Evaluation of Image Warping Algorithms for Implementation in FPGA

Examensarbete utfört i Elektroniksystem vid Tekniska högskolan i Linköping
av

Anton Serguienko

LITH-ISY-EX--08/3993--SE
Linköping 2008
Evaluation of Image Warping Algorithms for Implementation in FPGA

Examensarbete utfört i Elektroniksystem vid Tekniska högskolan i Linköping
av

Anton Serguienko

LITH-ISY-EX--08/3993--SE

Handledare: Kent Palmkvist
isy, Linköpings universitet
Peter Fredriksson
Enea AB

Examinator: Kent Palmkvist
isy, Linköpings universitet

Linköping, 24 February, 2008
Ev aluation of Image Warping Algorithms for Implementation in FPGA

The target of this master thesis is to evaluate the Image Warping technique and propose a possible design for implementation in FPGA. The Image Warping is widely used in the image processing for image correction and rectification. A DSP is a usual choice for implantation of the image processing algorithms, but to decrease a cost of the target system it was proposed to use an FPGA for implementation.

In this work a different Image Warping methods was evaluated in terms of performance, produced image quality, complexity and design size. Also, considering that it is not only Image Warping algorithm which will be implemented on the target system, it was important to estimate a possible memory bandwidth used by the proposed design. The evaluation was done by implemented a C-model of the proposed design with a finite datapath to simulate hardware implementation as close as possible.

Nyckelord  
image processing, FPGA, Image Warping, image interpolation
Abstract

The target of this master thesis is to evaluate the Image Warping technique and propose a possible design for implementation in FPGA. The Image Warping is widely used in the image processing for image correction and rectification. A DSP is a usual choice for implantation of the image processing algorithms, but to decrease a cost of the target system it was proposed to use an FPGA for implementation.

In this work a different Image Warping methods was evaluated in terms of performance, produced image quality, complexity and design size. Also, considering that it is not only Image Warping algorithm which will be implemented on the target system, it was important to estimate a possible memory bandwidth used by the proposed design. The evaluation was done by implemented a C-model of the proposed design with a finite datapath to simulate hardware implementation as close as possible.
## Contents

1 Introduction ........................................ 1
   1.1 Background ........................................ 1
   1.2 Objectives ........................................ 2
   1.3 Limitations ........................................ 2
   1.4 Method ........................................... 2
   1.5 Development environment and target platform ........... 2
   1.6 Thesis Decomposition ............................ 2

2 Theoretical background .......................... 5
   2.1 Image Warping .................................... 5
   2.2 Fixed point mathematic .......................... 9

3 Design considerations and tests description ... 11
   3.1 Proposed design ................................... 11
   3.2 Transformation table compressor design ............... 12
   3.3 Image Cache ....................................... 13
   3.4 Interpolation algorithms .......................... 14
   3.5 Fixed point mathematic .......................... 14

4 Evaluation of algorithms and design .......... 17
   4.1 LUT compression ................................... 17
   4.2 Image Cache ....................................... 18
      4.2.1 Image Cache with fixed cache block position ... 18
      4.2.2 Image Cache with floating cache block position 20
      4.2.3 Image Cache Summary .......................... 20
   4.3 Image interpolation algorithms evaluation .......... 23

5 Conclusions ......................................... 27

Bibliography ........................................ 29

A Abbreviations ...................................... 31

B Graphs ............................................. 32
Chapter 1

Introduction

This section presents a short overview of objectives and background for this master thesis work. Methods and limitations are also mentioned here. At the end of the chapter document decomposition is given.

1.1 Background

The market of programmable logic devices (PLD) is rapidly growing. In comparison with ASICs, PLD allows fast prototyping and easily supports and maintains. Programmable logic is often associated with Field-Programmable Gate Arrays (FPGA) which is a main target in this project.

DSP is commonly used for image processing and is considered the best solution for heavy computation tasks involving floating point computations on large set of data. However for the less computation intensive applications where precision is not crucial FPGA solutions can be used or some preprocessing tasks can be implemented in programmable logic before data is delivered to a DSP. Such approach is especially effective for designs where the FPGA is also used as a microcontroller for DSPs since the controller unit does often not utilize the whole FPGA. In this case, system design will be simpler regarding the number of components on the board, such as memory controllers, preprocessing units and others which can be implemented in FPGA.

In this work a similar approach will be used. A preprocessing unit which corrects images coming from two stereo mounted camera sensors is going to be implemented in FPGA and images then will be delivered to DSPs for further processing. In this thesis different methods of image warping will be evaluated with respect to design size and performance and a design proposal will be presented. On a target platform (see Section 1.5) it is expected to implement other algorithms which are also going to use external memory. Because of it, it is important to evaluate different aspects of the design in order to minimize memory bandwidth and number of accesses to the memory. It is also important to keep design size small so that it is possible to use a smaller FPGA, which is cheaper.
1.2 Objectives

- The main goal of this project is to evaluate different methods of image warping for future implementation in FPGA. Evaluation is targeting resulting image quality, memory bandwidth, design size and performance.

- As a subsequent objective an evaluation of the interpolation algorithms will be performed.

- A secondary objective was to use as much of freely distributed tools as possible.

- A choice between different methods of calculating transformation function for interpolation will be done.

1.3 Limitations

Implementation of the algorithms in the FPGA will not be performed due to time limitations.

1.4 Method

The algorithm evaluation will be performed by constructing a C-model with adjustable finite precision data path. Quality of the images produced by the different interpolation algorithms using different fixed point precision will be compared with the reference algorithm. Memory bandwidth measurement will be performed by comparing different methods of caching of the incoming data with respect to interpolation algorithms.

1.5 Development environment and target platform

A system model is written in C using the GNU gcc compiler and the Eclipse development environment with C/C++ extension. For documentation of C-code the freely available documentation generator Doxygen was used. Matlab was used for the result data processing and visualization.

The possible target platform consists of an Xilinx Spartan3E FPGA and a DDR memory where images for resampling are stored. The camera sensors used produces 720x480 10-bits images which are stored in the DDR memory.

1.6 Thesis Decomposition

This section contains a description of the chapters in the document.

- Chapter 1, this chapter, gives a short introduction to the document and goals of the thesis.
• Chapter 2 describes the theoretical background needed to understand the thesis.

• Chapter 3 contains a system design proposal and a test description.

• Chapter 4 presents the results of the evaluation of the algorithms and the design.

• Appendix A contains explanations of abbreviations used in the text.

• Appendix B contains the graphs which were not included in text.
Chapter 2

Theoretical background

In this chapter the theoretical aspects of image warping and fixed point arithmetic are described which are necessary to know in order to understand the rest of the thesis. The explanation of these terms is brief. For more information on the subject it is recommended to follow references given in the text.

2.1 Image Warping

Image Warping is an image processing technique used for example for correction of lens distortion, rectification of stereo images, image resampling and video stabilization. Due to distortion in lenses, especially cheap ones, images need to be corrected with the help of interpolation locations measured from lens. See Figure 2.1. Mostly this is unnecessary if the images are not intended for the human eye. Other types of distortions can occur because of camera sensor mounting mismatch which might produce rotation, resizing, shifts and other geometrical transformations of images. This kind of distortion is especially important to correct when stereo setup of camera sensors is used and images from both sensors should match each other.

Figure 2.1. Lens Distortion
Image Warping performs in practice in two steps: determining a new location of points, this step is called reconstruction, and resampling of an image into the new coordinates [2]. See Figure 2.2. For the first step there exist many methods mostly based on determining contours or pivot points on the original and the malformed images, determining the correlation of these points between two images and calculating necessary continuous transformation function which could be also a geometrical transformation or some mixture of geometrical transformations. Another method is based on the assumption that the image distortion is constant and can be measured once, thus producing a look-up table with new pixel locations. The look-up table can be used later, during resampling, instead of transformation function.

Resampling of images are performed by interpolation. Interpolation is a mathematical method of reproducing missing values from a discrete set of known values. For image processing it means that with some given samples, interpolation produces missing values between these samples. Many image interpolation algorithms exist and vary in the image quality they produce and the computation cost. Most known of them are the nearest neighbor, bilinear interpolation, quadratic interpolation, cubic interpolation and spline interpolation. All of these algorithms are based on piecewise local polynomials which are easy to implement and evaluate [1]. Other algorithms exists but they tend to be mathematically more complex [5] and therefore hard to implement.

Nearest neighbor is the simplest among of the interpolation algorithms and has lowest computation cost, but it produces strong aliasing artifacts along sharp edges. A new image sample is computed by taking the nearest sample to the new
2.1 Image Warping

one. See Figure 2.3 and Equation 2.1.

\[ O(u, v) = P_{mn} \]
\[ m = \lfloor x + 0.5 \rfloor \quad (0 \geq x \leq 1) \]
\[ n = \lfloor y + 0.5 \rfloor \quad (0 \geq y \leq 1) \]  \hspace{1cm} (2.1)

Figure 2.3. Nearest neighbour Interpolation

Bilinear interpolation is a more advanced interpolation method in which the two nearest known samples are used. A new sample is computed by taking a weighted average of the surrounding samples. This interpolation method gives better image quality than the nearest neighbor, but tends to blur the image. See Figure 2.4 and Equation 2.2

\[ O(u, v) = (1 - \Delta x)(1 - \Delta y)P_{ij} - (1 - \Delta x)\Delta yP_{ij+1} + \Delta x(1 - \Delta y)P_{i+1j} + \Delta x\Delta yP_{i+1j+1} \]  \hspace{1cm} (2.2)

Figure 2.4. Bilinear Interpolation

Quadratic and cubic interpolations use the same technique as a bilinear but takes more surrounding samples - nine for quadratic and sixteen for cubic. For

\hspace{1cm} \footnote{Hereby \( O(u, v) \) and \( P_{mn} \) is a sampling and a neighbor points respectively}
quadratic interpolation the three nearest samples per row are used and for cubic
two sample to the left of the interpolation value and two to the right. For the
first step a weighted average of each of the row is computed and then a weighted
average of produced values is calculated. The samples close to the interpolation
point gets more weight. See Figure 2.5, Equation 2.3 and Figure 2.6, Equation 2.4.

\[
O(u, v) = \left[ \begin{array}{ccc}
  f\left(\frac{1}{2} + \Delta y\right) & f(\Delta y) & f\left(\frac{1}{2} - \Delta y\right)
\end{array} \right] 
\cdot 
\left[ \begin{array}{ccc}
  P_{11} & P_{12} & P_{13} \\
  P_{21} & P_{22} & P_{23} \\
  P_{31} & P_{32} & P_{33}
\end{array} \right] 
\cdot 
\left[ \begin{array}{c}
  f\left(\frac{1}{2} + \Delta x\right) \\
  f(\Delta x) \\
  f\left(\frac{1}{2} - \Delta x\right)
\end{array} \right]
\]

\[
f(s) = \begin{cases}
  -2|s|^2 + 1 & (|s| \leq 1) \\
  |s|^2 - \frac{5}{2}|s| + \frac{3}{2} & \left(\frac{1}{2} < |s| \leq \frac{3}{2}\right) \\
  0 & \left(\frac{3}{2} \leq |s|\right)
\end{cases}
\]

(2.3)

\[
O(u, v) = \left[ \begin{array}{cccc}
  f(1 + \Delta y) & f(\Delta y) & f(1 - \Delta y) & f(2 - \Delta y)
\end{array} \right] 
\cdot 
\left[ \begin{array}{cccc}
  P_{11} & P_{12} & P_{13} & P_{14} \\
  P_{21} & P_{22} & P_{23} & P_{24} \\
  P_{31} & P_{32} & P_{33} & P_{34} \\
  P_{41} & P_{42} & P_{43} & P_{44}
\end{array} \right] 
\cdot 
\left[ \begin{array}{c}
  f(1 + \Delta x) \\
  f(\Delta x) \\
  f(1 - \Delta x) \\
  f(2 - \Delta x)
\end{array} \right]
\]

\[
f(s) = \begin{cases}
  1 - 2|s|^2 + |s|^3 & (0 \leq |s| < 1) \\
  4 - 8|s| + 5|s|^2 + |s|^3 & (1 \leq |s| < 2) \\
  0 & (2 \leq |s|)
\end{cases}
\]

(2.4)

Cubic interpolation requires high computation cost, but generates the best
image quality. Images produced by quadratic interpolation lays somewhere in-
between linear and cubic, in quality approaching later one. The computation
cost for quadratic interpolation is much less than for cubic. Other high order
interpolation methods, such as spline or sinc, produce more accurate results and retain most of image information, but needs heavy calculations which are often unacceptable. Combination of the named interpolation methods are also possible. For example combining quadratic and bilinear interpolation (3 by 2 surrounding samples used) will give more sharp vertical edges and blurry horizontal.

2.2 Fixed point mathematic

Floating point mathematic is not always the best alternative to use in programmable logic devices because of implementation complexity and increased design size. In such case fixed point mathematic is used. The concept of fixed point arithmetic is based on representation of floating point number as an integer where a decimal point imaginary placed between two bits somewhere in integer. Bits to the right of the point represent the fractional part of a value and bits to the left represents the integer part. Because of limited amount of bits in integer variables, fixed point values have limited precision which depends on how many bits are chosen to represent the fractional part. The precision can be calculated by a simple formula:

\[ e = \frac{1}{2^{f_{\text{bits}}}} \]  

(2.5)

where \( f_{\text{bits}} \) is a number of fractional bits and \( e \) is the precision. Fixed point mathematic is the same as for floating point except that if overflow or underflow occurs, fractional or integer part should be truncated or rounded to keep same integer size. For more information on this topic it is recommended to read [4].

Figure 2.6. Cubic Interpolation
Chapter 3

Design considerations and tests description

3.1 Proposed design

The proposed design consists of three main parts: transformation table (hereby referred to as LUT) compression, image cache and interpolation parts. The image data is coming from the camera sensors and stored in memory. General overview of the design can be seen on Figure 3.1. For the reconstruction part of the Image Warping, a solution with LUTs was chosen because of their availability for given sensors configuration and the lower design complexity compared to computing of a reconstruction kernel in real-time. The only drawback with this solution is that it requires additional memory resources to store the LUT inside the FPGA. To reduce internal memory usage, LUT compression was considered which at the same time should not significantly decrease the resulting image quality. LUTs for model testing were generated from the camera model. Image cache was introduced because of the structure of the interpolation algorithms (see Chapter 2.1) where the same sample point is used multiple times. Thus by caching samples, memory accesses and bandwidth could be decreased.

Each of the parts mentioned above is responsible for memory bandwidth and/or image quality. Compression of the LUTs influence image quality and, in some cases, memory bandwidth. The size of the image cache and its organization is the main factor which affects memory bandwidth. The chosen interpolation algorithm affects output image quality and overall performance.

To reduce test time and memory requirements, it was decided to divide test into four categories which will be tested separately. The first set of the tests was designed only for the LUT compression and never included interpolation and image cache modules. The second set of the tests included only the image cache module which was tested on uncompressed LUTs, fully bypassing the compression module and processing no actual image data, only its coordinates. The third set of the tests are designated only to test image quality of different interpolation
algorithms with or without LUT compression. The last set of the tests analyzed overall system performance and functionality.

This model is implemented for 16 bits data per point in original image however results should be applicable for 10 bits data per point.

### 3.2 Transformation table compressor design

In the design it is assumed, that LUTs consist of two values: a difference for X and Y coordinate between point location in the reference image and in the new image. Values are of fixed point type. Based on given LUTs it was found that maximum difference between the reference image and the new image do not exceed 30 points in either direction so it is assumed that it will be enough with 6 integer bits per value. The number of fractional bits depends on how good precision is needed.
Other values can be chosen by measuring displacement in currently used LUT.

The compression algorithm chosen is simple. It computes the mean of the three values along each row in a cell, and then computes the mean of these results. For example to compress a cell of 2 by 2 samples into one, it computes a mean value along each of the lines in a cell and then computes a mean value of the results. When retrieving a value from a compressed LUT no computations are done, i.e. every position in the cell have the same value as in the compressed cell. See Figure 3.2.

Depending on how much the LUT can be compressed without losing too much data, parts of the LUT may be required to be loaded into the internal BRAM during interpolation process if the whole compressed LUT cannot be stored in it.

The test was done by comparing LUTs after different compression with the original uncompressed LUT and computing the average deviation and RMS of deviation in coordinates between the LUTs.

### 3.3 Image Cache

In order to reduce memory bandwidth it was decided to introduce an onboard image cache. It is 2-level cache. The first level is a simple small cache which contains the last n samples used in the interpolation. It resembles a cache used in microprocessors. The second level cache is a FIFO buffer which contains the last n blocks of samples read from memory. The whole image is divided into the blocks of the same size and with a fixed position. A “floating” block structure was also implemented which allows a block to begin at any 32 bit aligned location. A block is defined as a consecutive row of samples in an image. To load the samples located in the same interpolation cell into the cache the following steps are done:

- Because each row in a cell consist of samples located in the same row, the algorithm is looking for the whole row at once. In general all samples which are used during interpolation are added into sample cache if they are not already there.

- First it is determined if one row lays in the same image block and if this block is in memory.
• If the row is in different image blocks and one of the blocks is in memory, the algorithm searches for remaining samples first in the sample cache and then loads a missing image block if not all samples are found in the sample cache.

The algorithm always tries to find samples in block cache first and after it tries to find missing samples in sample cache.

The first level cache, which is also called sample cache in this document, is fully adjustable for test purposes. Both size of a cache and time-to-live (TTL) value for every entry in it are configurable. Second level cache or block cache have adjustable block size and number of blocks. Both caches are configurable during execution, i.e. those parameters can be changed before processing of an image.

The size and the number of blocks saved in the cache are highly dependent on the interpolation algorithm. In general, depending on how much neighbor samples along Y-axis are used at least so much cached blocks should be available. For example if it uses 3 by 3 neighbor points then it is needed at least 3 blocks cached to achieve acceptable memory bandwidth usage. It was also proposed to have as much cache blocks along one image line as the interpolation can use and save a whole line of an image to enhance performance and simplify design. See Section 4.2 for the explanation.

3.4 Interpolation algorithms

The bilinear, quadratic and cubic interpolation algorithms are implemented. In addition to these three, a general interpolation algorithm is implemented which can interpolate on a custom number of neighbor pixels in a range of maximum 4 points per axis. All these algorithms have straight forward implementation without any extensive optimization done. The non-linear response of the sensor was not considered in the model implementation, because it does not influence results of comparison between reference algorithm and evaluated interpolations if they are performed on the same set of data.

3.5 Fixed point mathematic

The multiple fixed point mathematic c-libraries exists but none of them suited purposes of the project. Most of them were part of some larger libraries such as image and video processing and others were not freely distributed. In 2005 ISO released a draft Technical Rapport [3] on a fixed point implementation for the C programming language but until now no major C compiler vendor have implemented it. Hence it was decided to write a new fixed point library. This library includes addition, subtraction, multiplication and shift functions \(^1\). There is a single signed fixed point data type which includes number of integer and

\(^1\)Division is not included because it is not used anywhere in algorithms. Instead of it shift operation is used.
fractional bits. All functions allows control of the word length of the result and automatic rounding or truncating of the result if necessary.

All computations are done with fixed point mathematic to simulate the behavior of algorithms in a FPGA as close as possible. All values have a fixed length. Simulations are done with different word length in different parts of the design to find the acceptable range of word lengths.
Chapter 4

Evaluation of algorithms and design

This section presents the results of algorithm evaluation based on given images and transformation tables. Figures in this section show a pixel difference between images produced by reference interpolation algorithm and proposed interpolation algorithms, except for the figures describing the image cache performance. The graphs are produced by taking a RMS on results from 100 different transformation tables for each input image. For the graphs presenting memory access/bandwidth the results are produced by taking the mean of results from 100 transformation tables.

4.1 LUT compression

Different compression sizes and word lengths have been tested. Tests showed that the best compression (measured in how much it compresses a LUT) which is acceptable to use, is a 7 by 7. It gives acceptable quality results but even in the best case the whole LUT can not be placed into the BRAM of the FPGA so it needs to be partially loaded into the on-board memory. If better compression is needed, other algorithms should be used. Figure 4.1 presents results of this compression. 32 bit word length was used in LUTs.

As can be seen the difference is less than 0.3% (images are 16 bit) and it could be decreased if a LUT word length is increased but it would lead to increased LUT size. More graphs can be found in Section B.
4.2 Image Cache

The maximum number of the cache blocks needed can be calculated from LUT’s. It means that because the interpolation algorithm follows coordinates from the LUTs it could start traverse an image line with sample coordinates in the first row and finish with coordinates on completely different row, located up or down from the beginning row. It is also dependent on which interpolation method is used. Examining LUTs it was found that for 3x2, 2x2 and 4x2 interpolations needs 26 cache blocks and for 3x3 - 28.

Furthermore the order in which an image is read matters. All given 100 test LUT’s have such a structure so that the maximum number of cache blocks needed is larger if image read from top to bottom. That means that all images are more or less rotated clockwise. If an image was rotated counterclockwise then it should be more practical to read it from bottom to top.

The point cache in all test are set to 20 points with beginning TTL value 8. A small increase of these values does not give any effect and decrease gives substantive fall in performance.

4.2.1 Image Cache with fixed cache block position

Different number of cache blocks and sizes of one cache block were tested. See Figure 4.2 and Figure 4.3.
Figure 4.2. Memory accesses for 3x2 interpolation with fixed positioning

![Memory Accesses Graph](image1)

Figure 4.3. Memory bandwidth for 3x2 interpolation with fixed positioning

![Memory Bandwidth Graph](image2)
First Figure 4.2 shows memory accesses and Figure 4.3 shows an actual number of bits read during one 3x2 image interpolation. As shown in Figure 4.3, the memory bandwidth does not decrease if more cache blocks are added into the block cache. In fact it almost does not change because with more cache blocks added the role of the point cache becomes insignificant and values becomes the same as if there were no point cache at all. The same is happening with the total number of memory accesses per one interpolation, except for the case where two cache blocks are used which are not enough to achieve best performance.

The size of the cache block influence the total number of memory accesses by substansially decreasing it but at the same time it increases memory bandwidth. For the case when cache block size is set to the size of one image line, memory bandwidth in some cases became even bigger then if direct memory access was used and drops below it only if maximum number of cache blocks added. But at the same time memory access becomes very low in comparison with other setups.

Figures for other interpolation methods can be found in Appendix B.

4.2.2 Image Cache with floating cache block position
This implementation of image cache gives in overall the same results. Results looks more stairlike (see Figure 4.4) because of aligned position of image blocks. All number of blocks which are dividable with 3 and up to the next divisible number gives almost the same results.

4.2.3 Image Cache Summary
Like it could be seen memory bandwidth almost does not change with increasing number of cache blocks. It could be explain by examining the following graphs: Figure 4.6 and Figure 4.7. These graphs have been generated with the same settings as prevision except for that they have point cache disabled. In this case it is easy to see that memory bandwidth and accesses decreases with increase of number of cache blocks.
Figure 4.4. Memory accesses for 3x2 interpolation with floating positioning

Figure 4.5. Memory bandwidth for 3x2 interpolation with floating positioning
Figure 4.6. Memory accesses for 3x2 interpolation with fixed positioning without point cache

Figure 4.7. Memory bandwidth for 3x2 interpolation with fixed positioning without point cache
4.3 Image interpolation algorithms evaluation

Four main interpolation algorithms were evaluated. They are quadratic, bilinear, a combination of bilinear and quadratic algorithms where quadratic interpolation were used along X axis and bilinear interpolation were used along Y axis, and a combination of bilinear and cubic algorithms where cubic interpolation were used along X axis and bilinear interpolation were used along Y axis. Results were compared with a reference interpolation.

Analysis of these algorithms shows that bilinear interpolation needs only 13 operations to perform interpolation on one sample, but at the same time its produces very blurry images. The 3x2 and 4x2 needs 22 (11 multiplication) and 35 (19 multiplication) operations respectively. 3x2 interpolation produces acceptable image quality. 4x2 interpolation shows better results along the X-axis then the 3x2 interpolation. An example can be seen in Figure 4.8. The original image which was used for this figure shows a chess board. The combination of black and white squares on the board gives a good image for test of sharp transition and edges. The resulting figure shows a pixel difference between image produced by the reference algorithm and images produced by interpolation algorithms under test.

Quadratic interpolation gives the best images among other tested interpolations (except cubic) but needs 34 operations (18 multiplications). Results can be seen on Figure 4.9. A 23 bit word length was used for interpolation computations and 7x7 LUT compression. The results are the RMS of 100 LUTs per image. In the Appendix B other graphs are presented.

Because of the target FPGA have 18 bits mutiplicators embedded, it was obvious to test with an 18 bits word length for interpolation, where most extensive computations are done. The results of the test can be seen on graphs in Appendix B which shows minimal difference in comparison with 23 bits word length tests.
Figure 4.8. Comparison of image interpolations with 2x2 LUT compression, 32 bit LUT word length and 23 bit word length for interpolation computations.
4.3 Image interpolation algorithms evaluation

Figure 4.9. Image interpolation with 7x7 LUT compression and 23 bit word length for interpolation computations.
Chapter 5

Conclusions

Like it can be seen on graphs presented in previous chapter and in Appendix B increase in size of the image cache does not decrease significantly memory bandwidth and access. The only exception to this is when a size of one line of the block cache is the same as an image line, and the number of lines is at least 28 which will require a minimum of 168 kbit BRAM. By adding to it expenses, which is needed for a LUT caching, the number will grow to something like 6400 kbit \(^1\) which is already unacceptable. With a best LUT compression, see Section 4.1, which could be accepted, the size of the LUT will shrink to 130 kbit. In this case the total size is becoming acceptable but for a small (and cheaper) FPGA’s it will take almost whole BRAM which is opposite to the main goal - to keep the design size small.

The choice between interpolation algorithms is quite simple. Cubic interpolation is too computation costly and nearest neighbor produces unacceptable image quality. What is left is quadratic interpolation, linear interpolation and mixtures of them. If we look at Figure 4.9, Figure B.18 and Figure B.17 the best results gives quadratic interpolation (excluding cubic) and the worst results - the linear interpolation. Not so bad results shows 3x2 and 4x2 interpolations, which is almost the same, but 4x2 interpolation is more computation costly, see Section 4.3. Hence the actual choice is between 3x2 interpolation and quadratic interpolation. First of them is less computation intensive but produces blurry horizontal lines \(^2\) but the overall result is acceptable. Quadratic interpolation gives the image quality almost the same as cubic but with less computation cost which on the other hand larger then for 3x2 interpolation. The choice is dependent on in which application interpolated images will be used, if it is only the overall image quality which is important then 3x2 interpolation is recommended, otherwise quadratic is doing to perform well in all applications.

LUT compression is highly dependent on the required image quality and a size of the FPGA used for implementation. It is also dependent on how much memory bandwidth is available in case if not whole LUT located in BRAM. The quality of the LUT compression is also dependent on how varies values in neighbor

\(^1\) Assuming that every LUT cell needs 19 bit and a size of a LUT is 720x480.

\(^2\) Because of linear interpolation in vertical direction

27
cells. If variations are too large then, with the compression method proposed here, the compression will not be effective because of the unacceptable resulting image quality.
Bibliography


Appendix A

Abbreviations

FPGA  Field-Programmable Gate Array. Large scale programmable logic device.
ASIC  Application-specific integrated circuit.
DSP   Digital Signal Processor.
DDR memory  Double data rate SDRAM.
LUT   Look-up table. A memory structure.
BRAM  Block RAM. On-board RAM used in Xilinx’s devices such as FPGA.
FIFO  First in First out. A que structure where value which was saved first is also read first.
TTL   Time to live. Used to indicate when value in cache is no longer needed.
RMS   Root mean square. It is a statistical measure of the magnitude of a varying quantity.
Appendix B

Graphs

A data presented here were created with 32 bit LUT word length. The last figures which presents interpolation methods are comparison with a reference algorithm, i.e. they shows the differences in pixel values on images produced with reference algorithm and with tested algorithms.
Figure B.1. Memory accesses for 3x2 interpolation, floating

Figure B.2. Memory accesses for 3x2 interpolation, fixed
Figure B.3. Memory accesses for 3x3 interpolation, floating

Figure B.4. Memory accesses for 3x3 interpolation, fixed
Figure B.5. Memory accesses for 4x2 interpolation, floating

Figure B.6. Memory accesses for 4x2 interpolation, fixed
Figure B.7. Memory accesses for 2x2 interpolation, floating

Figure B.8. Memory accesses for 2x2 interpolation, fixed
Figure B.9. Memory bandwidth for 3x2 interpolation, floating

Figure B.10. Memory bandwidth for 3x2 interpolation, fixed
Figure B.11. Memory bandwidth for 3x3 interpolation, floating

Figure B.12. Memory bandwidth for 3x3 interpolation, fixed
Figure B.13. Memory bandwidth for 4x2 interpolation, floating

Figure B.14. Memory bandwidth for 4x2 interpolation, fixed
Figure B.15. Memory bandwidth for 2x2 interpolation, floating

Figure B.16. Memory bandwidth for 2x2 interpolation, fixed
Figure B.17. Image interpolation with no LUT compression and 18 bit word length for interpolation computations.

Figure B.18. Image interpolation with no LUT compression and 32 bit word length for interpolation computations.
Figure B.19. Image interpolation with 2x2 LUT compression and 23 bit word length for interpolation computations.