Abstract

The paper presents a new interactive tool for editing images with constraints and using these images to generate a new High Dynamic Range image. The project aims to explore the possibility to edit the LDR images to generate an HDR image drawing application. In this project, Chia- Kai Liang et al constraint algorithm will be used in the constraints function, the function of generate a new High Dynamic Range image is written according to Greg Ward's Radiance theory and the Tone Mapping function is implement by the existing Photographic Tone Reproduction Operator. The most challenge thing in this project is using these existing algorithms to solve a new problem. In the result system, several different types of brushes and some filters will be developed in this application. There are two canvases in this application. While operating this result system, an image will be loaded in the first canvas. Then the user could use brushes to paint in the second canvas. The filers in this application will provide some constraints to the image editing, so that the image features present in the first canvas will spatially affect the brushes effects in the second canvas. Finally, the system will recombine these two images to create a new High Dynamic Range image.

Contents

1.Introduction

In present day, digital images always created in two different methods. One is using equipment to create digital images, like modern digital cameras or scanners. The other method is writing programs to produce digital images, like rendered with computer graphic technologies, or produced with drawing programs. However, while these images displayed on the monitor, sometimes people will discover the qualities of these images are not as good as expectation. These images may be darker or brighter than the actual scene. So it is common to manipulate a digital image before it is displayed, printed or transmitted. However, global image adjustments are impossible to satisfy all demands, it is usually essential to use the region selection to implement local image adjustments. Modern photographic editing software, like Photoshop or Gimp provides a large amount of tools to allow editing the image. Although these softwares are great utility, their operations are complex and professional. It is difficult to operate for novice user.

Currently, the majority of color images include red, green and blue channels for each pixel. There are 256 values for each of the red, green, blue channels, so more than 1.6 million different colors can be represented to each pixel [28]. This may seem to be a quite large number at first, but it should to be noticed that there are still only 256 values for each channel and the real world has a larger range of values for red, green, blue channels. Having just 256 values is unsatisfactory for representing many scenes. If the color in real world out of the range that the color image can be presented, it will appear darker or brighter. This is the reason why images may be darker or brighter than the actual scene.

In the interest of solve this problem to make images more realistic, it is necessary and essential to produce and use higher-resolution images. High-Dynamic-Range (HDR) imaging technologies can use floating point numbers to provide a wider range of color than the present Low-Dynamic-Range (LDR) image. This technology can solve this problem efficiently and effectively.

The paper concerns the development of a simple easy operating drawing application that allows image local adjustments. The project aims to explore the possibility to create an image drawing application. Designer needs to develop several different types of brushes and some filters in this application. There are two canvases in this application. While user operates this program, an image will be loaded in the first canvas. Then the user could use brushes to paint in the second canvas. The filers in this application will provide some constraints to the image editing, so that the image features present in the first canvas will spatially affect the brushes effects in the second canvas. Finally, pixel-wise multiply the images afterwards to create a high dynamic range image. The project main flow-process diagram is showing in Figure 1:

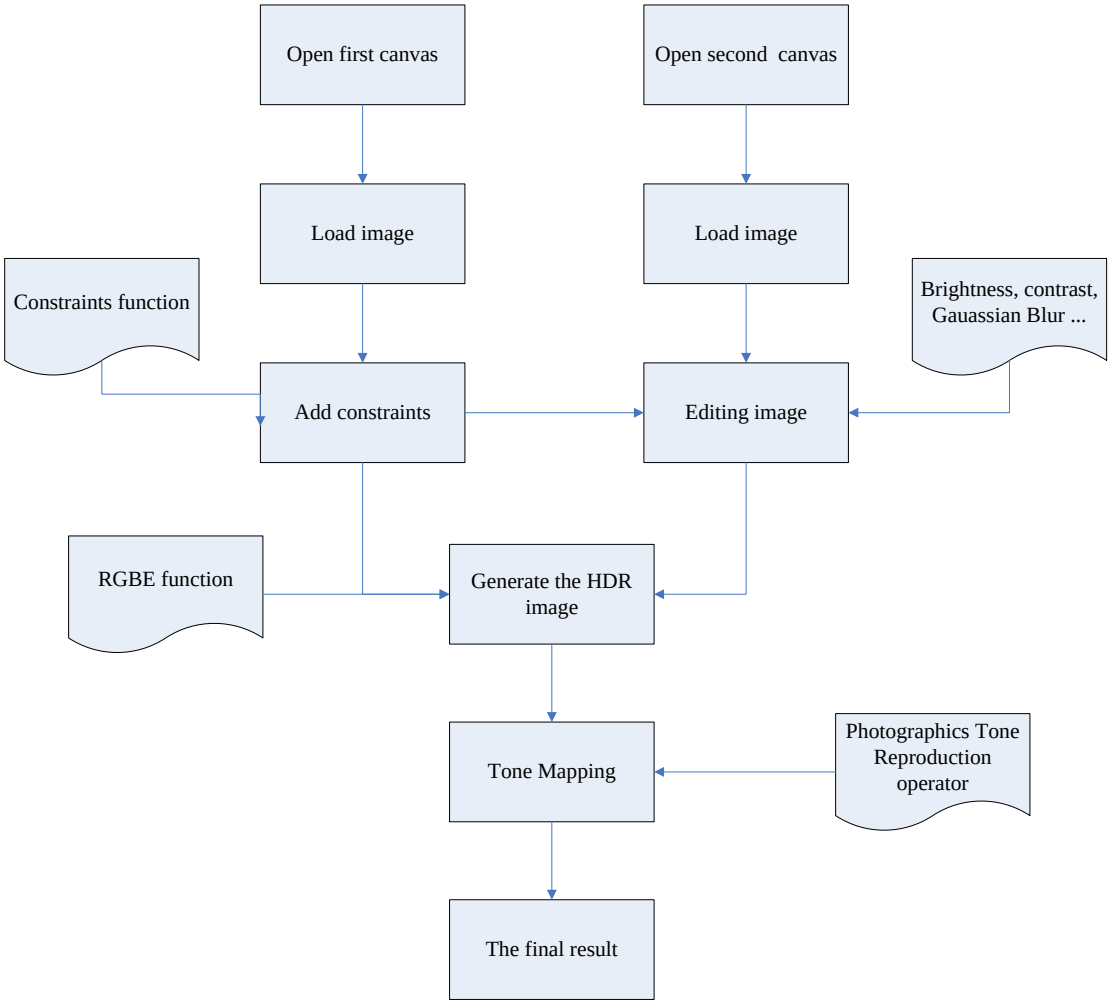


Figure 1: The project flow-process diagram.

Then, the objectives to meet this aim can be broken down as follows:

- Load LDR images and display images on two canvas.
- Design various local editing operations, include: Gaussian Blur, brightness, contrast adjustment and brushes.
- Implement the constraint filter using Chia-Kai Liang et al algorithm.
- Develop the RGBE function.
- Develop the Tone Mapping function using Reinhard et al Tone Reproduction Operator.
- Achieve all the objectives by C++ programming.

2. Background

This chapter introduces the relevant principle of High Dynamic Range Image. This knowledge will be used in the later section.

2.1 HDR definition

In order to understand what High Dynamic Range is, it is necessary to understand the concept of Dynamic Range. Dynamic Range is the ratio between max value and min value in physical measurement. For different objects it has different meaning. For practical environment, it refers to the ratio of brightest and darkest part in this scene. For digital cameras, it means the ratio from saturation to lens noise. In terms of digital images, Dynamic Range d is defined as the ratio of max luminance and min luminance in digital images.

$$d = L_{max} / L_{min}$$

The detail effects are largely depended on the dynamic range of digital images. It is an important factor to measure images quality.

An evident characteristic of the human eyes is during the course of day it can catch a huge range of illumination

It is an obvious feature to the human visual system that during the course of day it can catch a huge range of illumination it encounters. On a clear night, the intensity of moonlight is about 0.001 cd/m^2, the intensity of starlight can be 1/1000 of the moonlight. On the other hand, sunlight can be as much as a million times more intense than moonlight, about 100000 cd/m^2[28].

Because the calculated value of Dynamic Range d is a huge number, so usually it use logarithmic form to denote the value D.

$$D = log_{10} (L_{max} / L_{min})$$

Human visual system can obverse the dynamic rage value from min 3.0 to max 9.0, which is totally out of the range that the popular image representation methods can be represent.

The dynamic range in a normal scene is 10000 : 1, in extreme environment it even can be reach 10^9 : 1. At present, most digital images only could record limit value of dynamic range. We call it Low Dynamic Range (LDR) image. However, an HDR image can accurate record whole range color value each pixel in actual scene. That means an HDR image can has a much wider range value, like 0 ~10^9. Another character for an HDR image is the record is a linear value. Exactly, the value in each pixel is directly proportional to the measure value in real environment.

2.2 HDRI Capture

Because of the limitations inherent in most digital image equipments, it is impossible to create a single exposure to capture the full dynamic range of a digital image. Nevertheless, it is possible to use a conventional camera to record multiple exposures and then operate the relevant software to create an HDR image.

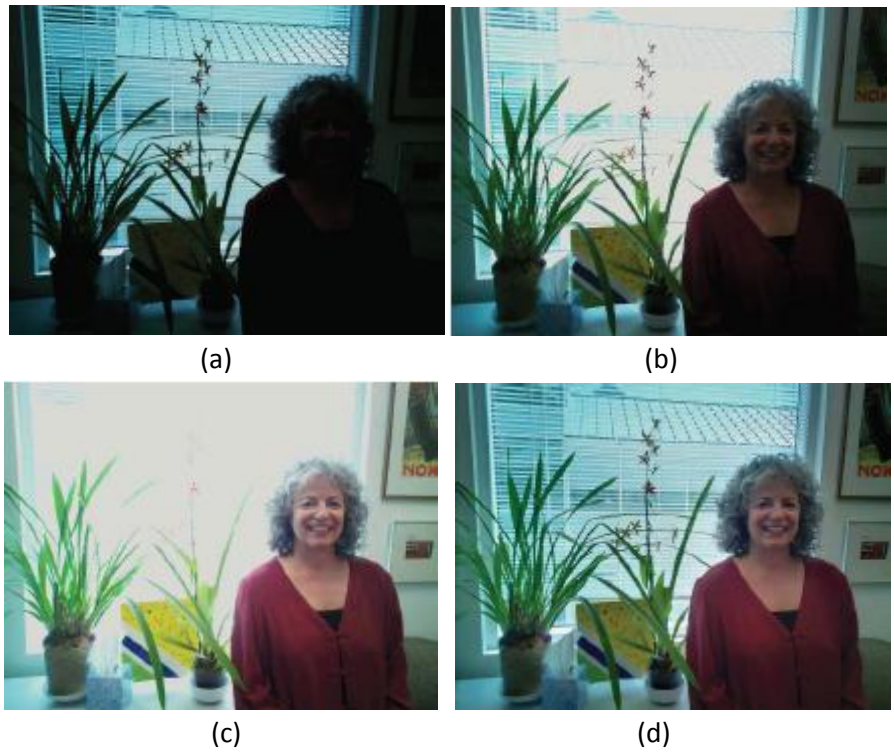


Figure 2: Three exposures sequence. (a) – (c) show the short, middle, and long exposures, (d) show the combined result. Source: Ref. [14].

In multiple exposures processing, every image in the sequence will have different pixels properly exposed, and except correct exposures, other pixels either overexposed or underexposed. However, every pixel will be suitably exposed in some images in this sequence. It is therefore feasible and satisfying to ignore too dark or too bright pixels in next computational procedures [28].

Then it is necessary to adjust all images to the same unit of measurement, exclude underexposed and overexposed pixels and other homologous pixels should be averaged across exposures. Finally, combine these images to create a new image. The result is an HDR image. Figure 2 shows an example of an HDR image capture from multiple exposures.

3.Basic framework implementation

3.1 Development platform

The project was implemented in Qt, Qt is Open source and Cross platform, so that designer could uses it free and in different operating systems. Then, Qt is Object oriented and has capable IDEs and rich APIs, there are more than 250 C++ classes and a large number of self-define classes could be used, which are convenient and use friendly, After that, Qt has helpful documentations and example applications, Finally, Qt support openGL image processing library, in this application many important functions are developed by using this library.

3.2 Implementation

3.2.1 User Interface

The QMainWindow library provides a simple framework for building the application's user interface. Designer can add QMenuBar, QToolBar and QStatusBar. In this project the main application window include a QMenuBar, four QToolBars and a QStatusBar(Figure 3). Menus are accomplished in QMenu and kept in the QMenuBar, while QActions are added to the menus and display as menu items. Toolbars are implemented in the QToolBar class, I used addToolBar() function to add toolbars to the main window, then add relevant QActions to the toolbars. In the toolbars, it is required to apply addSeparator() to determine toolbars position.

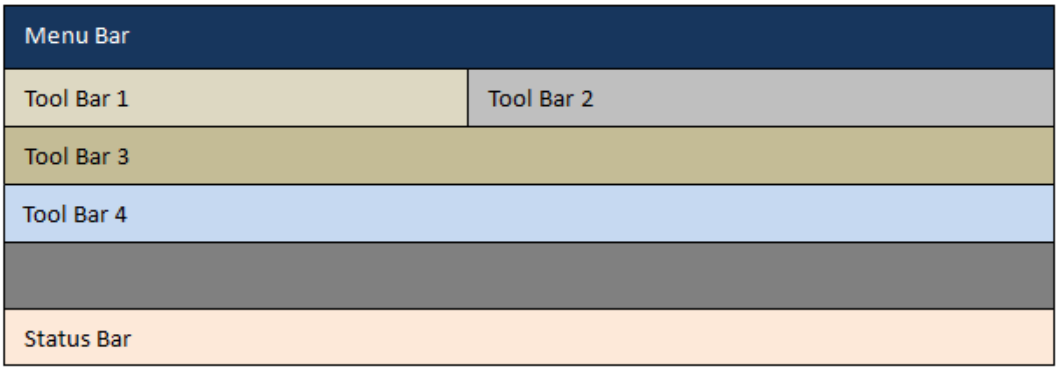


Figure 3: The main application window.

3.2.2 Load LDR image

At beginning, The QLabel class is chosen to load the image, it has the ability to load and display an image, but after research, I found it is better to use the QGraphicsScene class to load an image and use the QGraphicsView class to display this image. The QGraphicsScene class allows handling a large amount of 2D graphical items and these graphical items will visible with the help of QGraphicsView. In this application, the Canvas class is inheriting from QGraphicsScene, the mianwindow function cites Canvas and two canvases are defined in the mainwindow function. While images are opened as the QImage format, canvases will use addPixmap() to load images, then they are used together with QGraphicsView for displaying images on the screen.

3.2.3 Basic editing tools.

The QGraphicsScene class provides a large quantity of convenience functions to support images editing. But if user wants to use mouse to editing the image in the canvas, it is essential to acquire the mouse event. The QGraphicsSceneMouseEvent class provides mouse events in the QGraphicsView.

Line

The CanvasLine class is written to the line drawing. While the mouse is pressing in the second canvas, a new item will be added. scenePos() returns the mouse start cursor position in scene coordinates and the position stores in p_1, when the mouse is moving p_2 will record the current cursor position constantly, then use the moveTo() and lineTo() to add wispy straight line constantly. Finally, the drawPath function in the QPainter class to draw the given path. After the mouse is released the item will be deleted.

DirectLine

The CanvasDirectline class provides the tool that allows painting direct line in the canvas. The CanvasDirectline class is quite similar to the CanvasLine class, the start point and the end point is stored in point1 and point2, respectively. Then the QLineF uses point1 and point2 to generate a finite direct line. After that, the setline function in QGraphicsItem sets the item’s line to be the given line [25].

Rectangle

The CanvasRect class defines the function of painting rectangles in the canvas. While the mouse is pressing in the seconde canvas, a new recntangle item will be added and scenPos() will return the mouse start cursor position in scene coordinates and store in r, then r will be separated into rx and ry, which represents the value in the

X and Y axis, respectively. When the mouse is moving to the end point, this position will be recorded in e and its XY axis values are ex and ey. After that， we can use the following equation to get the rectangle's height and width.

$$\text{Width} = ex - rx$$
$$\text{Height} = ey - ry$$

Lastly, we send the start point XY values, the width and height to the QRect function to generate a new rectangle and use setRect() to set the item's rectangle to be the given rectangle.

Ellipse

The CanvasEllipse class provides the ability to draw ellipses in the canvas. The ellipse function is similar to the rectangle function, the only different between them is in ellipse function we use the QRect in QGraphicsEllipseItem and in rectangle function we use the QRect in QGraphicsRectItem. The QRect function in different graphical item classes will create different shapes. The first QRect function will generate an ellipse and the second QRect will generate a rectangle.

In the application, I also defined functions that have abilities to change pen width, pen color and brush color, so that user could set these attributes to draw different style shapes and lines in the canvas.

3.2.4 Advanced editing tools

The OpenGL image processing library provides a large amount of classes to help designer to develop image editing tools. In this application, I used OpenGL to develop the Gaussian Blur function, Opacity function, Brightness function and Contrast function.

Gaussian Blur

There is a QGraphicsBlurEffect class in the OpenGL, it provides a blur effect. At beginning, A QGraphicsItem is defined to load the canvas information. While the user set the radius value, a QGraphicsBlurEffect instance is constructed in the application, then, the user parameter is sent to QGraphicsBlurEffect's constructor by using setBlurRadius(). Lastly, the blur effect adds to the previous QGraphicsItem. Figure 4 (a) shows the Gaussian Blur effect in the application.

Opacity

The OpenGL image processing library also provides a QGraphicsOpacityEffect class, the implement steps of opacity function also as same as the Gaussian Blur function. Figure 4(c) demonstrates the Opacity effect.

Brightness

In this function, in order to avoid floating point numbers, the brightness value is multiplied by 100. While user decides the brightness, the brightness value for each RGB value pre-identified as

$$\text{Brightness} = \text{value} + \text{brightness} * 255 / 100$$

At the same time, the application sets an upper limit value 255 and a lower limit value 0. If the brightness value less than the lower limit value, return the lower limit value, else if the brightness value greater than the upper limit value, return the greater value, otherwise, return the brightness value. After the brightness value is determined. Each pixel's RGB values add the brightness from left to right, from top to bottom. At last all pixels' RGB values add the brightness value in the image. Then, the new image will be added into the previous QGraphicItem to replace the original image. Figure 4 (b) shows the brightness effect.

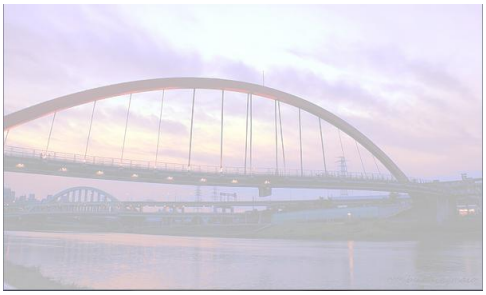
Contrast

In this function, in order to avoid floating point numbers, the contrast value is multiplied by 100. While user decides the contrast, the contrast value for each RGB value pre-identified as

$$\text{Contrast} = (\text{value} - 127) * \text{contrast} / 100 + 127$$

At the same time, the application also sets the upper limit value 255 and the lower limit value 0. If the contrast value less than the lower limit value, return the lower limit value, else if the contrast value greater than the upper limit value, return the greater value, otherwise, return the contrast value. After the contrast value is determined. Each pixel's RGB values add the contrast value from left to right, from top to bottom pixel by pixel. At last all pixels' RGB values add the contrast value in the image. Then, the new image will be added into the previous QGraphicItem to replace the original image. Figure 4 (d) shows the contrast effect.

(a)



(c)

(b)

(d)

(e)

Figure 4: (a) the Gaussian Blur effect, radius value 5, (b) the brightness effect, brightness value 50, (c) the opacity effect, opacity value 0.4, (d) the contrast effect, contrast value 50, (e) the original image.

3.3 Summary

The User Interface is developed in Qt by using C++. The final User Interface is demonstrated in Figure 5. There are one toolbar and two canvases. Users could choose different kinds of editing tools to editing the images that loaded in these two canvases and set the tone mapping operator user parameters in the toolbar. The toolbar provide pen drawing tools that could paint line, rectangle, direct line and ellipse, and also offer the brightness, opacity, contrast and Gaussian Blur filter.

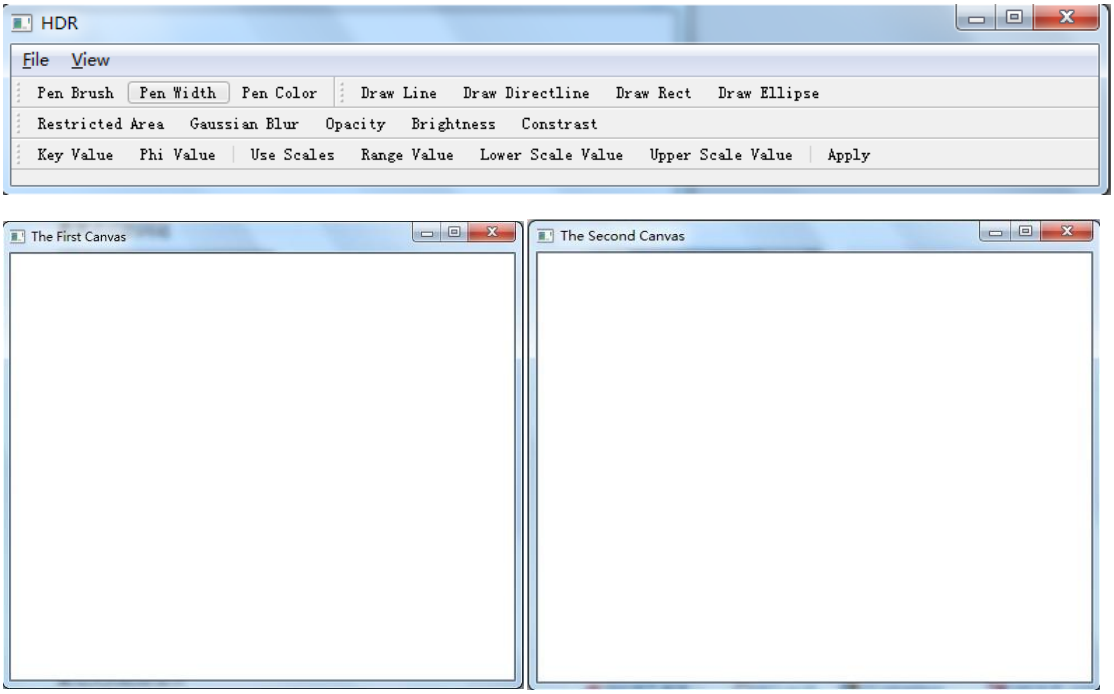


Figure 5: The application User Interface.

4. Contrast Function

Filters in this application need provide contrast functions that allow local adjustment through scope selection. During the development process, at beginning, I used an unsuitable theory to develop this function. Firstly, I declare a dynamic array A to store the input image's pixel values in the first canvas. Secondly, I implemented a transparent rectangle in the first canvas as the constraint area. Thirdly, I got the coordinate of the constraint area in the first canvas, so that while user paints in the second canvas, the modified pixel values in the setting scope in the second canvas will be stored in another dynamic array B. Fourthly, I used the values in the dynamic array B to replace the values in the dynamic array A in relevant place, so the new dynamic array A contains the modified pixel values in the constraint area. Finally, I used the dynamic array A to generate the modified image (Figure 6). However, this method seems possible to be the constraints function, actually this method is unsuitable. Because the effect in the constraint area is average and it has obvious edges, which feel stand out to other parts in the image. After discussion and experiments, an existing algorithm will be used as the constraints function, which has good performance in the selection of the constraint area. In this section, an existing algorithm invented by Chia- Kai Liang et al [18] will be introducing, this algorithm is used in the application to achieve the goal of editing image with contrasts.

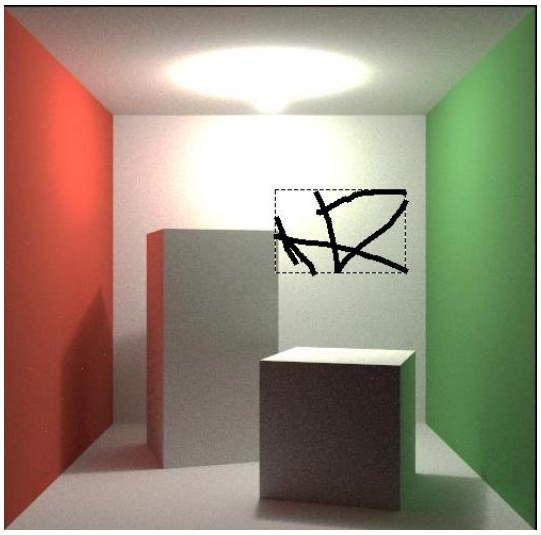


Figure 6: The effect of the initial wrong constraint function.

4.1 Local Image Adjustment

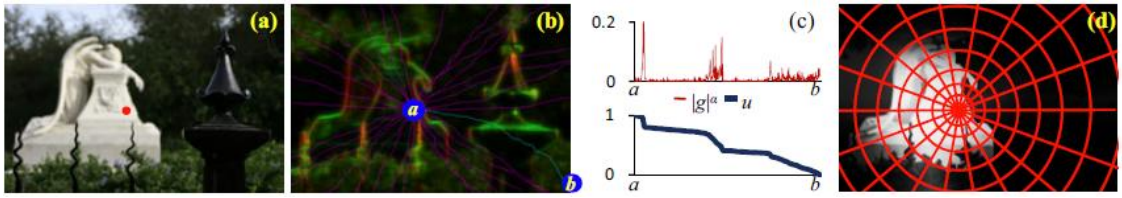


Figure 7: the Main steps of the contrast algorithm [19].

The main steps of this algorithm are demonstrated in Figure 7. At beginning, a single point in the image is selected by user and at the same time, the point's coordinate is capturing to the application (Figure 7a). Then, the algorithm computes the selected point's diffused gradient field and generates paths from the chosen point to the image boundaries according to the diffused gradient field (Figure 7b). After that, the algorithm uses Lischinski et al method [19] to compute the influence value along each path (Figure 7c). Finally, a scattered bilateral interpolation will be used to generate an influence map (Figure 7d). In this application, the influence map stands for the scope that will be effect while editing the image and it also shows the effect strength to all pixels.

The algorithm is written base on the algorithm in [19]. So it is necessary and essential to introduce the Lischinski et al's algorithm. Firstly, this algorithm set some user-drawn strokes. Then, these strokes combine adjustment parameters and send them to other pixels in an edge-aware style, which is obtained by solving a large linear system **Af = b**.

$$b = \sum_k V_k W_k$$

A is an n * n sparse symmetrical positive definite matrix, the sparse matrix **A** can be defined as a total of **A = H + W**, where the matrix H only depend on the original input image and $\mathbf{W} = \sum_j \mathrm{diag}(w_k)$ is a diagonal matrix that stores all pixels contrast effect value and vector w_k in this equation denotes the pixels of the k-th stroke.

$$\text{Where } H = h_{i,j} = \begin{cases} -\lambda / (|L_i - L_j|^a + \varepsilon) & \\ = -\lambda / (|g_{i,j}|^a + \varepsilon) & i \neq j \;\&\; j \in N_{4(i)} \\ \mathbf{W_i} - \sum_{\mathbf{k} \in \mathbf{N_{4(i)}}} \mathbf{h_{ik}} & i = j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

And $\mathbf{b_i = w_i g_i.}$

In the equation (1), subscripts i and j means the pixel i and pixel j in the image and $N_{4(i)}$ denote the four neighbor pixels around the pixel i. $|L_i - L_j|$ denotes the gradient between the pixel i and the pixel j, for HDR images is computed as the log average luminance and for LDR images is computed as the luminance differences. In the implement,λ , a and ε are all use the default value to produce the result, $\lambda = 0.2$, $a = 1$, and $\varepsilon = 0.0001$. The parameter ε uses to make the linear system stability and a and λ use to control the gradient changes.

According to the Lischinski et al's suggestion, all constraints should be solved separately as the basic influence function u_k, $Au_k = w_k$, so that $f = \sum_k V_k u_k$. In this algorithm, the influence map of constraint w_k then is defined by the u_k and the algorithm use the linear combinations of $\{u_k\}$ to generate a new image with different adjustment values without resolving the method again.

Finally, as shown in the result of the Lischinski et al's method, this algorithm obvious has the ability to effective generate the solution in 1D. So that, it is possible to us to compute the solution along these 1D paths originating from the selected point, and then generate the final influence map.

4.1.1 Constraint Propagation

In Equation 1, if we consider each pixel only has two neighbors, so that the matrices H and A all become symmetrical tridiagonal matrices when the pixel in the continuous path in the original image [18]. As a result, this problem looks like a traditional 1D partial differential equation. Suppose we could offer the new system contain l pixels and this system only with two boundary conditions in the shape of a single-pixel constraint at the end of all paths, we have:

$$\mathbf{A}\mathbf{u}_\mathbf{k} = \mathbf{w}_\mathbf{k}, \quad k = \{0, 1\} \qquad (2)$$

where

$$\begin{cases} \mathbf{A} = \mathbf{H} + \mathbf{W} = \mathbf{H} + \text{diag}(w_0) + \text{diag}(w_1) \\ \mathbf{w_0} = [1, 0, 0, \dots, 0]^\mathbf{T}, \text{and} \\ \mathbf{w_1} = [0, 0, 0, \dots, 1]^\mathbf{T} \end{cases} \qquad (3)$$

So that, it is possible to denote $h_i = h_{i,i+1}$, $g_i = g_{i,i+1}$, and set the values on a path, which length is l, as $u_0 = [u_0, u_1, u_2, \dots, u_{l-1}]^T$ [18]. Then, getting two neighbor rows (i, i +1) from A and replacing the equation 1 into the system to get the new relationship:

$$h_i u_i - (h_i + h_{i+1})\, u_{i+1} + h_{i+1} u_{i+2} = 0 \qquad (4)$$

$$\Rightarrow h_i(u_i - u_{i+1}) = h_{i+1}(u_{i+1} - u_{i+2}) \qquad (5)$$

$$\Rightarrow \triangle u_i |g_{i+1}|^a = \triangle u_{i+1} |g_i|^a \qquad (6)$$

That means all pixels in the image, their influence map value $\triangle u_i = u_i - u_{i+1}$ is inversely proportional to h_i, after replacing the equation 1, the gradient proportional went up while the value of a increased. Because the equation 6 is stable so the parameter ε is removed in the replacing process, which make the system equation easier. In the equation 1, we set$\lambda = 0.0001$, so $\lambda \to 0$, and user constraints predominate the solutions at the end points and u_0 could be simply effectively look as a reduction from one to zero.

$$\begin{cases} u_0 = 1, \\ u_i = u_{i-1} - |g_{i+1}|^a / \sum_i |g_i|^a \qquad i = \{1, 2, ..., l-2\}, \\ u_{l-1} = 0. \end{cases} \tag{7}$$

With the help of this formulation, we can generate the influence maps along each path in the input image efficiently and effectively. In the following part, we will look at the method to create reasonable paths and desirable boundary conditions.

4.1.2 Path generation and Boundary Conditions

The purpose of this part is using importance sampling in 1D paths to access the full 2D optimization process [18]. The distribution of paths plays a quite important role in the algorithm. Normally, the distribution should observe three standards [18]:

1. In order to get higher accuracy, the density of samplings will be higher around the selected point.

2. The distribution of samples should be evenly from the selected point to the image boundaries for good coverage

3. The edges in the same object should only be chosen one time, the sampling bias need to be minimized.

In the cause of satisfy Standard 1 and 2, it is possible to generate paths evenly away from the selected point. Nevertheless, the function is not stable while the path produces at a glancing angle to an image edge, that means even the selected point change slightly, it may cause the path cross the same object several times or under-sampling other objects. Chia-Kai Liang et al found it is possible to solve these problems and improve the stability of this algorithm by cutting through the margins as orthogonal as possible [18], so that the solutions will satisfy Standard 3, but the samples may no more distribute plainly away from the clicked point.

In order to balance the conflict between Standard 2 and Standard 3, this algorithm uses a particle system. Firstly, all particles have an original unit velocity v_o from the selected point toward the image boundary. To satisfy Standard 1 and Standard 2, the particles' radial directions are spread evenly. Then we use the gradients in the image to aim the particles to make the solution meet Standard 3. A reasonable method is to handle the image gradients as a gravitational field so that weaker edges will make the particles at shorter distance and stronger edges will cause the particles at biggish distance. Whereas, the particles will wrongly oscillate round strong edges due to the classic mechanics. In this algorithm, when we summarize the gravitational force, the signs will be removed to achieve the goal to solve the problem. The summarize value let us to compute the field again according to diffuse the image gradient $g = [g_x, g_y]$ between points and their neighbor points.

$$\boldsymbol{g_d}(x, y) = \sum_{u \neq x} \sum_{v \neq y} \frac{[|g_x(u,v)|, |g_y(u,v)|]}{(u-x)^r + (v-y)^r} \qquad (8)$$

In this formulation r = 1. The diffused gradient field g_d makes it possible for us to imitate the margin attraction, which satisfies Standard 3. The particle's speed is changed from v to v' while it across a point (x, y).

$$\boldsymbol{v'} = \frac{\boldsymbol{v + w_u v_0 + w_g s(v) \odot g_d(x,y)}}{\boldsymbol{||v + w_u v_0 + w_g s(v) \odot g_d(x,y)||_2'}} \qquad (9)$$

In equation 9, s(v) stands for the symbols of the constituents in v, using s(v) will provide all particles' speed cross the same quadrant, that ensure all particle have the ability to exit the image. $||.||_2'$ means the l_2 criterion, and the operator $\odot$ denotes the multiplication of front vector. After many experiments, Chia-Kai Liang et al set

the parameter w_u = 1.3 and w_g = 1.0, respectively, because these two values have best effect to balance Standard 2 and Standard 3. Finally, we can generate the image's sampling paths according to the particles' tracks.

In equation 7, we can clear get the information that the value of simple boundary conditions u_0 starts from one to zero from the selected point to the image boundaries. However, if the selected point and the image boundaries are belonging to the same object, the equation 7 could not solve the problem well. To solve the problem, we decay the largest cumulative gradient over all paths to renormalize the ratio of influence value [18]. Then the $G_{max} = \{G_0, G_1, \dots, G_{m-1}\}$ and $G_j = \sum_i |g_i^m|^a$ represents the cumulative gradient in the m-th path, so the equation 7 is changed to :

$$\begin{cases} u_0 = 1, \\ u_i = \max(0,\ u_{i-1} - |g_{i-1}|^a / G_{max}) \quad i = \{1, 2, \dots, I-2\}, \end{cases} \tag{10}$$

A path will in the same region where the selected point in, while it does not across to any strong edges. The equation 10 demonstrates that the influence map's entire quality will be improved, if the solution along the path is close to one.

4.1.3 Scattered Bilateral Interpolation

After generating the paths, it is necessary use these paths to create the influence map for the input image. Paths in the image are resolved differently and independently, so the solutions of these paths may nonuniform, and suitable filtering must be added in the algorithm to remove this alteration. Moreover, it is necessary to mention that influence values in two neighbor pixels are similar and parallel. To satisfy all demands, a cross bilateral filter is used in this part, this bilateral filter is introduced in the Flash Photography Enhancement via Intrinsic Relighting [11], which is written by Elmar Eisemann and Fr ́edo Durand. If we give pixels q' s that on the path with their solutions u_q' s, then we can get the pixel p 's interpolated influence value u_p' according to :

$$\boldsymbol{u_p}' = \frac{\sum_{q\in\delta} \boldsymbol{f_d}(||\boldsymbol{p}-\boldsymbol{q}||_2)\boldsymbol{f_r}(||\boldsymbol{I_p}-\boldsymbol{I_q}||_2)\boldsymbol{u_q}}{\sum_{q\in\delta} \boldsymbol{f_d}(||\boldsymbol{p}-\boldsymbol{q}||_2)\boldsymbol{f_r}(||\boldsymbol{I_p}-\boldsymbol{I_q}||_2)} \qquad (11)$$

Where in the equation 11, δ stands for the collection of all pixels on the same path, I_p and I_q mean the pixel value in the pixel p and the pixel q, respectively, f_d denotes the distance filter kernels and f_r is the range filter kernels. Paris and Durand's research has demonstrated the bilateral grid technique could accomplish this non-linear filter effectively and efficiently [7] [23].

Nevertheless, due to the distributions of the paths are not plainly away from the selected point to the image boundaries, the result is not as good as expectation. Because the path density becomes lower and lower while it apart away from the selected point, so it is necessary to use a much larger filter kernel to fulfill the losing value. On the other side, the paths density is much higher beside the selected point, so it is essential to use a smaller filter kernel. The normal bilateral grid only could solve the problem efficiently while the filter kernel is spatially constant, while using a consistent filter kernel, this filter kernel may lead to holes or divulgation in the solution. Because this application need support real-time display, so it is important to explore a efficient and quick method to solve this problem.

After research and exploration, Chia-Kai Liang et al found while the distance between the selected point and the path is increasing, its density reduce linearly, and hence the width of the kernel also linear enhance with the increase of the distance. So that f_d becomes a Gaussian filter:

$$f_d\ (||\mathrm{p}-\mathrm{q}||_2) = \exp\left(-||\mathrm{p}-\mathrm{q}||_2^2/\sigma_p^2\right) \qquad (12)$$

It is possible to use the distance between selected point c and the point p to write a function to get its variance σ_i^2,

$$\sigma_p^2 = (||p-c||_2)\,\sigma_0^2 \quad . \qquad (13)$$

It is reasonable for us to use the spatial variance frame to generate a fresh grid structure in polar coordinates focus on the selected point. The Three-Dimension grid includes the radius dimension, the range dimension and the angular dimension [18]. When the width of the grid in the radius dimension and the angular dimension is steady, the gird linear increase or decrease along the range dimension. So while we implement the filter kernel on the gird, this outcome and the result that generated by equation 12 will be quite similar and approximate.

When the path across strong edges, it has the ability to get scattered in the Three-Dimension grid in the range dimension, though this path still in the input image. Consequently, it is essential to rasterize the One-Dimension paths in the Three-Dimension grid to assure the One-Dimension paths is still range continuity. We then get the filtered samples by implementing three in demountable One-Dimension filters along the range, radius and angular dimension, which are followed by trilinear interpolation. Finally, we need use Levin et al sigmoid function to raise the contrast effect in the output image [21]. Particularly, we use the value u_p'' to replace the influence value u_p':

$$u_p'' = (1 + exp(-\beta(u_p' - 0.5)))^{-1} \quad (14)$$

In the application, the value of β is 10.

Now, we could use the paths in the input image to generate the influence map. It is a fixed, reasonable, and designable influence map to the selected point. After generate the influence map, we need added a method to make it possible to change the scope size of the influence map. To achieve this goal, a parameter s is added in equation 10 and we use sG_{max} to replace the original parameter G_{max}. If the value of s is larger than one, the scope size of the influence map will become bigger. Otherwise, if s < 1, the scope size of the influence map will become smaller. In the application, while the mouse move up and move down the value of the parameter s will be changed, and the system will generate a fresh influence map.

4.2 Summary

The main technological process is user selects a point in the input image. Then the method collects the paths from the clicked point in the image. After that, the algorithm computes the adjustment solution for each path. Finally, it uses a scattered bilateral interpolation to create the influence map of the input image. Generally, there are three

important steps in this algorithm: path generation, constraint propagation and scattered bilateral interpolation. The result demonstrates that this algorithm obviously cut down the time and calculational costs for the image adjust, and this algorithm is effective enough to support running operations in the application.

5.RGBE Function

After finish the image editing, According to project demands, it is essential and necessary to combine the input image and the modified image together to generate a new High Dynamic Range image. In this part, a RGBE function is written according to the Greg Ward's theory. In the following section, these RGBE function will be explained clearly.

5.1 Background

5.1.1 RGBE File Format

RGBE is an image format that invented by Greg Ward. Each pixel in this format needs four bytes to store. The RGBE file format stores pixels as a one byte normal RGB value (red, green and blue) with a one byte extra share exponent value [33]. The advantage of this RGBE file format is that this format provides the extended range floating point value to the pixels. Usually, the range of pixels value is only from 0 to 255 in the standard 24-bit image format. Therefore, many colors values are totally out of this range, like the extreme bright light. As a result, we have to either reduce the bright value in pixels to make it suitable or absolutely remove the bright value, which make the image unreal. However, by using the shared exponent, it could operate very bright pixels or very dark pixels without loss the pixel details.

5.2 Algorithm

At beginning, the mapping function stores gamma-corrected bytes to give the pixels higher dynamic range and this method will keep the pixels accuracy. Here is a simple routine to demonstrate the transform steps [33]:

The Initialization of gamma table:

```
gamtable: array [0-1024] of byte;
        for j: int j = 0, j < 1024, j = j+1 do
            gamtable[i] = 256.0*((j + 0.5)/1024)^(1.0/gamma);
```

To convert each pixel primary:

```
if rvalue >= 1.0
```

```
then ivalue = 255;
    else ivalue = gamtable[floor(1024.0 * ravlue)];
```

The reason we use a relative large array is the accuracy will not be influenced for some smaller values. In order to solve the problem of how to store the accurate pixel, we store the three primaries at each pixel separately. Each primary at the pixel use an 8-bit mantissa and we also use another 8-bit mantissa to the shared exponent, so that each pixel needs 32 bits space. The fractional part is always standardized to lie from 0.5 to 1.0. While we add the exponent to three primaries' mantissas, the normalization affection is only guaranteed to the largest value, and the other two mantissas may be less than 0.5 [33].

For instance, the colour

$$[0.3 \quad 0.02 \quad 0.1]$$

would be changed to

$$[0.6 \quad 0.04 \quad 0.2]^* \ 2^{-1}$$

Then in the 32-bits floating format, the colour value becomes:

$$[153 \quad 10 \quad 51 \quad 127]$$

The shared exponent value changed from -1 to 127, because it is essential to add a compensatory to make the unsigned value become a signed value. In this application, we add all unsigned value with 128 to give colors a nearly even range. Larger values are favor in a smaller compensation and smaller values are interest in a larger compensation.

The colour present in the image is determined by the largest primary value, so the format pays more attention to the largest primary value at the cost of the accuracy in the others primaries. The gold of the 32-bit format is preserving the bits, which are most important and significant [33].

The math routines *frexp* and *ldexp* play significant roles in the format conversion. *Frexp* use a real value to change to a mantissa, which value is between 0.5 and 1.0. Hence, we use the *frexp* function to convert from machine float value to 32-bit real pixel value. This is the *frexp* routines:

machine float convert to 32-bit real pixel:

```
rmf, gmf, bmf          real: machine RGB values
rma, gma, bma, ex;     int:  RGB mantissas and exponent

rmf = max(rmf, max(gmf, bmf));

if value < 1e – 32
        then begin
                rma = 0;
                gma = 0;
                bma = 0;
                ex = 0;
                end;
        else begin
                v = frexp (v, e) * 256 /v;      e is returned exponent
                rma = rmf * v;
                gma = gmf * v;
                bma = bmf * v;
                ex = e +128;
                end;
```

We can rely on *Ldexp* to make the 32-bits real pixel convert to the machine float, the routine is [33]:

32-bit real pixel converts to machine float

```
if ex = 0
        then begin
                rmf = 0;
                gmf = 0;
                bmf = 0;
                end;
        else begin
                v = ldexp(1.0/256.0, ex -128);
                rmf = (rma + 0.5) * v;
                gmf = (gma + 0.5) * v;
                bmf = (bma + 0.5) * v;
                end;
```

To the project, it is only need generate a High Dynamic Range image, so it is only necessary to convert from the machine float to the 32-bit real pixel.

5.3 Implement.

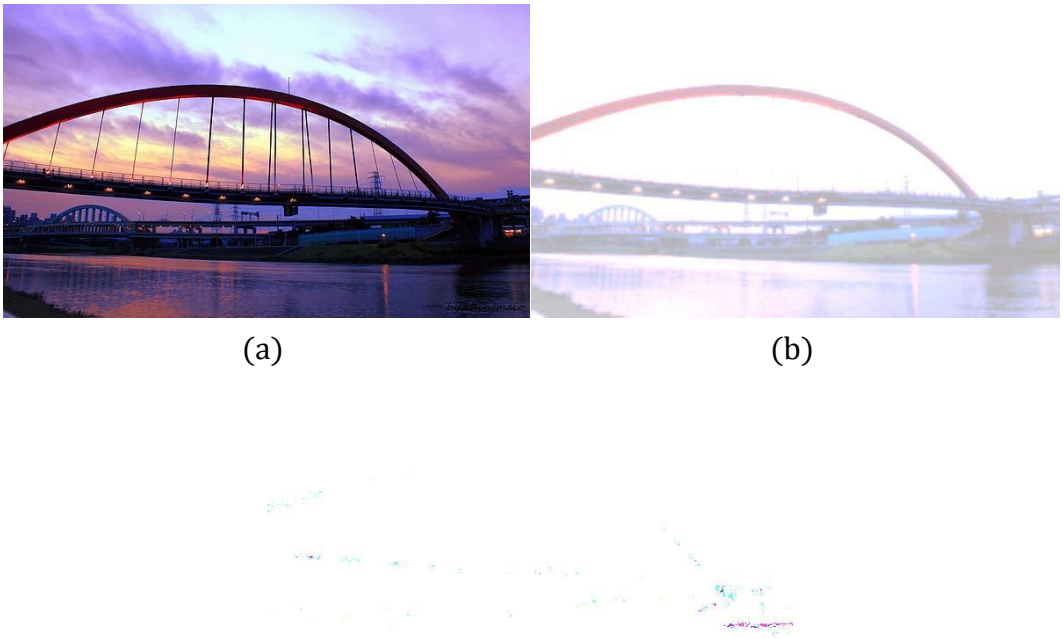
While the image is loaded in the application, the application will get the image information that includes: width, height, exposure and gamma. Then we use these values to write the header of the High Dynamic Range image. At beginning of the header, we need define the program type after "#?", the default value is "RGBE". Then, we add the image gamma value "GAMMA = %g" and exposure value "EXPOSURE = %g" to the header, respectively. After that, we insert the sentence of "FORMAT=32-bit_rle_rgbe" into the header to declare the new image format is 32-bit RGBE file format. Finally, we gain the width and height to the new image header in " –Y height +X width" format. So far, the header of the new High Dynamic Rage image has been defined and declared.

After finish defining the header, we get the pixels' values both in the original image and the modified image and store all pixels' RGB values into a One-Dimension dynamic array. Afterwards, we use the algorithm that has been explain in the preview section to convert the RGB values to the 32-bits RGBE values pixel by pixel and store these new RGBE values into another dynamic array. Lastly, it is possible to use the new dynamic array to generate a new High Dynamic Range image.

5.4 Summary

The RGBE function could use the input image information to create a new High Dynamic Range image header and converts the RGB value to the RGBE values. Then the function could generate a new High Dynamic Range image by using these RGBE values.

Nevertheless, the pixels value in the output High Dynamic Range image always totally out of the range that the ordinary screen or monitor could display (Figure 8c). That means we could not see the effects in the High Dynamic Range image, in order to make this image displayable on the screen, it is necessary and essential to reduce the pixel values to let the image suitable to the screen, but the contrast in the image must maintain as much as possible, so we need use a tone reproduction operator to tone mapping the High Dynamic Range image to decrease the pixel values and keep the High Dynamic Range effects at the same time, which we will discuss in the following section.



(a) (b)

(c)

Figure 8: (a) The original image, (b) the modified image, (c) the output new HDR image.

6.Tone Reproduction Operator

6.1 Introduction

Even we can capture a much wider dynamic range to create a new HDR image. Unfortunately, there is still a serious problem need to be solved for HDR images. At present, most display devices only have a limited dynamic range (less than 100 ~ 1) can be used. The difference between the large ranges of illumination that can be captured and the small ranges that can be displayed by available display devices makes HDR images difficult to display accurately. That means the output level reproduced by display devices may be completely unable to meet the light intensity level in the scene. This is the HDR tone-mapping problem.

Normally, using visual models is a sufficient and effective method to solve the HDR tone-mapping problem. Figure 9 shows a pictorial outline of tone-mapping problem.

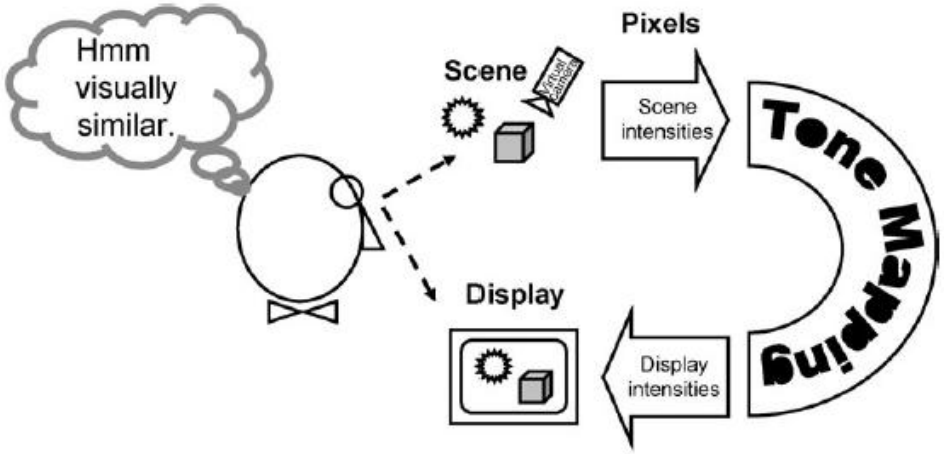


Figure 9: Pictorial outline of tone-mapping problem. The ultimate goal is a visual match between the observed scene and the tone-mapped version of the captured HDR image on the display. Source: Ref. [28].

It is not easy to define and quantify the visual appearance. So the value of HDR image must be reduced to fit realistic display. Maybe simple scaling can be easily to achieve this goal. However, such simple scaling usually produces images with complete loss of detail in the final display. Even though, this simple scaling seems can not solve the problem well, it also give us a apparently simpler problem to resolve: find a method to both compress the dynamic range of HDR images to fit into the range of display

devices and preserve details in images. Algorithms that make HDR images display into LDR display devices are called tone-mapping operators or tone-reproduction operators.

Tone-reproduction operators usually can be classified in four different approaches: global operators, local operators, frequency domain operators, and gradient domain operators. In this project, it is essential and necessary to compare with different Tone-reproduction operators to find out the most suitable and reasonable operator to the application.

6.2 Comparison

6.2.1 Global Operator

Global operators use the whole images as the neighborhood for each pixel, in an image, all pixels are compressing through a common compression curve. As a result, global operators are the simplest operators. Global operators are computationally efficient. Most of them can be executed immediately. Because global operators are much faster and more convenient than any of other type of operators, so most of applications in normal digital cameras are use global operators. On the other hand, if the dynamic range of an image is extraordinary high, the global operators may not have the ability to preserve the whole details like other operators [28]. Nevertheless, the effects are quite different among global operators. Tumblin-Rushmeier Brightness-Preserving Operator [28], Ward Contrast-Based Scale Factor [28], Ward Histogram Adjustment [28] and Reinhard and Devlin Photoreceptor model [28] all have the ability to tone mapping the High Dynamic Range image globally. The effects of these operators are shown in Figure 10. Among these global operators, Reinhard and Devlin Photoreceptor model is the first choice whether the project decide to use global operator in the Tone-reproduction function. Because this global operator could presents the best effect on the final result. Because there are four parameters in this operator and all of these parameters have obvious effects on the output image. Equations about these four parameters are showing as follows.

$$\sigma\left(I_a(x,y)\right) = \left(f\, I_a(x,y)\right)^m$$

$$m = 0.3 + 0.7k^{1.4}$$

$$k = \frac{L_{max} - L_{av}}{L_{max} - L_{min}}$$

$$I_a(x,y) = c\, I_{r|g|b}(x,y) + (1-c)\, L(x,y)$$

$$I_a(x,y) = a\, I_{r|g|b}(x,y) + (1-a)\, I^{av}_{r|g|b}$$

Simply speaking, the value of f is used to control the total luminance of the tone-mapped image, which make the whole appearance darker or lighter. The exponent m is used to affect the overall of impression of contrast. The amount of color correction is controlled by c, the value between 0and 1. Finally, the value of a affect the level of light adaptation.

6.2.2 Local Operator

Local operators would calculate a local adaptation level for every pixel according to the pixel value itself, as same as a neighborhood of pixels around the pixel of interest. This local adaptation level then impels the compression curve for the target pixel. Because the process of compress is partly depend on the neighborhood of a pixel, a dark pixel in a light neighborhood will be handled differently than a dark pixel in a dark neighborhood. The problem of global operators may be loss the information of visibility and contrast can be solved by local operators well. But algorithms of local operators are always much more complicate than global operators. The question for local operators is how many neighborhood pixels required to be contained in the calculation, how to balance contribution of each pixel to the local adaptation level, and how to utilize this adaptation level in the compressing process. These issues have been solved differently be different local operators. In the previous research, I observed some different local operators to try to find out the more adaptive Tone-mapping operator. Rahaman Retinex[28], Fairchild iCAM[28], Pattanik Multiscale Observer Model[28], Ashikhmin Spatially Variant Operator[28] and Reinhard et al. Photographic Tone Reproduction[28] are all in consideration in this project. Among this different types of local operators, Reinhard et al. Photographic Tone Reproduction currently is more suitable than other local operators, the reason is this operator has stronger overall performance than others and it is running time is appropriately.

Photoreceptor Model

Histogram Adjustment

Logarithmic and Exponential Mappings

Revised Tumblin-Rushmeier

Uniform Rational Quantization

Figure 10: the result of images for different global operators. Source: Ref. [27].

6.2.3 Frequency Domain Tone Reproduction

Frequency domain operators selective cut down the dynamic range of image components according to image components spatial frequency. Bilateral filtering and trilateral filtering are two important operators in this approach. Durand Bilateral Filtering is a very significant technique in this area. This technique is an edge-preserving smoothing operator. It divides an image into two layers: base layer and detail layer. The base layer contains high frequency and HDR, and the detail layer is low frequency and LDR. Figure 4 shows examples of an HDR image, a base layer and a detail layer [28]. After that, the bilateral filtering scales the base layer and reduces the dynamic range to a user-specified contrast. Then the bilateral filtering recombines these two layers to create the final output result. Bilateral filtering is a useful technique. It can smooth images without blurring across their sharp edges, which effective reduces outliers and other abnormalities. Choudhury Trilateral Filtering is the extension of bilateral filtering. Bilateral filtering has an obvious disadvantage. If the image contains high gradient and high curvature areas that the filter would poorly smooth it. Trilateral filtering could solve this problem and present a better performance than bilateral filtering. Figure 11 shows the comparison between bilateral filter and trilateral filter. Although these two images look similar, it is easy to discover that the trilateral filter created a better visualization of clouds and waters [28].

6.2.4 Gradient Domain Operators

Gradient domain operators reduce the dynamic range of images according to alter the derivative of images [28]. The main principle of this operator is computing the gradient field to achieve the density of high frequently components, then the algorithm attenuate high frequently components and preserve or even increase the density of low frequently components appropriately to get the new gradient domain, finally the operator use this new gradient domain to compress the new image.

6.2.5 Summary

Global operators are fastest and most convenient. For real time applications, it is the first choice. Local operators have better performances on extreme high dynamic range images, but they also take much more time than global operators. Frequency domain operators have excellent capabilities for images contain specular reflections, directly light sources, and so on. Gradient domain operators are suit to images that need exaggerated small details. In general, every operator has its own advantages and disadvantages, it is impossible to indicate which one is the best and which single operator is suit to all images. Global operators and local operators are based on image formation, while gradient and frequency domain operators operating in the spatial

domain. When we choose an operator, it is necessary to consider its running time, computational complexity, detail lost, and so on.

Figure 11: The input HDR image (left), the HDR base layer (right) and the LDR detail layer (bottom). Source: Ref. [28].

Finally, Reinhard et al. Photographic Tone Reproduction operator is chosen to be the Tone-mapping operator in this project. Generally, it is impossible to determine which operator is the best. Each operator has its own advantages and disadvantages. The reliable method is user chooses the operator according to his requirements and the features of the input image. The reasons I choose Photographic Tone Reproduction operator is this operator has a large amount of user parameters, if we only use the default parameters like phi value, Photographic Tone Reproduction operator offers global tone mapping. Whilst we set other user parameters such as range value, white value, gamma value, upper scale value and lower scale value, then the operator change to be the local operator. So that, Photographic Tone Reproduction can offers different operations to different kinds of images to help users to find out the more suitable and desirable tone mapping method. On the other hand, this operator has stronger overall performance than other operators, and with the help of graphic hardware this operator has the ability to support real time applications.

6.3 Photographic Tone Reproduction

Reinhard et al. created global Tone Reproduction operator and local Tone Reproduction operator. The different between them is local Tone Reproduction operator apply an algorithm like traditional dodging and burning photography. According to Kuang et al. [4] Photographic Tone Reproduction operator has better rendering preference than other operators. So in the following section, this operator will be careful analysis.

The zone system is used to develop this new Photographic Tone Reproduction operator. First, a linear scaling is applied to the target image, which is similar to setting the image's exposure in a digital camera. In order to achieve this goal, it is necessary to calculate the average luminance value of the scene. Reinhard et al. view the log average luminance $\bar{L}_w$ as a useful approximate value to the scene's key [99].

$$\bar{L}_w = \frac{1}{N} \exp\left(\sum_{x,y} \log\left(\delta + L_w(x, y)\right)\right)$$

In this equation, $L_w(x, y)$ is the global luminance for the given pixel (x, y), N stands for the whole number of pixels in the input image andδ is a small defined value to avoid appearing the singularity while black pixels are existed in the input image. After many photographic practice, the log average luminance always to be mapped to 18% of the display range for average key scenes. Then the original scaling equation is:

$$L(x, y) = \frac{a}{\bar{L}_w} L_w(x, y) \qquad (1)$$

In this equation a is a user specified parameter, the default value is 0.18, in this application, the value of parameter a is to be called "key value", the key value can be vary from 0.18 down to 0.0045 and vary it up to 1.00. Figure 12 shows an example of varying. L(x, y) is a luminance after scaling. There is a serious problem in Equation 1. Although many scenes have a main average dynamic range, there are still having some high luminance areas near highlights. In order to solve this problem, unlike traditional transfer functions, modern photography uses a new compressive function to solve this issue.

$$L(x, y) = \frac{L(x,y)}{1+L(x,y)} \qquad (2)$$

This function linear scales small values, while high luminances are compressed by huge amounts. All possible values in this function are between 1 and 0, in practical

terms, the maximal display luminance is impossible to reach 1. On the side, because high luminances sometimes may uncontrollable and undesirable burn out in Equation 2, that it need to be extended to be a controllable function:

$$L_d(x, y) = \frac{L(x,y)\ (1+\frac{L(x,y)}{L_{white}^2})}{1+L(x,y)} \quad (3)$$

The equation 3 is the algorithm of the global Photographic Tone Reproduction operator. In this equation, a new parameter Lwhite is denoted the smallest luminance value which will be mapped to pure white. By default, if the value does not set infinity, the value of Lwhite is set to the maximum luminance in the scene. While tone mapping a low dynamic range image, the Lwhite value in this operator is using to reduce the loss of contrast to keep the image more reality.

In previous steps, the paper introduced the principle of the global Photographic Tone Reproduction operator. In the following section, it will explain the theory of the local Photographic Tone Reproduction operator. Nevertheless, the most different between global tone reproduction operator and local tone reproduction is local tone reproduction adds an algorithm to simulate dodging and burning in traditional photography to present a tone reproduction technique for HDR image rendering. The local tone reproduction operator chooses a key value from each pixel, and finds the largest surrounding region that does not have any sharp contrast. Reinhard et al. utilize a center around function that contains Gaussians to decide the suitable surrounding area (Equation 4). If there is no obvious contrast around the pixel, the Gaussians around the pixel are similar. Otherwise, these two Gaussians will different significantly.

$$L_S^{blur}(x, y) = L\ (x, y) \ \otimes \ R_s(x, y)$$

$$V_s(x, y) = \frac{L_S^{blur} - L_{S+1}^{blur}}{L^{\varphi}\frac{a}{s^2} + L_S^{blur}} \quad (4)$$

a b

c d

e f

Figure 12: Key value varies from 0.09 to 0.72. a is 0.045 b is 0.09, c is 0.18, d is 0.36 , e is 0.72 and f is 1.00.

In this equation, φ is a user parameter that act as a sharpening parameter, in the application, we name it as Phi value, the effect of Phi value is displayed in Figure 13. If the Phi value is small, the effect is quite detailed. While the Phi value is huge, the output image may have haloing artifacts. In our application, the default value is 1.0. However, after experiments, the value of 8 is advised to be used. In this formulation, V_s is now independent, which will not under the influence of absolute luminance values.

Then the process produces some different Gaussians to find the largest surrounding area S_{max}. Once S_{max} is confirmed, the global Tone Reproduction operator Equation 3 can be converted to a local Tone Reproduction operator Equation 5.

$$L_d(x, y) = \frac{L(x,\ y)}{1+L_{S_{max}}^{blur}(x,\ y)} \quad (5)$$

The luminance of a bright pixel in a relatively dark region will content $L > L_{S_{max}}^{blur}$, so this operator will increase the display luminance $L_d(x, y)$, as the result of decreasing the contrast effect at that pixel. This is the "burning", equally, a dark pixel in a relatively bright will be compressed more, and this is "dodging". Either way the pixel's contrast effect involved in the effecting scope is increased [28].

In summary, the Photographic Tone Reproduction operator has both local and global forms. The global tone reproduction operator is more fast and efficient for medium dynamic range images. However, for extreme high dynamic range images the local tone reproduction operator is more available and suitable. An example of different rendering effects between global and local tone reproduction operator is shown in Figure 14. In this application, users can set different arguments to adjust the operator to determine either local form or global form is used. The operator toolbar is shown in Figure 15. In this toolbar, user could set five parameters to the operator: Key value defines the log average luminance, Phi value is the sharpen parameter, it is default value is 1 that means no sharpening, Use Scale parameter decides whether the operator need advanced version or not, it is default value is o that means the application uses the based version, Range value is the number of scales that will be used in the computation, Lower Scale value defines the size in pixels of smallest scale, similarly, Upper Scale value defines the size in pixels of largest scale.

Figure 13: The sharpening parameterφ in the Photographic Tone Reproduction operator is chosen to be 1.0, 4.0, 8.0, 16.0 and 32.0 (from left to right and from top to bottom). (f) is the original image.

Global Photographic Tone Reproduction

Local Photographic Tone Reproduction

Figure 14: HDR images. Source: Ref. [27].

Figure 15: the Photographic Tone Reproduction operator toolbar.

6.4 Summary

Global operators are fastest and most convenient. For real time applications, it is the first choice. Local operators have better performances on extreme high dynamic range images, but they also take much more time than global operators. Frequency domain operators have excellent capabilities for images contain specular reflections, directly light sources, and so on. Gradient domain operators are suit to images that need exaggerated small details.

In general, every operator has its own advantages and disadvantages, it is impossible to indicate which one is the best and which single operator is suit to all images. Global operators and local operators are based on image formation, while gradient and frequency domain operators operating in the spatial domain. When we choose an operator, it is necessary to consider its running time, computational complexity, detail lost, and so on.

After research and exploration, Reinhard et al Photographic Tone Reproduction Operator becomes the project's tone mapping operator. The reason is this operator has stronger overall performance than other operators, and the running time is desirable. More important, with the help of graphic hardware this operator has the ability to support real time applications. The final result has demonstrated this operator has well performance to the High Dynamic Range image tone mapping operation

7.Experiment Results

In this section varieties of drawing results will be shown and all of these results are using the project system. After that, I will compare my method to other methods to find out my system's advantages and disadvantages. All the coming results are generated with 32-bits RGBE file format. All algorithms used in this system are using the default values.

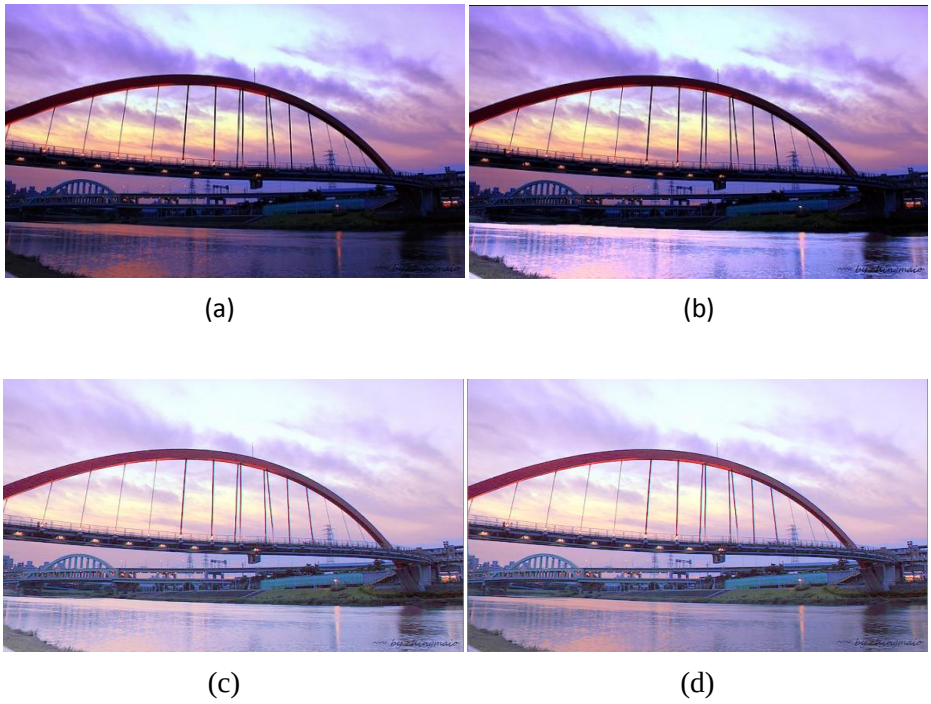
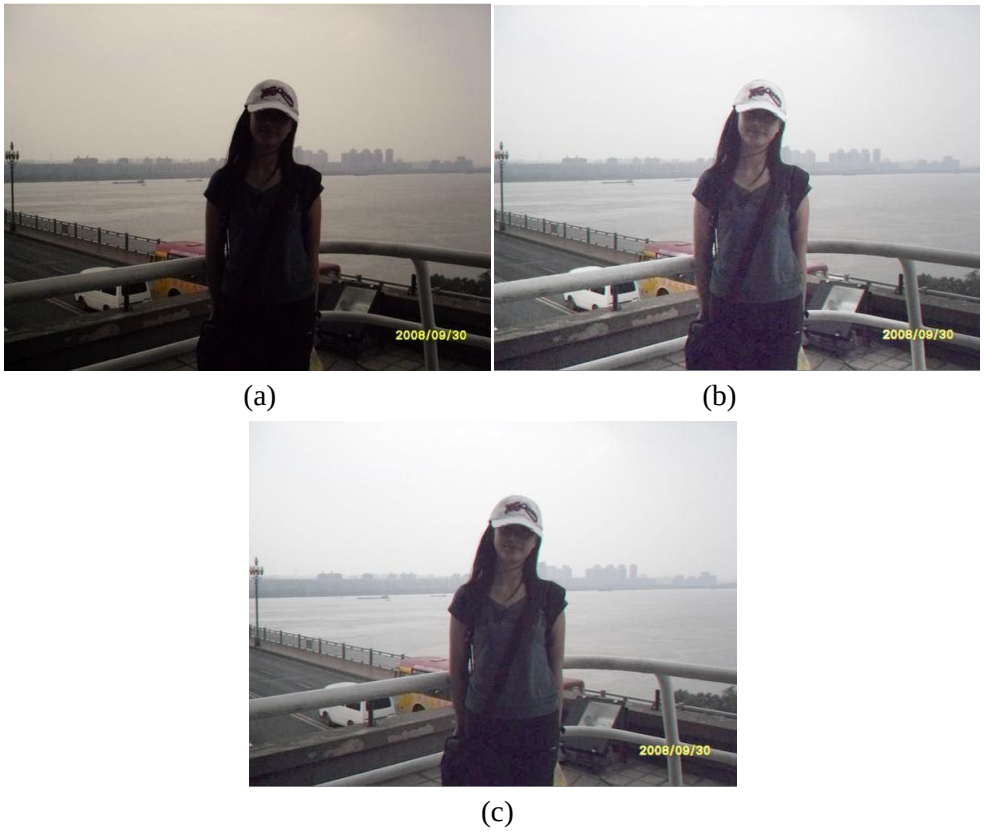


Figure 16: (a) the input image in the first canvas, (b) the modified image in the second canvas, (c) the output HDR image, (d) the HDR image that using global tone mapping operator.

The first example is demonstrated in the Figure 16, because the input image is JPG format (Figure 16a) and it miss the information of exposure, in order to generate the High Dynamic Range image successfully, the input image need to be given a default exposure value, where exposure = 1.6, so the output image (Figure 16c&d) will look a bit of brighter than we expect. In this example, we set the river as the constraint area in the first canvas. Then, we edit the image to make the river brighter in the second canvas (Figure 16b). Afterwards, we use RGBE function to generate the High Dynamic Range image and use Reinhard et al Photographic Tone Reproduction operator to tone mapping the High Dynamic Range image to make it displayable on the screen (Figure 16c). Compare with the image b and the image c in Figure 16, we could find that the details in the image b is much more clear than the image c. the river in the image b is much brighter and more clear. The reason is our method not

only using the operator to global adjustment image, but also using the constraint function to editing the input image partially.



(a) (b)

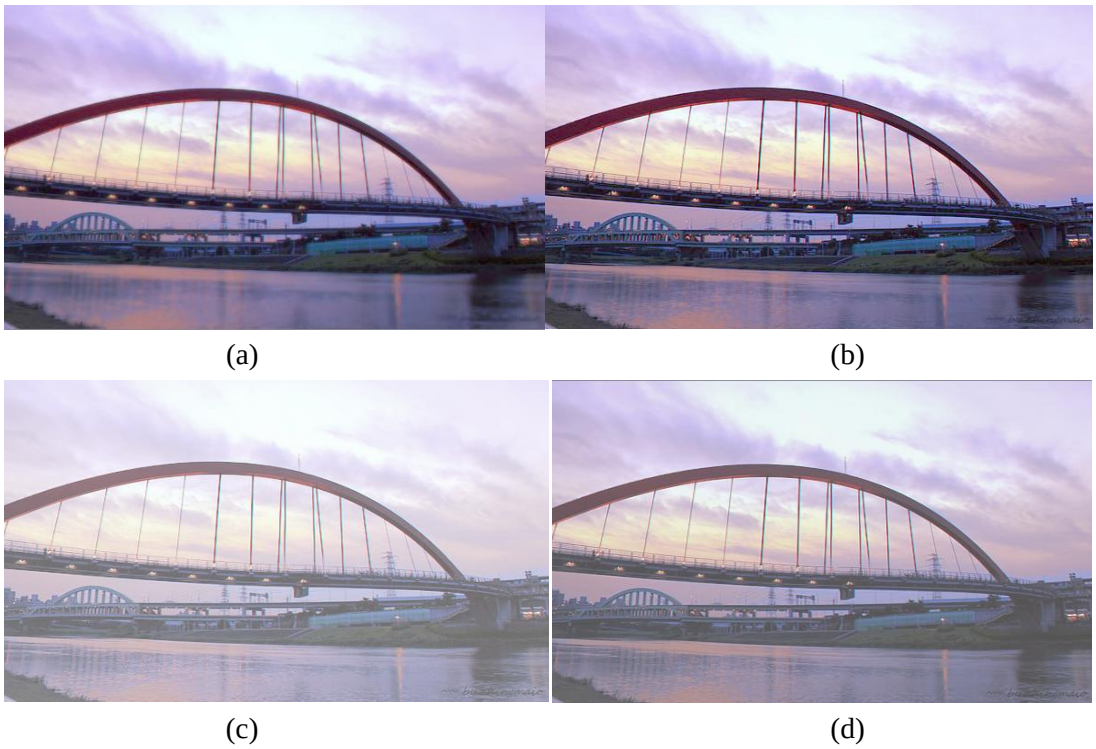
(c)

Figure 17: (a) the original image, (b) our result, (c) globally adjusted image.

The second example is shown in the Figure 17. We select the person in the image as the constraint area and editing the image to create the output image. We could find out that due to the single Low Dynamic Range image always does not has enough information and parameters to generate the High Dynamic Range image, so the output image which using the RGBE function and the Photographic Tone Reproduction operator will add some parameters to the image, thus the result not only improve the quality of the scope that we paint but also globally adjust the quality of the final output image (Figure 17b). Personal speaking, this measure has both advantage and disadvantage. The advantage is it is also improve the image quality globally and the disadvantage is it is difficult for someone to partial keep effect in the original image. In future, I will try to find out the solutions for this problem.

In this application, we not only could adjust the brightness, we also could use the Gaussian blur and other drawing tools to global editing the image to generate the 32-bits RGBE file format image, like Figure 18 shows.

It is quite important to compare our results with other global adjusting methods. The Comparison will help us to discover our method's advantages and disadvantages. Compared with the image c and d in Figure 16, both of these two images are using the Reinhard et al Photographic Tone Reproduction operator to generate the High Dynamic Range image, the only difference between them is the image c using the constraint function to let the river much brighter before the Tone mapping operating. Hence, it is easy to perceive the image c has better performance than the image d, the river in the image c is brighter and clearer, the contrast effect in this image is also better and more intensity. The image b and c in the Figure 17 also certify our method has the ability to perform the local adjustment, the person in the image become clear and the Figure 19 demonstrates the face feature between the image b and c in the Figure 17. The person in the right image is clear to see, the five sense organs are easier distinguish than the left image.



(a) (b)

(c) (d)

Figure 18: (a) global Gaussian Blur, (b) only Gaussian Blur the river in the image, (c) change the global opacity value, (d) changed the global contrast value.

Figure 19: Left: the face feature that generated by the project application. Right: the face feature that generated by globally adjusting application.

8.Conclusion

In this project, we have developed an application that allow user to load a image in the first canvas, then the filter in this application could provide constraints in the second canvas. While user use painting tools to edit the image in the second canvas, the image features present in the first canvas will spatially affect the painting effects in the second canvas. After painting, the original image and the modified image will be recombined to generate a new High Dynamic Range image.

To design the user interface, we use the Qt as the development platform. Qt provides a large amount of classes and library to allow us to design a user friendly interface. In this part, we also use these classes and libraries to develop some editing tools. The OpenGL library plays a significant role in the Gaussian Blur function, the Brightness function, the Opacity function and the Contrast function.

After that, in the constraint step, after experiments and research, we use an existing algorithm to achieve the goal. There are three major steps in this algorithm: path generation, constraint propagation and scattered bilateral interpolation [99].

Then, after the image editing, it is necessary to recombine the input image and the editing image to generate a new High Dynamic Range image. In this step, we wrote a RGBE function base on the Greg Ward's theory [33]. Firstly, we use the information of the input image to create a new image header. Secondly, we use a dynamic array to convert all pixels' RGB values to RGBE values pixel by pixel. Finally, we use these RGBE values to create the new radiance image.

The pixels' values in new High Dynamic Range image always totally out of the scope that the ordinary screen, monitor, or printer could display, so we use the Reinhard et al Photographic Tone Reproduction [88] to reduce the pixel value and maintain the contrast effect as much as possible to generate a new image, which is displayable on the ordinary screen, monitor and print. And this new image is the final output image. Although, the output image is a Low Dynamic Range image, it maximum preserve the initial High Dynamic Range image effects. Photographic Tone Reproduction operator has a large amount of user parameters so that user could well adjust the image quality by setting different values of user parameters.

Now the application basically satisfy the project demands, it has two canvases, provides constraints filter, allows user editing the image and uses two images to generate the new High Dynamic Range image. The application could locally adjust the image to make it has better effect than the normal global image adjustment software. The computational costs for local image adjustment and the High Dynamic

Range image tone mapping operation are acceptable and reasonable. The method is efficient and effective enough to run on the basic configuration laptop and computer.

9.Future Work

In the future, I am trying to extending the application to let it has more functions and its user interface become easier to use.

In the User Interface design, I will try to use icon buttons instead of all word buttons, it will make the user interface more user-friendly and much easier for novice user, and these users could easily operate the application to complete the image editing and High Dynamic Range image generating. Also, I am looking forward to increase the types of image that could load and display in the system, in future, the system not only could load and display Low Dynamic Range image, but also could load and display other image formats, like exr, hdr, pic and so on.

On the other hand, I am expecting developing the algorithm to support numerous constraints, these constraints could implement through point or stroke based operation. It is also possible to explore the feasibility to add more sophisticated interpolation algorithms to instead of the influence between spatial support and gradient [99]. In the tone mapping part, I am going to develop more tone reproduction operators to allow user to choose their favorite operator. Because every operator has its own advantages and disadvantages, it is impossible to indicate which one is the best and which single operator is suit to all images. Novice users could try different types of operators and compare result to find out the most suitable operator. These improvements will make image editing both easier and more fun.

Bibliography

[1] ADAMS, A., GELFAND, N., DOLSON, J., LEVOY, M. 2009. Gaussian kd-trees for fast high-dimensional filtering. ACM Trans. Graph. 28, 3 (2009), 21:1 – 21:12.

[2] ADOBE SYSTEMS, INC. 2009. Adobe Photoshop CS4. Adobe Systems, Inc., San Jose, CA.

[3] AKYUZ, A., FLEMING, R., RIECKE, B., REINHARD, E., BULTHOFF, H. 2007. Do HDR Displays Support LDR Content? A Psychophysical Evaluation. ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2007. 26(3).

[4] AN, X., PELLACINI, F. 2008. AppProp: all-pairs appearancespace edit propagation. ACM Trans. Graph. 27, 3 (2008), 40:1 – 40:9

[5] BAE, S., PARIS, S., DURAND, F. 2006. Two-scale tone management for photographic look. ACM Trans. Graph. 25, 3 (2006), 637 – 645.

[6] BAI, X., SAPIRO, G. 2007 A geodesic framework for fast interactive image and video segmentation and matting. Proc. ICCV (2007).

[7] CHEN, J., PARIS, S., DURAND, F. 2007. Real-time edge-aware image processing with the bilateral grid. ACM Trans. Graph. 26, 3 (2007), 103:1 – 103:10.

[8] COLBERT, M., REINHARD, E., AND HUGHES, C.E. 2007. Painting in High Dynamic Range. Journal of VISUAL Communication & IMAGE Representation, 18(2007), pp. 387-396.

[9] CRIMINISI, A., SHARP, T., BLAKE, A. 2008. Geos: Geodesic image segmentation. In Proc. 10th European Conference on Computer Vision (2008), Springer-Verlag, pp. 99 – 112.

[10] DURAND, F., DORSEY J. 2002. Fast bilateral filtering for the display of high-dynamic-range images. ACM Trans. Graph. 21, 3 (2002), 257 – 266.

[11] EISEMANN, E., DURAND, F. 2004. Flash photography enhancement via intrinsic relighting. ACM Trans. Graph. 23, 3 (2004), 673 – 678.

[12] FARBMAN, Z., FATTAL, R., LISCHINSKI, D., SZELISKI, R. 2008. Edge-preserving decompositions for multi-scale tone and detail manipulation. ACM Trans. Graph. 27, 3 (2008), 67:1 – 67:10.

[13] FATTAL, R., LISCHINSKI, D., WERMAN, M. 2002. Gradient domain high dynamic range compression. ACM Trans. Graph. 21, 3 (2002), 249 – 256.

[14] GELFAND, N., ADAMS, A., PARK, S.H., AND PULLI, K., 2010. Multi-exposure Imaging on Mobile Devices. In Proceedings of ACM Multimedia'2010. pp.1595-1598.

[15] KUANG, J., YAMAGUCHI, H., LIU, C., JOHNSON, G.M., AND FAIRCHILD, M.D. 2007. Evaluating HDR Rendering Algorithms. ACM Transactions on Applied Perception, 4(2), ACM New York, NY, USA.

[16] LI, Y., ADELSON, E. H., AGARWALA, A. 2008 Scribble-Boost: adding classification to edge-aware interpolation of local image and video adjustments. Computer Graphics Forum 27, 4 (2008), 1255 – 1264.

[17] LI, Y., SHARAN, L., AND ADELSON, E. H. 2005. Compressing and companding high dynamic range images with subband architectures. ACM Trans. Graph. 24, 3, 836 – 844.

[18] LIANG, C.K., CHEN, W.C., AND GELFAND, N. 2010. TouchTone: Interactive Local Image Adjustment Using Point-and-Swipe. Comput. Graph. Forum(2010) 253-261

[19] LISCHINSKI, D., FARBMAN, Z., UYTTENDAELE, M., SZELISKI, R. 2006. Interactive local adjustment of tonal values. ACM Trans. Graph. 25, 3 (2006), 646 – 653.

[20] LEVIN, A., LISCHINSK,I D., WEISS, Y. 2004. Colorization using optimization. ACM Trans. Graph. 23, 3 (2004), 689 – 694.

[21] LEVIN, A., LISCHINSKI, D., WEISS, Y. 2008. A closed-form solution to natural image matting. IEEE Trans. PAMI 30, 2 (2008), 228 – 242.

[22] MASIA, B., AUUSTIN, S., FLEMING, R.W., SORKINE, O., AND GUTIERREZ, D. 2009. Evaluation of Reverse Tone Mapping Through Varying Exposure Conditions. ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2009. 28(5).

[23] PARIS, S., DURAND ,F. 2009 A fast approximation of the bilateral filter using a signal processing approach. IJCV 81, 1 (2009), 24 – 52.

[24] PATTANAIK, S. N., FERWERDA, J. A., FAIRCHILD, M. D., AND GREENBERG, D. P. 1998. A multiscale model of adaptation and spatial vision for realistic image display. In Proc. ACM SIGGRAPH 98, M. Cohen, Ed., 287 – 298.

[25] Qt Reference Document. http://doc.trolltech.com/4.7/tutorials.html

[26] REINHARD, E., DEVLIN, K. 2005. Dynamic Range Reduction Inspired by Photoreceptor Physiology. IEEE Transactions on Visualization and Computer Graphics, 11(1), pp. 13–24.

[27] REINHARD, E., STARK, M., SHIRLEY, P., AND FERWERDA, J. 2002. Photographic tone reproduction for digital images. ACM Transactions on Graphics. 21(3). pp. 267–276.

[28] REINHARD, E., WARD, G., PATTANAIK, S., AND DEBEVEC, P. 2005. High dynamic range imaging: acquisition, display, and image-based lighting, Morgan Kaufmann.

[29] REINHARD, E. 2002. Parameter estimation for photographic tone reproduction. Journal of Graphics Tools 7, 1, 45 – 52.

[30] TUMBLIN, J., AND TURK, G. 1999. LCIS: A boundary hierarchy for detail-preserving contrast reduction. In Siggraph 1999, Computer Graphics Proceedings, Addison Wesley Longman, Los Angeles, A. Rockwood, Ed., Annual Conference Series. pp. 83 – 90.

[31] TUMBLIN, J., HODGINS, J. K., AND GUENTER, B. K. 1999. Two methods for display of high contrast images. ACM Transactions on Graphics 18 (1), 56–94.

[32] WARD, G., RUSHMEIER, H., AND PIATKO, C. 1997. A visibility matching tone reproduction operator for high dynamic range scenes. IEEE Transactions on Visualization and Computer Graphics 3, 4 (December).

[33] WARD, G. 1991. Real pixels. Graphics Gems II, James Arvo (ed.), Academic Press, pp. 80-83.

[34] WANG, J., COHEN, M.F. 2005. An iterative optimization approach for unified image segmentation and matting. Proc. ICCV (2005), 936 – 943.

[35] YATZIV, L., SAPIRO, G. 2006. Fast image and video colorization using chrominance blending. IEEE Trans. Image Processing 15, 5 (5 2006), 1120 – 1129.

Appendix A: important algorithms

Convert2RGBE class:

```
/*This code was written based on Greg Ward's code*/
#include "convert2rgbe.h"
#include <math.h>
#include <malloc.h>
#include <string.h>
#include <ctype.h>
#include <QDebug>
int writeHeader(FILE *file, int width, int height, header_information *information)
{
    char *imageType = "RGBE";
    if(information &&(information->valid & 0x01))
        imageType = information->type;
    if(information && (information->valid & 0x02))
        {
            if(fprintf(file, "GAMMA = %g\n", information->gammaValue) < 0)
                return false;
        }
    if(information && (information->valid & 0x04))
        {
            if(fprintf(file, "EXPOSURE=%g\n", information->exposureValue)< 0)
                return false;
        }
    if(fprintf(file, "FORMAT=32-bit_rle_rgbe\n\n") < 0)
        return false;
    if(fprintf(file,"-Y %d +X %d\n", height, width) < 0)
        return false;
    return 1;
}
static inline void floatToRGBE(unsigned char RGBE[4], float red, float green, float blue)
{
    float value;
    int exposure;
      value = red;
    if(green > value)
        value = green;
    if(blue > value)
            value = blue;
```

```
if(value < 1e-32)
        {
            for(int i = 0; i < 4; i++)
                RGBE[i] = 0;
        }
    else
        {
            value = frexp(value, &exposure) * 256.0/ value;
            RGBE[0] = (unsigned char)(red * value);
            RGBE[1] = (unsigned char)(green * value);
            RGBE[2] = (unsigned char)(blue * value);
            RGBE[3] = (unsigned char)(exposure + 128);
        }
}
int RGBEPixels(FILE *file, float *image, int totalPixels)
{
    unsigned char RGBE[4];
    while(totalPixels-- > 0)
    {
        floatToRGBE(RGBE, image[1], image[2], image[0]);
        image += 3;
        if(fwrite(RGBE, sizeof(RGBE), 1, file) < 1)
            return false;
    }
  return 1;
}
```

Reinhard et al Tone Reproduction Operator algorithm:

```
/* Copyright (c) University of Utah / Erik Reinhard / October 2001 */
double bessel (double x)
{
  const double f = 1e-9;
  int          n = 1;
  double       s = 1.;
  double       d = 1.;
  double t;
  while (d > f * s)
  {
    t = x / (2. * n);
    n++;
    d *= t * t;
    s += d;
  }
  return s;
}
void compute_bessel ()
{
  bbeta = bessel (M_PI * alpha);
}
```

```
double kaiserbessel (double x, double y, double M)
{
   double d = 1. - ((x*x + y*y) / (M * M));
   if (d <= 0.)
      return 0.;
   return bessel (M_PI * alpha * sqrt (d)) / bbeta;
}
void Yxy_rgb ()
{
   int            x, y, i, j;
   double         result[3];
   double         X, Y, Z;
   for (x = 0; x < cvts.xmax; x++)
      for (y = 0; y < cvts.ymax; y++)
      {
         Y              = image[y][x][0];
         result[1] = image[y][x][1];
         result[2] = image[y][x][2];
         if ((Y > 0.) && (result[1] > 0.) && (result[2] > 0.))
         {
            X = (result[1] * Y) / result[2];
            Z = (X/result[1]) - X - Y;
         }
         else
            X = Z = 0.;
         image[y][x][0] = X;
         image[y][x][1] = Y;
         image[y][x][2] = Z;
         result[0] = result[1] = result[2] = 0.;
         for (i = 0; i < 3; i++)
            for (j = 0; j < 3; j++)
               result[i] += XYZ2RGB[i][j] * image[y][x][j];
         for (i = 0; i < 3; i++)
            image[y][x][i] = result[i];
      }
}
```

```
double get_maxvalue ()
{
   double max = 0.;
   int       x, y;
   for (y = 0; y < cvts.ymax; y++)
      for (x = 0; x < cvts.xmax; x++)
         max = (max < image[y][x][0]) ? image[y][x][0] : max;
   return max;
}
void tonemap_image ()
{
   double Lmax2;
   int       x, y;
   int       scale, prefscale;
   if (white < 1e20)
      Lmax2 = white * white;
   else
   {
      Lmax2    = get_maxvalue();
      Lmax2 *= Lmax2;
   }
   for (y = 0; y < cvts.ymax; y++)
      for (x = 0; x < cvts.xmax; x++)
      {
         if (use_scales)
         {
            prefscale = range - 1;
            for (scale = 0; scale < range - 1; scale++)
               if (fabs (ACTIVITY(x,y,scale)) > threshold)
               {
                  prefscale = scale;
                  break;
               }
            image[y][x][0]/=1. + V1(x,y,prefscale);
         }
```

```
else
            image[y][x][0] = image[y][x][0] * (1. +
(image[y][x][0]/Lmax2))/(1. + image[y][x][0]);
        //image[y][x][0]/= (1. + image[y][x][0]);
      }
}
void clamp_image ()
{
   int x, y;
   for (y = 0; y < cvts.ymax; y++)
      for (x = 0; x < cvts.xmax; x++)
      {
         image[y][x][0] = (image[y][x][0] > 1.) ?
1. : image[y][x][0];
         image[y][x][1] = (image[y][x][1] > 1.) ?
1. : image[y][x][1];
         image[y][x][2] = (image[y][x][2] > 1.) ?
1. : image[y][x][2];
      }
}
double log_average ()
{
   int       x, y;
   double sum = 0.;
   for (x = 0; x < cvts.xmax; x++)
      for (y = 0; y < cvts.ymax; y++)
         sum += log (0.00001 + luminance[y][x]);
   return exp (sum / (double)(cvts.xmax *
cvts.ymax));
}
void scale_to_midtone ()
{
   int       x, y, u, v, d;
   double factor;
   double scale_factor;
   double low_tone       = key / 3.;
   int    border_size = (cvts.xmax < cvts.ymax) ?
cvts.xmax / 5. : cvts.ymax / 5.;
   int    hw= cvts.xmax >> 1;
int    hh = cvts.ymax >> 1;
   scale_factor = 1.0 / log_average ();
   for (y = 0; y < cvts.ymax; y++)
      for (x = 0; x < cvts.xmax; x++)
      {
         if (use_border)
         {
            u = (x > hw) ? cvts.xmax - x : x;
            v = (y > hh) ? cvts.ymax - y : y;
            d = (u < v) ? u : v;
            factor = (d < border_size) ? (key -
low_tone) * kaiserbessel (border_size - d, 0,
border_size) +low_tone : key;
         }
         else
         factor               = key;
         image[y][x][0]    *= scale_factor * factor;
         luminance[y][x] *= scale_factor * factor;
      }
}
void copy_luminance ()
{
   int x, y;
   for (x = 0; x < cvts.xmax; x++)
      for (y = 0; y < cvts.ymax; y++)
         luminance[y][x] = image[y][x][0];
}
void allocbufs ()
{
   int rsiz = CHK(C_RFLT) ? sizeof(COLOR) :
sizeof(COLR);
   cvts.r.p = (char *)malloc(rsiz*cvts.xmax);
}
void allocate_memory ()
{
   int y;
   luminance = (double **) malloc (cvts.ymax *
sizeof (double *));
   image = (COLOR **) malloc (cvts.ymax *
sizeof (COLOR *));
```

```
for (y = 0; y < cvts.ymax; y++)
   {
      luminance[y] = (double *) malloc
(cvts.xmax * sizeof (double));
      image[y] = (COLOR *) malloc (cvts.xmax
* sizeof (COLOR));
   }
}
void open_out_file ()
{
   if (!(outfp = fopen (outfilename, "w")))
      fprintf (stderr, "Cannot open output
file %s\n", outfilename);
}
void write_header ()
{
   fputc('\n', outfp);
   fputresolu(pixorder(), (int)cvts.xmax,
(int)cvts.ymax, outfp);
}
void write_image ()
{
   int x, y;
   if (write_ppm)
   {
      fprintf(outfp, "%d %d\n%d\n", cvts.xmax,
cvts.ymax, 255);
      for (y = 0; y < cvts.ymax; y++)
         for (x = 0; x < cvts.xmax; x++)
            fprintf(outfp, "%c%c%c",
                  (char)(255. * pow
(image[y][x][0], 1./gammaval)),
                  (char)(255. * pow
(image[y][x][1], 1./gammaval)),
                  (char)(255. * pow
(image[y][x][2], 1./gammaval)));
      fprintf(outfp, "\n");
   }
   else
      for (y = 0; y < cvts.ymax; y++)
         fwritescan (image[y], cvts.xmax, outfp);
}
void open_in_file ()
{
   if (!(infp = fopen (infilename, "rb")))
      fprintf (stderr, "Cannot open input
file %s\n", infilename);
}
void read_image ()
{
   int y;
   allocate_memory ();
   for (y = 0; y < cvts.ymax; y++)
      freadscan (image[y], cvts.xmax, infp);
}
long getint(int siz, FILE *fp)
{
   register long c;
   register long r = 0;
   int s = 0;
   while (--siz > 0)
   {
      if ((c = getc(fp)) == EOF)
         return(EOF);
      c <<= (s*8);
      r  |= c;
      s++;
   }
   if ((c = getc(fp)) == EOF)
      return(EOF);
   c    = 0x80&c ? -1<<8|c : c;
   c <<= (s*8);
   r  |= c;
   return(r);
}
void read_input_file ()
{
   int i1, po;
      open_out_file ();
      if (!write_ppm)
      {
         open_in_file ();
         getheader (infp, NULL, NULL);
         fclose (infp);
      }
```

```
open_in_file                                    ();
CLR(C_RFLT|C_TFLT|C_XYZE|C_PRIM|C_GAMMA
|C_CXFM);
      SET(C_RFLT);
      cvts.stonits = cvts.pixrat    = 1.;
      getheader              (infp, NULL, NULL);
      po = fgetresolu (&width, &height, infp);
      cvts.xmax = width;
      cvts.ymax = height;
      for (i1 = 0; i1 < 8; i1++)
         if (ortab[i1] == po)
         {
            cvts.orient = i1 + 1;
            break;
         }
      if (!(po & YMAJOR))
         cvts.pixrat = 1./cvts.pixrat;
      if (!CHK(C_XYZE))
         cvts.stonits *= WHTEFFICACY;
      allocbufs                    ();
      if (!write_ppm)
         write_header           ();
      read_image                  ();
   }
}
void dynamic_range ()
{
   int x, y;
   double minval =    1e20, maxval = -1e20;
   for (x = 0; x < cvts.xmax; x++)
      for (y = 0; y < cvts.ymax; y++)
      {
         if ((luminance[y][x] < minval) &&
               (luminance[y][x] > 0.0))
            minval = luminance[y][x];
         if (luminance[y][x] > maxval)
            maxval = luminance[y][x];
       }
   fprintf (stderr, "\tRange of values    = %9.8f
- %9.8f\n", minval, maxval);
   fprintf      (stderr,      "\tDynamic      range
= %i:1\n", (int)(maxval/minval));
}
void Reinhard02(const QString &fileName,
float _key, float _phi, int _range, int _lower,
int _upper, bool _usescale, QImage _sImage)
{
      QByteArray ba =fileName.toLatin1();
      const    QString    &outFileName    =
"demo.ppm";
      QByteArray              bb              =
outFileName.toLatin1();
      strcpy(outfilename,bb.data());
      strcpy(infilename, ba.data());
        ::key =(double) _key;
        ::phi =(double) _phi;
        ::range = _range;
        ::scale_low = _lower;
        ::scale_high = _upper;
        ::use_scales = _usescale;
      sigma_0    = log ((double)scale_low);
      sigma_1    = log ((double)scale_high);
      compute_bessel                 ();
      read_input_file               ();
      fclose                        (infp);
      rgb_Yxy                        ();
      copy_luminance                 ();
      scale_to_midtone               ();
      if( use_scales )
      {
         build_pyramid(luminance,
cvts.xmax, cvts.ymax);
      }
      tonemap_image                  ();
      Yxy_rgb                        ();
      clamp_image                    ();
      write_image                    ();
      fclose                        (outfp);
      print_parameter_settings       ();
      diplay_new_image(outFileName);
}
```

Mianwindow class:

```
MainWindow::MainWindow()
 {
        canvasScene = new Canvas;
        newCanvasScene = new Canvas;
        view = new QGraphicsView;
        newView = new QGraphicsView;
        createAction();
        createMenu();
        createToolBar();
        view->setBackgroundRole(QPalette::Base);
        view->setSizePolicy(QSizePolicy::Ignored,
QSizePolicy::Ignored);
        view->setScene(canvasScene);
        view->resize(500,400);
        view->setWindowTitle("The Second
Canvas");
        view->show();
        newView->setBackgroundRole(QPalette::B
        ase);
   newView->setSizePolicy(QSizePolicy::Ignored,
QSizePolicy::Ignored);
        newView->setScene(newCanvasScene);
        newView->resize(500,400);
        newView->setWindowTitle("The First
Canvas");
        newView->show();
        rKeyValue = 0.19;
        rPhi = 8.00;
        useScales = 0;
        rRange = 8;
        rLowerScale = 1;
        rUpperScale =43;
 }
void MainWindow::penBrush()
 {
    QColor brush =
QColorDialog::getColor(canvasScene->penColor(
));
    if (brush.isValid())
    {
         canvasScene->setPenBrush(brush);
       }
void    MainWindow::gaussianFilter()
 {
        bool status;
        int radius =
QInputDialog::getInteger(this, tr("Canvas"),
tr("Select gussian radius:"),
2, 1, 50, 1, &status);
        if(status)
        {
        QGraphicsBlurEffect *effectBlur = new
QGraphicsBlurEffect(this);
        effectBlur->setBlurRadius(radius);
        item2->setGraphicsEffect(effectBlur);
        item2->update();
        }
 }
void    MainWindow::OpactiyFilter()
 {
        bool status;
        double opacity =
QInputDialog::getDouble(this,tr("Canvas"),
tr("Select Opacity Value"),
0.1,0.0,1.0,1,&status);
        if(status)
        {
        QGraphicsOpacityEffect *effectOpacity =
new QGraphicsOpacityEffect(this);
        effectOpacity->setOpacity(opacity);
        item2->setGraphicsEffect(effectOpacity);
        item2->update();
        }
 }
void MainWindow::brightImage()
{
        bool status;
        int brightvalue =
QInputDialog::getInteger(this,tr("Tone
Map"),tr("set bright value"),0,0,100,1,&status);
        if(status)
             brightness = brightvalue;
             sImage = changeBrightness(sImage,
brightness);
```

```
        item2=canvasScene->addPixmap(QPixma
p::fromImage(sImage));
        view->setScene(canvasScene);
        view->adjustSize();
}
void MainWindow::constrastImage()
 {
        bool status;
        int                constrastvalue                =
QInputDialog::getInteger(this,tr("Tone
Map"),tr("set bright value"),0,0,100,1,&status);
     if(status)
        constrast = constrastvalue;
     sImage= changeContrast(sImage,constrast);
      item2=canvasScene->addPixmap(QPixmap::
      fromImage(sImage));
     view->setScene(canvasScene);
     view->adjustSize();
 }
void MainWindow::applyClicked()
 {
       const QString tempImage = "pic.jpg";
        QPainter painter1(&sImage);
        canvasScene->render(&painter1);
        sImage.save(tempImage,"jpg");
        QImage tempsImage(tempImage);
        QPainter painter2(&image);
        newCanvasScene->render(&painter2);
        int width = tempsImage.width();
        int height = tempsImage.height();
        int x,y;
        static float         *image2;
        static char          testfilename[1024];
        image2=(float*)malloc(width*height*sizeo
f(float)*3);
        const QString &testFileName = "test1.hdr";
        QByteArray bc = testFileName.toLatin1();
        strcpy(testfilename, bc.data());
        FILE *outfp = fopen(testfilename,"wb");
        writeHeader(outfp,width,height,NULL);
        for (y = 0; y < height; y++)
           for (x = 0; x < width; x++)
{
        QRgb value = tempsImage.pixel(x,y);
        QRgb value2 = image.pixel(x,y);
        int z = width*y+x;
       image2[z*3+0]=((float)qRed(value)+(
       float)qRed(value2));
       image2[z*3+1]=((float)qGreen(value)
       +(float)qGreen(value2));
       image2[z*3+2]=((float)qBlue(value)+
       (float)qBlue(value2));
}
  RGBEPixels(outfp,image2,width*height);
 fclose(outfp);
 const QString tempFile = testFileName;
 Reinhard02(tempFile, rKeyValue, rPhi,
rRange,rLowerScale,rUpperScale,useScales,
tempsImage);
 }
```

Appendix B: poster

Drawing with constraints

University of BRISTOL

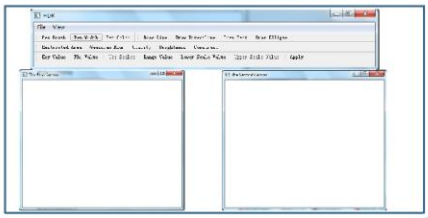
Feng Wang
***Supervisor:* Dr. Erik Reinhard**
Department of Computer Science

Introduction and Objectives

The project aims to create an HDR image drawing application. Designer needs to develop several different types of brushes and some filters in this application. There are two canvases in this application. An image will be loaded in the first canvas. Then a set of brushes will allow painting in the second canvas, filters in this application will provide some constraints to the image editing, so that the image features present in the first canvas will spatially affect the brushes effects in the second canvas. The purpose of this is to pixel-wise multiply the images afterwards to create a high dynamic range image.

Framework and UI design

Development platform: Qt
Advantages: open source, cross platform, object oriented, capable IDE, rich API, helpful documentations and example applications, support openGL.



Important Functions

Constraint function

1. Given a single selected point in the image.
2. Compute its diffused gradient field.
3. Use diffused gradient field to generate paths from the point toward the image boundaries
4. Compute the influence values along each path.
5. Use a scattered bilateral interpolation to generate an influence map.

Ref: Liang Chia-Kai et al.: TouchTone: Interactive Local Image Adjustment Using Point-and-Swipe. Volume 29 (2010), Number.

RGBE function

1. Write the image header (height, width, exposure).
2. All pixels RGB values store in dynamic array A.
3. RGB values convert to RGBE values pixel by pixel.
4. Create a new RGBE image.

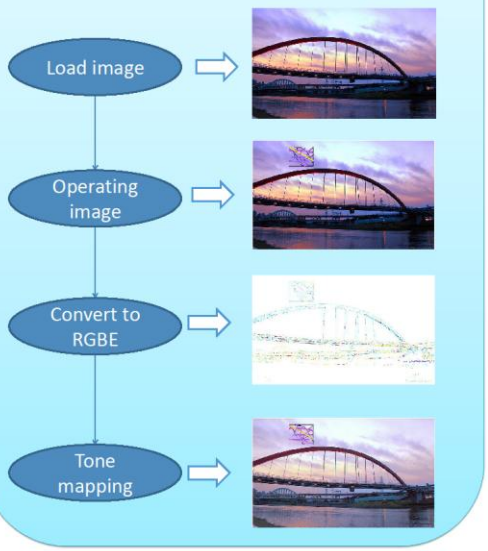
Tone Mapping Function

1. Set the tone mapping attributes.
2. Send the image information and attributes to function.
3. Tone Mapping.*(Reinhard et al. 2002)*

$$L(x,y) = \frac{a}{L_w} L_w(x,y)$$

4. Create the output image.

Implement & Experiment Result



Conclusion & Further Work

Conclusion: The research and result demonstrate that it is possible to use the brushes to painting image in the second canvas and the constraints in the first canvas will spatial effect the image operating , and produce a new HDR image. The visualization in the output image is better than the input image.

Further Work: Develop more suitable filters for this application.