PERFORMANCE IMPROVEMENT OF IMAGE PROCESSING SOFTWARE APPLIED TO PHOTOMOSAIC CREATION SOFTWARE

S. Stanković[1], B. Predić[2]

1. Ecole Polytechnique de Lausanne, Lausanne, Switzerland, e-mail: stanislav.stankovic@epfl.ch

2. Faculty of Electronic Engineering, University of Niš
Beogradska 14, 18000 Niš, Serbia and Montenegro, e-mail: bpredic@elfak.ni.ac.yu

Abstract: In this paper we give an example of performance improvement of image processing software. We focus on reduction of number of slow hardware dependant operations executed during work process of software in question. We also give the methodology for this particular optimization. A brief introduction to the problem of creating photomosaic images is also given to give the reader better understanding of the process that is being optimized.

Keywords: image manipulation, algorithm optimization, photomosaic creation

1 Introduction

As a result of previous research project a software application was developed.[1][2] In this paper we are discussing algorithm optimizations for this software. For a reader to get better understanding of the processes that are being optimized we believe we should give a brief introduction to photomosaic creation algorithm first.

Photomosaic is a method for image transformation used mainly for artistic purposes. For this method original image is divided into a number of rectangular segments identical in size. Each of these segments is replaced with the most similar image chosen from a set of prepared images called tile library. The result is the image that retains main geometrical and colour characteristics of the original image when viewed from a distance. When viewed up close new details introduced by tiles are visible. (Fig 1.) These details are in no logical relation to the original image.

The main problem is finding the most similar tile to a segment of the original image. There is a number of common and widely used image similarity metrics [3], [4], [5]. The simplest and most efficient approach is calculating and summing colour similarity per pixel of the segment. Since RGB is the most commonly used colour space in computer imaging and it is a three-dimensional model space Euklidean measure appears to be well suited.

$$\| c_1 - c_2 \| = \sqrt{(c_{1,x} - c_{2,x})^2 + (c_{1,y} - c_{2,y})^2 + (c_{1,z} - c_{2,z})^2} \qquad (1)$$

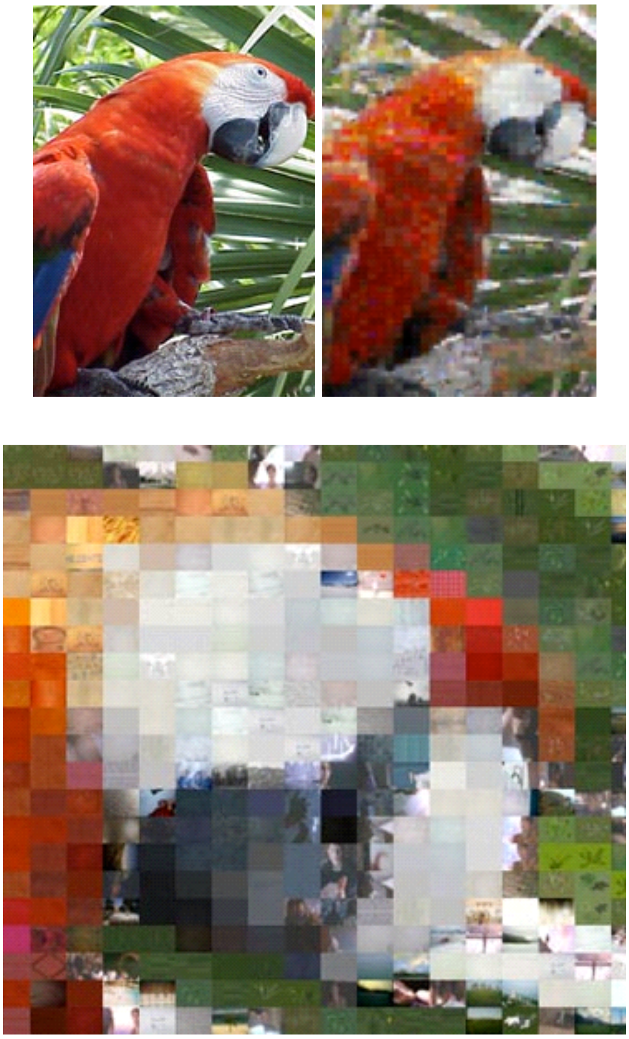

Fig 1: Example photomosaic

The job that this application performed was to compare each piece of given source image to every image in large library and to calculate their difference by given formula. The goal was to find the image from library with smallest possible difference to each piece of source image. Only the best result for each piece were stored. Source image was divided typically in to grid of 60 by 60 pieces, 3600 pieces in total. Image library consist of several thousands of small images, in this particular case 12000. Size of each image in library is 32x24 pixels. Software performed this task in fallowing steps:

1. Piece of source image was selected. Crop operation was performed on original image for given coordinates and size of piece.
2. Selected piece was scaled down to the size of images in library.

3. Image from library was loaded from hard disk.
4. Difference was calculated for given piece and image from library. Calculation of difference between the piece and certain image from library is a sum of differences of corresponding pixels of piece and that particular image, 32x24=768 in total.
5. Last two steps were repeated for every image in library that is 12000 in total.
6. Whole process was repeated for all 3600 pieces of original image.

It can easily be calculated how many times each operation was performed during the whole process. 3600 crop operations, 3600x12000=43200000 loading operations from hard disk and 32x24x43200000=33 177 600 000 calculations of difference formula (1) that was slightly modified for purposes which are of no interest to this paper, for every pair of pixels on every piece of original image and images from library. Software managed to finish this task in roughly 36 hours. For this experiment a PC with AMD Thunderbird processor at 900mhz with 256MB of RAM was used. OS platform was Microsoft Windows 2000 Server Edition. Software was developed in C++ programming language using Borland C++ Builder 5.0. Images in library were stored in uncompressed Windows BitMap format. This particular format was selected because of deterministic size of files that it produces. Any uncompressed format or format with loseless compression might have been used.

The structure of software gave many possibilities for improvement. In following text we first give the results of test of performance of each application component. After that we propose possible optimizations of application code.

2 Program testing with RAM disk

We have analyzed the work process of application by using AQtime 2.0 Profiler software. In this way we could determine how much of total processing time did each of steps approximately take. Results of profiler testing are shown in fig. 2.

MyBitmap::MyBitmap overloaded 2 is the procedure that performs actual loading from hard drive of images from library. It can be seen that more then 85% of total processing time is used by this procedure. This procedure is called approximately 43200000 times by main program. Such great time consumption is a result of well known long response time of hard disk drives in modern computers. In conclusion 85% of total time needed for completing the main task was used by an operation that is not crucial for the operation of software. Even more important time taken by this procedure is heavily hardware dependant, as response time of hard disk is something that can not be much improved.

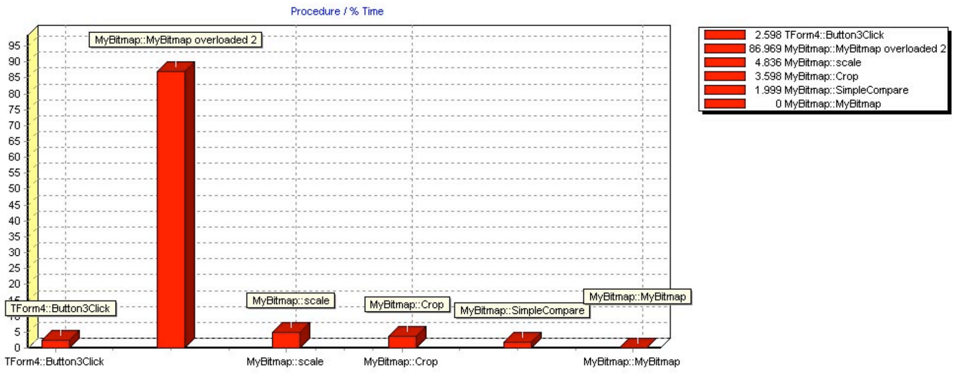

Fig. 2: Results of the profiler test

To prove this assumption we conducted another test. To eliminate the influence of hard disk response time on software performance we created one RAM Disk in RAM memory of our test machine. MS RAM Disk Beta unauthorized driver was used for this [6]. Complete image library was then stored in RAM of the machine. This method was selected because in that way we could perform our experiment without changing the source code of tested software, insuring the objectivity of the test.

As expected, the performance of software greatly improved. Time needed for completion of whole process was reduced many times. What is more interesting the time consumption by each individual step of process changed dramatically. Repeated tests by profiler software can be seen of fig. 3.

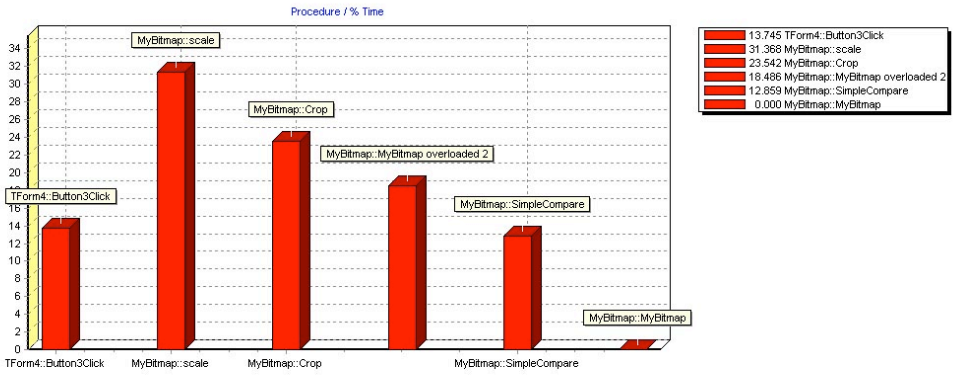

Fig. 3: Results of profiler testing using the RAM disk

It can be observed that MyBitmap overloaded 2 procedure, that had been using by far the greatest percent of processing time dropped to third place of time consumption with only 18% of total time. Calculation heavy procedures Scale and Crop with approximately 31% and 23% of total are now most time consuming operations in process.

By storing complete image library in RAM of the machine influence of hardware was reduced to minimum. However RAM drives are not particularly suitable way of storing large libraries of images, because of limitations of capacity and price or RAM in today's PCs.

3 Optimization of application code

Since this is numerically intensive algorithm further complicated by repeated and numerous access to hardware dependent functions redesign of core comparison algorithm was needed. [7] Another solution was found in different organization of process. Software needed to compare each piece of original image to each image in library, calculate the difference and store the result. This could be achieved in fallowing steps:

1. Source image is divided in individual pieces. Each piece was then scaled to appropriate dimensions (32x24 pixels) All of the pieces are kept in memory during the whole process. As there is only 3600 of relatively small pieces this does not demand too large amount of the available memory.
2. Image is loaded from library.
3. Image is compared to each of the pieces in memory and difference is calculated.
4. Previous step was repeated for every image from library. Only best results, (with smallest difference) for each piece were stored.

Code was changed in a way that each of 12000 images from library was loaded only once during the process instead of 3600x12000 as former version of software. Because of its sheer volume library of tiles still had to be stored on slow hard drive, but segments of the original image were moved to the system RAM during this process of optimization. These two changes of the workflow made considerable improvement in application performance. No change in end result of process was made what so ever. Number of cropping and scaling operations was not altered. Subsequent testing with profiler gave the results that can be seen on fig. 4.

Total calculation was cut from approximately 36h to 3h that is 12 times faster.

4 Conclusion

This example shows importance of optimization of software designed for processing a large number of relatively small images.

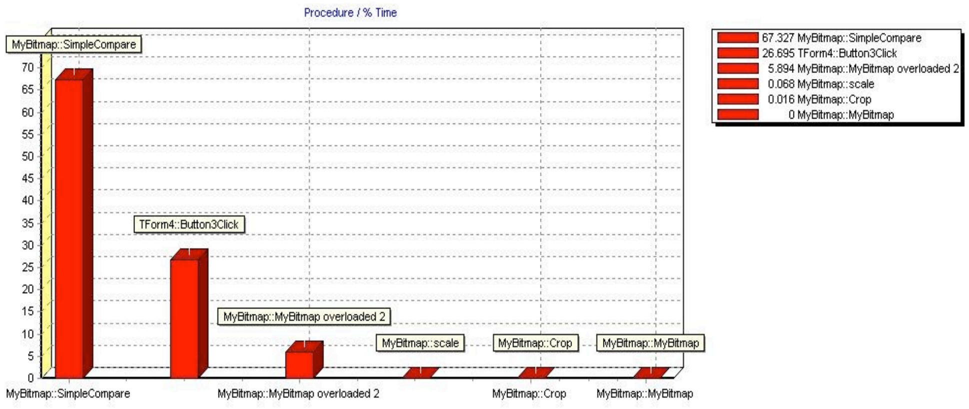

Fig. 4: Profiler results after the final optimization

5 References

1. Stanislav Stanković, Bratislav Predić, "Photomosaics and Image Similarity", *TELFOR 2002*, Beograd, 26-28.11.2002., Proceeding, p. 691.
2. Bratislav Predić, Stanislav Stanković, "Linearization of RGB Colour Space and its Application in Solving Photomosaic Problem", *YU INFO 2003*, Kopaonik, 10-14.3.2003., Abstract proceedings, p. 23.
3. Bogdan Cramariuc, Ilya Shmulevich, Moncef Gabbuj, Asko Makela, “A New Image Similarity Measure Based on Ordinary Correlation”, *IEEE International Conference on Image Processing*, Vancouver, BC, Canada, September 10-13, 2000.
4. Faouzi Alaya Cheikh, Azhar Quddus and Moncef Gabbouj, "Shape Recognition based on Wavelet-Transform Modulus Maxima", *4th IEEE Southwest Symposium on Image Analysis and Interpretation,* Austin, Texas, April 02 - 04, 2000, 8-12.
5. Ioannis Fudos, Leonidas Palios and Evaggelia Pitoura, "Geometric-Similarity Retrieval in Large Image Bases", *Proceedings of the 18th International Conference on Data Engineering (ICDE.02)*, 2002.
6. Microsoft example software driver for RAM disk http://support.microsoft.com/support/kb/articles/Q257/4/05.ASP
7. Stefan Goedecker, Adolfy Hoisie, “Performance Optimization of Numerically Intensive Codes (Software, Environments, Tools)”, Society for Industrial & Applied Mathematics, March 19, 2001.