smoothness term is expressed as follows:

$$E_s\left(f\right) = \sum_{p}\sum_{q\in N(p)} \lambda\left(p,q\right) V_{pq}\left(f_p, f_q\right) \tag{22}$$

3.2.2.3 Initial Disparity and Camera Response

The total energy function given in equation (4) is then minimized using the graph cut algorithm [9, 14] to produce the initial disparity estimate. The resulting disparity map contains errors mainly in the over-exposed and under-exposed regions of the images. To obtain estimates for these regions, we calculate two disparity maps. The first has the left image as the reference and the second has the right image as the reference. Then we cross

validate the resulting maps. The pixels in the two disparity maps that match are treated as valid disparities. The remaining pixels are marked as erroneous and are represented by black pixels for further correction. The initial disparity map is formed of the valid disparities and the erroneous black pixels. Figure 3.6 to Figure 3.9 show the initial disparity map obtained after matching the left and right disparities for the images: Cloths, Dolls, Arts and Baby.

The matched pixels in the two disparity maps are considered as valid disparity values. Therefore, only these pixels are used to compute the coefficients in the polynomial model of the camera inverse response function using the algorithm in [5]. The coefficients c_n are found by minimizing the cost function given by equation (12).

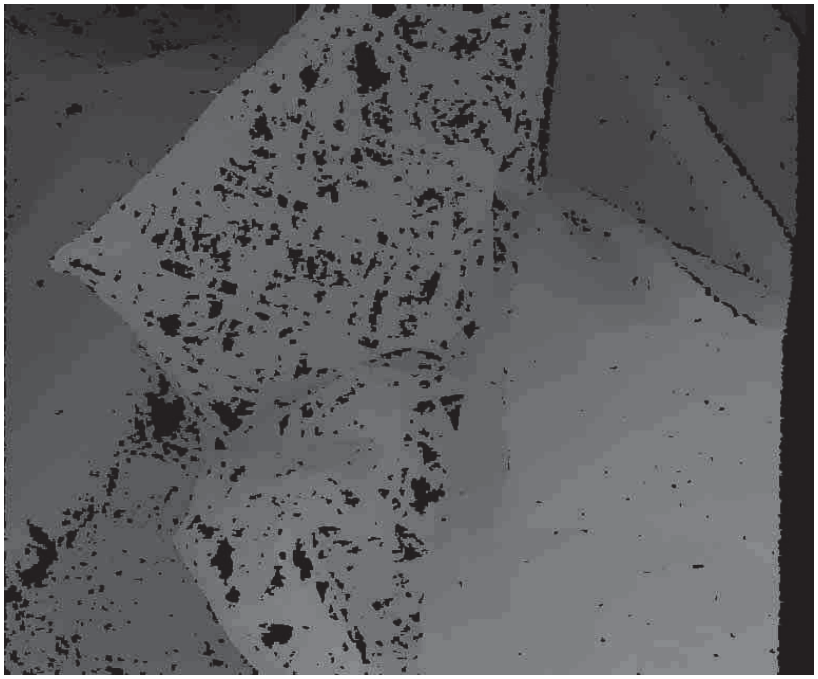


Figure 3.6 Initial disparity map obtained by our algorithm after running step 1: Clothes

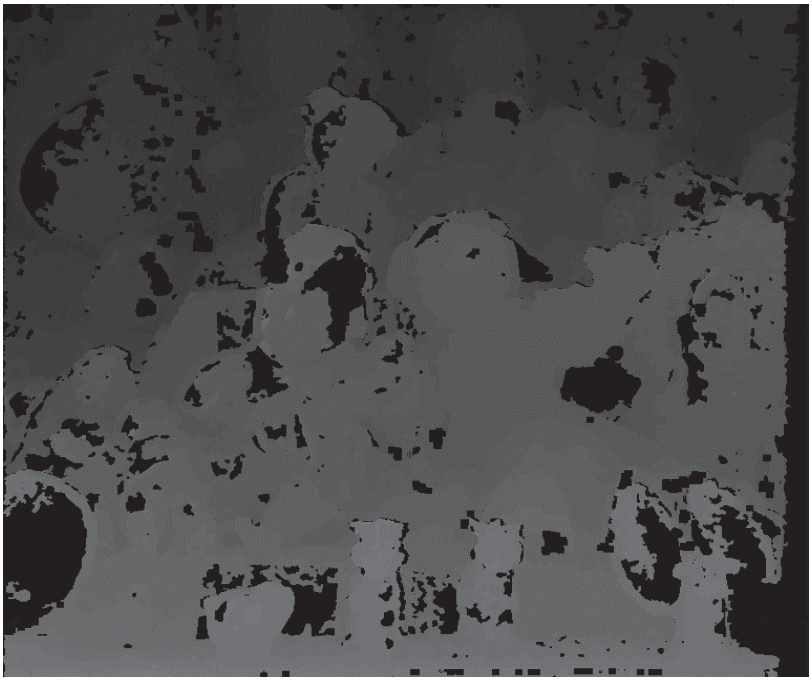


Figure 3.7 Initial disparity map obtained by our algorithm after running step 1: Dolls

Figure 3.8 Initial disparity map obtained by our algorithm after running step 1: Arts

Figure 3.9 Initial disparity map obtained by our algorithm after running step 1: Baby

3.2.3 Refinement

After finding the coefficients of the inverse polynomial camera response function, equation (10), we use this inverse polynomial camera response function to convert the left and right images into their respective radiance maps in radiance space $\tilde{R}$. The radiance values of the brighter input image are also multiplied by the exposure ratio between the two input images so that the radiance of the corresponding pixels in the left and right images has the same value. We use the two resulting radiance maps to correct the erroneous pixels identified in the initial disparity map. This is done using the following interpolation method.

We formulate the interpolation problem as a minimization problem of an energy function defined by equation (4). In this step, we define a different pixel dissimilarity cost $E_d(f)$. In addition, the values of some entries in vector f are already known. The known pixels are the pixels with the disparity values that match each other, obtained as the initial estimates when each of the left and the right LDR images was used as the reference. Therefore, we force the solution of the new energy function to pick up the same value at these pixels. This is achieved by assigning the minimum possible cost 0 to each pixel p at the desired value and a large cost at the disparity values other than this known value. Namely, assume $\tilde{f}_p$ is the valid known disparity at a pixel p, then

$$D_p\left(f_p\right)=\begin{cases}0 & if\left(f_p=\tilde{f}_p\right)\\ K & if\left(f_p\neq\tilde{f}_p\right)\end{cases} \tag{23}$$

where K is a large number.

This means that our refinement step finds the vector f that minimizes the new energy function and this vector has some of its values as already determined. The smoothness term in the energy function ensures that similar disparity values are assigned to pixels representing the same object. Therefore, an accurate disparity value that was determined in the initial

estimate for a pixel is expected to propagate to the neighboring pixels that belong to the same object but had erroneous values in their initial estimates.

For the erroneous pixels in the initial disparity map, instead of using equation (19) as the cost function for stereo matching before, we define the cost function as the sum of two terms: the differences between the radiance of the corresponding left and right pixels and a Hamming distance, i.e.

$$D_p\left(f_p\right)=\left\|\tilde{R}_l\left(p\right)-\tilde{R}_r\left(p+f_p\right)\right\|+C_p\left(f_p,W\left(p\right),\tilde{R}_l,\tilde{R}_r\right) \qquad (24)$$

where $\tilde{R}_l$ is the radiance map of the left LDR image after applying the Census transform [15] over the window $W\left(p\right)$. $\tilde{R}_l$ is the radiance map of the right LDR image after applying the Census transform [16] over the window $W\left(p+f_p\right)$. $C_p\left(f_p,W\left(p\right),\tilde{R}_l,\tilde{R}_r\right)$ is a cost function that calculates the Hamming distance between pixel p in the left radiance map $\tilde{R}_l$ and pixel $p+f_p$ in the right radiance map.

The Census transform is a non-parametric summary of local spatial structure. Let p be a pixel with intensity $I(p)$ and let $N(P)$ be the set of pixels in a square neighborhood of diameter d surrounding pixel p. The census transform $R_t(p)$ maps the pixels in $N(p)$ to a bit string representing the set of neighboring pixels whose intensity is less than $I(p)$, the value of the central pixel p. The value of a neighboring pixel in the bit string is set to 1 if its intensity is less than $I(p)$ and to 0 if its intensity is greater than $I(p)$. Such a transform tolerates the presence of small variations in the values of the corresponding pixels in the left and right radiance estimates. This is because the variations are unlikely to alter the relationship between the surrounding pixels and the central pixel. Detailed explanation of Census transform is covered in the Appendix.

We apply the Census filter to the left and right radiance estimates. Then we add the Hamming distance to the cost function in the error correction step for the following two reasons:

- As a non-parametric local image transformation, the Census filter differs from other operations in that it only responds to the relative ordering of intensities instead of the intensities themselves. This makes it robust to the variations in intensities between the two input images in our setting.
- The speed of the algorithm: after the Cencus filter is applied, the image patches corresponding to the two LDR radiance maps become two strings containing only binary numbers. The correlation between the two patches is calculated from the Hamming distance which is the number of bits the have different values between the two binary strings. Binary operators greatly reduce the computational complexity in determining the correlation between two image patches.

Notice that the new dissimilarity cost function D_p for erroneous pixels is composed of two terms. The first term ensures a smooth transition across object boundaries in the radiance map, while the second term ensures that pixel locations are accurately matched. However, strict disparity matching can cause edge artefacts in the final HDR image which result from occlusions in the stereo setup. Therefore, we enforce smooth transitioning in the radiance map so as to remove any possible artefacts that may arise. Finally, in order to speed up the minimization process, we bound the search range of feasible disparity values by the minimum $f_{v,\min}$ and maximum $f_{v,\max}$ valid disparity values found in the initial disparity estimate $\hat{R}$, such that $f_{v,\min} \le \hat{f} \le f_{v,\max}$.

3.3 Image Synthesis

Once the disparity map and the coefficients c_{ni} in equation (10) are computed, the left and right images can be fused into a single HDR radiance image [3]. This is achieved by the following three steps:

1) we construct the radiance maps of the two input LDR images. Each of the pixel p in the input LDR images is converted to a radiance value that represent the amount of light fallen onto the sensor at pixel p using the inverse camera response function defined by equation (10).
2) We normalize the values in the two radiance maps by dividing the values in the right radiance map (with greater exposure) by the exposure ratio between the two input LDR images. According to the image model defined by equation (8) and (9), the amount of light fallen on the camera sensor depends on both the scene radiance and the exposure value used to capture the image. Therefore, the two radiance maps obtained in the first step differs by a scale equal to the exposure ratio between the two input LDR images.
3) We assign pixels values in the HDR image as a weighted average of the corresponding pixels in the two LDR images:

$$I_{HDR}(p) = \frac{w_l R_l(p) + w_r R_r(p)}{w_l + w_r}$$

In order to determine the weights used to calculate the pixel values in the HDR image, we put great emphasis on the signal-to-noise ratio (SNR). A good estimate of the radiance in the HDR image should maximize both the SNR and the sensitivity to radiance changes. The SNR for the scaled radiance value is given by:

$$SNR = \frac{R(I)}{\sigma_N(I)R'(I)} \tag{25}$$

where, $\sigma_N(I)$ is the standard deviation of the measurement noise and can be assumed to be independent of I in our case. In addition, a large value $R(I)$ indicates that the sensor is set at the right exposure to detect the changes in the scene radiance. Therefore, we define the weighting function used to combine the input radiance maps as

$$w = \frac{R(I)}{R'(I)}$$

This weighting function ensures that for the two corresponding pixels in the input LDR images, the pixel with a greater SNR and sensitivity to radiance change is assigned a greater

weight than the other pixel. As a result, the pixel value that contains clearer scene information and less noise in the input LDR image determines the value of this pixel in the constructed HDR image.

Chapter 4 Experimental Results

4.1 Disparity Map Accuracy

We tested our algorithm using stereo images provided by Middlebury College [16]. The images provided are rectified so that we only need to search along the horizontal line for each point in the image when finding disparity maps. Before we can evaluate the quality of the disparity maps calculated using our algorithms, we have to find the true (ideal) disparity maps for the four scenes, Figure 3.1 to Figure 3.4, as the standard. The reference disparity map for each scene in Figure 3.1 to Figure 3.4 accurately represents the horizontal displacement of each pixel in the higher exposed image using the less exposed image as the reference.

4.1.1 True Disparity Maps

In this paper, we present the results for the scenes Arts, Dolls, Baby and Clothes using input LDR images shown in Figure 3.1 to Figure 3.4. Figure 4.1, Figure 4.3, Figure 4.5 and Figure 4.7 show the reference disparity maps of the scenes Clothes, Dolls, Arts and Baby we used to test the performance of our algorithm. The true disparity maps are calculated using our algorithm when the input LDR images are captured under the same exposure.

4.1.2 Disparity Maps Obtained Using Our Algorithm

The disparity maps for the four scenes obtained by using our algorithm are presented in Figure 4.2, Figure 4.4, Figure 4.6 and Figure 4.8. The input LDR images of Clothes and dolls, shown in Figure 3.1 and Figure 3.2, contain small saturated areas scattered over the images whereas the input LDR images Arts and Baby, shown in Figure 3.3 and Figure 3.4, have large areas of both under-saturated and over-saturated pixels.

In our experiments, the size of the window used by our algorithm is (5×5) pixels. The window size is chosen to be small in order to reduce the amount of computation for the NCC

for every disparity value under consideration. After running the algorithm several times with different parameters, we found the following parameters give optimal results. The standard deviations s and r used in calculating the bilateral weights, equation (16) when estimating the initial disparity map are set to 2.6 and 14.0. The standard deviation s and r used in calculating $\lambda\left(p, q\right)$, equation (21), in the potential function $V_{pq}\left(f_p, f_q\right)$, equation (20) are set to 2.6 and 16.0.

Comparing the disparity maps obtained by our algorithm, shown in Figure 4.2, Figure 4.4, Figure 4.6 and Figure 4.8 with the true disparity maps, shown in Figure 4.1, Figure 4.3, Figure 4.5 and Figure 4.7, we obtain the following conclusions:

- If the input images contain scattered saturated regions of small areas, such as in the case of the scenes Clothes and Dolls, shown in Figure 3.1 and Figure 3.2, the disparity maps obtained by our algorithm, shown in Figure 4.2 and Figure 4.4, follow closely the true disparity maps, shown in Figure 4.1 and Figure 4.3.
- If the input LDR images have large saturated regions such as the case of the scenes Arts and Baby, shown in Figure 3.3 and Figure 3.4, the final disparity maps obtained using our algorithm, shown in Figure 4.6 and Figure 4.8 has discernible difference from their true disparity maps, shown in Figure 4.5 and Figure 4.7.
- The error pixel values in the disparity maps obtained by our algorithm have little effect on the constructed HDR images for Cloths, Dolls, Arts and Baby, shown in Figure 4.9 to Figure 4.12. This is because of the first term in the cost function (24) in our refinement step ensures that the pixels in one input LDR radiance map are mapped to the pixels in the other LDR radiance map with similar radiance. This is the case even for pixels with erroneous disparity values.
- The constructed HDR images show in Figure 4.9 to Figure 4.12 have a higher dynamic range than either of the input LDR images for the scenes in Figure 3.1 to Figure 3.4. The constructed HDR images in Figure 4.9 to Figure 4.12 clearly display the details in both dark and bright regions of the scenes in Figure 3.1 to Figure 3.4.

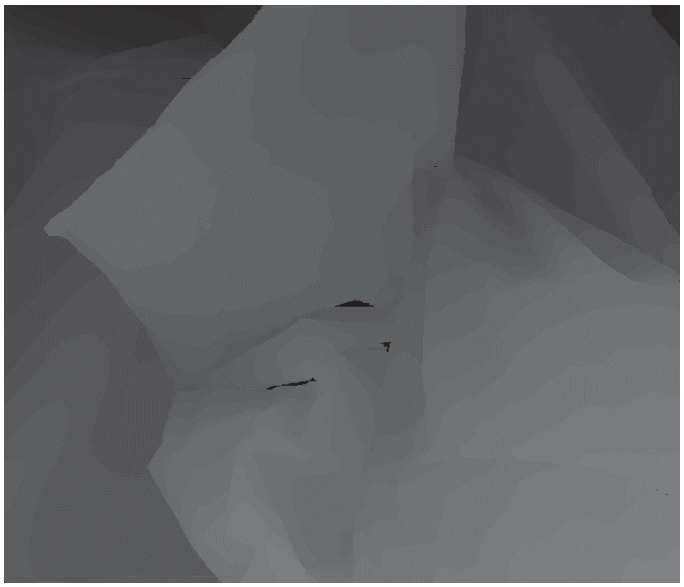


Figure 4.1 Reference disparity map obtained using input images of the same exposure: Clothes

Figure 4.2 The final disparity map obtained using our algorithm: Clothes. The values at the erroneous pixels in the initial disparity map are successfully obtained by the refinement step. This is because the saturated regions in the input images are small and scattered.

Figure 4.3 Reference disparity map obtained using input images of the same exposure: Dolls

Figure 4.4 The final disparity map obtained using our algorithm: Dolls. Most of the pixels have the correct disparity value except the head of the left doll at the center. This is because the area is over exposed in the input images, causing the interpolation process in the refinement step to take the wrong value.

Figure 4.5 Reference disparity map obtained using input images of the same exposure: Arts

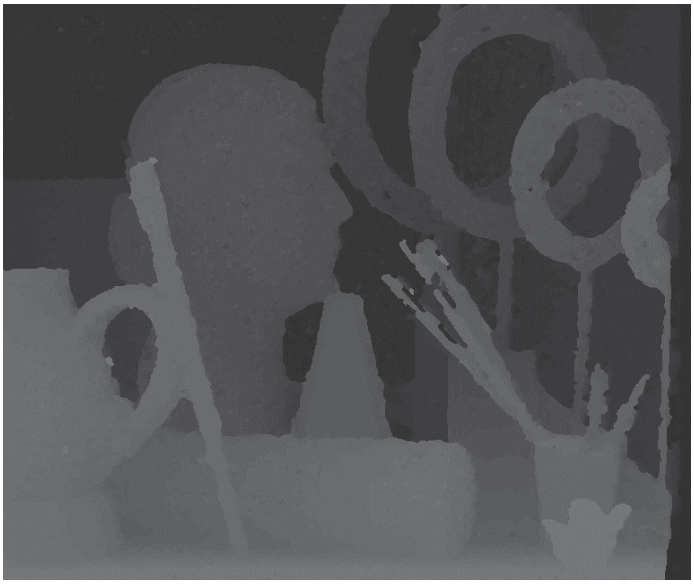


Figure 4.6 The final disparity map obtained using our algorithm: Arts. The disparity map is generally accurate and smooth except at the edges of different objects and the pencils. This is because it is very difficult to completely remove the fattening effect caused by using a window based cost function in stereo matching.

Figure 4.7 Reference disparity map obtained using input images of the same exposure: Baby

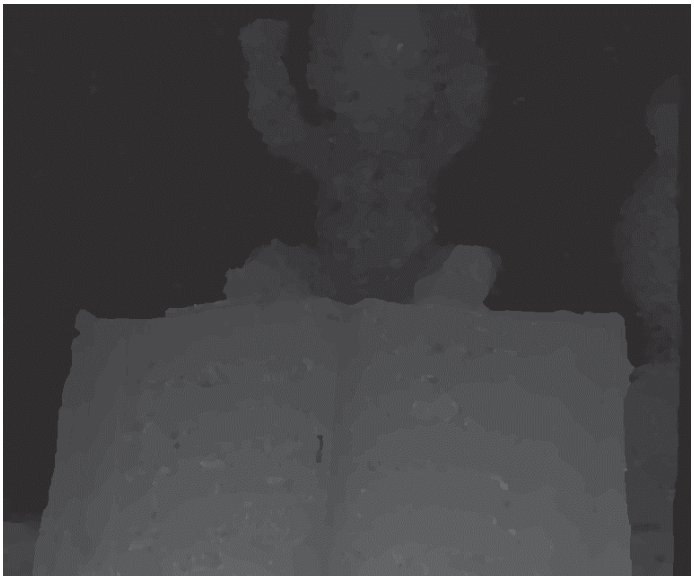


Figure 4.8 The final disparity map obtained using our algorithm: Baby. The disparity map is not as smooth as the previous three maps. This is because of the large areas of under-saturated and over-saturated regions in the input images. However, the refinement step has still assigned correct values close to the true disparity values of the erroneous pixels in the initial disparity map, Fig. 3.13.

Figure 4.9 The tone-mapped reconstructed HDR image of Clothes. The image shows the details in the folds which are under-saturated in one input image. The image also shows the textures of the central piece of the cloth which are over exposed in the other input image.

Figure 4.10 The tone-mapped reconstructed HDR image of Dolls. The picture shows the detail of both dark and bright cloths the dolls are wearing

Figure 4.11 The tone-mapped reconstructed HDR image of Arts. The statue at the center and the pot on the left of the image are not saturated. It also displays the details in the dark region of the paint in the background wall.

Figure 4.12 The tone-mapped reconstructed HDR image of Baby. The picture displays both the words and equations in the book and the details of the maps on the wall.

4.1.3 Disparity Maps Obtained Using ANCC and Multi-view Multi-exposure Algorithms

Figure 4.13 to Figure 4.16 show the disparity maps obtained using ANCC [10]. Figure 4.17 to Figure 4.20 show the disparity maps obtained using the multi-view multi-exposure algorithm [7]. In this paper, we evaluate the performance of the multi-view multi-exposure algorithm by using three LDR images captured under different exposures as the input for the scenes Clothes, Dolls, Arts and Baby.

The disparity maps of Clothes and Dolls have fewer errors because the saturated regions in these images are scattered small patches. The smoothing term in the energy function is successful in propagating the correct disparity values to the saturated pixels from their surrounding unsaturated pixels. However, the disparity maps of Arts and Baby contain significant errors because both matching cost functions in [7] and [10] do not have any term in the energy function designed specifically to propagate the correct disparity values for pixels in the saturated areas from the disparity values of the surrounding unsaturated pixels in the input LDR images. As a result, when the input LDR images have large saturated regions, instead of assigning correct disparity values to saturated pixels, the smoothing term tends to propagate the erroneous disparity values and thus assign wrong disparity values at pixels surrounding the saturated regions.

Compared to the disparity maps for Arts obtained using the algorithms proposed in [7] and [10], the disparity map we computed has fewer errors and better smoothness. Our calculated disparity maps compared to those obtained using algorithms in [7] and [10] are presented in Table 4.1 to Table 4.3. The root mean square error (RMSE) of invalid pixels and the numbers of error pixels in the disparity maps obtained by our algorithm are significantly less than those obtained using algorithms in [7] and [10]. The percentage of error pixels in the disparity maps obtained by our algorithm is on average 16% less than that in [7] and 20% less than that in [10].

Figure 4.13 The disparity maps of Clothes obtained using ANCC [10]

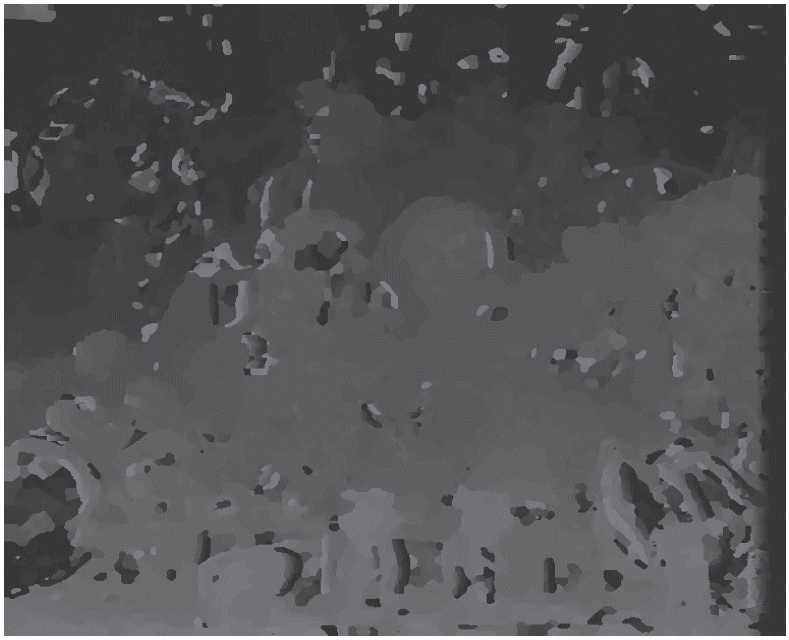


Figure 4.14 The disparity maps of Dolls obtained using ANCC [10]

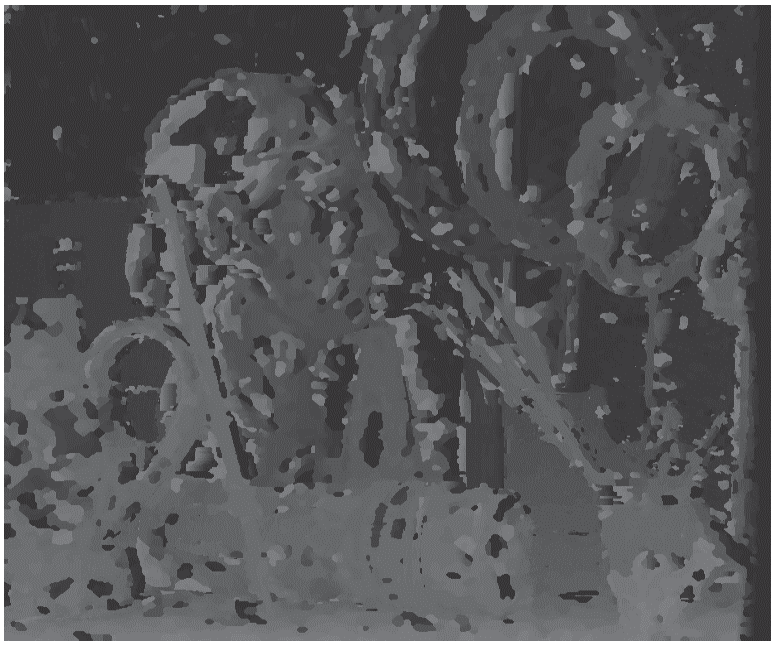


Figure 4.15 The disparity maps of Arts obtained using ANCC [10]

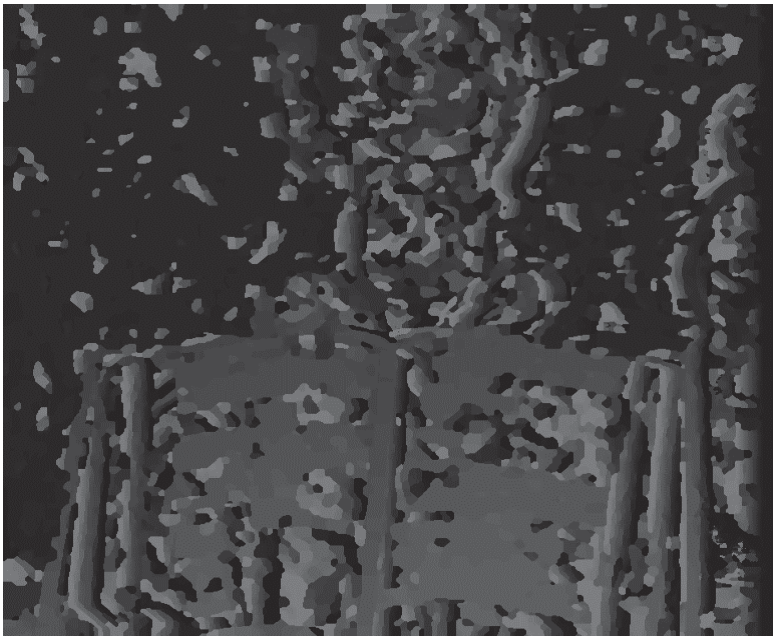


Figure 4.16 The disparity maps of Baby obtained using ANCC [10]

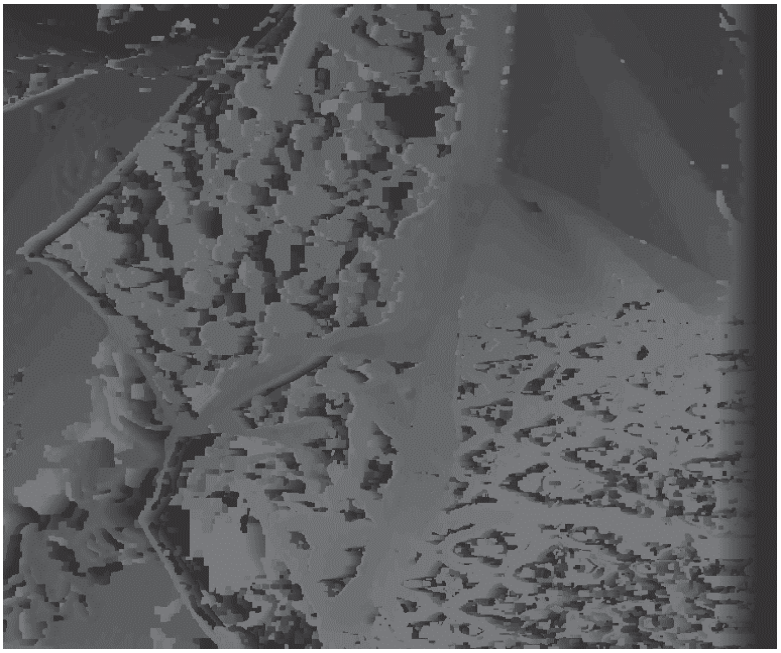


Figure 4.17 The disparity maps of Clothes obtained using multi-view multi-exposure algorithm [7]

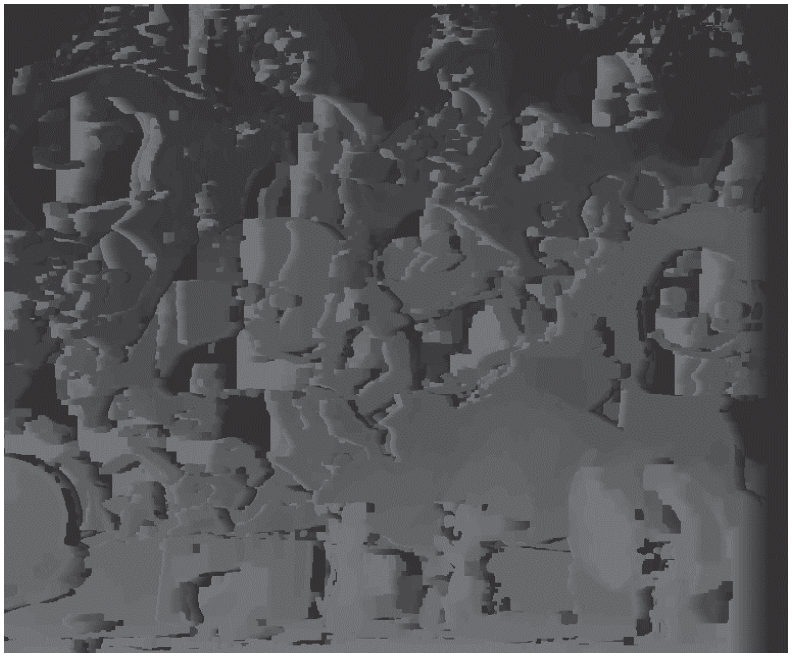


Figure 4.18 The disparity maps of Dolls obtained using multi-view multi-exposure algorithm [7]

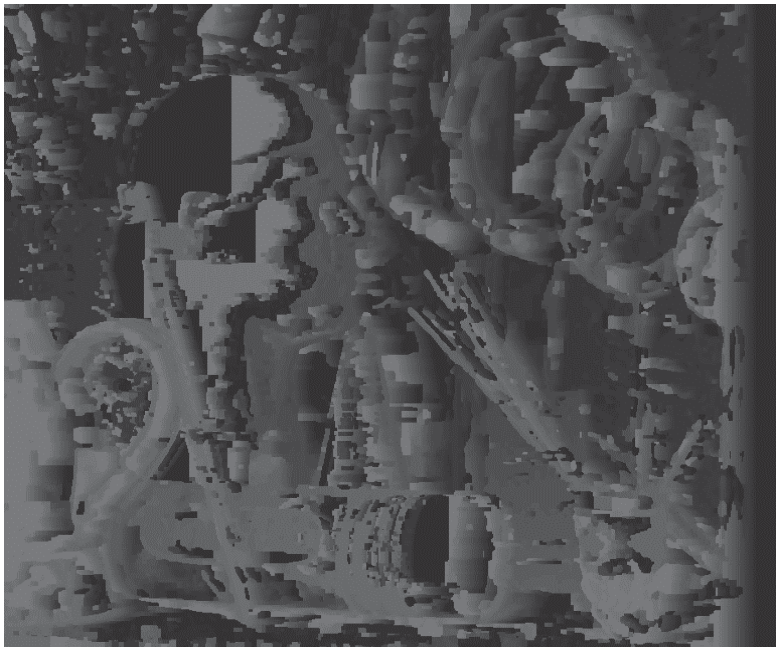


Figure 4.19 The disparity maps of Arts obtained using multi-view multi-exposure algorithm [7]

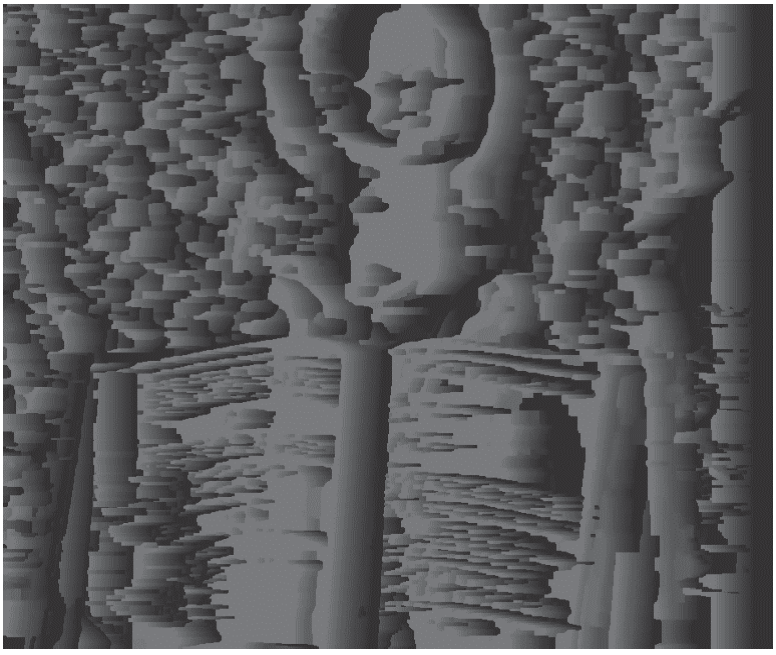


Figure 4.20 The disparity maps of Baby obtained using multi-view multi-exposure algorithm [7]

Table 4.1 The root mean square error and percentage of invalid pixels in the final disparity maps using our algorithm

	Exposure Ratio	RMSE	Error Percentage
Clothes	4	0.9934	8.23
	16	0.976	8.82
Dolls	4	0.8454	4.77
	16	0.8591	5.58
Arts	4	1.5459	7.43
	16	1.1556	8.15
Baby	4	1.432	9.42
	16	1.4642	10.13

Table 4.2 The root mean square error and percentage of invalid pixels in the final disparity map using ANCC as the matching cost function [10]

	Exposure Ratio	RMSE	Error Percentage
Clothes	4	20.218	29.0
	16	21.268	29.2
Dolls	4	15.689	14.4
	16	16.512	15.3
Arts	4	10.216	9.35
	16	11.653	9.72
Baby	4	28.265	38.0
	16	28.044	37.6

Table 4.3 The root mean square error and percentage of invalid pixels in the final disparity maps using multi-view multi-exposure algorithm[7]

	Exposure Ratio	RMSE	Error Percentage
Clothes	4	25.705	53.3
	16	25.798	53.7
Dolls	4	19.983	32.8
	16	20.385	33.7
Arts	4	20.665	31.6
	16	20.989	32.1
Baby	4	46.767	88.9
	16	48.246	90.3

4.2 Dynamic Range

The quality of an HDR image is determined not only by the accuracy of the pixel values, but also by the range of the scene radiance it can capture. The dynamic range of constructed HDR images depends on the number of input LDR images and the exposure values of the input images. It is bounded by the lowest radiance value in the image with the shortest exposure and the highest radiance value in the image with the greatest exposure. The larger the number of the input LDR images and the greater the difference among their exposure values, the larger the dynamic range of the scene radiance that can be captured by the reconstructed HDR image.

The computational complexity however, increases exponentially with the increase in the input LDR images. Reducing the computational complexity is crucial to transfer a technology from the lab to the market. This is also the reason why we limit our input LDR images to two. In addition, Figure 4.2, Figure 4.4, Figure 4.6 and Figure 4.8 show that a large exposure ratio between the two input LDR images results in a decrease in the accuracy of the disparity map generated by our algorithm. This is because when the exposure ratio between the two input images is large, the amount of the overlapping radiance ranges between the two input LDR images decreases. This results in the deviation of the calculated camera inverse response function from its true value increases. Therefore, the exposure values of the input LDR images should be chosen carefully so that the dynamic range of the reconstructed HDR image is maximized and the error in the calculated camera inverse response function is minimized.

One way to find the optimal exposure values for a scene is to minimize an objective function that is based on the derivative of the camera response function [17]. In order to reduce the amount of calculation during the runtime, the exposure ratios of the different kinds of camera response function can be pre-calculated and stored in a table. For the cameras whose inverse response functions are modeled by a Gamma curve, equation (3), the optimal ratio of the exposure values is 1:3.094 [17].

Chapter 5 Conclusion

In this thesis, we address the problem of obtaining a HDR image using two LDR images. The two images should be captured at the same instance by the two cameras, which should be placed close to each other. The images should have different exposures. One image captured under lower exposure registers the bright part of the scene, i.e. it does not have over-saturated but under-saturated areas. The other image captured under higher exposure register the details in the dark part of the scene, i.e. it does not have under-saturated but over-saturated areas. We presented a novel algorithm that calculates the disparity map of two differently exposed LDR images to generate the HDR images. Compared to existing methods, our algorithm has several advantages.

In many stereo matching algorithms, such as the ANCC [10] and the Multi-view, Multi-exposure [7] algorithms we discussed in detail how the three R, G and B channels are used in the matching cost functions to find the disparity map of the input LDR images. These algorithms are developed for the case when the multiple input LDR images are taken by multiple cameras. Under such set up, the cameras at the two ends are not placed in proximity. The angle between the light and the norm of the surface, i.e. the factor $\rho\left(p\right)$ in equation (3), affects the amount of light fallen on the sensor of cameras placed further apart from each other. This means that the amount of chrominance form one channel fallen on one of the input LDR images are not the same as that fallen on another input LDR images. As a result, most stereo matching algorithm such as the ones proposed in [7] and [10] cannot use the simplified camera model as is the case in our algorithm (equation (8) and (9)). These algorithms have to include both the chrominance and luminance information when computing the matching cost for every disparity value under consideration.

Our set up enable our algorithm to simplify the camera model and thus uses only the information in the luminance channel of input LDR images to evaluate the pixel dissimilarities, equation (15). As the computation of the cost function is done for one channel

in our case compared to three channels in other stereo matching algorithms such as [7] and [10], our algorithm is less computationally intensive.

Our algorithm also designs a unique refinement process to cope with changes in the exposures under which the two input LDR images are captured. This refine process also copes with the existence of saturated regions in the input LDR images. Most of the available algorithms such as [10] just run the matching process once. These algorithms focus on designing a single cost function which can tolerate radiometric changes between and the existence of saturated areas in the input stereo images. Some algorithms such as [7] have a refinement step. Their refinement step is simply a rerun, i.e. it applies the first stereo matching process again either once or several time, using the same matching cost using radiance maps of the input LDR images. Such refinement step increases the accuracy of the disparity map because there are less radiometric changes among the radiance maps of the input LR images. However, the refinement step does not increase the probability of propagating the correct disparity values of unsaturated pixels to the nearby saturated pixels in the input LDR images.

Our algorithm proposed a refinement step which differs from the refinement steps in other state-of-the-art algorithms in the following steps:

- We take the pixels with accurate disparity values found in the initial estimate of the disparity maps into consideration. This is done by pre-setting the disparity values of these pixels in the solution vector f when solving the energy function. As a result, we ensure that the accurate disparity values of unsaturated pixels are assigned to the nearby saturated pixels belonging to the same object.
- We take the luminance discontinuities in the input LDR images into consideration. Include this information in the smoothness term in the energy function also increase the accuracy and sharpness of the edges of objects in the image.
- We use a different matching cost function in energy function to measure the dissimilarities between two pixels in the radiance maps of the input images. We apply Census filter first to the radiance maps and then calculated the Hamming

distance between corresponding pixels. This cost function is much less computationally intensive than NCC.

As a result, the disparity maps calculated using our algorithms show less errors and better smoothness than other state-of-art algorithm for input stereo images with saturations and large radiometric changes.

Our algorithm can be used with fast motion scenes since the proposed setup captures images with different exposures at the same instance. Therefore, no temporal adjustment is required. Every pair of images represents the same scene and is used to generate the HDR image for that instance. For constructing HDR videos, we can use the same set up, i.e. two LDR cameras placed next to each other to record the same video. Then the corresponding frames of each video can be paired up. Each pair can be treated separately as one pair of stereo LDR images. Our algorithm can then be used to process these pairs of images to form the HDR images. Finally, the constructed HDR images from each pair for input LDR frames from the LDR videos can be combined together to form the HDR video.

In the future, we would like to explore the field of constructing 3D HDR images or videos since our algorithm is capable of computing accurate disparity map for a variety of scenes such as scenes with large radiometric change, existence of saturated areas and motions. There are many interesting and useful work and projects that can continue from this work and that we would like to explore in the future. These include:

- Extension of our algorithm to the field of constructing HDR videos. This will be done using the framework outlined in the previous paragraph to obtain each frame of the HDR video. One of the problems we might want to address is the flickering problem. The defects in constructed HDR frame may results in shifts of corresponding pixels between consecutive frames. Special steps may be required to make the video appear smooth.
- We would also like to find the optimal exposure ratio which results in HDR images with the largest possible dynamic range, following the direction proposed in [17]. The algorithm in [17] takes several predefined camera response functions. These are the response functions of several popular LDR cameras in the market.

Such limitation prevents it from being accurate for all the cameras in the market. We would like to modify the algorithm so that it can calculated the camera response function dynamically and calculates the optimal exposure ratio specifically for the camera being used.

Bibliography

[1] F. Banterle, P. Ledda, K. Debattista, A. Chalmers, and M. Bloj, “A framework for inverse tone mapping,” The visual Computer: International Journal of Computer Graphics, vol. 23, pp. 467–478, 2007.

[2] P. E. Debevec and J. Malik, “Recovering high dynamic range radiance maps from photographs,” in Proceedings of SIGGRAPH 97, August 1997, pp. 369–378.

[3] T. Mitsunaga and S. K. Nayar, “Radiometric self calibration,” in IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, 1999, pp. 374–380.

[4] R. A. Varkonyi-Koczy, R. Hashimoto, S. Balogh, and S. Y., “Gradient based synthesized multiple exposure time HDR image,” in Instrumentation and Measurement Technology Conference Proceedings, May 2007, pp. 1–6.

[5] T. Mitsunaga and S. K. Nayar, “High dynamic range imaging: spatially varying pixel exposures,” ACM Transaction On Graphics, vol. 3, pp. 267–276, 2002.

[6] S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, “High dynamic range video,” in International Conference on Computer Graphics and Interactive Techniques, ACM SIGGRAPH 2003, 2003, pp. 319–325.

[7] B. Troccoli, S. B. Kang, and S. Seitz, “Multi-view multi-exposure stereo,” in Third International symposium on 3D Data Processing, Visualization, and Transmission, June 2006, pp. 861–868.

[8] Y. Boykov, O. Veksler, and R. Zabih, “Fast approximate energy minimization via graph cuts,” IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 23, no. 11, pp. 1222–1239, 2001.

[9] H. Hirschnuller and D. Scharstein, “Evaluation of cost function for stereo matching,” in Proceedings of Computer Vision and Pattern Recognition, 2007.

[10] Y. S. Heo, K. M. Lee, and S. U. Lee, “Illumination and camera invariant stereo matching,” in IEEE Conference on Computer Vision and Pattern Recognition, 2008.

[11] S. Z. Li, “Markov random field models in computer vision,” in ECCV ’94: Proceedings of the Third European Conference-Volume II on Computer Vision. London, UK: Springer-Verlag, 1994, pp. 361–370.

[12] C. Tomasi and R. Manduchi, “Bilateral filtering for gray and color images,” in Proceedings of International Conference on Computer Vision, 1998.

[13] R. Brockers, “Cooperative stereo matching with color-based adaptive local support,” in Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns, vol. 5702, 2009.

[14] http://www.adastral.ucl.ac.uk/ vladkolm/software.html.

[15] R. Zabih and J. Woodfill, “Non-parametric local transforms for computing visual correspondance,” Proceedings of ECCV, pp. 131–158, 1994.

[16] http://cat.middlebury.edu/stereo/data.html.

[17] S. P. P. D. E. Reinhard, G. Ward, “High Dynamic Range Imaging: Acquisition, Display and Image-Based Lighting,” Morgan Kaufmann, 2005.

[18] R. Fattal, D. Lischinski, M. Werman, “Gradient domain high dynamic range compression,” Proceedings of the 29th annual Conference on Computer Graphics and Interactive Techniques, Vol. 21, Issue 3, 2002

[19] R. A. Street, “High dynamic range segmented pixel sensor array”, U.S. Patent 5789737, 1998

[20] M. Murakoshi, “Charge coupling image pickup device,” Japanese Patent 59-217358, 1994

[21] M. D. Grossberg and S. K. Nayr. “Determining the camera response from images: What is knowable?” IEEE Transaction Pattern Analysis and Machine Intelligence, Vol 25, No. 11, 2003

[22] S. J. Kim, M. Pollefeys, “Robust Radiometric Calibration and Vignetting Correction.” IEEE Transaction Pattern Analysis and Machine Intelligence, Vol. 30, Issue 4, pp 562-576 , 2008

[23] M. D. Grossberg and S. K. Nayar, “What can be known about the radiometric response from images?” Proceedings of the 7th European Conference on Computer Vision-Part IV, pages 189–205, London, UK, 2002.

[24] T. Kanade and M. Okutomi. “A stereo matching algorithm with an adaptive window: Theory and experiment.” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.12, Issue 9, 1994.

Appendices

Appendix A Census Transform

The Census transform is one form of non-parametric local transforms for computing vision correspondence. Non-parametric local transforms rely on the relative order of the neighboring intensity values to the central pixel value, thus do not rely on the local intensity values themselves as in the case of parametric local transforms such as NCC which rely on the exact intensity values of the pixels in the input images. Therefore, non-parametric local transforms have the advantage of tolerating more significant differences between the corresponding pixel values in the left and right stereo images. It also improves the accuracy of pixel values near object boundaries in the disparity map.

Similar to other local transforms, the Census transform is based on a window and computes the correspondence between pixels in the left and right stereo images. Let p be a pixel, $I(p)$ be its intensity value and $N(p)$ be the neighboring pixels q within a $d \times d$ square window surrounding pixel p. The Census transform only depends on the sign between the intensities at pixels p and q, i.e. $I(p)$ and $I(q)$. The sign is determined as:

$$\xi(p, q) = \begin{cases} 0 & I(p) \leq I(q) \\ 1 & I(p) > I(q) \end{cases} \tag{26}$$

The Census transform depends solely on the set of ordered pairs:

$$\mathrm{E}(p) = \bigcup_{q \in N(p)} (q, \xi(p, q)) \tag{27}$$

where $\mathrm{E}(p)$, the transform of p, is a binary vector that represents the bit string of the set of neighboring pixels q whose pixel values less than $I(p)$.

To compare the similarity of two binary vectors (each representing a pixel belonging to one of two Census transformed images), the Hamming distance is used. The Hamming distance $d(x, y)$ between two vectors $\underline{x}$ and $\underline{y}$ is defined as the number of bits in which they differ. For example:

$$d(00111, 11001) = 4$$

And

$$d(0122, 1220) = 3$$

The correspondence between two pixels (one pixel belonging to the left image and the other belonging to the right image) is computed by minimizing the Hamming distance after applying the Census transform to the two images.

Appendix B Code

StereoMatching.cpp:

```
#include "stdafx.h"

////////////////////////////////////////////////////////////////////////////
// Example illustrating the use of GCoptimization.cpp
//
////////////////////////////////////////////////////////////////////////////
//
//  Optimization problem:
//  is a set of sites (pixels) of width 10 and hight 5. Thus number of pixels is 50
//  grid neighborhood: each pixel has its left, right, up, and bottom pixels as neighbors
//  7 labels
//  Data costs: D(pixel,label) = 0 if pixel < 25 and label = 0
//          : D(pixel,label) = 10 if pixel < 25 and label is not  0
//          : D(pixel,label) = 0 if pixel >= 25 and label = 5
//          : D(pixel,label) = 10 if pixel >= 25 and label is not  5
// Smoothness costs: V(p1,p2,l1,l2) = min( (l1-l2)*(l1-l2) , 4 )
// Below in the main program, we illustrate different ways of setting data and smoothness costs
// that our interface allow and solve this optimizaiton problem

// For most of the examples, we use no spatially varying pixel dependent terms.
// For some examples, to demonstrate spatially varying terms we use
// V(p1,p2,l1,l2) = w_{p1,p2}*[min((l1-l2)*(l1-l2),4)], with
// w_{p1,p2} = p1+p2 if |p1-p2| == 1 and w_{p1,p2} = p1*p2 if |p1-p2| is not 1

#include <sstream>
#include "LocalCost.h"
#include "ImageIO.h"
#include "PostProc.h"

using namespace std;

int *resultl, *resultr;
LocalCost *lc = new LocalCost();

////////////////////////////////////////////////////////////////////////////
// smoothness and data costs are set up one by one, individually
// grid neighborhood structure is assumed
```

```
////////////////////////////////////////////////////////////////////////////
int factor = 6;
int smoothFn(int p1, int p2, int l1, int l2)
{
	int w = 768;
	int h = 768;
	int col1 = p1 % w;
	int row1 = (p1-col1)/w;
	int col2 = p2 % w;
	int row2 = (p2-col2)/w;
	double power = 0;

	double exp1 = 0;
	double exp2 = 0;

	exp1 = ((row1-row2)*(row1-row2)+(col1-col2)*(col1-col2))/2.04/2.04;

	if ((col2+p2) < w && (col2+p2) >= 0 && (col1+p1) < w && (col1+p1) >= 0 && (p2+l1-l2) < w
&& (p2+l1-l2) >= 0)
	{
		exp2 = pow((::lc->images.left[p1]-::lc->images.left[p2+l1-l2]), 2.0);
		//exp2 = pow((::lc->images.left[p1]-::lc->images.left[p2]), 2.0);
		exp2 = exp2/16.0/16.0;
		double cost = (l1-l2)*(l1-l2) <= 5  ? (l1-l2)*(l1-l2):5;
		return (int)(cost*factor*exp(-0.5*(exp1+exp2)));
	}
	else
	{
		int cost = (l1-l2)*(l1-l2) <= 5  ? (l1-l2)*(l1-l2):5;
		return cost*factor;
	}
}

void GridGraph_Individually(int width,int height,int num_pixels,int num_labels, int reverse, int
window, int lo)
{
	try{
		GCoptimizationGeneralGraph *gc = new
GCoptimizationGeneralGraph(num_pixels,num_labels);

		// first set up data costs individually
		for (int i = 0; i <height; i++ )
```

```
		{
			for (int j = 0; j < width; j++)
			{
				for (int l = 0; l < num_labels; l++ )
				{
					if(reverse == 0)
						gc->setDataCost(i*width+j,l,lc-
>dataCoFn_ncc(i*width+j, l, window, width, height, lo));
					else
						gc->setDataCost(i*width+j,l,lc-
>dataCoFn_ncc(i*width+j, -l, window, width, height, lo));
				}
			}
		}

		// now set up a grid neighborhood system
		// first set up horizontal neighbors
		for (int y = 0; y < height; y++ )
			for (int  x = 1; x < width; x++ )
			{
				gc->setNeighbors(x+y*width,x-1+y*width);
				//gc->setNeighbors(x+y*width,x-2+y*width);
				if((y-1)>=0)
				{
					gc->setNeighbors(x+y*width,x-1+(y-1)*width);
					//gc->setNeighbors(x+y*width,x-2+(y-1)*width);
				}
				if((y+1)< height)
				{
					gc->setNeighbors(x+y*width,x-1+(y+1)*width);
					//gc->setNeighbors(x+y*width,x-2+(y+1)*width);

				}
				/*if((y-2)>=0)
				{
					gc->setNeighbors(x+y*width,x-1+(y-2)*width);
					gc->setNeighbors(x+y*width,x-2+(y-2)*width);
				}
				if((y+2)< height)s
				{
					gc->setNeighbors(x+y*width,x-1+(y+2)*width);
					gc->setNeighbors(x+y*width,x-2+(y+2)*width);
```

```
                        }*/
                }

                // next set up vertical neighbors
                for (int y = 1; y < height; y++ )
                {
                        for (int  x = 0; x < width; x++ )
                        {
                                gc->setNeighbors(x+y*width,x+(y-1)*width);
                                //gc->setNeighbors(x+y*width,x+(y-2)*width);
                        }
                }

                // next set up smoothness costs individually
                /*for ( int l1 = 0; l1 < num_labels; l1++ )
                        for (int l2 = 0; l2 < num_labels; l2++ ){
                                int cost = (l1-l2)*(l1-l2) <= 5  ? (l1-l2)*(l1-l2):5;
                                gc->setSmoothCost(l1,l2,cost*10);
                        }
                */
                gc->setSmoothCost(&smoothFn);
                printf("\nBefore optimization energy is %d",gc->compute_energy());
                gc->expansion(3);// run expansion for 2 iterations. For swap use gc->swap(num_iterations)
                //gc->swap(5);
                printf("\nAfter optimization energy is %d\n",gc->compute_energy());

                for ( int  i = 0; i < num_pixels; i++ )
                {
                        if (reverse == 0)
                                resultl[i] = gc->whatLabel(i);
                        else
                                resultr[i] = gc->whatLabel(i);
                }
                delete gc;
        }
        catch (GCException e){
                e.Report();
        }
}

int main(int argc, char **argv)
```

```
{
	int num_labels = 0;
	char *leftImageFileName = "";
	char *rightImageFileName = "";
	char *leftIntermediateDispFileName = "";
	char *rightIntermediateDispFileName = "";
	char *leftDispFileName = "";
	char *rightDispFileName = "";
	int window = 3;
	int lo = 8;


	int width = 768;
	int height = 768;

	try
	{
		leftImageFileName = argv[1];
		rightImageFileName = argv[2];
		leftIntermediateDispFileName = argv[3];
		rightIntermediateDispFileName = argv[4];
		leftDispFileName = argv[5];
		rightDispFileName = argv[6];
		window = atoi(argv[7]);
		num_labels = atoi(argv[8]);
		lo = atoi(argv[9]);
	}
	catch (char *e)
	{
		//ask user for the inputs if it's not provided
		cout << "This program takes in the intensity values of left and right images and outputs the disparity maps\n.";
		cout << "File name of left image (.txt file): ";
		cin >> leftImageFileName;
		cout << "File name of right image (.txt file): ";
		cin >> rightImageFileName;
		cout << "Please specify the file which stores the result of initial matching from left image view: ";
		cin >> leftIntermediateDispFileName;
		cout << "Please specify the file which stores the result of initial matching from right image view: ";
		cin >> rightIntermediateDispFileName;
```

```
		cout << "Please specify the file which stores the final disparity map from left image view: ";
		cin >> leftDispFileName;
		cout << "Please specify the file wich stores the final disparity map from right image view: ";
		cin >> rightDispFileName;
		cout << "Window size (recommended value: 3): ";
		cin >> window;
		cout << "Number of labels considered in graph cut (This places a cap number to reduce the amount of computation): ";
		cin >> num_labels;
		cout << "Minimum label value accepted in the finally disparity map (This is used in the tuning step for more accurate result, recommended value: 8-10): ";
		cin >> lo;
		factor = 6;
	}

	int num_pixels = width*height;

	//factor = atoi(argv[10]);

	resultl = new int[num_pixels];   // stores result of optimization
	resultr = new int[num_pixels];

	// Read image files
	ImageIO *imgio = new ImageIO(height, width);
	imgio->openImage(leftImageFileName, rightImageFileName);

	// smoothness and data costs are set up one by one, individually
	GridGraph_Individually(width,height,num_pixels,num_labels, 0, window, lo);
	imgio->writeData(leftIntermediateDispFileName, resultl, num_pixels);

	imgio->swapImage();
	GridGraph_Individually(width,height,num_pixels,num_labels, 1, window, lo);
	imgio->writeData(rightIntermediateDispFileName, resultr, num_pixels);

	printf("\nFinished %d (%d) clock per sec %d\n",clock()/CLOCKS_PER_SEC,clock(),CLOCKS_PER_SEC);

	int *disp = new int[num_pixels];
	PostProc *pproc = new PostProc(width, height, num_labels);
	//pproc->txtRead("disp_books_f11.txt", resultl, num_pixels);
```

```
        //pproc->txtRead("disp_books_r11.txt", resultr, num_pixels);
        pproc->MatchLeftRightDisp(resultl, resultr, disp);
        pproc->RemoveSmallAreas(num_labels, leftDispFileName, disp);
        pproc->MatchRightLeftDisp(resultl, resultr, disp);
        pproc->RemoveSmallAreas(num_labels, rightDispFileName, disp);

        //clean the memory
        delete imgio;
        delete [] resultl;
        delete [] resultr;
        delete pproc;
        delete lc;
        delete [] disp;
        printf("Finish.....\n");
        return 0;
}
```

LocalCost.h:

```
#ifndef __LOCALCOST_H__
#define __LOCALCOST_H__

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <iostream>
#include <fstream>

#include "GCoptimization.h"

struct Images
{
	double left[768*768];
	double right[768*768];
};
class LocalCost
{
public:
	LocalCost(void);
	virtual ~LocalCost(void);
	struct Images images;
	int dataCoFn_ncc(int p, int l, int window, int width, int height, int lo);
	int dataCoFn_Census(int p, int l, int window, int width, int height, int lo);

private:
	void subAvg(int width, int height);
	double weightingFn(int colt, int rowt, int colp, int rowp, int width, int height, int left);
	double normConst(double *weights, int size);
	int ncc(int p, int l, int row, int col, int width, int height, double *weights_l, double *weights_r, double leftAvg, double rightAvg, int offset);
};

#endif
```

LocalCost.cpp:

```
#include "stdafx.h"
#include "LocalCost.h"

LocalCost::LocalCost(){}
LocalCost::~LocalCost(){}

void LocalCost::subAvg(int width, int height)
{
	double suml = 0.0;
	double sumr = 0.0;
	int num_pixels = width*height;
	for (int i = 0; i < num_pixels; i++)
	{
		suml = suml + images.left[i];
		sumr = sumr + images.right[i];
	}
	suml = suml / num_pixels;
	sumr = sumr / num_pixels;
	for(int i = 0; i < num_pixels; i++)
	{
		images.left[i] = images.left[i] - suml;
		images.right[i] = images.right[i] - sumr;
	}
}

double LocalCost::weightingFn(int colt, int rowt, int colp, int rowp, int width, int height, int left)
{
	double sigmad = 14.0;
	double sigmas = 3.8;
	double weight = 0.0;

	int t = rowt*width + colt;
	int p = rowp*width + colp;

	if (colt >= 0 && colt < width && rowt >= 0 && rowt < height)
	{
		weight = -(double)((colt-colp)*(colt-colp) + (rowt-rowp)*(rowt-rowp)) / 2.0 / sigmad
/ sigmad;
		if (colt >= 0 && colt < width && rowt >= 0 && rowt < width)
		{
```

```
                        if (left == 0)
                                weight = weight - (images.left[p]-images.left[t])*(images.left[p] -
images.left[t]) / 2.0 / sigmas / sigmas;
                        else
                                weight = weight - (images.right[p]-images.right[t])*(images.right[p] -
images.right[t]) / 2.0 / sigmas / sigmas;
                }
                else
                {
                        if (left == 0)
                                weight = weight - (images.left[p]-0.0)*(images.left[p] - 0.0) / 2.0 /
sigmas / sigmas;
                        else
                                weight = weight - (images.right[p]-0.0)*(images.right[p] - 0.0) / 2.0 /
sigmas / sigmas;
                }

                return exp(weight);
        }
        else
        {
                return 0.0;
        }
}

double LocalCost::normConst(double *weights, int size)
{
        double sum = 0.0;

        for (int i = 0; i < size; i++)
        {
                sum = sum + weights[i];
        }
        return sum;
}

int LocalCost::ncc(int p, int l, int row, int col, int width, int height, double *weights_l, double
*weights_r, double leftAvg, double rightAvg, int offset)
{
        //Calculate NCC
        double numerator = 0.0;
        double denorminator1 = 0.0;
```

```
	double denorminator2 = 0.0;
	double pleft = 0;
	double pright = 0;

	double weightl = 0.0;
	double weightr = 0.0;
	int indexl = 0;
	int indexr = 0;

	int count = 0;

	for (int i = row-offset; i <= row+offset; i++)
	{
		for (int j = col - offset; j <= col + offset; j++)
		{
			indexl = i*width+j;
			indexr = i*width+j+l;
			if (i >= 0 && i < height && j >= 0 && j < width)
				pleft = images.left[indexl];
			if (i >= 0 && i < height && j+l >= 0 && j+l < width)
				pright = images.right[indexr];

			weightl = weights_l[count];
			weightr = weights_r[count];
			count++;

			numerator = numerator + fabs((pleft-leftAvg) * (pright-
rightAvg))*weightl*weightr;
			denorminator1 = denorminator1 + pow((pleft - leftAvg)*weightl, 2);
			denorminator2 = denorminator2 + pow((pright - rightAvg)*weightr, 2);
		}
	}
	double denorminator = sqrt(denorminator1)*sqrt(denorminator2);
	int cost = (1.0 - fabs(numerator / denorminator))*400;
	delete [] weights_l;
	delete [] weights_r;
	return cost;
}

int LocalCost::dataCoFn_Census(int p, int l, int window, int width, int height, int lo)
{
	int col = p % width;
```

```
int row = (p-col) / width;

if (abs(l) < lo)
	return 400;

if(col+l >= width || col+l < 0)
	return 400;

int offset = (window-1)/2;
int wPixels = window*window;
double pleft = 0;
double pright = 0;
int indexl = 0;
int indexr = 0;
int *label1, *label2;
int count1 = 0;
int count2 = 0;

label1 = new int[wPixels];
label2 = new int[wPixels];

double center1 = images.left[p];
double center2 = images.right[p+l];

//apply Census transform to the image
for (int i = row-offset; i <= row+offset; i++)
{
	for (int j = col - offset; j <= col + offset; j++)
	{
		if (j >= 0 && j < width && i >= 0 && i < height)
		{
			indexl = i*width+j;
			pleft = images.left[indexl];
			if (pleft < center1)
				label1[count1] = 1;
			else
				label1[count1] = 0;
			count1++;
		}
		else
		{
			label1[count1] = 0;
```

```
                        count1++;
                    }

                    if (j+l >= 0 && j+l < width && i >= 0 && i < height)
                    {
                        indexr = i*width+j+l;
                        pright = images.right[indexr];
                        if (pright < center2)
                            label2[count2] = 1;
                        else
                            label2[count2] = 0;
                        count2++;
                    }
                    else
                    {
                        label2[count2] = 0;
                        count2++;
                    }
                }
            }

            //Calculate the Hamming distance between the two bit streams
            int hammingd = 0;

            for (int i = 0; i < wPixels; i++)
            {
                if (label1[i] != label2[i])
                    hammingd++;
            }
            int cost = ((double)hammingd)/wPixels*400;
            delete [] label1;
            delete [] label2;
            return cost;
}

int LocalCost::dataCoFn_ncc(int p, int l, int window, int width, int height, int lo)
{
            int col = p % width;
            int row = (p-col) / width;
            int offset = (window-1)/2;
            double suml = 0.0;
            double sumr = 0.0;
```

```
        double weightl = 0.0;
        double weightr = 0.0;
        int indexl = 0;
        int indexr = 0;

        if(abs(l) < lo)
                return 400;

        if(col+l >= width || col+l < 0)
                return 400;

        double *weights_l, *weights_r;
        int wPixels = window*window;
        weights_l = new double[wPixels];
        weights_r = new double[wPixels];
        int count = 0;

        //Calculate average of left and right window
        for (int i = row-offset; i <= row+offset; i++)
        {
                for (int j = col - offset; j <= col + offset; j++)
                {
                        indexl = i*width+j;
                        indexr = i*width+j+l;
                        weightl = weightingFn(j, i, col, row, width, height ,0);
                        weightr = weightingFn(j+l, i, col+l, row, width, height, 1);

                        weights_l[count] = weightl;
                        weights_r[count] = weightr;
                        count++;

                        if (i >= 0 && i < height && j >= 0 && j < width)
                                suml = suml + images.left[indexl]*weightl;
                        if (i >= 0 && i < height && j+l >= 0 && j+l < width)
                                sumr = sumr + images.right[indexr]*weightr ;
                }
        }

        double leftAvg = suml / normConst(weights_l, wPixels) ;
        double rightAvg = sumr / normConst(weights_r, wPixels);
        return ncc(p, l, row, col, width, height, weights_l, weights_r, leftAvg, rightAvg, offset);
}
```

ImageIO.h:

```
#ifndef __IMAGEIO_H__
#define __IMAGEIO_H__

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <iostream>
#include <fstream>

#include "LocalCost.h"

class ImageIO
{
public:
	ImageIO(int height, int width);
	virtual ~ImageIO(void);
	void openImage(char *leftn, char *rightn);
	void swapImage(void);
	void writeData(char *name, int *data, int num_pixels);

private:
	int h, w;
};

#endif
```

ImageIO.cpp:

```
#include "stdafx.h"
#include "ImageIO.h"

using namespace std;
extern LocalCost *lc;

ImageIO::ImageIO(int height, int width)
{
	w = width;
	h = height;
}

ImageIO::~ImageIO() {}

void ImageIO::openImage(char *leftn, char *rightn)
{
	int value;
	ifstream inFile;

	inFile.open(leftn);
	if (!inFile)
	{
		cout << "Unable to open file";
		exit(1);
	}

	for (int i = 0; i < w*h; i++)
	{
		inFile >> value;
		::lc->images.left[i] = log((double)value);
	}
	inFile.close();

	inFile.open(rightn);
	if (!inFile)
	{
		cout << "Unable to open file";
		exit(1);
	}

	for (int i = 0; i < w*h; i++)
```

```
	{
		inFile >> value;
		::lc->images.right[i] = log((double)value);
	}
	inFile.close();
}

void ImageIO::swapImage()
{
	double tmp = 0;
	for (int i = 0; i < w*h; i++)
	{
		tmp = ::lc->images.left[i];
		::lc->images.left[i] = ::lc->images.right[i];
		::lc->images.right[i] = tmp;
	}
}

void ImageIO::writeData(char *name, int *data, int num_pixels)
{
	ofstream outFile;
	outFile.open (name);
	for (int i = 0; i < num_pixels; i++)
	{
		outFile << data[i] ;
		outFile << "\t";
	}
	outFile.close();
}
```

Segmentation.h:

```
#ifndef __SEGMENTATION_H__
#define __SEGMENTATION_H__

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <iostream>
#include <fstream>
#include <stack>
#include <map>

class Segmentation
{
public:
	Segmentation(int width, int height);
	virtual ~Segmentation(void);
	int AssignLabel(int* disp);
	void InvalidateSmallSegments (int* disp, int label);

	int *seg;

private:
	int w, h;
	int MIN_REGION_SIZE;
};

#endif
```

Segmentation.cpp:

```
#include "stdafx.h"
#include "Segmentation.h"
using namespace std;

Segmentation::Segmentation(int width, int height)
{
	w = width;
	h = height;
	MIN_REGION_SIZE = 150;
	seg = new int[w*h];
}

Segmentation::~Segmentation()
{
	delete [] seg;
}

int Segmentation::AssignLabel(int *disp)
{
	int label = 0;
	int left = -999;
	int up = -999;
	int p = 0;

	for (int i = 0; i < h; i++)
	{
		for (int j = 0; j < w; j++)
		{
			p = disp[i*w+j];

			if (p == 0)
			{
				seg[i*w+j] = 0;
				continue;
			}

			if (i > 0)
				up = disp[(i-1)*w+j];
			else
				up = -999;
```

```
                    if (j > 0)
                            left = disp[i*w+j-1];
                    else
                            left = -999;

                    if (abs(p-left)<3)
                            seg[i*w+j] = seg[i*w+j-1];
                    else
                    {
                            if (abs(p-up)<3)
                                    seg[i*w+j] = seg[(i-1)*w+j];
                            else
                            {
                                    label = label+1;
                                    seg[i*w+j] = label;
                            }
                    }
                    if(abs(p-left) < 3 && abs(p-up) < 3 && abs(left-up) < 3 && seg[(i-1)*w+j] !=
seg[i*w+j-1])
                    {
                            int min = 0;
                            int max = 0;
                            //merge the two labels
                            if(seg[(i-1)*w+j]>seg[i*w+j-1])
                            {
                                    max = seg[(i-1)*w+j];
                                    min = seg[i*w+j-1];
                            }
                            else
                            {
                                    min = seg[(i-1)*w+j];
                                    max = seg[i*w+j-1];
                            }
                            for(int m = 0; m <= i; m++)
                                    for(int n = 0; n <= j; n++)
                                    {
                                            if(seg[m*w+n] == max)
                                                    seg[m*w+n] = min;
                                    }
                    }
            }
```

```
	}
	return label;
}

void Segmentation::InvalidateSmallSegments(int *disp, int label)
{
	stack<int> *index = new stack<int>[label];
	int *count = new int[label];
	int pIndex = 0;
	int l;

	for(int i = 0; i < label; i++)
	{
		count[i] = 0;
	}

	//count the size of each segment
	for(int i = 0; i < h; i++)
	{
		for(int j = 0; j < w; j++)
		{
			pIndex = i*w+j;
			l = seg[pIndex];
			count[l]++;
			if(count[l] < MIN_REGION_SIZE)
				index[l].push(pIndex);
		}
	}

	//check invalide segments
	for(int i = 0; i < label; i++)
	{
		if(count[i] < MIN_REGION_SIZE)
		{
			while(!index[i].empty())
			{
				disp[index[i].top()] = 0;
				index[i].pop();
			}
		}
	}
}
```

PostProc.h:

```
#ifndef __POSTPROC_H__
#define __POSTPROC_H__

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <iostream>
#include <fstream>
#include <vector>

#include "Segmentation.h"

class PostProc
{
public:
	PostProc(int width, int height, int num_labels);
	virtual ~PostProc(void);
	void RemoveSmallAreas(int num_labels, char *filename, int *disp);
	void writeData(char *name, int *data, int num_pixels);
	void txtRead(char *name, int *data, int num_pixels);
	void MatchLeftRightDisp(int *displ, int *dispr, int *disp);
	void MatchRightLeftDisp(int *displ, int *dispr, int *disp);
	//void getDisp(int *d);
	//void getValidLabels(int *labels, int num_labels);

private:
	int *pixels, *flag; //*disp,
	int w, h, min_region;
	//void MatchCbCrDisp(int *dispcb, int *dispcr);
};

#endif
```

PostProc.cpp:

```
#include "stdafx.h"

#include "PostProc.h"
#include "ImageIO.h"

using namespace std;

PostProc::PostProc(int width, int height, int num_labels)
{
	w = width;
	h = height;
	//disp = new int[h*w];
	min_region = 1000;
	pixels = new int[min_region];
	flag = new int[num_labels];

}

PostProc::~PostProc()
{
	//delete [] disp;
	delete [] pixels;
	delete [] flag;
}

void PostProc::txtRead(char *name, int *data, int num_pixels)
{
	int value;
	ifstream inFile;

	inFile.open(name);
	if (!inFile)
	{
		cout << "Unable to open file";
		exit(1);
	}

	for (int i = 0; i < num_pixels; i++)
	{
		inFile >> value;
```

```
                data[i] = (int)value;
        }
        inFile.close();
}

/*void PostProc::getDisp(int *d)
{
        for(int i = 0; i < h*w; i++)
        {
                d[i] = disp[i];
        }
}*/

/*void PostProc::getValidLabels(int *labels, int num_labels)
{
        for(int i = 0; i < num_labels; i++)
                labels[i] = 0;

        for(int i = 0; i < h*w; i++)
        {
                if(disp[i] != 0)
                        labels[disp[i]]++;
        }
}*/

void PostProc::writeData(char *name, int *data, int num_pixels)
{
        ofstream outFile;
        outFile.open (name);
        for (int i = 0; i < num_pixels; i++)
        {
                outFile << data[i] ;
                outFile << "\t";
        }
        outFile.close();
}

void PostProc::MatchLeftRightDisp(int *displ, int *dispr, int *disp)
{
        int num_pixels = w*h;
        int dl = 0;
        int dr = 0;
```

```
	for (int i = 0; i < h; i++)
	{
		for (int j = 0; j < w; j++)
		{
			dl = displ[i*w+j];
			dr = dispr[i*w+j+dl];
			if(abs(dl-dr)<3)
				disp[i*w+j] = dl;
			else
				disp[i*w+j] = 0;
		}
	}
	//output the matched disparity map
	//writeData("disp_statue.txt", disp, h*w);
}

void PostProc::MatchRightLeftDisp(int *displ, int *dispr, int *disp)
{
	int num_pixels = w*h;
	int dl = 0;
	int dr = 0;

	for (int i = 0; i < h; i++)
	{
		for (int j = 0; j < w; j++)
		{
			dr = dispr[i*w+j];
			if (j-dr < 0)
				disp[i*w+j] = 0;
			else
			{
				dl = displ[i*w+j-dr];
				if(abs(dl-dr)<3)
					disp[i*w+j] = dr;
				else
					disp[i*w+j] = 0;
			}
		}
	}
	//output the matched disparity map
	//writeData("disp_statue.txt", disp, h*w);
```

```
}

void PostProc::RemoveSmallAreas(int num_labels, char *filename, int *disp)
{
	Segmentation *segProc = new Segmentation(w, h);
	int num_l = segProc->AssignLabel(disp);
	cout << "Number of labels is " << num_l << endl;
	segProc->InvalidateSmallSegments(disp, num_l+1);

	//output the matched disparity map
	//writeData("statue_labels.txt", segProc->seg, h*w);

	//output the matched disparity map
	writeData(filename, disp, h*w);
	delete segProc;
}
```

GenerateHDR.m

```
function hdrImg = generateHDR(dimg1, dimg2, nleft, nright, disp, height, width, c)

num_pixels = width*height;

left_image = reshape(textread(nleft, '%d', num_pixels), width, height)';
right_image = reshape(textread(nright, '%d', num_pixels), width, height)';

left_image = double(left_image) / 255;
right_image = double(right_image) / 255;

hdrImg = dimg1;
%generate HDR image
for m = 1:size(dimg1, 1)
   for n = 1:size(dimg2, 2)-30
      r1 = dimg1(m,n);
      if r1 < 0
         r1 = 0;
      end
      l1 = left_image(m,n);

      r2 = dimg2(m,n);
      if r2 < 0
         r2 = 0;
      end
      if (n+disp(m,n)) <= width
         l2 = right_image(m,n+disp(m,n));
      else
         r2 = 0;
         l2 = 0;
      end

      w1 = r1/diff(c, l1);
      w2 = r2/diff(c, l2);
      if r1 > 0 || r2 > 0
         hdrImg(m,n) = (r1*w1+r2*w2)/(w1+w2);
      else
         hdrImg(m,n) = 0;
      end
   end
end
```

```
function v = diff(c, b)
power = length(c);
v = 0;
for p = 2:power
   v = v + c(p)*b^(p-2);
end
v = abs(v);
```

ToneMapping.m

```
function hdr = toneMapping(img)
height = size(img, 1);
width = size(img, 2);
Lw = exp(sum(sum(log(img + 10^-6)))/width/height);
L = 0.18*img/Lw;
Ld = L./(1+L);
Ld = Ld/max(max(Ld));
hdr = uint8(Ld*255);
%hdr = uint8((0.9-power(Ld-1, 2))*255);
```