Development of an Efficient Super-Resolution Image Reconstruction Algorithm for Implementation on a Hardware Platform

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Engineering

By

THOMAS C. PESTAK
B.S. Electrical Engineering, Wright State University, 2007

2010
Wright State University

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

May 28, 2010

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Thomas C. Pestak ENTITLED Development of an Efficient Super-Resolution Image Reconstruction Algorithm for Implementation on a Hardware Platform BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Engineering

John Emmert, Ph.D.
Thesis Director

Kefu Xue, Ph.D.
Department Chair

Committee on Final Examination

John Emmert, Ph.D.

Ray Siferd, Ph.D.

Saiyu Ren, Ph.D.

John A. Bantle, Ph.D.
Vice President for Research and
Graduate Studies and Interim Dean
of Graduate Studies

ABSTRACT

Pestak, Thomas C. M.S. Egr., Department of Electrical Engineering, Wright State University, 2010. Development of an Efficient Super-Resolution Image Reconstruction Algorithm for Implementation on a Hardware Platform

There is a growing demand from numerous commercial and military applications for images with ever-improving spatial resolution. However, there are resolution-limiting factors inherent in all imaging systems. Decreasing pixel sizes and/or increasing sensor arrays are not always viable. Super-Resolution (SR) Image Reconstruction is an image processing technique that restores a high-resolution (HR) image from a series of low-resolution (LR) images of a particular scene. Recently, there has been extensive research on robust SR algorithms used for post-processing. The goal of this thesis is to explore the current SR research and design computationally efficient SR algorithms for real-time processing based on a non-uniform interpolation approach.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to first and foremost extend my deepest gratitude to my teacher, advisor, and friend, Dr. Marty Emmert, for his tremendous guidance and support through the years. I would like to thank the other members of my thesis committee, Dr. Raymond Siferd, and Dr. Saiyu Ren. I am indebted to the NEWSTARs program for funding this research, and enabling me to complete my degree. I would like to thank my dear friends in the EE Department -Vicky Slone, Marie Donahue, and Barry Woods - for their remarkable professionalism, dedication, and kindness. I would also like to thank my teacher, friend, mentor, and workout partner, Dr. Fred Garber, for sharing so much of his time, wisdom, faith, and humor with me. I would especially like to thank Andrew and Nisha Kondrath for their loyalty, friendship, and space. I thank my parents, Christopher and Brenda Pestak, for lighting within me a passion for the future. Finally I would like to thank my beautiful fiancée, Melanie Platfoot, for her uncompromising love. This thesis would not have been possible without these people. I would like to dedicate this work to the late Father Christian (Chris) Rohmiller who, in a short time, blessed my life forever.

SUMMARY

The goal of this thesis is to attain an understanding of Super-Resolution (SR) image reconstruction and use that knowledge to design SR algorithms efficient enough for near real time implementation. This thesis is divided into four sections. Section 1 presents a review of the theory, development, applications, and implementations of SR image reconstruction. Section 2 employs the models and theorems of the first section to design an efficient SR algorithm. Section 2 also provides an in-depth look into image registration, the relationship between sampling grids, and an efficient approach to non-uniform interpolation. Section 3 submits five different benchmarks for measuring the quality of the output images of the algorithm. These are: A sum squared error analysis, a measure of average pixel error, a count of correct pixels, a structural similarity measurement index, and a visual analysis aided by generating a scaled-pixel-error image. Using these benchmarks, three separate sets of undersampled images (low, medium, and high detail) were resolved to higher resolutions. The results were compared with the output of MATLAB's cubic interpolation function to demonstrate the effectiveness of the Super-Resolution algorithm. The optimization and performance of the algorithm are presented as they pertain to hardware implementation. Section 4 reviews the effectiveness of the SR image reconstruction (based on the results of section 3), maintains the desirability of the design approach over matrix-inversion based methods, and proposes future research to enhance the capabilities of the algorithm. Lastly, there is an appendix with some coding examples, followed by a list of references used in this research.

SECTION 1
BACKGROUND

1.1 Definition

Super-Resolution (SR) image reconstruction refers to image processing techniques designed for increasing (or restoring) the spatial resolution of images – in most cases by fusing a series of undersampled , low-resolution (LR) images to generate one high-resolution (HR) image. Definitions given in SR research vary although they generally describe:

1.) The mathematical model for SR image reconstruction

2.) The goals of SR image reconstruction

Elad, Takeda, and Milanfar define SR image reconstruction using elements of the model which gives: "…an inverse problem that combines denoising, deblurring, and scaling-up tasks, aiming to recover a high quality signal from degraded versions of it." [6] A goal-oriented definition, given by Chadhuri reads: "…super-resolution is largely known as a technique whereby multi-frame motion is used to overcome the inherent resolution limitations of a low-resolution camera system." [4]

1.2 Early Work

Thomas S. Huang and R.Y. Tsai were the first to propose a method for SR image reconstruction [8]. They proved that a HR image could be restored given a sequence of LR images of the same scene. They referred to their method as Multiframe Image Restoration and

Registration. The defined goal was "restoring a high-resolution image from a sequence of low-resolution, undersampled, discrete frames of a moving object." The motivation for their work was the images taken from a LandSat satellite. These images were slightly different from each other following each orbit of the satellite. The problem consisted of two parts – registration, or the estimation of the relative shifts between the slightly-differing images, and restoration, or the interpolation on a higher resolution grid. The flow diagram developed by Huang and Tsai is given in Figure 1 [8].

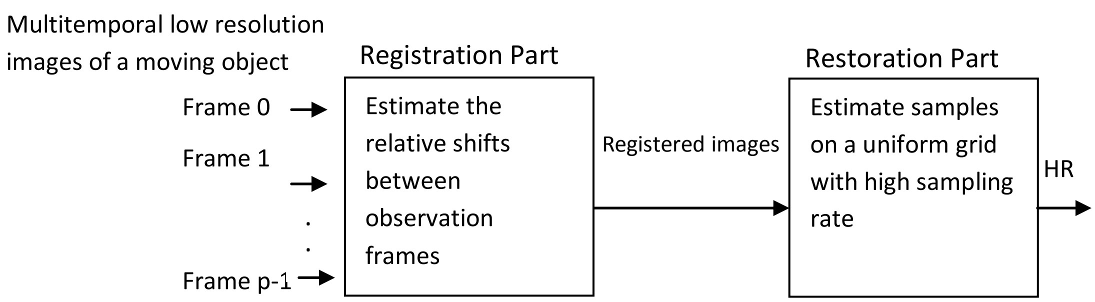


Figure 1 – Original flow diagram

Huang and Tsai exploited the aliasing relationship between the discrete Fourier Transform (DFT) and the continuous Fourier Transform (CFT) to estimate the relative shifts between frames, or samples, of an ideal image signal.

S.P. Kim, N. K. Bose, and H. M. Valenzuela broadened this technique to include removal of the noise and blur present in the LR samples. They expanded the flow diagram (shown in Figure 2) by including a filtering stage before reconstruction [11].

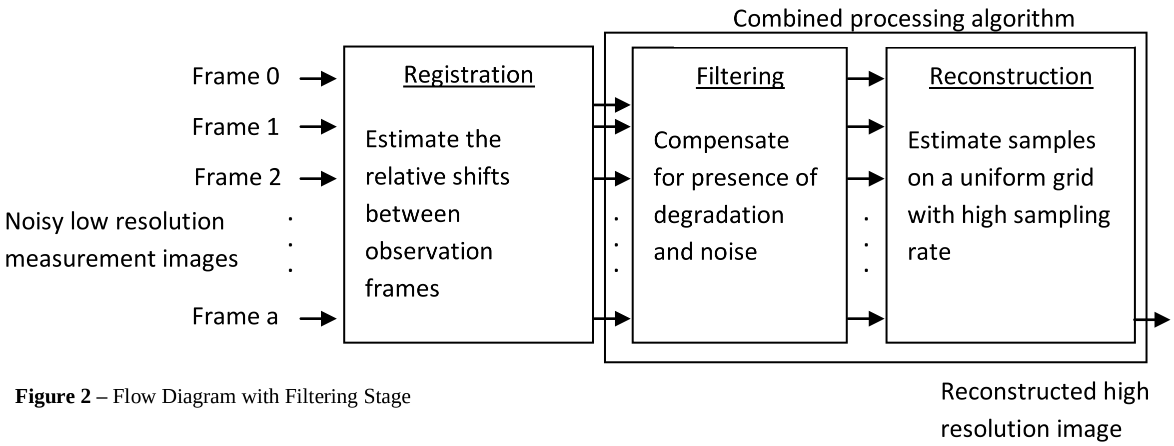


Figure 2 – Flow Diagram with Filtering Stage

This flow diagram remains the basis for most SR algorithms, with differences appearing only in terminology. The main difference is the term ‘reconstruction’ is generally given as ‘Interpolation’ and the removal of noise and blurring is given as ‘restoration’. Also, the interpolation and restoration stages can be interchanged. Most SR algorithms follow this R-I-R, or R-R-I sequence. Figure 3 is an example.

Figure 3 – Common SR Flow Diagram

1.3 Observation Model

SR image reconstruction is modeled as an inverse problem. That is, the goal of SR is to reverse the effects of undersampling, blurring, and warping that relate the LR images to a desired HR image. Figure 4 is an observation model that includes these effects [12].

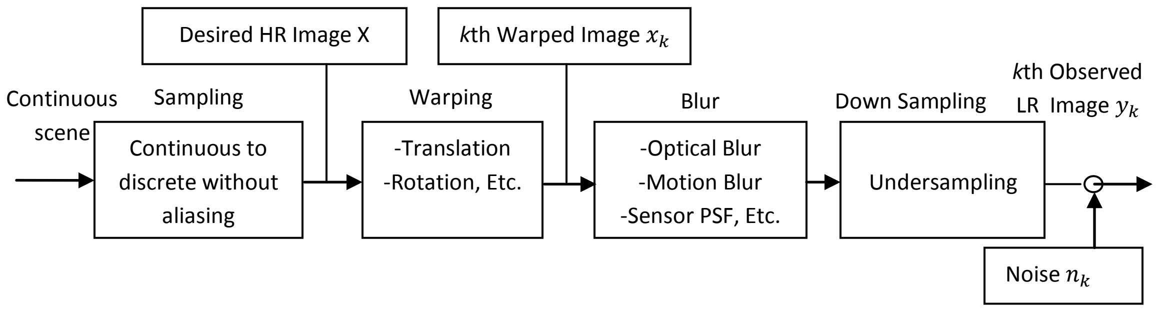


Figure 4 – Observation Model [12]

1.4 Mathematical Model

Mathematically, this is given as [12]:

$$\boldsymbol{y}_k = DB_kM_k\boldsymbol{x} + n_k \ \text{ for } 1 \leq k \leq p$$

Where y_k represents p LR images

D is the subsampling matrix which downsamples a HR image of dimensions $L_1N_1 \times L_2N_2$ into an aliased LR image of dimensions $N_1 \times N_2$

B_k is the blur matrix which could also be represented as simply B if the only blurring is from the sensor which remains unchanged between image acquisitions– i.e. the point spread function (PSF) of the imaging system.

M_k is a warping matrix that describes the motion that occurs during acquisition of the LR images

x is the desired HR image

n_k is additional noise

1.5 Motivation

Trade-offs exist between the physical size, economic cost, quality, and subsequent spatial resolution of imaging systems. For the case of optical digital cameras using either Charge-Coupled Devices (CCD) or Complimentary Metal-Oxide Semiconductor (CMOS) sensors, the manufacturing techniques for greater spatial resolution (more pixels per image) requires either:

1.) Reduction in sensor size
2.) Increase in chip size

Reducing sensor size (pixels) has limits. Decreasing optical pixels naturally decreases the number of captured photons per exposure, which *increases* noise. This happens because light sources generally emit photons randomly (approaching Gaussian Distribution), and the uncertainty (standard deviation) associated with this randomness is directly proportional to the width of a photon-capturing pixel. This phenomenon is known as photon shot noise. Therefore, doubling the width of a pixel effectively doubles the signal to noise ratio (SNR) and inversely, shrinking the width decreases the SNR [3].

Increasing chip size allows more pixels per frame without decreasing the physical size of a pixel. Modern digital SLR cameras are designed with sensor sizes greater than compact digital cameras, leading to larger overall chip sizes. However, increased chip sizes generate increased capacitance which slows the rate of charge transfer. In addition, large chips sizes cost more and are not feasible for many applications with size and weight limitations.

In infrared imaging systems, the ratio of the size of the sensor array to the detection area of individual sensors (otherwise known as Fill Factor) is small (compared to optical systems) because of the need to properly isolate sensors [7]. Because of this requirement, the number of sensors is limited, the Nyquist sampling rate is not met, and aliasing occurs. SR image reconstruction, therefore, is a viable technique for taking the undersampled LR images and using them to reconstruct a HR image.

There are physical limits and cost trade-offs to image acquisition. Because of this, image processing algorithms represent opportunities for increased spatial resolution when manufacturing techniques are not feasible. With exhaustive research on single image interpolation, which is inherently limited in its ability to recover higher frequency components, SR image reconstruction is a promising and relatively new development in image processing.

1.6 The Sub-Pixel Requirement

Single image interpolation (which acts as a low-pass filter) cannot recover the higher frequencies lost or aliased during acquisition (sampling). Because of this, SR image reconstruction is inherently different from single image interpolation. SR image reconstruction, therefore, requires that the LR images contain some independent information describing the scene from which they were sampled. An integer shift in pixels contains only redundant information which renders SR image reconstruction impossible. What is required for successful reconstruction is a fractional or "sub-pixel" shift in data across the sensor array between samples

[10].   If such sub-pixel shifting occurs between aliased LR images, there is new information that can be used to restore a HR image.

1.7 Sub-Pixel Shifting

Understanding sub-pixel shifts reveals an intuitive look at the registration stage of SR image reconstruction. In the images below, the black, curved line is a continuous scene sampled 4 times. The dotted, curved line represents the position of the black line in the reference frame. Despite the fact that the motion between the reference frame and subsequent "looks" of the same scene is only a fractional or sub-pixel distance, the sampled representation is very different. In all cases, there is unique information about the scene. This new information can be exploited to reconstruct a HR image.

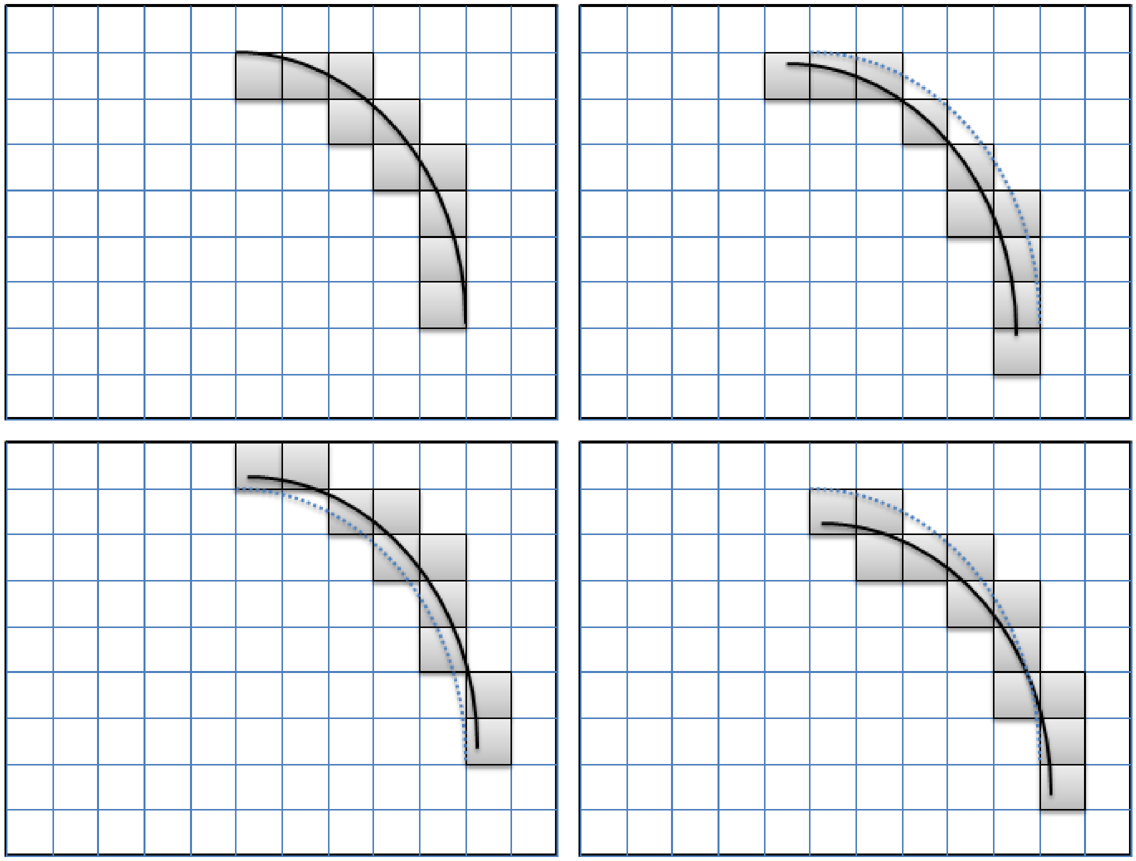


Figure 5 – Illustration of Sub-Pixel Translations Highlighting Structure

The above example is trivial and assumes a binary pixel precision – that is, if any part of the curved black line falls within a pixel, that pixel is on. Imaging sensors convert light into voltage over a range of intensities. For the case of 8-bit, grayscale pixel values, where 0 represents black and 255 represents white, the scene above would produce sampled images similar to the Figure 6.

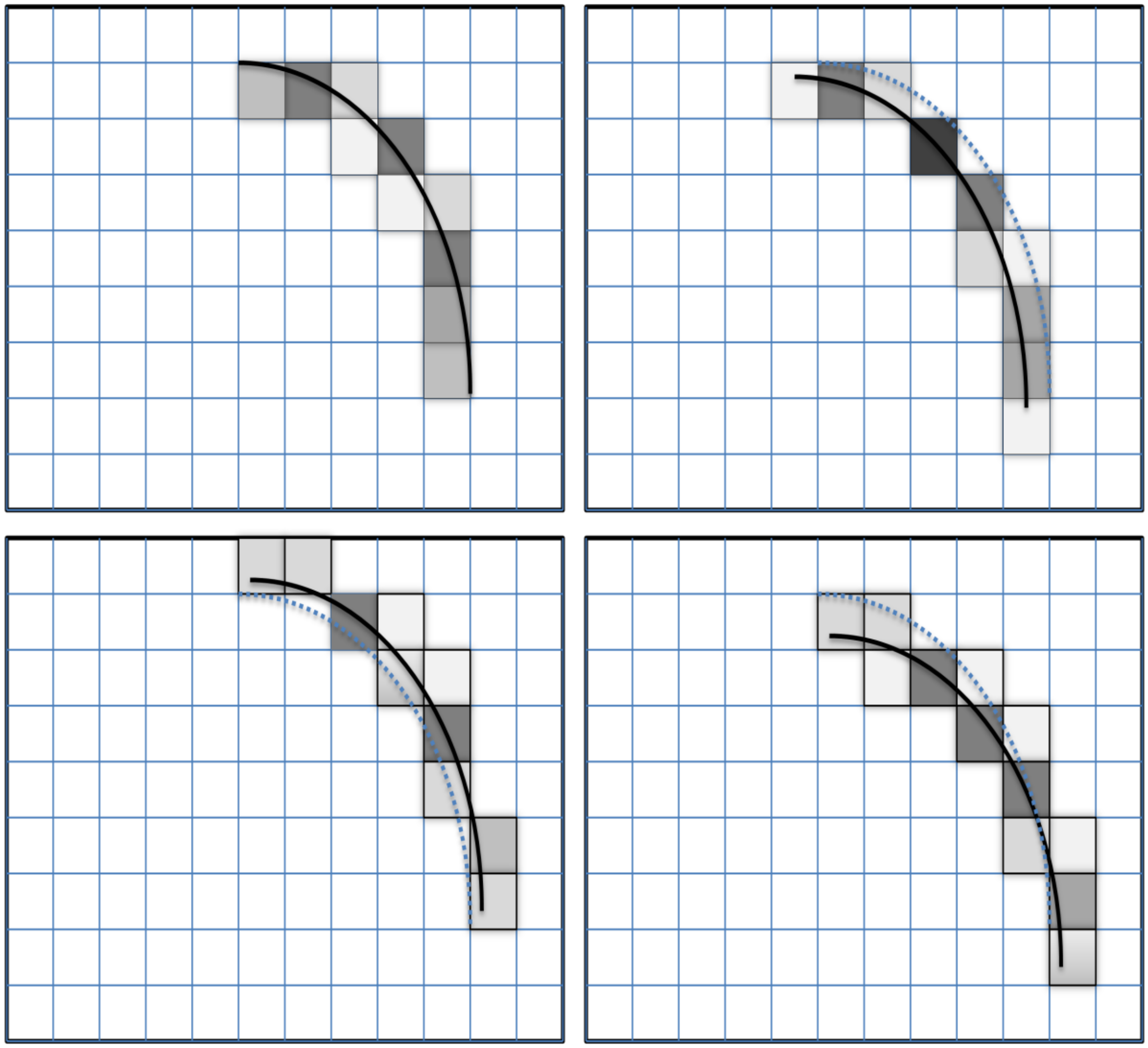


Figure 6 – Illustration of Sub-Pixel Translations Highlighting Intensity

The 8-bit precision of the imaging sensors provides unique information between frames about the intensities of the sampled scene in addition to the position on the sensor grid.

1.8 Application

SR image reconstruction is practical in any instance where multiple images of the same scene can be obtained. As such, there are many practical applications for SR image reconstruction.

1.8a Satellite Imagery

Satellite imagery is one obvious application, and was the driving motivation behind Huang and Tsai's SR development. If the slight offsets between images taken from separate orbits are of sub-pixel accuracy, SR image reconstruction is viable. Land-cover-mapping is an area where SR techniques are necessary for resolving landscape features, especially rural hedges and other thin formations.

1.8b Infrared Imaging

The maximum resolution of a far-field imaging system is diffraction limited by the size of the wavelength being detected - for Mid-Wave Infrared (MWIR), this is 3 to 8 micrometers. This is significantly longer than the 400 to 700 nanometer wavelengths of visible light. Pixels sizes (detector elements) must be large enough to stay within diffraction limits for MWIR imaging. Using detector materials with very high quantum efficiency, MWIR systems can operate at very high speeds despite being resolution limited. For less state of the art MWIR systems, resolutions are limited by large fill factors required for electrical and thermal isolation of pixels [1]. In both cases, SR image reconstruction can be used to overcome these limitations. A technique called controlled microscanning is the basis for the sampling. Controlled microscanning involves calibrated, sub-pixel, translational movement of the imaging sensor which, over time, projects multiple sub-pixel shifted images of the same scene onto the sensor grid [15]. The controlled movement is usually accomplished with a piezo-electric element [7]. In essence, microscanning provides all the necessary requirements for SR image reconstruction.

1.8c Video

The sub-pixel motion requirement necessary for SR image reconstruction does not have to come from movement of the imaging system. In the case of a video sequence, global motion of objects in the scene may be adequate in the temporarily shifted frames, even if the camera remains static. For static scenes, SR image reconstruction is viable so long as long as sub-pixel accuracy can be attained due to vibrations in the camera [2].

1.8d Region of Interest (ROI) Enhancement and Digital Zoom

The enhancement of smaller, ROI objects within a field of view (FOV) is very important in imaging, especially surveillance. An example of this is digital zoom, where a ROI is upsampled ("blown up") to the dimensions of the original FOV. This is almost always accompanied by some form of single image interpolation.

Using SR image reconstruction, multiple LR images of the ROI can improve upon the limitations of single image interpolation. For example, an imaging system capable of recording a particular FOV with pixel dimensions of 100 x 100 at a rate of 10 frames per second (FPS) can theoretically record a 10x10 pixel ROI at 1000 frames per second without exceeding the bandwidth limitations of the system - in both cases, 100,000 pixels are captured per second. This increased sampling rate is necessary to capture multiple images of a fast-moving target and maintain sub-pixel accuracy between the frames. Tracking a license plate leaving the scene of a crime or an enemy missile streaking across the sky are two such examples.

1.8e Medical Imaging

SR image reconstruction is useful in resolution-limited imaging systems such as computed tomography (CT) and magnetic resonance imaging (MRI) which can easily acquire multiple images of the same scene.

1.9 Methods

1.9a Non-Uniform Interpolation

The non-uniform interpolation method of SR image reconstruction is the most intuitive. It is based on the Non-Uniform sampling theorem developed by Clark *et al*. [5]. They developed an algorithm for the 2-D case based on nearest neighbor interpolation. The algorithm begins with a search of samples within a "sector" where the point to be interpolated is at the center of the sector. The samples are given a weight based on their Euclidean distance from the center point. The advantage of non-uniform interpolation is low computational cost, making it ideal for real-time applications. It is limited in that it requires either a priori knowledge of sub-pixel shifts between frames or very accurate sub-pixel registration through a motion estimation algorithm. In addition, this method assumes equal noise and blur across all LR images.

1.9b Frequency Domain

The Frequency Domain approach, developed by Tsai and Huang exploits the shifting property of the Fourier transform and the aliasing relationship between the CFT of an HR image and the DFT of observed LR images. Global translational motion between the HR image (x) and the "new look" of the same scene (x_k) can be written as [12]:

$$X_k(w_1, w_2) = X\left(w_{1,} w_2\right) e^{j2\pi(\delta_{k1} w_1 + \delta_{k2} w_{2)}}$$

After X_k is sampled to produce y_k, the relationship between X (the CFT of the HR image) and Y_k(the DFT of the kth sampled LR image) can be written as:

$$Y = \Phi X$$

Where Y contains the DFT coefficients of all the LR images

X contains the unknown samples of the CFT of the HR image x

Φ is an invertible matrix that relates the DFT of the LR images to X

Solving for the unknown HR image x requires first solving the invertible matrix Φ

The Frequency Domain approach needs no prior knowledge of motion (sub-pixel offsets) in the spatial domain. This is beneficial when that information is unknown. The drawback to the Frequency Domain approach is that it can only consider global translational motion since it relies on the shifting property of the Fourier Transform.

1.9c Regularization Methods

Most forms of SR image reconstruction are ill-posed because of the inverse nature of the model. However, the problem can be regularized assuming there are estimates of the motion parameters (sub-pixel shifts) and a priori knowledge of the solution. This allows SR image reconstruction to be well-posed . An advantage to regularization methods is the ability to more robustly model noise in the system [12].

1.9d Other Methods

A Projection onto Convex Sets method was developed by Stark and Oskoui [16]. An Iterative Back-Projection method was developed by Irani and Peleg [9] to reduce the difference between LR images. An adaptive filter method was developed by Elad and Feuer [6]. The benefit to this method is it an iterative optimization algorithm more suitable for hardware implementation (much less computationally costly) instead of a matrix inversion.

Boorman and Stevenson present an overview of various SR image reconstruction approaches in [2].

1.9e Assumptions

The methods for SR image reconstruction vary in complexity depending on the assumptions made about the system. Modeling noise and blur is much easier if it is assumed constant between LR samples. Also, assuming known global translational motion greatly simplifies the computational complexity of the problem since non-uniform interpolation, or adaptive filtering can be utilized. Since the computational complexity given in Big-O notation of matrix inversions is $O(n^3)$ for standard matrix inversion algorithms such as the Gauss-Jordan elimination matrix, inversions do not scale well in hardware and are ill-conceived for real-time hardware implementation[13]. Because of this, I propose to develop an SR image reconstruction algorithm based on a non-uniform interpolation approach with the goal of high quality image reconstruction at near real-time performance. The design will assume known global translational motion across image acquisitions.

SECTION 2
ALGORITHM CONSIDERATIONS

2.1 Software

All algorithm modeling, testing, and simulation were completed using MATLAB 7.

2.2 Motion

The inputs to the algorithm are the LR image acquisitions and the sub-pixel shifts that exist between them. Using techniques like microscanning enables a priori knowledge of the global sub-pixel motion between images before processing. If the motion is not known after image capture, a Fourier analysis using the Fast Fourier Transform (FFT) can estimate the motion between images based on the shifting property of the Fourier transform. In either case, the algorithm assumes known global translational motion between images – which is necessary for non-uniform interpolation.

2.3 LR Image Acquisitions

In all cases, a digital HR image was used to represent the "continuous scene" of the model. This was for ease of testing and analysis. LR images were acquired from the HR image by upsampling, shifting over integer pixel distances before downsampling by a severe factor. This practice has been used by Vandewalle *et al* [17] and Jahanbin *et al* [10].

Three different images with varying levels of detail were used to represent the continuous scene. Figures 7-9 shows the images used and examples of sampled LR frames.

Figure 7 – Low Detail Scene. The globe contains large areas of uniform intensities as well as sharp edges on the coastlines. The scene is 168x168 pixels and the shifted acquisitions are downsampled by a factor of 4.

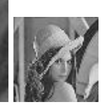


Figure 8 – Medium Detail Scene. Much of the information in the Lenna image is contained in lower frequencies. The areas around her hat and feathers provide higher frequency components. The scene is 512x512 pixels and the shifted acquisitions are downsampled by a factor of 8.

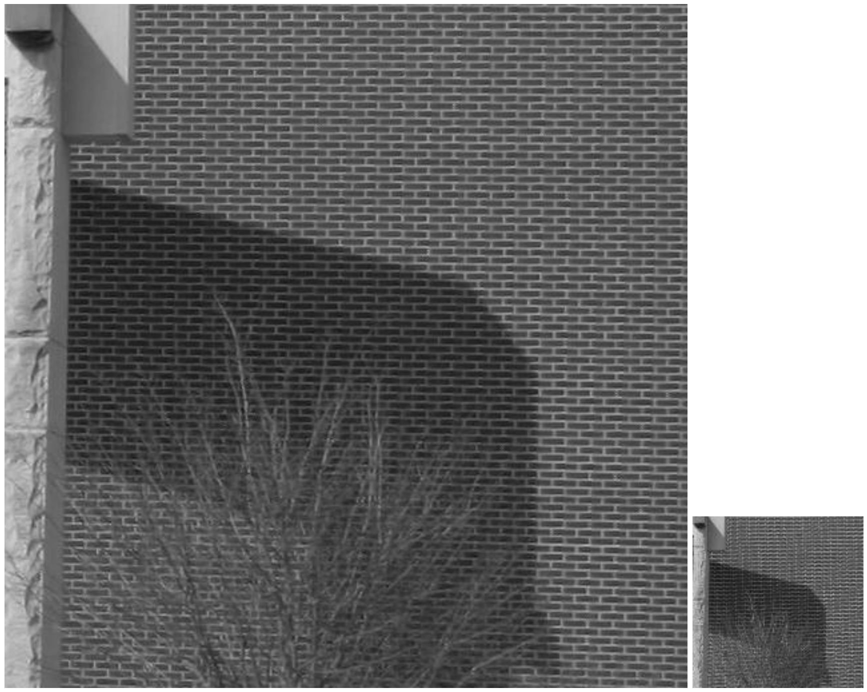


Figure 9 – High Detail Scene. This image of bricks is often used as an example of Moire patterns. The periodic nature of the bricks and their high frequency reveals strong aliasing artifacts after sampling. The scene is 512x512 pixels and the shifted acquisitions are downsampled by a factor of 4.

2.4 Upsampling Factor

The upsampling factors chosen were values that resolved the LR images to the exact dimensions of the scene image.

2.5 Number of LR Images

The optimal number of LR Images depends on the amount of detail present in the images, the sub-pixel offset values, the upsampling factor, as well as the interpolation approach. In all cases, at least 12 LR image acquisitions were used to ensure that there was ample data present to resolve to a higher resolution grid.

2.6 Grids

2.6a HR Grid

Given the LR image acquisitions of the dimension $m \times n$ the HR grid dimension is:

$$l_1 m \times l_2 n$$

where l_1 and l_2 are the respective scaling factors for the x and y dimensions, as shown in Figures 10 and 11. The HR grid points are interpolated from the LR data in the final stage of the algorithm.

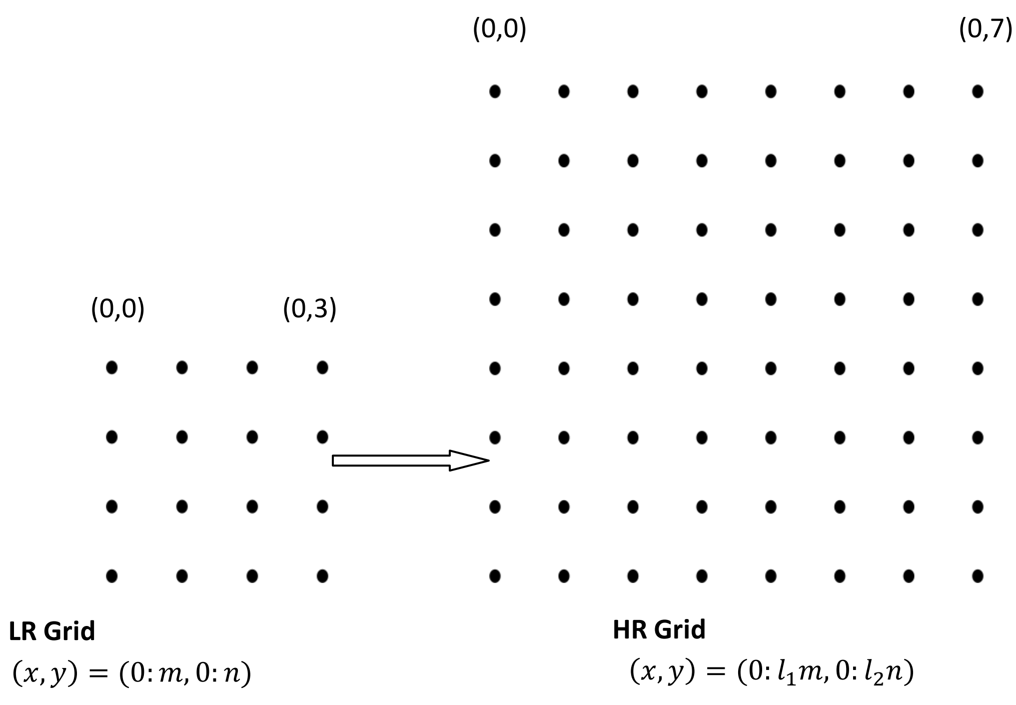


Figure 10 – LR Grid

Figure 11 - HR Grid

2.6b Registration Grid

Given the LR image acquisitions of the dimension $m \times n$ and the HR grid of dimension $l_1 m \times l_2 n$, the registration grid is a fractional grid containing $l_1 m \times l_2 n$ grid points in the x and y dimensions respectively, with the addresses of those grid points scaled between 0 and l_1 for the x dimension and 0 and l_2 for the y dimension. Each grid point in the registration grid corresponds to an HR grid point. In Figure 12, two LR images with sub-pixel translations are overlaid against the registration grid to illustrate the Euclidean distance between pixel locations of the grid points

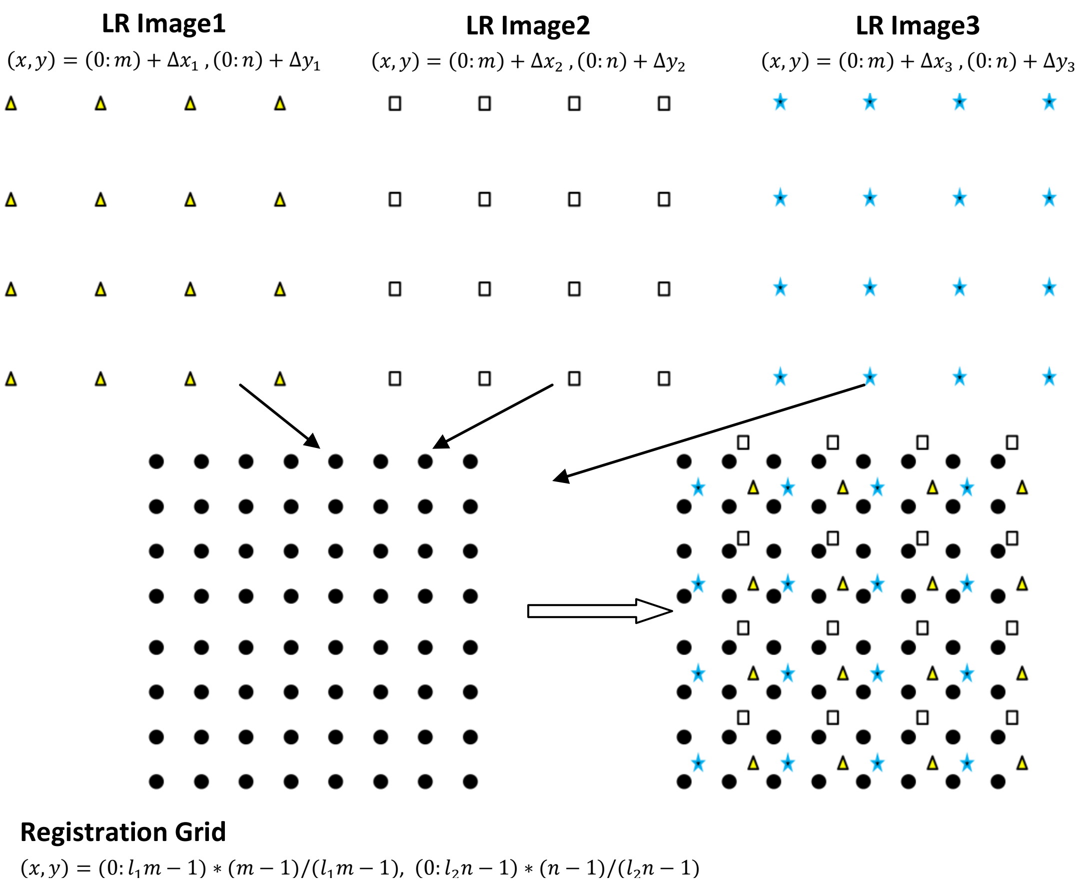


Figure 12 – Illustration of Relationship between LR Image Samples and the Registration Grid

2.7 Gate Sizing

Clark *et al.* described a mapping algorithm based on nearest neighbor interpolation for reconstructing two-dimensional functions from non-uniformly spaced samples [5]. This algorithm assumed a hexagonal lattice as the sampling set as opposed to a square pixel grid. A search process divided into 6 hexagonal sectors mapped the vector angle between sample points and the central grid point. It was shown that for homogenously-distributed, non-uniform sample data, this isotropic search scheme produced quality results. Taking this into consideration, I chose an isotropic search scheme since the global sub-pixel motion between LR images in SR image reconstruction is homogenous - due to global disturbances to the sample grid.

In my algorithm I chose to use an isotropic search to store the radial distance of LR sample points from the grid point to be interpolated. Instead of nearest neighbor interpolation, I chose an interpolation scheme that linearly weighs LR data points. The searching parameter is known as the interpolating gate. Figure 13 demonstrates the gate search layout. It is important to recognize that the algorithm developed by Clark *et al.* considered only the *locations* of samples to be relevant data and used this information to smooth an unknown function. For an image processing application, the intensities of the LR sample points (8-bit grayscale values) are as important to determining the value of the HR grid point as the locations of the LR sample points. The gate search and linear weighting interpolation uses both sets of data from the LR sample points to interpolate each HR grid point.

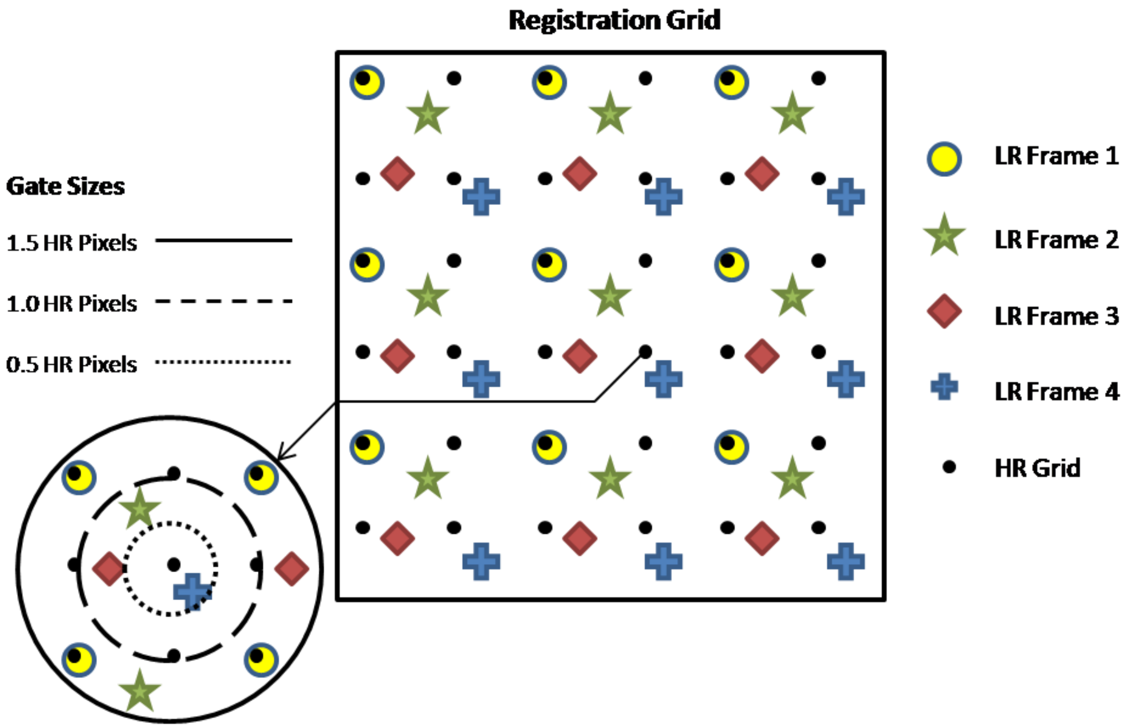


Figure 13 – HR grid point (3,3) shown with 3 different interpolation gates. The smallest gate shown, which searches 0.5 HR pixels isotropically, finds only 1 LR sample point: (2,2) from LR Frame 4. The largest gate shown finds 9 different LR sample points.

Optimal gate sizing depends on the number of LR images, the homogenous nature of the LR samples, (dependant on the sub-pixel offsets) and the frequency components of the scene image. A gate size which is too small will result in certain HR grid points without even a nearest neighbor LR sample to be interpolated from. A gate size that is too large will filter out all the higher frequency components – which SR image reconstruction is meant to restore. Figure 14 shows both cases:

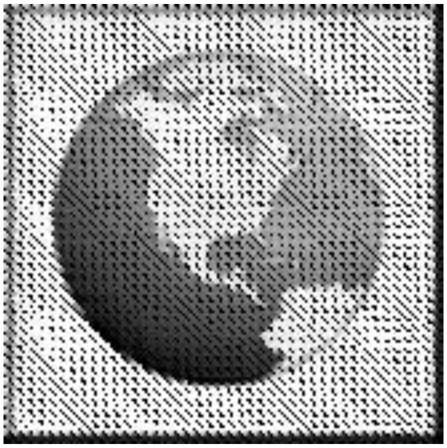
 

Figure 14 – A very small gate size produced the image on the left. All of the missing HR grid points are black because they contain no data. (Grayscale 0 = Black) A very large gate filtered all the high frequency components out of the image on the right.

2.8 Linear Interpolation

The non-uniform interpolation I used assigns weighted coefficients to the grayscale values of the LR pixels that are proportional to the distance from the edge of the gate, or, inversely proportional to their distance from the interpolated pixel. The value of the interpolated pixel, then, is a linear sum of products divided by the total weight within a gate search. If $S(x, y)$ is a grid point of our desired HR image, and there are k LR sample data $p(xi, yi)$ within g HR units of (x, y), then $S(x, y)$ can be written as:

$$S(x,y) = \frac{\sum_{n=1}^{k} a_n p_n}{\sum_{n=1}^{k} a_n}$$

Where:

$$a_n = g - |(x, y) - (x_i, y_j)_n|$$

Figure 15 is an example of this weighted interpolation.

Suppose the represents a LR sample located .3 HR pixel units away from the HR point S, with a grayscale intensity of 100 as shown in Figure 15. Similarly, the represents a LR sample .7 HR pixel units away with a grayscale intensity of 150. And the represents a LR sample .8 HR pixel units away with a grayscale intensity of 200. The gate value is 1 HR pixel units, represented by the dashed circle. From the equation, the value of $S(3.0,3.0)$ is :

$$S(3.0,3.0) = \frac{(0.7)(100) + (0.3)(150) + (0.2)(200)}{0.7 + 0.3 + 0.2} \cong 129$$

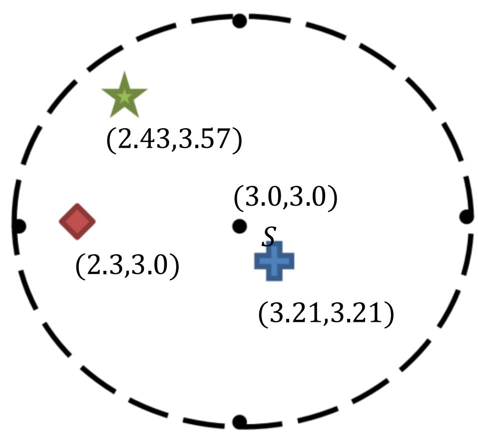


Figure 15 – Interpolation Example.

2.9 Process

2.9a First Stage

The first stage of the algorithm is acquiring all relevant parameters for the SR image reconstruction. These parameters include:

1. The LR input images
2. The sub-pixel translational shifts, associated with the LR input images
3. Grid dimensions for HR, from a scale factor applied to the LR dimensions
4. The value of the interpolating gate
5. The continuous scene image, used for analysis in post-processing

2.9b Second Stage

The generation of the registration grid, the associated (scaled) SR grid, and the weighting matrix occurs during the second stage of the algorithm.

2.9c Third Stage

The following pseudocode describes the third stage of the algorithm – the search and store stage:

```
for (number of images)
    for (x_index of registration grid)
        for(x_index of LR images)
            find(x_distance(registration grid point, LR sample))
            if(x_distance < gate)
                for(y_index of registration grid)
                    for(y_index of LR images)
                        find(y_distance(registration grid point, LR sample))
                        if (y_distance < gate)
                        store(x_distance^2 + y_distance^2)
                        add(weight->weight_matrix(x_index, y_index))
                        add(LR pixel value -> SR grid(x_index, y_index))
end end end end end end

store(divide(SR_grid/weight_matrix)->SR_grid)
```

The algorithm compares the distances between the addresses of the LR data points and the registration grid points and stores the pixel value and the distance into the appropriate matrices as long as the distance is smaller than the gate value. The "SR_grid" generated at the end of stage 3 is the desired HR image.

2.9d Fourth Stage

Stage 4 is where filtering, testing, displaying, and all other post-processing takes place. Specifically, the algorithm can be adjusted to perform any of the following tasks:

1. Filter the HR image with any deblurring matrix, performing value scaling to stabilize any noise.
2. Convert SR_grid, a matrix of values, into an 8-bit integer image file format.
3. On a pixel by pixel basis, quantitatively compare the HR image to the continuous scene using a number of measurement techniques.
4. Compare the results of step 3 to those of a single interpolated LR image.
5. Display the HR image, continuous scene, and upsampled LR image to qualitatively compare the results of the SR image reconstruction algorithm.

The entire process can be wrapped in a loop that profiles the results against varying interpolation gate sizes.

2.10 Optimization

A simple, yet effective optimization of the code was to window the searches in the y dimension. Instead of an exhaustive search checking every LR sample in the y-dimension against a particular registration grid point, the code was adjusted to only compare LR samples within a gate's width. This dramatically cut down on the number of loop iterations.

2.11 VHDL Considerations

Since most FPGAs only synthesize fixed point values, precision errors become a significant factor when dealing with floating point numbers. Initially, the IEEE double-precision floating point standard was the data type used for calculations in the algorithm. A separate port of the algorithm was written in MATLAB which scaled all floating point values by a large factor (2^6) to maintain some of the precision before converting the values to an integer format. All calculations were performed on the integer values which introduced rounding error. Fortunately, there is very little error propagation in the algorithm. The results of the MATLAB simulations show that the rounding errors introduced by the integer conversion were not substantial, since the large scaling factors limited the amount of precision lost in the process.

SECTION 3

TESTING AND ANALYSIS

The primary goal of SR image reconstruction is a HR output image. The testing benchmarks for the algorithm are:

1.) A Sum Squared Error analysis of the HR image

2.) A Measure of Average Pixel Difference between Scene image and HR image

3.) A Count of the Total Correct Pixels

4.) Structural Similarity Index (SSIM)

5.) Visual (qualitative) Analysis

The secondary goal of this algorithm is near real-time processing. Processing time was recorded for every test.

A non-shifted LR sample of each scene image was interpolated and used as a baseline measure of quality. The interpolation method used was MATLAB's cubic interpolation function. All charts and images were generated using MATLAB.

3.1 Globe Test (Low Detail Scene)

The globe contains large areas of uniform intensities as well as sharp edges on the coastlines. The LR images were sub-pixel shifted and then downsampled by a factor of 4. Figure 16 shows the scene image, Table 1 lists the X and Y sub-pixel translations of the LR images for the Globe test.

Figure 16 – The continuous scene image (168x168) for the Globe test

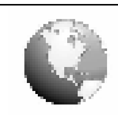
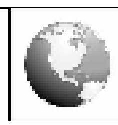
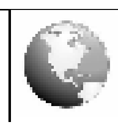
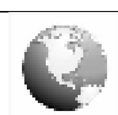
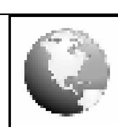
LR Image #	LR Image	Xshift	Yshift
1		0.7	0.3
2		0.3	0.7
3		-0.3	0.7
4		-0.7	0.3
5		-0.7	-0.3
6		-0.3	-0.7
7		0.3	-0.7
8		0.7	-0.3
9		0.3	-0.3
10		0.3	0.3
11		-0.3	0.3
12		-0.3	-0.3

Table 1 – LR Inputs to Globe Test

3.1a Sum Squared Error

Sum Squared error measures the precision of the algorithm. The majority of the error is contained in the highly divergent pixels since their values are squared. Missing pixels can greatly increase the sum squared error of images since they are forced to '0' (black). This occurs when the gate values are too small to find any LR image data. The sum squared error e_H between the scene image S_C and the super-resolved HR image S_H was calculated using:

$$e_H = \sum_{1}^{m} \sum_{1}^{n} [S_C(x,y) - S_H(x,y)]^2$$

The results were calculated over a range of interpolating gate values for the algorithm. As a baseline, the sum squared error was calculated between the scene image and a single-frame interpolation of a non-shifted LR sample. This value is 12×10^6. Table 2 shows three different HR image outputs and their associated sum squared error. Figure 17 shows the sum squared error over a range of gate values.

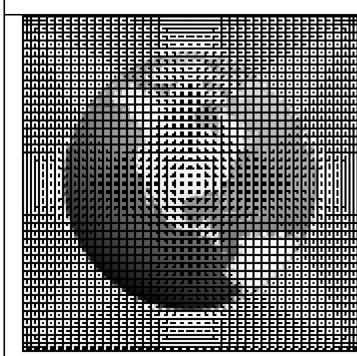
Gate Value = 0.2	Gate Value = 0.8	Gate Value = 1.5
		
$e_H = 5.8 \times 10^8$	$e_H = 5.4 \times 10^6$	$e_H = 8.8 \times 10^6$

Table 2 – Various HR outputs and their associated sum squared error

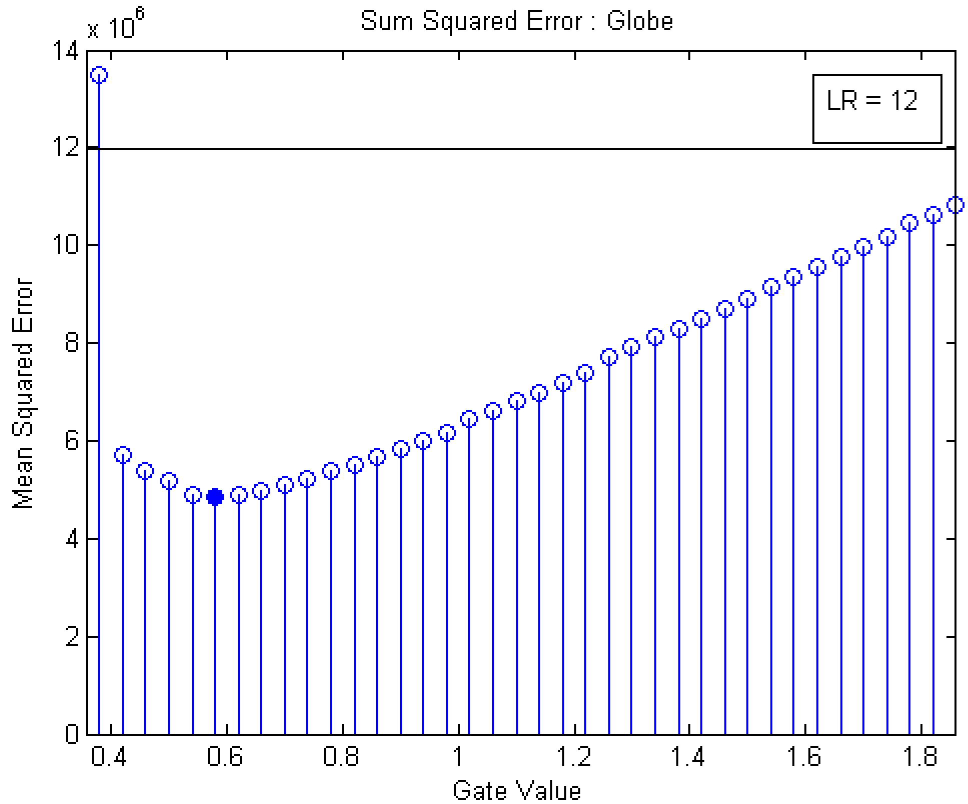


Figure 17 – Sum Squared Error Measurement for the Globe Test

3.1b Average Pixel Difference

The average pixel difference is less susceptible to highly divergent pixels than the sum squared error measurement. As such, missing pixels, caused by small gate values, do not contribute as much to noise. Instead, large gate values which low-pass-filter the data have a greater effect on the average pixel difference than they do in the sum squared error measurement (where they stabilize divergent pixels). The average pixel difference ρ_H between the scene image S_C and the super-resolved HR image S_H was calculated using:

$$\rho_H = \frac{\sum_1^m \sum_1^n |S_C(x,y) - S_H(x,y)|}{m \times n}$$

Figure 18 shows the average pixel difference (in 8-bit grayscale units) over a range of gate values. As a baseline, the average pixel difference was calculated between the scene image and a single-frame interpolation of a non-shifted LR sample.

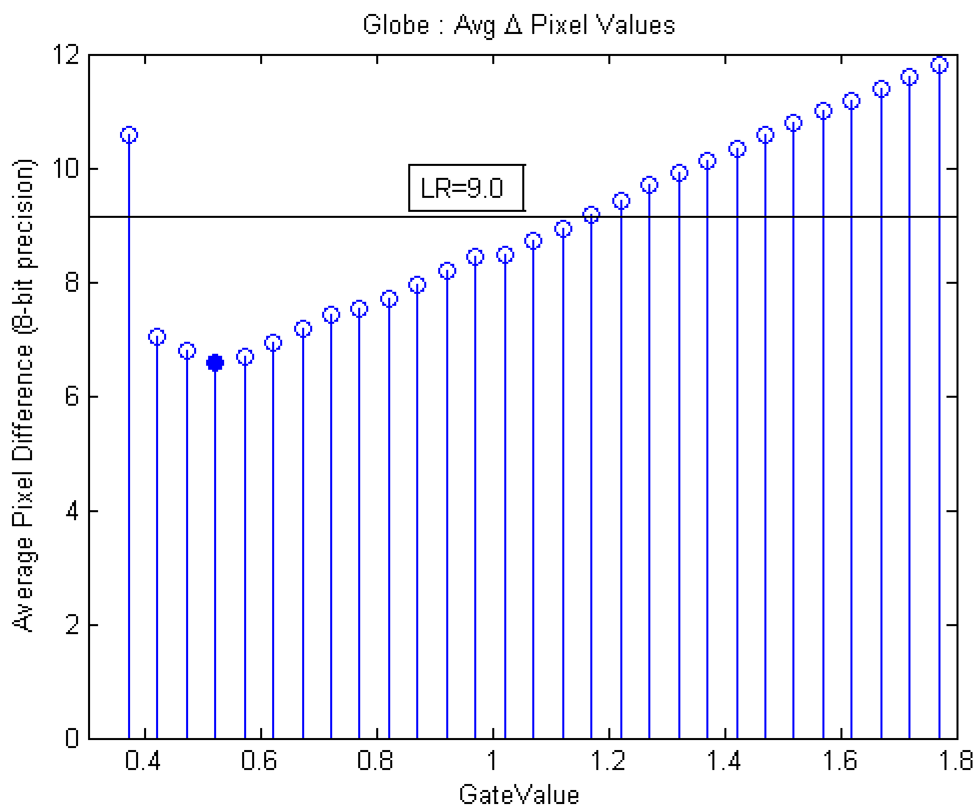


Figure 18 – Average Pixel Difference Measurement for the Globe test

3.1c Correct Pixels Count

A count of correct pixel values is a simple way to determine the algorithm's ability to resolve a target HR image. However, higher correct pixel counts do not necessarily indicate higher quality images. An image processed with a severely low gate value (resulting in high sum squared error) can have as many correct pixels as images with much lower sum squared error. Table 3 demonstrates this. Figure 19 shows the correct pixel count over a range of gate values.

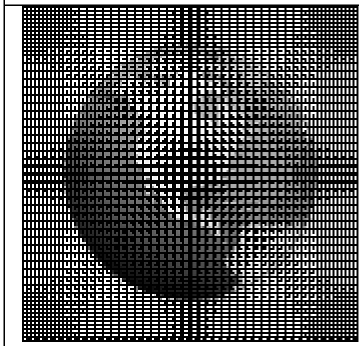
Gate Value = 0.17	Gate Value 0.75	Gate Value = 1.0
		
Correct Pixels = 1279	Correct Pixels = 1300	Correct Pixels = 970

Table 3 – The number of correct pixels between various HR output and the scene image

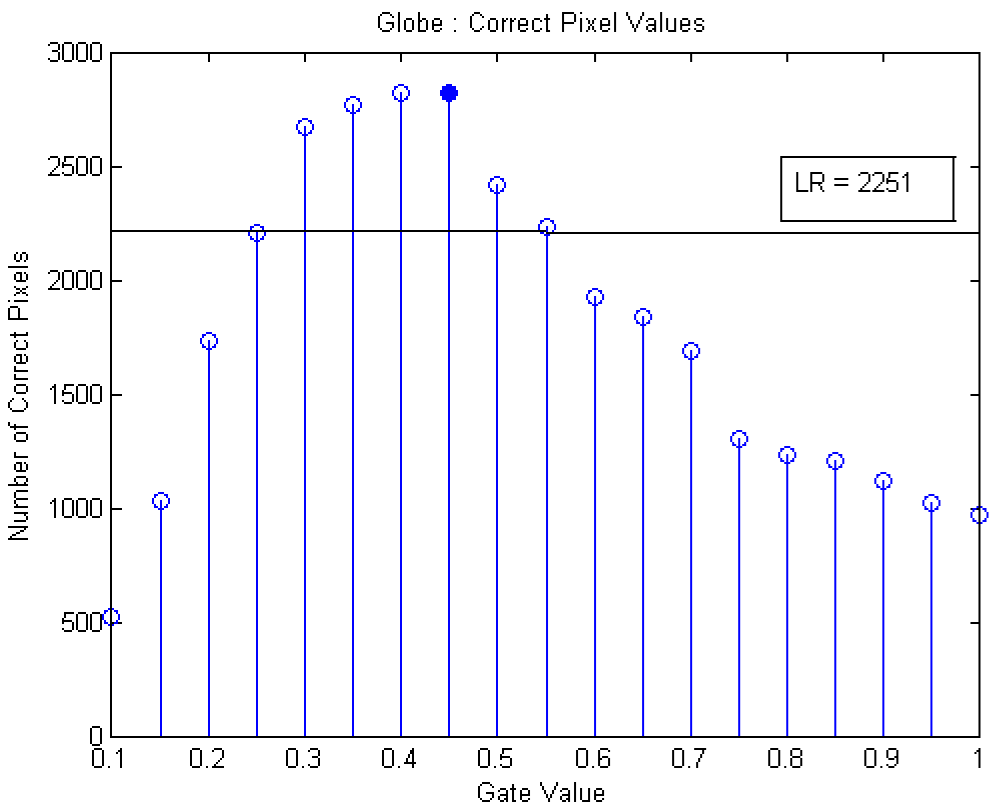


Figure 19 – Correct Pixel Counts for the Globe test

3.1d Structural Similarity

The quality of SR image reconstruction is often demonstrated visually. As has been shown in the results above, error is not always strongly correlated with visual perception (or lack thereof). Wang *et al.* developed a quality index called Structural Similarity (or SSIM) based on the degradation of structural components. It has been demonstrated it to be a more accurate measure of visual quality[10][18]. The range of SSIM is $-1 \le SSIM \le 1$ where 1 is perfectly equal data. The mean structural similarity (MSSIM) is calculated using the source code from [19]. Table 4 shows 3 different HR output images and their associated MSSIM index. Figure 20 shows the MSSIM index over a range of gate values.

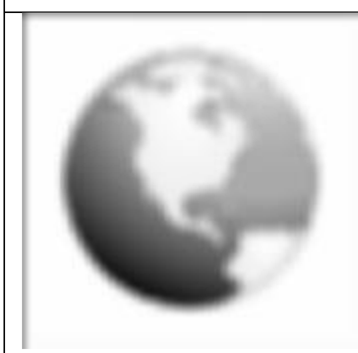
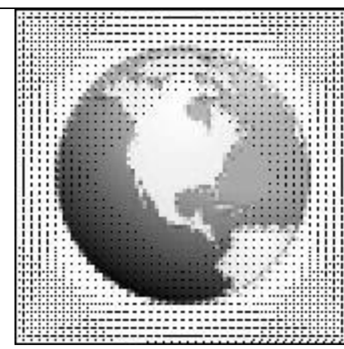
Gate Value = 1.5	Gate Value 0.4	Gate Value = 0.3
		
MSSIM = .76407	MSSIM = .82801	MSSIM = .22039

Table 4 – The calculated MSSIM index between various HR output images and the scene image

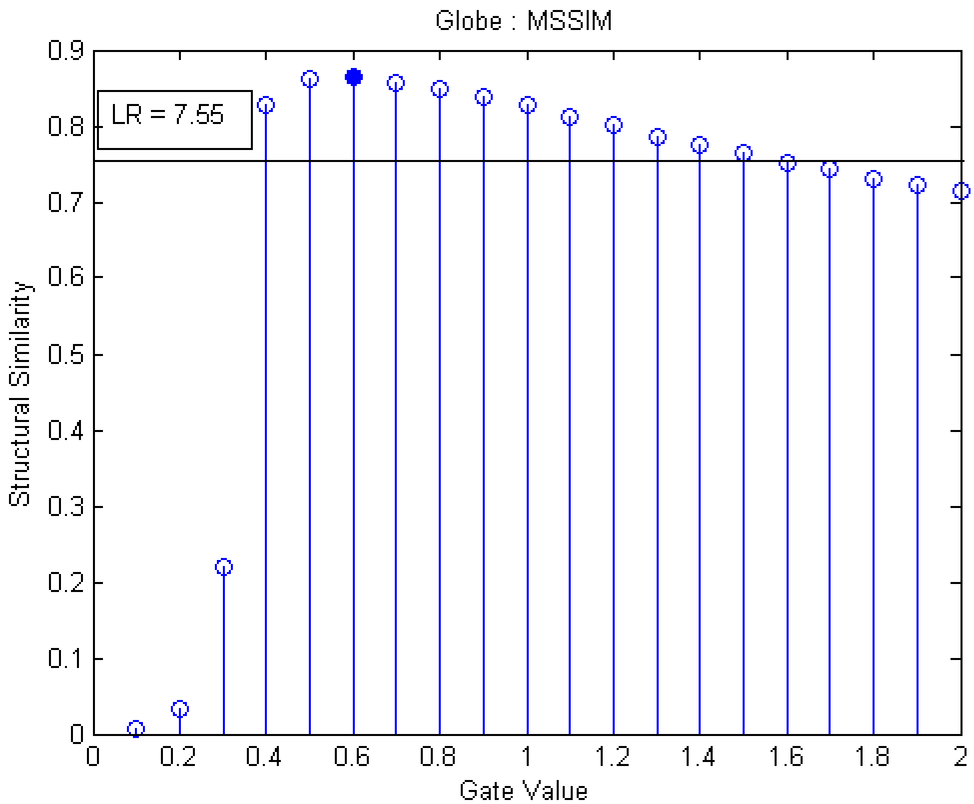


Figure 20 – Measure of MSSIM index for the Globe test

3.1e Visual Analysis

Table 5 shows the scene image, an interpolated non-shifted LR image, and the highest quality SR reconstructed image based on subjective observation. Another way to visually determine image quality (as a comparison) in the spatial domain is to generate an image of scaled pixel error, where white represents high divergence and black represents equal pixels. These images generally show the frequency response in the spatial domain. Very sharp white edges demonstrate an image that passes most high frequency components. Blurred edges, inversely, demonstrate an image's inability to capture the higher frequency components of the scene image. Table 6 compares the cubic interpolation of a single non-shifted LR sample of the scene image with the highest quality SR reconstructed image. Tables 5 and 6 demonstrate the superior reconstructive ability of the SR image reconstruction algorithm over a cubic interpolation of a single LR sample.

Scene Image	LR Interpolated Image	Subjective Best HR Image
		

Table 5 – Visual Demonstration of SR Quality for the Globe test

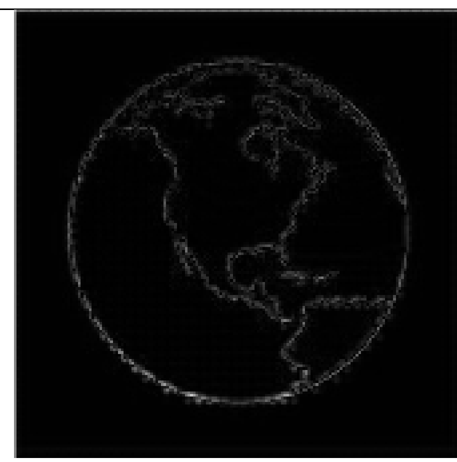
LR Interpolated Image	Subjective Best HR Image (Gate Value = 0.56)
	

Table 6 – Scaled Pixel Error for the Globe test. Higher intensities (white) demonstrates greatest error.

3.2 Lenna Test (Medium Detail Scene)

The famous Lenna image contains medium detail overall with low frequency components in the background and high frequency components around her hat and feathers. The LR images were sub-pixel shifted and then downsampled by a factor of 8. Figure 21 shows the scene image, Table 7 lists the X and Y sub-pixel translations of the LR images for the Lenna test.

Figure 21 – The continuous scene image (512x512) for the LennaTest (Image scaled to 90%)

LR Image #	Xshift	Yshift
1	.1	.9
2	.9	.9
3	.9	.1
4	.7	.3
5	.7	.7
6	.3	.7
7	.9	-.1
8	.9	-.9
9	.1	-.9
10	.3	-.7
11	.7	-.3
12	.7	-.7
13	-.1	-.9
14	-.9	-.9
15	-.9	-.1
16	-.7	-.3
17	-.7	-.7
18	-.3	-.7
19	-.9	.1
20	-.9	.9
21	-.1	.9
22	-.3	.7
23	-.7	.7
24 	-.7	.3

Table 7 – LR Inputs to Lenna test with LR_24 shown as an example

3.2a Sum Squared Error

Figure 22 shows the sum squared error over a range of gate values.

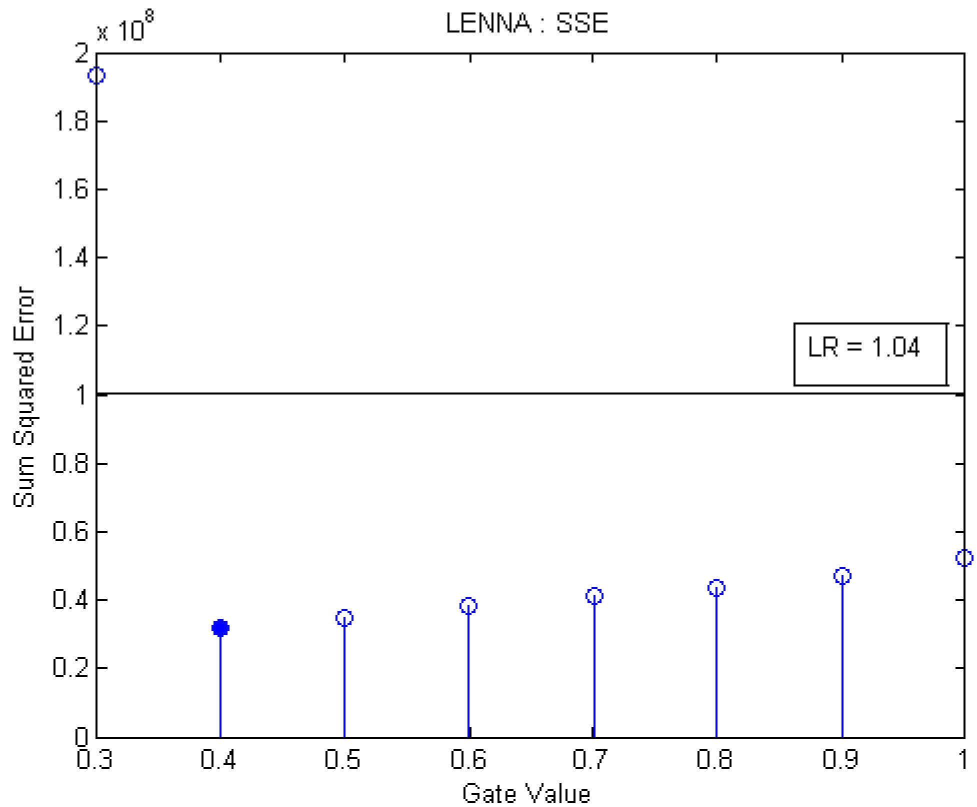


Figure 22 – Sum Squared Error Measurement for the Lenna test

3.2b Average Pixel Difference

Figure 23 shows the average pixel difference (in 8-bit grayscale units) over a range of gate values.

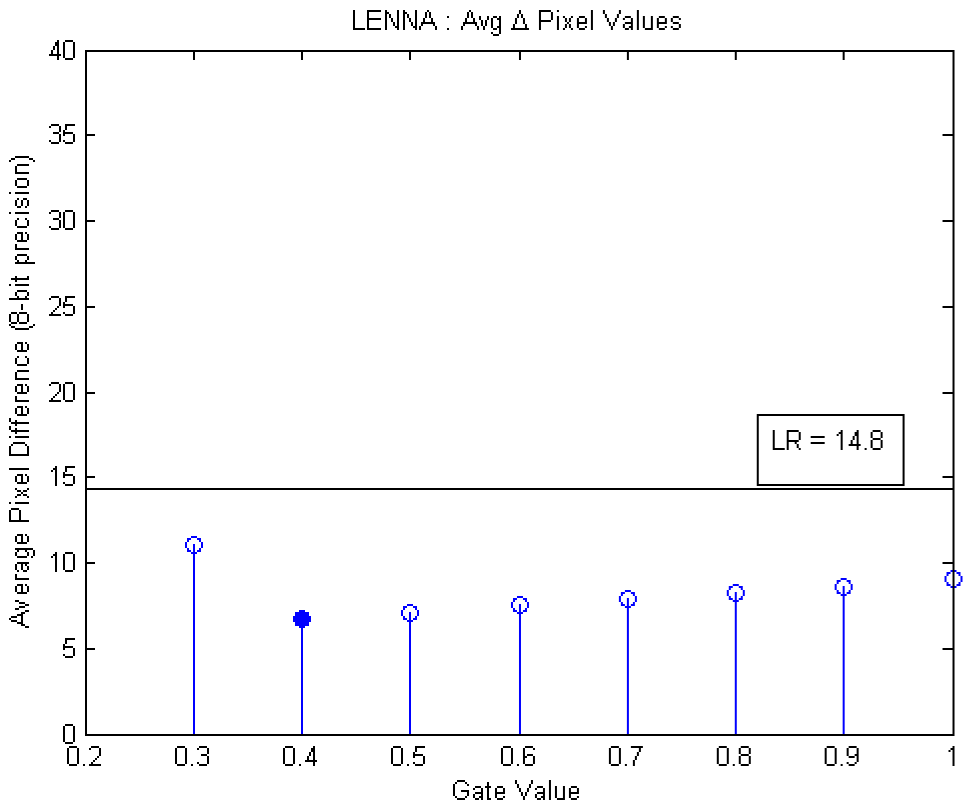


Figure 23 – Average Pixel Difference Measurement for the Lenna test

3.2c Correct Pixel Values

Figure 24 shows the correct pixel count over a range of gate values.

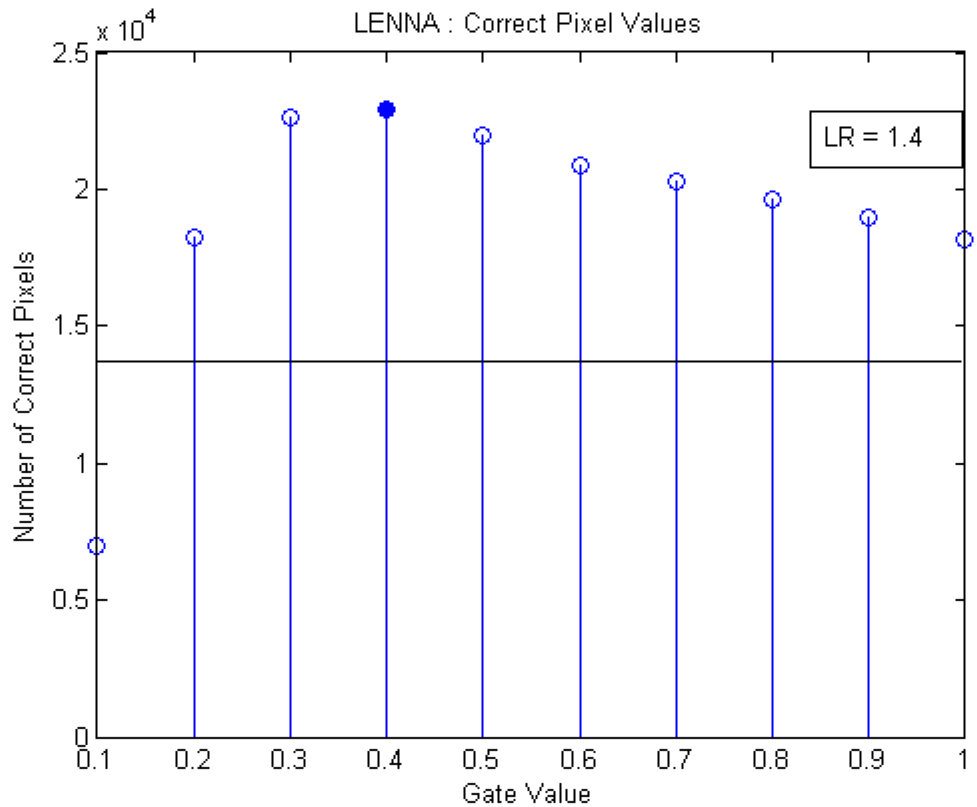


Figure 24 – Correct Pixel Counts for the Lenna test

3.2d Structural Similarity

Figure 25 shows the MSSIM index over a range of gate values.

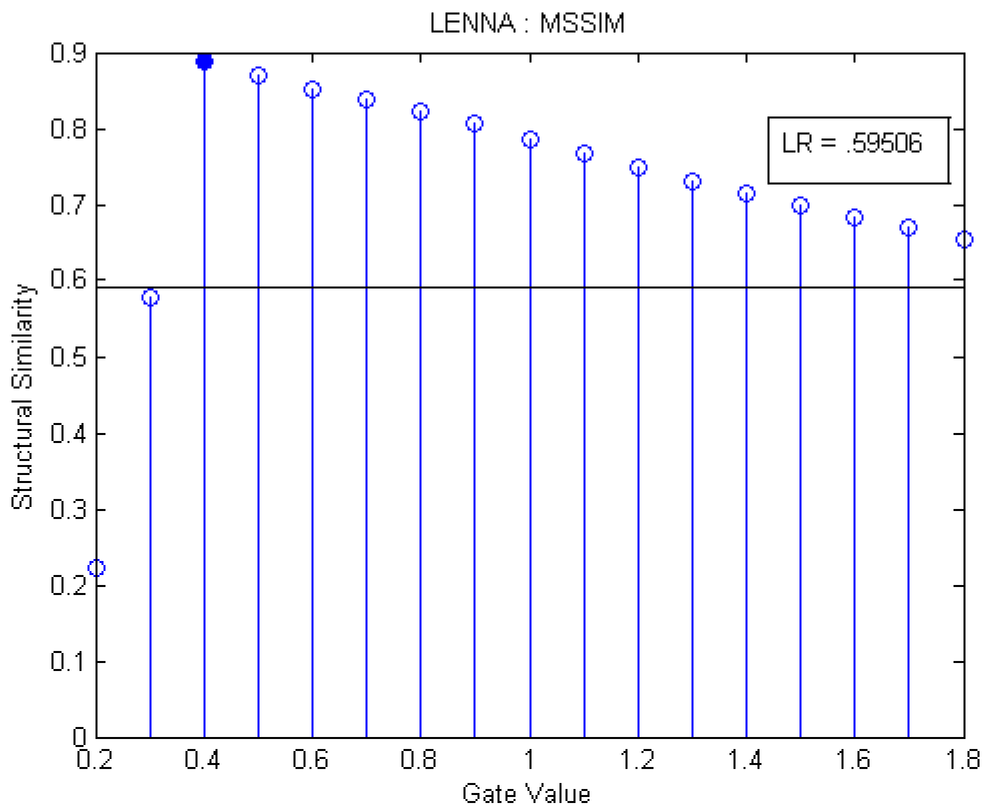


Figure 25 - Measure of MSSIM index for the Lenna test

The MSSIM of HR image with gate value 0.37 is .88976, compared to .59506 of the LR interpolated image. This result confirms the superiority of the SR image reconstruction algorithm over single image interpolation.

3.2e Visual Analysis

Figure 26 is the continuous scene, Figure 27 is the interpolated, non-shifted LR image, and Figure 28 is the highest quality SR reconstructed image based on subjective observation.

Figure 29 is the scaled pixel error for the LR image interpolated, and Figure 30 is the scaled pixel error for theHQ SR image.

Figure 26 – Continuous Scene Image of Lenna (scaled 80%)

Figure 27 – LR interpolated Image of Lenna (scaled 80%)

Figure 28 – Subjective Best HR Image (Gate Value = 0.38) (Scaled 80%)

Figure 29 – Scaled Pixel Error for the interpolated LR image in the Lenna test. Higher intensities (white) demonstrates greatest error.

Figure 30 – Scaled Pixel Error for the Best HR image in the Lenna test.

3.2f Analysis of Lenna Test

The MSSIM results were particularly impressive for this test case, esecially compared to the interpolated LR image. The visual results from this test show the effects of using isotropic interpolation gates, as the image shows slight circular artifacts throughout.

3.3 Brick Test (High Detail Scene)

The Brick image is often used as an example of aliasing – specifically Moire patterns. The periodic, high-frequency sections of bricks can become heavily aliased when the image is sampled. The LR images were sub-pixel shifted and then downsampled by a factor of 4. Figure 31 shows the scene image, Table 8 lists the X and Y sub-pixel translations of the LR images for the Brick test.

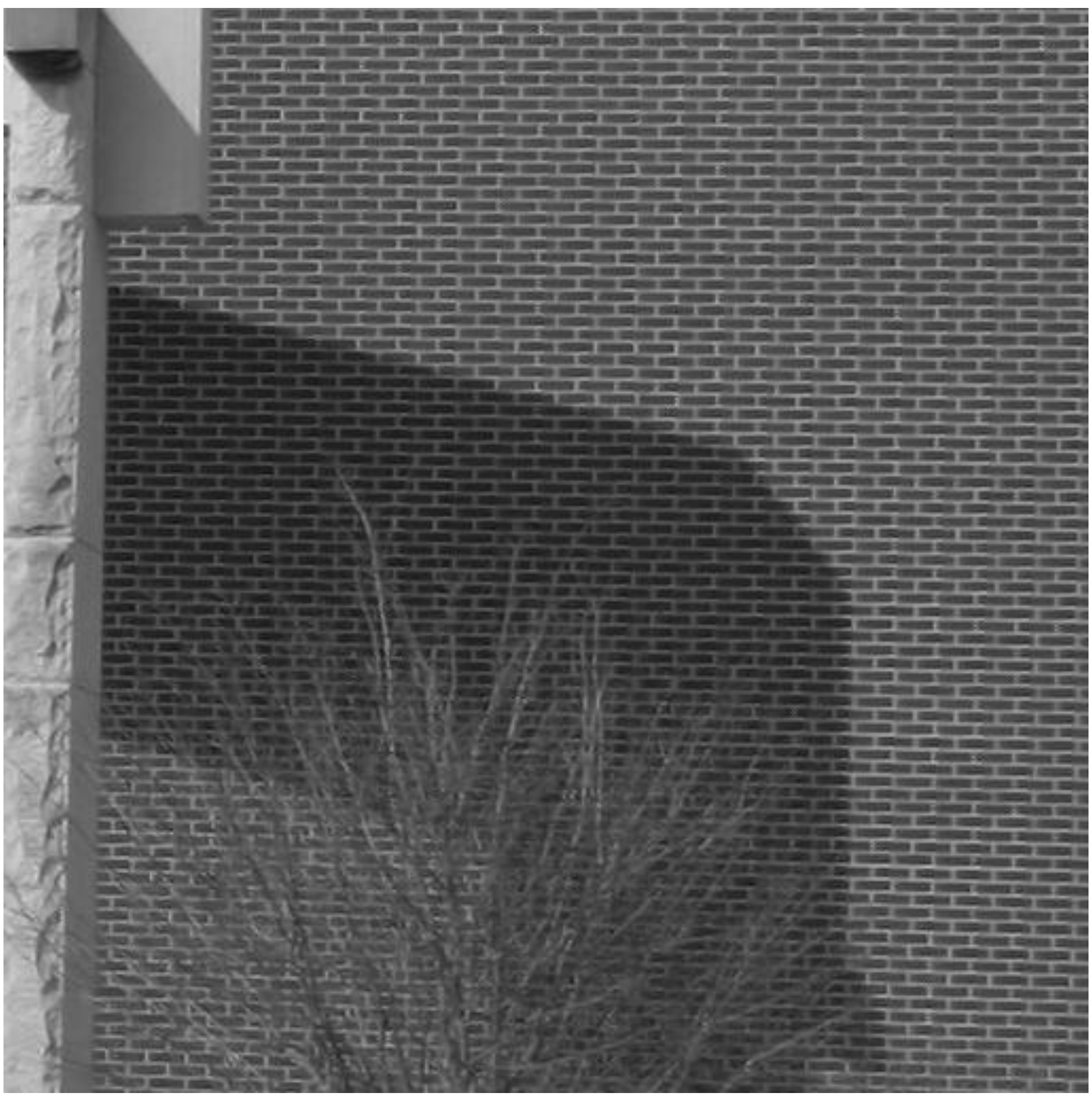


Figure 31 – The continuous scene image (512x512) for the Brick test (Image scaled to 80%)

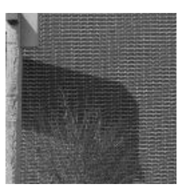
LR Image #	Xshift	Yshift
1	.1	.9
2	.9	.9
3	.9	.1
4	.7	.3
5	.7	.7
6	.3	.7
7	.9	-.1
8	.9	-.9
9	.1	-.9
10	.3	-.7
11	.7	-.3
12	.7	-.7
13	-.1	-.9
14	-.9	-.9
15	-.9	-.1
16	-.7	-.3
17	-.7	-.7
18	-.3	-.7
19	-.9	.1
20	-.9	.9
21	-.1	.9
22	-.3	.7
23	-.7	.7
24 	-.7	.3

Table 8 – LR Inputs to Brick test with LR_24 shown as an example

3.3a Sum Squared Error

Figure 32 shows the sum squared error over a range of gate values.

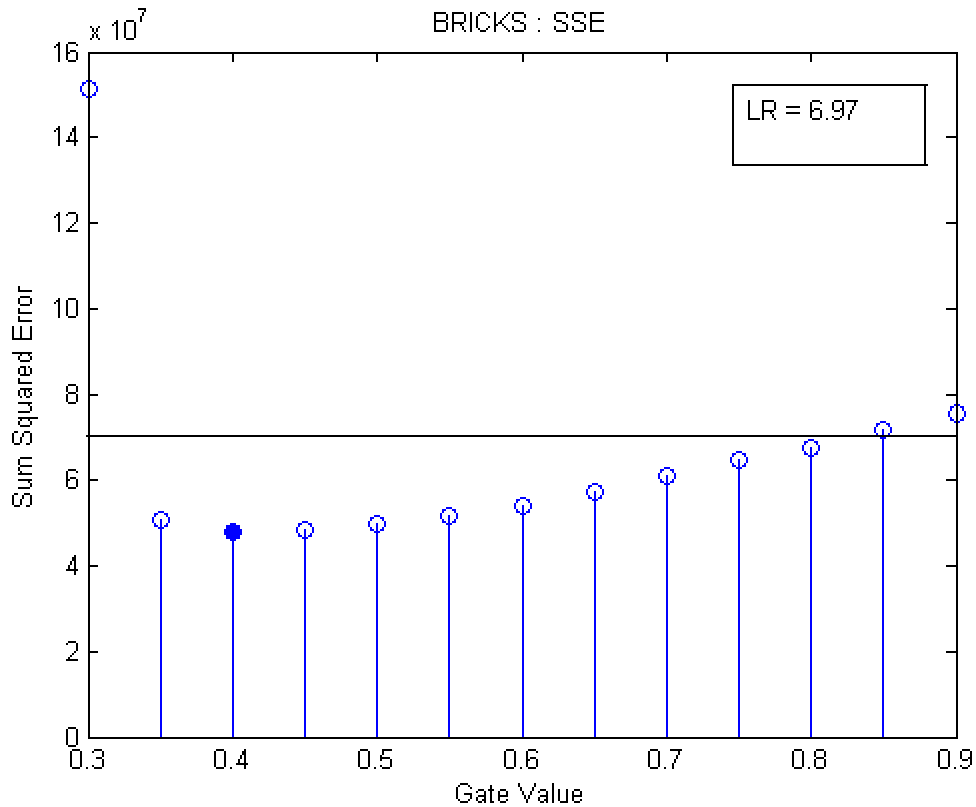


Figure 32 – Sum Squared Error Measurement for the Brick test

3.3b Average Pixel Difference

Figure 33 shows the average pixel difference (in 8-bit grayscale units) over a range of gate values.

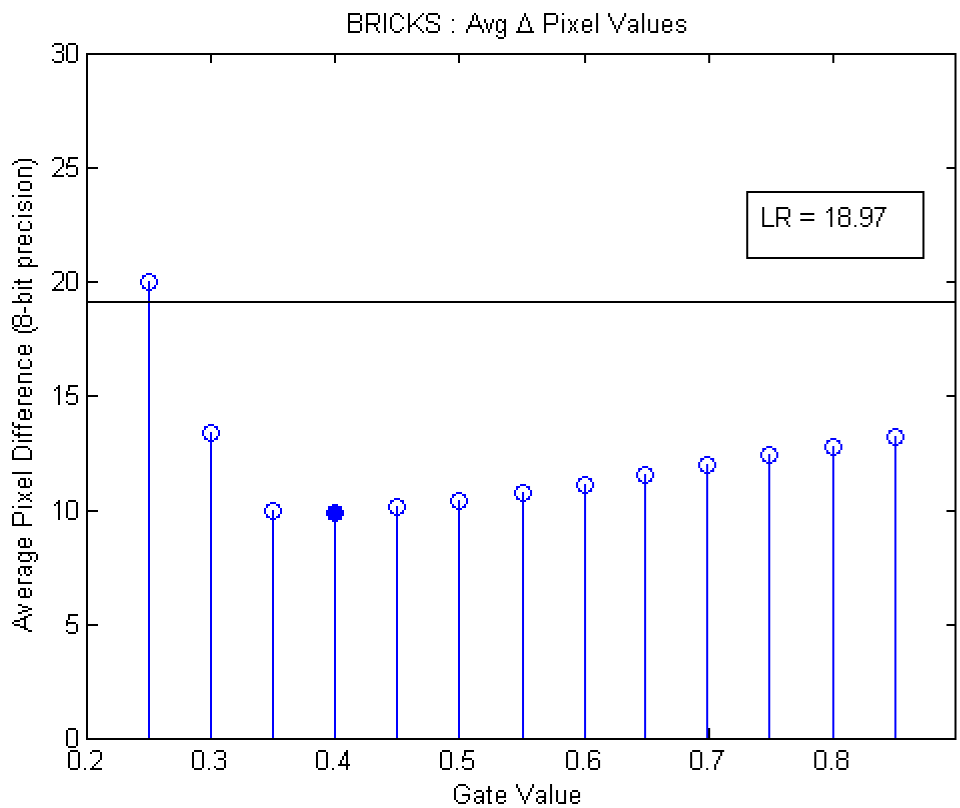


Figure 33 – Average Pixel Difference Measurement for the Brick test

3.3c Correct Pixel Values

Figure 34 shows the correct pixel count over a range of gate values.

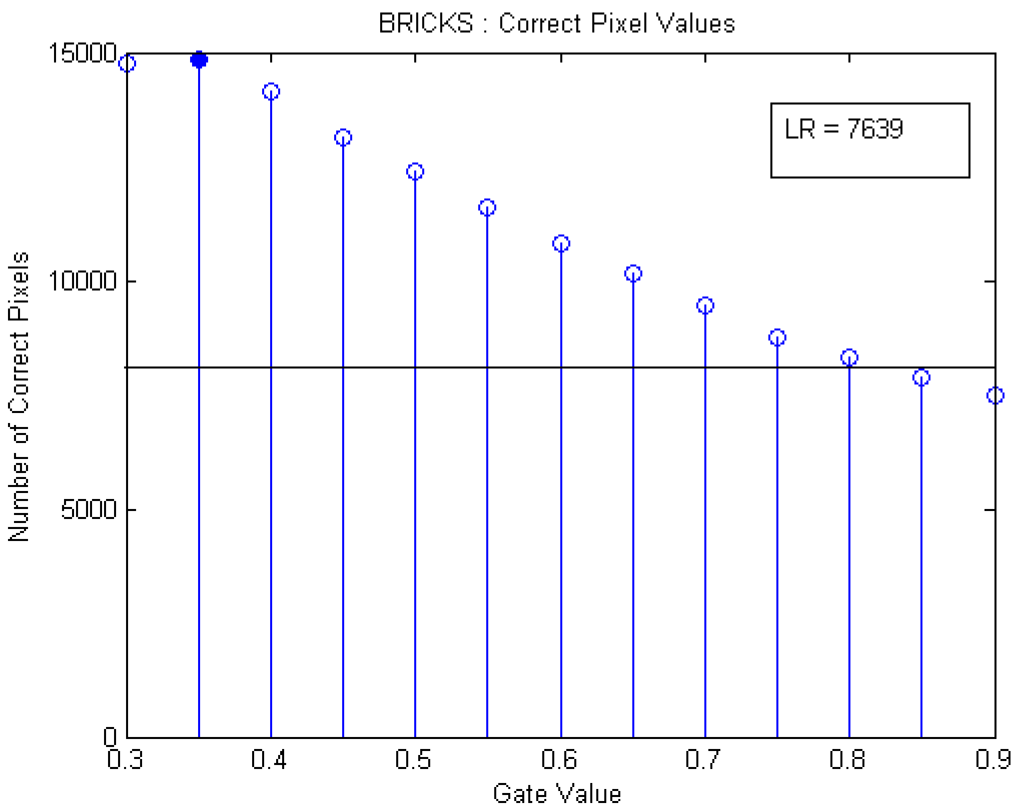


Figure 34 – Correct Pixel Counts for the Brick test

3.3d Structural Similarity

Figure 35 shows the MSSIM index over a range of gate values.

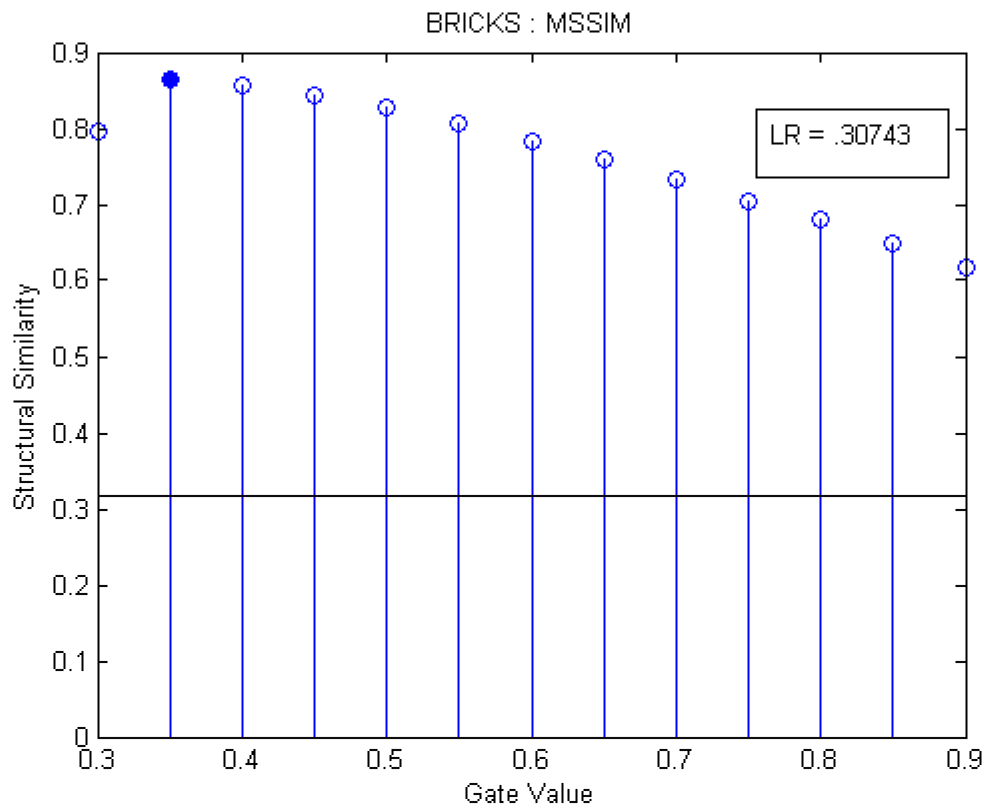


Figure 35 - Measure of MSSIM index for the Brick test

The MSSIM of HR image with gate value 0.35 is .86361, compared to .30743 of the LR interpolated image. Once again, MSSIM shows the superiority of the SR image reconstruction algorithm over interpolation.

3.3e Visual Analysis

Figure 36 is the continuous scene, Figure 37 is the non-shifted LR image interpolated, and Figure 38 is the highest quality SR reconstructed image based on subjective observation.

Figure 39 is the scaled pixel error for the LR image interpolated, and Figure 40 is the scaled pixel error for theHQ SR image.

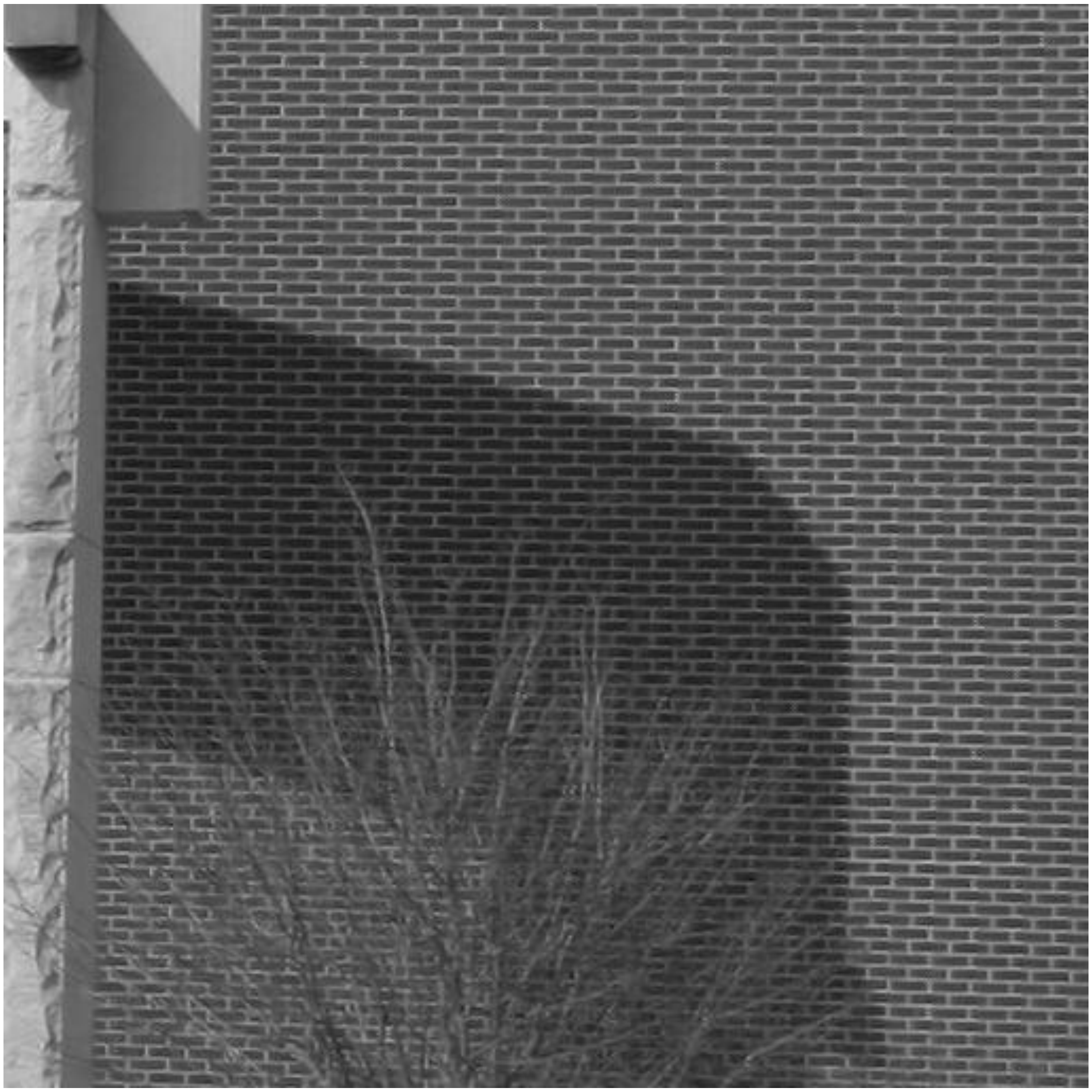


Figure 36 – Continuous Scene Image of Brick Image (scaled 80%)

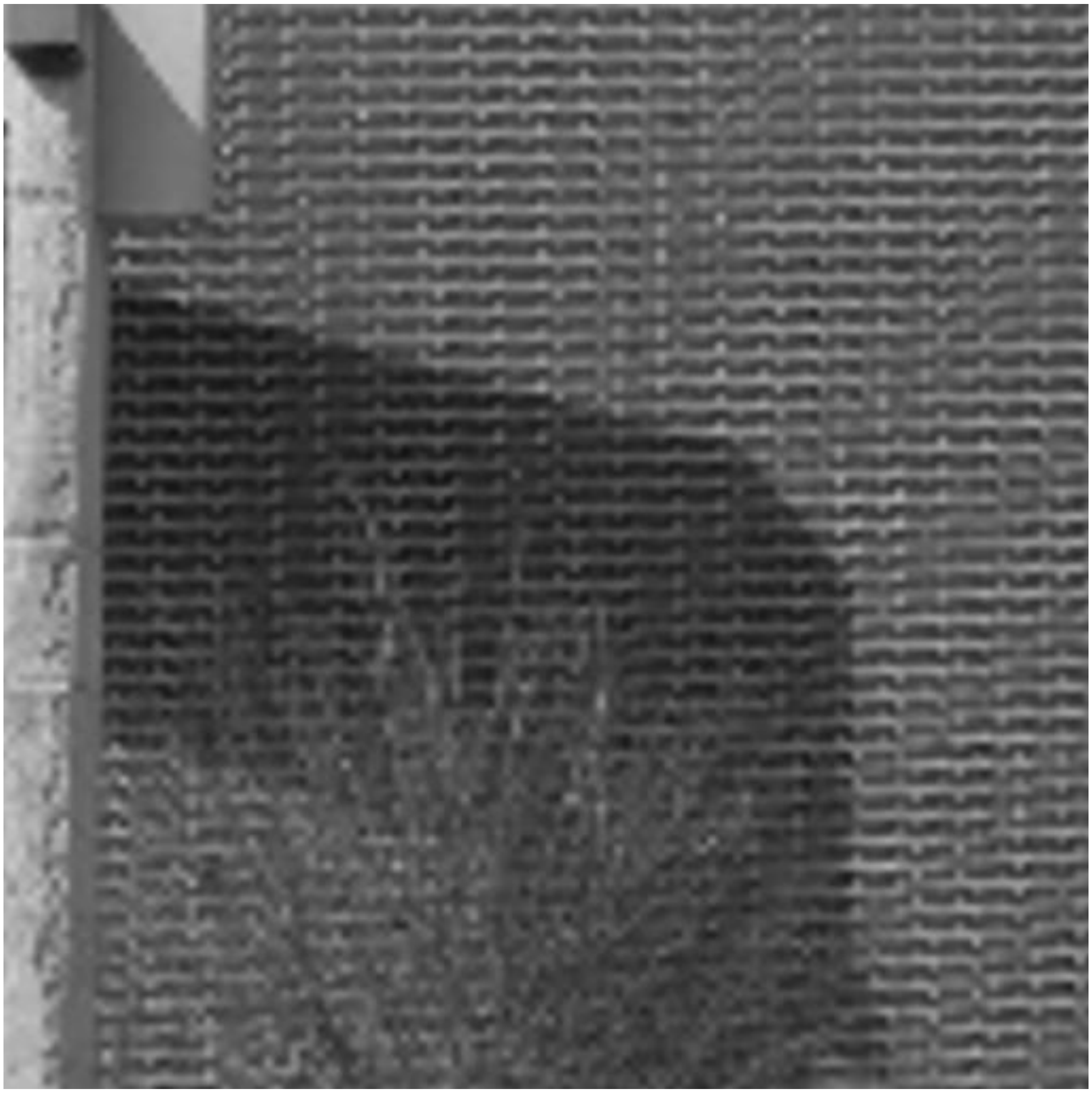


Figure 27 – LR interpolated Image of Brick Image (scaled 80%)

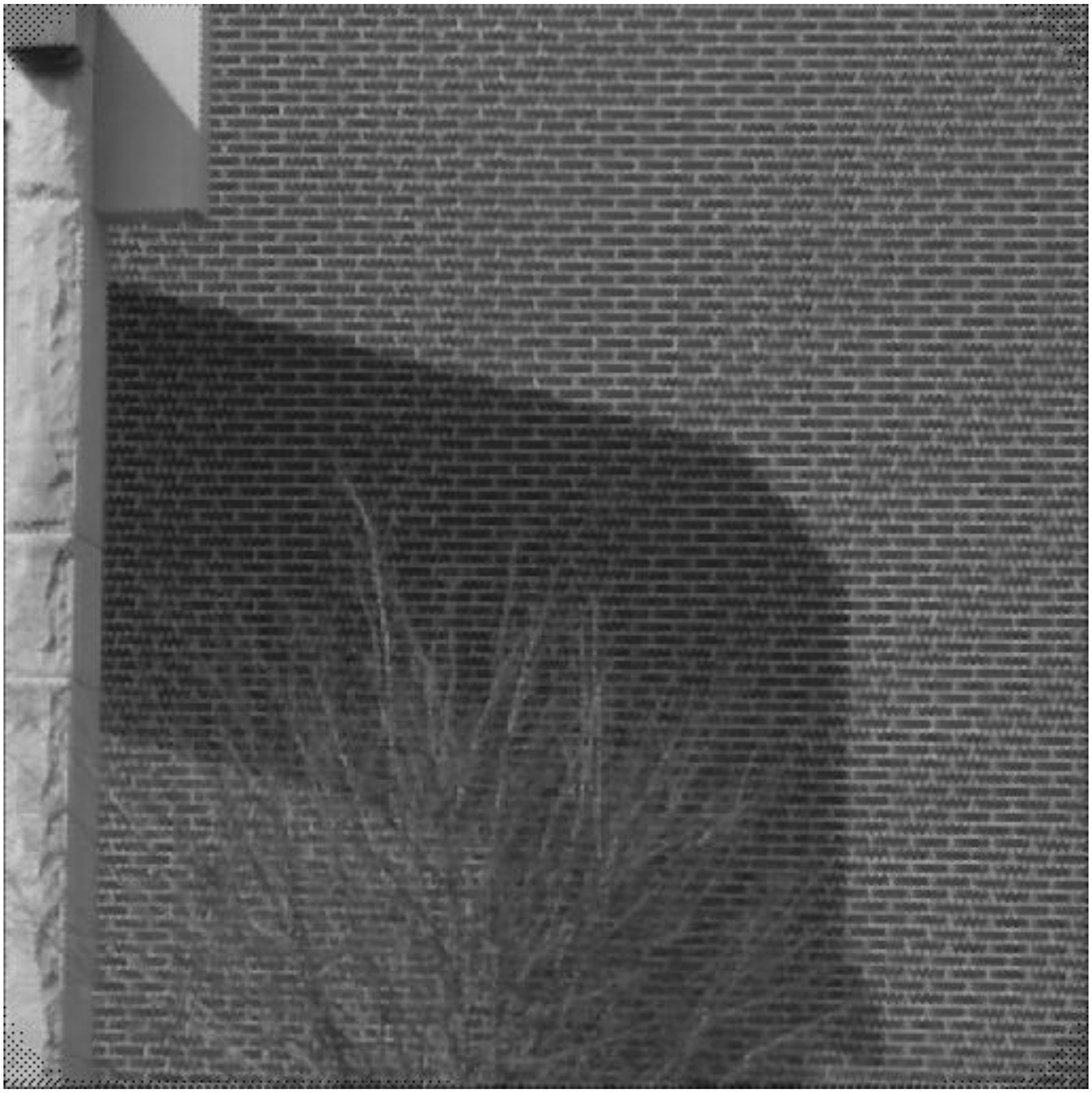


Figure 38 – Subjective Best HR Image in the Brick test (Gate Value = 0.35) (scaled 80%)

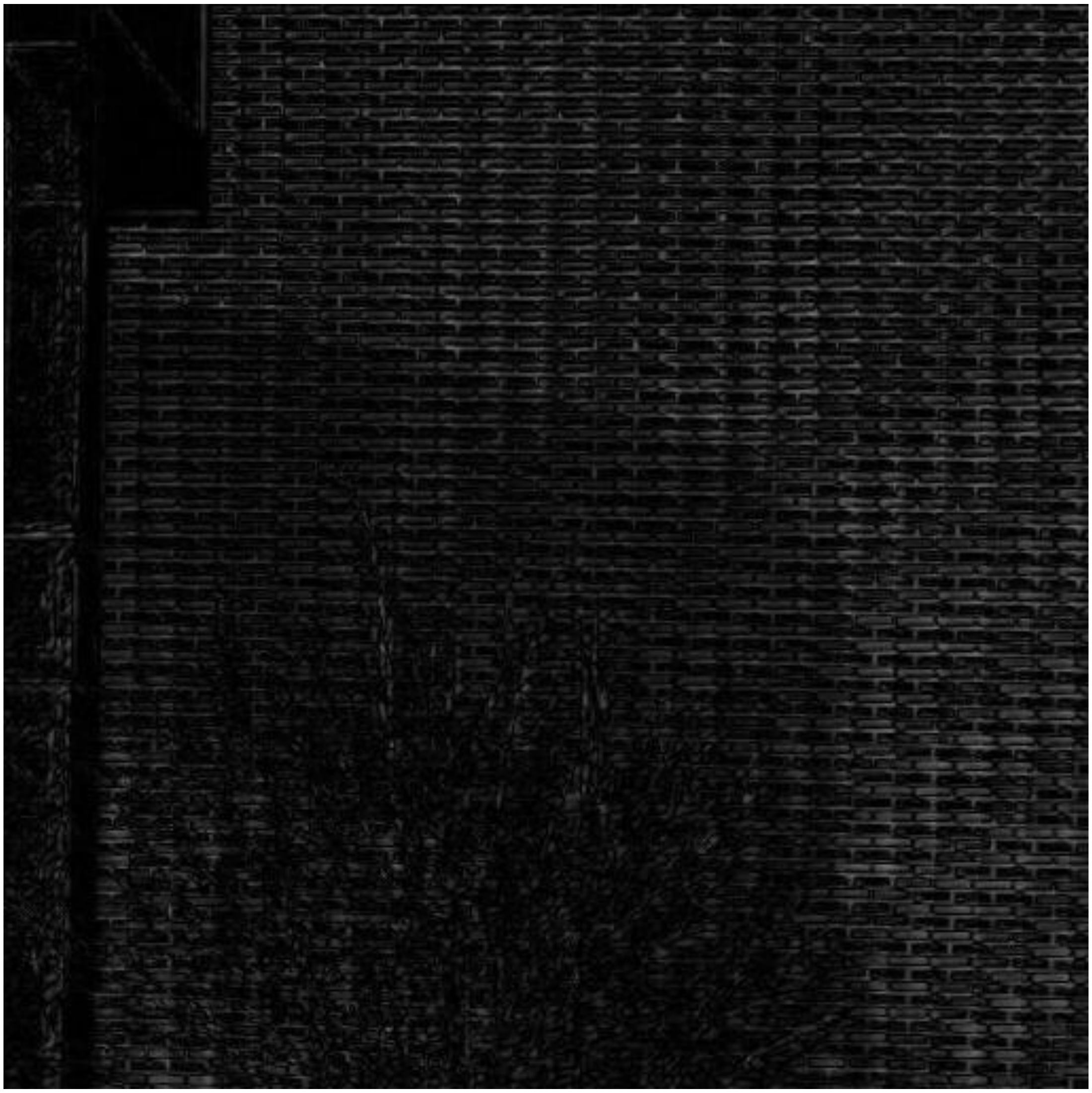


Figure 29 – Scaled Pixel Error for the interpolated LR image in the Brick test. Higher intensities (white) demonstrate greatest error. (Scaled 80%)

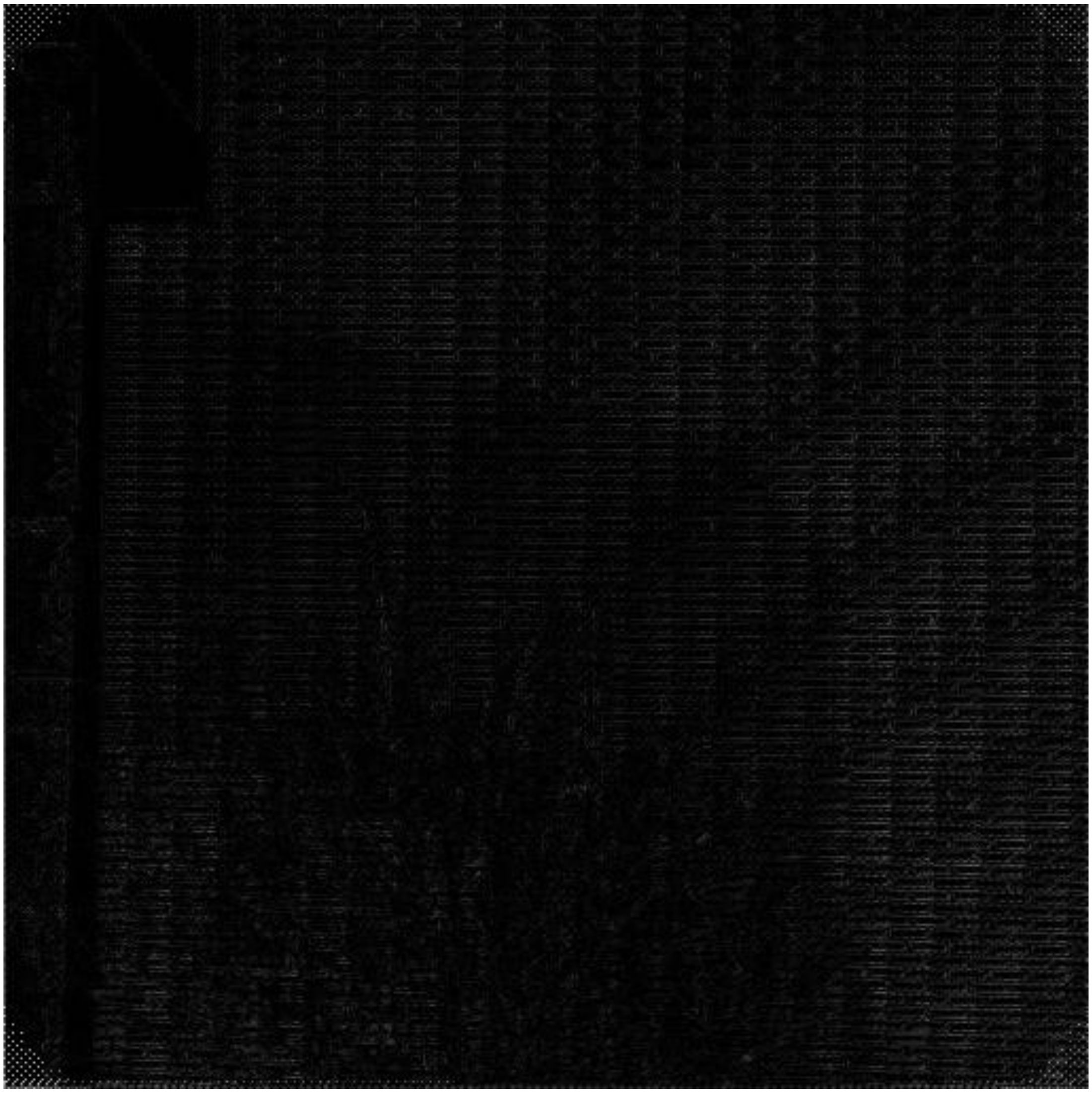


Figure 40 – Scaled Pixel Error for the Best HR image in the BrickTest. (scaled 80%)

3.3f Analysis of Brick test

The Brick test is a great example of aliasing and how SR image reconstruction can lessen aliasing effects. Upon first glance, the interpolated LR image in this case looks like it has been zoomed in to a region of interest. Only after examining the size of the bush and stone column is it apparent that no zooming has occurred, and that there is something clearly wrong with the bricks. The high frequency components of the bricks are lost after the initial downsampling (acqusition). The cubic interpolation of a single LR image cannot recover those frequencies.

This is particularly harmful in this image because of the periodic nature of the bricks. The effect of bricks being aliased with one another creates the illusion of larger bricks, which greatly distorts the structure of the image (confirmed by the MSSIM test). The results of the Brick test are particularly impressive and demonstrate the power of SR image reconstruction.

3.4 Speed

The processing time for the 3 best case gate values is given below. The algorithm was running on in MATLAB for Windows XP on an Intel X86 Dual Core 3.20 GHz Processor with 1.0 GB of RAM. Table 9 lists the results.

Test	# LR Images	LR Dimesion	HR Dimension	Gate Val	Original Time	Optimized Time
Globe	12	42x42	168x168	.56	650 ms	327 ms
Lenna	24	64x64	512x512	.38	10600 ms	4190 ms
Brick	24	128x128	512x512	.35	16420 ms	3988 ms

Table 9 – Processing Time for SR algorithm

The algorithm used for the Globe test processed 12 frames in 327 ms in MATLAB. At this rate, the algorithm can process 1 frame in 27.3 ms, which is faster than standard digital video captured at 30 frames per second, or 33.3 ms per frame, making it capable of real-time video processing. The nature of the algorithm allows it to process 1 frame at a time without the need for a buffer of frame data. Since the HR image is a weighted sum of LR data, and there are no data dependencies between LR frame data, each frame can be processed independently.

3.5 VHDL Model

Hardware platforms like FPGAs store values in fixed point formats. Since floating point values cannot be synthesized to a hardware platform, the algorithm must be converted from a double precision floating point format to an integer format. Converting from integer representation to fixed point representation represents no precision loss, so we can assume the results in fixed point would follow the integer results of MATLAB. However there is significant precision loss when converting from double floating point to integer. An integer-only port of the algorithm was written and tested to determine if the precision losses would be too great. Figure 41 shows the reconstructed image. For a gate value of .5 the MSSIM index was .8108, compared to the interpolated LR sample which was .6151. Fortunately, in this algorithm, quantization errors do not propagate through the system, making it suitable for a hardware implementation.

Figure 41 – Lenna Image computed using only Integer Values. MSSIM > .8

SECTION 4
CONCLUSIONS

4.1 Review

This thesis introduced a computationally efficient SR image reconstruction algorithm suitable for near real-time applications. Without the complexity of matrix inversions, a spatial domain approach using non-uniform interpolation is an ideal SR method for near real time implementation. The results of the algorithm tests were promising, and clearly demonstrated the usefulness of SR image reconstruction. In all cases, the SR image reconstruction algorithm more accurately represented the continuous scene than MATLAB’s cubic interpolation algorithm of a single LR image sample. And since the interpolation stage of the algorithm was linear as opposed to the higher order cubic filter used by MATLAB, it is clear that the SR technique resolves information that mere interpolation cannot. Processing times for the Globe test were less than 1 second, and the Lenna and Brick test completed in under 5 seconds despite processing the data from 24 LR images. The integer-only port of the algorithm received a comparable MSSIM index despite introducing quantization errors from the floating point to integer conversions and subsequent truncations.

4.2 Future Work

The research behind this thesis often posed more questions than delivered answers. Along the way, I discovered areas that will require more research. These are:

1. A formula for determining appropriate gate sizes for the weighted interpolation algorithm.
2. A method for intelligently choosing sub-pixel motion between frames.
3. A pipelined implementation of the algorithm since there are no dependencies in the application

4.2a Gate Sizing

For this thesis, I determined the appropriate gate sizes to use by profiling the test for different values of the gate. This is not practical for on-the-fly processing. One possible solution would be a non-uniform cubic spline or other higher-order interpolation following the registration of images. This would be a better filter in the frequency domain for the fusion of registered LR data. Another approach would be to research the effects that different distributions of LR data have on various gate sizes. As an example, the optimal gate sizes were different between the Lenna test and the Brick test despite the distributions of LR data being the same. Finally, insight into gate sizing based on the frequency components of the scene could be helpful in choosing the best gate.

4.2b Sub-Pixel Motion Parameters

With techniques designed to generate precise sub-pixel offsets between image samples, a study into the optimal distribution of LR data onto a registration grid would be very useful.

4.3c Parallel Processing

Dinechin, *et al.* studied the feasibility mixed-mode fixed point/floating point computations on an FPGA [20]. They discussed the problems of using micro-processor-optimized floating point standards and argued for a more flexible standard capable of taking advantage of the massive parallelism of FPGAs. Since this application has no data dependencies during execution, it is highly optimized for a parallel implementation, using fixed point operations for the additions, subtractions, and multiplications, and replacing a cordic divide with a floating-point division when dividing the pixel values by the weighting coefficients.

APPENDIX

MATLAB CODE

Included are some various MATLAB code segments written for this research. The first segment is an optimized sample algorithm that resolves 24 LR images from the Lenna test.

```
% ------------------------------------------------------------------------
% -- Company: WSU Dept of Electrical Engineering
% -- Engineer: Thomas Pestak
% --
% -- Create Date:    Q1 2010
% -- Design Name:    Super Resolution Image Reconstruction - Matlab Algrthm
% -- Module Name:    optimalMakeSuperResImg_Lenna_power2_512_24images.m
% -- Project Name:   Real Time SR Image Reconstruction
% -- Target Device:  xilinx virtex2
% -- Description:    This program is the SR image reconstruction algorithm
%                    for implementing on an FPGA as part of the Engineer's
%                    Master's Thesis.
% ------------------------------------------------------------------------

% -- This block sets up performance profiling by varying gate sizes.

% -- Objects
% -- trial : incremented for report files to distinguish trials of algrthm
% -- loopstart : used to set initial gateVal for profiling
% -- loopend : used to set final gateVal for profiling
% -- loopincrement : loop incrementor
% -- mean_pxl_vals_store : stores the avg pixel diff btwn scene(HR) & SR
% -- errImg_store : stores Mean Squared Error between HR & SR
% -- maxdiff_pixel_SR_store : stores the max single pxl error btwn HR & SR
% -- correct_pixels_SR_store : stores the # of correct pxls btwn HR & SR
% -- inc : integer incrementor for storing into appropriate indices
% -- gateVal : the "sector" size of Nearest Neighbor Interp
trial = 1;
loopstart = .4;
loopend = .44;
loopincrement = .02;
mean_pxl_vals_store = zeros(((loopend - loopstart)/loopincrement +1 ),2);
errImg_store = zeros(((loopend - loopstart)/loopincrement + 1),2);
maxdiff_pixel_SR_store = zeros(((loopend - loopstart)/loopincrement + 1),2);
correct_pixels_SR_store = zeros(((loopend - loopstart)/loopincrement + 1),2);
```

```
inc = 1;  %incrementor

% Begin Profiling
for gateVal = loopstart:loopincrement:loopend
%gateVal = .67; %gate value used to select which values to interpolate
%about integer location
mean_pxl_vals_store(inc,1) = gateVal;
errImg_store(inc,1) = gateVal;
maxdiff_pixel_SR_store(inc,1) = gateVal;
correct_pixels_SR_store(inc,1) = gateVal;
% -- Objects
% -- m is the length of the LR frame in the x dimension
% -- n is the length of the LR frame in the y dimension
% -- imgdim is the length of the SR image in the x/y dimension
% -- scaleX/scaleY are the scaling factors
% -- newM/newN are the SR img dimensions
% -- sceneImg is the continuous scene to be sampled
% -- newImg is the SR image to be reconstructed
m = 64;
n = 64;
imgdimx = 512;
imgdimy = 512;
scaleX = imgdimx/m; %power 2
scaleY = imgdimy/n;
newM = scaleX * m; %8 X Resolution
newN = scaleY * n;
sceneImg = double(imread('CONTINUOUS\lena1_gs.tif')); %emulating analog scene used to test SR
against later
newImg = zeros(newM,newN); %HR Grid with dimesions 512x512

% -- Objects
% -- img1-24 are the LR sub-pixel shifted images used as input
img1 = double(imread('LR_LENNA_24\lena1_gs_LR_1.tif'));
img2 = double(imread('LR_LENNA_24\lena1_gs_LR_2.tif'));
img3 = double(imread('LR_LENNA_24\lena1_gs_LR_3.tif'));
img4 = double(imread('LR_LENNA_24\lena1_gs_LR_4.tif'));
img5 = double(imread('LR_LENNA_24\lena1_gs_LR_5.tif'));
img6 = double(imread('LR_LENNA_24\lena1_gs_LR_6.tif'));
img7 = double(imread('LR_LENNA_24\lena1_gs_LR_7.tif'));
img8 = double(imread('LR_LENNA_24\lena1_gs_LR_8.tif'));
img9 = double(imread('LR_LENNA_24\lena1_gs_LR_9.tif'));
img10 = double(imread('LR_LENNA_24\lena1_gs_LR_10.tif'));
img11 = double(imread('LR_LENNA_24\lena1_gs_LR_11.tif'));
img12 = double(imread('LR_LENNA_24\lena1_gs_LR_12.tif'));
img13 = double(imread('LR_LENNA_24\lena1_gs_LR_13.tif'));
img14 = double(imread('LR_LENNA_24\lena1_gs_LR_14.tif'));
img15 = double(imread('LR_LENNA_24\lena1_gs_LR_15.tif'));
img16 = double(imread('LR_LENNA_24\lena1_gs_LR_16.tif'));
img17 = double(imread('LR_LENNA_24\lena1_gs_LR_17.tif'));
img18 = double(imread('LR_LENNA_24\lena1_gs_LR_18.tif'));
img19 = double(imread('LR_LENNA_24\lena1_gs_LR_19.tif'));
img20 = double(imread('LR_LENNA_24\lena1_gs_LR_20.tif'));
img21 = double(imread('LR_LENNA_24\lena1_gs_LR_21.tif'));
img22 = double(imread('LR_LENNA_24\lena1_gs_LR_22.tif'));
img23 = double(imread('LR_LENNA_24\lena1_gs_LR_23.tif'));
```

```
img24 = double(imread('LR_LENNA_24\lena1_gs_LR_24.tif'));

%*************************************
tic
% -- Objects
% -- imgs is a matrix of img1-24
%stores 64x64 LR imgs into one 64x64x8 matrix of doubles
imgs(:,:,1) = img1;
imgs(:,:,2) = img2;
imgs(:,:,3) = img3;
imgs(:,:,4) = img4;
imgs(:,:,5) = img5;
imgs(:,:,6) = img6;
imgs(:,:,7) = img7;
imgs(:,:,8) = img8;
imgs(:,:,9) = img9;
imgs(:,:,10) = img10;
imgs(:,:,11) = img11;
imgs(:,:,12) = img12;
imgs(:,:,13) = img13;
imgs(:,:,14) = img14;
imgs(:,:,15) = img15;
imgs(:,:,16) = img16;
imgs(:,:,17) = img17;
imgs(:,:,18) = img18;
imgs(:,:,19) = img19;
imgs(:,:,20) = img20;
imgs(:,:,21) = img21;
imgs(:,:,22) = img22;
imgs(:,:,23) = img23;
imgs(:,:,24) = img24;
% -- Objects
% -- xNew/yNew are SR grids with scaling so that imgdim grid points appear
% -- through 0 to m.

xNew = (0 : (newM - 1)) * (m-1) / (newM - 1); %HR grid scaled to fractional LR
yNew = (0 : (newN - 1)) * (n-1) / (newN - 1); %grid with 512x512 locations scaled to 64x64 fp

%pre-defined translational motion between images
xref = 0 : (m - 1);
yref = 0 : (n - 1);

x1 = xref + .1;
x2 = xref + .9;
x3 = xref + .9;
x4 = xref + .7;
x5 = xref + .7;
x6 = xref + .3;
x7 = xref + .9;
x8 = xref + .9;
x9 = xref + .1;
x10 = xref + .3;
x11 = xref + .7;
x12 = xref + .7;
x13 = xref + -.1;
```

```
x14 = xref + -.9;
x15 = xref + -.9;
x16 = xref + -.7;
x17 = xref + -.7;
x18 = xref + -.3;
x19 = xref + -.9;
x20 = xref + -.9;
x21 = xref + -.1;
x22 = xref + -.3;
x23 = xref + -.7;
x24 = xref + -.7;

y1 = yref + .9;
y2 = yref + .9;
y3 = yref + .1;
y4 = yref + .3;
y5 = yref + .7;
y6 = yref + .7;
y7 = yref + -.1;
y8 = yref + -.9;
y9 = yref + -.9;
y10 = yref + -.7;
y11 = yref + -.3;
y12 = yref + -.7;
y13 = yref + -.9;
y14 = yref + -.9;
y15 = yref + -.1;
y16 = yref + -.3;
y17 = yref + -.7;
y18 = yref + -.7;
y19 = yref + .1;
y20 = yref + .9;
y21 = yref + .9;
y22 = yref + .7;
y23 = yref + .7;
y24 = yref + .3;

% -- Objects
% -- numImgs is number of LR images
% -- x is matrix of x LR data locations
% -- y is matrix of y LR data locations
% -- gateValSquared is square of gateVal to prevent against any SQRTs
% -- weightMat stores the total weight for each interpolated SR point
% -- curweightMat holds the current weight so it can be added to weightMat
% -- loop_check_... are for debugging only
% -- xDist_check is for determining dynamic range of xDist for HDL
% -- considerations
numImgs = 24;
x = [x1; x2; x3; x4; x5; x6; x7; x8; x9; x10; x11; x12; x13; x14; x15; x16; x17; x18; x19; x20; x21; x22;
x23; x24]; %offsets in columns, each image different row 24x64 for x
y = [y1; y2; y3; y4; y5; y6; y7; y8; y9; y10; y11; y12; y13; y14; y15; y16; y17; y18; y19; y20; y21; y22;
y23; y24]; %24x64 for y.

gateValSquared = gateVal^2;
```

```
weightMat = zeros(newM, newN); %512x512 matrix used to store the weights of distances between
data points and integer pixel locations
curweightMat = zeros(newM, newN);
loop_check_mm = 0;
loop_check_ii = 0;
loop_check_kk = 0;
loop_check_jj = 0;
loop_check_ll = 0;
loop_check_i  = 0;
loop_check_j  = 0;
xDist_check = 10;

% -- this is the main SR algorithm
% --
for mm = 1 : numImgs %24 images
    loop_check_mm = loop_check_mm +1;
    for ii = 1:length(xNew) %newM % 1:512 %Must interpolate every SR pixel
        loop_check_ii = loop_check_ii +1;
        kkStart = 1;
        kkEnd = length(x1);
        % for every xLR location (0:63 +offset) compare to every xSR grid
        % location (0:63 scaled) to see if within gate
        for kk = kkStart : kkEnd %m %1 to 64
            loop_check_kk = loop_check_kk +1;
            xDist = (xNew(ii) - x(mm, kk))^2; %change x(lr img 0 to 7, x_pixel address in range 0.0 to
63.0)
            %the above line takes the LOCATION of x(1, kk), in this case the location x = 0 when kk =
1, subtracts it
            %from the location of xNew(1) (the HR grid) in this case x = 0,
            %in order to see if the location of x(1,kk) (the ACTUAL
            %location of a LR pixel point) is close enough to the
            %corresponding HR grid location to be within the "gate" in
            %other words, to be USED in the interpolation.
             if xDist > xDist_check
                xDist_check = xDist;
                q = xNew(ii);
                w = x(mm,kk);
             end %-- the above if statement code is for finding dynamic
                 %-- range of xDist
             if gateValSquared > xDist
             %if the x location of a LR pixel point isn't within the gate,
             %no sense worrying about the corresponding y location of the same LR pixel point
            for jj = 1:length(yNew) %newN
                %exhaust through all yLR locations(0:63+offset) compare to
                %every ySR grid location (0:63 scaled) to see if within
                %gate
                loop_check_jj = loop_check_jj +1;
                temp = floor((jj-1)/scaleY - gateVal);
                if temp < 1
                    llStart = 1;
                    %6355 operations
                else
                    llStart = temp;
                    %535855 operations
                end
                %the following is just a windowing so that not all 64 y LR
```

```
%addresses are checked against the HR grid.

%the point is to determine where to start and end checking
%the y location of LR pixels.  Lowest starting place
%obviously 1, highest 64 (since this is scaled)
temp = ceil((jj-1)/scaleY + gateVal);
if temp > length(y1)
    llEnd = length(y1);
else
    llEnd = temp;
end
weight = 0;
%Check not 1:64, but maybe 1:3 (for trying to resolve SR
%grid point (x,2.1234) instead of wasting time checking 64
%diff addresses, check those around in the vincinity.
%Remember they will be off by a sub-pixel, ie, y(mm,32)
%will be somewhere between y coordinates 31 and 33.  Window
%is based on current euclidean location of yNew(jj) which
%is somewhere between 1:64.  Scaling function /ScaleY figures out
%appropriate euclidean location.
for ll = llStart : llEnd %n
    loop_check_ll = loop_check_ll +1;

    dist = xDist + (yNew(jj)-y(mm,ll))^2;
    %xDist hasn't changed, check location of LR data point
    %y(mm,#) against SR grid location yNew(jj). ex.  Check
    %31.9 (LR data location from sub-pixel shift of +.9
    %against SR grid location 32.1879 to see if it falls
    %within gateVal sector.

    %dist = x_diff^2 + y_diff^2
    %if dist is outside sector, LR data point location
    %wasn't within Y sector, don't include
    %if dist is within sector, now you have both the x,y
    %coordinates of particular LR data point that can be
    %included into interpolation of some SR point.  ex.  if
    %x(mm,2) was 1.1 and y(mm,7)was 6.9, that LR data point
    %would definitely be in the sector for xNew(10)==1.1272
    %and yNew(56)==6.8885
    if gateValSquared > dist
        %this way no square root need
        curWeight = gateValSquared - dist;
        %for values essentially ON the gate they have no
        %weight, the further from the gate (closer to the
        %integer pixel location) the heavier the weight
        weightMat(ii,jj) = weightMat(ii,jj) + curWeight;
        %curweightMat(ii,jj) = curWeight; --range check
        %weight matrix has weights corresponding to all
        %HR pixel locations.
        newImg(ii,jj) = newImg(ii,jj) + ...
            curWeight * imgs(kk,ll,mm);
        %HR_pix_value is LINEAR sum of
        %weights*LR_pix_values
    end
```

```
                end
            end
            end



        end
    end
end
%the code below merely divides every pixel value of the Resolved image with
%the total weight of it's pixels to compute the appropriate grayscale
%intensity
for ii = 1 : newM
    loop_check_i = loop_check_i +1;
    for jj = 1 : newN
        loop_check_j = loop_check_j +1;
        if weightMat(ii,jj) > 0
            newImg(ii,jj) = newImg(ii,jj) / weightMat(ii,jj);
        end
    end
end

% rounds pixel values to integer values
 newImg = uint8(newImg);
 newImg = double(newImg);


toc
% if weight > 0
%                   newImg(ii,jj) = newImg(ii,jj) / weight;
% end
s = sprintf(['SR_Lenna_' num2str(trial) '_' num2str(numImgs) '_' num2str(gateVal*100)]);
imwrite((newImg./255),  ['HR_LENNA_24\' s '.tif'], 'tif');
%imwrite(newImg, 'HR_LENNA\SR_Lenna_' num2str(numImgs) '_' num2str(gateVal*100) '.tif');

figure, imagesc(newImg), colormap(gray), axis image %Super-Resolved Image
title('SR Image Displayed Using imagesc(SR)')
figure, imshow(uint8(newImg))                        %Displayed using imshow
title('SR Image Displayed Using imshow(uint8(SR))')
%Prepare Report CSV file
report_file = fopen(['HR_LENNA_24\SR_report_' s '.txt'], 'w');
fprintf(report_file, ['\nTrial,' num2str(trial) '\n']);
fprintf(report_file, ['Filename,' s '.tif\n']);
fprintf(report_file, 'SizeHR,256x256\n');
fprintf(report_file, ['SizeLR,' num2str(m) '\n']);
fprintf(report_file, ['#LR,' num2str(numImgs) '\n']);
fprintf(report_file, ['Gate,' num2str(gateVal) '\n']);
fprintf(report_file, 'Super-Resolution Stats\n');

%***********************************
%Statistics for Super-Resolved Image

diffImg = abs(sceneImg - newImg); %both images have pixel values of DOUBLE here

%Mean Pixel Difference SR
avgdiff_pixel_SR = mean(diffImg(:));
```

```
mean_pxl_vals_store(inc,2) = avgdiff_pixel_SR;
fprintf(report_file, ['Mean pxl diff,' num2str(avgdiff_pixel_SR) '\n']);

%Maximum Pixel Difference SR
maxdiff_pixel_SR = max(diffImg(:));
maxdiff_pixel_SR_store(inc,2) = maxdiff_pixel_SR;
fprintf(report_file, ['Max pxl diff,' num2str(maxdiff_pixel_SR) '\n']);

%Number of Correct Pixels SR(8 bit accuracy)
iszeros = (diffImg(:) == 0);
correct_pixels_SR = sum(iszeros);
correct_pixels_SR_store(inc,2) = correct_pixels_SR;
fprintf(report_file, ['Correct pxls,' num2str(correct_pixels_SR) '\n']);
figure, imagesc(abs(sceneImg - newImg)./255), colormap gray, axis image
xlabel('White = Most Incorrect Pixel')
ylabel('Black = Most Correct Pixel')
title('Correct Pixels between SR and HR')
%Sum of the difference of all Pixel Values SR
diffImg = sum(diffImg(:)); %sum of difference in 8-bit pxl values
fprintf(report_file, ['Sum pxl diff,' num2str(diffImg) '\n']);

%MSE (Mean Squared Error) SR
errImg = sceneImg(6:506,6:506) - newImg(6:506,6:506); %ignore borders
errImg = errImg(:);
errImg = errImg' * errImg
errImg_store(inc,2) = errImg;
fprintf(report_file, ['MSE,' num2str(errImg) '\n']);
%newImg = zeros(newM,newN);
inc = inc + 1; %extra loop incrementor
trial = trial + 1;
fclose(report_file);
end %END SIMULATION LOOP
%end stats SR

%***********************************
%Statistics for Interpolated Low-Resolved Image (interpolated to 256x256)
report_fileLR = fopen(['HR_LENNA_24\LR_report_' s '.txt'], 'w');

fprintf(report_fileLR, '\nInterpolation Stats\n');

[xLow, yLow] = meshgrid(0 : 63);
[xHigh, yHigh] = meshgrid((0 : 511) / 511 * 63);
LRImg = interp2(xLow, yLow, img1, xHigh, yHigh);    %interpolated here
LRImg = uint8(LRImg);                               %remove decimal precision
LRImg = double(LRImg);
imwrite(LRImg./255, 'LR_Lenna_interpolated.tif', 'tif');
figure, imagesc(LRImg), colormap(gray), axis image
title('LR Image')

%Mean Pixel Difference LR
diffLRImg = abs(LRImg(:) - sceneImg(:));
avgdiff_pixel_LR = mean(diffLRImg(:));
fprintf(report_fileLR, ['Mean pxl diff,' num2str(avgdiff_pixel_LR) '\n']);
```

```
%Maximum Pixel Difference LR
maxdiff_pixel_LR = max(diffLRImg(:));
fprintf(report_fileLR, ['Max pxl diff,' num2str(maxdiff_pixel_LR) '\n']);

%Number of Correct Pixels LR(8 bit accuracy)
iszerosLR = (diffLRImg(:) == 0);
correct_pixels_LR = sum(iszerosLR);
fprintf(report_fileLR, ['Correct pxls,' num2str(correct_pixels_LR) '\n']);
figure, imagesc(abs(LRImg - newImg)./255), colormap gray, axis image
xlabel('White = Most Inaccurate Pixel')
ylabel('Black = Most Accurate Pixel')
title('Correct Pixels between LR and HR')

%Sum of the difference of all Pixel Values LR
diffLRImg = sum(diffLRImg(:));
fprintf(report_fileLR, ['Sum pxl diff,' num2str(diffLRImg) '\n']);

%MSE (Mean Squared Error) LR
errLR = LRImg(6:506,6:506) - sceneImg(6:506,6:506); %ignore borders
errLR = errLR(:);
errLR = errLR' * errLR
fprintf(report_fileLR, ['MSE,' num2str(errLR) '\n']);
fclose(report_fileLR);
%end stats LR
%**********************************

[val, ind] = min(mean_pxl_vals_store(:,2));
figure, stem(mean_pxl_vals_store(:,1), mean_pxl_vals_store(:,2)), hold on
stem(mean_pxl_vals_store(ind,1), val, 'fill') %fills in optimum gateVal
xlabel('GateValue')
ylabel('Average Pixel Difference (8-bit precision)')
title('\Delta Pixel Values')

[val1, ind1] = min(errImg_store(:,2));
figure, stem(errImg_store(:,1), errImg_store(:,2)), hold on
stem(mean_pxl_vals_store(ind1,1), val1, 'fill') %fills in optimum gateVal
xlabel('GateValue')
ylabel('Mean Squared Error')
title('MSE')

[val2, ind2] = max(correct_pixels_SR_store(:,2));
figure, stem(correct_pixels_SR_store(:,1), correct_pixels_SR_store(:,2)), hold on
stem(correct_pixels_SR_store(ind2,1), val2, 'fill') %fills in optimum gateVal
xlabel('GateValue')
ylabel('Number of Correct Pixels')
title('\Delta Pixel Values')

[val3, ind3] = min(maxdiff_pixel_SR_store(:,2));
figure, stem(maxdiff_pixel_SR_store(:,1), maxdiff_pixel_SR_store(:,2)), hold on
stem(maxdiff_pixel_SR_store(ind3,1), val3, 'fill') %fills in optimum gateVal
xlabel('GateValue')
ylabel('Max Pixel \Delta')
title('Maximum Pixel Difference')
```

VHDL CODE

Included are some various logic blocks written in VHDL for this research. This code uses floating point values and was used only for a behavioral simulation of the various parts of an SR algorithm.

The first segment is an SR package developed for HDL implementation of an SR algorithm.

```
--------------------------------------------------------------------------------------------------------

package SR is
constant width_g : Natural :=63; --width(x)of LR samples
constantdepth_g : Natural :=63; --depth(y)of LR samples
constant num_imgs: Natural :=7;  --number of images minus 1
constant width_r : Natural :=255;--width(x) of HR grid
constant depth_r : Natural :=255;--depth(y) of HR grid
constant smallGrid_bits : Natural :=6; --bits used for LR grid
constant largeGrid_bits : Natural :=8 –bits used for HR grid
subtype y_dim_LR  is Integer range 0 to depth_g; --address range
subtype x_dim_LR  is Integer range 0 to width_g;
subtype  num_imgs_range is Integer range 0 to num_imgs;
subtype x_dim_HR        is Integer range 0 to width_r;
subtype y_dim_HR        is Integer range 0 to depth_r;
subtype data_typ_PIXEL is real range 0.0 to 255.0;--possible pixel (gray) values
subtype offsets              is real range 0.0 to 1.0;   --possible subpixel offset values
subtype addr_typ_LR          is real range 0.0 to 64.0; --possible LR address values
subtype  weights             is real range 0.0 to 2.0;  --possible values for weights of weight_matrix
typex_offsets           is array(0 to num_imgs) of offsets;
typey_offsets           is array(0 to num_imgs) of offsets;
type      row_img_typ          is array(x_dim_LR)       of data_typ_PIXEL; --array of PIXELS
type      col_img_typ     is array(y_dim_LR)       of row_img_typ;    --array of ROWS! (image)
typeLR_SEQ                is array(0 to num_imgs) of   col_img_typ;    --array of images!
typeaddresses           is array(y_dim_LR)       of addr_typ_LR;
typeaddr_matrix       is array(num_imgs_range)of addresses;
typeaddr_hr             is array(x_dim_HR)       of addr_typ_LR; --256 points scaled between 0 and 63
typeweight_mat_x    is array(x_dim_HR)       of weights;
typeweight_mat       is array(y_dim_HR)             of weight_mat_x; --256x256 of weights
typegrid_x              is array(x_dim_HR)       of data_typ_PIXEL;
typeSR_IMG                is array(y_dim_HR)       of grid_x;              --256x256 of PIXELS
constant gateval      : real := 0.52;
constant gateval_sqr : real := 0.2704;
constant white  : data_typ_PIXEL := 255.0;
constant black  : data_typ_PIXEL := 0.0;
constant gray   : data_typ_PIXEL := 127.0;
end package SR;
--------------------------------------------------------------------------------------------------------
```

The image_test entity sets up an array of an array for storing the pixel values of the LR frames.

```
-----------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.SR.all;

entity image_test is
    port(   clk          : in bit;
            stimulate    : in bit;
            LR                : out col_img_typ;
            LR_images  : out LR_SEQ);
end image_test;

architecture Behavioral of image_test is
signal   LR_IMAGE : col_img_typ;
signal   LR_sequence : LR_SEQ;
begin --behavioral
process(clk, stimulate)
variable lr_image_v : col_img_typ := (others => (others => white));
variable LR_sequence_v : LR_SEQ := (others => (others => (others => white)));
begin   --process
lr_image_v(0)(0)  := 0.0;   --black
lr_image_v(0)(63) :=   black;
lr_image_v(1)(1)  := 0.0;   --black
lr_image_v(1)(62) := black;
lr_image_v(0)(1 to 62) := (others => gray); --OK
lr_image_v(2)(0 to 63) := (1 to 62 => gray, others => black);
lr_image_v(6)(5 to 61) := (5 to 10 => 0.0, others => 1.0);
lr_image_v(7)(0 to 63) := (0 to 4 => 2.0, 5 to 10 => 0.0, others => 1.0);
lr_image_v(8)(0 to 63) :=  (others => 1.1);
lr_image_v(9 to 63)(9 to 63) := (9 to 15 => (1 to 62 => 8.0, others => 0.0), 16 to 62 => (1 to
62 => 6.0, others => 7.0), others => (0 to 63 => 4.0));
LR_sequence_v(0) := lr_image_v;
LR_IMAGE <= lr_image_v;
LR_sequence <= LR_sequence_v;
end process;
LR <= LR_IMAGE;
LR_images <= LR_sequence;
end Behavioral;
-----------------------------------------------------------------------------------------------------------
package data_types is
   type PIXEL is range 0.0 to 255.0;
     type IMG_LR is array (0 to 63, 0 to 63) of PIXEL;
     type SET_LR is array (0 to 63, 0 to 63, 0 to 7) of PIXEL;
     type counter is range 0 to 7;
end package data_types;
```

The reg entity synchronizes the data dumps from an external frame buffer.

```
--------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity reg is
   Port ( clk : in std_logic;
            f_buffer : in std_logic; --'0' indicates frame buffer empty.
            img_data : in work.data_types.IMG_LR;
        set_data : out work.data_types.SET_LR);
end entity reg;

architecture Behavioral of reg is

begin

process(img_data, clk)
    variable set_data_temp : work.data_types.SET_LR;
    variable img_counter : work.data_types.counter := 0; --this has discrete range 0 to 7
begin
    if img_data'event and f_buffer = '1' then
        set_data_temp(0 to 63, 0 to 63, img_counter) := img_data;
        img_counter := img_counter + 1;
        if img_counter = 8 then  --This implementation uses 8 LR images to achieve SR
            set_data <= set_data_temp;      --finished collecting LR images
            img_counter := 0;    --reset counter for future sequences
        end if;
    end if;
end process;

end Behavioral;
--------------------------------------------------------------------------------------------------------
```

The create_grid entity initializes the Euclidean addresses of the LR frames as well as the registration grid. This is the second stage of the algorithm.

```
--------------------------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.SR.all;

entity create_grid is
   Port ( clk          : in bit;
            delta_x  : in x_offsets;
        delta_y  : in y_offsets;
```

```
        x_address : out addr_matrix;
        y_address : out addr_matrix;
        x_HR_address: out addr_hr;
        y_HR_address : out addr_hr);
end create_grid;

architecture Behavioral of create_grid is
signal x_address_s : addr_matrix;
signal y_address_s : addr_matrix;
signal x_HR_address_s, y_HR_address_s : addr_hr;
begin
process(clk)
variable x_address_v, y_address_v : addr_matrix;
variable x_HR_address_v, y_HR_address_v : addr_hr;
begin
for img in 0 to 7 loop --loops 0 to 7
    for x_ind in 0 to 63 loop  --loops 0 to 63
        x_address_v(img)(x_ind) := real(x_ind) + delta_x(img);
        --the above line takes the index value (0), converts it to real (0.0)
        --and adds the appropriate offset (0.4)
        --and then stores that value into the x_address_matrix
        --it creates an 8x64 matrix of x addresses for all images
    end loop;
    for y_ind in 0 to 63 loop
        y_address_v(img)(y_ind) := real(y_ind) + delta_y(img);
    end loop;
end loop;
for i in 0 to width_r loop
    x_HR_address_v(i) := real(i) * (64.0/255.0);  --change values to constants
end loop;
for i in 0 to depth_r loop
    y_HR_address_v(i) := real(i) * (64.0/255.0);
end loop;

x_HR_address_s <= x_HR_address_v;
y_HR_address_s <= y_HR_address_v;
x_address_s <= x_address_v;
y_address_s <= y_address_v;
end process;
x_HR_address <= x_HR_address_s;
y_HR_address <= y_HR_address_s;
x_address <= x_address_s;
y_address <= y_address_s;
end Behavioral;
----------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
USE work.SR.all;

ENTITY create_grid_tb_vhd IS
END create_grid_tb_vhd;

ARCHITECTURE behavior OF create_grid_tb_vhd IS
```

```
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT create_grid
    PORT(   clk : in bit;
            delta_x  : in x_offsets;
        delta_y  : in y_offsets;
            x_address : out addr_matrix;
            y_address : out addr_matrix;
            x_HR_address: out addr_hr;
            y_HR_address : out addr_hr);
    END COMPONENT;

    --Inputs
    SIGNAL clk_sig :  bit := '0';
    SIGNAL delta_x_sig :  x_offsets;
    SIGNAL delta_y_sig :  y_offsets;

    --Outputs
    SIGNAL x_address_sig :  addr_matrix;
    SIGNAL y_address_sig :  addr_matrix;
    SIGNAL x_HR_address_sig : addr_hr;
    SIGNAL y_HR_address_sig : addr_hr;
BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: create_grid PORT MAP(
        clk => clk_sig,
        delta_x => delta_x_sig,
        delta_y => delta_y_sig,
        x_address => x_address_sig,
        y_address => y_address_sig,
        x_HR_address => x_HR_address_sig,
        y_HR_address => y_HR_address_sig
    );

    tb : PROCESS
    BEGIN

        -- Wait 100 ns for global reset to finish
        wait for 100 ns;
        delta_x_sig <= (0  => 0.0, 1 => 0.4, 2 => 0.2, 3=> 0.1, 4 => 0.8, 5 => 0.5, 6 => 0.3
, 7 => 0.9);
        delta_y_sig <= (0  => 0.0, 1 => 0.4, 2 => 0.2, 3=> 0.1, 4 => 0.8, 5 => 0.5, 6 => 0.3
, 7 => 0.9);
        clk_sig <= '1';
        wait for 1000 ns;
        clk_sig <= '0';
        wait for 10000 ns;
        assert (false) report "sim done :)" severity FAILURE;
        -- Place stimulus here

        wait; -- will wait forever
    END PROCESS;

END;
------------------------------------------------------------------------------------------------------
```

The to_fixed_point entity converts from integer to synthesizable fixed point standard.

```
-------------------------------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

entity to_fixed_point is
port (x, y : in integer range 0 to 255;
          z_int : out integer range 0 to 255;
          z_slv : out std_ulogic_vector(7 downto 0));

end to_fixed_point;

architecture Behavioral of to_fixed_point is
signal x_sig, y_sig, z_int_sig: integer range 0 to 255;
signal z_slv_sig : std_ulogic_vector(7 downto 0);

begin
     x_sig <= 100;
     y_sig <= 77;
     z_int_sig <= x_sig + y_sig; --integers
     z_int <= z_int_sig;
     z_slv <= z_slv_sig;
     behavior: process(x,y)
          variable temp : integer range 0 to 255;
          variable result : std_ulogic_vector(7 downto 0);
          begin
          temp := x_sig + y_sig;
          temp := temp + y_sig; --should be 30 at this point
          --this code converts integers to std_ulogic_vector
          for index in result'reverse_range loop
               result(index):= to_X01(bit'val(temp rem 2));
               temp := temp/2;
               exit when temp = 0;
          end loop;
          result(0) := '1';
          z_slv_sig <= result;
      end process behavior;
end Behavioral;

-------------------------------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY to_fixed_point_tb_vhd IS
END to_fixed_point_tb_vhd;

 ARCHITECTURE behavior OF to_fixed_point_tb_vhd IS
```

```
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT to_fixed_point
    port (x, y : in integer range 0 to 255;
        z_int : out integer range 0 to 255;
        z_slv : out std_ulogic_vector(7 downto 0));

    END COMPONENT;

    --Inputs
    SIGNAL x_tb :  integer range 0 to 255;
    SIGNAL y_tb :  integer range 0 to 255;

    --Outputs
    SIGNAL z_int_tb :  integer range 0 to 255;
    SIGNAL z_slv_tb :  std_ulogic_vector(7 downto 0);
BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: to_fixed_point PORT MAP(
        x => x_tb,
        y => y_tb,
        z_int => z_int_tb,
        z_slv => z_slv_tb
    );

    tb : PROCESS
    BEGIN

        -- Wait 100 ns for global reset to finish
        wait for 100 ns;
        x_tb <= 1;
        y_tb <= 2;
        wait for 100 ns;
        x_tb <= 2;
        y_tb <= 1;
        wait for 100 ns;
        -- Place stimulus here

        assert (false) report "sim done :)" severity FAILURE;
    END PROCESS;

END;
--------------------------------------------------------------------------------------------------------
```

REFERENCES

[1] M.S. Alam, J.G. Bognar, R.C. Hardie, and B.J. Yasuda. Infrared image registration and high-resolution reconstruction using multiple translationally shifted aliased video frames. *IEEE Transactions on instrumentation and measurement*, 49(5):915–923, 2000.

[2] S. Borman and R. Stevenson. Spatial resolution enhancement of low-resolution image sequences-a comprehensive review with directions for future research. *University of Notre Dame*, Tech. Rep, 1998.

[3] A.E. Burgess. The Rose model, revisited. *Journal of the Optical Society of America A*, 16(3):633–646, 1999.

[4] S. Chaudhuri. Ed. , Super-*Resolution Imaging*. Norwell. MA: Klulwer, 2001.

[5] J. Clark, M. Palmer, and P. Lawrence. A transformation method for the reconstruction of functions from nonuniformly spaced samples. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 33(5):1151–1165, 1985.

[6] M. Elad and A. Feuer. Superresolution restoration of an image sequence: adaptivefiltering approach. *IEEE Transactions on Image Processing*, 8(3):387–395, 1999.

[7] C.L.L. Hendriks and L.J. van Vliet. Resolution enhancement of a sequence of undersampled shifted images. In *Proc. 5th Annual Conference of the Advanced School for Computing and Imaging (Heijen, NL, June 15-17)*, pages 95–102, 1999.

[8] TS Huang and RY Tsai. Multi-frame image restoration and registration. *Advances in computer vision and Image Processing*, 1(317-339):2, 1984.

[9] M. Irani and S. Peleg. Improving resolution by image registration. *CVGIP: Graphical Models and Image Processing*, 53(3):231–239, 1991.

[10] S. Jahanbin and R. Naething. Super-resolution Image Reconstuction Performance. 2005.

[11] SP Kim, NK Bose, and HM Valenzuela. Recursive reconstruction of high resolution image from noisyundersampled multiframes. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(6):1013–1027, 1990.

[12] S.C. Park, M.K. Park, and M.G. Kang. Super-Resolution Image Reconstruction. *IEEE signal processing magazine*, 2003.

[13] D. Parkinson and H.M. Liddell. The measurement of performance on a highly parallel system. *IEEE Transactions on Computers*, 32(1):32–37,1983.

[14] M. Protter, M. Elad, H. Takeda, and P. Milanfar. Generalizing the non-local-means to super-resolution reconstruction. *IEEE Transactions on Image Processing,* 18(1):36–51, 2009.

[15] J. Shi, S.E. Reichenbach, and J.D. Howe. Small-kernel superresolution methods for microscanning imaging systems. *Applied optics*, 45(6):1203–1214, 2006.

[16] H. Stark and P. Oskoui. High-resolution image recovery from image-plane arrays, using convex projections. *Journal of the Optical Society of America* A, 6(11):1715–1726, 1989.

[17] P. Vandewalle, S. Susstrunk, andM. Vetterli. A frequency domain approach to registration of aliased images with application to super-resolution. *EURASIP Journal on Applied Signal Processing*, 10, 2006.

[18] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600-612, Apr. 2004.

[19] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli , The SSIM Index for Image Quality Assessment.

http://www.ece.uwaterloo.ca/~z70wang/research/ssim/

[20] F. de Dinechin, J. Detrey, I. Trestian, O. Cret¸, and R. Tudoran. When FPGAs are better at floatingpoint Than microprocessors. Technical Report ensl00174627, ´ Ecole Normale Sup´erieure de Lyon, 2007. http://prunel.ccsd.cnrs.fr/ensl-00174627.