PhETA: An Interactive Tool for Analyzing the Quality of Digital Photographs from Edge Transitions

Anthony James Allowatt

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Roger Ehrich, Chair
James D. Arthur
Stephen Edwards

November 15, 2005
Blacksburg, Virginia

Keywords: Digital imaging, picture processing, edge detection, interpolation

Copyright 2005 Anthony James Allowatt
PhETA: An Interactive Tool for Analyzing the Quality of Digital Photographs from Edge Transitions

Anthony James Allowatt

(ABSTRACT)

The goal of this thesis is to build an interactive tool for analyzing the quality of a digital image and predicting the scale at which it may be published. Since edges are present almost everywhere in most digital images, we use a mathematical edge model as the basis of analysis. In particular, we are interested in the luminance and chromaticity behavior at edge boundaries. We use this model to develop PhETA – Photograph Edge Transition Analyzer – an interactive tool that allows novice users to view and understand the results gained from this analysis in a clear and simple manner.
Acknowledgements

I would like to thank my advisor Roger Ehrich for his constant guidance and direction throughout my time on this thesis, and committee members James D. Arthur and Stephen Edwards for their continued support both in and out of class.
# Contents

1 Introduction

1.1 Image Quality Factors .................................................. 2
1.2 Goals .............................................................................. 4
1.3 Related Work ................................................................. 4
1.4 Testing Conditions ........................................................... 5
1.5 Structure of This Document ............................................... 5

2 Digital Imaging Background .................................................. 6

2.1 Sampling and Quantization ................................................ 6
2.2 Digital Image Topology ....................................................... 7
2.3 Digital Camera Construction and Operation .......................... 8
2.4 Measuring Luminance ....................................................... 12
  2.4.1 HSV: Hue/Saturation/Value ......................................... 13
  2.4.2 HLS: Hue/Luminance/Saturation ................................... 13
  2.4.3 YIQ and YUV ............................................................ 13
  2.4.4 Choosing the Best Luminance Measurement .................. 14
2.5 Theoretical Basis for Edge Analysis ...................................... 15
2.6 Edge Detection Foundations ............................................. 16
  2.6.1 Types of Edges ........................................................ 17
  2.6.2 Spatial Filtering ......................................................... 18
2.6.3 Traditional Edge Detection Techniques ............................ 18
2.6.4 Canny Edge Detector ........................................... 22
2.6.5 The Laplacian and Image Sharpening ............................ 23
2.7 Modulation Transfer Function ...................................... 25

3 Sub-pixel Level Image Interpolation ................................. 27
3.1 First Approach: Trigonometric Series .............................. 28
  3.1.1 Increasing the Resolution of the DFT ........................ 29
  3.1.2 The DFT Method and Discontinuities ....................... 32
  3.1.3 Windowing Functions ........................................ 33
  3.1.4 Conclusions from the DFT Method .......................... 34
3.2 Second Approach: Spline Surface Interpolation ................. 34
  3.2.1 B-Spline Theory ........................................... 35
  3.2.2 Results of Various Order Interpolants ..................... 38
  3.2.3 Conclusions ................................................ 40

4 Designing and Implementing PhETA ................................. 41
4.1 Goals .............................................................. 41
4.2 Development Environment .......................................... 42
4.3 Design Philosophies ............................................... 42
4.4 User Interface Overview ........................................... 43
  4.4.1 Available Analysis Tools ................................... 44
  4.4.2 Edge Detection and Visualization ........................... 47
  4.4.3 Document Area ............................................. 48
  4.4.4 Quick Info Pane ............................................. 48
  4.4.5 Profile Graph Pane ......................................... 49

5 Results ................................................................. 54
## List of Figures

2.1 The sampling and quantization processes. ........................................ 7
2.2 A Bayer array, one of the most common CCD arrangements in digital cameras. ... 8
2.3 Examples of Bayer array demosaicking algorithms. ............................... 9
2.4 Three common color spaces used in image processing. .......................... 12
2.5 Example of a realistic ramp edge. .................................................. 17
2.6 An ideal edge ramp and its derivatives ........................................... 19
2.7 Example of edge detection with the Sobel operator. ............................ 21
2.8 Canny edge detector with various parameters. ................................. 23
2.9 The visual phenomenon known as Mach banding. .............................. 24

3.1 The absolute value function $|x|$ resampled with the DFT method .......... 31
3.2 Increasing the resolution of a 2-dimensional DFT. .............................. 31
3.3 A near-ideal step edge resampled with the DFT method. ...................... 32
3.4 Step edges resampled with the DFT in two dimensions. ...................... 32
3.5 The effects of some common windowing functions on a step edge .......... 34
3.6 The B-spline basis functions of orders 0 through 3. .......................... 37
3.7 Interpolating a step edge with various order B-splines. ...................... 39
3.8 Step edges resampled with various order B-splines in two dimensions. .... 39

4.1 The main PhETA application window. ........................................... 44
4.2 The pixel inspector tool. ................................................................. 45
4.3 The profile line tool. ................................................................. 46
4.4 The edge detection and visualization dialog. ........................................ 47
4.5 The Quick Info pane. ................................................................. 49
4.6 The Profile Graph pane, showing the spatial-domain Image Profile plot. .... 50
4.7 Image metrics computed by PhETA in the spatial domain. .................... 51
4.8 The Profile Graph pane, showing the frequency-domain MTF plot. .......... 52
Chapter 1

Introduction

In recent years digital cameras have become more popular for the average consumer, thanks to significant reductions in cost and advancements in ease of use and image quality. Digital cameras are of a significantly different construction than that of their film counterparts, however, and users should be aware of these intricacies in order to obtain the best results from their equipment.

Many amateur photographers who use film cameras may take for granted that they can simply “point and shoot” and then manipulate the resulting photograph – creating an enlarged print is a common task, for instance – without giving considerable thought to the quality conditions inside the camera. Digital cameras, being far younger by comparison, are not quite so “foolproof,” so it would be beneficial for users to be able to better understand and analyze the quality of their images.

The work presented here addresses a deficiency in the digital photography application market; there does not exist a tool directed toward amateur end-users that gives them the ability to analyze their images with regard to image metrics and view the results in a straightforward manner.
1.1 Image Quality Factors

Image quality tends to be a somewhat subjective, perceptual concept, so if we intend to measure quality, then we must consider which particular attributes and artifacts contribute to the perceived quality of the final image, and define ways to quantify them. We describe some of these factors below.

The resolution of the camera is an obvious measure of its quality. When we refer to resolution, we mean not only the final dimension in pixels of the image, but how much detail the camera is able to resolve. For example, consider a striped or checkered pattern. The resolution of the camera determines whether this pattern will be captured properly or if it will degenerate into a solid gray level or develop moire patterns.

There are many kinds of artifacts that can harm the quality of photographs. Chromatic aberrations occur because different wavelengths (that is, colors) of light are refracted at slightly different angles by a curved lens. This means that for a particular point in a scene being photographed, the individual red, green, and blue components of the light at that point may potentially be captured by different, but nearby, sensors. This phenomenon is most clearly observed on edge transitions, where color components from light on one side of the edge bleed across to the other side of the edge.

A type of discoloration that is perceptually similar to the one above but with a different cause is “sensor blooming,” which occurs in regions of the image that transition from very bright or saturated light to an area of less saturation. In this case, the sensors capturing the saturated light become overpopulated by electrons, and these excess electrons spill over into adjacent active sites, oversaturating the neighboring pixels as well. This tends to produce a colored halo – most frequently purple – that traces the edge transition.

The demosaicking process is another cause of discoloration. This process will be described in more detail in Chapter 2, but in short, most digital cameras only sample a single color component (red, green, or blue) at each pixel, so the other two components must be interpolated from the
surrounding context. The accuracy of the interpolated colors depends on the quality of the de-
mosaicking algorithm used by the camera; a poorly designed algorithm will incorrectly interpolate the
colors across edges.

Another measurable aspect of an image deals less with the sensors and other hardware in
the camera, but with the manner in which the camera’s software enhances the images. Many
cameras perform a degree of sharpening on the images that they capture, in order to present what
is presumed to be a better looking result for most users. This sharpening is created by adding
overshoot to each side of the edges in the image; that is, pixels along the light side of the edge are
brightened further, and pixels along the dark side are darkened. Our visual system perceives this as
a more well-defined edge, which causes the image to seem sharper. However, if the amplitude of the
overshoot is too great, this can result in ghosting around the edges. By measuring this overshoot,
we can determine how sharp an image may be perceived by a viewer.

The previous types of artifacts affect critical edge transitions, but uniform regions are also not
immune to aberrations. Noise manifests itself as small deviations in pixel values over a neighbor-
hood, and can be either luminance-based (much like film grain) or appear across multiple color
channels. The two most common causes of noise are small variations in the intensity of light that is
sampled during long exposures and the signal interference generated by the electrical components
in the camera itself.

Factors external to the camera can also contribute to a poor image. Motion blur can result when
a photograph of a moving object is taken, depending on the velocity of the object and the shutter
speed of the camera. This blurring effect is usually restricted to the immediate neighborhood of
the object in motion. If the camera itself is moving, however (due to an unsteady camera operator,
for instance), then the entire image is likely to be defocused.

Lastly, most low-to-mid-range cameras support only lossy JPEG compression, as opposed to
providing a way to retrieve the raw image data or a lossless TIFF-formatted file. Depending on
the level of compression used, JPEG artifacts may not be easily visible when images are viewed at
100% zoom but could become objectionable if the image is enlarged.

1.2 Goals

The goal of this project is to develop an interactive tool that allows users to analyze the quality of their digital images. To this end we have created a mathematical model that is used to quantify many of the factors detailed above. The tool makes use of this model in performing its calculations and presents the results to the information gathered in a manner that can be understood by the novice consumer.

1.3 Related Work

*Imatest* is an application written by Norman Koren to measure the quality of cameras and scanners, and it performs some computations similar to those presented in this work. *Imatest* is a camera-centric tool, however, while PhETA is designed to be image-centric. The distinction is reflected by the manner in which the tools are operated. Users of *Imatest* are instructed to examine edges that are oriented at a particular angle – in the neighborhood of $5^\circ$ – from standardized camera test charts or custom targets with similar properties. This allows very specific algorithms and calculations to be made so that consistent results can be obtained for a particular camera being tested or that the performance of different cameras can be compared.

*Imatest* also performs some slight manipulation of the image data in order to fairly compare cameras. For instance, it obtains what Koren refers to as “standard sharpening” for an image – desharpening an oversharpened image or sharpening an undersharpened image. This allows for better evaluation of a camera’s lens, sensors, and other physical characteristics rather than its sharpening algorithm, which in many mid-to-high-range cameras is adjustable by the user.

PhETA, on the other hand, is designed to analyze the images themselves, so we make no modifications to the image data being tested. Furthermore, we place no limits on the orientation...
or extent of the edges that can be investigated. This approach increases the implementation complexity, but it allows us to develop a tool that is general enough that it can be used to analyze any source image with ease.

1.4 Testing Conditions

Test images used in this document were taken either with a Canon PowerShot A520 digital camera (released February 2005) or a Polaroid PhotoMAX PDC 2300Z digital camera (released early 2001), unless otherwise specified. The A520 is a 4.0 megapixel camera with a maximum resolution of $2272 \times 1704$; the 2300Z, 2.3 megapixels at $1792 \times 1200$. Since the 2300Z does not have user-adjustable shutter speed or aperture settings, the automatic mode will be used on the A520 to create as fair comparison conditions as possible. Both cameras deliver their images as JPEG-compressed files.

Development and testing was performed on a Toshiba Satellite 1955 laptop equipped with a 2.53 GHz Intel Pentium 4 CPU and 512 MB of RAM running Windows XP.

1.5 Structure of This Document

Chapter 2 provides an brief introduction to digital camera hardware and digital imaging concepts such as edge detection that are crucial to the remaining work in this thesis. Chapter 3 explores several approaches to image interpolation and evaluates their effectiveness. Chapter 4 describes the design goals of PhETA and how it was implemented. Finally, Chapter 5 uses the tool to analyze some real images and interpret the results.
Chapter 2

Digital Imaging Background

In order to properly analyze the quality of an image captured by a digital camera, we must know not only how the camera samples light from a scene in order to generate a discrete set of pixels, but also what post-processing the camera’s software performs to complete the image before it is presented to the user.

We first introduce some basic theory about the sampling process and how digital images are represented. Then, we bring this theory into practice by presenting a general overview of the construction of digital cameras and how sampling actually occurs in the hardware.

Afterward, we introduce some mathematical foundations to image post-processing, with a focus on traditional edge detection techniques that lead to the approaches we have used in the following chapters.

2.1 Sampling and Quantization

When a digital camera photographs a continuous scene, this scene must be digitized into a discrete grid of pixels that will constitute the image. This is achieved when the sensor in the camera captures light at equally spaced points in the overall scene, a process that is known as sampling.

A real scene is not only continuous in space; it is also continuous in intensity. Before the
Intensities at those sampled points can be recorded in an image; they must be converted into a set of discrete quantities—for example, a camera might record the values as integers between 0 and 255 for storage in a single byte. This process is called quantization.

Figure 2.1 gives an example of how the sampling and quantization processes perform the conversion from a continuous to a discrete signal.

2.2 Digital Image Topology

After a scene has been sampled and quantized, the digital image can be represented as a function of two variables $f(x, y)$ such that $x$ and $y$ represent the integer coordinates of the pixels in the image. In the case of a grayscale image, the function values represent the luminance (intensity) at a particular point $(x, y)$.

To extend this to color images, we have the option of treating a function $f(x, y)$ as a single color channel (red, green, or blue, for example), or we can define $f(x, y)$ to return a vector that contains the intensity values for all channels.

As discussed above, since the sampling process digitizes a scene into a discrete grid of pixels, the function $f(x, y)$ has values only at integer coordinates. At all other locations the function is undefined. This fact will play an important role in the need for interpolation that we will discuss.
in Chapter 3.

2.3 Digital Camera Construction and Operation

Most digital cameras are comprised of a rectangular array of charge-coupled device (CCD) sensors that sample light from a continuous scene to produce a grid of discrete pixels that is further processed by software in the camera to create the final image. For an image region with little variation in color or intensity, this discrete sampling does not pose a problem, but near edge transitions – particularly those between highly contrasting regions – the manner by which the camera reconstructs those edges greatly affects the perceived quality of the final image.

In the case of color images, with the exception of the Foveon sensor [8], single-sensor cameras do not sample the red, green, and blue components at each sensor cell. Instead, they use an arrangement such as the Bayer array (Figure 2.2), which samples green light on half of the sensors, and red and blue on equal numbers of the remaining sensors. The green channel is given a higher priority because it correlates most highly with the luminance of the image, which is perceptually more important than the chrominance information.

Because each sensor only captures a single color component, software inside the camera must calculate the missing two components for each pixel in the image, a process called “demosaicking”
Figure 2.3: Examples of Bayer array demosaicking algorithms.

A poor demosaicking algorithm can result in moire patterns and severe discoloration in the region around the edge. While many demosaicking algorithms can be found in the published literature, most are manufacturer-proprietary.

Figure 2.3 shows the results of some published demosaicking algorithms on some test images. The first two images were taken with the Polaroid PDC 2300Z camera; the first in low-light, low-contrast conditions, and the second in a more luminous outdoor setting. The final two images are common black-and-white test patterns that were acquired using a Hewlett-Packard 5470c flatbed.
The leftmost image in each row is the original image as it was obtained from the camera or scanner. For each image, the pixels were stripped of all but one color channel in a pattern corresponding to that of the Bayer array diagram in Figure 2.2, to simulate the data as it would be retrieved from the sensor. This data was then processed by three demosaicking algorithms that are briefly described below.

It should be stressed that since the original images in the first two cases were retrieved from a digital camera that uses a Bayer array sensor layout, these images are effectively being demosaicked twice in this example, which will reduce the quality of those images. However, this example is being presented merely as an illustration of some simple demosaicking algorithms, and we make the assumption that the algorithm used in the actual camera from which these images were acquired is “good enough” that we can use them as a reasonable approximation of the original scene.

**Bilinear Interpolation**

A simple bilinear interpolation algorithm can be implemented in five passes. First, the entire green plane is interpolated on the red and blue centers by performing a simple average of the four surrounding known green pixels.

The second and third passes interpolate the red pixels; first, at the blue centers by averaging the four diagonally adjacent known red values, and then at the green centers by averaging the two vertically adjacent known red values along with two horizontally adjacent red values that were interpolated during the previous pass.

The fourth and fifth passes interpolate the blue pixels in a similar fashion.

Bilinear interpolation for demosaicking does not use any contextual information about the image, such as the locations of edges, in its calculations. This results in an image that appears significantly defocused, because pixels near edge boundaries will be averaged with pixels in the neighborhood on each side of the edge.
Muresan, Luke, and Parks

The demosaicking algorithm from [4] can be implemented very similarly to bilinear interpolation. It is also a five-pass algorithm, in which the green channel is interpolated first, and then two passes each are required for the red and blue channels.

This algorithm differs from bilinear interpolation in that local first-order derivatives are calculated to help prevent interpolation across edges. That is, if a pixel being interpolated lies on an edge, the surrounding averaging neighborhood will be restricted to pixels parallel to the direction of the edge. This minimizes the defocusing effect that occurs with simple bilinear interpolation.

Hamilton and Adams

The algorithm presented in [9] is a two-pass process. During the first pass, the entire green plane is interpolated, using a $5 \times 5$ cross-shaped interpolating neighborhood that calculates first-order derivatives (for the green data) and second-order derivatives (for the red/blue data) to improve the results around edge boundaries, which we want to interpolate along rather than across.

The second pass interpolates simultaneously the red and blue planes using a square $3 \times 3$ neighborhood. Conversely to the first pass, first-order derivatives are calculated for the red/blue data, and second-order for the green data.

The use of higher-order derivatives produces markedly improved results in all of the cases above.

Effects on Analysis

Most common digital cameras contain demosaicking algorithms that are far more complex than the three examples cited above. Unfortunately these algorithms are usually proprietary and very rarely is any information about them made publicly available, which creates some difficulties in being able to thoroughly analyze the quality of a digital image versus the actual scene it represents. Without knowing precisely how the camera is operating in this regard, we must accept that an image will contain camera-specific artifacts.
2.4 Measuring Luminance

As discussed above, digital cameras capture light in terms of red, green, and blue signals that are combined to form color. Output devices that display these images likewise operate in RGB space. Humans, however, do not interpret colors as a percentage of red, green, and blue energy. Rather, we tend to view color in terms of chromaticity (a combination of hue and saturation) and luminance (which goes by a variety of other names, such as lightness, brightness, value, and intensity).

We wish to evaluate edge transitions that primarily occur due to shifts in the luminance that we perceive in an image, so we need to be able to convert our data from RGB space into a color space that has a luminance component. However, different color spaces use different methods to calculate luminance. Some methods are solely energy-based; that is, given the amount of the red, green, and blue energy at a point, the luminance is determined from a simple selection or average of these. Other methods are perceptual, meaning that they adjust the RGB values based on knowledge of the human visual system. We investigate some of these methods below.
2.4.1 HSV: Hue/Saturation/Value

The HSV model is defined in a cylindrical coordinate system in which the colors fall inside a cone (see Figure 2.4a). Hue values fall between $0^\circ$ and $360^\circ$ and represent points around the base of the cone. Value falls on the cone’s axis, ranging from 0 at the apex of the cone to 1 at its base. Saturation is measured perpendicular to the axis, ranging from 0 at the center to 1 at the surface. Note that this implies that the hue contributes less information as saturation approaches 0; indeed, when saturation equals 0, hue is irrelevant.

Value, in this color space, is calculated simply by choosing the maximum of the red, green, and blue values. Thus it is an energy-based conversion.

2.4.2 HLS: Hue/Luminance/Saturation

The HLS color space is a slight variation of HSV, formed by joining two cones at the base (see Figure 2.4b). Thus, total saturation occurs only when luminance is 0.5, eliminating the problem of adjusting the saturation of pure white or black. In this color space, luminance is calculated as the average value of the maximum RGB component and minimum RGB component. Therefore, like HSV, it is energy-based.

2.4.3 YIQ and YUV

The YIQ color space is primarily used by the NTSC television standard to transmit color information, YUV by the PAL television standard. These color spaces are represented by a cube; Figure 2.4c shows the UV-plane when Y is fixed at 0.5.

The Y component represents luminance, while IQ/UV represent chromaticity. The conversion from YUV to RGB is a simple linear transformation defined by

$$
\begin{pmatrix}
Y \\
U \\
V
\end{pmatrix} =
\begin{pmatrix}
0.299 & 0.587 & 0.114 \\
-0.147 & -0.289 & 0.436 \\
0.615 & -0.515 & -0.100
\end{pmatrix}
\begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
$$

(2.1)
A similar conversion exists for YIQ. Since we are only interested in the Y component, it suffices to note that the top row of the transformation matrix is identical in YIQ and YUV.

Note the computation of the luminance component: \( Y = 0.299R + 0.587G + 0.114B \). The green component of the original data carries nearly twice as much weight as the red component, and five times that of the blue component. This is based again on the fact that green light is more highly correlated with perceived luminance than red or blue light. Thus, luminance in YIQ or YUV space is a perceptual measure.

### 2.4.4 Choosing the Best Luminance Measurement

Since we are primarily concerned with how a person viewing an image will perceive the strength or quality of an edge transition, this implies that a color conversion with a perceptual luminance value would be preferred to one that is energy-based. YIQ and YUV are the only spaces discussed above that satisfy this.

The benefits of using this Y component are two-fold. First, it allows us to define edge transitions based on *what the user sees* rather than *what the camera recorded*. Secondly, recall that the Bayer array discussed above used twice as many green sensors as it did red and blue sensors. By employing a luminance measure that weighs green more heavily, we will gain more accurate results because we are making use of more actual captured data, versus data that was interpolated during the demosaicking process.

Therefore, for the rest of this paper, when we refer lightness or luminance at a point in an image, we are referring to the quantity Y as defined by the YIQ and YUV color spaces. We will also refer to a “modified HSV” space, which uses the hue and saturation definitions of HSV, but with luminance calculated using the perceptual Y component of YUV/YIQ.
2.5 Theoretical Basis for Edge Analysis

In systems theory, a system $f(x)$ is linear if

$$f(a \cdot x_1(k) + b \cdot x_2(k)) = a \cdot f(x_1(k)) + b \cdot f(x_2(k)),$$

where $a, b$ are scalars and $x_1(k), x_2(k)$ are data sequences indexed by $k$.

The system is spatially invariant if

$$y(k - m) = f(x(k - m));$$

that is, a shift in the input causes the same shift in the output.

Digital cameras are neither linear nor spatially invariant. A Bayer array sensor is not spatially invariant because small image shifts will cause the same pixels to be captured by active sites that filter different colors of light. They are also not linear in intensity, because the sensor cannot capture an infinite amount of light – eventually active sites will become saturated, and worse, the oversaturation may cause sensor blooming in adjacent sites. However, linear system theory still provides a basis for edge analysis.

We can predict the behavior of this system for any input if we know its impulse response by using convolution, which is defined in a continuous space as

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(t)g(x - t)dt.$$

Let $i(x)$ be the input to the system and let $n(x)$ be an additive noise process. Then the response of the system to $i(x)$ is

$$o(x) = (i(x) + n(x)) * h(x)$$

$$= i(x) * h(x) + n(x) * h(x)$$

$$= \int_{-\infty}^{\infty} i(t)h(x - t)dt + \int_{-\infty}^{\infty} n(t)h(x - t)dt$$

where $h(x)$ is the impulse response, which describes the behavior of the system. In optical systems, $h(x)$ is called the point spread function. It is this $h(x)$ that we wish to assess.
To measure $h(x)$ we would need to locate in the image input an isolated impulse $\delta(x)$, in order to compute

$$\delta(x) * h(x) = h(x),$$

where

$$\delta(x) = \begin{cases} 1, & \text{if } x = 0, \\ 0, & \text{otherwise}. \end{cases}$$

Unfortunately, a photographed scene rarely contains isolated impulses. It usually does, however, contain a great number of edge transitions, and it is from these edges that we can derive the impulse response.

Using the linearity property of the system, we can write

$$h(x) = \delta(x) * h(x)$$

$$= \frac{\partial}{\partial x} \left[ \int \delta(x) * h(x) \right]$$

$$= \frac{\partial}{\partial x} [u(x) * h(x)]$$

where $u(x)$ is the step input. The expression $u(x) * h(x)$ is the response of the camera to an ideal scene edge – that is, the resulting image – and this equation tells us that the edges in the image contain all the information necessary for us to compute the point spread function. It remains to determine how to compute the derivative of this expression in a discrete space, which is the topic of the next section.

### 2.6 Edge Detection Foundations

When we examine an image, we can consider regions in which there is little change in intensity, and regions where there is greater change. These areas of greater change form the edge transitions in the image, and the quality of these edges contributes to the quality of the overall image. For instance, consider an image that has a certain degree of inherent defocusing. Solid regions will be mostly unaffected by the blurring effect, but the detail in the edge transitions will be lost.

16
In order to evaluate these transitions, we need a robust method to detect them and determine their strength compared to other edges and regions in the image. Edge detection is a very well-studied area in the field of image processing, and in the following sections we describe some of the most common approaches to this problem.

2.6.1 Types of Edges

The most common types of edges are step edges and ramp edges. The step edge is what we consider an ideal edge – an instantaneous transition from intensities on one side of the edge to intensities on the other side. Such edges are rarely captured even under the best conditions due to random noise, the quality of the lens and the sensors in the camera, the sampling process, and demosaicking, among other factors. The effect of this is that an edge will appear more like a ramp function than a step function. The slope of the ramp is inversely proportional to the degree of smoothing on the edge.
See Figure 2.5 on the previous page for an example of a realistic edge. Note the noise that is prevalent along the edge and the “solid” regions. If we examine a profile taken in the direction across the edge (the vertical axis represents luminance), we see that the noise causes small variations in the grey level in the vicinity of the edge, and the edge itself is a ramp, rather than an ideal step. A close inspection of the light side of the edge also reveals a narrow strip of lighter pixels to the left of the edge transition. This is a result of sharpening known as Mach banding and is discussed later in this section.

2.6.2 Spatial Filtering

The traditional edge detectors discussed below are implemented as spatial filtering masks. Spatial filtering is accomplished by cross-correlating a filter mask with the image to produce a new image. A filter mask \( g(i,j) \) is centered on each pixel \( f(x,y) \) in the image, the overlapping values are multiplied, and then these products are summed. The result becomes the value of the pixel \( h(x,y) \) in the new image. That is, for an \( n \times n \) filter mask, the filtered image can be computed by

\[
h(x,y) = \sum_{i=-m}^{m} \sum_{j=-m}^{m} f(x+i,y+j)g(i,j)
\]  

where \( m = (n-1)/2 \).

Since \( O(n^2) \) calculations are required to compute a single pixel in the filtered image, the size of the filter masks is usually kept very small. The edge detectors introduced below, with one exception, are \( 3 \times 3 \) masks.

2.6.3 Traditional Edge Detection Techniques

Edge detection using spatial filtering is based on the derivative operator, or in the case of digital images, approximations of the derivative using discrete differences. Consider the one-dimensional function and its derivatives shown in Figure 2.6 on the following page. A filter mask designed to approximate a first derivative must have a response of zero across a constant region and a nonzero
response at the ends of and along ramps. Likewise, a second derivative mask must have a zero response across constant regions and ramps, and a nonzero response at the ends of ramps.

The simplest approximation of the first derivative uses forward differences:

$$\frac{\partial f}{\partial x} = f(x + 1) - f(x). \quad (2.3)$$

We can see that this satisfies the conditions stated above for a first derivative operator. This difference is equivalent to a cross-correlation with the mask $[-1 \ 1]$.

Another common approximation that is less sensitive to sensor noise uses central differences:

$$\frac{\partial f}{\partial x} = f(x + 1) - f(x - 1). \quad (2.4)$$

This difference is equivalent to a cross-correlation with the mask $[-1 \ 0 \ 1]$.

These one-dimensional approximations will be recalled later when building the two-dimensional operators.

To extend this to two-dimensional images, we use the gradient vector

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} \quad (2.5)$$

where the partial derivatives are now two-dimensional masks that are constructed in a similar fashion to the differences above. Note that while the $x/y$ notation seems to imply that the masks must be horizontally and vertically oriented, any two orthogonal masks can be used.
Given the gradient vector at a particular point, we can determine the strength of an edge by calculating the magnitude $\nabla f$ as follows:

$$\nabla f = \| \nabla f \|_2 = \left[ \left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2 \right]^{1/2} \quad (2.6)$$

However, since the square root makes this computationally difficult, an approximation is typically used in practice:

$$\nabla f \approx \left| \frac{\partial f}{\partial x} \right| + \left| \frac{\partial f}{\partial y} \right| \quad (2.7)$$

For the purposes of this work, we also need to be able to compute the orientation of an edge. We compute the quantity

$$\theta = \tan^{-1} \left( \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right) \quad (2.8)$$

which represents the angle of the vector $\nabla f$. This vector points in the direction of maximal change – that is, it points across outward from the edge – so the direction of the edge itself is orthogonal to $\theta$.

All that remains is what masks should be used to approximate the derivatives $\partial f/\partial x$ and $\partial f/\partial y$. The most common masks used in practice are briefly discussed below.

**Roberts Cross-Gradient**

The Roberts cross-gradient operator is one of the simplest ways of implementing a first derivative operator. It uses $2 \times 2$ masks that detect discontinuities at orientations of $45^\circ$ and $-45^\circ$ – these masks are rotations of the one-dimensional forward differences given in Equation (2.3).

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

While their small size makes them computationally efficient, it also makes them very sensitive to noise in the image. Also, masks with even dimensions are less desirable because they cannot be centered precisely on the pixel being calculated.
The Prewitt operators overcome some of the problems of the Roberts cross-gradient by using $3 \times 3$ masks. These masks are based on the central differences of Equation (2.4), and detect horizontal and vertical discontinuities.

$$
\begin{bmatrix}
-1 & -1 & -1 \\
0 & 0 & 0 \\
1 & 1 & 1
\end{bmatrix}
\quad \begin{bmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1
\end{bmatrix}
$$

The Sobel operators are a slight modification of the Prewitt operators. By weighting the center coefficients twice as much as the outlying pixels, the Sobel operators suppress noise better than Prewitt and produce a stronger response on the edge itself.

$$
\begin{bmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1
\end{bmatrix}
\quad \begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{bmatrix}
$$

Figure 2.7 gives a visual representation of the results of using the Sobel operator to detect edges. This example takes a color image, extracts the luminance channel information, and then computes the gradient magnitude at each point. For the purposes of visualization, these magnitude values
form the luminance channel of a new image; dark values indicate low magnitude (areas where edges were not found or are weak), and brighter values indicate strong edge transitions. The Roberts-Cross and Prewitt masks also produce similar results.

2.6.4 Canny Edge Detector

The edge detectors discussed thus far have been purely mathematical; that is, they can be viewed as functions that take as input an image and a set of masks, and produce another “image” – a description of the edge strength at each point – as the result.

Canny [3] devised an algorithmic approach to edge detection that is viewed as the optimal edge detector. Rather than computing the edge strength from simple functions, he developed a process that meets the following criteria:

- The process should simply return a value indicating whether an edge exists or does not exist at a given point. This is in contrast to the previously discussed edge detectors, which return a set of values indicating relative rates of change.

- Edges should be as continuous as possible; that is, there should not be small gaps between edge segments in the event that the gradient magnitude falls briefly below a particular value.

- There should only be a single response to a given edge.

The process works as follows. First the image is smoothed to eliminate noise, which removes small local fluctuations that get amplified when derivatives are calculated. Canny uses Gaussian filtering to accomplish this smoothing, due to the ease with which the mask can be computed for a filter of a particular radius (standard deviation).

Once the image is smoothed, the gradient magnitudes are computed using one of the detectors described above. Nonmaximum suppression is then performed on these results; this has the effect of thinning the edges so that only the strongest response in the vicinity of a particular point remains.
Finally, hysteresis is used to prevent small gaps or other fluctuations as stated in the criteria above. Canny uses a low threshold and a high threshold to determine whether the remaining points belong to an edge. If the magnitude at a point is below the low threshold, it is not an edge; if the magnitude is above the high threshold, then it is an edge. If it lies between the thresholds, then it is an edge point only if there exists a path from that point to another point that lies above the high threshold.

The result of this is that the Canny edge detector produces a much more refined edge profile than the other methods, with one drawback – appropriate values for the smoothing and threshold parameters must be chosen, and this can involve some trial and error. Figure 2.8 shows some examples of the Canny detector using a fixed Gaussian standard deviation ($\sigma$) and various low and high hysteresis thresholds ($l$ and $h$, respectively).

### 2.6.5 The Laplacian and Image Sharpening

The Laplacian is a rotation-invariant second derivative operator, defined as follows:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$  \hspace{1cm} (2.9)

Two two-dimensional approximations of this equation that satisfy the conditions for second derivatives stated at the beginning of the chapter are shown below. To compute the Laplacian, we simply...
Uses of the Laplacian for edge detection are limited, because it generates a response on each side of an edge rather than on the edge itself, and as a second derivative operator it is highly sensitive to noise. However, it has a very important application in image sharpening.

Image sharpening algorithms take advantage of a phenomenon in the way that humans perceive image boundaries. When two contrasting regions are juxtaposed, the eye tends to see a darker strip on the dark side and a lighter strip on the light side along the edge. This is known as Mach banding and can be better illustrated by the graph in Figure 2.9.

The key to sharpening is to duplicate this overshoot around edges to make those edges appear more pronounced to the observer. The Laplacian helps us do exactly that, since it produces a positive response at one side of the edge and a negative response at the other side. By adding (or subtracting, depending on the sign of the Laplacian mask coefficients) a fraction of the Laplacian to the image itself, we can introduce a small amount of overshoot at the edge boundaries while leaving untouched any regions of constant luminance or ramps.
This is only one of many methods used to perform image sharpening, however. Another popular one found in many image processing applications is *unsharp masking*, which involves subtracting a blurred copy of the image from the original image.

An understanding of image sharpening techniques is important because many digital cameras perform one of these methods on the images they capture before they are presented to the user. In more advanced cameras it may be possible to deactivate this feature, but many consumer-range cameras do this unconditionally. If the original (pre-sharpened) image is not retained, then this sharpening is an irreversible process, so we must be aware of any overshoot that may be inherent in the image data when we do our analysis.

### 2.7 Modulation Transfer Function

As discussed previously, the rise time of an edge is a good measure of its sharpness and quality; the lower the rise time, the closer the edge is to an ideal step. This spatial domain characteristic, however, represents the output of the entire imaging system – the lens, the sensors, the demosaicking algorithm, and any other post-processing that may have been performed on the image.

For this reason, calculations are often also performed in the frequency domain, using the Fourier transform to calculate the *modulation transfer function* (MTF). In the frequency domain, the response of a complete imaging system is simply the product of the responses of its components; thus, if the responses of some components is known, it is possible to isolate other components and calculate their individual responses.

In our work, we *are* in fact interested in the output of the entire imaging system, rather than individual components, so this property does not provide us a great advantage. However, the MTF still provides a reliable measure of image quality that we can use alongside the spatial domain characteristics.

Frequency is measured in cycles per unit distance. A cycle is represented by a line pair (alternating black and white lines), and distance in traditional film cameras is measured in millimeters.
Since we are analyzing images acquired directly from digital cameras, using millimeters as a measure of distance is less appropriate. Instead, we will consider cycles per pixel, and by extension, cycles per image height in pixels.

Just as we use rise time to measure edge quality in the spatial domain, we can determine a similar characteristic in the frequency domain to measure quality. A well-known result from the use of frequency domain filtering in digital imaging is that low frequencies correspond to small shifts in image intensity and high frequencies correspond to larger shifts. Since edges in an image are the visual representation of large changes in intensity, we can correlate edge quality with the amount of high frequency content in the image or in a smaller region of interest that we wish to analyze.

The MTF is defined as the magnitude of the Fourier transform of the impulse response (or first derivative) of a data sequence. This function is normalized such that the lowest frequency is valued at 1, and it tends to fall off as frequency increases. If the sequence contains more high frequency data, it will fall slowly to zero; in other words, edge quality is inversely related to the rate of MTF falloff. For this reason, the “MTF50” – the point at which the MTF falls to 50% of its initial value – is used as the indicator of quality.

Details of how the MTF is calculated and how the MTF50 is used in PhETA are given in Chapter 4 where the implementation of the tool is discussed.
Chapter 3

Sub-pixel Level Image Interpolation

The calculations that we will perform involve positioning a line segment normal to an edge transition in an image and collecting statistics along that line. Depending on the location and orientation of this line, it is more likely than not that the points we sample will not fall directly pixel centers in the image. For this reason, we need a sub-pixel interpolation method that ensures smooth and unskewed results.

Image interpolation is a well-explored area, but while most applications are concerned with the perceptual results, we are instead concerned with image metrics. To alter the input data as little as possible, our chosen interpolator should meet two criteria:

- The interpolated data should be identical to the original data when evaluated at the lattice points.
- The interpolator should not introduce overshoot between pixels.

Imposing these constraints is crucial to our goal of analyzing only the image in question and not any artifacts that could be introduced in the interpolating process. The goal of this chapter is to evaluate several interpolation methods and select the best one according to the previous conditions.
3.1 First Approach: Trigonometric Series

The most common use of the Fourier transform in image processing is to implement efficient image enhancement operations. Many operations (of which the edge detection operators discussed earlier are a small example) are based on convolving\(^1\) a filter mask with the image in order to produce a new image as a result.

An equivalent way to perform this convolution is to use the convolution theorem from Fourier analysis. This theorem states that the convolution of two functions can be computed by multiplying the Fourier transforms of the functions and then computing inverse Fourier transform of the result. Since digital images require that we use discrete Fourier transforms, these can be computed with a Fast Fourier Transform (FFT) algorithm, which requires only \(O(n \log n)\) calculations, where \(n\) is the number of pixels in the image.

The 1- and 2-dimensional Discrete Fourier Transforms (DFTs) of an image can be computed by using Equations (3.1) and (3.2). The 1-dimensional variants are provided because they will be used to simplify discussion; all of the methods that we will use trivially extend to the 2-dimensional case.

Let \(f(m)\) be defined at integer values of \(m\) for \(0 \leq m < W\), where \(W\) is the length of the sequence of data being transformed. Then the 1-dimensional DFT is computed by

\[
F(p) = \frac{1}{W} \sum_{m=0}^{W-1} f(m)(\cos(pm\omega_W) - i \sin(pm\omega_W)) \tag{3.1}
\]

where \(\omega_W = \frac{2\pi}{W}\).

The 2-dimensional DFT is defined similarly. Let \(f(m,n)\) be defined at integer values \((m,n)\) for \(0 \leq m < W\) and \(0 \leq n < H\). Then

\[
F(p,q) = \frac{1}{WH} \sum_{m=0}^{W-1} \sum_{n=0}^{H-1} f(m,n)(\cos(pm\omega_W + qn\omega_H) - i \sin(pm\omega_W + qn\omega_H)) \tag{3.2}
\]

where \(\omega_W = \frac{2\pi}{W}\) and \(\omega_H = \frac{2\pi}{H}\).

\(^1\)The previous discussion referred to cross-correlation rather than convolution, but the two operations are equivalent if one of the sources is mirrored, so we will ignore this distinction.
Given the DFT of a set of data computed as above, the inverse DFT can be computed by a similar process to regain the original data:

\[ f(m) = \sum_{p=0}^{W-1} F(p)(\cos(pm\omega_W) + i\sin(pm\omega_W)) \]  

\[ f(m, n) = \sum_{p=0}^{W-1} \sum_{q=0}^{H-1} F(p, q)(\cos(pm\omega_W + qn\omega_H) + i\sin(pm\omega_W + qn\omega_H)) \]

If \( f \) represents a set of data that we want to interpolate, first we obtain the DFT of that data by using Equation (3.1) or (3.2). Now, instead of viewing the inverse DFT of Equation (3.4) as a computational process that returns the original data, let us instead view it more abstractly as a trigonometric polynomial surface that matches the values of \( f \) when evaluated at integer coordinates. If we change the discrete variables \( m \) and \( n \) to continuous variables \( x \) and \( y \) and evaluate \( f(x) \) (in the 1-dimensional case) or \( f(x, y) \) (in the 2-dimensional case) at non-integer coordinates, then we obtain values that are interpolated between the original data points.

It is trivial to obtain these values by simply evaluating Equation (3.4) at the points we need, but in the 2-dimensional case, each point requires \( O(WH) \) trigonometric calculations. Depending on the number of points required for a particular computation, this is an unacceptable performance penalty.

### 3.1.1 Increasing the Resolution of the DFT

Fortunately, we can use the FFT to greatly increase the efficiency of this task. A result from Fourier analysis states that the DFT of a sequence of real numbers has a conjugate-symmetric property; that is, \( F(-p) = F^*(p) \). Additionally, since the DFT is periodic, we have \( F(p) = F(W + p) \). This symmetry is key in the method that we will use to increase the resolution of the DFT for interpolation.

Each value in the DFT contributes a sinusoid and a cosinusoid with a particular coefficient to the overall polynomial that interpolates \( f \). For example, in the 1-dimensional case, the value of \( F(1) \) provides the coefficient for the \( \cos x + i\sin x \) term, \( F(2) \) provides the coefficient for the
\[ \cos 2x + i \sin 2x \] term, and so forth.

We can increase the resolution of the DFT by padding the middle with zero-coefficients. By doing so, we increase the period of the function while keeping the function itself the same (down to a scaling factor, because the \( \omega \) frequencies do change with the period), because the zeros do not introduce any higher-order terms.

As an example, consider a nine-point DFT of a set of real data, \( F(0) \) through \( F(8) \).

| \( F(0) \) | \( F(1) \) | \( F(2) \) | \( F(3) \) | \( F(4) \) | \( F(5) \) | \( F(6) \) | \( F(7) \) | \( F(8) \) |

Using the symmetry relationship, we can renumber this as such:

| \( F(0) \) | \( F(1) \) | \( F(2) \) | \( F(3) \) | \( F(4) \) | \( F(-4) \) | \( F(-3) \) | \( F(-2) \) | \( F(-1) \) |

Consider the effect of inserting a zero between the \( F(4) \) and \( F(-4) \) coefficients in the list above. We have now lengthened the period of \( F(p) \), but the shape of the curve (again, down to a scale factor) is identical, because the \( F \) values still represent the coefficients of the same sine and cosine terms. If we now compute the inverse DFT of the modified list, then we are effectively resampling the same curve at different coordinates, distributed equally across one period of the trigonometric polynomial represented by the coefficients.

We can continue to pad the DFT with as many zeros as required to resample at the resolution necessary for a particular problem, but it is most convenient to resample at integral multiples of the length \( W \) of the data sequence. Thus, to increase the resolution \( n \) times, we insert \((n - 1)W \) zeros into the middle of its DFT. An example is shown in Figure 3.1 on the following page.

Increasing the resolution of a 2-dimensional DFT is a simple extension of the 1-dimensional case. To do this, we divide the 2-D DFT into four quadrants, place each quadrant at one corner of an extended grid, and then fill the remaining entries with zeros. An example is shown in Figure 3.2 on the next page.
Figure 3.1: The absolute value function $|x|$ resampled with the DFT method.

| $F(0, 0)$ | $F(1, 0)$ | $F(2, 0)$ | 0 | $\cdots$ | 0 | $F(-2, 0)$ | $F(-1, 0)$ |
| $F(0, 1)$ | $F(1, 1)$ | $F(2, 1)$ | 0 | $\cdots$ | 0 | $F(-2, 1)$ | $F(-1, 1)$ |
| $F(0, 2)$ | $F(1, 2)$ | $F(2, 2)$ | 0 | $\cdots$ | 0 | $F(-2, 2)$ | $F(-1, 2)$ |
| 0 | 0 | 0 | 0 | $\cdots$ | 0 | 0 | 0 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| 0 | 0 | 0 | 0 | $\cdots$ | 0 | 0 | 0 |
| $F(0, -2)$ | $F(1, -2)$ | $F(2, -2)$ | 0 | $\cdots$ | 0 | $F(-2, -2)$ | $F(-1, -2)$ |
| $F(0, -1)$ | $F(1, -1)$ | $F(2, -1)$ | 0 | $\cdots$ | 0 | $F(-2, -1)$ | $F(-1, -1)$ |

Figure 3.2: Increasing the resolution of a 2-dimensional DFT.
3.1.2 The DFT Method and Discontinuities

Examining Figure 3.1 closely, we see that the interpolated points do not quite match the actual function, especially at the slope discontinuity. No finite or infinite sum of continuous functions can model a discontinuous function exactly, and this is to be expected for any interpolation method for which we require any degree of smoothness. The problem is even further exacerbated when we attempt to interpolate a step edge, as in Figures 3.3 and 3.4.

The “ringing” exhibited here is known as *Gibbs’ phenomenon*. These oscillations occur whenever there is a discontinuity in the data being interpolated. As we resample the function at higher resolutions, then the oscillations will become narrower and dissipate more quickly, but their amplitude at the peaks will never vanish. The following sections show our experiments in attempting to
eliminate this problem.

3.1.3 Windowing Functions

Gibbs’ phenomenon is a common and well-studied problem in signal processing, and as such, several methods have been developed to try to lessen its effects. These methods come in the form of windowing functions that smooth the original data to various degrees to make it more amenable to a trigonometric fit.

To obtain a smoother interpolation, we multiply each coefficient of the DFT by a windowing function. Since multiplication of DFTs is equivalent to convolution of the original functions, these windowing functions are usually the Fourier transforms of averaging filters in image space.

The first window we tested was the Lanczos filter, which is the Fourier transform of a rectangular spatial filter. In one dimension, it is represented by the frequency-domain equation

\[ L(p) = \frac{\sin(\pi p/W)}{\pi p/W} \quad (3.5) \]

where \( W \) is the length of our data sequence, as before.

Figure 3.5a shows the results of applying the Lanczos window to the step edge examined previously. The thin line is the data as interpolated without any window applied; the thick line, the data as interpolated with the Lanczos window. This window does a fair job of eliminating the heavy ringing around the flat areas of the curve, but it has the unfortunate side effect that it blurs the step edges as well; the approximation near sharp discontinuities no longer passes through the actual points from the source data. Depending on the nature of the edge being examined, this has unacceptable consequences for the statistics we wish to collect.

Two other popular windowing functions are the Hann and Hamming windows. In one dimension, they are defined in the frequency domain as follows:

\[ \text{Hann}(p) = 0.5 + 0.5 \cos(\pi p/W) \quad (3.6) \]

\[ \text{Hamming}(p) = 0.54 + 0.46 \cos(\pi p/W) \quad (3.7) \]
Figures 3.5b and 3.5c show the results of using these windows. The Hamming window does a particularly good job of flattening the areas before the discontinuity, but it also greatly blurs the actual edge as the Lanczos window did.

### 3.1.4 Conclusions from the DFT Method

Regardless of the window function that we choose, there will always be a tradeoff between how well we can remove the ringing and how accurately the interpolation fits near sharp discontinuities. Modeling these transitions is made especially difficult by the fact that the sinusoidal terms of our trigonometric polynomial have infinite support; thus, it is not possible to manipulate the curve in a local area without greatly affecting the entire approximation.

Using the DFT for image interpolation initially seemed promising due to both its simplicity and efficiency, but the above results have made this approach unacceptable for our purposes.

### 3.2 Second Approach: Spline Surface Interpolation

Spline functions are constructed from a sequence of piecewise polynomials of a particular degree. The points at which the polynomials are joined are called the knots, and at each knot, we require that the polynomials fit together with certain smoothness conditions.

Splines are commonly employed in data fitting problems in one dimension, and they can be easily
extended to two dimensions to fit a surface polynomial to interpolate an image. The most commonly cited advantage of using splines as opposed to polynomial interpolation or the Fourier-based method above is that splines have compact finite support; therefore, sharp local discontinuities can be more accurately modeled without their effects rippling through the rest of the approximation.

There are many spline-based methods that can be used to piece together smooth curves. Bezier curves provide a simple process for constructing an approximation function that is locally controllable but have the disadvantage that the curves do not necessarily pass through the control points. Recall that we wish our interpolating function to be identical to the original data at the integer coordinates, so this makes these curves unacceptable for our purposes.

Methods also exist to compute simple piecewise polynomials of various orders – linear splines, quadratic splines, and so forth. We will use a generalization of these called B-splines, which give us the flexibility to easily experiment with splines of varying degree and have an interesting and very efficient signal processing-based approach to computing the interpolated values between the original data.

3.2.1 B-Spline Theory

Traditionally, calculating the coefficients of the B-spline basis functions involved solving a band-diagonal system of linear equations. However, recently developed signal processing techniques allow B-spline approximations to be computed in a simpler and more efficient manner, by using an approach that is analogous to the Fourier transform. Recall that computing the DFT of a data sequence converts this discrete sequence into a continuous representation, the coefficients of a trigonometric polynomial that fits the sequence. This B-spline method performs a similar conversion, but rather than using sine and cosine as basis functions, it uses \(n\)-degree polynomials that decay very quickly to zero.

The digital signal technique that we will use has a restriction that is not present in more general B-spline methods: the data points to be interpolated must be each unit distance apart. This does
not pose a problem for us because that is precisely the situation that occurs in a digital image.

The order-$n$ B-spline approximation of a 1-dimensional set of uniformly spaced data can be uniquely specified by the sum

$$s(x) = \sum_{k=-\infty}^{\infty} c(k)\beta^n(x - k) \quad (3.8)$$

where $c(k)$ are the B-spline coefficients and $\beta^n$ is the B-spline basis function of degree $n$. These splines are generated recursively by convolving a rectangular pulse with itself $(n + 1)$ times. We begin with the base case:

$$\beta^0(x) = \begin{cases} 
1, & -\frac{1}{2} < x < \frac{1}{2} \\
\frac{1}{2}, & |x| = \frac{1}{2} \\
0, & \text{otherwise} 
\end{cases} \quad (3.9)$$

Higher order splines are then defined simply as follows:

$$\beta^n(x) = \beta^{n-1} * \beta^0(x) \quad (3.10)$$

The spline basis functions of orders 0 through 3 are shown in Figure 3.6 on the following page.

For zero-order and first-order splines, the coefficients $c(k)$ are exactly the data points to be interpolated. Computing $c(k)$ for orders $n > 1$ is more complicated. The details can be found in Unser [19], but in brief, we start by rewriting Equation (3.8) in the form of a convolution:

$$s(k) = (\beta^n * c)(k) \quad (3.11)$$

If we use an inverse convolution operator $(\beta^n)^{-1}$, then we can solve for $c(k)$ by computing

$$c(k) = (\beta^n)^{-1} * s(k) \quad (3.12)$$

This filter can be implemented very efficiently with a two-pass algorithm that requires only $O(n)$ operations for $n$ data points to transform the values $s(k)$ to coefficients $c(k)$. 
Figure 3.6: The B-spline basis functions of orders 0 through 3.
Another advantage of this technique is that this filter is separable, so in order to extend this procedure to 2-dimensional images, we first transform every row in the image and then transform every column of the result. The B-spline expansion is then the tensor product of one-dimensional splines:

\[ f(x, y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} c(k, l) \beta^n(x - k) \beta^n(y - l) \] (3.13)

In performing the actual computation of the above polynomial, we take advantage of the compact support of the \( \beta^n \) splines. The function \( \beta^n \) extends out to a distance of only \( \frac{n+1}{2} \) from its center and is zero-valued elsewhere. Thus, it is only necessary to compute the sums within an interval extending that far from the particular point \((x, y)\) being evaluated.

### 3.2.2 Results of Various Order Interpolants

Higher order splines yield smoother approximations, but in image processing applications, the general rule is that cubic interpolation tends to produce the best visual results; higher order interpolation increases the computational complexity without yielding significantly better results. For this reason we begin by looking at how third-order splines would interpolate our ideal step edge.

Using Equations (3.9) and (3.10), we can obtain a closed form equation for the third-order spline:

\[
\beta^3(x) = \begin{cases} 
\frac{|x|^3}{2} - x^2 + \frac{2}{3}, & 0 \leq |x| < 1 \\
\frac{(2-|x|)^3}{6}, & 1 \leq |x| < 2 \\
0, & \text{otherwise}
\end{cases}
\] (3.14)

If we interpolate our ideal step edge with this B-spline basis function, we obtain the curve shown in Figure 3.7a. This curve unfortunately exhibits a similar overshoot near sharp discontinuities that the Fourier-based methods produced. The rippling does not extend as far out from the discontinuity as it would using the Fourier methods due to the spline’s local support, but the amplitude of the overshoot at the actual discontinuity is nearly as strong. The reason for this is that a cubic B-spline enforces continuity of the function itself and its first and second derivatives, so this overshoot
(a) Cubic interpolation  (b) Quadratic interpolation  (c) Linear interpolation

Figure 3.7: Interpolating a step edge with various order B-splines.

(a) Original image  (b) Bicubic interpolation  (c) Biquadratic interpolation  (d) Bilinear interpolation

Figure 3.8: Step edges resampled with various order B-splines in two dimensions.
is necessary to satisfy the smoothness conditions as well as the requirement that the curve pass through the data points.

Quadratic B-splines only guarantee continuity of the function and its first derivative, but not the second, so we investigated these next in the hopes that they may still yield a smooth curve but with less overshoot than the cubic splines. The closed form equation for the quadratic B-splines is

$$
\beta^2(x) = \begin{cases} 
\frac{1}{2} \left( x^2 - 3|x| + \frac{9}{4} \right), & \frac{1}{2} \leq |x| < \frac{3}{2} \\
-x^2 + \frac{3}{4}, & |x| < \frac{1}{2} \\
0, & \text{otherwise}
\end{cases}
$$

(3.15)

As we can see in Figure 3.7b, the result has not considerably improved compared to the cubic case; the overshoot still exists, and the quadratic curve looks somewhat less smooth or “natural” because the discontinuities in the second derivative result in abrupt changes in concavity.

The next lowest order is first-order, or linear, B-splines. The closed form equation for these B-splines has been omitted since it can be more efficiently implemented as a simple weighted average of the lattice points on each side of the point being interpolated.

These splines only guarantee continuity of the function values, but not of any derivatives, as we can see in Figure 3.7c. Linear interpolation of images is typically found to be perceptually deficient compared with cubic interpolation, because the absence of smoothness creates a visual banding phenomenon when transitioning between regions of varying slope. However, it also does not exhibit the overshoot problems of the higher-order splines.

### 3.2.3 Conclusions

After considering each of the interpolation methods above, we see that the linear B-spline is the only method that satisfies the conditions we defined at the beginning of this chapter. Therefore, the techniques discussed in the next chapter will use this method to calculate the values of pixels that do not fall precisely on the lattice.
Chapter 4

Designing and Implementing PhETA

4.1 Goals

The primary goal of PhETA is to provide rich and easily comprehensible visualizations of the quality of an image. When we analyze an image, we wish to isolate a very small subset of features of the data, such as a single color channel in a small region of the image without regard to what happens outside that region. For humans, trying to isolate these factors on a purely perceptual basis is difficult, because our visual systems are designed to view a source as the sum of its parts and are not as sensitive to characteristics like small changes in intensity or color. Additionally, our visual systems can tend to be deceptive, resulting in a wide array of optical illusions and other inaccurate interpretations of the data being viewed.

For these reasons, we have developed several alternative visualizations of the source data in order to filter out extraneous information and focus on only those factors that contribute to the region of interest being analyzed. Some of these visualizations are simple extensions of the 2-dimensional image, such as the ability to zoom or inspect the numerical values of a color channel over a small neighborhood of pixels.

Other visualizations transform a 2-dimensional problem into a 1-dimensional one, by investigating only the data along a thin line or swath of pixels. The 1-dimensional plots more thoroughly
reflect aspects of the data such as rates of change of intensity or other color components, which
greatly assists in evaluating not only large shifts, but also smaller shifts (such as those contributed
by noise) that our visual systems may not as easily detect.

Still other visualizations, such as the graph of the modulation transfer function, cast the problem
into other domains that are essential for analysis but are impossible for us to perceive natively.

The details of these various visualization techniques and how they are implemented are discussed
in the appropriate sections below.

4.2 Development Environment

PhETA is written in Java J2SE 5.0, using Eclipse as both the development environment and runtime
framework. Java was chosen in order to make use of the JAI (Java Advanced Imaging) library,
which is a robust and extensible library for image processing, and which contains many efficient
implementations of common image processing algorithms.

Eclipse is an integrated development environment for Java that can also be used as a framework
for standalone applications. Using its RCP (Rich Client Platform) functionality, we can easily
construct an application with a professional look and feel similar to the Eclipse IDE; that is, an
application window with a tabbed editor/document area, toolbars, and user-configurable docking
views.

4.3 Design Philosophies

The Eclipse framework is built upon a small executable that acts as a loader, along with a minimal
set of required “plug-ins” that provide basic functionality. An application built on this framework
is itself implemented as a plug-in that extends the feature set of the Eclipse core. Moreover, a
well-designed application would be divided into multiple plug-ins, each containing an appropriate
related set of functionality.
In this vein, the major functionality of PhETA has been split into the following plug-ins:

- The main application plug-in, containing the major user-interface components (that is, the editor and docking views);
- The core plug-in, containing internal classes and algorithms that are required by other plug-ins, as well as extension points that allow further extensibility of the application;
- The minor UI contributions plug-in, containing various user-interface components used by other parts of the application (reusable widgets and such).

As mentioned above, the core plug-in exposes “extension points” so that the application can be extended. This is the same manner in which the original application extends the Eclipse framework; by providing its own extension points, a developer can allow others to add useful functionality to an application. This ability is used by the edge visualization feature (described later).

### 4.4 User Interface Overview

The main PhETA application window is shown in Figure 4.1. It is divided into several areas:

1. **Menu and Toolbar.** Various tools used to analyze an image are available from here. (See Sections 4.4.1 and 4.4.2.)

2. **Document area.** Images that are loaded for analysis will be displayed in a tabbed pane in this portion of the window. (See Section 4.4.3.)

3. **Quick Info pane.** This pane displays various useful information such as position and color information about a location in the image, and is discussed in Section 4.4.4.

4. **Profile Graph pane.** The graphs displayed in this pane provide the majority of the useful image analysis results, and are discussed in Section 4.4.5.
4.4.1 Available Analysis Tools

Zoom Image Tool

The zoom tool allows users to increase or decrease the magnification of the current view an image, if it is too large to fit on the screen or to see a more detailed view of a subregion of the image.

Zoom levels ranging from 1% to 6400% are available. After selecting the zoom tool, left-clicking on a point in the image will increase the zoom factor, centered around the selected point. Likewise, right-clicking will decrease the zoom factor.

Rectangular Selection Tool

The rectangular selection tool is used by the edge detection and visualization tools (see Section 4.4.2). The edge visualizer by default operates on the entire image; this may be undesirable when dealing with very large images. Using this tool, one can select a rectangular subregion of the image, and the edge visualizer will restrict its calculations to the selected area.
Pixel Inspector Tool

The pixel inspector provides an alternative way to zoom in on a portion of the image being analyzed. Rather than changing the zoom level of the entire view of the image, the pixel inspector displays a localized $9 \times 9$ close-up of an image region. By clicking on a point in the image when the pixel inspector is active, a popup window appears centered on that point that displays the close-up of the surrounding neighborhood.

Furthermore, if a channel overlay (such as red, green, blue, hue, etc.) is chosen from the drop-down list in the toolbar, the numeric values of that color component will be displayed on each of the enlarged pixels.

Profile Line Tool

The profile line tool is used to select a path in the image along which statistics will be collected, and these statistics will be displayed in the Profile Graph view. Typically the user will wish to use this tool to position a line perpendicular to an edge in the image to view the behavior of the edge as it transitions from the pixel values on one side to those on the other side.
After activating the profile line tool, the user should click and hold the mouse at the point where the path will begin, then drag it to the desired endpoint and release the mouse. The path can be further realigned if necessary by positioning the mouse over one of the endpoints until the cursor changes to a four-directional arrow; clicking the endpoint allows it to be dragged elsewhere without disturbing the opposite endpoint.

Across the center of the profile line lies a shorter line that is drawn perpendicular to the path. This line determines the size of the “averaging swath” that will be used in the Profile Graph calculations (the details of these calculations will be described in more detail below). To change the size of the swath, position the mouse over one of the endpoints of this line and drag it away from or toward the actual path until the desired distance is achieved.

Snap to Nearby Edge Tool

The “Snap to Nearby Edge” tool provides access to an optional feature that lets the user make an initial guess of the best orientation for an edge and then automatically determine a more accurate
edge profile via an iterative relaxation process.

First, the user uses the profile line tool to define an initial guess for the edge. Then, invoking the “snap to nearby edge” option will cause PhETA to evaluate spatial derivatives in a local neighborhood surrounding the point at the center of the profile line. From this neighborhood the point with the maximum magnitude is determined and the profile line is recentered at this point, with an orientation determined by computing the angle of the gradient vector at that point.

This action can be repeated as many times as desired to obtain the best positioning of the profile line in a subregion of the image. In most cases, the iteration converges very quickly, in two or three attempts.

4.4.2 Edge Detection and Visualization

Invoking the “Image/Detect Edges...” menu will display the dialog box shown in Figure 4.4. This feature allows the user to obtain a new image that contains visualization of the edges in the currently active image. The resulting image will be akin to the Sobel demonstration shown in Chapter 2; dark pixels indicate areas of little change, while light pixels represent strong edge transitions.
If a subregion has been selected with the rectangular selection tool, then only this region will be processed by the edge detection algorithms. Otherwise, the entire image will be considered.

The list to the left of the dialog displays all of the currently available edge detectors (recall that these edge detectors are implemented as PhETA extensions in external plug-ins). Each edge detector can optionally provide its own user-interface to occupy the tabbed area on the right, in order to allow users to configure the parameters of the edge detector, if there are any. For instance, the basic spatial filtering operators provide options to scale the results (that is, increasing or decreasing the luminance of the result). The Canny detector [3] allows the user to set the required Gaussian smoothing and threshold values.

4.4.3 Document Area

The main document area is a tabbed pane that allows multiple documents to be analyzed simultaneously. By default, each tab occupies the full dimensions of this area, so that only one image is available at a time; however, the tabs can be rearranged at runtime to allow more than one image to be displayed at a time.

4.4.4 Quick Info Pane

The Quick Info pane, shown in Figure 4.5 on the following page, contains position and size information about the various tools available in the editor:

- The dimensions of the currently active image, in pixels;
- The current position of the mouse over the image, in pixels;
- The dimensions of the currently selected region (via the rectangular selection tool), if any;
- The location of the endpoints of the currently selected profile line, its orientation, length, and averaging swath width;
• The color of the pixel underneath the mouse cursor, in RGB space and the modified HSV space.

The profile line and rectangular selection information is updated as those tools are used, allowing the user to make precise measurements while using them.

4.4.5 Profile Graph Pane

The Profile Graph pane (Figure 4.6 on the next page) provides a linear graph of the image data along the profile line that the user has selected. Two graphs are available from the tabs along the bottom of the pane: the “Image Profile” graph, which shows the actual color component data along the line and shows quality characteristics in the spatial domain, and the “Modulation Transfer Function” graph, which shows quality characteristics in the frequency domain.

The user can choose which color channel will be displayed in the graph from the drop-down list at the top of the pane. In addition to separate views for each of the standard RGB channels and modified HSV channels, one can also view the RGB components on a single combined graph; this can be used to detect chromatic aberrations in the image.
If the selected channel is hue, then two other options are available. The scroll bar to the right of the graph allows the user to shift the origin of the hue graph within a range of $-180^\circ$ to $180^\circ$. This can be necessary because hue is represented by an angle between $0^\circ$ to $360^\circ$, so that a change, for instance, of $358^\circ$ at one point to $2^\circ$ at the next is not a change of $356^\circ$, but rather one of $4^\circ$. In order to display this properly on a linear graph, we allow the hue to be rotated in this fashion so that these large discontinuities can be visualized properly.

The other hue-related option causes the lines in the hue graph to be faded with regard to the saturation at those points. This is based on the fact that hue becomes irrelevant as saturation approaches zero, so fading the graph provides a more realistic view of the data by decreasing the visual importance of hue values that may not entirely accurate.

The “Image Profile” graph contains two plots for each channel being displayed. The thin dotted line represents the image data sampled precisely along the profile line set by the user. The thicker solid line shows averages of pixels sampled along a line perpendicular to the profile line, whose width is determined by the width of the “averaging swath” setting of the profile line. The precise dotted line is displayed only for comparative reference; the thick average line is used in calculations, in order to suppress noise and reduce any potential error caused by the use of bilinear interpolation to compute values that do not fall on precise pixel centers.

Clicking the mouse at a position on the graph view will flash a red circle on the active image

---

*Figure 4.6: The Profile Graph pane, showing the spatial-domain Image Profile plot.*
that represents the corresponding location along the profile line in that image. This allows the user to determine precisely where in an image a particular feature of the data, such as overshoot or an edge transition, occurs based on the presentation in the profile graph.

PhETA automatically determines which edge to analyze by calculating the maximum gradient magnitude across the entire profile line, and marks this edge in the graph with an arrow. In the event that the profile line spans more than one edge, the edge that is automatically detected may not be the one desired for analysis. In this case, the profile line needs to be adjusted such that it spans only one strong edge.

**Image Metrics**

Once the desired edge is pinpointed, PhETA will calculate several statistics about the nature of the edge; those in the spatial domain are shown in Figure 4.7. First, the extent or width of the edge is computed by starting at the center of the edge and branching outwards until a peak is found (that is, the data changes direction).

The amount of overshoot and undershoot on each side of the edge is computed from these peaks, if any, and it is represented as a percentage of the average amount of change between both sides of the edge transition. The 10% and 90% points represent the locations along the $x$-axis at which the edge is valued at 10% of its total rise and 90% of its total rise along the ramp, respectively. These values are used to compute the rise time of the edge, which is the absolute value of the difference
between the 10% and 90% points. As discussed in the mathematical foundations in Chapter 2, the slope of the ramp is inversely proportional to the degree of smoothing, or blurring, of the edge. Thus, the shorter the rise time of an edge, the sharper or clearer it will appear, resulting in a perceptually higher quality image.

Once the rise time has been calculated, PhETA divides the pixel height of the image by this value to obtain the number of rises per image height. This value is primarily intended to facilitate the comparison of images from multiple cameras with differing image dimensions. A camera with a very low resolution may produce images with lower rise times in pixels than those from a camera with a very high resolution, so that this measure does not necessarily reflect the perceived quality of the edges. For this reason, one should compare the rise times per image height rather than the absolute rise times.

The “Modulation Transfer Function” graph (Figure 4.8) displays a plot of the modulation transfer function (see Section 2.7) calculated from the same sampled averages used to form the primary plot in the “Image Profile” graph. The data for this plot is computed as follows:

1. A data sequence is extracted that represents the averaged swaths along the profile line. To ensure a smoother curve, the image is supersampled at every 1/16 of a pixel, using the same bilinear interpolation as before.
2. A first derivative approximation of this data is obtained by using forward differences.

3. This first derivative data is filtered by multiplying it by a trapezoidal-shaped filter, which is valued at 1 in the neighborhood of the edge and falls off to 0 on each side. This has the effect of suppressing noise that is present away from the edge without altering the edge itself.

4. The 1-dimensional Discrete Fourier Transform of this data is computed, containing both real and imaginary components.

5. The magnitude of the result of the DFT is taken, resulting in the MTF. The data is normalized so that the initial value (the zero frequency) evaluates to 1.

In addition to the MTF curve itself, the MTF50, measured in line pairs (cycles) per pixel, is also marked on the graph. Recall that higher MTF50 values typically correspond to higher quality edges, but it should be noted that very high MTF50 values may indicate a degree of oversharpening or aliasing in the image.

Since we are working with discrete data, we may not be able to calculate the MTF50 precisely. Instead, PhETA highlights the smallest interval in which the 50% drop occurs and uses the midpoint of this interval as an approximation of the actual MTF50.

Once the MTF50 has been determined, it is trivial to compute the number of line pairs per image height by multiplying the MTF50 by the height of the image in pixels, and PhETA also displays this value for the user. As was discussed in regard to the rise time above, the line pairs per image height measure is more useful for evaluating images from multiple cameras whose absolute MTF50 values are not directly comparable.
Chapter 5

Results

The following pages present results produced by PhETA for a set of test images. The first three examples are photographs of a standard IEEE-approved black-and-white test pattern from Edmund Optics, taken with the Canon PowerShot A520 at a resolution of $2272 \times 1704$ pixels. One horizontal, one vertical, and one slanted edge are each analyzed; the spatial and frequency domain statistics are given, as well as the image profile and MTF graphs of the luminance channel.

The next three examples are images of the same test pattern, taken with the Polaroid PhotoMAX PDC 2300Z camera at a resolution of $1792 \times 1200$ pixels. The same calculations and graphs as above are presented.

A further image of the test pattern is presented for the PDC 2300Z, displaying chromatic aberrations that arise as the width and spacing of the bars decreases. To emphasize the chromatic nature of these results, a combined RGB profile graph and a hue profile graph are shown in addition to the luminance profile graph.

Finally, two examples of realistic scenes are given from the Canon PowerShot A520. The first of these shows the analysis of a colored edge; the second, a look at discolorations caused by sensor blooming.
Edmund Optics Test Pattern, Vertical Edge, Canon PowerShot A520

Edge Direction: Falling
Overshoot: 5.25%
Undershoot: 11.31%
10% point: 38.60
90% point: 36.70
Rise time: 1.90 pixels
(MTF50: ≈ 0.291 pairs/px)
(896 rises/height)
(495 pairs/height)

Image Profile Graph, Luminance

Modulation Transfer Function Graph, Luminance
Edmund Optics Test Pattern, Horizontal Edge, Canon PowerShot A520

Edge Direction: Falling
Overshoot: 3.70%
Undershoot: 2.64%
10% point: 37.70
90% point: 36.10
Rise time: 1.60 pixels
(1065 rises/height)
MTF50: ≈ 0.277 pairs/px
(472 pairs/height)
Edmund Optics Test Pattern, Slanted Edge, Canon PowerShot A520

- **Edge Direction:** Falling
- **Overshoot:** 6.05%
- **Undershoot:** 6.90%
- **10% point:** 34.30
- **90% point:** 32.40
- **Rise time:** 1.90 pixels (896 rises/height)
- **MTF50:** $\approx 0.265$ pairs/px (451 pairs/height)

### Image Profile Graph, Luminance

<table>
<thead>
<tr>
<th>Distance across profile (pixels)</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>15</td>
<td>50</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>45</td>
<td>100</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>55</td>
<td>100</td>
</tr>
<tr>
<td>60</td>
<td>100</td>
</tr>
<tr>
<td>65</td>
<td>100</td>
</tr>
</tbody>
</table>

### Modulation Transfer Function Graph, Luminance

<table>
<thead>
<tr>
<th>Cycles (line pairs) per pixel</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1.00</td>
</tr>
<tr>
<td>0.1</td>
<td>0.90</td>
</tr>
<tr>
<td>0.2</td>
<td>0.80</td>
</tr>
<tr>
<td>0.3</td>
<td>0.70</td>
</tr>
<tr>
<td>0.4</td>
<td>0.60</td>
</tr>
<tr>
<td>0.5</td>
<td>0.50</td>
</tr>
<tr>
<td>0.6</td>
<td>0.40</td>
</tr>
<tr>
<td>0.7</td>
<td>0.30</td>
</tr>
<tr>
<td>0.8</td>
<td>0.20</td>
</tr>
<tr>
<td>0.9</td>
<td>0.10</td>
</tr>
<tr>
<td>1.0</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Edmund Optics Test Pattern, Vertical Edge, Polaroid PDC 2300Z

- Edge Direction: Falling
- Overshoot: 6.61%
- Undershoot: 11.76%
- 10% point: 41.40
- 90% point: 39.40
- Rise time: 2.00 pixels (600 rises/height)
- MTF50: \( \approx 0.281 \) pairs/px (337 pairs/height)

**Image Profile Graph, Luminance**

**Modulation Transfer Function Graph, Luminance**
Edmund Optics Test Pattern, Horizontal Edge, Polaroid PDC 2300Z

Edge Direction: Falling
Overshoot: 8.93%
Undershoot: None
10% point: 43.30
90% point: 39.50
Rise time: 3.80 pixels
(315 rises/height)
MTF50: \(\approx 0.206\) pairs/px
(247 pairs/height)

Image Profile Graph, Luminance

Response vs Distance across profile (pixels)

Modulation Transfer Function Graph, Luminance

Response vs Cycles (line pairs) per pixel
Edmund Optics Test Pattern, Slanted Edge, Polaroid PDC 2300Z

Edge Direction: Falling
Overshoot: None
Undershoot: 8.76%
10% point: 21.50
90% point: 19.70
Rise time: 1.80 pixels
(666 rises/height)
MTF50: ≈ 0.256 pairs/px
(307 pairs/height)

Image Profile Graph, Luminance

Modulation Transfer Function Graph, Luminance
Edmund Optics Test Pattern, Chromatic Aberrations in Polaroid PDC 2300Z

Image Profile Graph, Luminance

Image Profile Graph, Combined RGB

Image Profile Graph, Hue (no fading)
Colored Edge from a Naturally Lit Outdoor Scene, Canon PowerShot A520

Edge Direction: Falling
Overshoot: 0.43%
Undershoot: None
10% point: 35.30
90% point: 32.80
Rise time: 2.50 pixels (681 rises/height)

MTF50: ≈ 0.181 pairs/px (308 pairs/height)

Image Profile Graph, Luminance

Modulation Transfer Function Graph, Luminance
Sensor Blooming in a Naturally Lit Outdoor Scene, Canon PowerShot A520

Image Profile Graph, Combined RGB

Image Profile Graph, Hue (faded with respect to saturation)
Bibliography


Vita

Anthony Allowatt began attending Virginia Polytechnic Institute and State University in 1999, where he graduated cum laude in 2003 with a Bachelor of Science in Computer Science and a minor in Mathematics. He continued at Virginia Tech through 2005 to earn a Master of Science in Computer Science.