Large Image Streaming using ITKv4

Release 1.10

Maria A. Zuluaga[1,2], Luis Ibáñez[3], Françoise Peyrin[1,2]

May 18, 2011

[1]CREATIS; Université de Lyon; Université Lyon 1; INSA-Lyon; CNRS UMR5220; INSERM U630; F-69621 Villeurbanne, France
[2] European Synchrotron Radiation Facility, Grenoble, France
[3]Kitware Inc., Clifton Park, NY

Abstract

This document illustrates how to process large images (6 and 23 Gigabytes in size) by taking advantage of the streaming capabilities of the Insight Toolkit ITK. Here we illustrate two scenarios: (a) the case when the image itself is larger than the computer's RAM, (b) the case when the image is large but still can fit in the computer's RAM. This report is intended to serve as a tutorial on how to take advantage of this memory management capabilities of ITK.

This paper is accompanied with the source code, input data, parameters and output data that we used for validating the algorithm described in this paper. This adheres to the fundamental principle that scientific publications must facilitate **reproducibility** [2] of the reported results.

Contents

1 Introduction

Large images are becoming ubiquitous in many research fields. They are particularly common in microscopy, remote sensing and computer vision. Image sizes are growing at a higher rate than the size of computer memory and therefore there is a great interest in software methods that allow to process images by partitioning in pieces that can temporarily be fit into memory. Such a process is commonly denoted as streaming.

In this technical report we illustrate how this can be done with the current infrastructure of the Insight Toolkit (ITK). For this matter, we make use of two large images (6 and 23 Gigabytes in size) from the Creatis collection of the MIDAS database [3].

2 Implementation

The example we develop illustrates the pipeline executed in order to segment osteocytes from trabecular bone images. Osteocytes appear as small dark regions inside white larger structures (Figure 1). A simple pipeline to extract these dark regions consists of: 1) Image binarization, 2) Hole-filling on the binarized image and 3) Subtraction of the hole-filled and the binarized images. In the following, we present the different stages of the pipeline and how the whole process can be streamed.

2.1 Reading and Writing

A key requirement in order to perform such task through the partitioning of images is that the `itk::ImageIO` classes involved in the pipeline support streamed reading and writing. Otherwise, the entire image will be buffered in memory [1].

The following code illustrates the reading of a large image that is then written into another file. While this is the simplest case that can be illustrated, it contains the core elements that are required to perform streaming.

First we include the headers:

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
```

Then, we instantiate the reader and writer.

```
  ImageReaderType::Pointer reader = ImageReaderType::New();
  ImageWriterType::Pointer writer = ImageWriterType::New();
```

In order to trigger the use of streaming, it is necessary to specify to the writer into how many blocks the image should be partitioned. In that sense, the most important line in the streaming process is:

```
  writer->SetNumberOfStreamDivisions( numberOfDataBlocks );
```

Finally, we use the standard try / catch block that calls the Update method and triggers the whole process.

```
try
  {
  writer->Update();
  }
catch( itk::ExceptionObject & err )
  {
  std::cerr << err << std::endl;
  return EXIT_FAILURE;
  }
```

2.2 Binary Thresholding

The first filter used in the segmentation pipeline is a binary thresholding that allows to differentiate the osteocytes from the rest of the image. In this example, we connected a binary thresholding filter between the reader and the writer.

```
typedef itk::BinaryThresholdImageFilter<
             InputImageType, OutputImageType >  FilterType;
typedef itk::ImageFileReader< InputImageType >  ReaderType;
typedef itk::ImageFileWriter< OutputImageType >  WriterType;

ReaderType::Pointer reader = ReaderType::New();
FilterType::Pointer filter = FilterType::New();
WriterType::Pointer writer = WriterType::New();

writer->SetInput( filter->GetOutput() );
reader->SetFileName( argv[1] );
filter->SetInput( reader->GetOutput() );
```

As can be seen from the following code, there is no significant difference between the previous example and this one in order to activate the streaming. The streaming process is still driven by the writer:

```
writer->SetNumberOfStreamDivisions( numberOfDataBlocks );
```

2.3 Noise Elimination and Filling-Up Holes

It is important to differentiate the streaming concept from multithreading. While streaming refers to the process of sequentially processing sub-regions part of the largest possible region) of an image through the pipeline [1], multithreading refers to the possibility of making use of the multiple cores of a computer to do the processing.

In ITK, not all filters supporting streaming support the use of multi-threading. Such is the case of ITK's morphological filters, which are commonly used for hole filling. Therefore, we have selected to use `itk::VotingBinaryHoleFillingImageFilter`, for the hole filling task, instead of the morphological filters, since the former supports both streaming and multithreading.

The key elements required to make us of `itk::VotingBinaryHoleFillingImageFilter` are shown in the following:

```
typedef itk::ImageFileReader< InputImageType > ReaderType;
typedef itk::ImageFileWriter< OutputImageType > WriterType;
typedef itk::VotingBinaryHoleFillingImageFilter<
      InputImageType, OutputImageType > VotingFilterType;

ReaderType::Pointer reader = ReaderType::New();
WriterType::Pointer writer = WriterType::New();
VotingFilterType::Pointer filter = VotingFilterType::New();

reader->SetFileName( argv[1] );
writer->SetInput( filter->GetOutput() );

InputImageType::SizeType neighborhoodRadius;
neighborhoodRadius[0] = atoi( argv[5] );
neighborhoodRadius[1] = atoi( argv[5] );
neighborhoodRadius[2] = atoi( argv[5] );

filter->SetInput( reader->GetOutput() );
filter->SetBackgroundValue( atoi( argv[3] ) );
filter->SetForegroundValue( atoi( argv[4] ) );
filter->SetRadius( neighborhoodRadius );
filter->SetMajorityThreshold( atoi( argv[6] ) );
```

Depending on the foreground and background values that are provided to the filter, it can perform hole filling or noise removal from the background (see Figure 2). We have used it here for both purposes.

We recall again in the fact that there is only a single code line that need to be included to activate the streaming:

```
writer->SetNumberOfStreamDivisions( numberOfDataBlocks );
```

On the other hand, nothing needs to be added in order to activate the multi-threading. If the filter allows the use of multiple cores, it will make use of them.

2.4 Subtraction

The final stage of the pipeline consists in the subtraction of a noise-free version of the binarized image from the hole-filled image. The resulting image contains the segmented osteocytes. The implementation of this stage is straightforward and it has no additional difficulty in what respects to the streaming. For the sake of completeness, we limit ourselves to just mention this stage here but, we do not include a code example in the text of this report. Of course, we include in the annex materials the source code that we used for this subtraction operation, as required by **reproducible research standards** [2].

3 Results

We have evaluated the pipeline using two different images from the Creatis collection in the MIDAS database[3] with respective sizes of 6 and 23 Gigabytes. One of the requirements of the selected images

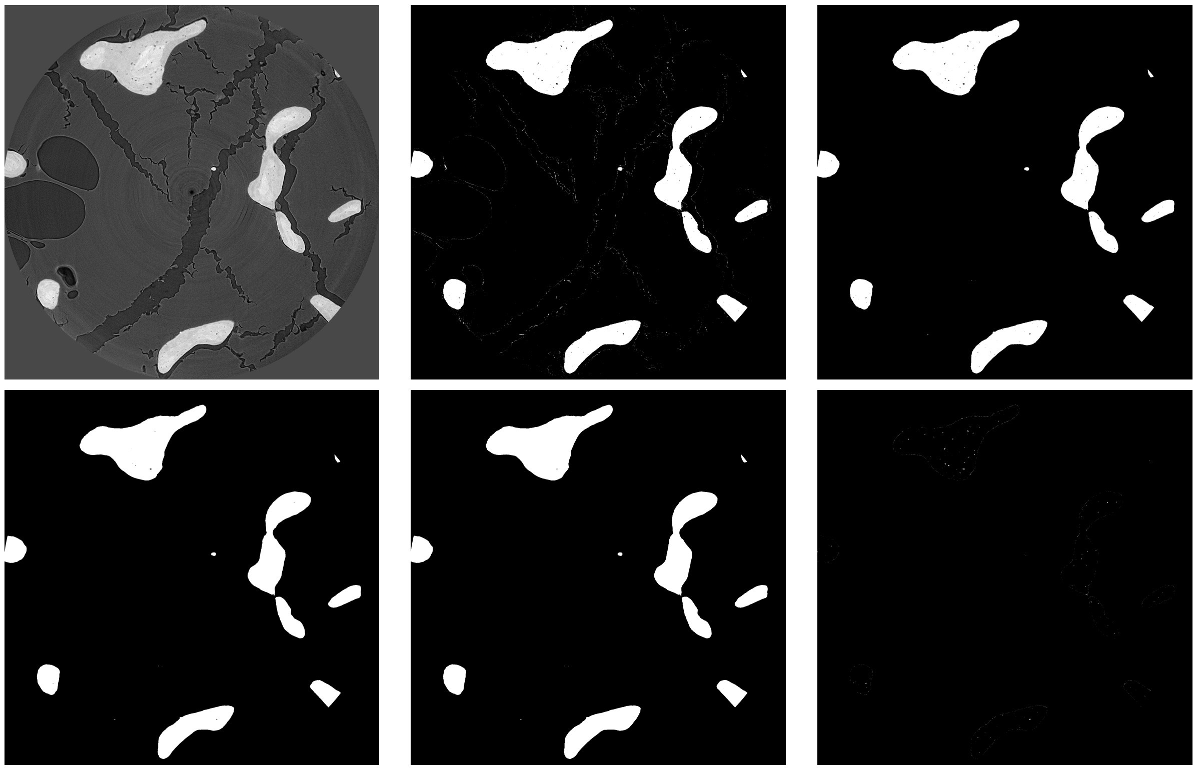

Figure 1: First row. Left. Original image. Center. Binarized image. Right. Image after noise removal. Second row. Left. Hole-filling after one iteration. Center. Hole-filling after three iterations. Right. Subtracted image.

is that they come on the MetaImage format which can be processed with streaming. Not every image format accepts such possibility. Figures 1 and 2 show the results obtained at different stages of the pipeline.

We should mention that, as can be seen from figures 1 and 2, the hole-filling process requires the iterative execution of the voting filter. While ITK provides a `itk::VotingBinaryIterativeHoleFillingImageFilter` that iterates until all the image holes are filled, we select to use the simpler `itk::VotingBinaryHoleFillingImageFilter` since it permits the use of streaming and multithreading.

The whole pipeline was tested on an Dual Core Intel Centrino with 4Gb in RAM. For the image of 6GB (image 1), the data was partitioned into 6 different blocks. For the 23 Gb image (image 2), the data was partitioned into 10 different blocks. Given f the number of filters involved in a processing pipeline and s the size in (bytes) of a slice, a simple rule that can be used to estimate the desired number of partitions p, should be

$$\frac{f*s}{p} < RAM \tag{1}$$

Afterwards, a trade-off between the speed of the process and the number of partitions should be evaluated. As the number of partitions increases, the pipeline execution will take longer, unless a sufficient number of CPU cores is available. As an example, Table 1 summarizes the computational times of each stage of the pipeline with the two evaluated images.

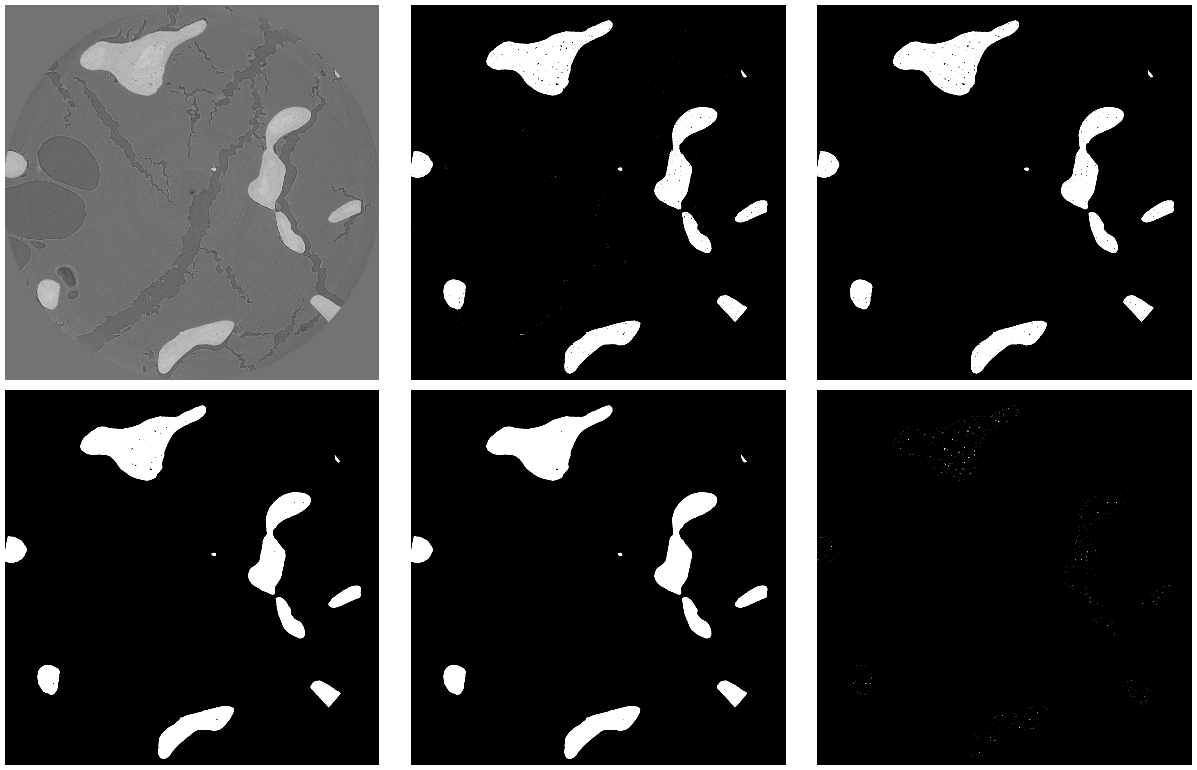

Figure 2: First row. Left. Original image. Center. Binarized image. Right. Image after noise removal. Second row. Left. Hole-filling after one iteration. Center. Hole-filling after three iterations. Right. Subtracted image.

Table 1: Computational times for each of the pipeline stages.

	Image 1 (6Gb)	**Image 2 (23Gb)**
Read and Write	93.26	141.137
Binarization	129.79	181.06
Noise removal	782.92	1075.76
Hole-filling (first pass, one iteration)	1147.51	2399.96
Hole-filling (second pass, one iteration)	1665.01	2436.57
Hole-filling (third pass, one iteration)	1656.96	2414.11
Subtraction	97.81	223.58

4 Conclusions

In this paper we have illustrated, through a simple example, how the streaming and multithreading capabilities of the Insight Toolkit can be exploited to process image larger than the computer RAM.

References

[1] B. Lowekamp and D. Chen. A streaming IO base class and support for streaming the MRC and VTK file format. *The Insight Journal*, pages 1–6, 2010. 2.1, 2.3

[2] V. Stodden. Enabling reproducible research: Open licensing for scientific innovation. *International Journal of Communications Law and Policy*, Winter 2009(13):2–25, 2009. (document), 2.4

[3] M.A. Zuluaga, A. Larrue, A. Rattner, L. Vico, and Peyrin F. Acquisition of synchrotron radiation micro-CT images for the investigation of bone micro-cracks. *The MIDAS Journal*, 2011. 1, 3