Tiling and demand driven evaluation for picture processing

Patrick Horain and Victor Dogaru

Ecole Nationale Supérieure des Télécommunications / Dép. Images(*)
46, rue Barrault, 75634 Paris Cedex 13, France

ABSTRACT

We propose a software architecture for picture processing with three objectives.

Images are represented as a set of tiles that are computed on demand, i.e. when needed as arguments to some operator. An image is indeed an operator instance that includes a call back function to respond to tile demands. Tiling avoids to store whole images. Demand driven evaluation shortens the time intermediate results must be kept in memory before they are used in further processing. This allows efficient memory management when many operators are applied to large images.

Second, this architecture is a path towards automated parallelization. The kernel layer can spread the computation of image tiles to several processing threads.

At last, application development should not be made more complex in comparison to classical techniques. Applications are written as functional programs that describe relationships between images. Writing a new operator class can mostly be done by copying other operator classes ; only the call back function for a tile has to be different. This is demonstrated with sample code.

Several tiling methods with and without overlapping, possibly with an adjunct region of interest, are discussed in function of the access mode : a pixel at random, linear tracking, sequential over a sub-image, or sequential over the whole image.

Strategies to choose image tiles to be destroyed when the cache is full are also proposed : the first created, the oldest referenced, the cheapest to compute, or a combination of the last two criteria.

This is well suited for neighborhood operators such as convolutions and mathematical morphology operators.

Keywords : Demand driven evaluation, tiling, memory management, parallelism, image processing.

(*) Telephone : +33 1 45 81 75 81, fax : +33 1 45 81 37 94, e-mail : horain@enst.fr.

Tiling and demand driven evaluation for picture processing

Patrick Horain and Victor Dogaru

Ecole Nationale Supérieure des Télécommunications / Dép. Images
46, rue Barrault, 75634 Paris Cedex 13, France

ABSTRACT

We propose a software architecture for picture processing that allows efficient memory management when algorithms with many operators are applied to large images, and that allows automated parallelization. This architecture relies on image tiling and operators with a call back function that evaluate image tiles on demand. Several tiling strategies with and without overlapping are discussed. The complexity of this evaluation strategy is hidden to application programs. This is shown with a sample program. This architecture is well suited for neighborhood operators such as convolutions and mathematical morphology operators.

Keywords : Demand driven evaluation, tiling, memory management, parallelism, image processing.

1. INTRODUCTION

Picture processing applications are often designed as a sequence of operators that transform the original image into intermediate images, up to the result image. A typical example would be for instance a low pass filter followed by edge detection, followed by contour linking, etc. Such processing of whole images is conceptually simple, but has several drawbacks. It will induce either sub-image copies or a waste of processing power and memory if the user is not interested in the whole image, but only in parts of it. Handling large intermediate images will require disk input/output operations (either explicitly or implicitly from the memory management mechanisms).

Furthermore, in spite of the fact that multi-processor general purpose workstations are getting more and more common, many picture processing scientists are not familiar with parallel programming.

The purpose of this study is to propose an efficient and easy to use software architecture for picture processing.
Our objective is threefold : efficient computing on dynamic regions of interest, efficient memory management of large intermediate images and data, and transparent parallel computing. These goals should be achieved while keeping application development easy.

1.1. Overview of the architecture proposed

A **functional program** defines an expression, *i.e.* the relationship between some objects that are combined together to produce another object[1, 2, 3]. It differs from the imperative approach that describes commands to be executed in a specified order. It does not use any side-effect in operators, so that complex expressions can be evaluated in any order.

This allows optimization based on the order of evaluation. For instance, **demand driven** (or **lazy**) **evaluation** consists in delaying the evaluation of expressions until their value is required (for instance to compute other expressions). Obviously, this leads to avoid unnecessary computations. Another advantage is that the demanded intermediate results are computed at a time close to the time they are used, which allows more efficient memory management.

Tiling is a technique that consists in manipulating images as a collection of regularly spaced non-overlapping rectangles that are all the same size. In the case of operators that need argument pixels just around a result pixel, all the argument pixels are in the same tile in the general case. Thus their locations in the system memory are closer, which allows to handle them more efficiently[4].

We propose a demand driven software architecture, where images are handled as sets of tiles, for efficient computation of complex algorithms applied on very large images.

1.1.1. Easy processing of regions of interest

Sometimes one needs only a part of a result image. The limits of this sub-image may be determined either interactively, or dynamically as the result of some processing on the whole image such as a multi-scale analysis or a filtering stage. Final processing should not be applied to the whole image, but only to regions of interest.

Processing a sub-image rather than the whole image can be done by limiting the demand to a set of tiles containing the region of interest.

1.1.2. Efficient memory management

With many picture analysis applications, the image size is growing bigger and bigger. Electronic archiving for fine arts, medical investigation, and satellite remote sensing use images up to 100 million pixels, or more. Modern computers can handle that much data as files on a mass storage, and/or as buffers in virtual memory. Many picture processing software[5], such as SPIDER[6] and Khoros[7], rely on that capability.

Demand driven evaluation allows to reduce the time between the moment when the result is available and the moment when it is used (consumed). This reduces the need for temporary storage, especially when used in conjunction with tiling of the image.

1.1.3. Multi-threading

Multi-threading is a facility offered by some operating systems (see [8] for example) that allows to split the execution control inside a process between several threads of execution. For instance, synchronous input/output can be handled simultaneously with computation in a single process, or several threads of computation can be executed in parallel on a multiprocessor platform.

Tiling allows to take advantage of multi-threads by computing each image tile in a different thread.

1.1.4. Keep development easy

An application program using picture processing operators should just describe the algorithm functional relationships between images. Tiling, memory management, demand driven evaluation and multithreading should be hidden to the application developer.

This can be achieved with a three layers software architecture.

1. The kernel layer implements memory management, demand driven evaluation of image tiles, and multithreading.
2. The operator layer describes the operator arguments, and implements the method to compute image data.
3. The application layer describes the algorithm as a set of functional dependencies.

1.2. Previous relevant works

Breuel[9] has discussed functional programming for computer vision, versus imperative programming. He proposes to represent images as functions, and uses a language primitive to create an array of pixels computed by a given function. One of his suggestions is to compute images on demand, with functions that could store previously computed results (pixel values), and retrieve them when called with the same arguments (pixel coordinates).
We have implemented such a technique. Instead of testing for each pixel whether it was stored previously, which is not efficient, we group pixels by the tiling mechanism.

Kozato and Otto[10] have proposed a lazy functional language for affine geometric transformations of images. Those transformations are described by a matrix ; an algorithm composed of such operators can be described by the resultant matrix. Geometric transformations are usually computed backwards. So laziness is simply achieved by applying the inverse composite matrix to each demanded pixel.
Although our architecture will be less efficient on geometric transformations, it can work with any image processing operators (such as convolution or mathematical morphology operators), not only with geometric transformations.

Silicon Graphics have developed the *Image Vision Library*[11], an object oriented, demand driven library for picture processing, that uses tiling to handle images[†].
Among the relevant works we know, this is the closest to our. However, demands on large image regions are handled by copying tiles of the argument images into " blocks " corresponding to the demanded region. This degrades performances. The architecture we propose suppresses or drastically reduces pixel copies because demand is transparently handled page by page, even if a larger region is needed.

2. SOFTWARE ARCHITECTURE

2.1. Tiles and pages

Each tile is associated with a computation unit called a **page** that is the elementary image region handled in demands. A page is centered on a tile. It may have exactly the same size, or it may be a bit larger and thus overlap other neighbor pages.

2.1.1. Non overlapping pages

In this case, pages exactly correspond to tiles. Each pixel belongs to only one page. Thus no pixel will happen to be computed several times. This is well suited for pixel to pixel operators that use only one pixel of argument image (or images) to compute one pixel of the result image.

Some other operators use a neighborhood of the pixel in the argument image(s) to compute the value of a pixel in the result image ; we call them **spatial** operators. To compute the value of pixels at the edge of a result page, spatial operators have to demand the argument page that contains those pixel plus one (or more) neighbor page. The number of pages demanded may grow rapidly when the demand is propagated towards source images because neighbor pages of neighbor pages get demanded. Retrieving the neighborhood of pixels at the edge of tiles requires to store all the neighbor pages. Furthermore, computing a pixel with neighbors in several pages requires copying that neighborhood in a contiguous memory zone, or requires more complex computing than for pixels inside the page.

2.1.2. Overlapping pages

To avoid those costs, pages can be made a bit larger than tiles. Most pixels belong to only one page, but those near tile edges are copied to two or four neighbor pages. Thus pixels at the edge of pages are computed several times in several pages. This induces a waste of computing.

In counterpart, neighborhood management can be made easier for spatial operators. If pages in the argument images are made larger than pages in the result image by the size of the neighborhood, then only one page of the argument image will be demanded to compute a page of the result image, and pixels at the edge of the page can be computed in the same way as those inside the page.

2.1.3. Non-overlapping pages with region of interest

Non-overlapping pages have the drawback to induce a number of demands for neighboring pages that grows when the demand is propagated from operator to operator towards the source images. A solution is to complement the demand for neighbor pages with the region of interest in it. Pixel out of that region may be filled with an arbitrary value because they will not be used. As long as the region of interest defined by all the neighborhoods does do not reach the other side of the neighbor page, then the demand is not extended to neighbor pages of neighbor pages.

2.1.4. Image lines as pages

A special case of non-overlapping pages is classical image lines. Sequential processing over the whole image will take advantage of demand driven evaluation on image lines because all the pixels are equivalently at edges of pages, so their neighborhood is always accessed the same way without any duplication of pixel values.

[†]In reference [11] a *tile* can be any part of an image. In the present paper, we use the word *tiling* to mean partitioning an image into regularly spaced non-overlapping rectangles that are all the same size.

2.1.5. Comparison

Those different paging strategies are evaluated according to three criteria.

- Efficiency, that is the ratio of the number of pixels of interest over the number of pixels computed.
- Minimum cache memory size required to compute a result page, that is the memory to hold pages of the argument image(s), possibly with their neighbor pages.
- Minimum number of pages in the cache, that is the length of the list that must be searched when a page is demanded before resorting to compute it.

Those parameters have been evaluated for different modes of access (*i.e.* demand) to the result image.

- Sequential processing of the whole image.
- Sequential processing of a sub-image.
- Linear displacement (*e.g.* road tracking).
- Pixel random access in the result image.

Dogaru has expressed the above evaluation criteria for each access mode as functions of the page size, of the eventual overlap type and size, and of the size of the region of interest. He has shown that line pages are more efficient when sequentially processing whole images, non-overlapping pages with region of interest when sequentially processing small sub-images, and overlapping pages in both the cases of random access and linear tracking[12].

2.2. Cache management

When a page of an image is computed, it is added to the list of pages available in a cache. Every time a page is demanded, the cache is first searched to check whether that page is available. If it is not, the operator call back function is used to compute it.

As the location of processing changes in the image, pages located elsewhere are no more needed. In order to keep the cache size under some given limit, “old” pages must be deleted while new pages are stored in memory.

2.2.1. Freeing memory policy

So far, we have used only a basic first in first out (FIFO) algorithm based on the creation order of pages to choose the next page to delete.

2.2.2. Page locking

When an operator demands several argument pages, pages that are available in the cache are marked locked. This prevents those pages from being deleted in order to free memory for the other argument pages demanded by the operator.

If all the pages are locked and the cache size has reached its limit, then this limit is raised up as necessary.

2.3. Demand driven operators

Because operators must respond to demands on image pages, they are implemented as objects. An image is an instance of the operator class that computes it. The operator includes a private function that is called back to compute demanded pages. When an image receives a demand for a page, that page is searched in the cache. If it is not already available, then the operator demands the argument pages to source images and computes the page. That page is finally returned to the demanding image. Thus the flow of demand is reverse of the data flow.

Demand driven evaluation insures that a page is computed at a time the algorithm needs it ("*just in time*"). Thus it will stay locked in the cache for a short period of time, whatever order the application declared intermediate images. In that way, demand driven evaluation contributes to memory management optimization.

2.4. Kernel classes for operators

Because operators are objects with embedded functions, the kernel and operator layers are written in an object oriented language. The popular C++ language has been chosen. Applications can be written in C++, and hooks are also provided to support applications written in C.

Images are instances of operator classes that are derived from kernel classes offered as an interface for the operator layer. Kernel classes are derived from a base class that provides methods to search the list of pages in the cache, and the locking mechanism. The kernel layer interface includes the following classes.

- The image file class is meant as a base for input output operators of various image file formats.
- Monadic and dyadic classes support demand mechanisms for operators with respectively one or two argument images.
- The spatial class supports neighbor pages management for spatial operators. This class is combined with the monadic and dyadic classes are available for spatial operators with one or two arguments.

3. EXAMPLE APPLICATION

Some operators have been implemented. For example, here is an application with an operator the output of which is demanded only for image pages that satisfy a criterion of interest.

```
// Coast line detection from two channels of a SPOT image.
// The coast detection operator (XS3-XS2)/(XS3+XS2) is computed only
// for selected pages, i.e. pages with XS3 mean value between some thresholds.
//
// XS3 ---- page mean ----- threshold --\
//       \                               > selector --- results
//        >--- (XS3-XS2)/(XS3+XS2) -----/
// XS2 --/

#include "coast.h"

int main()
{
    ddCCacheImg::InitCache(                  // Page cache initialization :
                  ddCSize(128,128),          // - tiles of 128x128 pixels
                  ddCSize(8192, 8192),       // - maximum image size
                  ddCCacheImg::nonOverlap,  // - no overlapping between pages
                  1000000L);                 // - cache initially limited to 1 Mb.

    DdCFileIMA     XS2("xs2.ima"),           // Source images
                   XS3("xs3.ima");

    ddCCoastDetect coast(&XS2, &XS3);        // (XS3-XS2)/(XS3+XS2)
    ddCMean        mean(&XS3);                // Mean of each page
    ddCThreshold   threshold(&mean, 50, 100); // Threshold the mean

    // Return "coast" pages if validated in "threshold", a page of zeroes if not :
    ddCSelector    selector(&threshold, &coast);

    // Store the result image (and so trigger the demand mechanism) :
    ddCFileIMA     results( "results.ima", selector );

    return 0;
}
```

Note how the application algorithm is described as functional relationships between image, that is as a functional program.

The call back function of the ddCCoastDetect operator follows.

```
void ddCCoastDetect::CalcPage(ddCPage& destPage, ddCPage& srcPage1, ddCPage& srcPage2)
{
    for (int y = 0; y < srcPage1.height; y++)
      for (int x = 0; x < srcPage1.width; x++)
      {
          int px1 = *(BYTE*)srcPage1(x,y);
          int px2 = *(BYTE*)srcPage2(x,y);
          *(BYTE*)destPage(x,y) = ((float)(px2-px1))/(px2+px1) * 300.0;
      }
}
```

Tiling and demand driven evaluation are hidden (with the exception of the call to InitCache at the beginning of the application program), so that source code of both application and operator is not more complex than it would be with classical software architectures.

In comparison to classical techniques that store whole intermediate images, tiling allows to run this algorithm with very few memory. Because the operators involved are not spatial, only three pages (of 10 kbytes each) are necessary to run this application. The cache size of 1 Mbyte is thus much over dimensioned, even with 64 Mbytes images.

4. DISCUSSION

Let's note that tiling and demand driven evaluation allow efficient memory management because a page of the result image only depends on a limited number of pages in the argument image. This is the case with convolutions and mathematical morphology operators.

No efficiency improvement should be expected with operators such as distance transforms, connected components labeling or recursive filters. With those operators, the value of a result pixel depends on all the pixels of the argument image. So the whole argument image must be available in memory. Nevertheless, if an application uses such an operator, tiling can still be useful with the other operators.

5. FUTURE WORK

5.1. Spatial operators

Non spatial operators have already been implemented and shown to work. Spatial operators such as convolution and mathematical morphology are still being developed.

5.2. Extension to data volumes

Extension of this architecture from 2D images image sequences and to data volumes is straightforward. Image tiles are just replaced with parallelepipedic volumes. The demand mechanism between operators is unchanged.

5.3. Automatic choice of the type of page

Given some information on each operator, and given an algorithm, choose automatically the best type of page (overlapping or not, lines or rectangle tiles), and eventually the page size.

5.4. Cache management policy

Memory cache for computed pages is a limited resource, so the cache will rapidly grow full. When a new page has to be stored, some other page, regarded as less valuable, must be chosen to be deleted. Suggestions to improve the FIFO choice strategy follow.

5.4.1. Oldest page referenced

When processing occurs at some location in the image, and a page is demanded at that location, there is a higher probability that that page will be demanded again.

Every time a page is demanded, it could be moved to the end of the list, even if it was found in the cache. Pages to be deleted are those at the head of the list.

5.4.2. Cheapest page to recompute

When a page is deleted, it may happen that it is demanded again later, and thus computed again. Some pages are more expansive than other as regards the computing time, depending on the operator. Thus it may be interesting to delete first pages that can be faster recomputed if needed.

The time spent to compute a page could be measured by the page management routine, and pages in the cache could be sorted accordingly. When a page is deleted, all pages that depend on it should have their cost incremented by the cost of the page being deleted.

5.4.3. Both old and cheap page

The best strategy might be to combine the reference order and the computing cost of a page into a single sorting value.

5.5. Multi-threading implementation

Parallelism will be supported transparently in the kernel layer. Mutual exclusion mechanisms will allow several threads of computing to access and compute pages in the cache in parallel.

6. REFERENCES

1. Hugh Glaser, Chris Hankin and David Till, *Principles of functional programming*, Prentice-Hall International, London, 1984.

2. Simon L. Peyton Jones, *The implementation of Functional Programming Languages*, Prentice-Hall International, London, 1987.

3. J. Sérot, G. Quénot, and B. Zavidovique, "Functional programming on a data flow architecture : applications in real-time image processing", *Machine Vision and Application*, vol. 7, pp. 44-56, 1993.

4. Craig M. Wittenbrink and Arun K. Somani, "Cache tiling for high performance morphological image processing", *Machine Vision and Application*, vol. 7, pp. 12-22, 1993.

5. Takashi Matsuyama, "A survey of Image Processing and Computer Vision Software Systems on Workstations", IAPR TC5 *Benchmarking and Software*, 1992.

6. H. Tamura, S. Sakane, F. Tomita, and N. Yokoya, "Design and implementation of SPIDER - A portable image processing software package", *Computer Vision, Graphics, and Image Processing*, vol. 23, pp. 273-294, 1983.

7. John Rasure and Mark Young, "Open environment for image processing and software development", SPIE Vol. 1659 *Image processing and interchange,* 1992.

8. M. L. Powell, S. R. Kleiman, S. Barton, D. Shah, D. Stein & M. Weeks, *SunOS 5.2 Multi-thread Architecture*, proc. of the USENIX Conference, 1991.

9. Thomas M. Breuel, "Functional programming for Computer Vision", SPIE Vol. 1659 *Image processing and interchange*, pp. 216-227, 1992.

10. Y. Kozato and G. P. Otto, "Geometric transformations in a Lazy Functional Language", proc. of the 11th IAPR, The Hague, pp. 128-132, 1992.

11. Silicon Graphics, *Image Vision Library*™, Technical Report, SGI, Mountain View (CA), 1992.

12. Victor Dogaru, *Evaluation à la demande pour le traitement des images*, Thèse professionnelle, Mastère IST, ENST, Paris, 1994.