EDITABLE VIEW OPTIMIZED TONE MAPPING
FOR VIEWING HIGH DYNAMIC RANGE PANORAMAS
ON HEAD MOUNTED DISPLAY

by

Yuan Li

A thesis
submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Boise State University

August 2017
DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Yuan Li

Thesis Title: Editable View Optimized Tone Mapping For Viewing High Dynamic Range Panoramas On Head Mounted Display

Date of Final Oral Examination: 30th June 2017

The following individuals read and discussed the thesis submitted by student Yuan Li, and they evaluated the presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Steven Cutchin, Ph.D. Chair, Supervisory Committee
Jerry Alan Fails, Ph.D. Member, Supervisory Committee
Amit Jain, Ph.D. Member, Supervisory Committee
Maria Soledad Pera, Ph.D. Member, Supervisory Committee

The final reading approval of the thesis was granted by Steven Cutchin, Ph.D., Chair of the Supervisory Committee. The thesis was approved by the Graduate College.
dedicated to my family
ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Professor Steven Cutchin of the Computer Science Department at Boise State University. The door to Prof. Cutchin’s office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right direction whenever he thought I needed it.

I would also like to thank the participants who were involved in the user survey for this research project. Without their passionate participation and input, the user survey could not have been successfully conducted.

I would also like to acknowledge Professor Jerry Alan Fails of the Computer Science Department at Boise State University for his very valuable advice on conducting user study.

Finally, I must express my very profound gratitude to my family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.
ABSTRACT

Head mounted displays are characterized by relatively low resolution and low dynamic range. These limitations significantly reduce the visual quality of photorealistic captures on such displays. This thesis presents an interactive view optimized tone mapping technique for viewing large sized high dynamic range panoramas up to 16384 by 8192 on head mounted displays. This technique generates a separate file storing pre-computed view-adjusted mapping function parameters. We define this technique as ToneTexture. The use of view adjusted tone mapping allows for expansion of the perceived color space available to the end user. This yields an improved visual appearance of both high dynamic range panoramas and low dynamic range panoramas on such displays. Moreover, by providing proper interface to manipulate on ToneTexture, users are allowed to adjust the mapping function as to changing color emphasis. The authors present comparisons of the results produced by ToneTexture technique against widely-used Reinhard tone mapping operator and Filmic tone mapping operator both objectively via mathematical quality assessment metrics and subjectively through user study. Demonstration systems are available for desktop and head mounted displays such as Oculus Rift and GearVR.
# TABLE OF CONTENTS

ABSTRACT ................................................................. vi

LIST OF TABLES ....................................................... x

LIST OF FIGURES ..................................................... xi

LIST OF ABBREVIATIONS ........................................... xiv

LIST OF SYMBOLS .................................................... xvi

1 Introduction .......................................................... 1
   1.1 High Dynamic Range Imagery .................................. 1
   1.2 Tone Mapping ................................................... 3
   1.3 Related Work .................................................. 5
   1.4 Thesis Statement ............................................... 9

2 Methodology ........................................................ 13
   2.1 Equirectangular Projection .................................... 14
   2.2 Tone Mapping Function ....................................... 21
      2.2.1 Luminance Histogram and Bernstein Curve .............. 21
      2.2.2 Cubic Spline .............................................. 28

3 Implementations ................................................... 35
   3.1 Viewing Window Capture ..................................... 36
LIST OF TABLES

3.1 Camera Configuration ............................................. 35
3.2 Half Precision Bits Conversion ................................. 57
4.1 Panorama Luminance Information .............................. 60
C.1 Data-set From User Evaluation Task 1 ......................... 103
C.2 Data-set From User Evaluation Task 3 ......................... 104
C.3 Data-set From TMQI ............................................. 107
LIST OF FIGURES

1.1 Screen Door Effect ............................................. 2
1.2 Tone Mapping Effects ............................................. 3
1.3 Viewer Location Example ........................................ 4
1.4 Fast Bilateral Filtering ......................................... 5
1.5 Gradient Domain HDR Compression ............................. 6
1.6 Slim Backlight Drive ............................................ 7
1.7 Naughty Dog Filmic Example .................................... 9
1.8 Spline Interpolation ............................................. 10

2.1 Equirectangular Panorama Example ............................ 14
2.2 Equirectangular Project Distortion ............................ 16
2.3 View Angle Effects .............................................. 17
2.4 Incorrect Window Identification ............................... 18
2.5 Capture Program Setup .......................................... 20
2.6 Histogram Processing Using Bernstein Polynomial ......... 26
2.7 Cubic Spline Example ........................................... 29
2.8 Boundary Constraints Effects .................................. 32

3.1 Asynchronous Reading Flow .................................... 37
3.2 Filmic Curve Components ....................................... 46
3.3 Filmic Curve Toe ................................................ 47

xi
LIST OF ABBREVIATIONS

API – Application Programming Interface
CPU – Central Processing Unit
DMA – Direct Memory Access
FBO – Frame Buffer Object
FOV – Field Of View
FPS – Frames Per Second
GPU – Graphics Processing Unit
HDR – High Dynamic Range
HMD – Head Mounted Display
HVS – Human Visual System
LDR – Low Dynamic Range
MPI – Message Passing Interface
OpenCV – Open Source Computer Vision Library
OpenGL – Open Graphics Library
PBO – Pixel Buffer Object
SDR – Standard Dynamic Range, equivalent to LDR
TMO – Tone Mapping Operator
TMQI – Tone-mapped Image Quality Index

VR – Virtual Reality
LIST OF SYMBOLS

$\sqrt{2}$  square root of 2

$\lambda$  longitude on the spherical surface.

$\varphi$  latitude on the spherical surface.
CHAPTER 1

INTRODUCTION

1.1 High Dynamic Range Imagery

With current virtual reality (VR) popularity, head mounted displays (HMD) are becoming widely available. These devices can be desktop-oriented headsets like Oculus Rift and HTC Vive, or small goggles that can turn into an immersive VR HMD after connecting to your cell phone.

Current HMDs have a severe screen door effect, a visual artifact of the display devices where the fine lines between pixels (or subpixels) are visible in the displayed image. This is caused by the close distance between the display screen and the viewer’s eye, the use of magnifying lenses to increase field of view [38] and the insufficient pixel density of the device [2, 21]. This combined with their low dynamic range display ability leads to a notably reduced visual quality under the device resolution [33]. As illustrated in Figure 1.1.

Given the fact that the screen resolution can not be easily increased, an alternative avenue for improving visual quality is to introduce high dynamic range (HDR) imagery to provide better image quality in terms of color space. However, most display monitors and HMDs have a limited color range display ability that only supports low dynamic range (LDR) imagery [6]. The term “dynamic range” is used to describe the luminance range between the brightest pixel and the darkest pixel in digital imagery.
The most widely used standard dynamic range (SDR) or LDR has 256 distinct degrees of luminance since each pixel is stored in 8-bit integers. Compared to what we perceive everyday, this is a dynamic range too narrow to represent real world scenes. HDR imagery, on the other hand, uses floating point numbers to represent pixel values and therefore can reproduce a far greater dynamic range of luminosity. If the image uses a 32-bit single precision floating point number, the dynamic range will have up to $2^{32}$ divisions in brightness as there are $2^{32}$ different binary combinations. Broad dynamic range like this can present a similar range of luminance to that experienced through the human visual system (HVS), such as many real-world scenes containing both very bright, direct sunlight and extreme dim shade [4]. In conclusion, HDR means more divisions in brightness and therefore more contrast.
1.2 Tone Mapping

Due to the limitations of display contrast, the extended luminosity range of an HDR image has to be compressed to be made visible. The technique to map one set of colors to a smaller one to approximate the appearance of HDR images in a medium that has a more limited dynamic range is called tone mapping. This method reduces the overall contrast of an HDR image to facilitate display on devices with lower dynamic range display ability, and can be applied to produce images with preserved local contrast or exaggerated for artistic effect. Tone mapping addresses the problem of severe contrast reduction from original scene radiance to the target displayable range while preserving the image details and color appearance that are important to appreciate scene content, as shown in Figure 1.2. Because of the reduction in dynamic range, tone mapping inevitably causes information loss [23, 24, 40, 41].

When viewing a panorama image via HMD, the user only sees a small region of the entire panorama at a particular moment, as illustrated in Figure 1.3. This means that a large portion of the panorama does not need to be rendered. Conventionally,
when a user tries to view an HDR panorama where range of luminance levels is on the order of 10,000 to 1 [28], a global tone mapping operator (TMO) is applied to the entire image. The resulting LDR panorama will have a much narrower luminance range throughout the whole image, normally from 0 to 255. This means that the user, who is only able to see a small region out of the entire panorama, will end up looking at a viewport where luminance range is even narrower than that of the LDR panorama. Therefore visual quality for the end user can be significantly improved by just applying tone mapping on the HDR pixels within the current viewable rectangle. In this way, every small region the user views can utilize the available 256 shades of luminance as much as possible so that coloring contrast is mostly preserved.
1.3 Related Work

There have been related works in localized TMOs over the past decade [12, 13, 32]. Many of them focus on optimizing viewing quality in local regions of high contrast scenes. The mapping function used in these local operators varies spatially, depending on the neighborhood of the pixel. Frédéric Durand and Julie Dorsey use a fast bilateral filtering to extract two layers from the input HDR image - the base layer which only consists of large-scale variations and the detail layer created via dividing input image’s intensity by the base layer [12]. Then contrast reduction is only applied to the base layer so that details from the input HDR image are preserved as much as possible. The decomposed layers as well as a tone mapped LDR image is shown in Figure 1.4.

The work done by Raanan Fattal, Dani Lischinski and Michael Werman introduced a conceptually simple, computationally efficient and robust tone mapping method [13]. Instead of directly modify on the luminosity, their focus is on the gradient field of the input HDR image. Their TMO begins by attenuating large gradients in the original field and then constructs an globally optimized LDR image by solving a Poisson equation on the modified gradient field. As illustrated in their
Figure 1.5: Gradient Domain High Dynamic Range Compression by Raanan Fattal, Dani Lischinski and Michael Werman. The darker shade in the attenuation factor map indicates stronger attenuation work, this TMO demonstrates that it is capable of drastic dynamic range compression, while preserving fine details and avoiding common artifacts, such as halos, gradient reversals, or loss of local contrast. The method is also able to significantly enhance ordinary images by bringing out detail in dark regions. Figure 1.5 provides the gradient attenuation factor map generated from an HDR input image using this TMO and the final tone mapped LDR image.

In [32], the authors presented a window-based tone mapping method that uses a linear function to constrain the tone reproduction inside predefined overlapping windows in order to naturally suppress strong edges while retaining weak ones. Then the HDR image is tone mapped by solving an image-level optimization problem that integrates all windows-based constraints. And it is generally noticed that local operators, which reproduce the tonal values in a spatially variant manner, perform more satisfactorily than global operators.

Even though these methods all generate LDR images with good visual quality while preserving contrast details, they have fundamental limitations to be applied to view HDR panoramas via HMDs. Firstly, they all generate one LDR image as the
output. No matter how good these resulting LDR images are, they cannot exploit as many levels of available luminance as possible, considering the user is only able to see a small portion of the whole image. Moreover, computationally efficient as these methods are, they are still too expensive to run in real time. That is to say, even when applying these methods just to the small viewable region, the computational cost will severely jeopardize the application’s frame rate. When the user is exposed to an immersive VR environment, insufficient frame rate is one of the major causes of virtual reality sickness [22], a term for symptoms that are similar to motion sickness symptoms like general discomfort, headache, stomach awareness, nausea, vomiting, pallor, sweating, fatigue, drowsiness, disorientation, and apathy [20]. Computational cost combined with hardware limitation makes these methods not ideal while using HMD as the display media.

Figure 1.6: Slim Backlight Drive on Sony’s new commercial HDR TVs that allows localized illumination based on image content

A more recent implementation in content-based local contrast optimization can be
found in *Sony’s Slim Backlight Drive* technology on the company’s new commercial HDR TVs, as illustrated in Figure 1.6. Note that these commercial HDR TVs only use extended integers for pixel luminance representation instead of floating point numbers. While they do possess a more extended dynamic range than traditional LDR TVs, the produced luminosity range is still no match for true HDR. Introduced for the first time in early 2016, this new technology aims to provide the best of both worlds between the slim aesthetics of edge-lit LEDs, and more precise local dimming allowed by direct-lit FALD sets. *Sony* won’t go into much technical detail on how the *Slim Backlight Drive* works, other than to say that it’s based on a grid lighting approach that allows edge LED lighting to illuminate relatively small sectors within the image independently of each other, rather than just being able to control top to bottom or left to right strips of the image as has been the case previously.

There hasn’t been much work in tone mapping editing. Video game developing company *Naughty Dog* creates and uses an adaptive Filmic TMO in their latest game *Uncharted4* to render a massive amount of HDR resources [14]. The developers first introduce a look-up table for tone mapping parameters for the Filmic curve. Then a graphical interface is designed to allow game artists to dynamically adjust different sections of this Filmic curve. Modification can be observed immediately after artists finish editing. This live update also enables the artists to freely look around the scene as they are tweaking colors to make sure the curve creates beautiful views everywhere, as shown in Figure 1.7. This technique is still aiming for a global view optimization whereas our proposed method optimizes visible content on sub-region scales while preserving coloring consistency across the entire HDR scene.
1.4 Thesis Statement

This thesis presents a novel technique that not only supports view optimized tone mapping of high resolution (up to 16384 by 8192 in pixels) photographic HDR panoramas, but also allows for real-time tone mapping editing to dynamically adjusting coloring emphasis. This technique is defined as ToneTexture, named after the generated extra texture file that stores pre-computed tone mapping parameters. The ToneTexture technique features dynamic tone mapping of an HDR panorama customized to the view direction of the user. This method applies a TMO customized to the panorama content that is visible to the user based on current view direction while also utilizing both global and regional panorama image luminance details to maintain a consistent coloring of the resulting LDR panorama in whatever direction the user may look.
Figure 1.8: Cubic spline interpolation of 11 data points

*ToneTexture* can utilize a variety of TMOs as determined to most optimally work for selected content. The implementation presented in the thesis combines the use of cubic spline functions with window-based image processing. The method is conceptually simple yet efficient, flexible and robust. By associating a window region to every possible view angle on the panoramic sphere, localized optimization can be achieved while covering the entire HDR panorama image. To assure real-time viewing performance, *ToneTexture* technique allocates most computational operations in pre-processing phase. The results - cubic spline function coefficients - are then stored in an extra texture file which is referred to as *ToneTexture*.

The reason for adopting cubic splines as TMO is because their curve shapes can be modified directly by manipulating on the control points [5]. For a piecewise smooth cubic spline, the shape is only determined by the knots\(^1\) [30]. By modifying one knot’s Y value, it can generate a high peak or a low trough on the curve. Likewise, by adjusting one knot’s X value, the newly generated peak or trough can be shifted in the X axis accordingly, as shown in Figure 1.8. In this way, the cubic spline func-

\(^1\)where the polynomial pieces connect.
tion grants the ToneTexture technique enough control while also ensuring flexibility and robustness to handle windows of various luminance distributions. Notably, it is not optimal to allow the artists to manipulate on all knots directly when they are immersed in the virtual scene via HMD. This thesis addresses this problem by developing an user-friendly interface that groups modification on knots from different luminance regions like bright regions and dim regions so that users can adjust coloring emphasis intuitively and easily.

Apart from the interface, another issue resolved for both viewing and editing functionalities is the smoothing of tone mapping parameters. Since ToneTexture only considers local luminance information when generated, each viewing window ends up using independent tone mapping parameters. This will introduce inconsistency among different regions. The result is that when a user turns the view direction, there could be unpleasant popping between scenes. Such popping issues affect tone mapping editing as well and is even more critical to resolve. After the user finishes editing the tone mapping parameters at one viewing window, the user will expect surrounding windows to be modified accordingly. The solution presented in this thesis consists of two major components: a content-based detection algorithm to determine nearby windows that will be affected, and a mapping method that properly adjusts the modification that will be applied to the identified windows based on their content-wise closeness to the original edited window.

Notably the same idea of window-based view-dependent tone mapping is adopted in [42], where the author applied simple log-average luminance adjustment on the viewing windows and stored the calculated results in a look-up table. Comparatively, ToneTexture advances in several ways. First, ToneTexture is targeting at processing large-sized HDR panoramas (up to 16384 by 8192 in width and height) with real
time viewing performance. Second, ToneTexture is more flexible and robust. Due to the nature of equirectangular projection, severe distortion was observed near the poles [42], ToneTexture copes with this problem in window-based processing by proper transformation. Last and most important, ToneTexture allows for easy access to tone editing.

To demonstrate the effectiveness of ToneTexture technique, this thesis carries out an objective tone mapped image assessment using Tone Mapped image Quality Index (TMQI) [41] to compare ToneTexture against the well-known Reinhard TMO [29] as well as one of the widely used Filmic TMOs created by Jim Hejl and Richard Burgess-Dawson [15]. Additionally, a user study is also conducted to evaluate ToneTexture results in a subjective venue. To get meaningful averaged ratings, ten users are invited for an image quality assessment study [17, 31, 39]. The results from both objective and subjective assessment suggest that ToneTexture generates better LDR image in terms of preserving coloring accuracy and contrast preservation.
CHAPTER 2

METHODOLOGY

Our work in view optimized tone mapping started with a simplistic two pass algorithm that kept track of a few statistical parameters such as minimum or maximum luminance values as the pixels were rendered by a fragment shader and adjusted these pixels during the barrel filter pass accordingly. However, this method suffers from the drawback of losing the HDR luminance prior to tone mapping operation unless the rendering of the image takes place in HDR framebuffers which are not readily available on all devices currently. In addition, computing the minimum, maximum, and average of pixel luminance values during the rendering process adds extra computation to the fragment shader. Thus this primitive algorithm is not practical for real time rendering.

The proposed solution computes optimized tone mapping curves at pre-processing time and stores the function parameters in an extra texture file named *ToneTexture*. In order to preserve flexibility and robustness, each viewable region is associated with one unique cubic spline that is optimized for this very region based on its luminance distribution. The statistic attributes are used during the generation of such a spline. Thus instead of directly using statistics like minimum and maximum, *ToneTexture* stores cubic spline coefficients and when rendering the panorama, the view program simply loads these coefficients and calculates new luminance using the curve in the fragment shader.
Figure 2.1: An equirectangular projected spherical panorama with grid lines from Ben Kreunen

The target panorama images are 16384 by 8192 \texttt{exr} files which use 16-bit half precision floating point numbers to store RGB and transparency values. With minimum precision of $2^{-10}$ in the range of [0, 2], \textit{ToneTexture} is dealing with true HDR images with far more than 256 degrees of luminance. The following sections in this chapter elaborate details in equirectangular projection, viewable region identification, \textit{ToneTexture} generation and how to use cubic spline for tone mapping.

### 2.1 Equirectangular Projection

Equirectangular projection, also known as spherical projection or direct polar, is a common way of representing spherical surfaces in a rectangular plane that simply uses the polar angles as the horizontal and vertical coordinates [34]. More precisely, the horizontal coordinate is directly represented by longitude and the vertical coordinate equals to latitude, without any transformation or scaling applied. Since longitude varies over 360° while latitude has a maximal degree of 180°, such projected rect-
angular maps are normally presented in a 2:1 width to height ratio. Figure 2.1 is a spherical panoramic image created by photographer Ben Kreunen. This projection is widely adopted in computer graphics as it is the standard way of texture mapping a sphere. Therefore the spherical panoramic HDR images used in the development and testing are all generated under this projection.

This projection introduces critical challenges. First of all, any attempt to map a sphere onto a plane naturally causes distortion [7]. The most noticeable distortion in this projection is the horizontal stretching that occurs as one approaches the Zenith\(^2\) to the Nadir\(^3\) from the equator, which extends the poles from a single point into a line of the whole width of the projected map. An example of such distortion is given in Figure 2.2. As illustrated, in an equirectangular panoramic image, all verticals remain vertical and the horizon becomes a straight line across the middle of the image. Coordinates in the image relate linearly to pan and tilt angles in the real world. Areas near the poles get stretched horizontally. The further the pixels are away from the equator, the more redundant they become. Because while sampling rate is constant over the panoramic sphere surface, such a sphere still only consists of discrete pixels. This means that the very circle at the equator has the most meaningful pixels on the spherical surface and circles that are parallel to the equator but near the poles end up with less unique pixels. When all circles are projected onto a plane, they will be mapped to the same length, suggesting that circles near the poles will have largely redundant pixels.

Another important factor that makes this thesis challenging is the requirement for

\(^2\)The point directly above the person viewing or above the camera, the north pole on the panoramic sphere.

\(^3\)The point directly below the person viewing or below the camera, the south pole on the panoramic sphere.
Figure 2.2: An example of equirectangular distortion using cubic grid scene [18]

large Field of View (FOV). FOV, which is regarded as the extent of the observable environment at any given time, is one of the more important aspects in VR. The wider the FOV, the more realistic the user is likely to feel in the experience. Binocular FOV is the more important type among different FOVs. Where the two monocular FOVs overlap, there is the stereoscopic binocular FOV, which is around 114°. This overlap is where humans are able to perceive things in 3D [16]. A wider binocular FOV is important for immersion and presence because this stereoscopic area is where most of the action happens every day and therefore most attention is drawn. This is why most popular HMDs provide an average stereoscopic FOV of 110°.

If the view angles are small, it is relatively easy to map content into an image on a flat surface since this viewing arc is relatively flat and the distortion is rather trivial. As the view angle increases, the viewing arc becomes more curved and the distortion gets more severe. Figure 2.3 illustrates such relation between view angle change and projection distortion. Since ToneTexture targets HMDs as its display medium, our technique has to present scenes on large FOV to the end user. Such distortion challenges the validity of ToneTexture technique and can not be simply
(a) Narrow angle of view, grid remains nearly square

(b) Wider angle of view, grid is highly distorted

Figure 2.3: How view angle changes the impact of projection distortion

disparaged.

Given that the location of the view point is fixed at the center of the panoramic sphere, view orientation can be identified solely by the longitude and latitude of viewing window center. Suppose \((\lambda, \phi)\) are longitude and latitude of a view direction respectively, with their origins located at the upper left corner of the unwrapped panorama. The viewable pixels from the fixed viewing point are bounded by a window whose center locates at \((\lambda, \phi)\). Since each view angle is associated with one window, the number of windows developed will only be determined by the accuracy of longitude and latitude. Therefore the *ToneTexture* file has a flexible size that only depends on how much HDR details need to be preserved. In current implementation, the precision in both longitude and latitude is down to 1°. As a result, the *ToneTexture* implementation has \(181 \times 360\) viewing windows covering from the Zenith to the Nadir. Extensions allow for a reduction in the size of *ToneTexture* file at the cost of HDR
accuracy by increasing the degree steps in view angles. Such reduction can also help eliminate redundant overlapping viewable regions near the poles and improve the generation process efficiency as well.

However, detecting the visible pixels on the unwrapped panoramic HDR image is difficult because of the distortion introduced by equirectangular projection and large FOV requirement. In the preliminary implementation of ToneTexture technique, such distortion was overlooked and viewing windows were directly taken from the projected rectangular image using fixed width and height pixelwise length, as shown in Figure 2.4. This window pixel identification was inaccurate because the actual viewable region should be stretched horizontally as it approached the Zenith and the Nadir. This method was also inefficient because it processed all pixels at high latitude while most of them were redundant. To locate the visible pixels on the source panorama, we can use the following equation to get its horizontal and vertical coordinates:
\[ x = (\lambda - \lambda_0) \cos(\varphi_1) \]
\[ y = (\varphi - \varphi_1) \]

Equation 2.1 is the mathematical representation of the forward projection that transforms spherical coordinates into planar coordinates, where \( \varphi_1 \) is the latitude of the Zenith, \( \lambda_0 \) is the longitude of the central meridian, \( x \) and \( y \) are the horizontal and vertical coordinates on the projected plane respectively. Since this transformation is apparently not linear, viewing windows at different latitude have different shapes, area size and therefore total number of pixels on the panorama image. While windows can be detected by using Equation 2.1, this approach is not optimal for several reasons. First of all, because windows at different latitude have different shapes, pixel data is hard to store in a constant 2D manner. Second, due to different total numbers of pixels in different windows, it is hard to find a universal implementation for statistical information gathering. At last, windows at high latitude have largely redundant horizontal pixels, especially near the poles where the entire row of pixels are duplicate of one pixel. Computation for locating these redundant pixels is unnecessarily repeated each time view angle is updated because the projected windows are not linearly mapped on the plane and thus there is no simple mean to locate new pixels.

Our solution is a simulate-capture method. The steps include: placing a virtual camera at the center of the panoramic sphere, configuring its parameters to simulate the human eye, rotating it to traverse all defined view angles and capturing raw HDR data directly from the graphics processing unit (GPU) by reading the rendered framebuffer. Figure 2.5 gives a visualization of this simulation capturing process. Comparing with the preliminary approach, this solution is:

- Efficient. The total number of windows identified for traversing through one
entire panorama image using the old solution equaled the total number of pixels of the image. For the targeted $16K \times 8K$ panoramic image, there were $134,217,728$ uniquely identified viewing windows. In the revised approach, the total number of windows is only: $181 \times 360 = 65,160$. Comparatively, the new method only needs to process $0.04\%$ of the data needed in the old implementation.

- **Accurate.** As expounded above, the preliminary approach was inaccurate due to neglecting distortion caused by equirectangular distortion. The accuracy of the new approach is not affected by equirectangular distortion since the data is take directly from the GPU as it is rendered and presented to the end user.

- **Flexible.** The viewing windows captured by the simulator solution can also be applied to the same panoramic image at lower resolution, whereas the old approach would require generating viewing windows over again even when the contents in both images were mostly the same. Moreover, these flexible viewing
windows can be used on HMDs of different resolutions as well. By adjusting the display configuration of the used HMD based on the hardware setup in the view program, the same visible content can be presented to the viewer. Therefore, tone mapping parameters generated based on the simulator-captured viewing regions can also be deployed in the fragment shader.

2.2 Tone Mapping Function

Using the simulate-capture method described in Section 2.1, we obtain 65,160 viewing windows covering all view angles on a panoramic sphere. The next step is to utilize this data to generate the same number of tone mapping functions used in every defined view angle that are optimized based on content’s luminance distribution. This section manages to answer two questions:

1. What type of statistical attributes should be collected?

2. What tone mapping function can exploit the collected attributes to be flexible and robust?

2.2.1 Luminance Histogram and Bernstein Curve

Based on the data collected in previous phase, we can create the histogram of pixel luminance for every viewable rectangle in the entire panorama. We classify these histograms into four categories:

- **Ascending**: The overall histogram tends to ascend as luminance grows. This is the region where most of its pixels are bright.
• **Descending:** The overall histogram tends to descend as luminance grows. This is the region where most of its pixels are dim.

• **Peak:** The pixels in this region are evenly distributed and most pixels are located at the middle of luminance range.

• **Trough:** Pixels in this region form a trough in the histogram. Most pixels are either in the bright range or the dim range. As a result, this region has both bright spots and dim spots.

To identify a viewing window’s histogram category, a simple but effective approach is to use a bin-counting algorithm to generate the luminance value histogram and categorize it based on the relative heights of the bins. As illustrated in Figure 2.6, histogram type can be determined by detecting where the peak is and how many peaks are there in the histogram.

To customize the mapping curve we use in the fragment shader, we need to first determine what we want to achieve after tone mapping. Our goal is to create one tone mapping curve optimized in one viewable region that can equalize this region’s original luminance histogram and stretch it as wide as possible to exploit the limited available luminance range. For each of the four histogram categories, its final output can be described as follow:

• **Ascending:** The whole region needs to be dimmed down and the brighter the pixels are, the more they will be dimmed down.

• **Descending:** The whole region needs to be brighten up and the dimmer the pixels are, the more they will be brighten up.
• **Peak:** For this type of region, we only need to stretch the histogram reasonably so that its center approximates the center of available luminance range.

• **Trough:** We want to scale the dim peak to brighten the dim regions and the bright peak down at the same time so that the resulting histogram is equally spread across available luminance LDR.

After we define the resulting histogram, we can have a basic outline of the curve for each type of histogram. In the early stage of this thesis, we chose Bernstein polynomial because its shape could be directly modified by changing its coefficients. Because we intended to store tone mapping parameters in another image file and we could keep up to four parameters as RGBA values in .exr format, we used 3rd order Bernstein curves which granted us the most controls under the limit of the maximum number of coefficients we could store. Equation 2.2 is the general form of a 3rd order Bernstein polynomial:

\[
B_3(x) = A(1 - x)^3 + B(3x(1 - x)^2) + C(3x^2(1 - x)) + Dx^3, \quad (2.2)
\]

\(A, B, C, D\) were called Bernstein coefficients. This smooth 3rd polynomial curve was directly determined by its coefficients in a clear pattern: in its domain, the smooth curve could be treated as a composition of four segments; from left to right, each segment was associated with \(A, B, C\) and \(D\) respectively; increasing the value of a specific coefficient would form a peak in the related segment and decreasing the same coefficient would result in a trough shape in that segment.

We used linear equation systems to calculate the values of \(A, B, C\) and \(D\). The statistical attributes needed in this approach were minimum, maximum and average luminance values. To solve four unknowns, we needed to conduct four equations. In
cases of **Ascending** and **Descending**, statistical attributes could contribute to the following 3 equations:

\[ B_3(\text{Min}) = \text{Min}, \quad (2.3) \]
\[ B_3(\text{Max}) = \alpha \text{Max}, \quad (2.4) \]
\[ B_3(\text{Average}) = \frac{\alpha}{\alpha + 1}(\text{Min} + \text{Max}), \quad (2.5) \]

\( \text{Min}, \text{Average} \) and \( \text{Max} \) were statistical attributes of current viewing window, \( \alpha \) was a pre-defined equalization factor which helped to equalize the final histogram in the range of \([0, 1.0]\) after exposure adjustment\(^4\). To be more precise, \( B_3(\text{Max}) \) would be scaled to approximate to 1.0, and \( B_3(\text{Average}) \) would approximate to the scaled arithmetic mean of minimum and maximum values. Equation 2.5 also determined whether the calculated Bernstein function was concave or convex. The 4th equation was different in these two cases. We used the first order derivative of the Bernstein function to constrain the curve’s shape. Equation 2.6 was applied in **Ascending** viewing windows and Equation 2.7 was used in **Descending** viewable regions.

\[ B'_3(\text{Min}) = 0, \quad (2.6) \]
\[ B'_3(\text{Max}) = 0, \quad (2.7) \]

Unlike above cases, **Peak** and **Trough** could use the same set of equations. We still used Equation 2.3 and Equation 2.4 to determine the ends of the smooth curve. Then we divided the pixels into two sub-intervals with average value as the midpoint.

\(^4\)A technique for adjusting the exposure indicated by a photographic exposure meter, in consideration of factors that may cause the indicated exposure to result in a less-than-optimal image, also known as exposure compensation.
and traversed the viewing window again to find new average value of each sub-interval, namely average\_left and average\_right. We conducted two more equations using the newly computed average values:

\[ B_3(Average\_left) = \frac{3\alpha}{4\alpha + 2}(Min + Average), \]
\[ B_3(Average\_right) = \frac{3\alpha}{2\alpha + 4}(Average + Max), \]

Using different sets of four equations for different categories, we could create a linear equation system. Then we used variable elimination that was implemented in CUDA to solve the system to calculate \( A, B, C \) and \( D \).

Based on above algorithm, we could compute a unique set of Bernstein coefficients for each viewable region and stored the result in an \texttt{exr} file where coefficients are stored as RGBA values.

Figure 2.6 shows the four base histogram types, the target histograms of each base type and the curve used to achieve the result. All data comes from viewable regions in the testing Redwood HDR panorama.

The Bernstein polynomial has advantages such as: computationally efficient, easily parallelizable and flexible to be applied to viewing windows of various luminance distribution. But there are several fatal limitations that make Bernstein polynomial inadequate for \textit{ToneTexture} technique.

- **Doesn’t possess second derivative constraint:** The constraint on Bernstein polynomial is not sufficient. The four equations can only put a first derivative constraint on the curve. The first derivative test locates the functional maximal and minimal. But to ensure they are global maximal and minimal
Figure 2.6: Histogram processing via 3rd order Bernstein curves. Frequency unit is thousand.
requires a second derivative test [37]. This lack of constraint leads to weird behavior of the curve, like the decreasing of the right end in the curve in Figure 2.6c.

- **Lack of simple intuitive controls:** Controls should be simple and easy to understand for artists. For 3rd order Bernstein polynomial, the adjustable four segments have uncontrollable lengths, meaning that the modification made on one coefficient has different effect on different curves.

- **No direct control over dynamic range:** Using the Bernstein curve is “all or nothing”. It’s not possible to make a plain linear curve using those controls because the intersection of the controllable segments can not be changed by adjusting coefficients. There are times where the optimal curve is a plain linear curve with a slight shoulder to produce softer transition to the overexposed highlights, and times where the highlights and shadows need to be heavily compressed.

- **No universal solution:** While Bernstein polynomial can handle various types of luminance distribution, it requires different solutions for each type. When the coefficients need to be adjusted after editing, the program has to identify the current visible region’s luminance distribution type to update the curve of current window as well as those in windows nearby.

- **Lerpable parameters absent:** There is no clean blend between different curves in different areas. Lerp is a term for basic operation of linear interpolation between two values that is commonly used in the field of computer graphics [25]. This is a requirement for editing functionality because after the user finishes
editing on the current visible region, the program should blend the adjustment into the surrounding windows’ tone mapping parameters. Otherwise, after the user turns to the window, even if it is one pixel away from the edited one, the modification will not be shown in the new window.

### 2.2.2 Cubic Spline

Cubic spline, on the other hand, not only possesses all the advantages Bernstein polynomial has, but also triumphs in the aspects where Bernstein polynomial performs poorly.

Cubic curves are commonly used in computer graphics because lower order curves are commonly lacking sufficient flexibility, while curves of higher order are often considered unnecessarily complex and easily introduce undesired wiggles if handled without caution. A *spline* originally means a common drafting tool, a flexible rod, which was used to help draw smooth curves to connect widely spaced points. The cubic spline curve accomplishes the same task for an interpolation problem. Suppose we have a data point table containing points represented as \((x_i, y_i)\) for \(i = 0, 1, \ldots, n\) for a function \(y = f(x)\), where for \(i \in [0, n - 1]\), \(x_i < x_{i+1}\). That leaves us \(n + 1\) points and \(n\) intervals between them. The cubic spline is a piecewise continuous curve, passing through each of the points in the table. There is a separate cubic polynomial for each interval, which is universally defined as:

\[
S_i(x) = A_i + B_i(x - x_i) + C_i(x - x_i)^2 + D_i(x - x_i)^3 \quad \text{for} \quad x \in [x_i, x_{i+1}]. \tag{2.10}
\]

Together, these polynomial segments are denoted as \(S(x)\), the spline. The merging
Figure 2.7: Cubic spline example for $n = 5$

points where curve pieces meet are called knots. Figure 2.7 gives an example of a cubic spline curve where $n = 5$. Since there are $n$ intervals and each one of them is fixed by four coefficients, we require a total of $4n$ parameters to define the spline $S(x)$. Thus we need to find $4n$ independent equations to construct the desired curve.

Here is a summary of the known conditions of the spline:

a) $n + 1$ points from the data table: $(x_i, y_i)$ for $i = 0, 1, \ldots, n$.

b) Each spline segment is a third order polynomial curve.

c) At the knots, the spline curve preserves second order parametric continuity.

d) Feature of the boundaries at both ends of the interval: Natural Boundary, Clamped Boundary and Not-A-Knot Boundary.

Condition c) forces the first and second derivatives between adjacent segments to equal at the knots, which implies that the curve tangents at the join points have not only the same direction but also the same magnitude. This condition ensures the
joined spline $S(x)$ is a smooth curve in its domain. The effect of condition d) will be explained later in this section.

We get $2n$ equations for each interval from condition a):

$$S_i(x_i) = y_i$$
$$S_i(x_{i+1}) = y_{i+1}, \text{ for } i = 0, 1, \ldots, n - 1.$$  \hspace{1cm} (2.11)

$$S_i(x_{i+1}) = y_{i+1}, \text{ for } i = 0, 1, \ldots, n - 1.$$  \hspace{1cm} (2.12)

Because the spline curve possesses second order parametric continuity, we can obtain another $2(n - 1)$ more constraints:

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$$  \hspace{1cm} (2.13)

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}), \text{ for } i = 0, 1, \ldots, n - 2.$$  \hspace{1cm} (2.14)

where the first and second derivatives of the spline can be easily constructed by differentiating Equation 2.10:

$$S_i'(x) = B_i + 2C_i(x - x_i) + 3D_i(x - x_i)^2$$  \hspace{1cm} (2.15)

$$S_i''(x) = 2C_i + 6D_i(x - x_i)$$  \hspace{1cm} (2.16)

So far we have $2n + 2(n - 1) = 4n - 2$ equations. The remaining two equations that we need to completely fix the spline are derived from condition d). Certain constraints should be put on the derivatives at $x_0$ and $x_n$ to limit the boundaries at the both ends of the domain. There are three traditional options:

- **Natural:** There is no force to bend the curve on the ends so that they arise naturally. Mathematically, this condition can be expressed as follows:
\( S''_0(x_0) = 0 \)
\( S''_{n-1}(x_n) = 0 \)

(2.17)

- **Clamped:** The first derivatives at both ends are set to a defined value, namely \( \alpha \) and \( \beta \). This is normally used when there is specific requirement on the tangents at \( x_0 \) and \( x_n \). Then the resulting equations are:

\[
S'_0(x_0) = \alpha \\
S'_{n-1}(x_n) = \beta
\]

(2.18)

- **Not-A-Knot:** Instead of specifying any extra conditions at the end points, this constraint forces third order parametric continuity across the second and penultimate knots of the spline. In this way, cubic polynomials are not changed when crossing \( x_1 \) and \( x_{n-1} \), leaving the curve to be more natural and accurate for data interpolation. This can be expressed as:

\[
S'''_0(x_1) = S'''_1(x_1) \\
S'''_{n-2}(x_{n-1}) = S'''_{n-1}(x_{n-1})
\]

(2.19)

There is no defined way to determine which constraint is better. The choice is made depending on the application’s expectation on both ends of the curve. Figure 2.8 gives an example of how different boundary conditions affect the ends of the resulting spline. For *ToneTexture*, Natural Boundary works sufficiently well as it won’t produce weird behavior at the ends while its equation is the easiest to construct.
Figure 2.8: How different boundary constraints influence cubic splines
Now that we have $4n$ linear conditions, next step is to establish the equations that determine $4n$ coefficients. The detailed linear equation matrix and solution will be elaborated in Chapter 3. The one last question for using cubic spline as the tone mapping curve is: what are the points? This question is equivalent to the one proposed at the beginning of this section: what type of statistical attributes should be collected?

When $S(x)$ is used to convert an HDR pixel into an LDR pixel, $x$ should be the original luminance value and $y$ should be the mapped luminance value. To make full use of the available LDR, $y_i$ can be easily determined by dividing the LDR into equal-length intervals. For normalized LDR, whose range is $[0, 1.0]$, $y_i$ can be defined as $y_i = i \times 1.0/n$ for $i = 1, 2, \ldots, n$. If we still use minimal, maximal and average values as we did in Bernstein polynomial method, $x_0$ and $x_n$ are the minimal and maximal luminance in a visible region. $x_{\lfloor n/2 \rfloor}$ is the average that will be mapped to the midpoint in the LDR as $y_{\lfloor n/2 \rfloor}$, which is 0.5. For $n = 2$, these three points are all that the data table contains. This limited data table is not sufficiently enough for cubic spline interpolation and the resulting curve is not accurate nor flexible for different types of luminance distributions. To refine the curve, we set $n = 4$, the next power of 2, so that $x_1$ and $x_3$ are still yet to be retrieved from the captured viewing window. To do that, we divide the HDR pixel luminance values into two intervals using $x_{\lfloor n/2 \rfloor}(x_2)$ as the midpoint and calculate the new average luminance in the left interval as $x_1$ and the average luminance in the other interval as $x_3$. Following the definition of $y_i$, $x_1$ is mapped to 0.25 and $x_3$ is mapped to 0.75. If $x_1$ is closer to $x_0$, suggesting that there are more pixels in $[x_0, \frac{x_0+x_2}{2}]$ than $[\frac{x_0+x_2}{2}, x_2]$. When pixels in $[x_0, x_1]$ are mapped to a dynamic range that is of the same length of that assigned to pixels in $[x_1, x_2]$, there will be more contrast revealed in the dim regions that contains
more data.

This refinement on the curve can keep going for \( n = 2^k \) where \( k = 0, 1, \ldots \). The more data points we use, the finer spline we’ll get. The cost includes the computation to retrieve average luminance values in sub-intervals and the memory needed to store more coefficients for more segments. Note that by using more average values, there is no need to identify the region’s distribution type as the spline will automatically be adjusted under the constraints. In our implementation, we set \( n = 8 \) which is computationally acceptable because there are only \( \log(8) = 3 \) recursion levels for retrieving average values and also generates a spline that is flexible and robust enough to handle different types of visible regions.

When the user wishes to change the coloring emphasis, the simplest way is to modify \( x_i \) and re-compute the spline on the fly. For instance, if the user wants to brighten up the scene, the program should decrease the values of \( x_i \) for \( i = 1, 2, \ldots, n - 1 \) and compute a new spline. As a result, pixels with lower luminance value will be mapped to higher values by the new spline because \( y_i \) stays the same. The opposite modification on the data points will result in a dim-down effect on the visible contents. However, the interface provided to the user to achieve the desired functionality should be: organized to save the artist the trouble of changing all data points individually; intuitive as the interface won’t confuse the user to perform intended coloring modification; easy to use so that even if immersed in a VR environment, the user should finish the modification task without many errors. Implementation details including interface design will be illustrated in Chapter 3.
Our work consists of two major components: the ToneTexture generation program using OpenGL on desktop computer and HDR panorama viewer implemented on both desktop computer and Samsung Note 4 with Gear VR HMD. Real-time tone mapping parameter editing is supported on both platforms. The demo viewer on Android uses Gear VR Framework (GearVRf), a lightweight, powerful, open source rendering engine with a Java interface for developing mobile VR games and applications for Gear VR and Google Daydream View. Both implementations support HDR panoramas up to $16K \times 8K$ in size. Hardware systems include Dell Workstations equipped with NVIDIA Quadro K620 GPU for development and testing, and Samsung Note 4 smartphone with GearVR HMDs. Implementation for both platforms share the same camera configuration so that contents observed by the viewer are constant and ToneTexture generated for the same panorama can be used in both application versions.

<table>
<thead>
<tr>
<th>Framebuffer Height</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framebuffer Width</td>
<td>1024</td>
</tr>
<tr>
<td>Color Format</td>
<td>RGBA (A is for $\alpha$ channel indicating transparency)</td>
</tr>
<tr>
<td>FOV</td>
<td>110°</td>
</tr>
<tr>
<td>Multi-sampling</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.1: Virtual camera configuration used for both Android and desktop implementation
While Chapter 2 addresses the methods used in this thesis, there are other critical challenges throughout the development, including implementation details, performance improvement and compatibility assurance. The following sections will elaborate on these matters.

### 3.1 Viewing Window Capture

Reading pixels back from the framebuffer in *OpenGL* is commonly accomplished with the `glReadPixels()` API. Understanding how this command functions is essential for achieving good application performance when such read back is incurred.

By default, `glReadPixels()` reads data from framebuffer objects (FBOs). This procedure blocks the rendering pipeline until all previous *OpenGL* commands are executed, and waits until all pixel data are transferred and ready for use before it returns control to the calling application. It is obvious that this has two negative performance impacts: forcing a synchronization point between the calling application and *OpenGL*, which should always be avoided, and the cost of the data transfer from GPU to central processing unit (CPU) across the bus, which can be fairly expensive depending on how much data is retrieved.

Since the content drawn in the framebuffer has been through rasterisation\(^5\), pixel values are stored as single-precision normalized floating point numbers in the range of \([0, 1.0]\). The size of the FBO equals that of the viewable region. Thus the data for a single transfer are: \(1024 \times 1024 \times 4 \times 4 = 16\text{MB}\). According to the datasheet for Quadro K620 [9], the raw memory bandwidth on the GPU is \(29.0\text{GBps}\). This is far more than what we need to transfer all the data in an FBO at 60 frames per second.

---

\(^5\)The task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (pixels or dots) for output on a video display.
Figure 3.1: Asynchronous glReadPixels() with 2 PBOs

(FPS). But in the development, the capturing program can not achieve such high FPS. So the bottleneck is not the data transferring but the synchronization where the GPU must wait for the CPU to complete the calling application’s tasks.

Alternatively, `glReadPixels()` bound with pixel buffer objects (PBOs) can schedule asynchronous data transfer and returns immediately without any stall. Therefore, the application can execute other processes like calculating tone mapping parameters right away while transferring the data by *OpenGL* at the same time. The other advantage of using PBOs is the fast pixel data transfer from and to GPU though direct memory access (DMA) without involving CPU cycles. In the conventional way, the pixel data is loaded into system memory by CPU, whereas using a PBO allows GPU to manage copying data from the framebuffer to a PBO. This means that *OpenGL* performs a DMA transfer operation without wasting CPU cycles. Figure 3.1 illustrates the architecture and processing flow when using two PBOs for asynchronous reading.

After acquiring pixel data from the framebuffer, the next question is how to use the data for tone mapping parameter computation. In initial design, since all
visible regions have the same rectangular size, the process of computing tone mapping parameters can be easily parallelized to achieve high performance. In order for parallel processing, all captured visible content should be stored as individual HDR images and uploaded to kestrel, a 32-node CPU/GPU cluster from the High Performance Simulation Laboratory and COEN IT department. Then a program using CUDA and Message Passing Interface (MPI) would be developed to distribute the computation among the nodes for shorter execution time. CUDA is a parallel computing platform and API model created by NVIDIA [10], and MPI is a standardized and portable message-passing system designed to function on a wide variety of parallel computing architectures [26]. However, there is a fatal drawback in this design. It requires excessive writing of pixel data to disk to save as individual image files. First, those image files consume colossal physical memory. Ignoring any compression, when using .exr as the image format, the total raw data takes: 16MB ÷ 2 × 64800 = 506.25GB (.exr stores pixel data in half precision floating point number per channel). Next, the function provided by OpenEXR library to write .exr file takes on average one second for each window of designated size, during which time the CPU is completely blocked without multi-threading. With 64800 visible regions, the total time for image writing will be an unacceptable 18 hours.

Due to this expensive I/O overhead, we abandoned the parallel processing method, combined the viewport capturing program with cubic spline coefficients calculation and used the CPU solely to bypass the I/O overhead in file writing. The serialized implementation takes on an average of 4000 seconds to process one 16K HDR panorama, which is only approximately 6% of the time needed to save viewing windows in files. This makes the program run at approximately 16 FPS. Given the fact that the rendering by OpenGL performs steadily at 30 FPS, it suggests that the
CPU is still blocked by calculating the spline during each rendering cycle. Further improvement can be achieved by implementing a thread pool for multi-threading to compute the coefficients concurrently. Figure A.1 in Appendix A provides implementation details of the key functions for asynchronous data reading.

When the CPU loads the pixel data from the designated PBO, the first thing to do is to get the luminance values of each individual pixel. Equation 3.1 is a formula to convert RGB color values to brightness [27]. The formula assigns different weights to colors in proportion to the human eye’s sensitivity on individual color channels. In general, humans are more sensitive on green and red components than on blue channels. If we use the same weight, for example, \((R + G + B)/3\), then pure red, pure green and pure blue result in same gray scale level, which conflicts with our perceived visual experience.

\[
\text{Luminance} = 0.299 \times R + 0.587 \times G + 0.114 \times B
\]  

Equation 3.1

After converting RGB values to luminance, the brightness information of a visible region can be now stored in a one-dimensional array. To find the data points for constructing tone mapping spline, we have to traverse the array multiple times to calculate the averages of the entire data domain and the sub-intervals separated by upper level average values. Since all \(n\) data points are monotone increasing in \(x_i\) for \(i \in 0, 1, \ldots, n\), one way to efficiently calculate the average values in sub-intervals is to sort the array before the calculation function is called. Another benefit for using sorted data is that calculation in the next step can be solved in a divide-and-conquer manner without worrying about different lengths of the sub-intervals. Suppose the total number of pixels in one visible region is noted as \(m\). The luminance data can
Algorithm 1 Finding averages values in $Luminance[m]$ as $x_i$ for $n$ data points

function findDataPoint($x$, $Luminance$, left, right)
    sum $\leftarrow$ 0, startIdx $\leftarrow$ 0
    mid $\leftarrow$ $(left + right)/2$
    while $Luminance[startIdx] < x[left]$ do
        $\triangleright$ find the starting index
        startIdx $\leftarrow$ startIdx + 1
    end while
    endIdx $\leftarrow$ startIdx
    while $Luminance[endIdx] < x[right]$ do
        $\triangleright$ sort the $Luminance$ array first
        sum $\leftarrow$ Luminance[endIdx]
        endIdx $\leftarrow$ endIdx + 1
    end while
    $x[mid] \leftarrow sum/(endIdx - startIdx)$
    if $mid - left > 1$ then
        findDataPoint($x$, $Luminance$, left, mid)
        findDataPoint($x$, $Luminance$, mid, right)
    end if
end function

procedure Find Data Points($Luminance$, $m$, $n$)
    mergeSort($Luminance$, 0, $m$)
    $x[0] \leftarrow Luminance[0]$, $x[n] \leftarrow Luminance[m]$
    findDataPoint($x$, $Luminance$, 0, $n$)
end procedure
be efficiently sorted using *merge-sort* whose average and worst-case performance is $O(m \log(m))$ [19]. Another reason for choosing *merge sort* over *quicksort* is that unlike some efficient implementations of *quicksort*, *merge sort* is a stable sort, even though it does not sort in place [8]. But since memory is not our concern, this sorting method becomes optimal for the task. The runtime for finding the average of raw HDR luminance is $O(m)$ because traversal through the entire array at least once is required. In the next level of recursion, the sorted data are divided into two halves using the newly-found average value as midpoint and the size of each sub-interval can be noted as $m_1, m_2$ respectively with $m = m_1 + m_2$. The total running time at the second recursion level is therefore $O(m_1) + O(m_2) = O(m)$. Thus the recurrence $T(m) = T(m_1) + T(m_2) + O(m)$ follows from the definition of the algorithm. Since there will be $\log(n)$ recursion levels, the complexity to retrieve all average values is $m \log(n)$. In our implementation, $n = 8$, meaning that we can treat $\log(n)$ as constant. As a conclusion, the runtime for finding needed data points should be: $O(m \log(m)) + O(m) = O(m \log(m))$. This procedure is concluded in Algorithm 1.

### 3.2 Cubic Spline Coefficients

After retrieving average values from a captured viewing window, we need to calculate for an optimized cubic spline using the data point table. In Subsection 2.2.2, the mathematical representation of a cubic spline is given as Equation 2.10:

$$S_i(x) = A_i + B_i(x - x_i) + C_i(x - x_i)^2 + D_i(x - x_i)^3 \text{ for } x \in [x_i, x_{i+1}]. \quad (2.10 \text{ revisited})$$

We also specify four constraints to construct this piecewise cubic spline with second order parametric continuity from a table of points $(x_i, y_i)$ for $i = 0, 1, \ldots, n$
for the function $y = S(x)$, in which for every $i \in [0, n - 1]$, $x_i < x_{i+1}$. The constraints are summarized in the following five equations:

- **Continuity in data points:**

  $$S_i(x_i) = y_i$$  
  \hspace{1cm} (2.11 revisited)

  $$S_i(x_{i+1}) = y_{i+1}, \text{ for } i = 0, 1, \ldots, n - 1.$$  
  \hspace{1cm} (2.12 revisited)

- **Continuity in derivatives:**

  $$S'_i(x_{i+1}) = S'_{i+1}(x_{i+1})$$  
  \hspace{1cm} (2.13 revisited)

  $$S''_i(x_{i+1}) = S''_{i+1}(x_{i+1}), \text{ for } i = 0, 1, \ldots, n - 2.$$  
  \hspace{1cm} (2.14 revisited)

- **Natural boundary:**

  $$S''_0(x_0) = 0$$

  $$S''_{n-1}(x_n) = 0$$  
  \hspace{1cm} (2.17 revisited)

from these constraints, $4n$ linear equations are derived to determine $4n$ coefficients.

This section will give details on establishing a linear equation matrix and applying the *Tridiagonal Matrix Algorithm* to solve the linear system.

Define the points’ step length in $X$ axis as: $h_i = x_{i+1} - x_i$ and start from Equation 2.10, then we can get:

a. Derived from Equation 2.11:

$$A_i = y_i$$  
\hspace{1cm} (3.2)
b. Substitute $x_{i+1} - x_i$ with $h_i$ in Equation 2.12:

$$A_i + h_i B_i + h_i^2 C_i + h_i^3 D_i = y_{i+1} \quad (3.3)$$

c. From Equation 2.13:

$$\begin{cases} 
  S'_i(x_{i+1}) = B_i + 2C_i h_i + 3D_i h_i^2 \\
  S'_{i+1}(x_{i+1}) = B_{i+1} 
\end{cases} \Rightarrow B_i + 2C_i h_i + 3D_i h_i^2 = B_{i+1} \quad (3.4)$$

d. Based on Equation 2.14:

$$2C_i + 6D_i h_i = 2C_{i+1} \quad (3.5)$$

Define the second order derivative as $m_i = S''_i(x_i) = 2C_i$. Then from Equation 3.5, $D_i$ can be represented as:

$$D_i = \frac{m_{i+1} - m_i}{6} \quad (3.6)$$

The only coefficient left is $B_i$. Substitute $A_i$, $C_i$ and $D_i$ with $m_i$ in Equation 3.3, we can get:

$$B_i = \frac{y_{i+1} - y_i}{h_i} - \frac{m_{i+1} + 2m_i}{6} h_i \quad (3.7)$$

Since all the coefficients are either known values ($y_i$ and $h_i$) or represented by $m_i$, we only need to establish one equation for the linear matrix. Substitute all coefficients in Equation 3.4:
\[
\frac{y_{i+1} - y_i}{h_i} - \frac{m_{i+1} + 2m_i}{6} h_i + m_i h_i + \frac{m_{i+1} - m_i}{2} h_i = \frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{m_{i+2} + 2m_{i+1}}{6} h_{i+1}
\]
\[
\Rightarrow h_i m_i + 2(h_i + h_{i+1}) m_{i+1} + h_{i+1} m_{i+2} = 6\left(\frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{y_{i+1} - y_i}{h_i}\right) \quad (3.8)
\]

Note that Equation 3.8 is only for \(i \in [0, n - 2]\), which can only contribute to \(n - 1\) rows in the linear matrix while there are \(n + 1\) rows in total for \(m_i\) from \(m_0\) to \(m_n\). The remaining two equations are derived from Natural Boundary constraint (Equation 2.17):

\[
m_0 = m_n = 0 \quad (3.9)
\]

Hence, the matrix can be established as:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & \ldots & 0 \\
h_0 & 2(h_0 + h_1) & h_1 & 0 & 0 & \ldots & 0 \\
0 & h_1 & 2(h_1 + h_2) & h_2 & 0 & \ldots & 0 \\
0 & 0 & h_2 & 2(h_2 + h_3) & h_3 & \ldots & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \ldots & 0 & 0 & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\
0 & \ldots & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
m_0 \\
m_1 \\
m_2 \\
m_3 \\
\vdots \\
m_{n-1} \\
m_n \\
\end{bmatrix}
= 6
\begin{bmatrix}
\frac{y_2 - y_1}{h_1} - \frac{y_1 - y_0}{h_0} \\
\frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1} \\
\frac{y_4 - y_3}{h_3} - \frac{y_3 - y_2}{h_2} \\
\vdots \\
\frac{y_{n-1} - y_{n-2}}{h_{n-2}} - \frac{y_{n-2} - y_{n-3}}{h_{n-3}} \\
\end{bmatrix}
\begin{bmatrix}
m_0 \\
m_1 \\
m_2 \\
m_3 \\
\vdots \\
m_{n-1} \\
m_n \\
\end{bmatrix}
\]

\( (3.10) \)

This linear equation system is a tridiagonal system since the matrix on the left is a tridiagonal matrix. In linear algebra, a tridiagonal matrix is a specific type of square matrix that has nonzero elements only on the main diagonal and along the subdiagonal\(^6\) and superdiagonal\(^7\). For such systems, the solution can be obtained in \(O(n)\) operations instead of \(O(n^3)\) required by conventional Gaussian elimination [1]. The efficient solution is named Tridiagonal Matrix Algorithm, also known as the

\(^6\)Comprised by elements directly under the main diagonal.

\(^7\)Comprised by elements directly above the main diagonal.
Thomas Algorithm (named after Llewellyn Thomas). It is a simplified form of Gaussian elimination that consists of two phases: a forward elimination to convert the matrix into an upper triangular matrix and a backward substitution to produce the solution. Detailed explanation on the algorithm can be found in Appendix B.

After solving the equation system, coefficients can be obtained using Equation (3.2), (3.6) and (3.7) while $C_i = \frac{m_i}{2}$. These coefficients are stored in a texture file which functions as a look-up table for tone mapping. Notably, since the most computationally expensive operation is the one retrieving data points and calculating average values in different intervals, $x_i$ is also saved for real time editing. After manipulating the data points, we can control the spline by re-computing coefficients to interpolate new data. Even though the spline curve can be affected by all data points individually, it is not an optimal interface design to grant user direct control on every point, especially when the user is isolated and immersed in the VR environment. This calls for an interface that groups modification on knots from different luminance regions while providing easy and intuitive control on the mapping curve.

### 3.3 Editing Interface

Even though HDR imagery isn’t the mainstream in digital imagery, it has been a research topic for decades. Tone mapping, however, has been studied even longer by traditional film photographers instead of computer scientists. In film photography, contrast is one of the most significant characteristics of an image [36]. As a display medium, films only have a limited exposure range in which they can produce contrast. If areas of a film receive exposure either below or above the useful exposure range, contrast will be diminished or even absent. Thus it is a key concern for photographers
for converting differences in exposure (subject contrast) into film contrast (differences in density). The “exposure-to-film-density” curve, referred to as Filmic curve in computer graphics, functions equivalently as a tone mapping curve in HDR-to-LDR conversion. Since film photographers have researched the curve and developed an intuitive customization system for different coloring emphasis, the editing interface for ToneTexture adapts from the Filmic curve adjustment to achieve simple and intuitive control while producing aesthetic result.

Figure 3.2: Components in a Filmic curve, example took from John Hable’s blog

As shown in Figure 3.2, similar to piecewise cubic spline, a Filmic curve is made of three distinct regions with different mapping characteristics. The part of the curve associated with low luminance is referred to as the toe, which corresponds to the dim portions of an LDR image. The shoulder is responsible for transfer contrast in areas that receive relatively high exposures to the maximal luminance that is mapped to pure white. The linear section is a direct mapping from HDR to LDR without
Figure 3.3: (a). Three types of toe. (b). Effect on dark luminance of the same color by different toes

changing the content contrast. This part is of least concern since it has minimal impact on the resulting image.

Different toes result in different coloring effects on the dark end. Toes can be categorized by its shape: concave downwards, linear and concave upwards, as shown in Figure 3.3a. A concave down curve is a curve that for every point on the curve, the tangent line to the curve at that point lies above the graph in the vicinity of the point and a concave up curve is defined in an opposite situation where the tangent line lies below. A concave downwards toe brings up the low value. This is reflected in the desaturated blacks in the dark regions. On the contrary, a concave upwards toe brings down high values, leading to more saturation in the blacks. This is illustrated in Figure 3.3b. There can be a bi-directional control on our spline to adjust the toe from concave downwards to concave upwards, using arrow key buttons or joysticks of a gamepad. To achieve this, we can modify $x_1$ to $x_3$ together. With both $x_0$, $x_4$ and $y_i$ unchanged, decreasing these values can generate a concave downwards toe...
whereas increasing them will lead to a concave upwards curve on the dark end. The modification is restricted that $x_i$ preserves to be monotone increasing. Moreover, to create a nice arc on the spline, modification on $x_2$ is enlarged by a small factor of 1.5 so that the curve segment pops out at that knot.

Unlike the toe, the shoulder normally has negative concavity. For HDR imagery, the bright regions are usually overexposed areas where detailed contrast appears to be washed out under linear mapping. To reveal more information, the pixel’s luminance should be mapped to get closer to 1.0 without actually hitting pure white. When cubic spline is computed using average values, the shoulder is concave downwards in all view angles. It is the same case for other TMOs like Reinhard TMO [28], Filmic curve by Jim Hejl and Richard Burgess-Dawson [15] and Filmic TMO developed by John Hable from *Naughty Dog* [15]. However, if the user is not concerned with the overexposed regions or there is not useful information in the bright areas for the user, then using a shoulder with positive concavity can allocate more dynamic range to the less bright areas. Unlike above TMOs where the shoulder is fixed concave downwards, cubic spline can change its shoulder in this case. Similar to how we handle the toe, the same interface is provided to adjust $x_5$ to $x_7$ together to change the spline’s concavity, under the same restriction and with small factor applied to $x_6$.

The last difficult issue in editing is to apply the change to the entire panorama. Because each visible region has its own set of cubic spline coefficients, modification made in one region won’t affect nearby windows by nature. If not resolved, when the user finishes editing one visible region, the editing effect can only be observed at that very view direction. It is not practical for the user to modify *ToneTexture* degree by degree. This issue has to be solved with a data smooth approach so that adjusted modification is applied to surrounding windows after the user is done editing.
Since viewing windows are identified in longitude and latitude, an initial solution is to mark neighboring regions within fixed radius, then adjust original modification based on the Euclidean distance from the affected region to the original edited window in longitude and latitude coordinates. However, this approach has a fatal error due to equirectangular distortion. Suppose the user edits tone mapping parameters when looking directly up (or down), meaning that the edited window is the one at the Zenith (or Nadir). Because all windows at the poles have almost the same content (slight differences are introduced by rotation), the modification should be applied to all of them, even those falling outside the defined radius. A valid but redundant way is to traverse all identified windows, but this will pose challenge for realtime rendering and tone mapping editing. The use of Euclidean distance to adjust modification can also be inaccurate. Windows at high latitudes are quite alike. Such distance may not reflect this relationship and therefore fails to make proper adjustment.

An effective solution is implemented using a content-based detection technique. First, we define two vectors, \( v_{\text{dim}} = [x_1, x_2, x_3] \) and \( v_{\text{bright}} = [x_5, x_6, x_7] \). Then we calculate the three-dimensional vector’s Euclidean distance between two windows to determine their “content-wise closeness” in toe-affected and shoulder-affected areas separately using Equation 3.11:

\[
d(v_a, v_b) = \sqrt{\sum_{i=1}^{k} (v_{ai} - v_{bi})^2} \tag{3.11}
\]

If the distance falls within a preset threshold, it means these two windows are “close” enough so that modification in one window should affect the other as well. The same distance value can be used to adjust the original modification to smooth the change across all involved view angles.
3.4 Implemented Demos

Even though implemented on different platforms, both Android and desktop panorama viewers share the same architecture, as shown in Figure 3.4. In panorama viewer program, we start by loading the original panorama into texture memory and apply rasterization. Meanwhile, the program reads the entire ToneTexture, retrieves one set of tone mapping parameters using the user’s current view angle and passes these coefficients to the fragment shader. Our view dependent TMO in the fragment shader then maps HDR luminance to LDR luminance using the optimized cubic spline. The viewer application keeps updating the view angle so that the TMO always deploys suitable coefficients to make full use of available LDR contrast. To avoid unnecessary computation, editing is handled solely by the CPU instead of the GPU. In this way, once a modification is made, a re-computation of a new cubic spline curve is done just once while the result can be used by all fragments.
However, due to hardware limitations and framework used for development, there are several issues on the Android platform that call for extreme attention.

3.4.1 Spheres and Cubes

To display the panorama, the viewer application needs to build a geometry as the canvas. Intuitively, such an object should be a spherical object so that the HDR panorama can be directly mapped on. A big question follows immediately: how to represent the spherical surface and how to divide it in discrete pixels to retrieve texture coordinates?

The answer is Cubemap. Beginning with a cube, whose faces have been divided into regular grids, we project every surface point out from the cubic center to an enclosing sphere. This results in a perfectly smooth sphere, built out of six identical round shells with curved edges. The texture coordinates on the cubic faces are easy to calculate and do not require complicated trigonometry. More importantly, reverse mapping arbitrary spherical points back onto the cube is in fact natively supported by GPUs as a texture mapping feature. The GearVRf provides APIs for direct use of a
Cubemap scene object. Figure 3.5a illustrates this texture mapping from a cubemap onto the enclosing sphere.

A Cubemap is made of six images, each representing a face of the cube, as illustrated in Figure 3.5b. It is important to convert a equirectangular panorama into Cubemap textures for the six faces. Mathematically, a sphere can be represented using polar coordinates as \((r, \lambda, \varphi)\) with \(r\) for radius and \(\lambda, \varphi\) for longitude and latitude. Following equation illustrates the forward transform from polar coordinates to xyz coordinates:

\[
\begin{align*}
x &= r \cos \lambda \sin \varphi \\
y &= r \sin \lambda \cos \varphi \\
z &= r \cos \varphi
\end{align*}
\]

Because the sphere encloses the cube, the length of the cube is then \(\sqrt{2}r\). To centrally project pixels, we simply set one of \((x, y, z)\) to fixed coordinates based on spherical radius \(r\) and loop through all source pixels to map them onto the cube with the help of \((\lambda, \varphi)\) to determine their designated face. Notably, pixels on the sphere are discrete. Therefore this is not one-to-one mapping and will leave artifacts on the resulting textures. A reverse transform can fix the problem by looping through the undefined pixels in the Cubemap texture and use the calculated polar coordinates for interpolation on the source equirectangular image to find the closest pixel values. Figure 3.6 gives an example of this process.

Then in the viewer program, we create a cubic object in the scene and bind the newly generated cubemap textures onto it. The GPU can automatically convert texture coordinates into world coordinates and efficiently render the virtual scene.
3.4.2 Texture Tiling

As stated, one of the advantages for using Cubemap is the easy calculation of texture coordinates since textures are bound to the Cubemap object in a regular 2D array layout. The usage of a 2D image storage format brings up another benefit on Android platform by enabling texture tiling.

With the source equirectangular HDR image being $16K \times 8K$, the generated Cubemap textures are $4K \times 4K$ square images. For smartphones, due to limited memory, insufficient CPU speed, slow data transferring bandwidth and strict requirement in efficiency, loading large sized texture directly from its memory can be time and storage consuming. To resolve this issue, the cubic textures are uploaded on Amazon Simple Storage Service (Amazon S3) so that the cell phone only downloads the textures when needed. The downloading seems to aggravate data transferring latency, but because the images are stored on the cloud, we don’t have to worry about storage consumption and can therefore be able to utilize a multi-resolution technique to present content to
the viewer even faster. This is done by decomposing and down-sample the $4K \times 4K$ square images into lower resolution textures and render them in the GPU as soon as possible while loading textures of higher resolution.

In the Android application, the Cubemap object is made of small quads, with each face having an $8 \times 8$ quad grid. Every quad itself is a square geometry of $512 \times 512$ vertex, as illustrated in Figure 3.7a. The program starts by fetching six first-level textures at a low resolution of $512 \times 512$ from the web server, and then binds them to their designated face on the Cubemap. On each face, all quads work together to render this texture by each displaying $1/64$ of the pixels in relation to the quad grid division. Thus the user doesn’t have to wait for a long loading time and can view content of $512 \times 512$ meaningful pixels on each face. The effective pixels the user ends up perceiving are therefore $2K^8 \times 1K^9$.

Meanwhile the program keeps downloading the second-level textures. For the second tiling level, every original $4K \times 4K$ texture is down-sampled to $1K \times 1K$ and decomposed into $2 \times 2$ images of resolution $512 \times 512$. The textures are passed on to the same targeting faces, only this time each face is associated with 4 textures. The quads on every face are also divided by a $2 \times 2$ sub-grid so that each unit in the sub-grid is rendering one corresponding $512 \times 512$ texture. At this point, the user can observe a refined panorama for an effective resolution of $4K \times 2K$.

This tiling hierarchy has four levels in total. At the forth level, each quad will be bound with one $512 \times 512$ texture and the program therefore presents content to the end user at an effective resolution of $16K \times 8K$, as shown in Figure 3.7b.

Apart from quickly bringing content to the user, the texture tiling also improves

---

$^8$from four horizontal faces.

$^9$512 pixels from horizontal face, 256 pixels from half of the top face and 256 pixels from half of the bottom faces.
the compatibility of ToneTexture viewer. For older smartphones whose GPU has less memory to fully load $16K \times 8K$ panoramas, the program can stop at lower tiling levels instead of crashing due to not having enough memory.

3.4.3 HDR Encoding

The last issue that has to be dealt with is introduced by GearVRf framework. As a relatively new project, developers at Samsung are still working to improve its stability as well as add more advanced functionality. Unfortunately, GearVRf hasn’t been able to support HDR textures because it only uses unsigned bytes for the internal texture format in its native OpenGL library.

Our solution is simple but effective. The HDR image format used in the thesis is .exr, which stores pixel data in half precision floating point numbers. Defined in IEEE 754, this is a binary floating-point number format that occupies 16 bits [43].
It has the following format:

- Sign bit: 1 bit
- Exponent width: 5 bits
- Significant precision: 11 bits (only 10 explicitly stored)

Notably, there is an implicit lead bit of value 1 when there are non-zero values in the exponent field.

![Figure 3.8: Encoding 16-bit floating point number into two 8-bit integers](image)

Since GearVRf can only pass 8-bit textures to the fragment shaders, we can traverse all pixels in the HDR texture, split 16-bit floating point numbers in each color channel into two 8-bit integers and encode them as unsigned LDR RGB values to store high bytes and low bytes separately into uncompressed .PNG images. The decomposition is visualized in Figure 3.8.

The viewer application then loads two .PNG files instead of one .exr texture and transfers both high byte and low byte to the GPU. In the fragment shader, before
we apply any tone mapping, a decoding function is called to restore half precision numbers. Based on the exponent bits, the decoding function uses different equations to calculate the original values:

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Significant=zero</th>
<th>Significant≠zero</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000₂</td>
<td>zero, 0</td>
<td>subnormal numbers</td>
<td>((-1)^{\text{sign}} \times 2^{11} \times 0.\text{significant}_2)</td>
</tr>
<tr>
<td>00001₂,...,11110₂</td>
<td>normalized value</td>
<td></td>
<td>((-1)^{\text{sign}} \times 2^{\text{exponent}-15} \times 1.\text{significant}_2)</td>
</tr>
<tr>
<td>11111₂</td>
<td>±infinity</td>
<td>NaN</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Equations to convert bits into half precision floating point numbers for different exponent field cases

A problem brought by this solution is the interpolation between pixels. Since all bits have to be exactly precise to restore the original floating point value, pixels that don’t go through interpolation can be restored correctly whereas pixels interpolated by OpenGL may have strange values. When OpenGL does interpolation between integers, the change made on bits has unpredictable impact on the restored floating point number. For example, if the first bit in the high byte is changed, which is identified as the sign bit during the decoding process, the resulting floating point number will have a completely different value that is unrelated to its neighbors. A custom interpolation can solve this problem so that the pixels are interpolated based on floating point value instead of integer. However, this falls out of the scope of this thesis as it requires a lot of effort and the potential improvement is irrelevant to our proposed tone mapping technique - ToneTexture.
CHAPTER 4

RESULTS

The results of this thesis are presented in two parts. The first part is focused on a user evaluation conducted on the desktop computer platform with advanced tasks for tone mapping editing aiming to improve visual quality to the user’s preference. The second part is focused on objective image quality assessment using a quality scoring measurement developed for tone mapped images.

Both objective and subjective evaluations compare LDR images generated by ToneTexture with those by two other TMOs:

- **Reinhard TMO:** This well-known TMO has many variations [29]. We use a simple form represented in Equation 4.1

\[
\text{Color}_{\text{mapped}} = \frac{\text{Color}}{1.0 + \text{Color}}
\]  

(4.1)

where Color is a three dimensional vector consisting of R, G, B. Simple as it is, this tone mapping gets rid of the brightness and spikes the areas with the most light while also flattens the saturation of the image.

- **Filmic TMO:** This operator is an optimized version by Jim Hejl and Richard Burgess-Dawson to simulate camera’s film response [15]. Their formula uses
arithmetic-logic units to achieve a faster approximation which is widely used in computer games:

\[
x = \max(0, Color - 0.004)
\]

\[
Color_{mapped} = \frac{x \times (6.2 \times x + 0.5)}{x \times (6.2 \times x + 1.7) + 0.06}
\] (4.2)

There are 14 HDR panoramas used for testing in the thesis. One HDR image was captured in Redwood National Park by the Visualization Lab at the Computer Science department of Boise State University while the other 13 HDR panoramas are shared by artist Mark Curtis through his blog and licensed for use in research projects [11]. All images are full 360° × 180° HDR panoramas at 16K × 8K resolution in .exr format. Most of the HDR images are 11 stops of exposure while five of early morning town panoramas are 9 stops. In photography, a ‘stop’ describes total light range by powers of 2. A contrast ratio of 1024 : 1 could therefore be described as having a dynamic range of 10 stops. Thus dynamic ranges of these panoramas are from 512 to 2048, suggesting the maximal luminance values in the source images are far beyond the display ability of traditional LDR monitors. Throughout the evaluations, the sample panoramas are identified by their assigned ID and Table 4.1 provides information about their extreme luminance values along with the percentages of pixels in each quarter of its dynamic range. Data are highly compressed in the [0 – 25%] quarter and pose challenges to TMO to restore information in the largely dark regions while preserving contrast in extreme bright areas.

Because ToneTexture is a tone mapping technique for HMD used in immersive environment, instead of using spherical panoramas as a whole, both evaluations focus on visible sub-regions of panoramas. In each HDR panorama, we select four regions
<table>
<thead>
<tr>
<th>PanoramaID</th>
<th>Minimal</th>
<th>Maximal</th>
<th>[0 – 25%]</th>
<th>[25% – 50%]</th>
<th>[50% – 75%]</th>
<th>[75% – 100%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000128876</td>
<td>486.492</td>
<td>95.1614%</td>
<td>1.96062%</td>
<td>0.473594%</td>
<td>2.40434%</td>
</tr>
<tr>
<td>2</td>
<td>0.000837471</td>
<td>1817.04</td>
<td>99.9606%</td>
<td>0.0186943%</td>
<td>0.0075385%</td>
<td>0.0131346%</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>973.29</td>
<td>99.9781%</td>
<td>0.00554696%</td>
<td>0.00304729%</td>
<td>0.013194%</td>
</tr>
<tr>
<td>4</td>
<td>0.00072259</td>
<td>970.105</td>
<td>99.9629%</td>
<td>0.0108339%</td>
<td>0.00501946%</td>
<td>0.0211984%</td>
</tr>
<tr>
<td>5</td>
<td>0.00078191</td>
<td>1683.86</td>
<td>99.9817%</td>
<td>0.00450909%</td>
<td>0.00188798%</td>
<td>0.0119247%</td>
</tr>
<tr>
<td>6</td>
<td>8.49362e−5</td>
<td>1253.97</td>
<td>99.9846%</td>
<td>0.0037387%</td>
<td>0.0116855%</td>
<td>3.72529e−5%</td>
</tr>
<tr>
<td>7</td>
<td>0.000176412</td>
<td>1186.86</td>
<td>99.948%</td>
<td>0.00933632%</td>
<td>0.00569001%</td>
<td>0.0369683%</td>
</tr>
<tr>
<td>8</td>
<td>0.000192773</td>
<td>763.673</td>
<td>99.9787%</td>
<td>0.00826642%</td>
<td>0.00315085%</td>
<td>0.0095861%</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1201.87</td>
<td>99.9767%</td>
<td>0.00822991%</td>
<td>0.00291914%</td>
<td>0.012105%</td>
</tr>
<tr>
<td>10</td>
<td>0.002618608</td>
<td>757.7376</td>
<td>99.763%</td>
<td>0.22221%</td>
<td>0.0137687%</td>
<td>1.56462e−5%</td>
</tr>
<tr>
<td>11</td>
<td>4.6001e−5</td>
<td>565.958</td>
<td>99.9512%</td>
<td>0.03438129%</td>
<td>0.0105113%</td>
<td>0.00402406%</td>
</tr>
<tr>
<td>12</td>
<td>0.000164136</td>
<td>860.102</td>
<td>99.9977%</td>
<td>0.00295415%</td>
<td>7.45058e−5%</td>
<td>4.47035e−5%</td>
</tr>
<tr>
<td>13</td>
<td>0.000457722</td>
<td>789.0816</td>
<td>98.4757%</td>
<td>1.17469%</td>
<td>0.247121%</td>
<td>0.102528%</td>
</tr>
<tr>
<td>14</td>
<td>0.000179714</td>
<td>914.744</td>
<td>99.9998%</td>
<td>6.92904e−5%</td>
<td>5.96046e−5%</td>
<td>6.70552e−5%</td>
</tr>
</tbody>
</table>

Table 4.1: Luminance information of the evaluated 14 panoramas, the last four columns are percentages of pixels in each quarter of its dynamic range.

with featured luminance distribution:

1. **High Region** is a region that has the most pixels with extremely high luminance and fewest pixels from the dim side of the dynamic range.

2. **Low Region** is a region that is opposite to the **High Region**.

3. **Mid-range Region** is a region that exists between **High Region** and **Low Region** without having extreme pixels.

4. **High&Low Region** is a mixed region of **High Region** and **Low Region** containing pixels from both maximal luminance and minimal luminance.

The process of selecting four regions in each panorama is automated by a program written in C++ with OpenCV. Therefore, the testing database contains $14 \times 4 = 56$ captured visible windows in total. Figure 4.1 provides example for a set of tone mapped images from four regions of the panorama 11 generated by ToneTexture, Reinhard and Filmic.
Figure 4.1: Tone mapped results from panorama 11 generated by different TMOs; from top to bottom, regions are: high luminance, low luminance, mid-range luminance and both high&low luminance
Figure 4.2: Histograms of original HDR High region and tone mapped images correspond to the first row in Figure 4.1.
Figure 4.3: Histograms of original HDR Low region and tone mapped images correspond to the second row in Figure 4.1
Figure 4.4: Histograms of original HDR Mid region and tone mapped images correspond to the third row in Figure 4.1
Figure 4.5: Histograms of original HDR High&Low region and tone mapped images correspond to the last row in Figure 4.1
4.1 User Evaluation

In the subjective evaluation of this thesis, we invited 10 viewers from the Computer Science Department at Boise State University to give a two-session image quality assessment test on the desktop platform separated by an advanced task to edit on randomly selected images using the interface provided by ToneTexture. This subsection details the study process as well as the gathered results.

4.1.1 Parameters

Process

The study has three tasks for an individual user:

Task 1. **Image Quality Selection:** User is presented with three tone mapped images from the same HDR content generated by different TMOs and asked to select the one that is most visually pleasing. The images are not labeled with the TMO used and sets of images are randomly chosen from the database of size 56. Thus the viewer is not likely to assess images from the same panorama so that the judgment is not based on the content. The 56 visible windows are randomly presented to the user and the order of tone mapped images from each window is also randomized. The user is allowed to switch between 3 candidates and is required to spend at least 10 seconds on each region. This task takes around 10 minutes.

Task 2. **Tone Mapping Editing:** Right after the first task is done, each user is asked to edit the tone mapping parameters for a subset of regions to improve visual quality with discretion. Each subset contains 5 or 6 unique regions of
different panorama randomly selected from the database so that 10 observers
can create edited tone mapping images for all 56 regions. A key personnel
for the user study teaches the users on the editing interface and provides
assistance throughout the process in case needed. It is expected that a user
will spend 5 minutes on a single image, leading to a total of 25 to 30 minutes.
This is the end of the first session.

Task 3. **Image Quality Selection (continued):** In the second session of the
study, previous users are invited to come back to give another image quality
selection test on the same set of regions. This time each region has four
candidates, including the one edited by all participants. Sequence of regions
and presented candidates are also randomized. This task takes 15 minutes.

For all three tasks, the expected time is one hour.

**Platform**

For all the tasks, the users were given a desktop PC with programs written in C++ with
**OpenCV** and **OpenGL** libraries. The reason that the PC version was used was because
that there was only one mobile device available and could not schedule multiple users
for testing at the same time. Besides, the desktop version made it easier to collect
and retrieve the users data.

Since the user study was for image quality encountered by ordinary users with
standard vision, the range and diversity of distortions tended to be quite small. No
user demographics were collected.

Each participant was presented with a consent form in paper and was required
to sign the form before the study began. There was one batch script written for a
single task so that the whole process was automated. A key personnel was in charge of which script to run and guided the user to give feedback.

After all users were done with the first two tasks, they were asked to come back for the third task which integrated edited tone mapped images for the same image quality selection test as in the first task.

Data

The data collected were: user’s selection for best quality image over different panorama and region types, the human edited tone mapped images. These data-sets were gathered by computer program during the user evaluation.

User edited images were also used in objective evaluation of the thesis. Appendix C.1 includes detailed counting data of user selections for best TMO in Task 1. Appendix C.2 contains the same type of data gathered in Task 3 as Appendix C.1, only adding user edited image as extra candidate for selection.

Participants

Ten users participated with in the user study. All of them were computer science students at Boise State University with normal to corrected vision. All users considered themselves proficient in digital image technology and general computer experience.

4.1.2 Results

The results of the user study is presented in two parts. The first one reveals data gathered in Task 1 while the second half details data gathered from Task 3.
Task 1

A complete table of user selections for the best quality tone mapped image is provided in Appendix C.1. Selections for the same TMO on the same HDR content by different users are summed together.

Figure 4.6 shows the subjective evaluation result of user preference for TMOs grouped by four region types from Task 1. A stacked bar chart is used to compare between each TMO in the same region as well as compare TMOs across the regions. It obviously shows the major preference for ToneTexture over the other two TMOs across all four regions. ToneTexture appears to be most favorable to the participants when viewing mid ranged regions. There is a slight regression in regions containing more low luminance values. Based on Table C.1, analysis of variance shows a p-value for ToneTexture of 0.48 > 0.05 (standard significance level). This indicates that while slightly less users chose ToneTexture over Reinhard and Filmic in low luminance regions, region type has no statistically significant influence on user’s selection.

Figure 4.7 illustrates the result of participants’ preference for TMOs grouped by source panoramas. The p-value for ToneTexture is 0.36, indicating that the average preference of 78% for ToneTexture is not affected by different HDR content.

Task 3

Detailed user selection summary in Task 3 is listed in Appendix C.2. This new selection summary table adds human edited LDR image as a candidate. Figure 4.8 serves as an extension to Figure 4.1 by adding user edited images to the forth column.

Figure 4.9 shows the subjective evaluation result of user preference for TMOs grouped by four region types from Task 3. A stacked bar chart is used to compare
Figure 4.6: User evaluation results in Task 1 of user preference for TMOs grouped by four region types using stacked bar chart. The y-axis represents number of times one TMO is selected as the participant’s favorite. Preference for *ToneTexture* over global TMOs averages 76% in all four regions.
Figure 4.7: User evaluation results in Task 1 of user preference for TMOs grouped by panorama using stacked bar chart. The $y$-axis represents number of times one TMO is selected as the participant’s favorite with a maximal of 40. Preference for ToneTexture over global TMOs averages 78% in all 14 panoramas.
Figure 4.8: Tone mapped results from panorama 11 generated by different TMOs including user edited images; from top to bottom, regions are: high luminance, low luminance, mid-range luminance and both high&low luminance.
between each TMO in the same region as well as compare TMOs across the regions. It demonstrates the effectiveness of tone mapping editing as there is an obvious trend of user endorsement on the edited images. The user edited images are even more favorable than the unedited ToneTexture results when viewing regions with extreme luminance values. ToneTexture is still viewer’s first choice in mid range regions as well as windows containing both dark areas and bright areas. Based on Table C.2, analysis of variance shows a p-value is 0.17 for ToneTexture and 0.15 for user edited images. These results indicate that region type has no statistically significant influence on user’s selection. Averaging 40% for human edited images and 45% user endorsement for ToneTexture in all four regions, the observation from user evaluation not only proves ToneTexture produces better image quality, but also demonstrates the effectiveness of tone mapping editing since the percentage of preference for global TMOs goes down from 24% to 15%.

Figure 4.10 illustrates the result of participants’ preference for TMOs grouped by source panoramas. The p-values for ToneTexture and edited image are both around 0.73, indicating that the major preference for our TMOs is not affected by different HDR content.

In general, tone mapping editing effectiveness is demonstrated in Task 3. Another interesting question is that whether the improvement from tone mapping editing can be observed based on users’ selections in Task 1 and Task 3. Such information can be found by analyzing the trend of users switching from original ToneTexture to human edited LDR images in the same HDR content. However, the data we collected from the user study are inadequate for this analysis because while ten participants are sufficiently enough for image quality evaluation, this number of users is quite limited to demonstrate a clear trend of switching choices from ToneTexture to user edited
Figure 4.9: User evaluation results in Task 3 of user preference for TMOs grouped by four region types using stacked bar chart. The $y$-axis represents number of times one TMO is selected as the participant’s favorite. 40% for human edited images and 45% user endorsement for ToneTexture in all four regions.
images. A further study could be done to illustrate such trend.

Figure 4.10: User evaluation results in Task 3 of user preference for TMOs grouped by panorama using stacked bar chart. The \( y \)-axis represents number of times one TMO is selected as the participant’s favorite with a maximal of 40. Reinhard TMO is rarely selected after including human edited images as candidate. There is also a noticeable reduction in user endorsement for Filmic TMO as well.

4.2 Objective Quality Assessment

4.2.1 TMQI

The objective tone mapped image assessment method - TMQI uses the original HDR image as the reference and scores the tone mapped LDR image on two competing factors: structural fidelity (\( S \)) indicating the structural similarity between the HDR and LDR images and naturalness measure (\( N \)) based on intensity statistics of natural images [41].
The first factor is measured using a multi-scale signal fidelity test that compares both signal strength and local standard deviations from both images. At each scale, the entire map is pooled by averaging the scores from different patches. Then an overall score $S$ is obtained by combining scores at different scale with assigned weights.

Meanwhile, faithfully preserving the structural fidelity of the HDR image alone doesn’t guarantee high quality of the tone mapped LDR image. It should also look natural to the observer. To quantify this subjective attribute, a statistical naturalness model is built upon 3,000 images and the score $N$ is calculated using a Gaussian and a Beta probability density functions.

A joint measure is scalarized by the following:

$$Q = aS^\alpha + (1 - a)N^\beta$$  \hspace{1cm} (4.3)

where $a = 0.8012, \alpha = 0.3046$ and $\beta = 0.7088$. $a$ adjusts the importance of the two components whereas $\alpha$ and $\beta$ determine their sensitivities. The parameters are trained to best fit the result of a subjective experiment on LDR image quality using different TMOs by authors of [35]. With 1.0 being the maximal possible value, the higher TMQI $Q$ score indicates better LDR quality in terms of structurally coherence and statistically naturalness.

As noted by the authors of TMQI, this measurement is designed to evaluate grayscale images only. In order to assess ToneTexture, which is intended for HDR images captured in color, we apply TMQI to each color channel independently and then combine the score based on HVS using Equation 3.1.

The objective evaluation is implemented in Matlab and evaluation is given to LDR images generated by ToneTexture, Reinhard, Filmic, as well as those edited by
participants during the user evaluation. A complete list of the scores is provided in Appendix C.3.

4.2.2 Analysis

Q Score

Figure 4.11 presents the Q score for LDR images produced by different TMOs. We apply analysis of variance on Q score for four TMOs across all panoramas and the p-value for ToneTexture is 0.91, for User-Edited image is 0.98 and for Filmic is 0.12. However, the p-value for Reinhard TMO’s Q score is 0.0005, which means that unlike the other three TMOs, the Q score of Reinhard is influenced by original HDR content. A closer observation shows that Reinhard receives high scores at all four types of regions in panorama 1. Panorama 1 is the only HDR image took by the Visualization Lab and from Table 4.1 we can know that it’s luminance distribution is distinct from the other ones where over 99% of the pixels are in the dim area of the entire dynamic range. If we exclude panorama 1 from the testing database, the p-values of all TMOs are then greater than the standard significance level and we can compare their Q score in general knowing the result is not influenced by original HDR content.

However, by analyzing TMOs’ Q scores by different region types, we find that types of region have great impact on the outcomes as the calculated p-values are all below significance level. This observation confirms the initial hypothesis that localized luminance details are an important factor for tone mapping HDR images. Therefore local optimization is the right venue to better visual quality. Figure 4.12 demonstrates that ToneTexture produces the best LDR images in regions with mid-range luminance and low luminance, which agrees with the result from subjective evaluation. But in
(a) High Region

(b) Low Region
Figure 4.11: TMQI $Q$ score of each TMO for 14 panoramas grouped by region type. Both ToneTexture and user edited images receive higher scores in Low and Mid regions. Filmic performs better in the other two regions. Reinhard receives abnormally high score at Panorama 1.
Figure 4.12: TMQI $Q$ score in boxplot to show average value for each TMO as well as its variance range by region type. Panorama 1 is excluded. Both ToneTexture and user edited images receive better $Q$ scores in Low and Mid regions. Filmic performs better in the other two regions, especially for High&Low region where it has outstanding scores. According to TMQI, tone mapping editing hardly generates better results as ToneTexture always has a better score comparing with user edited images.
high regions, ToneTexture has greater variance than Reinhard and Filmic whereas in high&low regions, ToneTexture is even considered to produce worse LDR image than those two global TMOs. This conflicts directly with the result from user evaluation, posing challenges to the validity of TMQI. We need to look into $S$ and $N$ scores separately to determine which component introduces conflict. According to TMQI, tone mapping editing hardly produces better LDR image while the only exception is observed in the high&low regions where some tone mapped images are improved and receive a higher score.

$S$ Score

Figure 4.13 shows the $S$ score of four TMOs by different panoramas. $S$ scores are generally higher than $Q$ scores, except for some user edited images. This indicates that the tone mapping editing needs to be further restricted as it damages preserving structural information from original HDR. Analysis of p-values eliminates the influence from different panorama but as what is discovered in $Q$ score analysis, region types still affects the quality of tone mapped image.

Figure 4.14 shows that ToneTexture achieves good results in preserving structural similarity to the original HDR panorama in regions with mid-range luminance and low luminance as its average $S$ scores are better than the others. While in high region and high&low region, ToneTexture still demonstrates its potential in maintaining structural coherence as some ToneTexture images receive top scores.

$N$ Score

Figure 4.15 shows the scores of statistical naturalness. By applying analysis of variance on the data by panorama, we locate the factor that causes Reinhard TMO
(a) High Region

S Score at High Regions

(b) Low Region

S Score at Low Region
Figure 4.13: TMQI $S$ score of each TMO for 14 panoramas grouped by region type. Generally all TMOs do well in preserving structural details. *ToneTexture* hardly produces bad result whereas tone mapping editing seems to jeopardize structural fidelity in most cases.
Figure 4.14: TMQI $S$ score in boxplot to show average value for each TMO as well as its variance range. All TMOs do well in preserving structural details. *ToneTexture* hardly produces bad result. Even in High region, some tone mapped images by *ToneTexture* still receive top score. Tone mapping editing seems to jeopardize structural fidelity in most cases. Notably, *Filmic* TMO has a stable performance over all regions.
to be influenced by HDR panorama content. $p < 0.0001$ indicates that Reinhard TMO’s $N$ score is influenced by the original HDR panorama’s luminance distribution. As authors of [41] mentioned in their paper, their implementation of statistical naturalness measure was based on brightness and contrast only as a compromise between simplicity and capability of capturing the most important ingredients of perceived naturalness. Again, such impact can be eliminated by ignoring panorama 1. Notably, all TMOs receive low $N$ scores in high region, indicating that brightness mapping is an inevitable issue in all TMOs.

By excluding panorama 1 from the testing database, Figure 4.16 illustrates where the conflict in $Q$ score comes from. Both ToneTexture and user edited images are considered statistically less natural than Reinhard and Filmic TMO in high region and high&low region. This may suggests that the statistical model used in TMQI to measure subjective naturalness might be oversimplified and therefore challenges the validity of this objective tone mapped image assessment.

In general, the effectiveness of tone editing is not demonstrated in the objective assessment while the user study reveals endorsement on the edited images.
N Score at High Region

(a) High Region

N Score at Low Region

(b) Low Region
Figure 4.15: TMQI N score of each TMO for 14 panoramas grouped by region type. All TMOs receive low naturalness scores in High region, indicating that brightness mapping is an inevitable issue in all TMOs. Global TMOs has abnormally high scores in Panorama 1.
Figure 4.16: TMQI $N$ score in boxplot to show average value for each TMO as well as its variance range. Panorama 1 is excluded. Both ToneTexture and user edited images are considered statistically less natural than Reinhard and Filmic TMO in High region and High&Low region. Filmic seems to have the best performance in regions excluding low region. This means for ToneTexture, we can simply use Filmic curve as mapping function in implementation in regions where it works the best.
CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

When viewing large sized HDR panorama, a global TMO fails to achieve full utilization of the limited dynamic range when the viewer is not able to observe the entire image. We present ToneTexture, a view optimized tone mapping technique for viewing large-sized HDR panorama on HMD with limited dynamic range display ability. The method notably improves the visual appearance while preserving color ratios and contrast details in viewed windows with extreme luminance.

Implementations presented includes two versions that run on Android devices with Samsung GearVR Headset and desktop PC. A user study was conducted to subjectively evaluate tone mapped images generated by different TMOs while also testing out the effectiveness of tone mapping editing. ToneTexture is competing with two widely used TMOs - Reinhard and Filmic. The results indicate major preference for ToneTexture over Filmic and Reinhard.

Objective image quality assessment is also provided. The objective assessment using TMQI partially coincides with the observation in user evaluation. The overall high structural fidelity score ToneTexture gets further proves better preservation in structural information during the luminance mapping than traditional TMOs while the naturalness score does conflicts with the results from user evaluation. But as the
authors of TMQI address in their work, the statistical naturalness measurement is based on intensity statistics only and may be improved with more advanced statistical models. Statistical analysis confirms that localized luminance details can greatly influence the quality of tone mapped LDR images.

5.2 Future Work

This work presents significant advancement in the application of localized tone mapping in immersive environment. This tone mapping technique not only generates visually pleasing results from large sized HDR panorama, but also minimizes the rendering cost for viewing application to run at real-time in VR setup. This represents a data structure that stores pre-computed tone mapping information and can be utilized in the GPU for fast rendering. Thus, artists or game designers are able to perform expensive optimization on the tone mapping process when using HDR resources without worrying about the performance to deliver exquisite graphics to the viewer.

An advanced application of ToneTexture is to apply multiple TMOs depending on the HDR content and performance requirement. The point is that each TMO has its own strength and weakness in various parts of the dynamic range while some operators are more expensive to perform than the others. Instead of making the trade-off decision in the early phase of the development, programmers can implement multiple TMOs and store the essential parameters and let the program decide which one to use based on the luminosity distribution and performance requirement.

As the first tool to provide editing functionality on tone mapping parameters, ToneTexture currently has limited interface for sophisticated coloring emphasis mod-
ification. With cubic spline implementation, the real-time editing feature can be further expanded to provide full control on different parts of the curve. In this way, users can directly manipulate all knots to form various curves. Thus the artists can create more coloring effects from HDR images.
BIBLIOGRAPHY


APPENDIX A

ASYNCHRONOUS PIXEL READING

IMPLEMENTATION
ViewportCapturer::initPBO() {
    // create two PBOs
    glGenBuffersARB(2, PBO_IDS);
    for (int i = 0; i < 2; ++i) {
        // bind pixel buffer and set it to be packed
        glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, PBO_IDS[i]);
        glBufferDataARB(GL_PIXEL_PACK_BUFFER_ARB, WIDTH * HEIGHT * 4,
                        0, GL_STREAM_READ_ARB);
    }
    // restore back to conventional pixel operation
    glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0);
    my_Active_PBO = 0;
}

ViewportCapturer::capture() {
    // set the target framebuffer to read
    glReadBuffer(GL_FRONT);

    // read pixels from framebuffer to PBO
    // glReadPixels() should return immediately.
    // "my_Active_PBO" is used to index target PBO
    glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, PBO_IDS[my_Active_PBO]);
    glReadPixels(0, 0, WIDTH, HEIGHT, GL_RGBA, GL_FLOAT, 0);

    // set the target PBO to another for update pixels
    my_Active_PBO = 1 - my_Active_PBO;

    // map the PBO to process its data by CPU
    glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, PBO_IDS[my_Active_PBO]);
    GLfloat* pixelData = (GLfloat*)glMapBufferARB(GL_PIXEL_PACK_BUFFER_ARB,
                                                   GL_READ_ONLY_ARB);
    if(pixelData) {
        calculateCubicSpline(pixelData, ...);
        glUnmapBufferARB(GL_PIXEL_PACK_BUFFER_ARB);
    }

    // back to conventional pixel operation
    glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0);
}

Figure A.1: 2 key functions for asynchronous pixel data reading.
APPENDIX B

TRIDIAGONAL MATRIX ALGORITHM
In linear algebra, a tridiagonal systems of equations can be represented in the following form:

\[ a_i x_{i-1} + b_i x_i + c_i x_i = r_i, \quad i = 1, 2, \ldots, n \]  

where \( a_i = c_n = 0 \). The Tridiagonal Matrix Algorithm, or the Thomas Algorithm, is derived from Gaussian elimination for solving the equations in \( O(n) \) operations in stead of \( O(n^3) \) by conventional Gaussian elimination. This improvement is achieved by two phases: a forward elimination to convert the matrix into an upper triangular matrix and a backward substitution to produce the solution.

Suppose \( n = 4 \) and we have a tridiagonal system as following:

\[
\begin{bmatrix}
  b_1 & c_1 & 0 & 0 \\
  a_2 & b_2 & c_2 & 0 \\
  0 & a_3 & b_3 & c_3 \\
  0 & 0 & a_4 & b_4
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4
\end{bmatrix}
= 
\begin{bmatrix}
  r_1 \\
  r_2 \\
  r_3 \\
  r_4
\end{bmatrix}
\]  

Forward Elimination

: Divide the first row by \( b_1 \):

\[ b_1 x_1 + c_1 x_2 = r_1 \Rightarrow x_1 + \frac{c_1}{b_1} x_2 = \frac{r_1}{b_1} \]  

Note \( \frac{c_1}{b_1} \) as \( \gamma_1 \) and \( \frac{r_1}{b_1} \) as \( \rho_1 \), then the matrix system can be written as:
Use new-formed first row to eliminate $x_1$ in the second row:

\[
\begin{align*}
    a_2 x_1 + b_2 x_2 + c_2 x_3 &= r_2 \\
    (b_2 - a_2 \gamma_1) x_2 + c_2 x_3 &= r_2 - a_2 \rho_1
\end{align*}
\]  

This can keep going until Matrix B.2 is transformed into:

\[
\begin{align*}
    \begin{bmatrix}
        1 & \gamma_1 & 0 & 0 \\
        0 & 1 & \gamma_2 & 0 \\
        0 & 0 & 1 & \gamma_3 \\
        0 & 0 & 0 & 1
    \end{bmatrix} 
    \begin{pmatrix}
        x_1 \\
        x_2 \\
        x_3 \\
        x_4
    \end{pmatrix} 
    &= 
    \begin{pmatrix}
        \rho_1 \\
        \rho_2 \\
        \rho_3 \\
        \rho_4
    \end{pmatrix}
\end{align*}
\]

with

\[
\gamma_i = \begin{cases}
    \frac{c_i}{b_i}; i = 1 \\
    \frac{c_i}{b_i - \gamma_{i-1} a_i}; i = 2, 3, \ldots, n - 1
\end{cases}
\]  

\[
\rho_i = \begin{cases}
    \frac{r_i}{b_i}; i = 1 \\
    \frac{r_i - \rho_{i-1} a_i}{b_i - \gamma_{i-1} a_i}; i = 2, 3, \ldots, n
\end{cases}
\]

**Backward Substitution**

Since the tridiagonal matrix has been converted into an upper triangular matrix in Matrix B.6, $x_i$ can therefore be solved with a backward substitution:
\[ x_i = \begin{cases} ho_i; & i = n \\ ho_i - \gamma_i x_{i+1}; & i = n - 1, n - 2, \ldots, 1 \end{cases} \] (B.9)
APPENDIX C

EVALUATION DATASET
C.1 Task 1

Table C.1: Data-set From User Evaluation Task 1, 14 panoramas are identified in numbers, while T is for ToneTexture, R is for Reinhard and F is for Filmic.

<table>
<thead>
<tr>
<th>PanoramaID</th>
<th>High Region</th>
<th>Low Region</th>
<th>Mid Region</th>
<th>High&amp;Low Region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T  R  F</td>
<td>T  R  F</td>
<td>T  R  F</td>
<td>T  R  F</td>
</tr>
<tr>
<td>1</td>
<td>9  1  0</td>
<td>7  1  2</td>
<td>10  0  0</td>
<td>10  0  0</td>
</tr>
<tr>
<td>2</td>
<td>6  1  3</td>
<td>9  0  1</td>
<td>9  0  1</td>
<td>5  1  4</td>
</tr>
<tr>
<td>3</td>
<td>8  0  2</td>
<td>9  1  0</td>
<td>8  1  2</td>
<td>7  0  3</td>
</tr>
<tr>
<td>4</td>
<td>8  0  2</td>
<td>6  0  4</td>
<td>6  0  4</td>
<td>8  0  2</td>
</tr>
<tr>
<td>5</td>
<td>7  0  3</td>
<td>8  0  2</td>
<td>5  0  5</td>
<td>9  0  1</td>
</tr>
<tr>
<td>6</td>
<td>8  1  1</td>
<td>4  2  4</td>
<td>7  3  7</td>
<td>1  2</td>
</tr>
<tr>
<td>7</td>
<td>6  0  4</td>
<td>9  0  1</td>
<td>8  0  2</td>
<td>6  1  3</td>
</tr>
<tr>
<td>8</td>
<td>9  0  1</td>
<td>8  1  1</td>
<td>8  0  2</td>
<td>6  0  4</td>
</tr>
<tr>
<td>9</td>
<td>4  0  6</td>
<td>8  0  2</td>
<td>10  0  0</td>
<td>6  0  4</td>
</tr>
<tr>
<td>10</td>
<td>10 0 0</td>
<td>6  0  4</td>
<td>9  0  1</td>
<td>8  0  2</td>
</tr>
<tr>
<td>11</td>
<td>9  1  0</td>
<td>9  1  0</td>
<td>10  0  0</td>
<td>10  0  0</td>
</tr>
<tr>
<td>12</td>
<td>8  0  2</td>
<td>5  0  5</td>
<td>8  0  2</td>
<td>7  1  2</td>
</tr>
<tr>
<td>13</td>
<td>9  0  1</td>
<td>6  0  4</td>
<td>10  0  0</td>
<td>7  1  2</td>
</tr>
<tr>
<td>14</td>
<td>7  0  3</td>
<td>9  0  1</td>
<td>7  0  3</td>
<td>8  1  1</td>
</tr>
</tbody>
</table>
C.2  Task 3

<table>
<thead>
<tr>
<th>PanoramaID</th>
<th>High Region</th>
<th>Low Region</th>
<th>Mid Region</th>
<th>High&amp;Low Region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T E R F</td>
<td>T E R F</td>
<td>T E R F</td>
<td>T E R F</td>
</tr>
<tr>
<td>1</td>
<td>9 1 0 0</td>
<td>4 5 0 1</td>
<td>6 4 0 0</td>
<td>10 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>2 5 0 3</td>
<td>3 6 0 1</td>
<td>5 4 1 0</td>
<td>3 4 0 3</td>
</tr>
<tr>
<td>3</td>
<td>2 6 0 2</td>
<td>2 7 0 1</td>
<td>5 5 0 0</td>
<td>3 5 0 2</td>
</tr>
<tr>
<td>4</td>
<td>2 5 0 3</td>
<td>3 7 0 0</td>
<td>3 3 0 4</td>
<td>2 6 0 2</td>
</tr>
<tr>
<td>5</td>
<td>3 4 0 3</td>
<td>8 2 0 0</td>
<td>4 6 0 0</td>
<td>4 4 0 2</td>
</tr>
<tr>
<td>6</td>
<td>6 2 0 2</td>
<td>1 8 0 1</td>
<td>8 0 1 1</td>
<td>3 6 0 1</td>
</tr>
<tr>
<td>7</td>
<td>6 1 0 3</td>
<td>1 9 0 0</td>
<td>3 7 0 0</td>
<td>4 5 0 1</td>
</tr>
<tr>
<td>8</td>
<td>2 6 0 2</td>
<td>5 5 0 0</td>
<td>8 1 0 1</td>
<td>4 3 1 2</td>
</tr>
<tr>
<td>9</td>
<td>3 5 0 2</td>
<td>4 5 1 0</td>
<td>4 3 0 3</td>
<td>6 2 0 2</td>
</tr>
<tr>
<td>10</td>
<td>5 5 0 0</td>
<td>3 3 0 4</td>
<td>6 3 0 1</td>
<td>7 3 0 0</td>
</tr>
<tr>
<td>11</td>
<td>4 6 0 0</td>
<td>6 4 0 0</td>
<td>9 0 1 0</td>
<td>8 2 0 0</td>
</tr>
<tr>
<td>12</td>
<td>1 7 0 2</td>
<td>5 0 0 5</td>
<td>5 3 0 2</td>
<td>3 6 0 1</td>
</tr>
<tr>
<td>13</td>
<td>4 6 0 0</td>
<td>7 2 0 1</td>
<td>6 4 0 0</td>
<td>3 1 0 6</td>
</tr>
<tr>
<td>14</td>
<td>4 4 0 2</td>
<td>6 4 0 0</td>
<td>5 0 0 5</td>
<td>5 5 0 0</td>
</tr>
</tbody>
</table>

Table C.2: Data-set From User Evaluation Task 3, 14 panoramas are identified in numbers, while T is for *ToneTexture*, E is for *Edited*, R is for *Reinhard* and F is for *Filmic*. 
## C.3 TMQI Result

<table>
<thead>
<tr>
<th>PanoramaID</th>
<th>RegionType</th>
<th>ToneTexture</th>
<th>Edited Image</th>
<th>Reinhard</th>
<th>Filmic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>N  S  Q</td>
<td>N  S  Q</td>
<td>N  S  Q</td>
<td>N  S  Q</td>
</tr>
<tr>
<td>1</td>
<td>high</td>
<td>0.276 0.942 0.866</td>
<td>0.081 0.872 0.802</td>
<td>0.196 0.907 0.840</td>
<td>0.036 0.870 0.786</td>
</tr>
<tr>
<td></td>
<td>low</td>
<td>0.458 0.963 0.905</td>
<td>0.504 0.936 0.906</td>
<td>0.661 0.979 0.943</td>
<td>0.888 0.974 0.977</td>
</tr>
<tr>
<td></td>
<td>mid</td>
<td>0.315 0.957 0.877</td>
<td>0.383 0.960 0.890</td>
<td>0.829 0.974 0.968</td>
<td>0.885 0.964 0.974</td>
</tr>
<tr>
<td></td>
<td>high&amp;low</td>
<td>0.295 0.942 0.869</td>
<td>0.083 0.851 0.797</td>
<td>0.694 0.933 0.938</td>
<td>0.302 0.903 0.861</td>
</tr>
<tr>
<td>2</td>
<td>high</td>
<td>0.046 0.783 0.765</td>
<td>0.037 0.779 0.760</td>
<td>0.141 0.895 0.824</td>
<td>0.255 0.873 0.843</td>
</tr>
<tr>
<td></td>
<td>low</td>
<td>0.278 0.897 0.851</td>
<td>0.229 0.895 0.841</td>
<td>0.018 0.849 0.774</td>
<td>0.030 0.899 0.791</td>
</tr>
<tr>
<td></td>
<td>mid</td>
<td>0.397 0.893 0.874</td>
<td>0.404 0.895 0.875</td>
<td>0.016 0.867 0.778</td>
<td>0.020 0.883 0.783</td>
</tr>
<tr>
<td></td>
<td>high&amp;low</td>
<td>0.043 0.896 0.795</td>
<td>0.036 0.896 0.793</td>
<td>0.022 0.929 0.797</td>
<td>0.011 0.903 0.784</td>
</tr>
<tr>
<td>3</td>
<td>high</td>
<td>0.044 0.770 0.759</td>
<td>0.036 0.765 0.755</td>
<td>0.082 0.712 0.756</td>
<td>0.154 0.803 0.802</td>
</tr>
<tr>
<td></td>
<td>low</td>
<td>0.848 0.922 0.958</td>
<td>0.748 0.917 0.942</td>
<td>0.086 0.846 0.796</td>
<td>0.136 0.897 0.823</td>
</tr>
<tr>
<td></td>
<td>mid</td>
<td>0.520 0.946 0.911</td>
<td>0.712 0.924 0.937</td>
<td>0.284 0.891 0.855</td>
<td>0.541 0.933 0.913</td>
</tr>
<tr>
<td></td>
<td>high&amp;low</td>
<td>0.203 0.905 0.841</td>
<td>0.203 0.905 0.841</td>
<td>0.422 0.906 0.885</td>
<td>0.729 0.932 0.943</td>
</tr>
<tr>
<td>4</td>
<td>high</td>
<td>0.014 0.552 0.677</td>
<td>0.025 0.469 0.649</td>
<td>0.063 0.545 0.694</td>
<td>0.091 0.683 0.749</td>
</tr>
<tr>
<td></td>
<td>low</td>
<td>0.624 0.953 0.931</td>
<td>0.723 0.953 0.946</td>
<td>0.137 0.937 0.834</td>
<td>0.271 0.970 0.873</td>
</tr>
<tr>
<td></td>
<td>mid</td>
<td>0.610 0.862 0.903</td>
<td>0.466 0.844 0.874</td>
<td>0.034 0.698 0.736</td>
<td>0.058 0.797 0.773</td>
</tr>
<tr>
<td></td>
<td>high&amp;low</td>
<td>0.033 0.801 0.766</td>
<td>0.033 0.801 0.766</td>
<td>0.341 0.843 0.853</td>
<td>0.568 0.866 0.900</td>
</tr>
<tr>
<td>5</td>
<td>high</td>
<td>0.007 0.726 0.731</td>
<td>0.005 0.645 0.705</td>
<td>0.010 0.664 0.715</td>
<td>0.027 0.791 0.761</td>
</tr>
<tr>
<td></td>
<td>low</td>
<td>0.550 0.964 0.920</td>
<td>0.641 0.962 0.934</td>
<td>0.006 0.873 0.774</td>
<td>0.004 0.930 0.787</td>
</tr>
<tr>
<td></td>
<td>mid</td>
<td>0.618 0.952 0.930</td>
<td>0.510 0.957 0.913</td>
<td>0.041 0.896 0.795</td>
<td>0.117 0.945 0.831</td>
</tr>
<tr>
<td></td>
<td>high&amp;low</td>
<td>0.023 0.835 0.771</td>
<td>0.024 0.822 0.769</td>
<td>0.091 0.841 0.796</td>
<td>0.174 0.891 0.831</td>
</tr>
<tr>
<td>6</td>
<td>high</td>
<td>0.124 0.827 0.800</td>
<td>0.215 0.770 0.806</td>
<td>0.192 0.792 0.808</td>
<td>0.353 0.856 0.859</td>
</tr>
<tr>
<td></td>
<td>low</td>
<td>0.372 0.637 0.796</td>
<td>0.103 0.824 0.794</td>
<td>0.003 0.810 0.754</td>
<td>0.001 0.704 0.722</td>
</tr>
<tr>
<td></td>
<td>mid</td>
<td>0.524 0.906 0.903</td>
<td>0.504 0.839 0.882</td>
<td>0.062 0.907 0.805</td>
<td>0.053 0.917 0.805</td>
</tr>
<tr>
<td></td>
<td>high&amp;low</td>
<td>0.057 0.861 0.791</td>
<td>0.053 0.853 0.788</td>
<td>0.105 0.862 0.806</td>
<td>0.167 0.888 0.828</td>
</tr>
<tr>
<td></td>
<td>high</td>
<td>low</td>
<td>mid</td>
<td>high&amp;low</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.003</td>
<td>0.301</td>
<td>0.316</td>
<td>0.056</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.281</td>
<td>0.338</td>
<td>0.416</td>
<td>0.056</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.420</td>
<td>0.665</td>
<td>0.416</td>
<td>0.064</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.077</td>
<td>0.338</td>
<td>0.683</td>
<td>0.064</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.005</td>
<td>0.015</td>
<td>0.416</td>
<td>0.207</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.281</td>
<td>0.416</td>
<td>0.665</td>
<td>0.207</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.301</td>
<td>0.338</td>
<td>0.416</td>
<td>0.207</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>high</th>
<th>low</th>
<th>mid</th>
<th>high&amp;low</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0.565</td>
<td>0.778</td>
<td>0.857</td>
<td>0.888</td>
</tr>
<tr>
<td>8</td>
<td>0.676</td>
<td>0.813</td>
<td>0.889</td>
<td>0.790</td>
</tr>
<tr>
<td>9</td>
<td>0.601</td>
<td>0.774</td>
<td>0.889</td>
<td>0.790</td>
</tr>
<tr>
<td>10</td>
<td>0.660</td>
<td>0.811</td>
<td>0.889</td>
<td>0.790</td>
</tr>
<tr>
<td>11</td>
<td>0.722</td>
<td>0.722</td>
<td>0.889</td>
<td>0.790</td>
</tr>
<tr>
<td>12</td>
<td>0.772</td>
<td>0.211</td>
<td>0.889</td>
<td>0.790</td>
</tr>
<tr>
<td>13</td>
<td>0.788</td>
<td>0.211</td>
<td>0.889</td>
<td>0.790</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>high</th>
<th>low</th>
<th>mid</th>
<th>high&amp;low</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0.001</td>
<td>0.015</td>
<td>0.041</td>
<td>0.029</td>
</tr>
<tr>
<td>8</td>
<td>0.512</td>
<td>0.118</td>
<td>0.401</td>
<td>0.414</td>
</tr>
<tr>
<td>9</td>
<td>0.033</td>
<td>0.022</td>
<td>0.003</td>
<td>0.029</td>
</tr>
<tr>
<td>10</td>
<td>0.597</td>
<td>0.967</td>
<td>0.880</td>
<td>0.961</td>
</tr>
<tr>
<td>11</td>
<td>0.076</td>
<td>0.791</td>
<td>0.774</td>
<td>0.790</td>
</tr>
<tr>
<td>12</td>
<td>0.765</td>
<td>0.791</td>
<td>0.790</td>
<td>0.790</td>
</tr>
<tr>
<td>13</td>
<td>0.765</td>
<td>0.791</td>
<td>0.790</td>
<td>0.790</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>high</th>
<th>low</th>
<th>mid</th>
<th>high&amp;low</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0.003</td>
<td>0.035</td>
<td>0.416</td>
<td>0.207</td>
</tr>
<tr>
<td>8</td>
<td>0.851</td>
<td>0.416</td>
<td>0.207</td>
<td>0.207</td>
</tr>
<tr>
<td>9</td>
<td>0.592</td>
<td>0.416</td>
<td>0.207</td>
<td>0.207</td>
</tr>
<tr>
<td>10</td>
<td>0.688</td>
<td>0.416</td>
<td>0.207</td>
<td>0.207</td>
</tr>
<tr>
<td>11</td>
<td>0.663</td>
<td>0.416</td>
<td>0.207</td>
<td>0.207</td>
</tr>
<tr>
<td>12</td>
<td>0.663</td>
<td>0.416</td>
<td>0.207</td>
<td>0.207</td>
</tr>
<tr>
<td>13</td>
<td>0.663</td>
<td>0.416</td>
<td>0.207</td>
<td>0.207</td>
</tr>
<tr>
<td></td>
<td>high&amp;low</td>
<td>high</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>------</td>
<td>------</td>
<td>-----</td>
</tr>
<tr>
<td>14</td>
<td>0.080</td>
<td>0.079</td>
<td>0.559</td>
<td>0.629</td>
</tr>
<tr>
<td></td>
<td>0.896</td>
<td>0.779</td>
<td>0.955</td>
<td>0.925</td>
</tr>
<tr>
<td></td>
<td>0.807</td>
<td>0.774</td>
<td>0.921</td>
<td>0.923</td>
</tr>
<tr>
<td></td>
<td>0.032</td>
<td>0.116</td>
<td>0.559</td>
<td>0.186</td>
</tr>
<tr>
<td></td>
<td>0.862</td>
<td>0.680</td>
<td>0.955</td>
<td>0.698</td>
</tr>
<tr>
<td></td>
<td>0.783</td>
<td>0.754</td>
<td>0.921</td>
<td>0.776</td>
</tr>
<tr>
<td></td>
<td>0.150</td>
<td>0.071</td>
<td>0.008</td>
<td>0.031</td>
</tr>
<tr>
<td></td>
<td>0.884</td>
<td>0.733</td>
<td>0.864</td>
<td>0.831</td>
</tr>
<tr>
<td></td>
<td>0.824</td>
<td>0.759</td>
<td>0.772</td>
<td>0.774</td>
</tr>
<tr>
<td></td>
<td>0.306</td>
<td>0.128</td>
<td>0.006</td>
<td>0.052</td>
</tr>
<tr>
<td></td>
<td>0.870</td>
<td>0.788</td>
<td>0.900</td>
<td>0.902</td>
</tr>
<tr>
<td></td>
<td>0.854</td>
<td>0.791</td>
<td>0.781</td>
<td>0.800</td>
</tr>
</tbody>
</table>

Table C.3: Data-set From objective evaluation using TMQI, 14 panoramas are identified in numbers, each associated with four regions. while N is for Naturalness, S is for Structural Fidelity and Q is for TMQI Score.