1

Introduction of Real-time Image Processing

1.1. General image processing presentation

The traditional view of an image derives heavily from experience in photography, television and the like. This means that an image is a two-dimensional (2D) structure, a representation and also a structure with meaning to a visual response system. This view of an image only accepts spatial variation (a static image). In parallel, a dynamic image has spatial and temporal variation. In most contexts, this is usually referred to as video. This more complex structure needs to be viewed as a sequence of images each representing a particular instance in time. On the other hand, an image can be formed by taking a sampling plane through that volume and so the variation in three dimensions is observed. This may be referred to as a volume image. An image linked to a volume that changes with time is a further possibility. This has particular significance in medical imaging applications [MYE 09].

Image processing is a method to convert an image into digital form and perform some operations on it in order to get an improved image or to extract some useful information from it. A digital image described in a 2D space is usually

considered as 2D signals while applying already set signal methods to it. Image processing forms a core research area within engineering and computer science too.

Image processing is an enabling technology for a wide range of applications including remote sensing, security, image data sets, digital television, robotics and medical imaging, etc. The goal of this technology can be divided into three levels: low, medium and high. Low-level image processing techniques are mathematical or logical operators that perform simple processing tasks. This "processing" level possesses both *image_in* and *image_out*. Medium-level image processing combines the simple low-level operators to perform feature extraction and pattern recognition functions. Using an *image_in*, this "analysis" level produces *measurements_out* (parameters). High-level image processing uses combinations of medium-level functions to perform interpretation. This "understanding" level treats *measurements_in* for high-level *description_out*. Usually, apart from these three levels, it is also necessary to apply preprocessing techniques that are designed to remove distortions introduced by sensors.

Figure 1.1. *Overview of the typical image acquisition process (see [MOE 12])*

Before any image processing can commence, an image must be captured by a camera and converted into a manageable entity. This is the image acquisition process (see Figure 1.1), which consists of three steps; energy reflected from the object of interest, an optical system that focuses on the energy and finally a sensor that measures the amount of energy. In order to capture an image, a camera requires

some sort of measurable energy. The energy of interest in this context is light or electromagnetic waves. Each wave can have different wavelengths (or different energy levels or different frequencies). The human visual spectrum is in the range of approximately 400–700 nm.

After having illuminated the object of interest, the light reflected from the object now has to be captured by the camera composed of an optical system and an image sensor. The role of the first is to focus the light reflected from the object onto the second (a material sensitive to the reflected light). An image sensor consists of a 2D array of cells. Each of these cells is denoted a pixel and is capable of measuring the amount of incident light and convert that into a voltage, which in turn is converted into a digital number. The more incident light, the higher the voltage and the higher the digital number.

In order to transform the information from the sensor into an image, each cell content is now converted into a pixel value in the range: (0, 255). Such a value is interpreted as the amount of light hitting a cell. This is denoted the intensity of a pixel. It is visualized as a shade of gray denoted gray-level value ranging from black (0) to white (255). Standardly, a monochromatic, static image corresponds to a matrix of m rows and n columns. Therefore, the camera records $m \times n$ pixels of 8 bit values.

In order to capture a color image, the color camera must record three matrices of three primary colors red, green and blue. Recently, a lot of applications are realized using multispectral image processing, since the multispectral cameras are more available with reasonable prices. According to application needs, multispectral images are captured using different wavelengths (bands). They can be considered as a cube formed by 2D gray-level images. Figure 1.2 displays two typical test images in the area of image processing research. Figure 1.3 gives two

multispectral cube examples; the right image is captured for a skin lesion assessment application.

Figure 1.2. *Lena (gray-level image) and landscape (color image). For a color version of the figure, see www.iste.co.uk/li/image.zip*

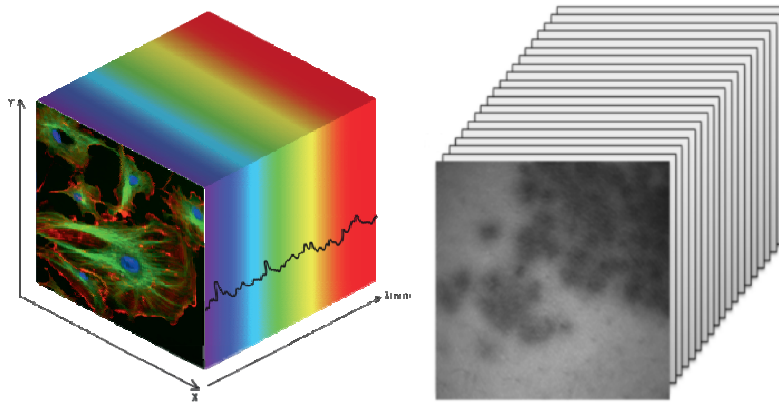

Figure 1.3. *Two multispectral image cubes. For a color version of the figure, see www.iste.co.uk/li/image.zip*

Certain tools are central to the processing of digital images. These include mathematical tools, statistical descriptions and manipulative tools. We can cite some more used such as *convolution*, *filtering in spatial and frequency domains*, *morphological operations* and *image transforms*,

etc. The types of basic operations that can be applied to digital images to transform an input image A(m, n) into an output image B(m, n) (or another representation) can be classified into three categories:

– Point: the output value at a specific matrix coordinate is dependent only on the input value at that same coordinate. In this case, generic operation complexity per pixel is constant.

– Local: the output value at a specific coordinate is dependent on the input values in the *neighborhood* of that same coordinate. In most cases, the type of neighborhood is rectangular with 8-connected (3 × 3 mask) or 24-connected (5 × 5 mask pixels). The complexity per pixel is proportional to the square of neighborhood size.

– Global: the output value at a specific coordinate is dependent on all the values in the input image. The complexity per pixel is equal to $N \times N$ (image size = N).

The complexity per pixel of each operation type participates in the total complexity of an image processing chain for a target application. This total complexity per image decides the processing speed (time performance).

1.2. Real-time image processing

Real-time image processing is the subfield of image processing focused on producing and analyzing images in real time. Generally, a real-time system has the following three interpretations within different senses [KEH 06]:

– real time in the perceptual sense, which is used to describe the interaction between a human and a computer device for a near instantaneous response of the device to an input by a human user;

– real time in the software engineering sense, which is used to describe a concept of bounded response time in the computer device. With this constraint, the device must satisfy both the correctness of the outputs and their timeliness;

– real time in the signal processing sense, which is used to describe the constraint within which the computer device has to complete processing in the time available between successive input samples.

Since image processing is a subfield of signal processing, in this book we base the interpretation of "real time" on the signal processing sense.

In order to satisfy the constraint of "real time", Kehtarnavaz [KEH 11] points out that the total instruction count of a real-time algorithm must be "less than the number of instructions that can be executed between two consecutive samples", and Ackenhusen *et al*. [ACK 99] describe the "real-time processing" as a computation of "a certain number of operations upon a required amount of input data within a specified interval of time" as well. Therefore, the key technique of real-time image processing is to ensure that the amount of time for completing all the requisite transferring and processing of image data is less than the allotted time for processing. For example, if the algorithm is aimed at an entire frame (a static image) and the frame frequency of the system is 30 frames per second (fps), the processing of a single frame should be finished during 33 ms.

Real-time image processing systems involve processing vast amounts of image data in a timely manner for the purpose of extracting useful information, which could mean anything from obtaining an enhanced image to intelligent scene analysis. Digital images and video are essentially multidimensional signals and are thus quite data intensive, requiring a significant amount of computation and memory resources for their processing. A common theme in real-time image processing systems is how to deal with their vast

amount of processing and computations. The key to cope with this issue is the concept of parallel processing, a concept well known to those working in the architecture area who deal with computations on large data sets [KEH 06]. In fact, much of what goes into implementing an efficient image processing system centers on how well the implementation, in both hardware and software, exploits different forms of parallelism in an algorithm, which can be data level parallelism (DLP) and/or instruction level parallelism (ILP). DLP manifests itself in the application of the same operation on different sets of data, while ILP manifests itself in scheduling the simultaneous execution of multiple independent operations in a pipeline fashion.

Low-level image processing transforms image data to another image data. This means that such operators deal directly with image matrix data at the pixel level. Examples of such operations include color transformations, linear or nonlinear filtering, noise reduction and frequency domain transformations. In this low-level processing, one can observe three operation categories. Point operations are the simplest since a given input pixel is transformed into an output pixel, where the transformation does not depend on any of the pixels surrounding the input pixel. Local neighborhood operations are more complex in that the transformation from an input pixel to an output pixel depends on a neighborhood of the input pixel. Such operations include 2D spatial convolution and filtering, smoothing, sharpening, etc. These operations require a large amount of computations. Finally, global operations build upon neighborhood operations in which a single output pixel depends on every pixel in the input image. An example of such an operation is the discrete Fourier transform that depends on the entire image. These operations are quite data intensive as well.

All low-level image operations involve nested looping through all the pixels in an input image with the innermost loop applying a point, neighborhood or global operator to the pixels forming an output image. Therefore, these are fairly data-intensive operations, with highly structured and predictable processing, requiring a high bandwidth for accessing image data. In general, low-level operations are excellent candidates for exploiting DLP.

Medium-level operations transform image data to a slightly more abstract form of information by extracting certain features from an image. This means that such operations also deal with the image at the pixel level for input, but the transformations involved cause a reduction in the amount of data from input to output. Medium-level operations primarily include segmenting an image into regions/objects of interest or extracting edges, lines, or other image attributes such as statistical features. The goal these operations is to reduce the amount of data to form a set of features suitable for further high-level processing [KEH 06]. Some medium-level operations are also data intensive with a regular processing structure, thus making them suitable candidates for exploiting DLP.

High-level operations interpret the abstract data from the medium level, performing high-level knowledge-based scene analysis on a reduced amount of data. Such operations include classification/recognition of objects or a control decision based on some extracted features. These types of operations are usually characterized by control or branch-intensive operations. Thus, they are less data intensive and more inherently sequential rather than parallel. Due to their irregular structure and low bandwidth requirements, such operations are suitable candidates for exploiting ILP, although their data-intensive portions usually include some form of matrix-vector operations that are suitable for exploiting DLP.

From the above discussion, one can see that there is a wide range of diversity in image processing. A typical image processing chain combines the three levels of operations into a complete system, as shown in Figure 1.4, where top shows the image processing chain and bottom shows the decrease in the amount of data from the start of the chain to the end for an N × N image with P bits of precision. The diversity of operations in image processing leads to the understanding that a single processor might not be suitable for a real-time image processing algorithm implementation. A more appropriate solution would thus couple multiple computation components of different characteristics [KEH 06].

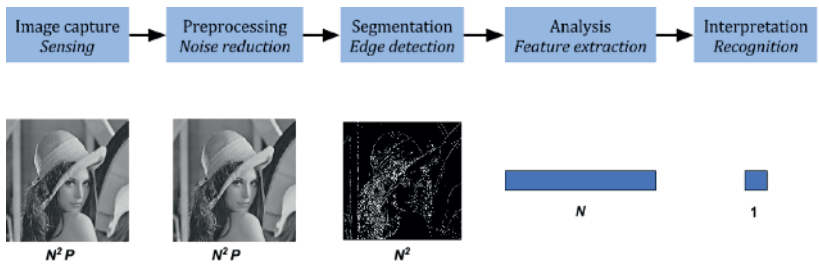

Figure 1.4. *Diversity of operations in image processing: typical processing chain (top) and decrease in amount of data across processing chain (bottom)*

Bearing in mind the above argument, developing a real-time processing system can be quite a challenge. The solution often ends up as some combination of hardware and software approaches. From the hardware point of view, the challenge is to determine what kind of hardware is best suited for a given image processing task among a myriad of available hardware platform choices. From the algorithm and/or software point of view, the challenge involves being able to guarantee that "real-time" deadlines are met, which could involve making choices between different algorithms based on computational complexity, algorithm optimization in the parallel execution sense, using a real-time operating system, and extracting

accurate timing measurements from the entire system by profiling the developed algorithm [KEH 06].

In the real-time image processing area, embedded systems are often involved. A precise definition of an embedded system is not easy. Simply stated, all computing systems other than general purpose computers (with monitor, keyboard, etc.) are embedded systems. An embedded system has software embedded into hardware, which makes a system dedicated to an application or a specific part of an application. It is an engineering artifact involving computation that is subject to physical constraints (reaction constraints and execution constraints) arising through interactions of computational processes with the physical world. Embedded image processing systems are typically designed to meet real-time constraints; a real-time system reacts to stimuli from the controlled object/operator within the time interval dictated by the environment.

When the complexity of the image processing applications leads to a high ratio between its computation volume and its reaction time, standard off-the-shelf sequential architectures are inadequate. Parallel, distributed and multicore architectures are required. Programming such architectures is an order of magnitude harder than with uniprocessor sequential ones, and even more so when architecture resources must be minimized to match cost, power and volume constraints required for embedded applications.

In this context, hardware platform selection and careful and fine-tuned application programming/development environments become more and more important. At the same time, the application market is quickly spreading which reduces the time available for the design of individual applications. These facts increase the demand of rapid prototyping of real-time image processing. The rapid prototyping of complex parallel real-time embedded applications is essentially based on the software/hardware

co-design. This notion is knows in two senses; for multicomponent architecture, this software/hardware (SW/HW) co-design step distributes some parts of the applications to processors by running software, while other parts must be implemented by hardware running on specific integrated circuits. On the other hand, SW/HW co-design usually means application design using both SW/HW development environments.

Designing real-time image processing systems is a challenging task indeed. Practical issues of speed, accuracy, robustness, adaptability, flexibility and total system cost are important aspects of an embedded system design. In practice, one usually has to trade one aspect for another. Since the design parameters depend on each other, the trade-off analysis can be viewed as a system optimization problem in a multidimensional space with various constraint curves and surfaces. This design space exploration task, from a mathematical viewpoint, consists of determining optimal design working points.

Today, we are at a crossroad in the development of real-time image processing systems. The advancements in integrated circuit technology make it now feasible to put to practical use the rich theoretical results obtained by the image processing community. In spite of the fact that the value of an algorithm hinges upon the ease with which it can be placed into practical use, the implementation challenges involved have often discouraged researchers from pursuing the idea further, leaving it to hardware experts to implement a practical version in real time. The purpose of the following chapters is to facilitate this task by providing a broad overview of the advanced strategies/tools for rapid prototyping of real-time image processing system.

The rest of the book is organized as follows: Chapter 2 describes commonly used hardware architectures for real-time image processing. After a comparison of the currently

available platforms and their development environments, we concentrate on rapid prototyping of image processing based on field programmable gate array (FPGA) technology. Chapter 3 presents research results about enabling the reconfigurable instruction set processor model by exploiting FPGA technology for discrete cosine transform algorithm implementation. High-level synthesis (HLS) technique is introduced in Chapter 4 with a skin lesion assessment application as an illustration example. In Chapter 5, we propose a novel source-to-source compilation strategy in order to improve HLS design performances. This CDMS4HLS technique is tested and validated by the embedded implementation of very high resolution satellite image segmentation in Chapter 6. Chapter 7 examines real-time image processing with very high level synthesis.