Bilinear interpolation algorithm optimalization with AVX vectorization unit

Jakub Pružinec
email: j.pruzinec@gmail.com, university email: xpruzi02@fit.vut.cz
Brno University of Technology, Faculty of Information Technologies

14.12.2018

Abstract

Resizing is one of the most frequently used image operations and therefore comes in many algorithm variations. The algorithms are designed to fulfill different criteria. Resizing algorithm is part of greater project primary focused on collection of media file metadata where execution speed is major consideration. Bilinear interpolation resizing is one of algorithms that are designed to be fast. In this project I design, implement and test sequential and parallel bilinear interpolation algorithm and then I consider its suitability for parallelism and image metadata gathering.

1 Introduction

1.1 Media info gatherer

Metadata info gatherer is a project focused on media file metadata collecting. Exported metadata is to be further processed in order to decet duplicity of images, videos, audio files etc. File system metadata is easy to obtain and is good for duplicity hints, however it is volatile and unreliable. For these reasons it is often necessary to inspect file content and produce fingerprint metadata.

1.2 Perceptual hashes

One option to identify file content is to produce its cryptic hashes like SHA256 or MD5, but they are expensive in terms on computation time. Also a minor change in file content is reflected by major change in its cryptic hash. All of these inconveniences can be avoided with perceptual hashes when it comes to image processing. Perceptual hashes are hashes that represent images in a way that their similarity can be determined based on them. One of computationally inexpensive perceptual hashes is average hash.

1.3 Average hash

Average hash is similar to low pass filter output when images are considered signals. Images are resized to 8x8 to filter out small frequencies effectively ignoring image details. Then they are converted to greyscale and to black and white afterwards. All of 64 pixels produced this way are either black or white. Every pixel can represented with a binary value, 0 for black and 1 for white, creating a 8 byte average hash.

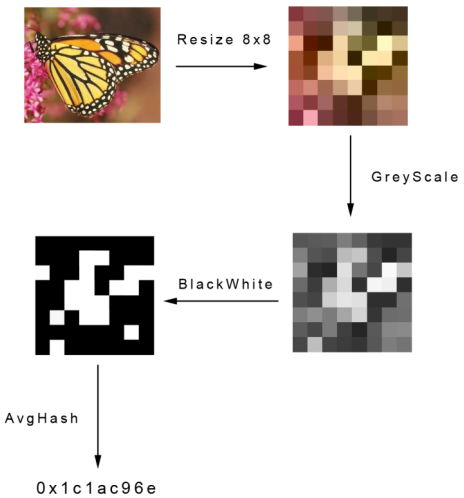

Figure 1: Average hash computation block schema

1.4 Similarity matric

Similarity of two images is determined by the number of similar pixels. Taking into cosidaration that every bit of average hash represents a pixel color the similarity of images can be expressed as Hamming distance of their average hashes. Images whose average hashes Hamming disntace is 3 or less are considered similar.

1.5 Bilinear interpolation

In process of average hash computation images are resized. In bilinear interpolation algorithm every pixel in new image P' is an interpolation of four pixels P_1, P_2, P_3, P_4 from original image. Choice of interpolated pixels is dependant on image horizontal and vertical scale.

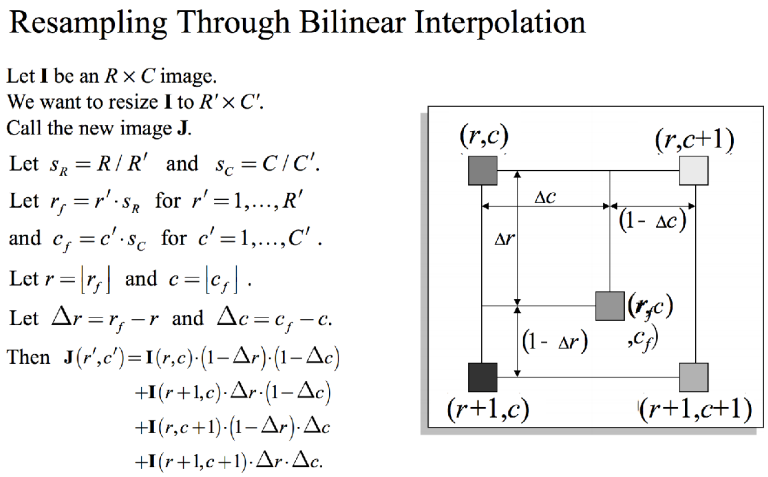

Figure 2: Bilinear interpolation

2 Design

2.1 Bitmap parser

Bitmap format is commonly used format for image files. To process image content it needs to be converted to internal representation first. Images are parsed to structures consisting of image dimensions and three color channel byte arrays. Channel arrays are allocated as continuous memory chunk to maximize the chance of loading them in a single cache line.

2.2 Naïve bilinear interpolation algorithm

Naïve bilinear interpolation resizing is depicted in following psedo code.

```
calculate horizontal and vertical scales sr, sc
for (rNew in newHeight)
    for (cNew in newWidth)
        calculate interpolated pixels coordinates
        bounds check
        calculate deltas
        for (every channel)
            read 4 pixles and calculate their weights
            multiply pixel colors with weights
        calculate new channel values as sum of four previous results
        store new pixel
```

2.3 Sequential bilinear interpolation algorithm

Removing unnecessary calculations from naïve algorithm leads to significant efficiency increase. Sequential bilinear interpolation algorithm has the following modifications:

- interpolated horizontal pixels coordinates calculated in outer loop
- interpolated pixels coordinates are incremented in every iteration to avoid multiplication
- horizontal delta calculated in outer loop
- horizontal fragments of weight formulas are calculated in outer loop
- weights are precalculated for all channels only once

```
calculate horizontal and vertical scales sr, sc
for (rNew in newHeight)
    increment rf with horizontal scale
    calculate horizontal coordinates of interpolated pixels
    calculate horizontal delta
    calculate horizontal fragments of weight formulas
    for (cNew in newWidth)
        increment cf with vertical scale
        bounds check
        calculate vertical coordinates of interpolated pixels
        calculate vertical delta
        calculate weights
        for (every channel)
            multiply pixel colors with weights
        calculate new channel values as a sum of four previous results
        store new pixel
```

2.4 Parallel bilinear interpolation algorithm

Since most of operations in inner loop are same for all of image pixels, the pixels are processed parallely. Vector can contain 8 pixel values, thus 8 pixels are processed simultaneously. To minimize parallel processing overhead the following algorithm substitutions are designed:

- incrementation of $cf \rightarrow$ vector of indices $V_i = (0.0, 1.0, ..., 7.0)$ is multiplied with horizontal scale vector $V_{\text{sc}} = (sc, sc, ..., sc)$ then in every inner iteration vector V_i is incremented with vector $V_{8\text{sc}} = (8sc, 8sc, ..., 8sc)$
- inner loop $\rightarrow$ the inner loop is iterated with 8 pixel step and all pixels are processed parallely. Remaining pixels are processed sequentialy in separate loop afterwards.
- bounds check $\rightarrow$ bounds check is done by sequence of vector masking and vector logical operations

```
calculate horizontal and vertical scales sr, sc
for (rNew in newHeight)
    increment rf with horizontal scale
    calculate horizontal coordinates of interpolated pixels
    calculate horizontal delta
    calculate horizontal fragment of weight formulas
    assign cf vector with mutiplication of index and sc vector
    for (8cNew pixels in newWidth)
        mask out values exceeding bounds
        calculate vector of vertical coordinates of interpolated pixels
        calculate vector of vertical deltas
        calculate vector of weights
        for (every channel)
            read 8 values sequentialy
            multiply pixel colors vector with weights
        calculate 8 new channel values as a sum of four previous vector results
        store new pixels
        increment cf vector with 8sc vector
```

The major bottle neck here is that interpolated pixel coordinates need to be computed in every iteration and thus **interpolated pixel colors have to be read from memory sequentialy**.

3 Implementation

3.1 Compilation and execution

To build demonstration project execute the following commad

```
make
```

Program synopsis

```
./image-info <imagefile1> <imagefile2>
./image-info_avx <imagefile1> <imagefile2>
```

Image-info parses two 24bit color depth Bitmap files, prints their average hashes and determines their similarity.

3.2 Resizing codes

Sequential code is written in C language and can be found in *./src/image_ resize.c*.
Parallel code is written in C language with AVX1 SIMD, single instruction multiple data, intrinsic functions and can be found in *./src/image_ resize_ avx.c*

4 Performance analysis

In order to estimate algorithm efficiency the *main()* function is altered to resize image 1000 times. When measuring execution time image parsing time is neglected. Results are compared to previous attempts of bilinear interpolation algorithm optimization by Xujie Zhang. Note that testing hardware, Intel Core i5 2540M, is comparable with the referenced one, AMD FX 8350.

Optimization method	Time/ms	Image info efficiency improvement
Zhang's sequential code	11.34	-
Zhang's SIMD code	6.93	-
Image-info sequential code	4.38	159%
Image-info SIMD code	5.8	19%

Table 1: Performance results

5 Conclusion

Despite outperforming both, sequential and parallel, previous implementations of algorithm it is necessary to consider the suitability of SIMD approach in bilinear interpolation. SIMD version of algorithm has a drastic decrease in performance. This is due to inevitability of sequential memory accesing and the fact that parallelism of few operations benefit is too small to cover overhead that use of SIMD instructions brings.

Further, it is worth to note that bilinear interpolation might not be the best choice when it comes to average hashes. When downscaling images with bilinear interpolation only a fraction of pixels is taken into account. This is particularly inconvenient when comparing images with noise. Noise can be significantly reflected in downscaled image if it overlaps with pixels from original image that are not ignored. If this is a concern, then e.g box sampling might be a better choice.

6 Resources

Bilinear interpolation algorithm optimization by Xujie Zhang
Digital image processing by Richard Alan Peter