Camera Functionality in Modern Mobile Terminals

- Software FIR filter for demosaicing with RISC -

Masters Thesis
in
Information and Communication Technology

Submitted to
Agder University College
Faculty of Engineering and Science

By

Tryggve I. Mikkelsen

Grimstad, May 2001

Abstract

This thesis discusses implementation and optimizing of the image pre-processing operation demosaicing on a typical hardware architecture for mobile terminals. Work has shown that the demosaicing operation is equivalent to the mathematical operation of interpolation, and that optimizing potential lies within the implementation of the anti-imaging filter for the interpolator. This filter can be realized as an FIR filter with the filter coefficients obtained from a suitable interpolation function (kernel). Through literature review, nearest neighbor replication, bilinear and cubic convolution kernels were implemented and optimized according to a typical hardware architecture found in mobile terminals. Typical hardware architecture used consisted of ARM710T RISC processor with 8 KB on-chip cache and external SDRAM memory.

The work has shown that the cubic convolution interpolation kernel offers the best image quality requiring only moderate computational effort. Though, dependent the application area, and taking the whole camera system into consideration, bilinear interpolation might be better suited as interpolation kernel for the demosaicing operation in the mobile terminal. The bilinear interpolation offers a good trade-off between the visual result and computational effort. The nearest neighbor replication interpolation kernel offers the poorest image quality, but is the most computational efficient of the kernels.

Preface

-"Any good idea is a least ten years old" -

(Anonymous)

This thesis is carried out as a masters thesis leading to the Norwegian master equivalent degree "Sivilingeniør". The thesis is a part of the ICT (Information and Communication Technology) program at Agder University College and was performed for Ericsson AS. The time schedule for producing the report was the period from 2001-01-09 to 2001-05-28.

I would like to thank my supervisors, Siv.ing. Morten Christiansen at Ericsson AS and Høgskolelektor Paul Bjørn Andersen at Agder University College for their valuable help and inspiration during the work. I would also like to thank Ragnar Johnsen at Agder University College, Jonny Blom and Stein Bergsmark at Ericsson AS for valuable help during the work.

Grimstad, May 2001

_____

Tryggve I. Mikkelsen

Table of contents

List of figures

List of tables

List of equations

1 Introduction

1.1 General field of interest

Multimedia functionality as MMS (Multimedia Messaging Service) and real time video is expected to be functionality when 3G (UMTS – Universal Mobile Telecommunications System) mobile networks are introduced. Furthermore, the users are given a taste of multimedia functionality through still images and possibly video clips in the 2.5G (GPRS – General Packet Radio Service) networks. Already now we find the first solutions for digital cameras on mobile terminals for the GPRS terminals (e.g. Ericsson CommuniCam™, figure 1). From mobile terminal vendors we see the convergence of the PDA, the mobile phone and the digital camera. The addition of the camera functionality to the mobile terminal motivates research in the area of the digital image processing chain in order to ensure power efficient solutions while having good visual quality.

From the world of digital still cameras and web cameras, the mobile terminal has adopted the CMOS (Complementary Metal Oxide Semiconductor) image sensor. These image sensors use a Color Filter Array (CFA) in order to reduce complexity and cost. The most common CFA is the Bayer pattern that reduces the data size to only include one color per pixel.

The image operation that sees to that we get full resolution RGB from the Bayer Pattern RGB is the de*mosaicing operation* – also known as pixel interpolation. This thesis contributes to the research in improving the camera system chain by finding efficient solutions for implementing the demosaicing operation to typical mobile terminal hardware (HW) architecture. In times where the user demands smaller products, “single-chip” HW solutions requires a HW architecture that can solve a lot of functionality. The overall goal for the architecture is low power consumption. In order to increase the performance to the special functionality, dedicated HW can be designed. Often this is not desirable due to increased physical size. This thesis therefore has the starting point that the existing HW in the mobile terminal is to be used. Moreover, this HW is expected to include the use of a RISC processor.

Figure 1. Ericsson CommuniCam™.

1.2 Thesis proposal

1. a) Describe typical hardware architecture in modern mobile terminals using RISC CPU. The architecture should be used in the further work. Also chose and describe possible development platform for the work.
   b) Describe a typical modern camera system. A sub part of the camera system should be chosen and used in the further work.

2. Related to 1a) and 1b); discuss factors involved in SW optimizing to HW architecture.

3. Related to 1a) and 1b); discuss suitable ways to measure performance/cost for (a sub-system of) the camera system.

4. Apply the work from 1, 2 and 3 on a sub-system of the camera system to indicate the performance/cost for this functionality. (This task could enter into 2 and 3.)

5. Suggest improvements for the implementation based on the work from 1, 2, 3 and 4.

Note:

- It is not the main scope for the diploma to *develop* code.
- It is not the scope for the diploma to compare SW implementations to HW implementations, but to provide sufficient information and performance/cost measures to allow designers to make a well-informed choice of HW vs. SW implementation of the (part of the) camera system.
- The main scope for the work is put to 2 and 3.

Deleted: Briefly describe
Deleted: work.¶
Deleted: b) Briefly
Deleted: be used
Deleted: Investigate
Deleted: code-optimising/profiling related to the chosen hardware architecture.
Deleted: Investigate
Deleted: based on the chosen hardware platform
Formatted: Bullets and Numbering
Deleted: <#>¶

1.3 Limitations and concretization

It was necessary to concretize the original thesis proposal. The original thesis proposal only required to chose a sub-system of the camera system for investigation. When concretizing the thesis theme, there are three aspects to discuss or describe:

- The mobile terminal HW architecture.
- The (sub-system) of the camera system.
- Measures for performance and cost.

The most important concretizations and limitations introduced are:

- The scope is narrowed to only include **demosaicing operation** in the pre-processing chain, that is, the image operations prior to compression in the image processing chain. Furthermore, only:
  - a subset of interpolation kernels is discussed.
  - a subset of these kernels is implemented.
- Development of C-code only for HW architecture constituted of:
  - ARM710T RISC processor, with *no coprocessors*, and using ARM/THUMB interworking instruction set.
  - SDRAM as external memory.
- Little effort is paid to *general* code optimizing for ARM. The term ‘general code optimizing’ is related to software optimizing as loops, lookup tables etc.

In understanding with my supervisor, this led to the final work theme for the thesis:

- *Describe* a typical HW architecture used in mobile terminals today.
- *Investigate* the demosaicing operation in the pre-processing chain. What is demosaicing? How can the operation be implemented efficiently on the HW used in mobile terminals?
- *Implement* and *measure* the performance and cost for the demosaicing operation. The performance and cost measures should be defined.

1.4 Method

In order to give the thesis a fundament and validity I chosed the method as follows:

- Review published literature from the area of digital signal processing, digital image processing and digital still cameras. Furthermore, published literature in the area of mobile terminal HW architecture using RISC processor.
- Based on this review, discuss and suggest an implementation of a chosen subpart of the pre-processing chain.
- Implement and run tests for the chosen subpart and measure its performance and cost on my own software. The measuring method should rely on methods and measures described in literature in order to give validity. Sufficient information for easy reproduction of test results put forward should be included.
- Discuss the results in relation to the literature review

In order to broaden scope and collect second and third party ideas, I've had numerous informal personal conversations, and meetings, with experts on different fields from Ericsson, Agder University College, Philips and authors of papers.

1.5 Report structure

The report structure is chosen in order to preserver three aspects:

- Give the framework to describe the work carried out to perform the work asked for in the thesis proposal
- Give the framework to reflect the method chosen.
- Give an educational presentation of the work.

Figure 2 illustrates the limitations, concretization and the method chosen for the thesis as an "hourglass model". This model is chosen to preserve the three aspects described above.

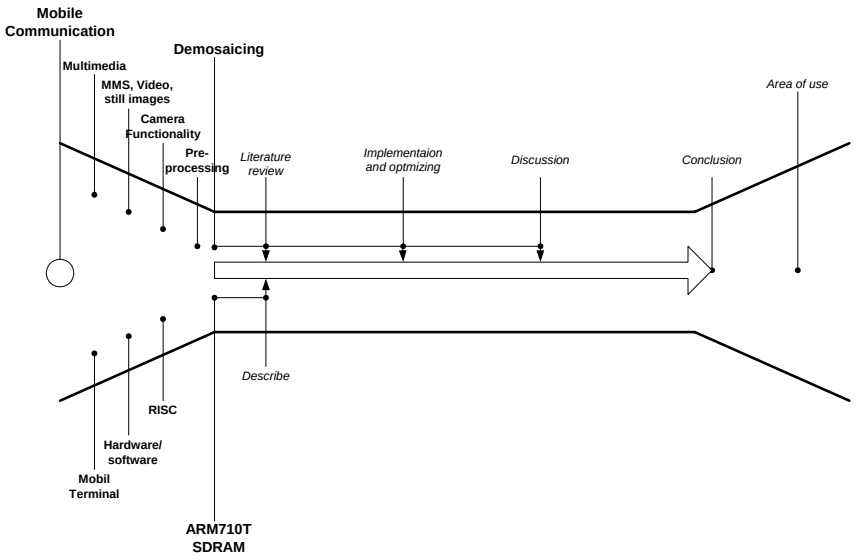


Figure 2. The "hourglass model".

Even though only the demosaicing operation from the camera system is chosen for study, the reader needs to be brought into the problem and status on the area. From this reason the report has a first introduction – or the 'popular explanation' - of the digital image and the camera system.

In order to relate the literature review to a HW (hardware) architecture, the HW architecture needs to be described first since the optimizing is based on the chosen HW architecture. When these matters have been described I have a literature review of the actual problem – the implementation of the demosaicing operation on RISC processor. Also, I include a discussion of suitable ways to measure the quality for the different implementations of the operation.

The review of the demosaicing operation leads to the part where I test the outcome from the literature review. I also present the information and means to reproduce the test results. The report finishes up with a discussion of the results put forward - and the validity of the results. Further the report discusses which interpolation kernel that is better or worse suited for mobile terminals.

2 What is demosaicing?

2.1 Introduction

This chapter gives the reader an introduction to the pre-processing chain in a modern camera system. The introduction serves to help understanding what demosaicing is and what it does. The chapter starts with a representation of the digital image and the two most used color models for representing digital color images. Then a presentation of a typical modern camera system and elaboration of the pre-processing part to show where the demosaicing operation fits in.

2.2 The digital image

2.2.1 General

A digital image is the representation of a continuous image f(x,y) by a 2-D array of digitized values both spatially and in amplitude. Digitization of the spatially coordinates (x,y) is called image sampling, and amplitude digitization is called gray-level quantization. For a digital image we can then represent it as a two-dimensional N x M array of gray values as in equation 1:

Equation 1. Representation of a digital image.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & .... & f(0,M-1) \\ f(1,0) & f(1,1) & .... & f(0,0) \\ .... & .... & .... & .... \\ f(N-1,0) & f(N-1,1) & .... & f(N-1,M-1) \end{bmatrix}$$

By dividing the term *resolution* into two parts, - the spatial resolution and the brightness resolution, these two parameters decide how closely the digital image will match the continuous image.

2.2.2 Spatial resolution

The spatial resolution refers to the 2-D (two dimensional) image space, and describes how many pixels that represent the digital image. In (near) all cases, the spatial resolution is fixed. Standard spatial resolutions and common for the mobile terminal world are:

- VGA: 640 x 480 = 307200 pixels
- QVGA: 320 x 240 = 76800 pixels
- VGA/8: 160 x 120 = 19200 pixels
- CIF: 352 x 288 = 101376 pixels
- QCIF: 176 x 144 = 25344 pixels
- 4CIF: 704 x 576 = 405504 pixels
- 16CIF: 1408 x 1152 = 1622016 pixels

VGA (Video Graphics Array) is common for PC's (Personal Computers). CIF (Common Interchange Format) is the standard for videoconferencing. The others are derivations of these. For mobile terminal display, VGA/8 and QCIF are common. However, when sending images as e.g. e-mail attachments, VGA is common.

2.2.3 Brightness resolution

The concept of brightness resolution addresses how accurately the digital pixel's brightness can represent the intensity of the original image. The term *intensity* refers to the magnitude or amount of light energy actually reflected or transmitted from a physical scene. The term *brightness* refers to the measured intensity after it is acquired, sampled, quantized, displayed and observed (by our eyes). The brightness of a pixel accounts for all the effects induced by the entire imaging system.

The number of *brightness* levels is given by the number of bits, and are a power of 2. 8-bit brightness scale gives a dynamic range of 256 brightness levels (0 to 255).

2.3 Color space models

2.3.1 The RGB color space model

To obtain a color image, the image needs to be represented by a color model. The most common model, and the one used in this thesis, is the RGB color space model. This model uses the primary colors red, green and blue and additive color mixing to represent all colors. Using the RGB color model with a brightness resolution of 8 bit, it is possible to represent $2^{(8x3)} = 2^{24}$ (approximately 16 million) colors, or often just referred to as 'millions of colors'. In this thesis only 3 (RGB) x 8-bit brightness scale is used.

2.3.2 The YUV color space model

The purpose with this color model is to take advantage of the eyes less sensitivity to chrominance than luminance. By splitting this image information, one can remove or compress chrominance information more than luminance information to reduce data size. The luminance information, represented by *Y*, and the chrominance information, represented by *U* and *V*, is given by the matrix in equation 2.

Equation 2. YUV matrix.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

From equation 2 to we can represent luminance information as in equation 3. We notice that green is weighted much more than red and blue. This is to reflect that the eye is more sensitive for green information than for red and blue. Thus green will give a greater contribution to the luminance.

Equation 3. Luminance information in terms of R, G and B.

$$Y = 0.299R + 0.587G + 0.114B$$

2.4 What does the demosaicing operation do?

2.4.1 General

In order to place the demosaicing operation in the camera system I present two figures. Figure 3 shows a simple camera system. The pre-processing part of the camera system is elaborated further in figure 4.

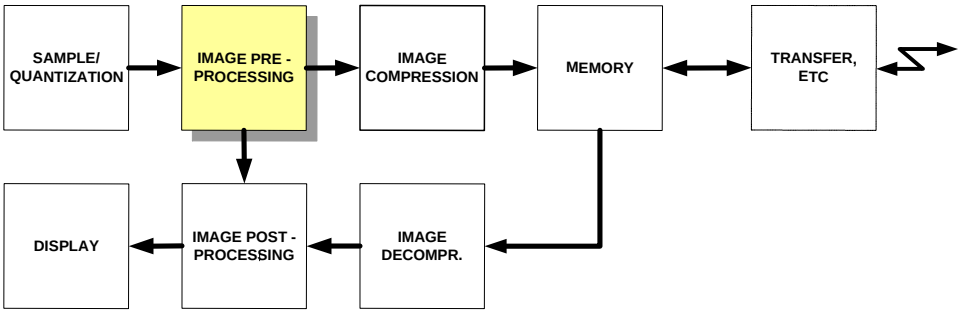


Figure 3. Simple camera system.

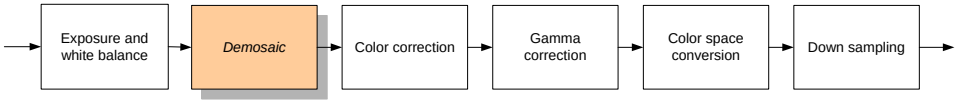


Figure 4. Image pre-processing operations.

2.4.2 The Bayer Pattern color filter array

Referring to figure 4, the **exposure** block decides how much light that is to be let in to the light sensitive photodiode. This exposure block sets the integration time and gain for each (or groups of) pixel to meet target average pixel luminance. The **white balance** equalizes average pixel luminance among color bands (RGB). If this task is not carried out, a white motif won't be white when reproduced!

Prior to the exposure and white balance block is the **capture and quantize** operation that captures the real life motif. In order to reduce complexity (and thus cost) and data rate, the image sensor performing these operations is typically equipped with a Color

Filter Array (CFA) that is placed between the lens and the photo sensors. A CFA typically has one color filter element for each sensor. Obviously, the color interpolation algorithms depend on the CFA configuration. There are different CFA configurations. The most popular and the *de facto* standard is the Bayer pattern [Bayer U.S Patent], which uses the three additive primary colors, red, green and blue , for the filter elements. For limiting scope, this CFA is the only one discussed in this thesis. The Bayer pattern uses the fact that the eye is less sensitive to chrominance than to luminance. Moreover, the Bayer pattern approximates green to be luminance information due to the eyes higher sensibility for green light, and red and blue to chrominance information. The green color is subsampled at a rate of ½ in horizontal direction. Red and blue are subsampled in both horizontal and vertical direction at a rate of ½. Effectively, the green elements fill the pattern with 50%, and red and blue with 25% each (se figure 5).

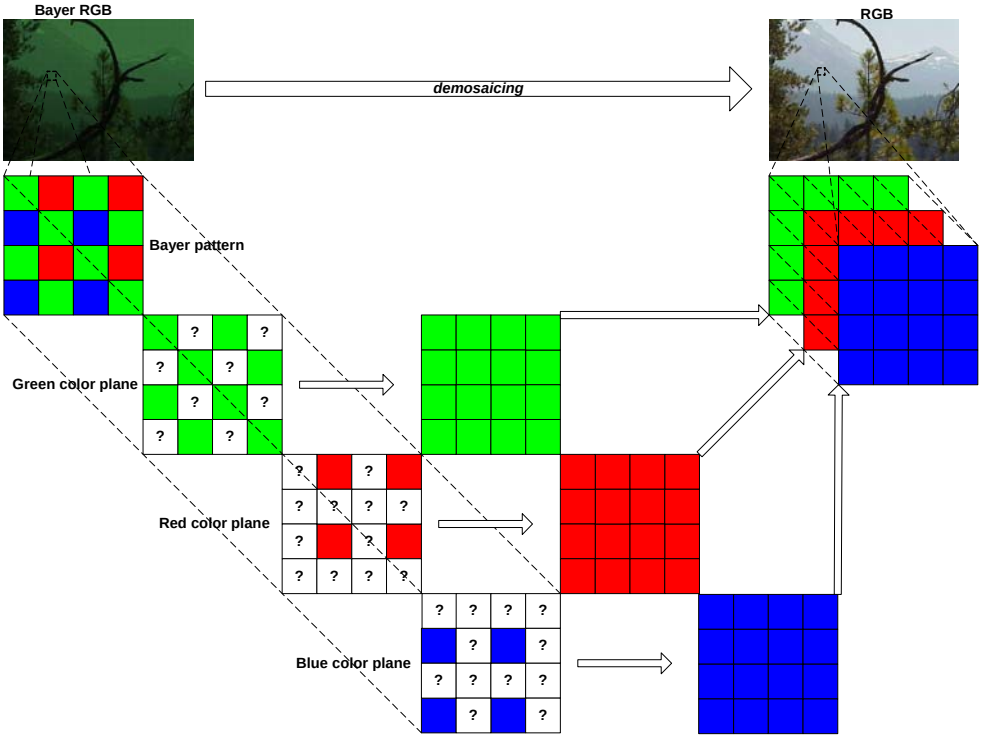


Figure 5. Illustration of the demosaicing operation and the Bayer pattern

2.4.3 The demosaicing operation explained in simple terms

Now that we know that the image sensor does not provide us with RGB values for each pixel, but only one of the colors, we needs to implement an operation that gives us the two remaining colors – the demosaicing operation. The demosaicing operation converts the raw CFA image data into full resolution RGB. The demosaicing operation is an important operation when considering both image quality and computational effort. This operation is to reproduce the real world motif by increasing the spatial resolution for the three color planes in the RGB color space. That is, the spatial resolution counting pixels is kept (e.g. VGA = 640 x 480), but the resolution expanded to the same resolution for all three color planes in the color space (see figure 5). The main goal for the algorithm used to find the RGB values is to get the same performance from the CFA array and the demosaicing operation as from three image sensor arrays for red, green and blue.

2.4.4 Other pre-processing operations

Referring to figure 4, I include a short description of the remaining pre-processing operations for overview.

The **color correction** is necessary due to compensate for the color filter response of the image sensor. The photo sensors have a differing spectral sensitivity to the color red, green and blue than the eyes. This needs to be corrected.

Since the eye has a non-linear perception of luminance **gamma correction** is performed to compensate. The gamma correction block therefore has a transfer function equal to the eye.

A **color space conversion** is necessary to meet the standards used in compression techniques in digital imaging systems today. Different compression schemes use different color spaces as input, but the most popular is JPEG (Joint Photographic Expert Group) compression. JPEG can be implemented both loss less or lossy, where the lossy algorithm is the one best suited for mobile terminal since this has a much better compression factor leading to less data to transfer. In this scheme, a **downsampling** is carried out prior to the actual coding to further reduce image data size. The color space usually used to subsample is YUV, where Y is the luminance (brightness) component and U and V are chrominance (color) components. The reason for using this color space is due to the fact that the eye is less sensible to color differences than luminance differences. Using this fact one can downsample chrominance components (and thereby reducing image data size) more than the luminance component without getting a perceived lower quality of the image.

3 A mobile terminal hardware architecture

3.1 Introduction

The motivation for this part is to *describe* a typical HW architecture used in mobile terminals today. There is not a thorough *discussion* why this architecture is chosen, since the thesis proposal assumes the use of a RISC processor. It is expected that the reader has some prior knowledge to RISC architecture since it is out of scope for this thesis to have an in depth explanation here. Although new functionality probably will force the HW architecture to change, many designers can use results put forward in this thesis to compare the cost considering HW vs. SW implementation on the mobile terminal HW architecture used today. The reader should also bear in mind that the HW in focus in this thesis is the HW concerned with the camera functionality. This HW will most likely be the same HW used for features as "calendar" and "organizer" on the terminal and is most likely embedded in an ASIC (Application Specific Integrated Circuit). It is therefore a paramount goal to reduce the size of the ASIC by reducing the number and the size of the IP blocks to include. (IP = Intellectual Property – that is, HW blocks implementing a functionality/performing a task embedded within the ASIC).

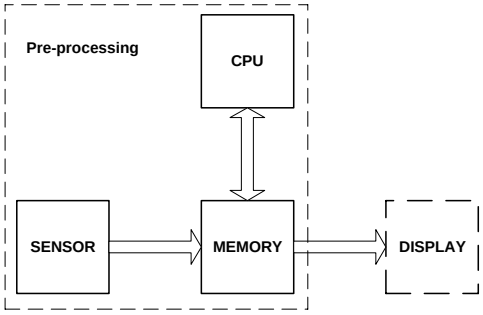


Figure 6. General HW components.

It is important for the HW in mobile equipment to use as little power as possible. Low power consumption means longer battery life and operational time that in turn is the end user requirements. Adding functionality has no meaning if the operational time is reduced drastically. Reducing power consumption can be done in several ways:

- Smaller physical size. The smaller size allows a higher level of integration that in turn will reduce the IC count and hence reduce cost and power consumption.
- Reducing circuit voltage supply. Lower circuit voltage means lower power consumption, but it also means reduced performance. [Furber, 1996].
- Reducing circuit activity. (Techniques as clock gating etc).
- Reducing clock frequency, *f*. Lower *f* means lower power consumption. But it also means lower performance. However, if the circuit voltage can be reduced due to the reduced *f*, then this will be beneficial to the MIPS/Watt performance.

In this thesis these issues are considered given by the architecture chosen. Instead, the interest in this thesis is put to the computational effort. Reduced computational effort leads to reduced power consumption and is therefore of interest for mobile terminals.

3.2 The image sensor

The **image sensor** is, as already described, for mobile terminals typically a CMOS (Complementary Metal Oxide Semiconductor) image sensor due to cost, power consumption and production volume. They also typically come with some HW for buffering and control built in. The means for control is carried out through a set of registers, and the control loop could be carried out in HW or SW. The interesting property of the image sensor is the image format output from the sensor described earlier – the RGB Bayer pattern.

3.3 The RISC processor

3.3.1 The ARM architecture

Research findings reported in the January 2001 issue of *Inside the New Computer Industry*, the industry newsletter published by Andrew Allison [Allison, 2001], reveal that ARM's share of the 32-bit embedded RISC microprocessor market grew to 76.8 percent in 2000, an increase of 19 percent over the same period in 1999. This makes ARM the industry's leading provider of 16/32-bit embedded RISC microprocessor solutions. ARM licenses its high-performance, low-cost, power-efficient RISC processors to leading international electronics companies. ARM's microprocessor cores are rapidly becoming the volume RISC standard in such markets as portable communications, hand-held computing and multimedia digital consumer. A glance into ARM’s website [www.arm.com] discloses that major mobile terminal vendors as Ericsson, Nokia and Motorola use ARM cores in their mobile terminals. In other words, ARM’s processors are typically found in HW architecture for mobile terminals.

The ARM processors are *Reduced Instruction Set Computers* (RISC). The RISC concept originated in processors research programs at Stanford and Berkeley universities around 1980. The ARM architecture incorporates a number of features from the Berkeley RISC design, but it also rejects a number of features. From the Berkeley RISC features like LOAD-STORE architecture, fixed-length instructions and 3-address instructions formats, were kept. ARM RISC has a simple hardware organization and implementation and in combination with an instruction set that is grounded in RISC ideas but retrains a few key CISC (Complex Instruction Set Computer) features has given ARM its power-efficiencies and its small core size. The memory organization can be viewed as a linear array of bytes. Data items may be 8-bit bytes, 16 bit half-words or 32 bit words. From the load-store architecture it follows that the processor instruction set will only process values that are in registers (or specified directly by the instruction itself), and will always place the result into a register. The only operations that apply for memory is load instructions that copy memory values into register, and store values that copy register values into memory. All ARM instructions will therefore be one of three categories:

- Data processing instructions - These use and change only register values.
- Data transfer instructions - Load/store instructions.
- Control flow instructions - Branch instructions.

All ARM instructions are 32 bits wide (except THUMB instructions) and are aligned on 4-byte boundaries in memory. Features of the ARM instructions are:

- Load-store architecture.
- 3-address data processing instructions (source operand registers and result registers are independently specified).
- Conditional execution of every instruction.
- Powerful load and store for multiple register instructions.
- Ability to perform a general shift operation and a general ALU operation in a single instruction that executes in a single clock cycle.
- Open instruction set extensions through the coprocessor instruction set, including adding new registers and data types to the programmer's model.
- Dense 16-bit compressed representation of the instruction set in the THUMB architecture.

3.3.2 ARM 710T

A typical ARM processor for mobile terminals is the ARM710T processor that is based on the ARM7TDMI core.

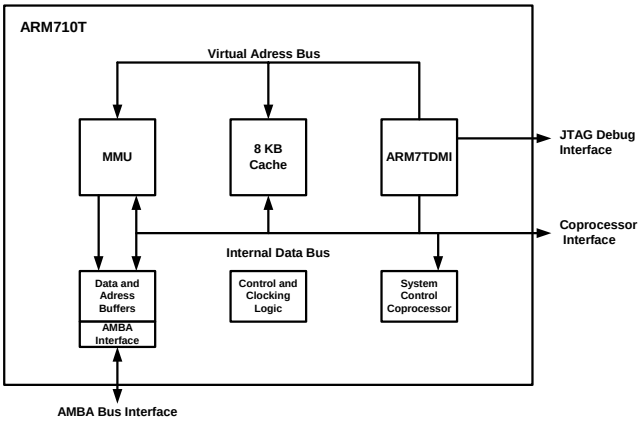


Figure 7. ARM 710T.

The ARM 7TDMI is a 32-bit RISC ARM microprocessor core designed for use in mobile terminals. The core is smaller ins size and consumes less power than typical 8 and 16 bit CISC processors. The ARM7TDMI uses a three stage pipeline, which allows it to execute an instruction on (nearly) every clock cycle, translating to higher performance at the same clock speeds than CISC devices. The ARM7TDMI is fully 32-bit including 32-bit ALU, barrel shifter, data and address bus. As the core is intended to be embedded in an ASIC, a choice can be made on how many of the address lines and data lines are brought out to pads.

3.3.3 The THUMB instruction set

The ARM7TDMI includes the THUMB Instruction decompressor. The THUMB decompressor allows the ARM7TDMI to support two different instruction sets – the traditional 32-bit wide ARM instruction set and the reduced 16-bit wide THUMB instruction set. The purpose with the THUMB instruction set is to reduce code size and thus reduce memory requirements. For a given piece of C-code the finals executable machine code will be larger for a 32-bit wide ARM instruction set than that of 8 and 16-bit CISC. With the ARM and THUMB instruction set available, the user is free to trade off performance against memory consumption, on a function by function basis. (For a better reference to THUMB instruction set, refer to [Furber, 1996, p.199]).

3.3.4 Cache.

The use of a cache in mobile terminals will effectively reduce power consumption. The main reason for this is that external memory accesses to SRAM or SDRAM are by far more power consuming than the cache accesses. Furthermore, the cache increases execution speed for the processes. A cache is therefore a “must” in mobile terminals. We want as few external memory accesses as possible, since this is both time expensive and a power consuming task. It is therefore of interest to have as much as possible of both code and data in cache when processing.

The organization of the cache varies. For the ARM710T processor a 8kb mixed cache for data and instructions is included (figure 8).

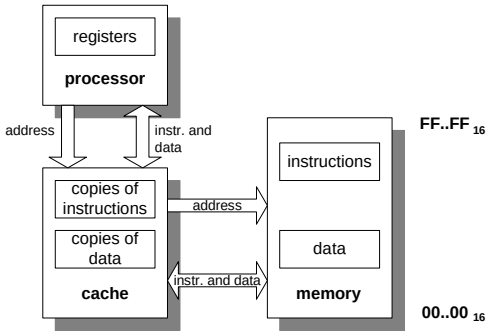


Figure 8. Unified cache.

Since the cache holds a dynamically varying selection of items from main memory it must have storage for both the data and addresses that the data is stored at in main memory.

3.3.5 Coprocessors

The ARM processors supports a general extension of its instruction set through the addition of HW coprocessors, and it also supports the SW emulation of these coprocessors through an *undefined instruction trap.* The "undefined instruction trap" is the way the ARM processors handle instructions that are not in ARM instruction set. The ARM coprocessor interface is based on "bus watching". The coprocessor is attached to a bus where ARM instruction stream flows into the ARM, and the coprocessor copies the instructions into an internal pipeline that mimics the behavior of the ARM instruction pipeline. The coprocessors of interest when considering image operations are DSP-coprocessor and floating-point support. However, there has traditionally been little or no use for floating-point support in mobile terminals - code is instead rewritten to integer arithmetic. This way we save die size in the ASIC, or computational effort (if SW coprocessor emulator were to be used). It is therefore legitimate to say that the mobile terminals today do not include any specific coprocessors.

3.4 External memory

Traditionally **SRAM** has been the preferred external memory in mobile terminals. The main reason for this is that the terminals traditionally have had a limited functionality. Up until recently a mobile terminal was a mobile *phone*, and the memory requirements were limited and the small size SRAM was sufficient. Furthermore, the SRAM has a simple interface and thus only a simple memory controller is necessary. The memory controller is typically embedded in an ASIC and a simple memory controller reduces the ASIC complexity and thus cost of the product. However, introducing new network technology and market trends, other functionality is to be implemented on the terminals. This functionality needs more memory space. This is why the SRAM is shifted to **SDRAM** since the SDRAM comes in larger sizes (typically 32 – 256 Mbit). Traditionally the PC (Personal Computer) market has been the major purchaser of SDRAM memory, and unfortunately the PC manufacturers pay little to power consumption and size. With the large growth in the number of battery powered handheld devices as PDA's and mobile terminals, a new JEDEC standard for Mobile SDRAM has been proposed. The Mobile SDRAM has lower voltage supply and also includes additional features for lowering the power consumption. Furthermore the Mobile SDRAM has smaller physical size. However, Mobile SDRAM is not available yet so I therefore use SDRAM as external memory due to its large storage capacity.

The mobile terminal also typically has a more permanent storage medium. Flash memory is commonly used for the file system and storage for user data purposes. The software can run either from SDRAM or flash memory.

3.5 Typical hardware architecture

Through the description previously and personal conversation with a HW team leader at Ericsson Mobile AS [RKN, 2001], it is quite legitimate to say that the simple illustration in figure 9 is a typical HW architecture used in mobile terminals. (Note! I only consider the HW components involved for image pre-processing).

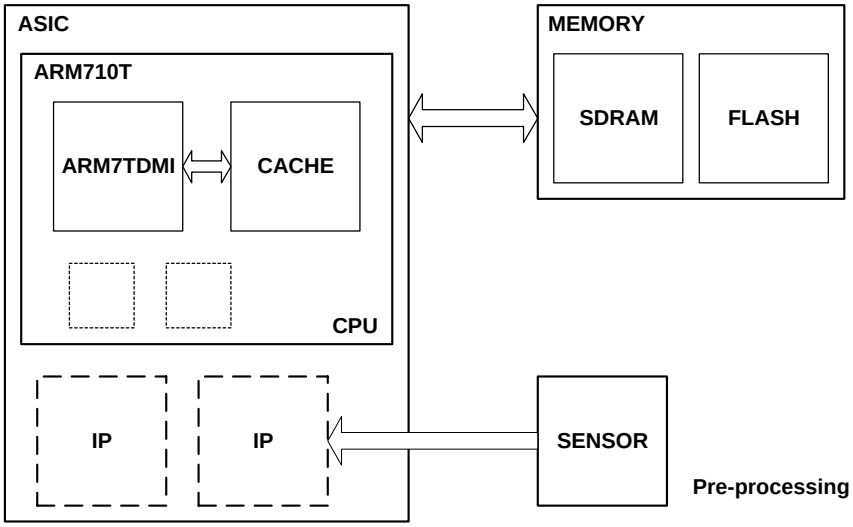


Figure 9. Typical hardware architecture.

3.6 Writing efficient code for the HW

3.6.1 General

The terminal has scarce resources when considering power, memory size and computational effort and optimizing the code is therefore a matter of necessity in order to reduce system cost. Optimizing code is tied up to the CPU vendor, and little published literature is found other than web references due to rapidly changing cores and processors. This discussion is therefore based on both published literature and online documentation from ARM Ltd. [ARM, App. Note 34] in addition to [Furber, 1996]. Optimizing the implementation of a functionality or operation can be done in various ways:

- Choosing the "right" *algorithm.*
- Optimizing the *code* (that is; the implementation of the algorithm) to HW architecture.
- Choosing the right data structure for the *data* so that the *data* is transferred and stored in the most efficient way in memory.

In this thesis the optimizing concentrates about the choice of algorithm rather than general *code* optimizing to the HW architecture. Though, a brief description of general code optimizing for ARM architecture is included for later improvement of

system cost for the algorithms. This thesis is nor about optimizing the *data structure* of the image data to the HW architecture. Starting to manipulate the way data is stored is very tied up to the individual implementation of the memory system. It is out of the scope for this thesis to look into detail of how a specific memory controller and memory chip arrange and stores data.

Speeding up the operation and reducing power consumption involves attention paid to how the operation is implemented. When writing source code it is always worthwhile to use programming techniques that work well with RISC processors and the HW architecture in general (related to memory access etc). The motivation for the effort paid to this task is to increase execution speed, lower code density and reduced power consumption. (There will most likely be a trade off between these). From increased execution speed follows less overhead per pixel processing, and from better code density less area occupied in memory. Also, a lower code size results in a bigger amount of code available in a limited cache size thus increasing execution speed.

3.6.2 Division

ARM/THUMB instruction set does not include (integer) division. Division is therefore typically implemented by calling a C-library function. Depending on the numerator and denominator, a 32-bit division takes 20-140 cycles. Since the division is an expensive operation it is desirable to avoid it where possible. Effort should therefore be put into how the division could be carried out most efficiently. If possible the denominator in a division operation should be a power of two since the compiler then uses shift operations to perform the division. This avoids calling the C library function implementing the division operation. Also the division should be done unsigned since this further reduces the number of instruction to perform the division.

3.6.3 Loops

The loop termination condition can cause overhead if written without caution. One should always write *count-down-to-zero* loops and use simple termination conditions. When using a decrementing loop towards zero fewer instructions are needed due to comparing to zero can be optimized away.

3.6.4 Lookup tables.

In algorithms where mathematical functions as logarithms or trigonometric (or divisions) functions are necessary using a lookup table is a good solution. Using lookup tables one should combine as many adjacent operations as possible into one single lookup table. This increases execution speed and uses less memory space.

3.6.5 Use of memory

Various data types supported in C require differing amounts of memory to store their binary representations. Further, the ARM instruction set, in common with many other RISC processors, is most efficient at loading and storing data items when they are properly aligned in memory. A byte access can be made to any byte address with equal efficiency, but storing a word to a non word-aligned address is very inefficient, taking up seven ARM instructions and requiring temporary work registers. The ARM C compiler therefore generally aligns data items on appropriate boundaries:

- Bytes are stored at any byte address.
- Half-words are stored at even byte addresses.
- Words are stored on four-byte boundaries.

Where several data items of different types are declared at the same time, the compiler will introduce padding where necessary to achieve this alignment.

4 Optimizing the demosaicing operation

4.1 Introduction

This chapter is a literature review to find efficient ways to implement the demosaicing operation on a typical HW architecture for mobile terminals. Doing so requires knowledge to both the HW architecture and the image operation. The HW architecture is described earlier, so is the image pre-processing chain. This chapter aim to investigate the following:

- What is demosaicing beyond the simple explanation given previously?
- How can an implementation be done in the most optimal way considering the HW architecture available?

4.2 Adaptive and non-adaptive algorithms

Algorithms for color demosaicing can be classified into two groups. These groups are the non-adaptive algorithms and the adaptive algorithms.

Non-adaptive algorithms denote those algorithms that perform interpolation in a fixed pattern for every pixel (within a group).

Adaptive algorithms imply that those algorithms can detect local spatial features present in the pixel neighborhood and then make effective choices as to which predictor to use for that neighborhood. In other words, adaptive algorithms possess some intelligence and therefore are more sophisticated in general. It is not the intention for this thesis to implement every known interpolation *algorithm*, but to implement some of them and measure their performance and cost. For this reason, I leave out the adaptive algorithms, and concentrate on non-adaptive algorithms. Also, a non-adaptive algorithm will always be the basis in an adaptive algorithm.

4.3 Mathematical review of the demosaicing operation.

For simplicity, the mathematical description of the demosaicing is shown in one dimension (1-D), but it can easily be adapted to 2-D images by separable extension [Keys, 1981].

From the previous description of the camera system, demosaicing was described as the operation that performed RGB Bayer to full resolution RGB color space. Mathematically the demosaicing operation can be described as a geometrical transformation for increasing the spatial resolution. This geometrical transformation is more precisely known by the term *interpolation*. Figure 10 shows a general interpolator. Further, figure 11 and figure 12 gives a spatial and frequency domain illustration of the interpolation process respectively. As seen from figure 10, the interpolation process is a two-step process:

- Up sampling by a factor L in a sample rate expander (with zero insertion).
- Low pass filtering (LPF) anti-imaging filter (the interpolation kernel).

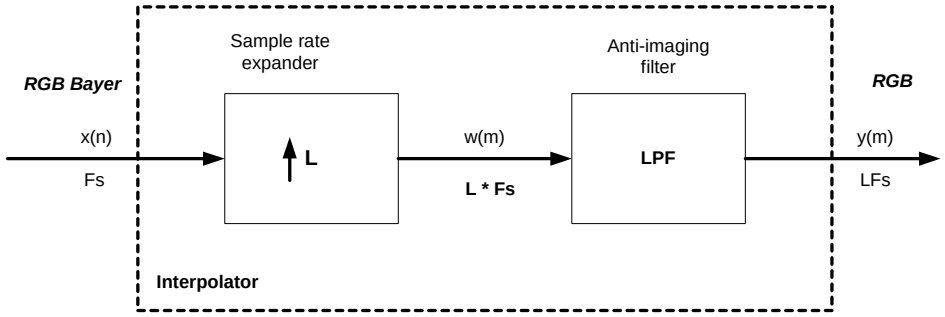


Figure 10. Interpolator.

The sample rate expander inserts L-1 zero valued samples to the input x(n) (the RGB Bayer) to form w(m) ("zero inserted" RGB). For the interpolation operation*) involved in demosaicing from RGB Bayer to RGB, there will for every x(n) be output two w(m) where one sample is equal to x(n) and one sample is the inserted zero, that is, L=2. This signal is then low pass filtered by an anti-imaging filter (interpolation kernel) to remove image frequencies created by the rate increase to yield y(m) (full resolution RGB). Also, the LPF produces the new interpolated values for the zero valued samples.

If the image frequencies in figure 12 of the signal are not removed (filtered away) by an anti-imaging filter they will appear as other frequencies and give a distorted result in the reconstructed signal. How much imaging the system experiences is dependent of steepness of the filters cut-off. The observant reader will recognize the anti-imaging filter as the reconstruction filter following a DAC (Digital to Analog Converter). The difference here though is that we are not interpolating towards a continuous (analog) signal, but to another digitized signal (the RGB pixel values). The interpolation operation is characterized by the input-output relationship in equation 4. Now, this equation is equivalent to the equation for discrete convolution. The reader should remember that convolution in spatial domain is equivalent with multiplying in the frequency domain. The correspondence between figure 11 and figure 12 should therefore be apparent. That is, the interpolation process *can* be considered to be a low pass filtering in the frequency domain (of course, dependent on the h(k)). Again, the steepness of the filter will decide how much of the high frequency detail in the signal that will be preserved since we want to preserve the details, but cut out imaging frequencies.

Equation 4. General interpolator.

$$y(m) = \sum_{k=-\infty}^{k=\infty} h(k)w(m-k)$$

$$w(m) = \begin{cases} x(m/L), & \text{m} = 0, +/-\text{L}, +/-2\text{L},... \\ 0 \end{cases}$$

*) The input and output for the interpolator are the separated R, G or B image plane.

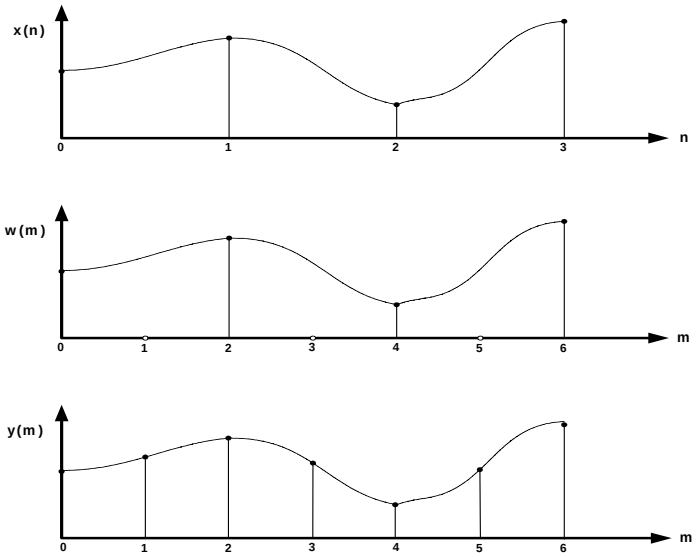


Figure 11. Spatial domain illustration of interpolation.

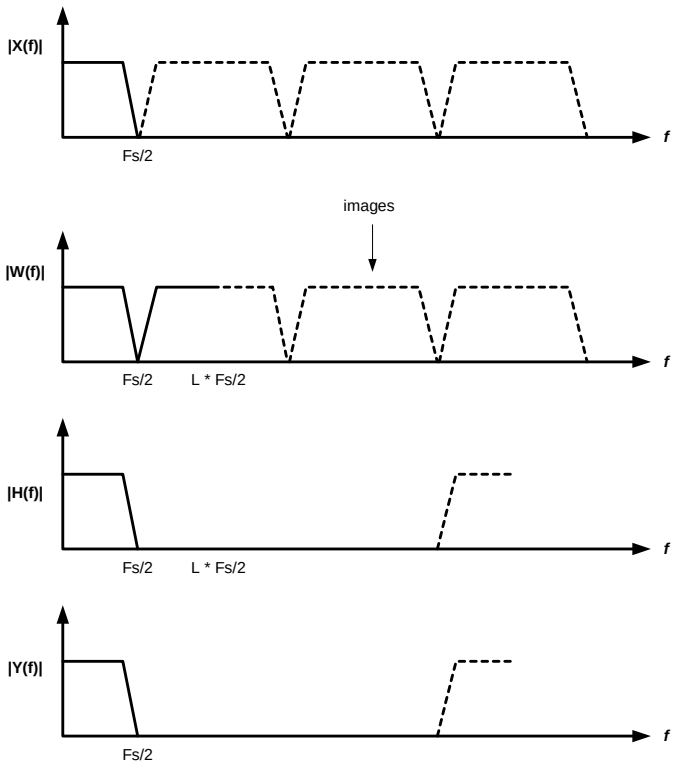


Figure 12. Frequency domain illustration of interpolation.

4.4 The anti-imaging filter - the interpolation kernel

4.4.1 General

Having unveiled the mathematical nature of the demosaicing operation, I now turn to how to optimize the interpolation process. Optimizing potential lies within the anti-imaging filter - the interpolation kernel h(k) in equation 3. Since we are to interpolate from discrete values (RGB Bayer) to discrete values (RGB pixel values), it is not a continuous interpolation function we shall use, but a sampled and quantized version. This enables us to use a FIR (Finite Impulse Response) filter. Still, which interpolation function to choose and the number of coefficients need to be discussed. From fundamental digital image processing Jain [Jain, 1989] present a table of interpolation kernels with rising complexity. Table 1 is copy of [Jain 1989, figure 4.13 p 96] included for convenience. The left side of the table 1 shows the spatial domain characteristics, and the right side shows the frequency domain characteristics for the different interpolation kernels.

Table 1. Interpolation kernels.

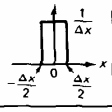
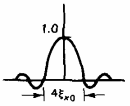
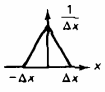
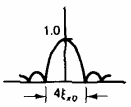
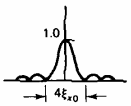
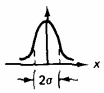
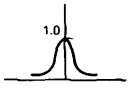
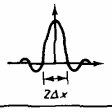
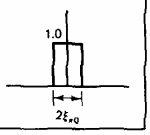
One-dimensional interpolation function	Diagram	Definition $p(x)$	Two-dimensional interpolation function $p_d(x, y) = p(x)p(y)$	Frequency response $P_d(\xi_1, \xi_2)$	$P_d(\xi_1, 0)$
Rectangle (zero-order hold) ZOH $p_o(x)$		$\frac{1}{\Delta x}\,\mathrm{rect}\left(\frac{x}{\Delta x}\right)$	$p_o(x)p_o(y)$	$\mathrm{sinc}\left(\frac{\xi_1}{2\xi_{x0}}\right)\mathrm{sinc}\left(\frac{\xi_2}{2\xi_{y0}}\right)$	
Triangle (first-order hold) FOH $p_1(x)$		$\frac{1}{\Delta x}\,\mathrm{tri}\left(\frac{x}{\Delta x}\right)$ $p_o(x) \circledcirc p_o(x)$	$p_1(x)p_1(y)$	$\left[\mathrm{sinc}\left(\frac{\xi_1}{2\xi_{x0}}\right)\mathrm{sinc}\left(\frac{\xi_2}{2\xi_{y0}}\right)\right]^2$	
nth-order hold $n = 2$, quadratic $n = 3$, cubic splines $p_n(x)$		$p_o(x) \circledcirc \cdots \circledcirc p_o(x)$ n convolutions	$p_n(x)p_n(y)$	$\left[\mathrm{sinc}\left(\frac{\xi_1}{\xi_{x0}}\right)\mathrm{sinc}\left(\frac{\xi_2}{\xi_{y0}}\right)\right]^{n+1}$	
Gaussian $p_g(x)$		$\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{x^2}{2\sigma^2}\right]$	$\frac{1}{2\pi\sigma^2}\exp\left[-\frac{(x^2+y^2)}{2\sigma^2}\right]$	$\exp\left[-2\pi^2\sigma^2(\xi_1^2+\xi_2^2)\right]$	
Sinc		$\frac{1}{\Delta x}\,\mathrm{sinc}\left(\frac{x}{\Delta x}\right)$	$\frac{1}{\Delta x \Delta y}\,\mathrm{sinc}\left(\frac{x}{\Delta x}\right)\mathrm{sinc}\left(\frac{x}{\Delta y}\right)$	$\mathrm{rect}\left(\frac{\xi_1}{2\xi_{x0}}\right)\mathrm{rect}\left(\frac{\xi_2}{2\xi_{y0}}\right)$	

Figure 4.13 Image interpolation functions; $\xi_{x0} \triangleq \frac{\Delta}{2\Delta x}$, $\xi_{y0} \triangleq \frac{\Delta}{2\Delta y}$.

Seen from the frequency domain a we remember from filter theory that a *rect* function (far right in the last row of table 1) constitutes the best anti-imaging filter preserving high frequency detail and cutting out undesired image frequencies. However, from

fundamental signal processing we remember that multiplication with a rect function in the frequency domain is equivalent to convolving with a *sinc* function in the spatial domain. Of practical reasons, it is undesirable to perform the convolution with the sinc function as the interpolation kernel since this (ideally) would we an infinite task. It is therefore necessary to choose an algorithm that uses an interpolation kernel that approximates the sinc function sufficiently well, or other interpolation kernels that give good enough filtering.

4.4.2 Nearest neighbor replication

Referring to table 1, the simplest kernel is *rectangle (zero order hold)*, often referred to *as nearest neighbor replication*. The spatial and its frequency domain counterpart characteristics are also shown in the first row in table 1. With this interpolation kernel each interpolated pixel is assigned the value of the nearest pixel in the image array. The nearest neighbor is, as the name suggests, any of the adjacent pixels. Equation 5 gives the interpolation kernel:

Equation 5: Nearest neighbor replication.

$$h(x) = \begin{cases} 1 & 0 \leq |x| \leq 0.5 \\ 0 & 0.5 < |x| \end{cases}$$

The nearest neighbor replication is a simple algorithm where no arithmetic operations need to be performed. The operations involved are load and store operation. An easy way to implement this algorithm is to consider blocks of 2 x 2 pixels consisting of the least repeating pattern. From theory, this kernel is expected to require little computational effort. Also, from the frequency characteristics found in table 1, theory suggests that high frequency will be preserved, but with possible image frequencies. Though, since we are copying the value from the nearest pixel, we expect ‘jaggies’ – stairway patterns – to occure.

4.4.3 Bilinear interpolation

The next step in complexity is linear interpolation, or *first order hold* as referred to in Jain (table 1). Linear interpolation in 2-D is also often referred to as bilinear interpolation. That is, we perform a *bilinear interpolation* when we interpolates in 2-D. Pratt [Pratt ,1991 p. 441-445] has shown that linear interpolation in 2-D is the same as interpolation with a pyramid function. The mathematical description of the linear interpolation kernel used is a linear function given in equation 6, and its spatial and frequency domain characteristics are shown in the second row in table 1:

Equation 6: Linear interpolation.

$$h(x) = \begin{cases} 1 - x & 0 \leq x \leq 1 \\ 0 & 1 < x \end{cases}$$

Care needs to be taken when implementing this algorithm. It is convenient just summing the adjacent pixels and divide by two. However, division is not a part of the ARM/THUMB instruction set. Implementing division, however, can be performed in

an elegant way if the denominator is a power of 2. If so, a division is simply a shift operation (which is taken care of by the ARM compiler).
From the frequency characteristics (far right in the second row in table 1), we observe a narrower amplitude response than for the nearest neighbor replication. This suggests a more 'blurry' (less sharp) picture than the nearest neighbor replication since high frequency detail is cut out. However, we will expect less imaging frequencies. Furthermore, we expect no 'jaggies' since we are no longer only replicating pixel values.

4.4.4 Cubic convolution kernel

The two preceding interpolation kernels were both very simple. If we want better quality, literature suggests a number of different kernels. Referring to Jain, next order of complexity is *nth order hold* functions. Here we find *cubic splines* that through literature have shown themselves as the superior interpolation functions regarding visual quality. The drawback for most of these kernels, including cubic splines, is that they are computational expensive. To find an alternative to the *cubic splines*, I turn to work by Keys [Keys 1981] in the area of digital still cameras. Keys deduce a *cubic convolution kernel* composed of piecewise cubic polynomials. The piecewise cubic polynomials constituting the cubic convolution interpolation kernel are defined by the subintervals:

$$< -2,-1 >, < -1,0 >, < 0,1 >, < 1,2 >$$

Outside these intervals, that is:

$$<\leftarrow,-2 >, < 2,\rightarrow>$$

the interpolation kernel is zero. In general, the cubic convolution kernel can be given in parametric form in terms of *a* as [Keys 1981]:

Equation 7. Cubic convolution kernel, parametric form.

$$h(x) = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1 & 0 < |x| < 1 \\ a|x|^3 - 5|x|^2 + 8a|x| - 4a & 1 < |x| < 2 \\ 0 & 2 < |x| \end{cases}$$

Now, I will not go into detail of how Keys arguments for the value of the parameter *a*, rather I present the outcome of his work. The interested reader should refer to [Keys 1981] or [Pratt, 1991, p.115] for the argumentation. Keys represent interpolation as in equation 8.

Equation 8. Keys representation of interpolation.

$$g(x) = \sum_k c_k h(\frac{x - x_k}{n})$$

Here n represents sampling increment, x_k represents the interpolation nodes, h is the interpolation kernel and g is the interpolated function. The c_k is parameters that depend on the sampled data.

- **c_k is simply replaced by the sampled data.** Keys states that this is a substantial computational improvement over interpolation schemes such as the method of cubic splines. The spline interpolation kernel is not zero for nonzero integers. As a result ck must be determined by solving a matrix problem.
- Cubic splines slopes of the ends of the interpolation function are not guaranteed to be zero. This leads to amplitude ripples in a reconstruction function.

Furthermore, we remember from previous conclusions in this thesis, it is the sampled and quantized interpolation kernels we are going to use in an implementation of a FIR filter. Sampling and quantization of *cubic splines* do not give coefficients with nice fractions, and not at all fractions with denominators at a power of two. From the discussion of typical hardware architecture, we remember that this implies that the function would work well on architecture with floating-point support, but for mobile terminals with typically only C-library floating support or SW floating-point emulator it would imply intensive computations. The nice feature of the method of cubic convolution is that by upsampling by a power of two always will produce coefficients as fractions with denominators as a power of two. This gives an additional computational advantage over the cubic splines.

Keys work yields the cubic interpolation kernel in equation 9. Keys work proposes setting a = -0.5, which provides an interpolation function that approximates the original unsampled image to as a high degree as possible in the sense of a power series expansion. Figure 13 shows the interpolation kernel h (x) (which is symmetrical about the vertical axis).

Equation 9: Cubic convolution kernel. (Keys).

$$h(x) = \begin{cases} \frac{3}{2}|x|^3 - \frac{5}{2}|x|^2 + 1 & 0 < |x| < 1 \\ -\frac{1}{2}|x|^3 + \frac{5}{2}|x|^2 - 4|x| + 2 & 1 < |x| < 2 \\ 0 & 2 < |x| \end{cases}$$

Keys work has found the order of accuracy of the cubic convolution method to be between that of linear interpolation and cubic splines. I therefore expect the method of cubic convolution to better preserve high frequency detail whilst cutting out undesired imaging frequencies. The spatial domain similarity to the sinc function also suggests less ‘jaggies’ than for nearest neighbor replication and bilinear interpolation.

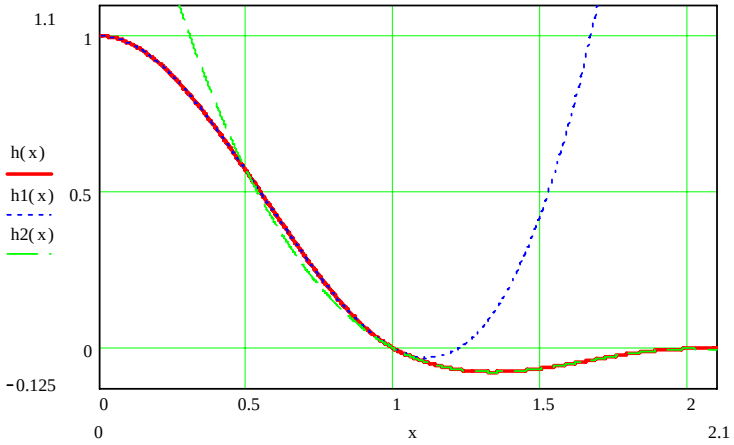


Figure 13. Cubic convolution kernel. (Keys).

4.4.5 Windowed sinc functions

From fundamental signal processing a common approach to the infinite number of coefficients required realizing a sinc function, is to use *windowed sinc* functions. Common window functions as Bartlett, cosine and Hanning are not included based on the work done by Turkowski [Turkowski 1990]. Turkowski concluded that the Lanczos-windowed sinc function offered the best compromise in terms of reduction of imaging and sharpness in the digital image.

Equation 10: Lanczos interpolation kernel

[Turkowski 1990]

$$h(x) = \sin c(x) \times \sin c(\frac{x}{3})$$

From the same reason as for the method of cubic splines (unsuitable coefficients), the implementation of the Lanczos kernel is left out, and so are other windowed sinc functions as Bartlett, cosine and Hanning since these kernels are found to give poorer results than the Lanczos kernel. Though, cubic convolution is spared since spatial domain characteristics for the Lanczos function is similar to the characteristics of the method of cubic convolution, only giving less suitable coefficients when sampled and quantized.

4.5 Implementing the demosaicing operation

4.5.1 General

Now that we have found the interpolation kernels to use, and we have seen that it will give suitable coefficients for the FIR filter, we turn to look at how we should implement the FIR filter. The FIR filter is characterized by the transfer function, H(z) given by equation 11.

Equation 11: Transfer function for FIR filter.

$$H(z) = \sum_{n=0}^{N-1} h(n) z^{-n}$$

We need to choose a *realization structure* for the implementation, which essentially is a block (or flow) diagram representation of one of the different theoretically equivalent ways in which the transfer function can be arranged. Examples of realization structures suitable for this thesis are:

- Fast convolution.
- Transversal structure.
- Linear phase structure.

4.5.2 Fast convolution

The interpolation functions are to be sampled at the rate corresponding to the pixel sample data rate. That is, if we want to include more pixels the number of samples, between the intervals defined for the function, need to increase. Notice that not only do we include more information by taking more pixels into consideration, but we also increase the number of FIR coefficients. Increasing the number of coefficients obviously gives a filter with a steeper slope and thus a better result. For large size kernels, in many cases, it may be more computationally efficient to perform the interpolation indirectly by Fourier transform filtering in the form of fast linear convolution (figure 14) rather than direct convolution. Ifeachor and Jervis [Ifeachor and Jervis, 1993] conclude that the direct method requires N^2 real multiplications while the method of fast convolution requires $12N\log_2 2N + 8N$ multiplications.

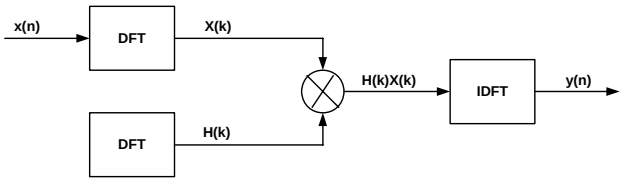


Figure 14. Illustration of fast convolution.

[Ifeachor and Jervis 1993, Table 4.1 p. 224] demonstrate the number of multiplication necessary for the convolution of two N-point sequences. Here it is shown that fast convolving is faster than the direct method for sequences containing in excess of 128 data points. However, it would be impractical for the typical HW found in mobile terminals to use to long sequences. We have to remember that one of the sequences is the actual pixel values, occupying place in cache. If the number of pixels causes a situation where the hit rate for the cache drops, this will have a severe negative impact on the performance of the algorithm. How many pixels that are reasonable to include in order to increase image quality must be compared to the computational effort. Also remember that fast convolution requires an FFT adding complexity to the image operation (the interpolation). Furthermore, it would be a highly time consuming, and computational intensive task to perform if we were to use 128 data points or more.

4.5.3 Transversal and linear phase realization structure

From the above discussion, fast convolution was found inefficient when only a few pixels were involved in the convolution sequences. This leaves us with transversal and linear phase structure. Initially we consider the transversal structure since this realization structure has a difference equation (equation 12 [Ifeachor and Jervis, 1993 p 344 equ. 6.33]) similar to the general interpolation equation.

Equation 12: Difference equation for transversal structure.

$$y(n) = \sum_{m=0}^{N-1} h(m)x(n-m)$$

This structure is used in most DSP processors. Though, in typical HW for mobile terminal we cannot rely on a DSP processor available since this is fully occupied with other tasks. Futher we remember from previous discussion that we did not want to include DSP coprocessor due to increased size of the ASIC embedding the ARM processor. For the typical HW in mobile terminals discussed and presented in this thesis, a more suitable and efficient realization structure is the linear phase filter structure. For the filter of the kind shown in figure 15, the transfer function can be written as equation 13 [Ifeachor and Jervis, 1993 p 345, equ. 6.34b], and the corresponding difference equation is given by equation 14 [Ifeachor and Jervis, 1993 p 345, equ. 6.35b].

Equation 13: Transfer function for linear phase structure FIR filter.

$$H(z) = \sum_{n=0}^{(N-1)/2-1} h(n)\left[z^{-n} + z^{-(N-1-n)}\right] - h\left(\frac{N-1}{2}\right)z^{-(N-1)/2}$$

Equation 14: Difference equation for linear phase structure FIR filter.

$$y(n) = \sum_{k=0}^{(N-1)/2-1} h(k)\{x(n-k) + x[n-(N-1-k)]\} + h\left[\frac{(N-1)}{2}\right]x\left[n-\frac{(N-1)}{2}\right]$$

A comparison of equation 12 and equation 14 shows that the linear phase structure is computational more efficient, requiring approximately half the number of multiplications and additions.

4.5.4 The FIR filter coefficients

Using the cubic convolution kernel as example, I find the coefficients for the FIR filter.

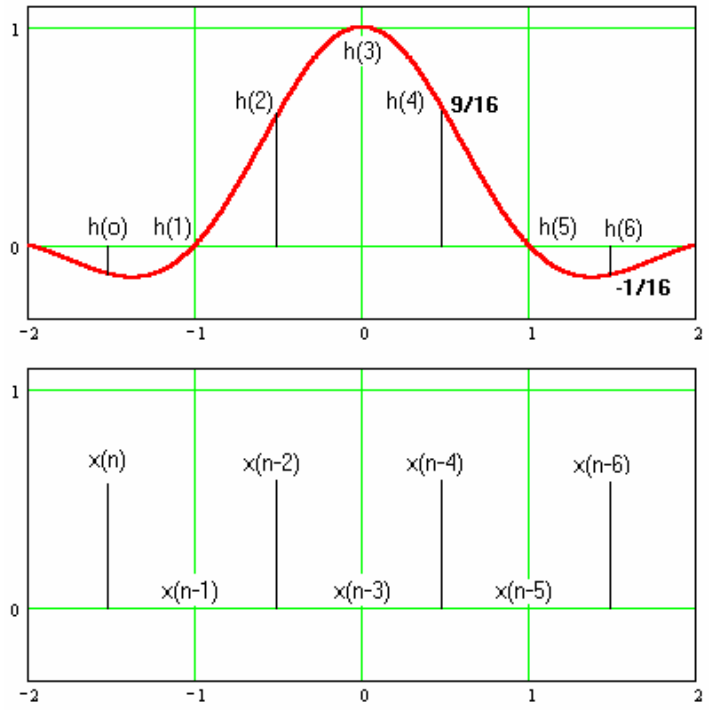


Figure 15. Sampled cubic convolution kernel (Keys) and neighboring pixels.

The upper part of figure 15 shows the cubic convolution interpolation kernel and the sampling points using Keys coefficients, and the lower part shows the neighboring pixels (with zero inserted samples). An illustration of the realization of a 7-point FIR is shown in figure 16. Using equation 14 yields equation 15 for implementation of the FIR filter with coefficients corresponding to the samples from Keys cubic convolution kernel in figure 15.

Equation 15: Expression used for implementation.

$$y(0) = h(0)\left[x(0) + x(-6)\right] + h(2)\left[x(-2) + x(-4)\right]$$

$$y(0) = \frac{-1}{16}\left[x(0) + x(-6)\right] + \frac{9}{16}\left[x(-2) + x(-4)\right]$$

This expression leaves us with two multiplications and two additions. This has to be done for every pixel we want to interpolate the value for. As seen from figure 15 and figure 16, the coefficients to the FIR filter are implementing cubic convolution (based on Keys results) are: h_{cub} {-/16, 0, 9/16, 1, 9/16, 0, -1/16}

Similarly, we can deduce the FIR filter coefficients for the bilinear interpolation kernel: h_{lin} {1/2, 1, 1/2}.

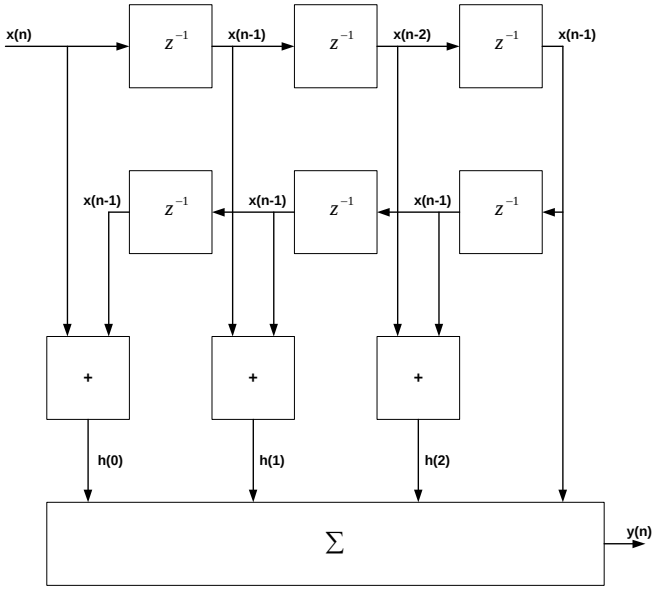


Figure 16. Linear phase structure for 7-point FIR filter.

When finding interpolating values, the image is divided into smaller sub blocks so the number of pixels values to include in the convolution task is limited to a few. For the nearest neighbor replication algorithm only the nearest pixel in is included. For the linear interpolation algorithm, the two neighboring pixels (in each direction) are used for interpolation. Corresponding to this number of pixels are a FIR filter with 3-coefficients for both dimensions. Moving on to the cubic convolution kernel the minimum number of pixels to include are the two nearest neighbors on each side in each direction. This corresponds to a 7-point (since the pixels have interleaved zeros) FIR filter for both directions.

When considering computational effort, we will expect the nearest neighbor replication to least computational expensive since only load and store operations are involved. The bilinear is expected to be more computational expensive due to multiplications and additions. The cubic convolution is expected to be the most computational expensive interpolation kernel since it minimum has to be realized as a 7-point FIR filter while it for the bilinear is sufficient with three coefficients. Furthermore, the cubic convolution filter requires arithmetic to prevent that over- and under-flow to appear due to the coefficient values 9/16 and –1/16. However, if more coefficients were included, the difference in computational effort between the bilinear and cubic convolution will decrease.

5 Measuring performance and cost for the demosaicing

5.1 Introduction

The motivation for the following discussion is to define a way to measure performance and cost of the implementation of the image operation. This part should preserve validity for test results put forward in this thesis. With this motivation, it is the scope to find a simple way of measuring the quality and cost for the different implementations of the image operation.

It is often not expedient to measure images quality per operation. E.g. if an advanced algorithm is used to output excellent RGB image from a Bayer pattern, and a poor RGB->YUV transformation is performed, the effect of the advanced algorithm may not be visible. In this thesis the operation is measured individually and isolated from the rest of the camera system in order to reduce scope of the work, and a possible system designer would then again need to choose which algorithm to implement when considering the overall system performance.

5.2 Subjective and objective image quality

5.2.1 General

Subjective quality means the visual result. But what is good visual quality? Standard images and test patterns have been introduced in order to establish some common evaluation criteria. Though, subjective testing will always depend on the individual persons perception of the test image.

An objective quality measure will not be influenced by personal perception, and is a better way of *reporting* image quality. The images might be viewed on different media as printed paper in various formats, various resolution monitors often as compressed images etc. With this in mind, I'll concentrate on finding an objective quality measure to adopt.

The purpose for introducing an objective quality measure is to replace or supplement the subjective quality measure. Also, objective quality measures makes it simpler to re-examine the results put forward. The important thing is to use an objective quality measure that preserves the special feature(s) in the image that we want to measure. Applying a 'general' objective quality measure is of little or no use compared to subjective tests otherwise.

5.2.2 Image quality

Generally, when measuring image quality, we need to define what the 'quality' is. In this thesis image 'quality' is defined as [Gerdfelder and Muller, 1999]:

"The quality of a presentation intended for human perception is defined by the degree in which the communicative goal of the presentation is reached. The goal is to find an

optimal ratio between the information provided for the human observer, and the amount of information that is perceived by the human observer".

From this Gerdfelder and Muller state the following factors that determine the quality of every presentation (image, audio, video, etc.) [Gerdfelder and Muller, 1999]:

- *The human observer (their perception faculty, experience, etc.)*
- *The information to be coded*
- *The communicative goal*
- *The application area (the common metaphor, etc.)*
- *The presentation and interaction devices and tools*

In this thesis the human observer is the consumer with various level of experience. The information to be coded is still images of various real world motifs. The communicative goal is to reproduce and preserve the image information with good quality. The application area is still images stored in an appropriate way for e-mail attachments. The presentation and interaction devices are e.g. mobile terminals and PC's.

5.2.3 A model for measuring objective image quality

In objective measurements, image quality and quality losses in a digital image system is determined be evaluations based on a given general mathematical, physical or psycho physiological model. In subjective tests, the quality of a visual presentation image is determined form the performance of test persons in subjective physiological test. Most quality measurement methods are based on a comparison of the original and the processed image.

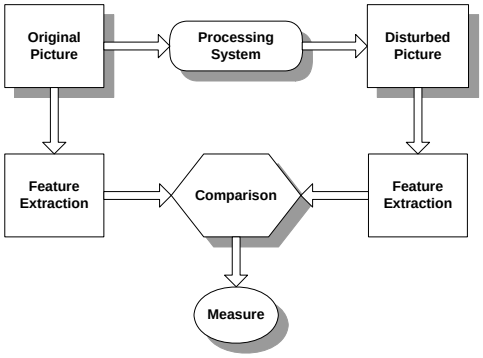


Figure 17. Model for measuring image quality.

In order to make a comparison, one needs to have a 'reference' image to compare to. This is ideally the real life motif, but in practice this is (for objective quality measuring) a high-resolution digital representation of the real life motif.

Gerdfelder and Muller introduce the model in figure 17 for measuring objective quality. The important issue from the discussion of quality measures is that the quality measure adopted must *extract* and reflect the feature(s) we want to measure. From the

previous discussions we have found that the demosaicing operation is an interpolation. This again is in the frequency domain the equivalent the same as low pass filtering the image pixel samples using a FIR filter. Using a filter we want to retain all spatial resolution, but exclude imaging frequencies. The algorithms used to find the missing information at each pixel will introduce 'distortion' compared to the 'reference' image. It is this distortion we want to measure in order to say how good the quality is. For overview I have a brief introduction to the features we want *extract* for quality measures for the demosaicing operation, namely imaging, 'jaggies' and the sharpness of the image.

5.2.4 Jaggies

The algorithms implemented in this thesis uses information from the neighboring pixels to decide the value for the missing colors. This approach will introduce a 'zipper' effect – or 'jaggies'. That is, lines and curves in the image will appear as a 'stairway' pattern. For the demosaicing operation, lines and curves are the features we want to extract and compare. This is what's referred to as feature extraction in figure 17. The reference image used in this thesis should therefore be chosen especially to emphasize this feature for simply extraction. That is, the image should include clearly defined curves.

5.2.5 Sharpness

Interpolation with the kernels discussed in this thesis constitutes a low pass filtering. Low pass filtering reduces the spatial bandwidth meaning that higher frequencies can be cut out. It should be apparent that removing high frequency contents would case the reproduced image to appear more "blurry" than the original. The reference image used in this thesis should therefore be chosen especially to emphasize this feature for simply extraction. That is, the image should contain high frequency information.

5.2.6 Imaging (post-aliasing)

From the mathematical discussion of the demosaicing operation I illustrated the spatial and frequency characteristics of *imaging*. But what does imaging look like in a picture? Figure 19 shows how imaging can occur in a picture with poor anti-imaging filter. The imaging is seen (especially down right) as rings within the picture. (Figure 18 shows the image without imaging artifacts).

5.2.7 Circular Zone Plate

In order to better extract the undesired features imaging, 'jaggies' and sharpness, I adopt the method of using a circular zone plate (CZP) as a reference image (figure 18). By using the CZP we can evaluate the difference in color restoration (for each color if desirable). The CZP implemented will have increasing frequencies moving away from the lower left corner in the image (0,0 for windows bmp). Making the image include circular curves will emphasis extracting the jaggies feature. Further, since the frequency is increasing, we can investigate the 'sharpness' and imaging features. (For information about implementation refer to the function *zone* in Appendix C – Software Listing under the file measure.c).

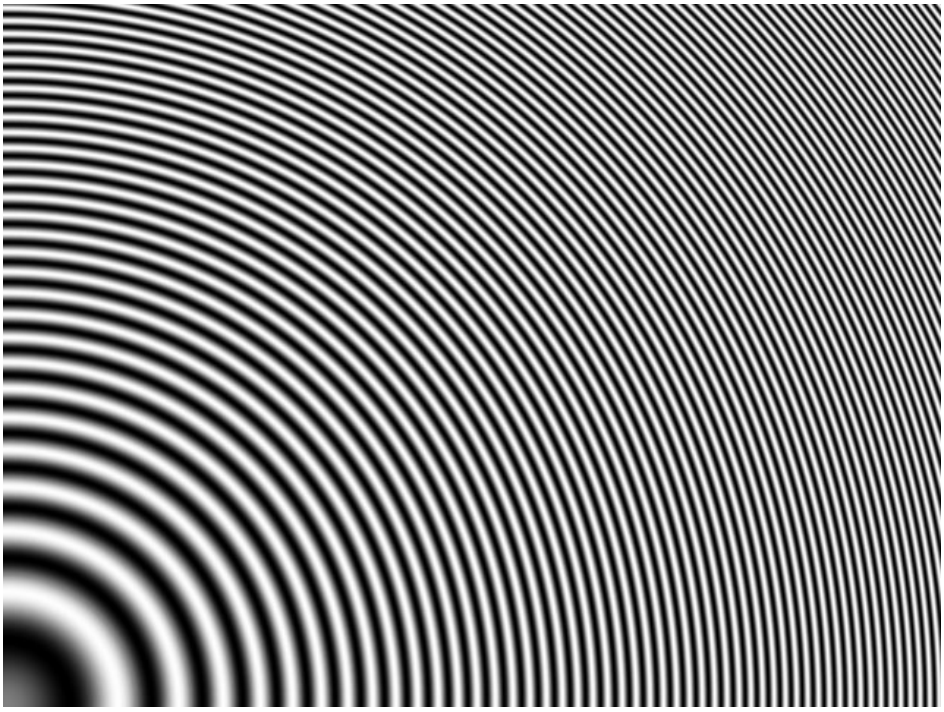


Figure 18. Circular Zone Plate

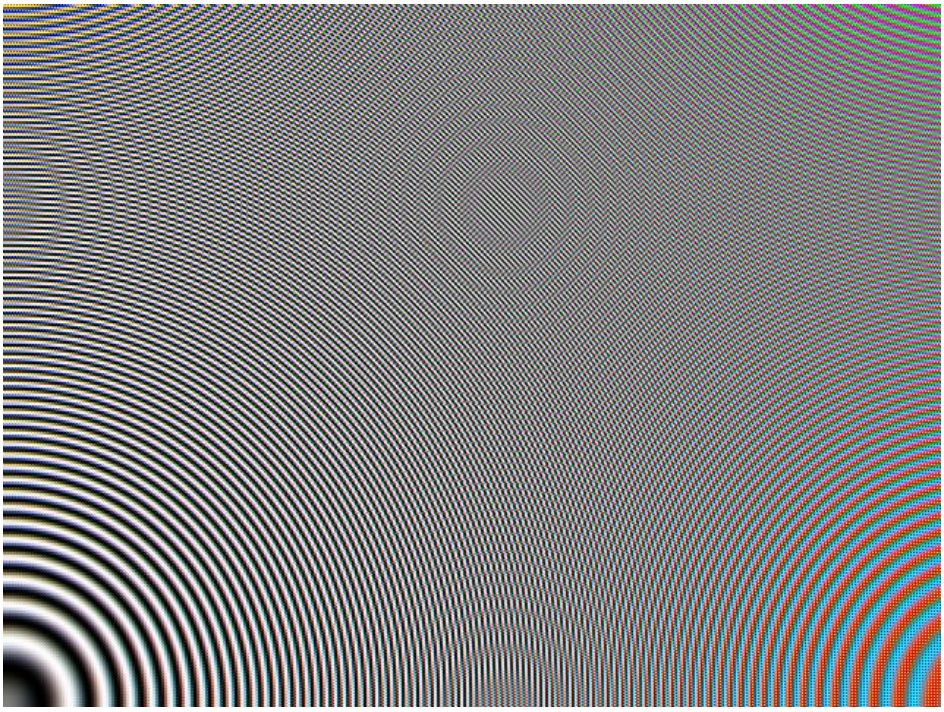


Figure 19. Circular Zone Plate with imaging artifacts.

5.2.8 Error images

An effective way for visually inspection of the image quality is to use error images. By looking at the difference between the original and the reproduced image, one can identify the quality of the reproduced image compared to the original. If there is a perfect match at the pixel, the pixel will be black. If the red, green and red values are completely wrong, the pixel will be white. The error image pixel values are simply the difference value between the original and the reproduced image.

5.2.9 Signal to noise ratio

A basic measure for objective evaluation is the normalized least square error (NLSE) (equation 16) of the pixel values. In order to make NLSE more computational efficient, the denominator is often replaced with the highest brightness quantization level to form the peak normalized least square error (PLNSE).

Equation 16. Normalized Least Square Error

$$\delta_{NLSE} = \frac{\sum_{y=1}^{M}\sum_{x=1}^{N}\left[f(x,y) - f(x,y)'\right]^2}{\sum_{y=1}^{M}\sum_{x=1}^{N} f(x,y)'^2}$$

The numerator in PNLSE and NLSE represent the power spectrum of the image noise, and the denominator describes the power spectrum of the distorted (reproduced) image. We now have a signal to noise ratio for the reproduced image (normalized to the number of pixels):

Equation 17. Signal to Noise Ratio (SNR)

[Sakamoto et.al. 1998].

$$SNR = -10\log_{10}\left[\frac{\frac{1}{KL}\sum_{k=0}^{K-l}\sum_{l=0}^{L-1}\{f(k,l) - f'(k,l)\}^2}{255^2}\right] \quad [\text{dB}]$$

Where KxL is the spatial resolution (e.g VGA = 640 x 480), f(k,l) is the original image and f'(k,l) is the reproduced image. The color depth is 8 bit, so the denominator is $2^{(8\text{-}1)*2} = 255^2$.

5.2.10 The objective quality measure adopted

The quality of the reproduces image is given as a SNR. To emphasize the features of imaging, jaggies and sharpness I use a CZP. The SNR will most likely change according to test image. A uniform test image will make little or none difference in SNR for the different interpolation kernels. That is, an image with little high frequency content and little curves will be reproduced with little visual distinguishing between the images. In addition to the features jaggies and sharpness will the possible imaging will contribute to the SNR. Since these rings are visual, they are also measurable and will contribute to a poorer SNR if present.

5.3 Computational cost

5.3.1 General

As found in work by [Sakamoto et.al, 1998], evaluating different algorithms for embedded systems should also include a measure for computational cost since it is of importance to reduce power consumption. The power consumption can be reflected through the number of processor clock cycles an operation requires to finish. A measure for computational effort will also reflect the power consumption.

5.3.2 MIPS

Computational cost is often measured in MIPS, - Million Instructions Per Second. If we counted the number of instructions for the code implementing a particular operation we would have an efficiency measurement for that code. One obvious thought is that we could calculate the time this operation would occupy the processor simply by dividing the number of instructions by the processors MIPS. This is not always the case due to the fact that one instruction not necessarily can be carried out by only one clock cycle. The ARM7 processors employ a simple 3-stage pipeline with the following pipeline stages:

- Fetch; the instruction is fetched from memory and placed in the instruction pipeline.
- Decode; the instruction is decoded and the data path control signals prepared for the next cycle. In this stage the instruction ‘owns’ the decode logic but not the data path.
- Execute; the instruction ‘owns’ the data path; the register bank is read, an operand shifted, the ALU result generated and written back into a destination register.

For single cycle instructions the pipeline enables one instruction to be completed every clock cycle. An individual instruction takes three clock cycles to complete, so it has a three-cycle latency, but the throughput is one instruction per cycle. Though, some instructions take more than one cycle. Typically load and store instructions takes more than one cycle. Further, the memory may not work at the same speed as the processor. This implies that the memory system needs to be implemented using wait states. One wait state means that we have to wait one clock cycle for a memory access.

5.3.3 Clock cycles per pixel

A more suitable cost measure is therefore to measure the number of clock cycles needed to process one pixel. (This approach was also used by [Sakamoto et.al. 1998]). This can be performed in an easy way simply by measuring the time the operation occupy the processor, and divide this number by the processors clock frequency. Further, this number can be divided by the spatial resolution in order to find the number of cycles used to process one pixel. Lower overhead means less computational effort needed. Also, when this measure is used, we no longer have to think of how to extract the number of instructions from the C code.

6 Test method and environment

6.1 Introduction

This chapter presents the test equipment - both HW and SW. My intention is to present enough information for later reproduction of results put forward. The code development has been divided into two implementations, one for ARMulator (emulator for ARM processors in ARM SDT v.250), and one for Ericsson Development Board (EDB). The reason for this was the limited time schedule for the thesis. The limited time forced removing as many practical problems as possible and focus on getting results produced. By dividing the code development into two SW builds, I could use one build for visual verification and objective quality measures (ARMulator build, - or actually any SDT since cost measures are not an issue), and the other build to perform cost measures (EDB build). Doing so released me from implementing the memory model and memory interface in ARMulator, and also opportunity to run on real HW. Though, I give enough information for readers to implement a memory model and memory interface for ARMulator or real HW to re-examine the results put forward.

6.2 The ARMulator build

6.2.1 General

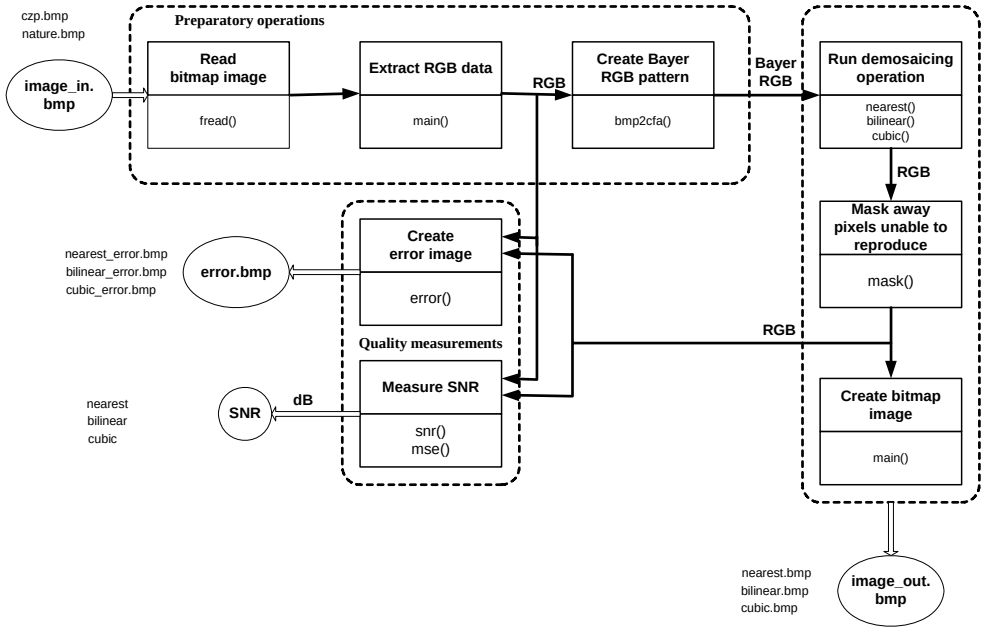


Figure 20. Overview of SW build.

Figure 20 is an illustration of the complete ARMulator build. The image read is chosen to emphasize feature we want to extract. The feature is extracted by subtracting the reproduced image data from the original image data. The ARMulator build is used for visual verification and objective quality measures. The build also

creates Bayer pattern RGB from a bitmap file (windows .bmp) by removing two out of three color in every pixel position. The build can also create a CZP for measurement purposes. Further, the build masks border pixels that can't be interpolated for correct SNR measures.

Below follows a short description of the files and functions in the ARMulator build. The complete code listing is included in Appendix C. Under the consumption taken in this thesis, the mobile terminal uses a color filter array that reduces the color resolution. Due to this, it is not straightforward to make a comparison since there are not digital reference image to compare to. In this thesis, the reference (original) image is produced taking a high quality digital image of bitmap format and remove two colors from each pixel in a defined manner, - in this thesis using a Bayer patter color filter array. This way an image similar to the digital image delivered from the CMOS image sensor is created. (The same approach was used by [Wu et.al., 1997]

6.2.2 Short description of files in ARMulator build

definitions.h
A header file defining different sizes of the input image to read is included. The file needs to be edited if the spatial resolution is to be changed.

main.c
A bitmap (windows bmp) input image are read from file and put into a buffer. The header and data are put in separate buffers. Possibly, a CZP image can be created if wanted. The content in the data buffer are then transformed to Bayer RGB data and put into another buffer by removing two out of three colors at every pixel position. There is also created a '*bayer_out.bmp*' for visual inspection of the content in the Bayer data buffer. Now we have Bayer data and can start the real job, - namely interpolating back to full resolution RGB. Testing different kernels are done by adding or removing functions implementing the from the '*main.c*' file. For all tested kernels the SNR are measured, and an output image together with an error image are created as windows bmp files. Notice that the header used to create these pictures is the header copied from the initial image read from file.

mosaic.c
This file includes the *bmp2cfa* function for means to convert bitmap files with full resolution RGB into Bayer RGB. Its input is the read files data content, and it outputs Bayer CFA data together with data bitmap content for visual inspection.

demosaic.c
Here the core of the work in the thesis lies. In this file the interpolation operation is performed. The implementation of the kernels lies here as the function *nearest*, *bilinear* and *cubic*. The interpolation is carried out calling the function from *main* to the actual kernel.

measure.c
The measure.c file includes the function *mse* and *snr* for measuring the SNR for the different kernels. The *snr* are called from *main*, and *snr* calls *mse* again. The *mse* function calculates the Mean Square Error (MSE) between the original and the reproduced image. This measure is used to calculate PNLSE, and later SNR. When interpolating, dependent of the kernel, there are boarder pixels that can't be interpolated. These pixels need to be masked away before the SNR calculations for correct results. The *mask* simply writes zeros into these positions in the buffers containing the original and reproduced image data. Thus, when subtracting these positions there are no contributions to the SNR. For visual inspection and also debugging, a function for creating an error image (*error*) is included. Additionally, this file includes the function *zone* for creating CZP image data.

6.3 Ericsson Development Board (EDB)

The build for the EDB is identical to the ARMulator build with the following exceptions:

- No means for performing quality measures (were not interested in doing this at real HW since it's already done in the ARMulator build), but means for measuring clock cycles for a function (without getting interrupted).
- The different functions are embedded into the simple existing SW build on the board.

The code for the EDB build is not included since the reader would only have little or none use of this information since the functions described in the ARMulator build are scattered throughout the proprietary SW build for the EDB.

For reproduction of test results the reader should implement a HW with the characteristics described in table 2. The implementation could either be a complete ARMulator model consisting of CPU (ARM710T model is included in ARM SDT 2.50), memory model and memory interface (these have to be specified according to datasheet for SDRAM specified in table 2), or in real HW. For information of how to implement the model refer to [ARM, Ref. Guide 1998, chapter 12]

Table 2. Ericsson Development Card characteristics.

Component	Actual	Characteristic	
Processor	ARM710T	CPU frequency	65 MHz
		8kb mixed cache	
		16 word[1)] write buffer	
SDRAM	K4S641632C-TC/L10	Memory frequency	65MHz
		CAS latency[2)]	2
		t_{RRD}[2)]	20 ns
		t_{RAS}[2)]	50 ns
		t_{RP}[2)]	24 ns
		t_{RCD}[2)]	24 ns

[1)] Word = 32bit
[2)] Refer to [Samsung K4S641632C] for explanation.

7 Results

7.1 Introduction

This chapter puts forward the results of the performance and cost measures from practical tests.

7.2 Test results

7.2.1 Performance and cost measures

In table 3 the performance and cost measures are put forward for the different interpolation kernels. The overhead [cycles/pixels] was measured for each 'RGB' pixel, and must be divided by three to indicate each R, G or B pixel-processing overhead. Further, the measurement is measured when code was run from SDRAM and with and without cache.

Table 3 presents the objective quality measurements and the cost for the different interpolation kernels.

Table 3. Test results.

Algorithm	Overhead [cycles/pixel] with cache	Overhead [cycles/pixel] without cache	SNR [dB] 'czp'	SNR [dB] 'nature'
Nearest neighbor	68	238	17,70	23,81
Bilinear interpolation	85	310	25,36	27,46
Cubic convolution	136	466	30,27	27,55

7.2.2 Test images and reproduced images.

The test images and the presentation of the reproduction by the different interpolation kernels are included as Appendix A for convenience. The images need to shown as bitmap files in order to maintain correct visual reproduction. It is of most importance that these images are not compressed (using a lossy algorithm) if difference in quality is to be visible.

The reader should be aware that printing the test results in Appendix A is not straight forwarded. Remember that the printer not necessarily will reproduce the images properly without care taken. The printer might interpolate or decimate the image data for its own good (refer to Appendix B for further information). The best viewing media is a monitor and an image processing software capable of showing windows 24-bit bitmap files with VGA resolution.

For each of the two test images, three images representing the reproduction for each of the interpolation kernels are included. Furthermore, the error images showing the difference between the original image and the reproduced images are included.

8 Discussion

8.1 Introduction

From the work done in this thesis several issues have arisen that need to be discussed:

- The test results and their correspondence to literature review
- The test results validity.
- Which interpolation kernel is most suited for mobile terminals based on literature review and test results?
- How does the work fit into the overall camera system?
- Can the work be applied in other areas than just the demosaicing operation?
- Other issues that has arisen through the work.

8.2 Test results

8.2.1 General

This chapter discusses the test results put forward and their correspondance to literature review.

First, if we look at the results for reproducing the CZP images (picture 3, picture 4 and picture 5 in Appendix A), we find a satisfactory coherence between the literature review performed and the test results. The SNR number in it selves provide little information of the quality, but when considering them, and comparing them to each other, we can see that the cubic convolution kernel gives the best performance. This is also confirmed through the error images. From the error images for the nearest neighbor replication (picture 6 in Appendix A), bilinear interpolation (picture 7 in Appendix A) and the cubic convolution (picture 8 in Appendix A), one can easily see that the cubic convolution has less error compared to nearest neighbor and bilinear from its larger black area. We also notice that for bilinear and cubic convolution it is at the sharp edges the errors are apparent. This is especially true when the frequency increases towards the upper right corner. Considering picture 6 and 11 in Appendix A we find the ‘jaggies’ as expected from literature review. Also we find that the nearest neighbor replication have errors spread over the whole image due to the replication of pixel values.

Also the *color* in the error image gives us valuable information. The errors mainly appear as magenta. From additive color mixing we know that this means mixing blue and red – hence we know that red and blue color planes have more errors than green. This makes sense since the red and blue color planes are subsampled in both horizontal and vertical direction.

8.2.2 Aliasing or imaging artifacts?

When considering the error images, we see undesired artifacts (rings) in the images. This is most apparent for the nearest neighbor, but is also visible in the error images for the bilinear interpolation and the cubic convolution. The question is; is this aliasing or imaging? They will both occur in the same manner in a digital image. In the SW build I created a Bayer RGB array from a bitmap file simply by throwing two out of three colors in each pixel. That is, I performed a hazardous decimation with no low pass filtering before the decimator! Discovering this, I performed a "manually" low pass filtering of the image by ensuring that the image did not contain frequencies that exceeded the Nyquist frequency (after decimation) to ensure that no aliasing occurred from the decimation process. In picture 2 in Appendix A we can see that no aliasing artifacts are present in the zero inserted Bayer RGB CZP. The artifacts visible in the error images are therefore image frequencies due to poor anti-imaging filter in the interpolator.

8.2.3 The interpolation kernel and the input image

When considering the test image 'nature', we see as expected that the SNR for the different kernels are more equal. I say expected because the large uniform sky is easy to recreate and hence introduce no errors. Which interpolation kernel that is used here is therefore unessential. This type of information can be used in adaptive algorithms. If the image has uniform areas, nearest neighbor replication or bilinear interpolation kernel is sufficient. Though, when details are to be reproduced cubic convolution kernel should be used. The motivation behind this is to reduce power consumption. The results put forward in table 3 shows that the cubic convolution kernel is twice as expensive as nearest neighbor replication.

8.2.4 Computational effort

Table 3 puts forward the cost measure for the processor in form of number of clock cycles/pixel to perform the demosaicing job. The number is normalized to reflect the job to perform the operation on RGB values – since each pixel is represented by red, green and blue values. From table 3 we find that substantial cost saving is accomplished in form of the cache. The code requires approximately 2,5 times less computational effort with cache present. The necessity to keep the pixel blocks small should therefore be apparent. If to many pixels and FIR coefficients are included there might not be enough space in cache and thus a decreased hit rate leading to increased computational effort.

Relating the measure to the time spent performing the demosaicing operation, we can calculate the time spent to produce a VGA (640 x 480) image on an ARM710T with 8 kb cache and SDRAM external memory @ 65 MHz with. The times would be 0.32 seconds for nearest neighbor replication, 0.40 seconds for the bilinear interpolation and 0.64 seconds for the cubic convolution. If we consider a display for a mobile terminal, the resolution can typically be QQVGA (160 x 120) the times are 0.02, 0.025 and 0.04 seconds respectively.

8.3 Test result validity

Have the measures adopted shown themselves suitable measuring the objective quality and computational cost? From the overall conformity between literature review, discussion and practical results I would say; yes. Though, criticism has been raised through the years claiming measures as SNR, MSE and NLSE are meaningless. Though, this criticism is pointed to the measures when looking at the complete camera system. For this thesis this has not been the issue since I am only concentrating on optimizing this single operation for the HW. On the contrary, the SNR combined with a suitable test image for feature extraction (enhancement) has shown to produce results according to the literature review and discussion.
Still, this operation is to coexist with other operations such as compression, and marginal differences in quality might not even be visual when considering the complete camera system.

The implementations of the different kernels do not take general code optimizing into account. The code optimizing is likely to decrease the number of clock cycles/pixel for the different interpolation kernels, but is expected to give the same reduction for all kernels since implementations only have minor differences. Also effort paid to the data structure of the image data and how these are stored in memory for efficient memory accesses and cache line fills are likely to reduce clock cycles/pixel for the implementations. Again, these savings are expected to give the same reduction for all three kernels.

8.4 Which interpolation kernel is best suited for the mobile terminal?

8.4.1 General

Now that the results are presented a question remains; which interpolation kernel is best suited for use in mobile terminals? As a beginning, the answer will depend on the requirements of the application. So far I have discussed the image operation demosaicing isolated from the camera system, and with no real time requirements. Trying to give an answer to the question in the heading, I use two application as examples – still images and video clips for e-mail attachments.

8.4.2 Still images for e-mail attachments

The still image ‘mode’ is divided into two parts:

- Searching for the motif using real time viewfinder (the display).
- Acquiring the motif by ‘pressing the button.

For the viewfinder functionality, the time the image operations require to finish is an issue. For this purpose the demosaicing operation needs to be fast. The faster operations, the more frames per second (fps) are possible. However, the quality of the operation is less important due to poor display reproduction quality. For this purpose the nearest neighbor replication kernel is a suitable interpolation kernel. However, when the user has ‘pressed the button’ image quality is an issue, and the time consumption is of less importance. Now the cubic convolution kernel would be the

natural choice. If switching between the two interpolation kernels is undesired, bilinear interpolation is a good trade-off offering a visual quality near the cubic convolution and requiring only a computational effort close to the nearest neighbor replication. Moreover, the image is to be compressed, and the compression scheme is most likely to be JPEG (Joint Photographic Experts Group). Also, considering the GPRS terminals today, typically only one time slot is offered for uploading of data offering a slow data rate (worst case 9,6kbps) hard compression is likely to be used in order to reduce image data size. With this in mind, the most suitable kernel would be the bilinear interpolation since the high frequency detail surviving the anti-imaging filter is in some extent removed again by the JPEG codec. Though, new storage media as MultiMedia Card and Memory Stick offer simple means for transferring data between the mobile terminals and PC's. With these solutions the user might want the uncompressed image stored, then the cubic convolution kernel would be preferable again. (Another issue is that the user might *want* to compress the images to fit as many images into the storage medium and therefore image quality is less important).

8.4.3 Video clips as e-mail attachment

Recording video clips we have a situation somewhat similar to the viewfinder functionality in the still image 'mode'. The difference is that now the operation needs to be both fast and offer good visual quality. The use for better quality is due to the possibility to send the clip as an e-mail attachment to be viewed e.g. at a high resolution PC monitor. However, the compression used scheme (e.g. MPEG-4) will again in some extent suggest the choice of interpolation kernel. If the visual improvement is not visible at the peer, then the use of the cubic convolution kernel might be unnecessary. Again, as for the still image application, the bilinear interpolation kernel might be the best trade-off between visual performance and computational effort.

8.5 Other application areas for the results

8.5.1 Downsampling

Mathematically, interpolation and the decimation operation are duals of each other [Ifeachor and Jervis, 1993, p. 494-498]. The duality property means that a decimator can readily be derived from an interpolator equivalent and vice versa. Following this property – the anti-aliasing filter prior to the decimator can use the same kernels for decimation as the interpolator uses for its anti-imaging filter. Hence, e.g. the downsampling operation in the camera system in figure 4 of the YUV to YUV 4:2:0 can apply the different kernels found in this thesis according to the trade-off between visual quality and computational effort as it did for the demosaicing process.

8.5.2 Scaling

The demosaicing operation is about pixel interpolation for the different color planes. Now, up-scaling and down-scaling is the same operation as demosaicing only considering all color planes together. That is, scaling is an interpolation (up-scaling) or a decimation (down-scaling) operation. Again, as a trade-off between visual quality and computational effort an interpolation or decimation kernel needs to be chosen.

8.6 A suggestion to a simplified image pre-processing chain

8.6.1 General

The motivation for this part is to discuss some issues that have arisen from previous discussions and work in this thesis. The discussion here takes a more general look at the whole pre-processing chain using the work from of the demosaicing operation.

8.6.2 Using the nature of Bayer pattern decimation

Considering the current image pre-processing chain (figure 21), we find duplication of interpolation and decimation schemes. In this discussion I argue for a simplified pre-processing chain for mobile terminals.

We remember from previous discussions that the CMOS image sensor output is a subsampled RGB image (RGB BAYER). The demosaicing operation interpolates this subsampled Bayer pattern to full RGB. Then there is a RGB to YUV color conversion. The main reason for this conversion is that e.g. JPEG uses YUV color space as input for compression. After this, a new decimation is performed, now in the YUV color space. After compression and decompression, we need to interpolate back to full YUV info again. Then, a new color space conversion back to full RGB resolution.

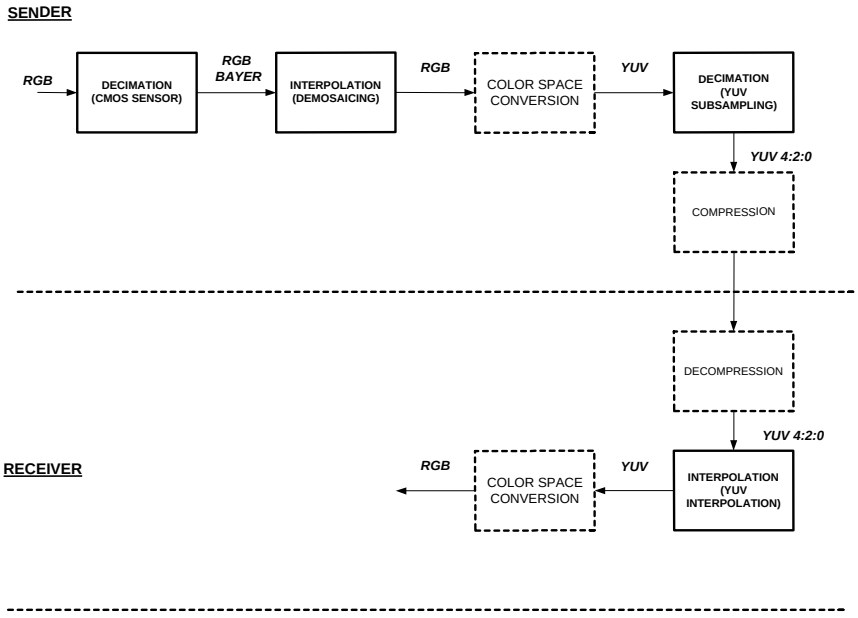


Figure 21. Interpolation and decimation in the pre-processing chain today.

The Bayer pattern is based upon the same thoughts used for the YUV color model. That is, the image can be represented as luminance and chrominance information. The Bayer pattern CFA is made on the approximation that green represent luminance

information, and red and blue represent chrominance information. Since the eye is less sensitive for chrominance information than luminance information, chrominance information can be subsampled more than luminance information without any visual degradation in image quality. This feature is exploited both in YUV subsampling and the Bayer pattern. That is, chrominance information is subsampled at a higher rate than luminance information. The most common subsampling used for YUV subsampling is YUV 4:2:0, which means that the luminance information is not subsampled, the chrominance information is subsampled in both vertical and horizontal direction. This leaves us with half the data size (of the full RGB resolution). The Bayer pattern gives a reduction in data size to one third of the full RGB resolution.

From the similarities in the YUV color model and the Bayer pattern, I suggest a simplified block diagram for the interpolation and decimation operations in the pre-processing chain in figure 22. Here we consequently remain in the RGB color space. This way we save two color space conversions that include multiplications and additions. Furthermore, we save one interpolation (the demosaicing) and one decimation (the YUV downsampling). Using the simplified pre-processing chain in figure 22, the mobile terminal only needs to run the demosaicing operation on the receiver side. This way both time and power consumption is saved.

The drawback for this scenario is that JPEG uses YUV color space as input. The switching to RGB color space might not be straight forwarded, but I leave this obstacle for later research. Further research has to look into how the JPEG compression scheme will be affected when RGB are input instead of YUV. How will the visual quality be affected and how hard can we compress are central questions. A simple way getting around the whole problem is changing the compression format. The drawback for this approach is a proprietary system with reduced area of application.

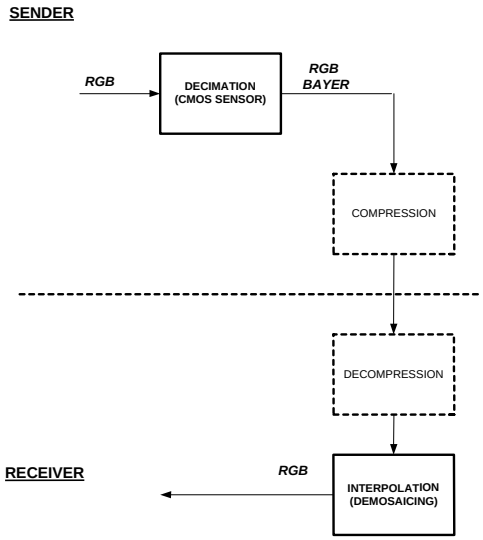


Figure 22. Simplified interpolation and decimation in the pre-processing chain.

8.7 Frequency domain versus spatial domain processing

Although this thesis has worked with the demosaicing operation isolated, the operation is to be implemented as part of an image processing chain. In compression schemes as JPEG, a frequency transformation as DCT for frequency domain transformation is already present, and this enables processing in frequency domain. Also, the processor experiences continuously increased execution speed that might enable applying more coefficients in the FIR filter and thereby including more pixels for better visual result. Thus the pixel block used in the demosaicing operation might be more efficient to process in frequency domain.

Knowing this, further research should look into whether it is efficient to perform more of the image pre-processing operations in frequency domain. If we also combined operations in order to reduce the number of memory accesses, noticeable reduction in computational effort and power consumption may be achieved. How this would be affected when JPEG 2000 (using wavelets instead of DCT) becomes common should also be investigated.

9 Conclusion

Work has shown that the demosaicing operation is equivalent to the mathematical operation of interpolation, and that optimizing potential lie within the implementation of the anti-imaging filter for the interpolator. This filter can be realized as a FIR filter with the filter coefficients obtained from a suitable interpolation function (kernel).

My work has investigated the nearest neighbor replication, bilinear and cubic convolution interpolation kernels from literature review. Furthermore I, have implemented and measured these kernels with respect to performance (SNR) and cost (clock cycles/pixel on typical hardware found in mobile terminals today.

Considering the interpolation kernels isolated from the pre-processing chain, the cubic convolution kernel offers the highest image quality (SNR = 30,27 dB for CZP) requiring only moderate computational effort (136 clock cycles/pixel at 65 MHz). The quality improvement gained using cubic convolution lies in preservation of high frequency details and reduced anti-imaging. Though, the demosaicing operation is to coexist with other image operations possibly masking away the visual improvement from the demosaicing operation (as e.g. compression). With this in mind, the bilinear interpolation kernel is the most suited interpolation kernel of the three kernels offering a good trade-off between image quality (SNR = 27,46 dB for CZP) and computational effort (85 clock cycles/pixel at 65 MHz). The simplest interpolation kernel, the nearest neighbor replication offers the poorest image quality (SNR = 23,81 dB for CZP) and should not be used unless the system lacks computational capacity. The nearest neighbor replication interpolation kernel only requires 68 clock cycles/pixel at 65 MHz.

Further work should put effort into investigate the performance of the different interpolation kernel when considering the whole camera system. Furthermore, research should look into potential cost savings when image operations are combined resulting in fewer memory accesses. Also, since the JPEG compression in the camera system uses a DCT, research should look into possible cost savings if processing was carried out in the frequency domain.

10 Abbreviations

1-D	-	One-Dimensional
2-D	-	Two-Dimensional
2,5G	-	2,5'd Generation
3G	-	Third Generation
ARM	-	Advanced RISC Machine
ASIC	-	Application Specific Integrated Circuit
CFA	-	Color Filter Array
CIF	-	Common Interchange Format
CISC	-	Complex Instruction Set Computer
CMOS	-	Complementary Metal Oxide Semiconductor
CPU	-	Central Processing Unit
CZP	-	Circular Zone Plate
DAC	-	Digital to Analog Converter
DCT	-	Discrete Cosine Transform
DFT	-	Discrete Fourier Transform
DSP	-	Digital Signal Processor
FIR	-	Finite Impulse Response
fps	-	frames per second
GPRS	-	General Packet Radio Service
HW	-	Hardware
IDFT	-	Inverse Discrete Fourier Transform
IP	-	Intellectual Property
JEDEC	-	Joint Electron Device Engineering Council
JPEG	-	Joint Photographic Experts Group
Kbps	-	Kilo bytes per second
LPF	-	Low Pass Filter
MIPS	-	Million Instructions Per Second
MMS	-	Multimedia Messaging Service
MPEG	-	Motion Picture Experts Group
MSE	-	Mean Square Error
NLSE	-	Normalized Least Square Error
PC	-	Personal Computer
PDA	-	Personal Digital Assistant
PNLSE	-	Peak Normalized Least Square Error
RGB	-	Red, Green and Blue
RISC	-	Reduced Instruction Set Computer
SDT	-	Software Development Toolkit
SW	-	Software
SRAM	-	Synchronous Random Access Memory
SDRAM	-	Synchronous Dynamic Random Access Memory
SNR	-	Signal-to-Noise Ratio
UMTS	-	Universal Mobile Telecommunication System
VGA	-	Video Graphics Array

11 References

[Baxes, 1994] Baxes, G.A., "Digital image processing - principles and application", John Wiley & Sons 1994, ISBN: 0471009490.

[Gonzales and Woods, 1992] Gonzalez, Woods, "Digital Image Processing", Addison-Wesley Pub Co 1992, ISBN: 0201508036.

[Jain, 1989] Jain, A.K.,"Fundamentals of Digital Image Processing", Prentice Hall 1989, ISBN: 0133361659.

[Ifeachor and Jervis, 1993] Ifeachor, E.C., Jervis, B.W., "Digital Signal Processing – A Practical Approach", Addison-Wesley 1993, ISBN: 020154413X.

[Defatta et al, 1988] Defatta, D.J., Lucas, J.G., Hodgkiss, W.S., "Digital Signal Processing – A System Design Approach", John Wiley & Sons 1988, ISBN: 047183788.

[Pratt, 1991] Pratt,W.K., "Digital Image Processing – Second Edition", John Wiley & Sons 1991, ISBN:0471857661.

[Furber, 1996] Furber, S., "ARM System Architecture", Addison-Wesley 1996, ISBN: 0201403528.

[Kernighan and Ritchie, 1989] Kernighan, B.W., Ritchie, D.W., "Programmeringsspråket C, 2. opplag", Tano Prentice Hall 1989, ISBN: 8251827051. (In Norwegian).

[Sakamoto et al, 1998] Sakamoto, T., Nakanishi, C., Hase, T., "Software Pixel Interpolation for Digital Still Cameras Suitable for 32-Bit MCU", IEEE Transactions on Consumer Electronics, Vol. 44, No. 4, November 1998, p. 1342-1352.

[Keys, 1981] Keys, R.G., "Cubic Convolution Interpolation for Digital Image Processing", IEEE Transactions on Acoustics, Speech and Digital Signal Processing, Vol. ASSP-29, No. 6, December 1981, p. 1153-1160.

[Rifman and McKinnon, 1974] Rifman, S.S., McKinnon, D.M., "Evaluation of Digital Correction Techniques for ERTS Images – Final Report", Report 20634-6003-TU00. TRW Systems, Redondo Brach, Calif., July 1974.

[Adams, 1995] Adams, J.E., "Interactions between color plane interpolation and other image processing functions in electronic photography", Proceedings of SPIE Vol. 2416, p. 144-151, 1995.

[Adams, 1997] Adams, J.E., "Design of Practical Color Filter Array Interpolation Algorithms for digital cameras", Proceedings of SPIE, Vol. 3028, p. 117-125, 1997.

[Hou et al, 1987] Hou, Hsieh S. et.al., "Cubic Splines for Image Interpolation and Digital Filtering", IEEE Transactions on Acoustics, Speech and Signal Processing", Vol.ASSP-26 P.508-517, 1987.

[Wu et.al. 1997] Wu, X., Choi, W.K., Bao, P., "Color Restoration from Digital Camera Data by Pattern Matching", Proceedings of SPIE, Vol. 3018, p. 12-17, 1997.

[Gerdfelder and Muller, 1999] Gerdfelder, N. and Muller, W., "Objective quality estimation for digital images in multimedia environments", in Color Imaging: Vision and Technology, edited by L.W. MacDonald and M.R. Luo, John Wiley & Sons Ltd. 1999, ISBN: 0471985317.

[Turkowski, 1990] Turkowski, K., "Filters for common Resampling Tasks", Graphics Gems I, April 1990 p 147-165. (http://www.worldserver.com/turk/computergraphics/ResamplingFilters.pdf)

[Gunasekara, 1996] Gunasekara, O., "Wireless applications for compact RISC mcus", in Electronic Product Design, May 1996, p. 72-77.

[Allison, 2001] Allison, A., "Inside The New Computer Industry", January 2001. (http://www.arm.com/news.ns4/iwpList125/)

[ARM, Ref. Guide 1998] ARM, "ARM Software Development Toolkit Version 2.50, Reference Guide", Nov 1998.

[www.arm.com] ARM Limited home page.

[ARM, ARM710T] [ARM, “ARM 710T Datasheet”, ARM DDI 0086B, July 1998]

[ARM, App. Note 34] ARM, “Writing Efficient C for ARM”, Application Note 34, ARM DAI 0034A, January 1998.

[Samsung K4S641632C] Datasheet for SDRAM used in real HW (K4S641632C-TC/L10).

[RKN, 2001] Personal conversation with HW team leader Rune Knutsen at Ericsson AS in Grimstad 26.03.2001.