Towards Real-time Hardware Gamma Correction for Dynamic Contrast Enhancement

Jesse Scott, Ph.D. Candidate
Integrated Design Services, College of Engineering,
Pennsylvania State University
University Park, PA
jus121@engr.psu.edu

Michael Pusateri, Ph. D. and Director
Integrated Design Services, College of Engineering,
Pennsylvania State University
University Park, PA
mpusateri@engr.psu.edu

Abstract – **Making the transition between digital video imagery acquired by a focal plane array and imagery useful to a human operator is not a simple process. The focal plane array "sees" the world in a fundamentally different way than the human eye. Gamma correction has been historically used to help bridge the gap. The gamma correction process is a non-linear mapping of intensity from input to output where the parameter gamma can be adjusted to improve the imagery's visual appeal. In analog video systems, gamma correction is performed with analog circuitry and is adjusted manually. With a digital video stream, gamma correction can be provided using mathematical operations in a digital circuit. In addition to manual control, gamma correction can also be automatically adjusted to compensate for changes in the scene.**

We are interested in applying automatic gamma correction in systems such as night vision goggles where both low latency and power efficiency are important design parameters. We present our results in developing an automatic gamma correction algorithm to meet these requirements. The algorithm is comprised of two parts, determination of the desired value for gamma and the application of the correction. The calculation of the gamma value update is performed based upon statistical metrics of the imagery's intensity. HDL code implementing the measurement of the statistical metrics has been developed and tested in hardware. Both the computation of a gamma update and the application of the gamma correction were simplified to basic arithmetic operations and two specialized functions, logarithm and exponentiation of a constant base by a variable exponent.

We present approximation methods for both specialized functions simplifying their implementation into basic arithmetic operations. The hardware implementations of the approximations allow the above requirements to be met. We evaluate the accuracy of the approximations as compared to full resolution double-precision floating point mathematical operations. We present the final results for visual judging to evaluate the impact of the approximations.

I. Background

A. Problem Statement

In addition to its other uses, gamma correction is an effective tool for manipulating the histogram of an image that is either over or under exposed, but not fully compromised with saturation. While it is available as a tool in most image processing software, the functions used to implement gamma correction have relatively complex implementations in hardware. We present an implementation that approximates gamma correction with a satisfactory level of visual quality, but with a tractable hardware implementation. Our design is meant to be capable of providing real-time, pixel-serial gamma correction to imagery generated by a focal plane array with an active area of 1600x1200 pixels and a pixel clock rate in excess of 150 MHz utilizing commercially available FPGA hardware.

B. Gamma Correction

Gamma correction is an intensity transformation that takes the form of a generalized power law with equivalent range and domain. Letting $x \in [0,1]$ represent the intensity domain and y represent the intensity range, a gamma correction transform is described by:

$$y = x^{\gamma} \tag{1}$$

where $\gamma > 0$. For gamma $\gamma < 1$, gamma compression occurs, moving the intensity histogram to the right. For gamma $\gamma > 1$, gamma expansion occurs, moving the histogram to the left. Figure 1 shows a normal probability distribution with both a gamma compressed and gamma expanded version of the distribution.

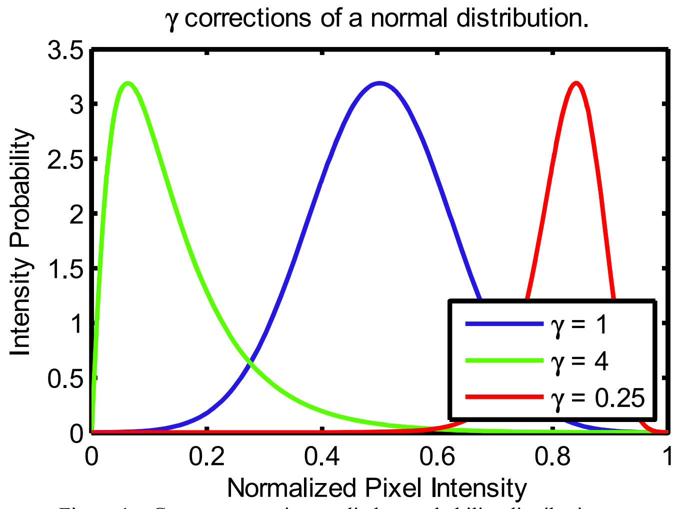

Figure 1 – Gamma correction applied to probability distributions

Figure 2 shows the visual impact of changing gamma by presenting the same image with three different values of gamma [3]. We see the general progression of overall lightening of the image as gamma increases. In this sequence, the most pronounced change can be observed between the image with γ=1 and the image with γ=2. The whole underwater ridge is ill defined and shadowy with γ=1, however, with γ=2, most of its features become sharply defined. A small prominence at the left end of the ridge is not visible with γ=1 while its left edge becomes well defined with γ=2. Note that both the left and right edges of the prominence are well defined with γ=4, but, this is at the expense of the bulk of the image looking overexposed.

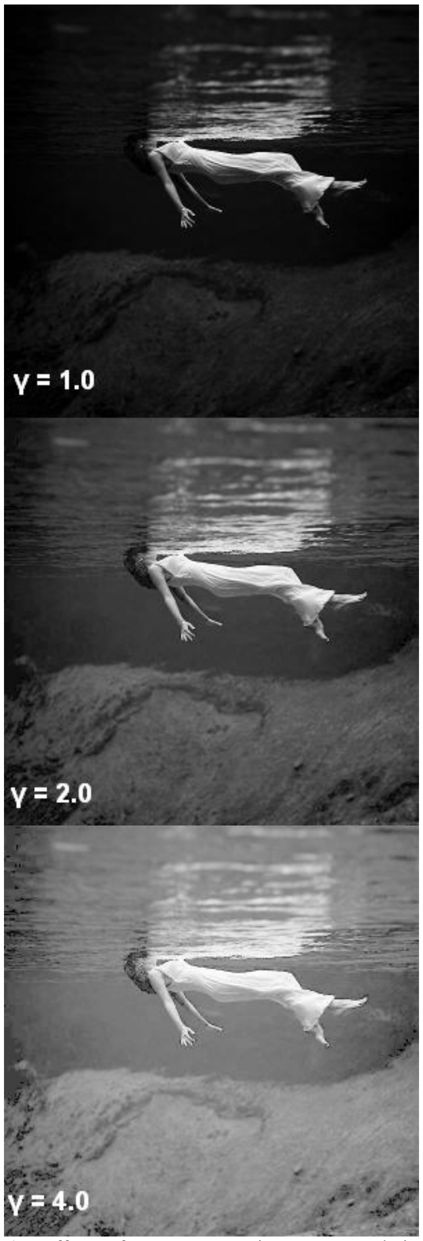

Figure 2 – Effects of gamma correction on grayscale imagery

II. Design

A. Modifications for Hardware Implementation

For real-time applications with actual hardware, it is useful to consider a slightly modified version of the gamma correction transformation. Letting $x \in [0, \Delta x]$ represent the intensity domain and $y \in [0, \Delta y]$ represent the intensity range, an equivalent form of the gamma correction transform is described by:

$$y = \Delta y \left(\frac{x}{\Delta x}\right)^{\gamma} \tag{2}$$

The introduction of the terms Δx and Δy allow us to also use the transformation to adjust the gray scale domain of the imagery from that of the focal plane array to the range of the display.

In order to automate the process of gamma correction, we need to define a metric, derived from the imagery, allowing the automatic determination of gamma for an image. We have chosen to utilize imagery statistics collected on each frame although they are not the only possible choice. When utilizing imagery statistics, we must introduce the real-time compromise that we either introduce a frame of latency to allow application of the actual imagery statistics or we utilize statistics from the previous frame to process the current frame. Our implementation uses the latter choice; however, it does not impact the mathematical development of the module.

Our imagery metric is used to determine a set point for the domain, denoted x_s, that is mapped to a pre-determined set point for the range, y_s. The set point pair (x_s, y_s) can then be used to compute an appropriate value of gamma for the frame as:

$$\gamma = \frac{\log\left(\frac{y_s}{\Delta y}\right)}{\log\left(\frac{x_s}{\Delta x}\right)} = \frac{\log(y_s) - \log(\Delta y)}{\log(x_s) - \log(\Delta x)}. \tag{3}$$

The computation of gamma needs to occur only once per frame.

In examining how to actually implement the gamma correction transform given in (2), it became clear that implementing a function that computed the result of a variable base raised to a variable exponent was not feasible within our hardware limitations. To overcome this problem, we found it useful to rewrite (2) in the form:

$$y = \Delta y 2^{\gamma(\log_2(x_s) - \log_2(\Delta x))}. \tag{4}$$

Given this representation, the implementation problem is simplified to computing a variable exponentiation of base two and computation of log base two, both with variable argument. While different bases would provide identical results, we selected base two due to hardware considerations. Both of these calculations have tractable hardware implementations.

B. Hardware Computation of $\boldsymbol{log_2(s)}$

For our problem, the general computation of $\log_2(s)$ is done over a limited, but potentially large range:

$$s \in [1, 2^{p+1}) \tag{5}$$

where p is an integer. We handle $s = 0$ as a special case. Imaging systems typically require $10 \geq p \geq 16$; that is, we need to find the logarithm over 10 to 16 octaves. We can express the logarithm argument as:

$$s = q2^p \qquad (6)$$

where q is a positive real number and where we can find:

$$p = \text{floor}(\log_2(s)) \qquad (7)$$

and

$$q = \frac{s}{2^p}, q \in [1, 2) \qquad (8)$$

In hardware, finding p can be easily accomplished by finding the largest nonzero bit in s. Finding q is accomplished by simple right shifting of s. We can now find:

$$\log_2(s) = p + \log_2(q) \qquad (9)$$

where the $\log_2(q)$ term can be approximated on its small range using a function with an acceptable hardware implementation.

C. Hardware Approximation of $\boldsymbol{\log_2(q)}$

There are many well known ways to approximate a logarithm over a closed interval. After evaluation of several methods, we chose direct fitting of a fixed order polynomial to the function over the desired interval. This provided a significant improvement in accuracy over a comparable order Taylor series approximation. It was also as accurate as a comparable order Padé approximation [1], but does not require a high-precision arbitrary argument division of the Padé. Arbitrary argument division was avoided because of its significant hardware requirements.

Based on a propagation of error analysis, we chose to implement the approximation to obtain precision at eight places to the right of the binary point. With this requirement, we found that a directly fitted third-order polynomial with 16-bit signed coefficients was sufficient. Using MATLAB™ to generate an initial direct fit of the logarithm with floating point coefficients, we then truncated to scaled integer coefficients using a simple iterative search to find the minimum error [2]. The approximation is given as:

$$\log_2(q) \approx \Big(\big((1261q - 8435)q + 24666\big)q - 17481\Big)2^{-13} \qquad (10)$$

where the equation is staged for three multiply accumulate (MAC) operations and the coefficients are shown as their decimal values. The absolute error of the approximation is shown in Figure 3; it meets our precision requirements within an acceptable margin. Because the $\log_2(q)$ approximation is additive in (9), the approximation error in $\log_2(s)$ repeats every decade of p.

D. Hardware Computation of $\mathbf{2^s}$

For our problem, the general computation of 2^s is done over a limited but potentially large range:

$$s \in [p, 0) \qquad (11)$$

where p is a negative integer. Our goggle systems typically produce $-16 \geq p$. We can express the exponent as:

$$s = p + q \qquad (12)$$

where q is a positive real number and where we can find:

$$p = \text{floor}(\log_2(s)) \qquad (13)$$

and

$$q = s - p, q \in [0, 1) \qquad (14)$$

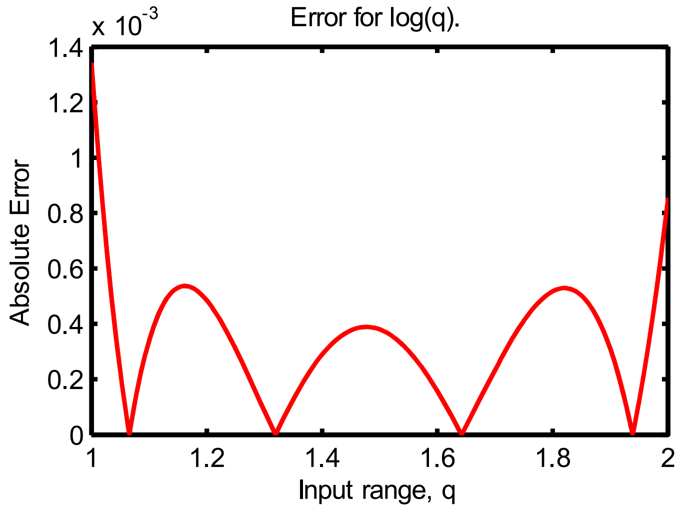

Figure 3 – Error of the polynomial approximation of $\log_2(q)$

In hardware, finding p can be easily accomplished by finding the position of the largest nonzero bit in s with respect to the binary point. Likewise, finding q is accomplished by subtraction. We can now find:

$$2^s = 2^p \cdot 2^q \qquad (15)$$

The operation of finding 2^q on its small range is performed using an approximation function with an acceptable hardware implementation. The multiplication of the result by 2^p is then handled by simple bit shifting.

E. Hardware Approximation of $\mathbf{2^q}$

After evaluation of several of the well known methods of approximating an exponentiation with constant base and variable exponential, we again chose direct fitting of a fixed order polynomial to the function over the desired interval. Again, the accuracy improvement over a comparable order Taylor series approximation was significant.

Based on a propagation of error analysis, we chose to implement the approximation to obtain precision at eleven places to the right of the binary point. With this requirement, we found that a direct fitted third-order polynomial with 16-bit unsigned coefficients was sufficient. Using MATLAB™ to generate an initial direct fit of the logarithm with floating point coefficients, we then truncated to scaled integer coefficients using a simple iterative search to find the minimum error [2]. The approximation is given as:

$$2^q \approx \left(\left((5179q + 14689)q + 45668\right)q + 65524\right)2^{-16} \quad (16)$$

where the equation is staged for three MAC operations and the coefficients are shown as their decimal values. The absolute error of the approximation is shown in Figure 4; it meets our precision requirements with some margin. Because the 2^q approximation is multiplicative in (15), the approximation error in 2^s changes with the magnitude of p and is presented in Figure 5.

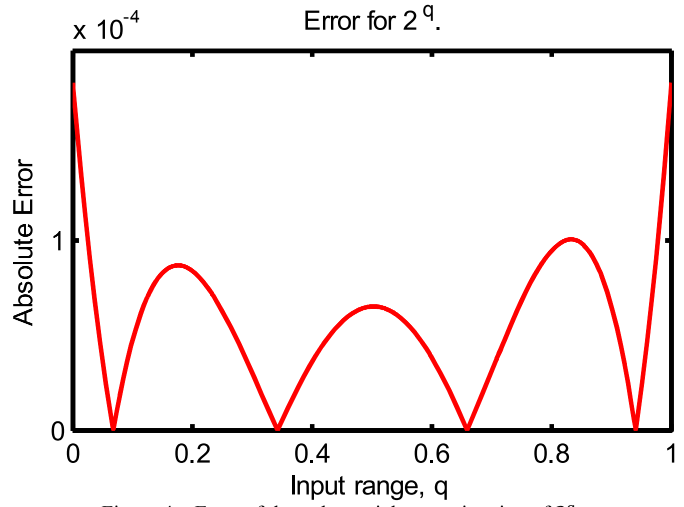

Figure 4 – Error of the polynomial approximation of 2^q

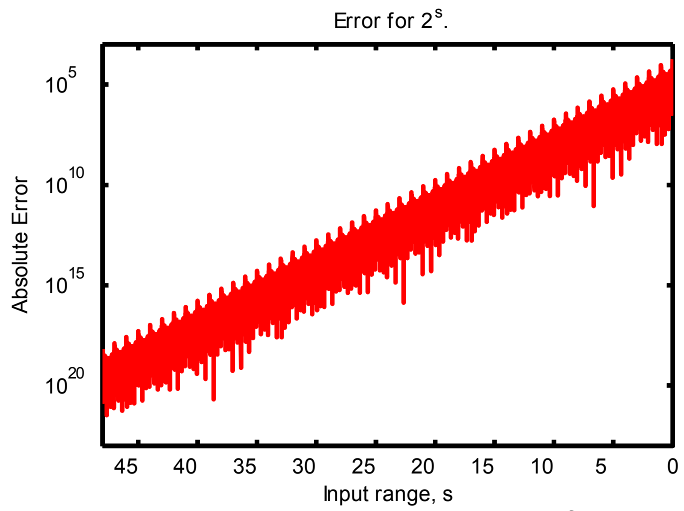

Figure 5 – Error of the polynomial approximation of 2^s

III. Results

We tested a MATLAB implementation of our approximate gamma correction transform versus a full floating point implementation using a thermal video sequence. The video was captured using a Video IR™ long wave infrared camera, sensitive between 8 and 14 µm. The camera utilizes an active array area of 640x480 and an intensity depth of 14-bits. Figure 6 below shows a typical source frame, top left, a linear correction, top right, the floating point exact gamma correction, bottom left, and the approximate gamma correction, bottom right. The gamma correction shown here represents an extreme for the error conditions of the approximation; it was not chosen for visual improvement.

Comparison of the approximation to the floating point result for the duration of the video shows that the vast majority of the pixel intensities matched within 1-bit of round-off. However, we also found anomalous intensities with tens of bits difference between exact and approximation. We are investigating the source of these errors as they are not consistent with our error prediction.

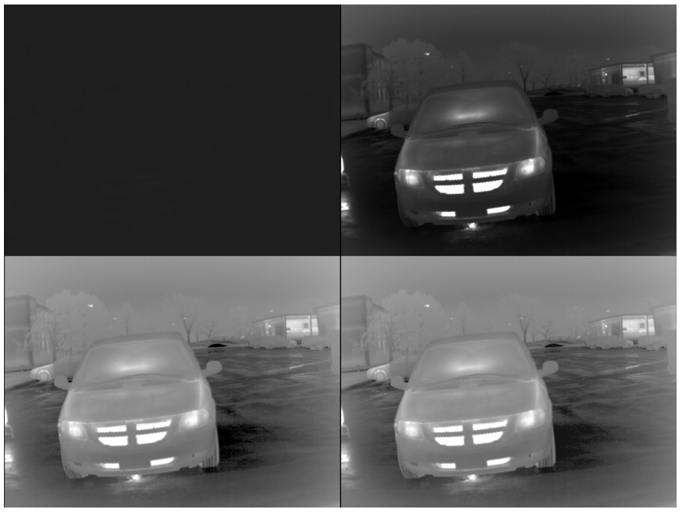

Figure 6 – Frame from video sequence

IV. Analysis

Hardware Implementation Resources and Latency

Because our gamma correction is intended to support an actual system under development, our solution took several requirements into consideration:

1. Execute as a pixel-serial operation
2. Latency on the order of tens of pixel clock cycles
3. Simple implementation for FPGAs, keeping mathematical complexity to MAC and less
4. Keep power and resource requirements to a minimum
5. Work at pixel clock rates in excess of 165 MHz

We felt that meeting the second and third requirements would be the key to meeting requirements four and five. At this phase of the project, the focus is primarily on the first three requirements.

Tables 1 and 2 present the expected major hardware requirements for the implementation of $\log_2(s)$ and 2^s and provide their total latency. Tables 3 and 4 present similar information for the overall modules used to compute the per frame gamma and the per pixel serial gamma correction. They both include appropriate counts for each instantiation of $\log_2(s)$ and 2^s. Overall, the expected latency of all modules is satisfactory. Likewise, the hardware requirements represent a reasonable portion of the overall available resources.

Table 1: Hardware requirements for $\log_2(s)$

Hardware	Count
shifter	1
adder	2
multiplier	0
MAC	3
divider	0
Latency in Pixel Clock cycles	21

Table 2: Hardware requirements for2^s

Hardware	Count
shifter	1
adder	1
multiplier	1
MAC	3
divider	0
Latency in Pixel Clock cycles	20

Table 3: Hardware requirements gamma computation

Hardware	Count
shifter	4
adder	10
multiplier	0
MAC	12
divider	1
Latency in Pixel Clock cycles	30

Table 4: Hardware requirements for gamma correction

Hardware	Count
shifter	3
adder	5
multiplier	3
MAC	9
divider	0
Latency in Pixel Clock cycles	50

References

[1] M. Vajta, “Some Remarks on Padé-Approximations”, in Proc. of 3rd TEMPUS INTCOM Symposium on Intelligent Systems in Control and Measurement (edited by J.Vass and D.Fodor), pp.53-58, Sept. 2000.

[2] F. B. Hildebrand, Introduction to Numerical Analysis, 2nd Ed, Dover Publications: June 1987.

[3] "Gamma Correction." *Wikipedia, The Free Encyclopedia*. Wikimedia Foundation, Inc. 22 July 2004. Web. Aug. 2009.