Workshop on Open-Source and Open-Data for MICCAI

Brisbane, Australia – November 2, 2007

Introduction

For this workshop, we encouraged researchers with papers accepted at MICCAI to submit companion papers on the role of open-source software and open-access data in their MICCAI research. Papers related to research not presented at MICCAI were also welcome.

There are several unique aspects regarding the submissions and review process used for this workshop:

- Authors were encouraged to include other files with their submission, particularly videos, code, and data.
- Every submission immediately appeared on the Insight Journal site in the Special Issue on the MICCAI Workshop. Therefore, every submission was available to any registered member of the Insight Journal (and registration is free
- Submission required licensing to the Insight Journal the right to distribute your submission under the Creative Commons' "by-attribution" license. Copyright transfer was not required.
- The Insight Journal is a registered digital library. Therefore, each submission has been assigned a unique open-archive "handle" that (unlike URLs) persists on the web and that (unlike other web-based database keys) allows your submission to be indexed by Google and other search engines. Handles also serve as a mechanism whereby submissions can be permanently referenced in publications.
- The Insight Journal and this workshop use an open-review process. Every submission is available for public peer-review. Any registered member of the Insight Journal was able to download, score, and comment on any submission to this workshop. All reviews are public. For this workshop, the 17 papers received a total of 56 reviews

Workshop Website

http://insightsoftwareconsortium.org/wiki/index.php/2007_MICCAI_Open_Workshop

Workshop Organizers

Dr. Stephen Aylward, Kitware, Inc.
Dr. Tina Kapur, Brigham and Women's Hospital
Dr. Kevin Cleary, Georgetown University
Dr. Luis Ibanez, Kitware, Inc.

Special Thanks To

MICCAI
The NA-MIC / NIH Roadmap
The Insight Software Consortium

Schedule

9:00 – 9:20 Introduction

9:20 – 9:40 Efficient Implementation of Kernel Filtering
Beare R; Lehmann G
Rank Filtering, Efficient Methods *(Page 4)*

9:40 – 10:00 Radial Thickness Calculation and Visualization for Volumetric Layers
Wang D; Shi L; Heng P
Volumetric Layers, Radial Thicknes *(Page 24)*

10:00 – 10:20 Multidimensional Arrays and the nArray Package
Sadowsky O; Li D; Deguet A; Kazanzides P
Vectors and Matrices, Multidimensional Arrays *(Page 32)*

10:20 – 11:20 Posters and Break

11:20 – 11:40 Diffeomorphic Demons Using ITK's Finite Difference Solver Hierarchy
Vercauteren T; Pennec X; Perchant A; Ayache N
Diffeomorphisms, Demons *(Page 46)*

11:40 – 12:00 An Open, Clinically-Validated Database of 3D+t cine-MR Images of the Left Ventricle with Associated Manual and Automated Segmentation
Najman L; Cousty J; Couprie M; Talbot H; Clément-Guinaudeau S; Goissen T
Image Processing, Myocardial Infarction *(Page 54)*

12:00 – 12:20 vtkINRIA3D: A VTK Extension for Spatiotemporal Data Synchronization, Visualization and Management
Toussaint N; Sermesant M; Fillard P
Spatiotemporal, Synchronization *(Page 63)*

12:20 – 12:40 A Framework for Comparison and Evaluation of Nonlinear Intra-Subject Image Registration Algorithms
Urschler M; Kluckner S; Bischof H
Evaluation, Nonlinear Registration *(Page 78)*

12:40 – 2:00 Lunch

2:00 – 3:00 Invited Speaker

3:00 – 3:20 A White Matter Stochastic Tractography System
Ngo T; Westin C; Golland P
DTI, Tractography *(Page 94)*

3:20 – 3:40 Non-rigid Groupwise Registration using B-Spline Deformation Model
Balci S; Golland P; Wells W
Non-rigid Registration, Groupwise Registration *(Page 105)*

3:40 – 4:40 Posters and Break

4:40 – 5:00 An Accessible, Hands-on Tutorial System for Image-Guided Therapy and Medical Robotics Using a Robot and Open-Source Software
Pace D; Kikinis R; Hata N
3D Slicer, LEGO Mindstorms NXT *(Page 122)*

5:00 – 5:20 Tagged Volume Rendering of the Heart: A Case Study
Mueller D
Tagged Volume Rendering, Coronary Arteries *(Page 142)*

Poster Presentations

Vessel Enhancing Diffusion Filter
Enquobahrie A; Ibanez L; Bullitt E; Aylward S
Hessian, Vesselness *(Page 154)*

Spherical Wavelet ITK Filter
Gao Y; Nain D; LeFaucheur X; Tannenbaum A
Shape Analysis, Spherical Wavelet *(Page 168)*

Fast BlockMatching Registration with Entropy-based Similarity
Suarez-Santana E; Nebot R; Westin C; Ruiz-Alzola J
Nonrigid Digistration, Multimodal Registration *(Page 178)*

Optimizing ITK's Registration Methods for Multi-processor, Shared-Memory Systems
Aylward S; Jomier J; Barre S; Davis B; Ibanez L
ITK, Parallel *(Page 186)*

GoFigure and The Digital Fish Project: Open tools and open data for an imaging based approach to system biology
Gouaillard A; Brown T; Bronner-Fraser M; Fraser S; Megason S
Confocal Imaging, Cell Tracking *(Page 199)*

Data, Data Everywhere, nor an Image to Read - Finding Open Image Databases
Holmes D; Robb R
Image Databases *(Page 213)*

Efficient implementation of kernel filtering

Richard Beare[1] and Gaëtan Lehmann[2]

August 15, 2007

[1]Department of Medicine, Monash University, Australia.
[2]INRA, UMR 1198; ENVA; CNRS, FRE 2857, Biologie du Développement et Reproduction, Jouy en Josas, F-78350, France

Abstract

Kernel based filtering is one of the fundamental tools of image analysis and processing. A number of approaches have been developed over the years that allow efficient implementation of such filters even when the kernel size is large. This article reviews some of these methods and introduces their ITK implementations.

Contents

1 Introduction **2**

2 Separability and recursive implementations **2**

3 Mathematical morphology operations **2**
3.1 Arbitrary structuring elements . . . 3
3.2 Decomposition of structuring elements . . . 3
3.3 Line structuring elements . . . 4

4 Rank filters **4**

5 Mean and standard deviation filters **6**

6 ITK implementation **8**
6.1 Common filters . . . 8
6.2 Morphology filters – consolidatedMorphology . . . 9
6.3 Rank and mean filters – fastRankMean . . . 10
6.4 Rectangular convolution – boxConvolution . . . 10
6.5 Extended languages support . . . 10

7 Performance **10**
7.1 Morpholgy . . . 11
7.2 Rank and Mean filters . . . 11
7.3 Box Convolution . . . 11
7.4 Timing notes . . . 11

8 Implementation Notes **11**
8.1 Moving histograms 11
8.2 16 bit data types 14
8.3 Line Operations 14
8.4 Refactoring 15

9 Comments and Conclusions **15**

10 Acknowledgments **15**

1 Introduction

A kernel based filtering process replaces the pixel at the kernel origin with the result of a function of all pixels defined by the kernel. Many useful filters, including edge detection and gradient filters, smoothing filters and rank and morphology filters fall into this category. Direct implementations of such filters typically involve visiting all pixels defined by the kernel in order to evaluate the filter function. Such an approach is usually easy to implement – in ITK it is made simple by the neighborhood iterators – but leads to an algorithm complexity proportional to the number of pixels in the kernel (or $O(n^d)$, where n is the kernel size and d is the dimensionality). Such complexity tends to restrict application of such filters to small kernels.

A number of classical methods exist for reducing this complexity to more manageable, in some cases kernel size independent, levels. ITK already exploits such techniques for Gaussian convolution operations. This paper describes the classical techniques for optimized mathematical morphology filters, rank filters and certain convolution filters. These filters make the use of large kernels, which can be very useful in many applications, practical on conventional computing hardware.

2 Separability and recursive implementations

The two approaches most typically used to reduce complexity of kernel based filters are separability and recursive computation, both of which are used in the ITK implementation of Gaussian convolution filters. A separable filter implements a multidimensional kernel by cascading several one dimensional kernels, therefore reducing complexity from $O(n^d)$ to $O(nd)$. The second approach exploits redundancy that might be present in the computations of kernel functions at neighboring locations, leading, in some cases, to a complexity independent of n.

3 Mathematical morphology operations

Two optimized forms of the mathematical morphology operations of erosion, dilation, opening and closing are presented here. The kernel is usually referred to as the structuring element in mathematical morphology. The methods described here are applied to “flat” structuring elements, that is structuring elements without weights. The first method can be applied to arbitrary structuring elements while the second can be applied to line structuring elements.

3.1 Arbitrary structuring elements

The method described in [11] relies on the simple concept of an up-datable histogram, and is often described as a "moving histogram" approach. A histogram is computed for a kernel located at the first voxel. The histogram at the neighboring voxel can then be computed by including newly included voxels and removing newly excluded voxels. The list of included and excluded voxels corresponding to movement in any direction can be computed when the structuring element is created, and the direction with the smallest number of changes should be selected as the direction for sweeping the kernel across the image. The erosion or dilation at each location is computed by selecting the minimum or maximum from the histogram. This approach is very efficient when 8 or 16 bit pixels are used because the histogram can be represented as an array and the histogram updated by incrementing or decrementing the appropriate bins. Using place holders to track the current maximum or minimum increases performance. More sophisticated histogram representations are necessary for larger pixel types. Our implementation uses c++ maps, which is an extension to the original paper.

This methodology reduces the complexity from $O(n^d)$ to $O(n^{d-1})$, while keeping the structuring element identical to the direct implementation.

3.2 Decomposition of structuring elements

Morphological erosions and dilations are separable – successive dilation by orthogonal lines is equivalent to dilation by a rectangle with sides equal to the line lengths. This means that any hyper-rectangular structuring element can be constructed using several orthogonal lines, typically parallel to the axes.

Approximations of more complex shapes, notably circles and ellipses can constructed using a number of lines at evenly spaced angles [2]. It is difficult to create a structuring element with a precisely defined radius using this method because line structuring elements from which the circle structuring element is composed must have odd length and there are practical limits due to the realities of underlying digital grid representation of images. In addition it is possible that the structuring elements may not be truly translation invariant due to the representation of line (e.g. Bresenham) used in the decomposition. However, precisely defined radii are rarely critical when a large structuring element is called for. An example of a structuring element created using line structuring elements is shown in Figure 1. Composition of regular shapes, such as hexagons and octagons is more accurate.

Figure 1: Approximate circular structuring element, radius 25, constructed using 8 lines.

It is also possible, in theory, to construct 3D structuring elements in similar ways. The construction of a hyper-rectangle is trivial, however construction of spheres is more problematic. The code discussed later provides preliminary implementations based on some platonic solids and various spherical approximations, but further testing and development is needed.

3.3 Line structuring elements

The decompositions discussed above are important because an efficient, recursive, implementation of erosion and dilations along lines exists. This method was introduced in [6, 12] and can compute an erosion or dilation in 3 operations per pixel, independent of structuring element length. [5] recently reduced the cost to 1.5 pixels per pixel, but the procedure is more complicated and there are reports of no speedup in practice.

The original algorithms utilize forward and backward running maxima (for dilation) for the length of the structuring element from a pixel of interest. The dilation can then be computed for a region the size of the structuring element around the point of interest by comparing values on the running extrema that are separated by the structuring element length. This is illustrated in Figure 2

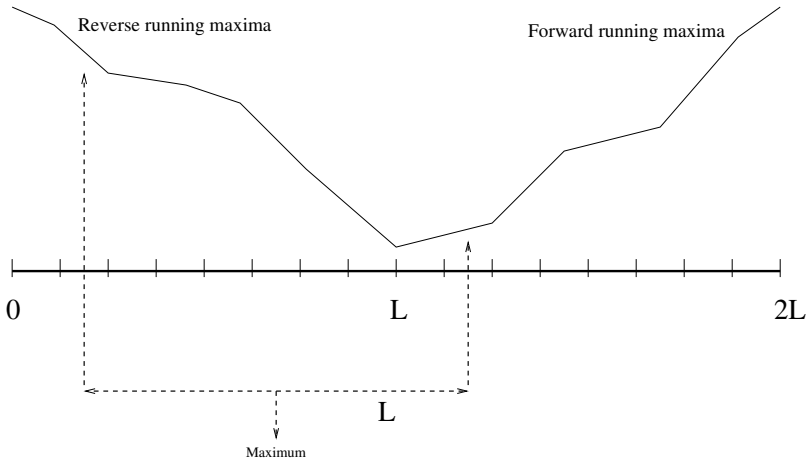


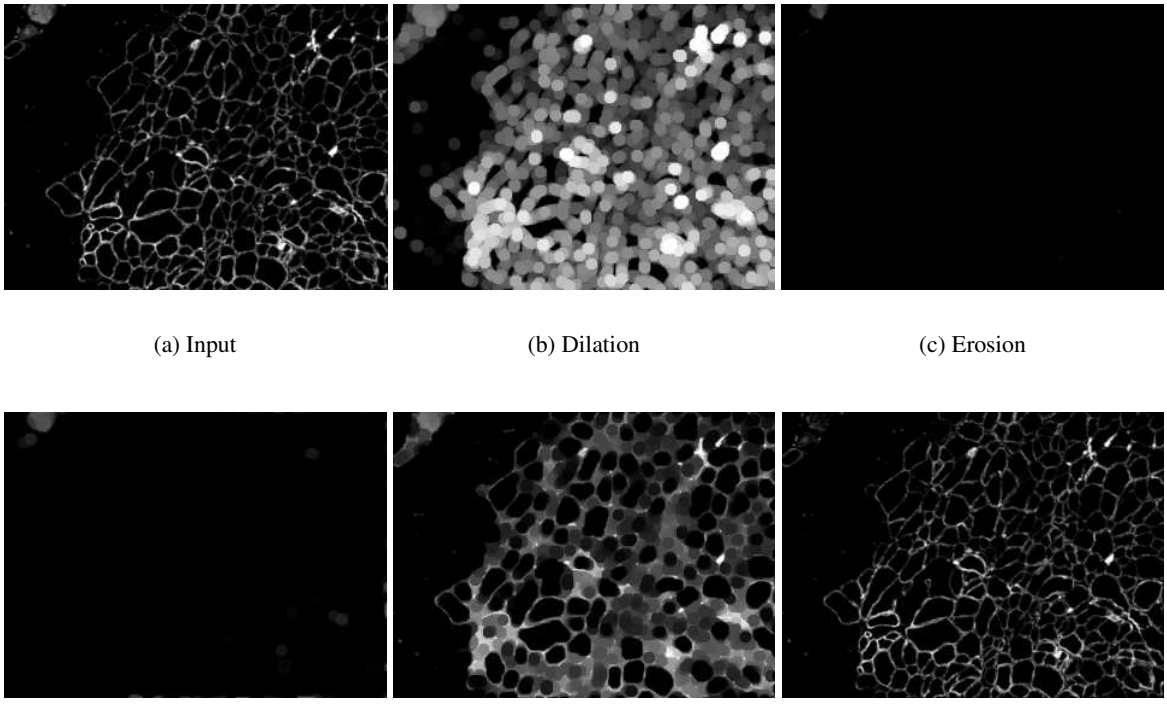
Figure 2: The van Herk, Gil, Werman method.

A method known as *anchor morphology* has been published recently [10] that offers improved performance and the ability to perform a line opening directly (rather than using an erosion/dilation cascade). This method has been implemented in the filters discussed later in the article, but performance issues are not yet clear. The method employs histograms and therefore needs more complex data structures when applied to higher precision data, potentially reducing any speed advantage.

4 Rank filters

Efficient implementations of median and rank filters can be carried out using exactly the same approach as discussed for morphological operations with arbitrary structuring elements. The method was originally proposed in [7]. The implementation discussed later supports arbitrary kernel shapes and pixel types as well as any choice of rank.

Rank filters are not separable. However the performance benefits offered by separability make it worth pretending they are. If median filtering is being used to provide robust noise filtering or background estimation then a separable approximation is worth testing. This concept was originally proposed in [9]. In fact it is also possible to apply the decomposition discussed in Section 3.2 to median filters to achieve a closer approximation to circular kernels.



(a) Input (b) Dilation (c) Erosion

(d) Opening (e) Closing (f) White top-hat

Figure 3: Effect of a dilation (b), an erosion (c), an opening (d), a closing (e) and a white top hat (f) with a ball structuring element.

5 Mean and standard deviation filters

A simplified version of the moving histogram approach can also be used to efficiently compute the mean or standard deviation of an arbitrarily shaped kernel. The histogram used for morphology and rank filters is a data structure that, once established for one pixel, can be easily updated to represent the kernel for a neighboring pixel. The data structure required to perform the equivalent function for mean and variance operations is much simpler – all that is necessary is the sum of pixel values under the kernel (or sum of squared values for variance calculations), and the kernel size. The sum can be updated by adding the new values and subtracting the old and the output computed by dividing by the kernel size.

A more efficient strategy was proposed in [3] for rectangular kernels. This approach first computes an accumulator image in which each i, j is replaced by the sum of voxels "left" and "above" it. This accumulator image may be computed recursively using local neighborhood values. The mean of rectangles of any size may then be computed using accumulated values at the rectangle corners. Figures 4 and 5 illustrate the 2D case and the update formulas are provided in Equations 1, 2, and 3.

$$\begin{aligned} b_{i,j} &= \sum_{x \le i, y \le j} a_{x,y} \\ &= b_{i,j-1} + b_{i-1,j} + a_{i,j} - b_{i-1,j-1} \end{aligned} \tag{1}$$

where $b_{i,j}$ is the accumulator value at location i, j and $a_{i,j}$ is the input image intensity at i, j.

$$S_{i,j} = b_{i+w/2,j+h/2} + b_{i-w/2,j-h/2} - b_{i-w/2,j+h/2} + b_{i+w/2,j-h/2} \tag{2}$$

where $S_{i,j}$ is the sum of the pixel values in the neighborhood centered at i, j.

$$m_{i,j} = \frac{1}{w.h} S_{i,j} \tag{3}$$

where $m_{i,j}$ is the output of the mean filter at location i, j.

Using the same strategy, the standard deviation can be efficiently computed. It requires an accumulator image of the squared pixels values, which can be efficiently computed at the same time as the original accumulator image. The update formulas are provided in Equations 4, 5 and 6.

$$\begin{aligned} b2_{i,j} &= \sum_{x \le i, y \le j} a_{x,y}^2 \\ &= b2_{i,j-1} + b2_{i-1,j} + a_{i,j}^2 - b2_{i-1,j-1} \end{aligned} \tag{4}$$

where $b2_{i,j}$ is the accumulator value at location i, j for the square image, and $a_{i,j}$ is the input image intensity at i, j.

$$S2_{i,j} = b2_{i+w/2,j+h/2} + b2_{i-w/2,j-h/2} - b2_{i-w/2,j+h/2} + b2_{i+w/2,j-h/2} \tag{5}$$

where $S2_{i,j}$ is the sum of the squared pixel values in the neighborhood centered at i, j.

$$\sigma_{i,j} = \sqrt{\frac{S2_{i,j} - \frac{S_{i,j}^2}{w.h}}{w.h - 1}} \tag{6}$$

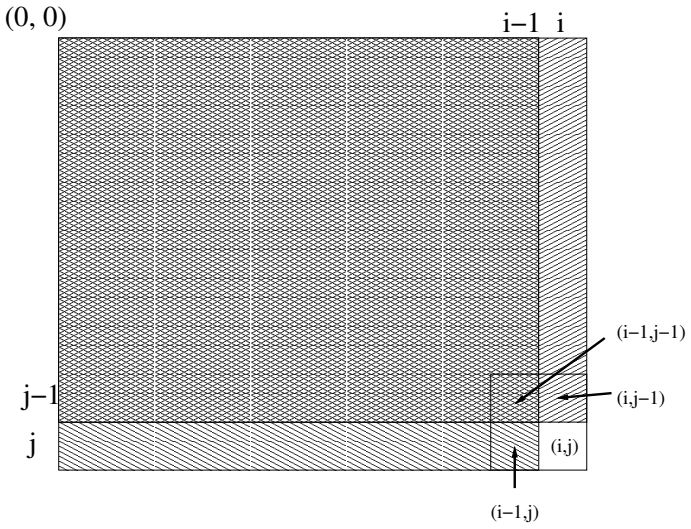


Figure 4: Recursive computation of the accumulator image - value at (i, j) computed using Equation 1. The double hashed region is subtracted because it is included in both $b_{i,j-1}$ and $b_{i-1,j}$.

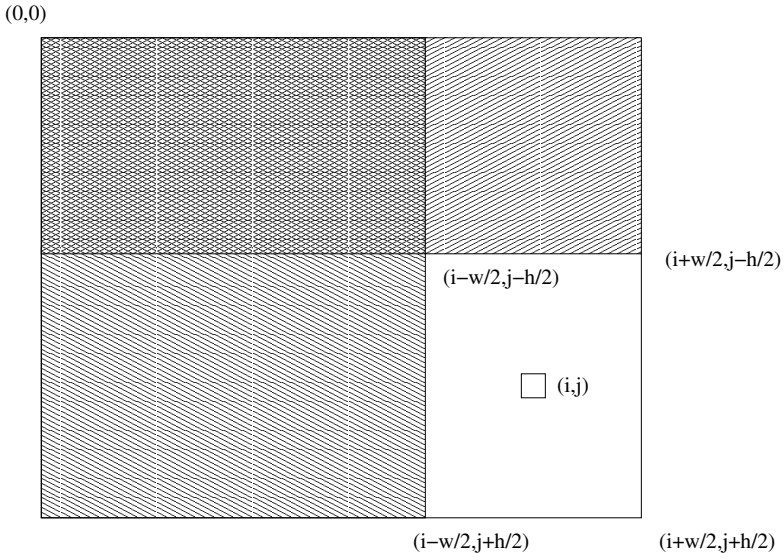
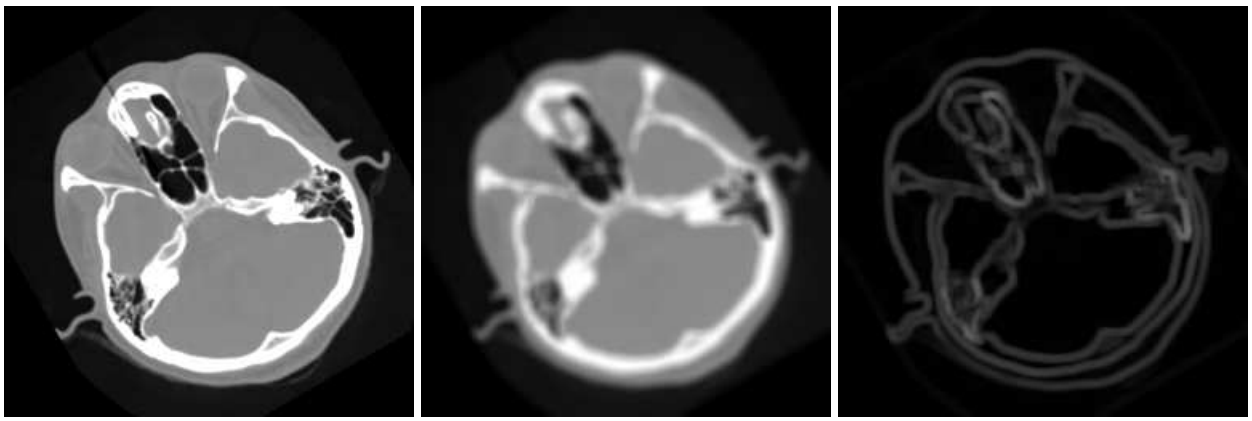


Figure 5: Computation of mean using the accumulated image - value at (i, j) computed using Equation 3.

where $\sigma_{i,j}$ is the output of the standard deviation filter at location i, j.

This approach to convolution has been used recently in face detection applications [13], where rectangular filters of many different sizes and shapes were needed and could be computed from the same accumulation image.



(a) Input (b) Mean (c) Sigma

Figure 6: A mean transform (b) and a standard deviation transform (c) with a kernel of radius 3.

6 ITK implementation

The filters discusses in this article are contained in 3 packages – *consolidatedMorphology*[1], *fastRankMean* and *boxConvolution*. The most recent versions of these packages can be obtained from `http://voxel.jouy.inra.fr/darcsweb/`[2]. All of the packages include multithreaded filters and include many examples and tests. Unless indicated otherwise, all filters will behave basically the same way as other kernel filters, requiring either a radius or a neighborhood to be specified.

6.1 Common filters

All the filters provided with those packages are implemented as subclasses of one of two new classes: *itk::BoxImageFilter* and *itk::KernelImageFilter*. *itk::BoxImageFilter* is a subclass of *itk::ImageToImageFilter* and provides some features used in all the box based filters:

- the management of the radius of the box,
- and the enlargement of the input requested region, according to the box size.

[1] An earlier version of the *consolidatedMorphology* package has been submitted to the InsightJournal. However many improvements and bug fixes have been included since then.

[2] The most recent versions can be obtained using darcs [1] with the command *darcs get http://voxel.jouy.inra.fr/darcs/contrib-itk/package*, where *package* must be replace by *consolidatedMorphology*, *fastRankMean* or *boxConvolution*.

itk::KernelImageFilter is a subclass of *itk::BoxImageFilter*, and provides some features used in all the kernel based filters, in addition to the features of the *itk::BoxImageFilter* class:

- the management of the kernel,
- a redefined SetRadius() method to create a box kernel with the provided radius,
- a default box kernel of radius 1[3].

A third filter, *itk::SeparableImageFilter*, is used to easily implement a separable version of an algorithm, by using its basic implementation in a subclass of *itk::BoxImageFilter*, or any other subclass of *ImageToImageFilter* with a *SetRadius()* method.

Some filters already in ITK have been modified to use *itk::BoxImageFilter* or *itk::KernelImageFilter*[4]. Their API is not modified.

6.2 Morphology filters – consolidatedMorphology

The morphology filters utilize a new class – *FlatStructuringElement* to describe, arbitary and decomposable structuring elements. It provides the following methods to create structuring elements with different characteristics:

- `Ball()` generate a ball structuring element (a circle in 2D). It takes the radius of the structuring element as parameter.
- `Box()` produces a box structuring element. It takes the radius of the structuring element as parameter.
- `Poly()` produces an approximation to a circle or sphere using line structuring elements, taking the desired radius and number of lines as parameters. In the 3D case an approximation of a sphere will be attempted. Caution is advised because this is not well tested yet.
- `Cross()` produces a cross structuring element. It takes the radius of the structuring element as parameter.
- `Annulus()` produces an annulus structuring element. It takes the radius of the structuring element as parameter, as well as the thickness of the annulus, and a boolean to indicate whether the center neighbors should be set inside the neighborhood or not[5].
- `FromImage()` produces a structuring element from an image. The image is passed as parameter to the method. A optional parameter can also be passed: the pixel value to be considered as the foreground value in the image. This value defaults to `NumericTraits< PixelType >::max()`. This lets the user visualize the structuring element.

The algorithm used by a filter is set by the *SetAlgorithm* method, with options of *BASIC*, *HISTO*, *VHGW* and *ANCHOR*, for the traditional ITK implementation, the moving histogram algorithm, the van Herk, Gil, Werman algorithm and the anchor algorithm respectively. The VHGW and ANCHOR algorithms can only

[3] This feature should help many ITK beginers who are often *not* providing a structuring element to the morphology filters, and thus get difficultly understandable crashes.

[4] itk::MedianImageFilter, itkMeanImageFilter, itk::NoiseImageFilter and itk::MorphologyImageFilter.

[5] The *Annulus()* constructor is a contribution of Dan Mueller.

be used for Box and Poly structuring elements. VHGW is more thoroughly tested. The HISTO algorithm produces the identical results to BASIC, and is always faster. It is the default and should be used if you need precisely defined, non rectangular, structuring elements. If a Box structuring element is needed then VHGW or ANCHOR should be used, as they offer the best performance and are exact in that case. Poly structuring elements are approximations of circles that will offer improved performance with VHGW or ANCHOR algorithms, but the user will need to consider whether the approximation is suitable for the application in question.

The availability of line morphology operations means that new operations, such as "union of openings" can be implemented easily.

6.3 Rank and mean filters – fastRankMean

The *fastRankMean* packge uses the sliding window method to implement rank and mean filters for arbitary kernel shapes. It also uses line versions of these kernels to provide a "separable rank" filter – an fast approximation to a real rank filter as discussed in Section 4. There are also a number of filters that accept a mask input and return the rank or mean value of the intersection of kernel and mask. These are experimental and haven't been thoroughly tested. The filters are *itkRankImageFilter, itkMovingWindowMeanImageFilter, itkFastApproxRankImageFilter, itkFastApproxMaskRankImageFilter* and *itkMaskedRankImageFilter*. itkRankImageFilter implements arbitary shaped kernels, itkFastApproxRankImageFilter implements the separable approximation and itkMovingWindowMeanImageFilter implements an arbitary shaped kernel mean.

6.4 Rectangular convolution – boxConvolution

The *boxConvolution* uses the accumulation method described in Section 5 to implement rectangular mean and variance computations. The filters implement different boundary conditions to *itkMeanImageFilter*, with the mean of the kernel inside the image being computed. The filters in this package are multithreaded and deal with arbitrary dimensions. The current filters do not support the mode of operation used in face recognition application in which an accumulation image is repeatedly "queried" for mean values of various sized kernels. However introducing such a capability is relatively easy.

6.5 Extended languages support

All the new filters provided are wrapped using WrapITK's external projects [4] and have been successfully tested with python.

7 Performance

All the execution times in this section are measured on a computer with four Intel®Xeon®CPU cores at 2.33GHz with 4MB cache, running CentOS 4.4 64 bits. Unless specified in the description, the tests are forced to run on a single core.

7.1 Morpholgy

Execution times for the dilation, the opening, and the gradient transform are shown in Tables 1, 3 and 2, and in Figure 7. The predicted linear complexity for the number of neighbors is observed for the basic algorithm, as well as the constant complexity of the van Herk / Gil Werman and the anchor algorithm. The linear complexity for the number of pixels added and removed per translation is observed for the moving histogram algorithm, as expected.

A good threading support is observed for all the implementations, as shown in Table 4 and Figure 7.

7.2 Rank and Mean filters

Relative times for sliding window median and mean filters and separable median filters are shown in Tables 6 and 7 and in Figure 8. The predicted linear complexity is observed for the sliding window approaches (complexity of direct approach is $O(n^2)$ in 2D, reduced to $O(n)$ by using the sliding window. The separable median exhibits a runtime independent of kernel size, as expected.

7.3 Box Convolution

The relative times for optimized and standard convolution are shown in Table 9 and Figure 9, and demonstrate significantly improved performance for kernel radii greater than 1. The times support the theoretical prediction that the complexity of the algorithm is independent of kernel size.

A good threading support is observed for all the implementations, as shown in Table 10 and Figure 9.

7.4 Timing notes

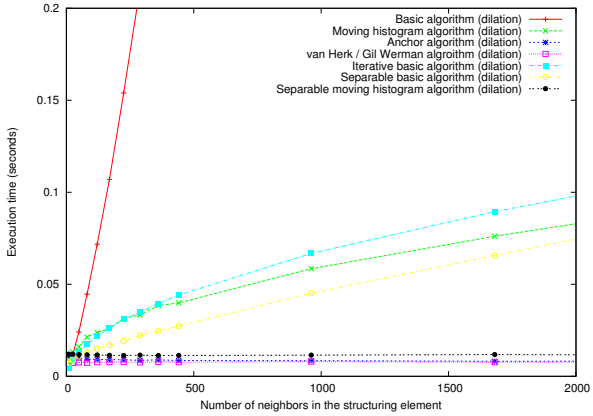
You may have noticed that the optimized filters of radius 1 seem to regularly perform slower than their larger counterparts. We can think of no reason for this. The algorithms that have kernel size independent complexity should not be slower for small kernels than large ones because the only difference occurs at borders and when the sliding window is being created, and both situations favour small kernels. We can only guess that there are caching issues involved.

8 Implementation Notes

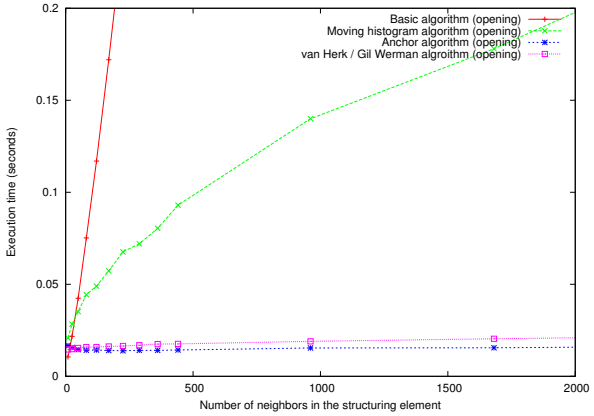
A lot of effort has been spent implementing and optimizing these filters while attempting to maintain good ITK style. This section summarizes some observations and experiences:

8.1 Moving histograms

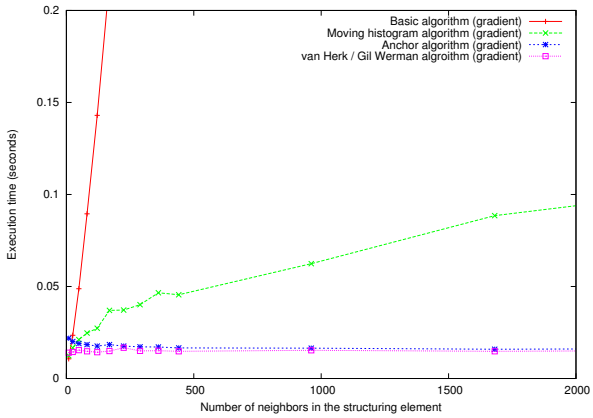
The most obvious way of implementing moving histograms involves neighborhood iterators. However the resulting performance wasn't good – it appears that neighborhood iterator complexity is proportional to radius rather than the number of active neighbors. A method using lists of offsets combined with image Get/SetPixel methods approach was used instead.



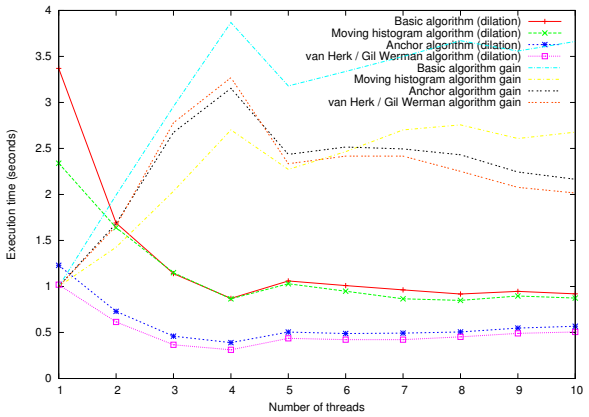
(a) Dilation



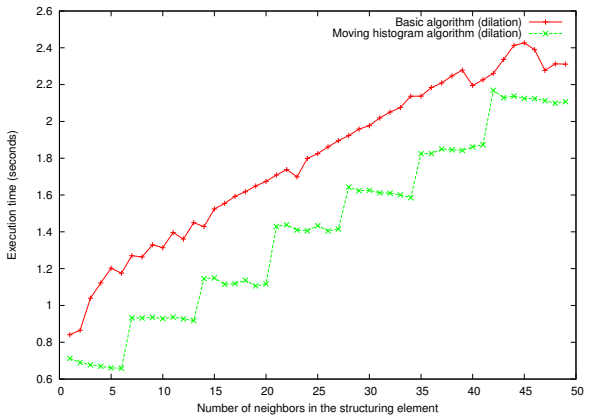
(b) Opening



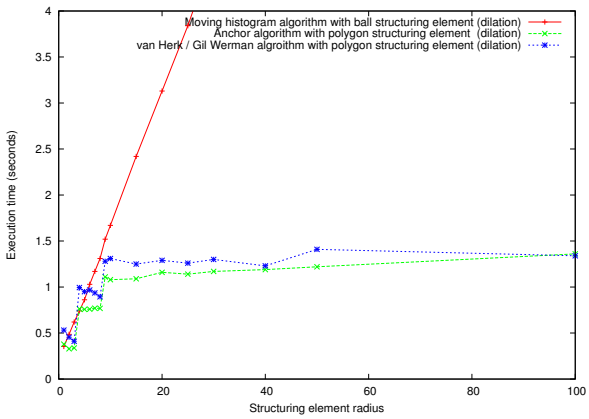
(c) Gradient



(d) Threads

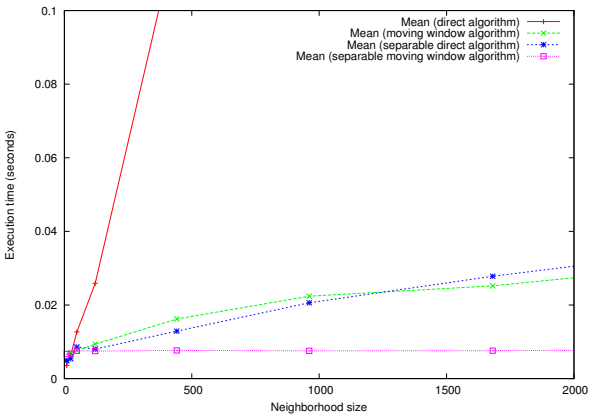


(e) Number of neighbors

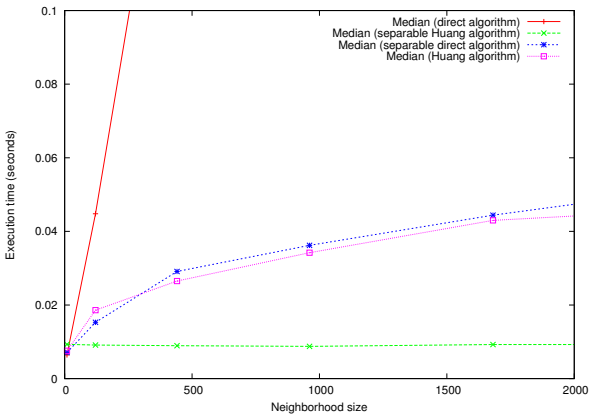


(f) Polygon

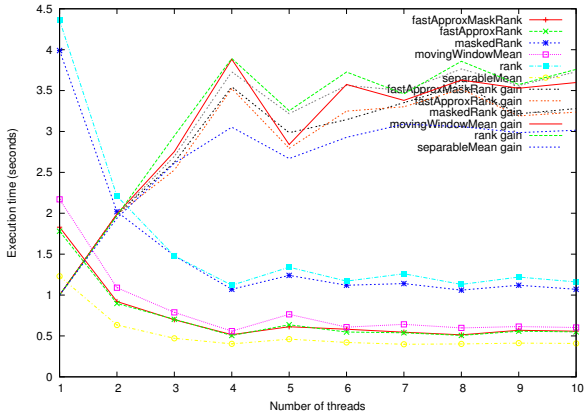
Figure 7: Execution times with a box structuring element of increasing size for different algorithms of dilation (a), opening (b) and gradient (c). Execution times with and increasing number fo threads (d). Execution times with an increasing number of neighbors in a box structuring of constant radius (e). Execution times with a polygon structuring element. The polygon approximations use the default number of lines (a maximum of 6 when radius is greater than 8 (f). Results are reported in Tables 1, 3, 2, 4 and 5.



(a) Mean

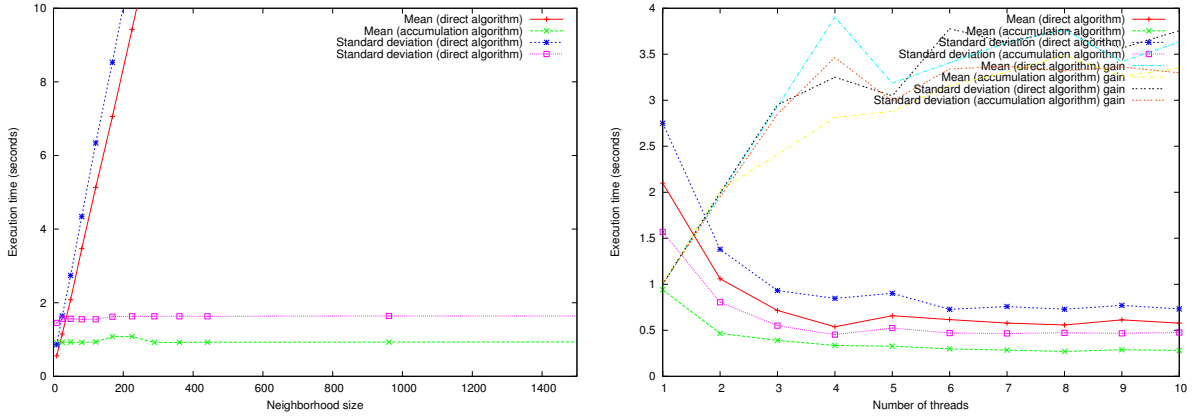


(b) Median



(c) Threads

Figure 8: Execution times for an increasing kernel size with several algorithms of mean filters (a) and median filters (b). Excution times with an an increasing number of threads (c). Results are reported in Tables 6, 7 and 8.



(a) Neighborhood size

(b) Threads

Figure 9: Execution times for and increasing kernel size (a), and an increasing number of threads (b). Results are reported in Tables 9 and 10

The original paper recommended moving the window in a zig-zag pattern, i.e horizontally across a the first row, down a step to the second row and then back, but this involves accessing data in a reverse raster order which can potentially reduce cache performance. An alternative has been implemented that always moves the window in forward raster direction, at the cost of additional histogram copies, has been implemented and exhibits improved performance for large images. Differences for small images weren't detectable.

8.2 16 bit data types

Histograms for 16 bit data types can be implemented using arrays or maps. One would expect arrays to offer a simpler and therefore faster implementation, but our results have been mixed. Presumably the performance depends significantly on voxel statistics.

8.3 Line Operations

The van Herk/Gil and Werman and anchor methods operate on image transects, potentially at any angle. The transect data needs to be extracted from the image and the processed transect written back, with successive parallel transects being processed. Existing iterators don't do this particularly efficiently, so image Get/SetPixel operations are being used. This approach isn't ideal either and needs to be worked on. In general there is no need for boundary checks because the intersection between transect and image region is calculated, so a potentially random access method without boundary checks would be ideal.

Building dimension independent code also adds complexity to this filter which can be avoided in dimension specific implementations. The process of sweeping the transect across the image region is much more complex (and therefore slower) when written in a dimension independent fashion.

The standard ITK threading approach isn't ideal for this style of filtering either – a better option would be splitting the image based on transects instead of blocks. Obviously this sort of change isn't practical.

8.4 Refactoring

The three packages do share some components, and some refactoring is called for. This most widely used components are histograms and sliding window infrastructure, which need to be consolidated.

9 Comments and Conclusions

This article has provided background theory for and implementations of a number of important approaches to kernel based filtering. These methods significantly reduce complexity and execution time, by some orders of magnitude in many cases. These approaches are all well established in the literature but weren't available in ITK. Availability of filtering algorithms which are always fast, irrespective of the kernel size, can make much simpler approaches to many problems practical.

10 Acknowledgments

We thank Dr Pierre Adenot and MIMA2 confocal facilities (http://mima2.jouy.inra.fr) for providing the 3D test image. We are grateful to the INRA MIGALE bioinformatics platform (http://migale.jouy.inra.fr) for providing the computational resources used for the timing tests.

References

[1] http://www.darcs.net. 2

[2] R. Adams. Radial decomposition of discs and spheres. *CVGIP: Graphical models and image processing*, 55(5):325–332, September 1993. 3.2

[3] F.C. Crow. Summed-area tables for texture mapping. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 207–212, New York, NY, USA, 1984. ACM Press. 5

[4] Zachary Pincus Gaëtan Lehmann and Benoit Regrain. Wrapitk - enhanced languages support for the insight toolkit. *Insight Journal*, January - June 2006. 6.5

[5] J. Gil and R. Kimmel. Efficient dilation, erosion, opening and closing algorithms. In *Mathematical Morphology and its Applications to Image and Signal Processing*, pages 301–310. Kluwer Acad. Pub, 2000. 3.3

[6] J. Gil and M. Werman. Computing 2-d min, median, and max filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):504–507, May 1993. 3.3

[7] T.S. Huang, G.Y. Yang, and G.Y. Tang. A fast two-dimensional median filtering algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 27:13–18, 1979. 4

[8] L. Ibanez and W. Schroeder. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, 2003.

[9] P.M. Narendra. A separable median filter for image noise smoothing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:20–29, 1981. 4

[10] M. Van Droogenbroeck and M. Buckley. Morphological erosions and openings: fast algorithms based on anchors. *Journal of Mathematical Imaging and Vision, Special Issue on Mathematical Morphology after 40 Years*, 22(2-3):121–142, May 2005. Algorithms in ANSI C code are available in libmorpho. 3.3

[11] M. Van Droogenbroeck and H. Talbot. Fast computation of morphological operations with arbitrary structuring elements. *Pattern Recognition Letters*, 17(14):1451–1460, 1996. 3.1

[12] M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recognition Letters*, 13(7):517–521, July 1992. 3.3

[13] P. Viola and M. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004. 5

Appendix

Radius	Basic	Iterative	Separable	Histogram	Separable histogram	Anchor	van Herk / Gil Werman
1	0.00437	0.00436	0.00734	0.00987	0.0121	0.0111	0.00715
2	0.0118	0.00872	0.00901	0.0131	0.0121	0.0101	0.00733
3	0.0241	0.013	0.0108	0.0161	0.0118	0.00979	0.00756
4	0.0446	0.0175	0.0127	0.0213	0.0117	0.00922	0.00736
5	0.0718	0.0219	0.0149	0.0237	0.0116	0.00923	0.00768
6	0.107	0.0262	0.017	0.0262	0.0114	0.00906	0.00768
7	0.154	0.0311	0.0192	0.0315	0.0113	0.00899	0.00779
8	0.211	0.0349	0.0223	0.0334	0.0115	0.00865	0.00759
9	0.281	0.0395	0.0246	0.0385	0.0113	0.00892	0.00784
10	0.365	0.0441	0.0273	0.0399	0.0113	0.0086	0.00767
15	1.03	0.0668	0.0451	0.0585	0.0115	0.00851	0.00798
20	2.23	0.0895	0.0657	0.0761	0.0118	0.00819	0.0077
25	4.11	0.114	0.0911	0.0958	0.0114	0.00826	0.00778
30	6.84	0.139	0.121	0.108	0.0115	0.00847	0.00801
40	15.5	0.194	0.195	0.145	0.0116	0.00872	0.00811
50	29.4	0.257	0.283	0.185	0.0118	0.00858	0.00798
100	227	0.719	0.974	0.348	0.0127	0.00858	0.0082

Table 1: Execution times in seconds for dilation by boxes using several algorithm implementations applied to the 256×256 cthead image.

Radius	Basic	Histogram	Anchor	van Herk / Gil Werman
1	0.0107	0.0124	0.0219	0.014
2	0.0235	0.0167	0.0201	0.0144
3	0.0488	0.0211	0.019	0.0154
4	0.0895	0.0247	0.0184	0.0148
5	0.143	0.0273	0.0176	0.0143
6	0.216	0.037	0.0184	0.0149
7	0.308	0.0372	0.0176	0.0167
8	0.423	0.0401	0.0172	0.015
9	0.563	0.0466	0.0171	0.0151
10	0.73	0.0455	0.0166	0.0148
15	2.07	0.0624	0.0165	0.0153
20	4.46	0.0885	0.0159	0.0148
25	8.22	0.104	0.0163	0.0153
30	13.6	0.118	0.0161	0.0151
40	30.9	0.153	0.0163	0.0153
50	58.6	0.196	0.0168	0.0156
100	452	0.339	0.0163	0.0157

Table 2: Execution times in seconds for morphological gradient transformation by boxes using several algorithm implementations applied to the 256×256 cthead image.

Radius	Basic	Histogram	Anchor	van Herk / Gil Werman
1	0.0105	0.0208	0.0167	0.0146
2	0.0217	0.0283	0.0157	0.0148
3	0.0424	0.0352	0.0145	0.0154
4	0.0752	0.0444	0.0142	0.0159
5	0.117	0.049	0.0143	0.0159
6	0.172	0.0573	0.014	0.0162
7	0.24	0.0676	0.0139	0.0165
8	0.324	0.072	0.0141	0.0169
9	0.425	0.0804	0.0141	0.0175
10	0.546	0.093	0.0143	0.0176
15	1.5	0.14	0.0154	0.019
20	3.19	0.178	0.0155	0.0204
25	5.9	0.235	0.0164	0.0219
30	9.9	0.274	0.0175	0.0236
40	23	0.383	0.0193	0.0272
50	45.3	0.522	0.0213	0.0298
100	421	1.6	0.0346	0.0493

Table 3: Execution times in seconds for morphological opening by boxes using several algorithm implementations applied to the 256×256 cthead image.

Threads	Basic	Histogram	Anchor	van Herk / Gil Werman
1	3.37	2.34	1.23	1.02
2	1.69	1.64	0.73	0.615
3	1.14	1.15	0.46	0.367
4	0.871	0.866	0.39	0.312
5	1.06	1.03	0.505	0.437
6	1.01	0.949	0.489	0.422
7	0.963	0.866	0.493	0.422
8	0.918	0.849	0.506	0.453
9	0.947	0.897	0.548	0.491
10	0.92	0.874	0.568	0.506

Table 4: Execution times in seconds for morphological opening by boxes using several algorithm implementations applied to the 3200×2400 image.

Radius	Histogram	van Herk / Gil Werman	Anchorn
1	0.354	0.379	0.532
2	0.486	0.329	0.455
3	0.62	0.338	0.411
4	0.74	0.763	0.995
5	0.862	0.757	0.95
6	1.03	0.759	0.968
7	1.17	0.77	0.936
8	1.31	0.768	0.894
9	1.52	1.11	1.28
10	1.67	1.08	1.31
15	2.42	1.09	1.25
20	3.13	1.16	1.29
25	3.84	1.14	1.26
30	4.64	1.17	1.3
40	5.95	1.19	1.23
50	7.6	1.22	1.41
100	19	1.36	1.34

Table 5: Execution times in seconds for morphological opening by boxes using several algorithm implementations applied to the 256×256 cthead image.

Radius	Direct	Moving window	Separable direct	Separable moving window
1	0.00355	0.00498	0.00487	0.00668
2	0.00642	0.00706	0.0054	0.00679
3	0.0127	0.00761	0.00861	0.00756
5	0.0259	0.00936	0.00803	0.00749
10	0.121	0.0162	0.0129	0.00765
15	0.35	0.0224	0.0206	0.00755
20	0.771	0.0252	0.0278	0.0076
40	5.59	0.059	0.0699	0.00871

Table 6: Execution times in seconds for mean filters using direct and sliding window implementations applied to the 256×256 cthead image.

Radius	Direct	Huang	Separable direct	Separable huang
1	0.00642	0.00758	0.0071	0.0093
5	0.0448	0.0186	0.0153	0.00913
10	0.177	0.0265	0.0291	0.00895
15	0.445	0.0342	0.0362	0.00876
20	0.93	0.043	0.0444	0.00925
40	6.21	0.061	0.0899	0.00936

Table 7: Execution times in seconds for median filters using direct and sliding window implementations applied to the 256×256 cthead image. The separable median is an approximation and therefore doesn't produce the same result as the direct or sliding algorithms.

Threads	fastApproxMaskRank	fastApproxRank	maskedRank	movingWindowMean	rank	separableMean
1	1.83	1.78	3.99	2.17	4.36	1.23
2	0.921	0.898	2.02	1.09	2.21	0.634
3	0.698	0.704	1.48	0.788	1.48	0.471
4	0.516	0.505	1.07	0.559	1.12	0.403
5	0.613	0.637	1.24	0.764	1.34	0.461
6	0.582	0.548	1.12	0.607	1.17	0.42
7	0.546	0.539	1.14	0.642	1.26	0.398
8	0.513	0.506	1.06	0.598	1.13	0.402
9	0.569	0.558	1.12	0.615	1.22	0.412
10	0.558	0.55	1.07	0.603	1.16	0.408

Table 8: Execution times in seconds for median and mean filters using direct and sliding window implementations applied to the 3200×2400 image.

Size	Direct Mean	Box Mean	Direct sigma	Box sigma
1	0.557	0.916	0.858	1.45
2	1.15	0.924	1.64	1.56
3	2.08	0.932	2.74	1.56
4	3.47	0.92	4.34	1.55
5	5.13	0.935	6.34	1.55
6	7.06	1.08	8.53	1.62
7	9.42	1.08	11.3	1.63
8	12.2	0.922	14.5	1.63
9	15.3	0.924	18	1.63
10	18.9	0.925	22.1	1.63
15	43.3	0.929	50.2	1.64
20	80.1	0.936	91.8	1.64
25	131	0.939	149	1.64

Table 9: Execution times in seconds for mean and standard deviation using direct and box convolution implementations applied to the 3200×2400 image.

Size	Direct Mean	Box Mean	Direct sigma	Box sigma
1	2.1	0.942	2.75	1.57
2	1.06	0.466	1.38	0.805
3	0.716	0.391	0.932	0.551
4	0.538	0.335	0.846	0.453
5	0.659	0.327	0.902	0.525
6	0.616	0.298	0.728	0.47
7	0.579	0.285	0.758	0.466
8	0.558	0.27	0.729	0.473
9	0.614	0.289	0.771	0.467
10	0.578	0.281	0.732	0.476

Table 10: Execution times in seconds for mean and standard deviation using direct and box convolution implementations applied to the 3200×2400 image.

Radial Thickness Calculation and Visualization for Volumetric Layers

Release 0.01

Defeng WANG[1,2], Lin SHI [1] and Pheng Ann HENG[1,2,3]

July 2, 2007

[1]Department of Computer Science and Engineering, The Chinese University of Hong Kong
[2]Shun Hing Institute of Advanced Engineering, The Chinese University of Hong Kong
[3]Shenzhen Institute of Advanced Integration Technology, Chinese Academy of Sciences/The Chinese University of Hong Kong

Abstract

Volumetric layers often encountered in medical image analysis are characterized by double and nested bounding surfaces. The thickness of a volumetric layer at a point on the bounding surface is the distance from that point to the opposite surface. There exist several definitions for the layer thickness. A newly proposed thickness definition is the radial thickness, which is defined as the distance between each pair of corresponding points on the two surfaces with the same polar coordinate. The thickness values calculated by the radial thickness definition are unique and does not depend on the starting surface. In this paper, we describe a class for calculating the radial thickness of one volumetric layer represented as coupled and nested triangle meshes.

The class, `vtkRadialThicknessCalculate`, is implemented using the Visualization Toolkit (VTK), `www.vtk.org`. In this document, we describe the radial thickness calculation algorithm and provide the user with the source code and the input data to reproduce the results. The radial thickness calculation described in this paper has a variety of applications including the thickness calculation for the skull vault, which is the original motivation for this work.

Contents

1 Introduction **2**

2 Algorithm Description **3**

3 User's Guide **4**

4 Results **5**
4.1 Required Packages 5
4.2 Results on Human Skull Vaults 5

5 Conclusion **6**

6 Acknowledgements **6**

A Implementation Notes **6**

1 Introduction

Volumetric layers encountered in medical image analysis usually contains double and nested bounding surfaces, for instance, the skull vault [5], the cerebral cortex [6], and the myocardium of the left ventricle [1]. In the recent work [6], we described a generic automatic landmarking method for structures with coupled surfaces by minimizing the description length. In that method, the local thickness gradients are considered in formulating the description length. Once the landmark on one surface is determined, its counterpart on the other surface can be found directly. It has been empirically shown that considering the thickness information in landmarking volumetric layers will lead to models with higher quality compared with performing the landmark optimization on two surfaces independently. In that work, we proposed a new thickness measurement that is suitable to measure the thickness of the volumetric layers. In this paper, we will give the detailed implementation of this thickness definition as well as the usage of the appended source code.

The thickness of a volumetric layer at a point on the bounding surface is the distance from that point to the opposite surface. There exist several definitions for the layer thickness. The most popular ones are the closest thickness (T_{close}) and the normal thickness (T_{normal}) [2]. T_{close} is the distance from a point on one surface to the closest point on the other. T_{normal} is the distance from a point on one surface to the point on the other in the direction of the surface normal. To find a generic measure that performs reasonably on every type of layers is impractical. We determine the layer thickness as the distance between each pair of corresponding points on the two surfaces with the same polar coordinate. This measure is named as the radial thickness (T_{radial}). We illustrate the measures of T_{close}, T_{normal}, and T_{radial} on an axial plane of the skull boundary (see Figure 1). Different from measures T_{close} and T_{normal} that depend on the starting surface, the T_{radial} is unique and landmarks are grouped in pairs using this measurement.

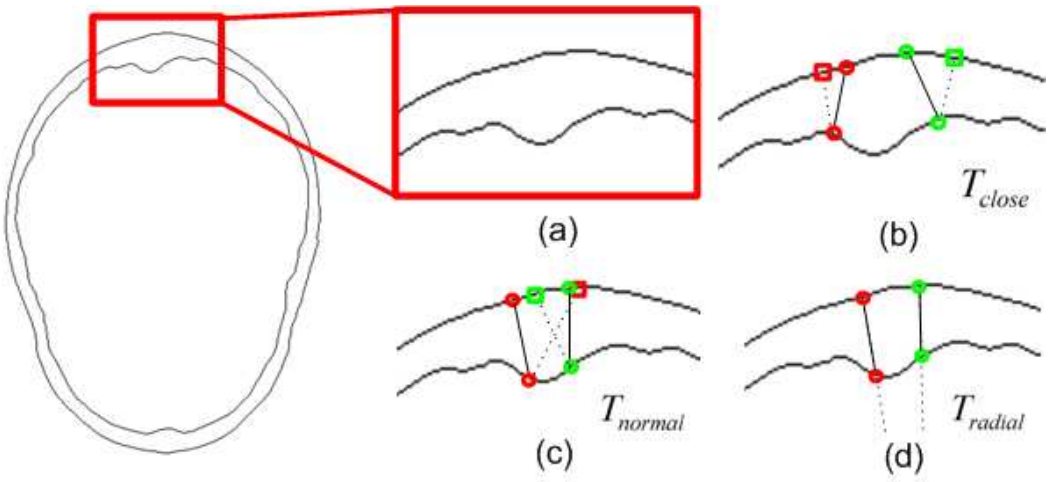


Figure 1: Different thickness definitions illustrated on a part of one axial plane of the skull boundary: (a) the coupled surfaces; (b) the closest thickness measure; (c) the normal thickness measure; (d) the radial thickness measure used in this paper.

2 Algorithm Description

To calculate the radial thickness, the two coupled triangle meshes are treated as a master mesh and a supplementary one. As illustrated in Figure 1 (d), each ray is determined by the vector from the center to each vertex in the master mesh. The radial thickness calculation is to determine the distance between each vertex on the master mesh and the intersection point of the corresponding ray with the supplementary surface. Specifically, this problem can be reduced to the problem of checking if there is any intersection of a ray and a triangle in the supplementary mesh, and determining its coordinates if it exists. We adopt the fast and minimum-storage algorithm of ray/triangle intersection examining proposed in [3]. A ray emitted from the center C_0 to one vertex V in the master mesh is defined as

$$R(t) = C_0 + tD, \tag{1}$$

where $D = (V - C_0)/\|V - C_0\|$ is the normalized direction. A point, $T(u,v)$, on a triangle defined by three vertices V_0, V_1, V_2 is represented by

$$T(u,v) = (1-u-v)V_0 + uV_1 + vV_2, \tag{2}$$

where (u,v) are the barycentric coordinates, which satisfies that u and v are non-negative, as well as the sum of u and v is not greater than one. It is apparent that the intersection between the ray $R(t)$ and the triangle $T(u,v)$, is equivalent to $R(t) = T(u,v)$, which leads to the following equation

$$C_0 + tD = (1-u-v)V_0 + uV_1 + vV_2. \tag{3}$$

By rearranging the terms in Eqn.(3), we can easily have,

$$[-D, V_1 - V_0, V_2 - V_0] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0. \tag{4}$$

By using the Cramer's rule and defining $E_1 = V_1 - V_0$, $E_2 = V_2 - V_0$, and $T = O - V_0$, we obtain the solution to Eqn.(4) as

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{[-D, E_1, E_2]} \begin{bmatrix} |T, E_1, E_2| \\ |-D, T, E_2| \\ |-D, E_1, T| \end{bmatrix}. \tag{5}$$

Considering that

$$C_1, C_2, C_3 = -(C_1 \times C_3) \cdot C_2 \tag{6}$$

$$= -(C_3 \times C_2) \cdot C_1, \tag{7}$$

we can further simplify Eqn.(5) to speed up the computations in the following form

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{[P \cdot E_1]} \begin{bmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{bmatrix}, \tag{8}$$

where $P = (D \times E_2)$ and $Q = T \times E_1$.

3 User's Guide

The `vtkRadialThicknessCalculate` class takes one VTK mesh input as the master mesh and the other VTK mesh input as the supplementary mesh. The output is the radial thickness values, or the normalized ones between 0 and 1. Each radial direction is determined by the vector from the center to each vertex in the master mesh. There is a distance value for every direction. If there is no intersection between the ray and the supplementary mesh, the radial thickness value will be set to 0. Alternatively, the user can also specify the directions in a text file, in which each row with 3 real numbers represent a direction.

Before calculating the radial thickness, we need to include the header file

```
#include "vtkRadialThicknessCalculate.h",
```

and declare an instance in the following way,

```
vtkRadialThicknessCalculate radialThicknessCal;
```

The master mesh and the supplementary mesh can be set by

```
radialThicknessCal.SetMasterMesh(inner); \\
// 'inner' is an inner skull mesh in the format of VTK
```

and

```
radialThicknessCal.SetSupplementaryMesh(outer); \\
// 'outer' is an outer skull mesh in the format of VTK
```

Users can specify the file name to save the radial thickness values by

```
radialThicknessCal.SetThicknessFileName("thicknessFile");
```

Actually there will be two output files. One is "thicknessFile.txt", which contains the radial thickness values, while the other output is "thicknessFile_normalized.txt", which saves the normalized ones.

Finally, the thickness calculation can be triggered by

```
radialThicknessCal.StartThicknessCalculate();
```

Aside from using the vector from the center to each vertex in the master mesh, which is the default mode, the user can also explicitly specify the directions by setting a direction file in the following way

```
radialThicknessCal.SetDirectionsFileName("directions");
```

4 Results

In this section, we firstly describe the packages required to generate the results presented in this study. The radial thickness calculation and visualization is performed on the human skull vaults afterwards.

4.1 Required Packages

In order to reproduce the results presented in this paper, the user needs to compile and run the attached code with the following packages:

- CMake 2.4.6
- Visualization Toolkit VTK 5.0.3
  The class `vtkRadialThicknessCalculate` is based on the Visualization Toolkit. The output can be a file containing the normalized radial distances according to the vertex order in the master mesh, or according to the direction order if the direction file is specified. The distance values can be plotted on the master mesh to visualize the distance variation explicitly.
- KWMeshvisu [4]
  KWMeshvisu is adopted to visualize the calculated radial distances on the master mesh. Each radial thickness for the corresponding vertex in the master mesh is color-coded for visualization. This kind of thickness representation provides a qualitative understanding of the layer data, such as the regions with high and low thickness values. The normalized thickness file is in the right form as required by KWMeshvisu, which looks as follows,

  ```
  NUMBER_OF_POINTS = 5685
  DIMENSION = 1
  TYPE = Scalar
  ```

- Neurolib [1]
  The VTK2Meta tool in the MetaMeshTools project from the Neurolib package is adopted to convert the mesh data from the VTK format to the Meta format, which can be loaded in KWMeshvisu for visualization.

4.2 Results on Human Skull Vaults

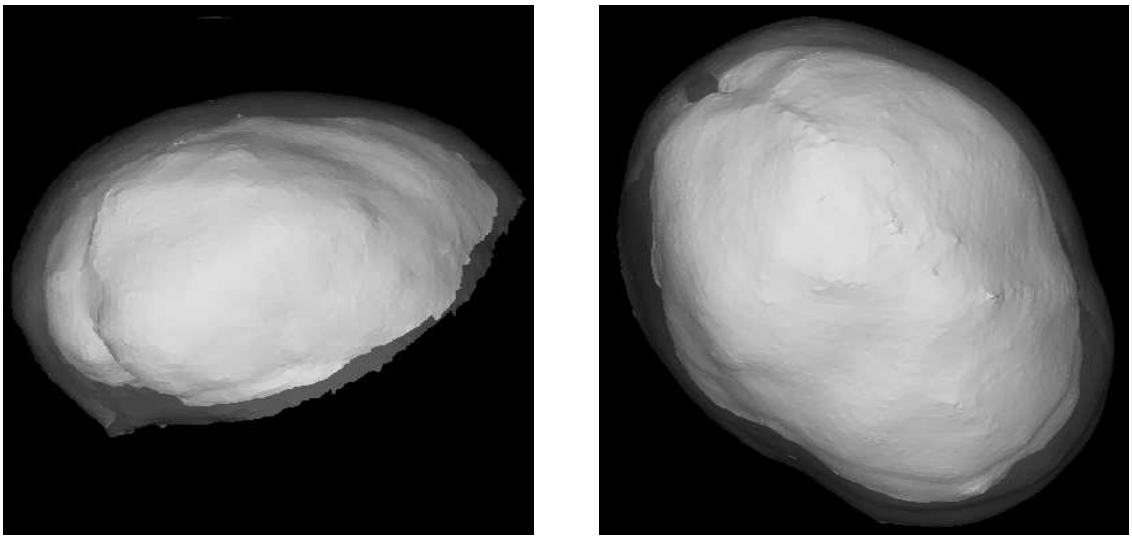
To illustrate the applicability of the proposed class, we apply it to calculate the local thickness of the human skull vault, which is defined as the upper part of the skull and is an open coupled-surface structure. The skull volume was segmented from the head CT data acquired at the Prince of Wales Hospital, Hong Kong. The field of view of the CT data is 512×512 and the voxel size is $0.49\text{mm} \times 0.49\text{mm} \times 0.63\text{mm}$.

Figure 2 shows the coupled (inner and outer) surfaces of the skull vault viewed from the back and the top respectively. The radial thickness values are calculated using the algorithm described in this paper. Figure 3 and 4 present the color-coded thickness values on the inner and outer surfaces of the skull vault, respectively. In Figure 3, the inner surface is taken to be the master surface, while the outer surface is the selected to be the master surface in Figure 4. Note that in our program, if there is no intersection between the ray and the

[1] http://www.ia.unc.edu/dev/

supplementary surface, we artificially set the thickness value at that direction to be zero, which causes the appearance of a ribbon of zero values in the inner surface of the skull vault as shown in Figure 3(a).

From the results, we can find that the resultant radial thickness values represent the layer thickness reasonably, which is consistent with our observation in Figure 2. Thus, it can be concluded that for the structures with a near-spherical morphology radial thickness is a suitable thickness measurement. Another observation is no matter whether the master surface is the inner surface or the outer surface, the resultant thickness value for each particular vertex will not be altered, except in the marginal regions.



(a) (b)

Figure 2: Coupled surfaces of the skull vault: (a) the back view; (b) the top view.

5 Conclusion

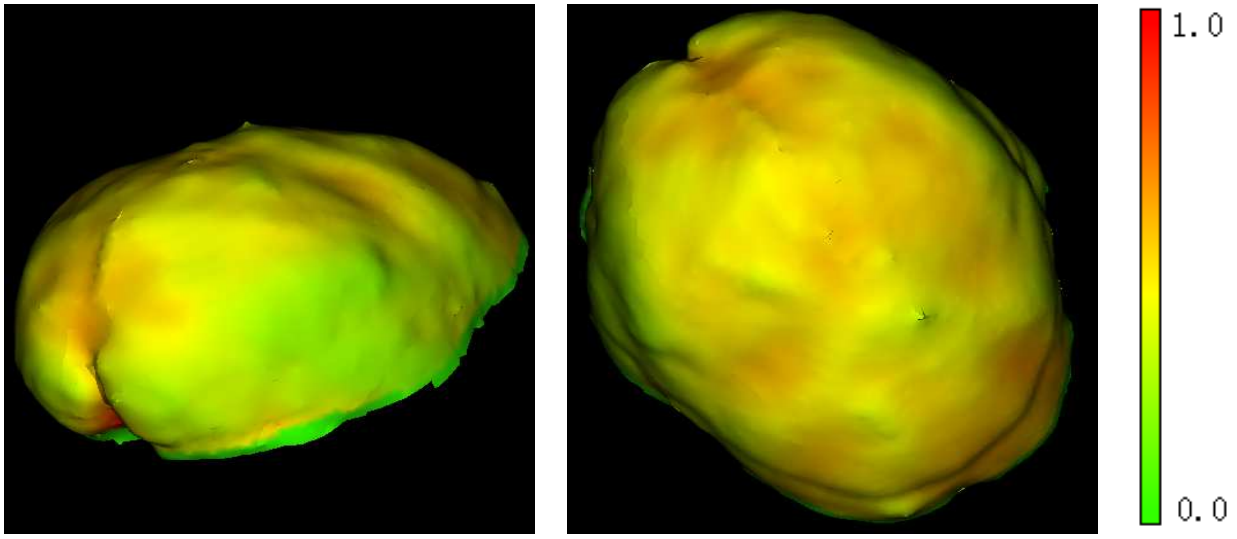
In this paper, we provide the implementation of a new thickness definition for the volumetric layer structures proposed in [6]. This implementation is encapsulated in the `vtkRadialThicknessCalculate` class, which also contains a realization of the fast ray/triangle intersection algorithm [3]. The provided class is potentially applicable to various problems involving the assessment of the thickness of the volumetric layer structures.

6 Acknowledgements

The work described in this paper was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4453/06M) and CUHK Shun Hing Institute of Advanced Engineering.

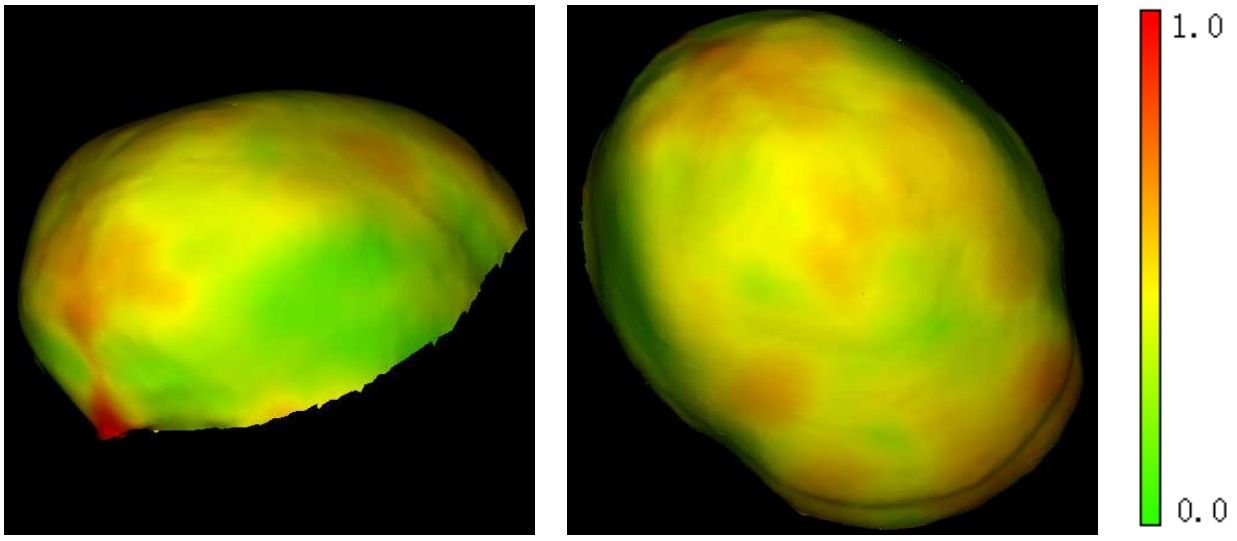
A Implementation Notes

The `vtkRadialThicknessCalculate` class takes two VTK mesh files as inputs. One is the supplementary mesh file, and the other is the master mesh. The center and the radial directions of the volumetric layer can



(a) (b) (c)

Figure 3: The color-coded thickness values plotted on the surface of the inner skull: (a) the back view; (b) the top view; (c) the color bar used to code the normalized thickness values.



(a) (b) (c)

Figure 4: The color-coded thickness values plotted on the surface of the outer skull: (a) the back view; (b) the top view; (c) the color bar used to code the normalized thickness values.

be calculated from the master mesh data. In both the user-specified mode and the default mode, for each direction, the distance from the center to the supplementary mesh is calculated as the distance between the center and the intersection between the ray and a triangle in the supplementary mesh. In the user-specified mode, the distance to the master mesh can be calculated in the same way as to the supplementary mesh. Whereas, in the default mode, the distance from the center to the master mesh in a certain direction can be achieved directly as the distance between the center and the vertex in that direction. Besides using the distance files as outputs, there exist other two APIs to access distances directly:

```
vtkFloatArray* GetRadialThickness();
```

```
vtkFloatArray* GetNormalizedRadialThickness();
```

References

[1] M. Lynch, O. Ghita, and P.F. Whelan. Left-ventricle myocardium segmentation using a coupled level-set with a priori knowledge. *Computerized Medical Imaging and Graphics*, 30:255C262, 2006. 1

[2] D. MacDonald, N. Kabani, and et al. Automated 3-d extraction of inner and outer surfaces of cerebral cortex from MRI. *NeuroImage*, 12(3):340–356, 2000. 1

[3] T. Möller and B. Trumbore. Fast, minimum storage ray/triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997. 2, 5

[4] Ipek Oguz, Guido Gerig, Sebastien Barre, and Martin Styner. KWMeshvisu: A mesh visualization tool for shape analysis. In *IJ - 2006 MICCAI Open Science Workshop*, Kopenhagen, 2006. 4.1

[5] L. Shi, P.A. Heng, T.T. Wong, and et al. Morphometric analysis for pathological abnormality detection in the skull vaults of adolescent idiopathic scoliosis girls. In *Proc. MICCAI*, pages 175–182, Kopenhagen, 2006. 1

[6] L. Shi, D. Wang, P.A. Heng, T.T. Wong, and et al. Landmark correspondence optimization for coupled surfaces. In *MICCAI*, 2007. 1, 5

Multidimensional Arrays and the *nArray* Package

Ofri Sadowsky, Daniel Li, Anton Deguet, Peter Kazanzides

July 2, 2007

Department of Computer Science, Johns Hopkins University
email:ofri@cs.jhu.edu

Abstract

At the Johns Hopkins University's Engineering Research Center for Computer-Integrated Surgical Systems and Technology (ERC-CISST) laboratory, we have designed and developed a platform-independent C++ software package, called the *nArray* library, that provides a unified framework for efficiently working with multidimensional data sets. In this paper, we present and discuss the core elements of the library, including its intuitive and uniform API, efficient arithmetic engine algorithm, and efficient sub-volume algorithm. We then compare the performance of the *nArray* library with that of an existing multidimensional array toolkit, ITK. We conclude that the *nArray* library is more efficient than ITK in many situations, especially in operations on sub-arrays, and that the two packages have comparable performance in many other scenarios. The underlying algorithms, if incorporated in ITK, can help improve its performance.

1 Introduction

Multidimensional data sets are becoming increasingly prevalent in today's world, especially in the fields of computer-integrated surgery and image processing. A canonical example in image processing is a collection of three-dimensional CT scans over time that defines a four-dimensional data set. A three-dimensional image whose pixels each have a vector quantity, such as color or direction, also defines a four-dimensional data set. Such multidimensional data sets grow in size extremely quickly, and so it is important to have a software package that efficiently handles them.

At the Johns Hopkins University's Engineering Research Center for Computer-Integrated Surgical Systems and Technology (ERC-CISST) laboratory, we have designed and developed a platform-independent C++ software package, called the *nArray* (pronounced "EN-array") library, that provides a unified framework for efficiently working with multidimensional data sets. In addition to standard features such as STL-compatible iterators to traverse the data sets, the *nArray* library also provides immutable classes for safe data handling, efficient referencing of sub-volumes of data, layout manipulations, and an extensible generic computational engine.

In this article, we explore the unique layout of the *nArray* library, focusing specifically on the computational engine and the efficient sub-volume algorithm. We then compare the efficiency of the *nArray* package with an existing toolkit with multidimensional array support, the National Library of Medicine's Insight Toolkit

(ITK) [1]. Through this paper, we will see how the efficiency of the *nArray* package proves to be an effective tool for managing multidimensional data sets.

1.1 Motivation

The conventional C notation for a multidimensional array looks similar to the following:

```
unsigned int *** uintVolume;   /* this is a "three-dimensional" array */
```

Elaborate allocation and deallocation methods are required to properly manage such a memory layout. Ultimately, the actual data typically is allocated as a single, continuous memory block, i.e., a "flat" structure. Smaller arrays of pointers are then used to dereference portions, or "slices," of the block. It is clear from the notation that the higher the dimensionality of the dataset, the more complicated the "bookkeeping" of its layout becomes. A main goal in designing the *nArray* package was to simplify this process for the user by providing a templated uniform interface. For example, the same three-dimensional array created with an *nArray* container looks like:

```
/* here the array's type and dimension are template parameters */
vctDynamicNArray<unsigned int, 3> uintVolume;
```

nArray containers of all dimensions have a common interface, i.e., methods and operations. This makes the learning curve shorter, gives scalability, and helps with the debugging. To support this generic usage, we have developed and implemented a set of algorithms to address the issues involved with bookkeeping. Some of these algorithms will be outlined in this paper later.

1.2 Inspiration

APIs and software tools that deal with multidimensional data sets have been around for a while. MATLAB® has many advantages in "rapid prototyping" of algorithms, e.g. in signal processing and in the running of interactive processes, but its poor handling of data structures, flow control, and memory allocation control makes it unsuitable for real-time application development. Other interpreted languages, such as Python, offer a similar functionality for similar performance costs. ITK has gained popularity in recent years in the medical image processing community. It is designed as a *data flow* oriented toolkit, where components are linked in a processing pipeline and final outputs are produced at the end of the pipeline chain. Our feeling is that this programming model is less intuitive to many C/C++ programmers and that it does not provide enough low-level access for optimizing one's code.

The *nArray* package was built as an extension to our prior development of the `cisstVector` library of vectors and matrices written in C++ [2, 3]. Some of the functionality of *nArray*s involves the use of vectors from this library. The `cisstVector` library was inspired by existing toolkits in C++, such as VNL [4] and the Standard Template Library, in many of its aspects. We have tried to implement the good concepts in these libraries and improve those that seemed to need improvement.

A discussion of generic programming cannot be complete without mentioning the work of Todd Veldhuizen, the "father" of template metaprogramming [5]. The `cisstVector` library relies on metaprogramming structures, though different from Veldhuizen's Blitz++ library [6].

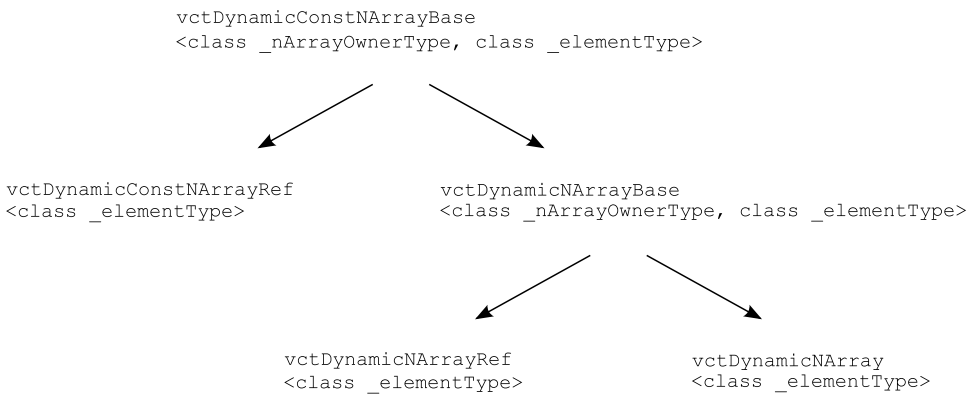


Figure 1: The *nArray* class inheritance hierarchy.

Finally, the development of the *nArray* computational engines, presented briefly in this paper, was an extension of an idea contributed by Robert Jacques, currently at the Johns Hopkins University. Jacques suggested an improvement to our matrix computation engines, which we later extended to *nArray*s.

2 Library Elements

2.1 *nArray* class hierarchy

The *nArray* package interfaces with a given data set through one of three *nArray* containers. These three API-level containers are organized via the five-class inheritance hierarchy shown in Figure 1. Shared methods are defined in the base classes and are inherited by the three child classes, which correspond to the three API-level containers. This inheritance hierarchy maintains a uniform interface across the three *nArray* containers.

The two base classes `vctDynamicConstNArrayBase` and `vctDynamicNArrayBase` contain the bulk of the API methods. The first includes only immutable methods, while the second extends it with mutable methods. These two classes are only accessible for the end-user through the following specializations. The classes `vctDynamicConstNArrayRef` and `vctDynamicNArrayRef` are *overlay* objects, which can be used to access externally allocated memory layouts as if they were multidimensional arrays. For example, `vctDynamicConstNArrayRef` is an immutable overlay, providing read-only structured access to a memory block identified by a `const _elementType *` (i.e., an address). Finally, the class `vctDynamicNArray` is an *allocating* object, and it allocates and releases a storage memory block.

2.2 *nArray* API sampler

The `cisstVector` library, of which the *nArray* package is an extension, was designed with the Abstract Data Type programming paradigm in mind. This means that the objects in the library are data containers and operations between them are methods of their classes. The "ideal" ADT paradigm would use arithmetic operators to resemble a mathematical notation in the programming language wherever possible. To this end, the *nArray* containers use *named methods* for all of its operations, while in applicable cases, overloaded operators that use the named methods as subroutines are additionally defined.

Operation Description	CISST Code	Equivalent C++ Notation	Overloaded Operator	Immutable Array Operands

Addition (two containers or a container and a scalar)	`c1.SumOf(c2, c3);`
	`c1.SumOf(c2, s);`
	`c1.Add(c2);`
	
	`c1.Add(s);`
		`c1 = c2 + c3;`
		`c1 = c2 + s;`
		`c1 += c2;`
		
		`c1 += s;`
			Yes
			Yes
			Yes
			
			Yes
				`c2,c3`
				
				`c2`

Elementwise multiplication	`c1.ElementwiseProductOf( c2, c3);`
	`c1.ElementwiseMultiply(c2);`
		`c1[i] = c2[i]*c3[i];`
		`c1[i] *= c2[i];`
			No
			
			No
				`c2,c3`
				
				`c2`

Division by scalar	`c1.RatioOf(c2, s);` `c.Divide(s);`	`c1 = c2 / s;` `c1 /= s;`	Yes Yes	`c2`
Sum of elements	`s = c.SumOfElements();`	N/A	No	`c`
Largest element	`s = c.MaxElement();`	N/A	No	`c`

Table 1: A sample of the operations in the *nArray* package.

Table 1 shows a few examples of the functional `cisstVector` API. When an equivalent C++ expression is available, it is listed in the third column. If `cisstVector` provides an overloaded operator, it is indicated in the fourth column. The fifth column indicates which operands are immutable, that is, not modifiable by the operation. Usually, if an operand is immutable, it appears in the method's signature as a template `vctDynamicConstNArrayBase`. This allows any of the *nArray* classes to be passed as the actual method parameter. In the table, `c` stands for a generic container, which in the `cisstVector` library can be a (fixed-size or dynamic) vector, a matrix, or an *nArray*; `s` stands for scalars of the same type as that of the array elements, e.g. `double`. Different operands, when they are involved, are distinguished by number. The table shows only a small selection of operations; the complete list is in the library's documentation.

2.3 Layout manipulation

A central feature of the *nArray* containers is their ability to reference other *nArray*s using different *layouts*, or subsections of an existing *nArray* . This is achieved by configuring an overlay *nArray* to span the desired region of an existing *nArray* container. Overlaying allows the user to operate on array elements in-place, without copying them out and in.

A new layout may have the same dimension as its parent container, focusing on a smaller region; or it may have a lower dimension. We call the region focusing a *window* and the dimensionality reduction a *slice*. Combining windows and slices allows the user to specify any subsection of any dimension of an existing *nArray* container. Another useful configuration is changing the order in which *nArray* elements are accessed; this is called *permuting* the order of element access, and is similar to MATLAB's `permute` function, without the memory allocation and copy overhead.[1]

Let us demonstrate these ideas through an example. Suppose we want to select a small region of interest in a sagittal cross-section of a CT volume. Since a CT volume is by default created by stacking transverse cross-

[1]The *nArray* library API uses the term `Subarray` for what we define in this paper as the *window* operation. Throughout the text of this paper, we will stick to the *window* terminology, although the code samples we include continue to refer to it as a `Subarray`.

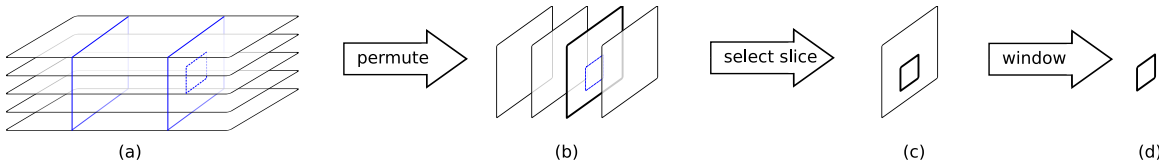


Figure 2: An illustration of *nArray*'s layout operations: permute, slice, window. (a) *Initial stack of transverse images.* Sagittal cross-sections are shown with blue lines; the region of interest is shown with blue dashes. (b) *Sagittal cross sections.* The slice of interest is shown with thick lines; the region of interest is shown with blue dashes. (c) *Slice of interest.* The region of interest is shown with thick lines. (d) *Region of interest.*

sections, we first perform a *permutation* on the volume to orient the order of access of elements correctly. Next, we perform a *slice* operation on the permuted volume to obtain a two-dimensional container that holds the desired sagittal slice. Finally, to focus on a particular region of the sagittal slice, we perform a *window* operation on the slice. This is illustrated in Figure 2 and in the code listing in Table 2.

Breaking up the overlay concept into these three distinct operations has two direct benefits: it creates easier-to-debug code, and it gives one flexibility in how one creates new layouts.

2.4 Strides

The windowed overlay and dimension permutation are achieved through a careful definition of *strides*, which indicate the increment in memory address between adjacent array elements in each stride's corresponding dimension; each stride can be either a positive or negative integer. Every *nArray* object includes a vector of stride values whose length is equal to the number of dimensions. In a simple, flat layout, the fastest changing dimension has a stride of 1, and the stride of any higher dimension is equal to the number of elements in all the lower-dimension slices. In a window overlay, however, a stride in one dimension can be larger than the number of elements in a lower dimension, which means that more than one memory cell is "skipped" in order to move from one slice to the next. When permutations are applied, the order of the strides changes arbitrarily. An example of overlay strides is shown in Figure 3.

3 Algorithms

3.1 *nArray* engines

The *nArray* engines are a set of classes and functions optimized to efficiently traverse the elements of *nArray* containers. The engines operate on both memory-allocating and overlaying containers, and the interface to call the engines on either type of container is identical, so the differences between operating on the two types of containers are transparent to the user. Also, the engines handle both contiguous and non-contiguous memory blocks.

Almost always the containers used with the engines play the part of operands of an operation (such as the addition of two arrays, with the sum stored into a third). Each engine is designed to support a specific combination of operands, such as unary, binary, or store-back operations (e.g. the `Abs`, `+` and `+=` operators), as well as various operand types (e.g. arrays, scalars, etc.).

Expression structure in the engines is abstracted and encapsulated. Any operation defined in the *nArray* API

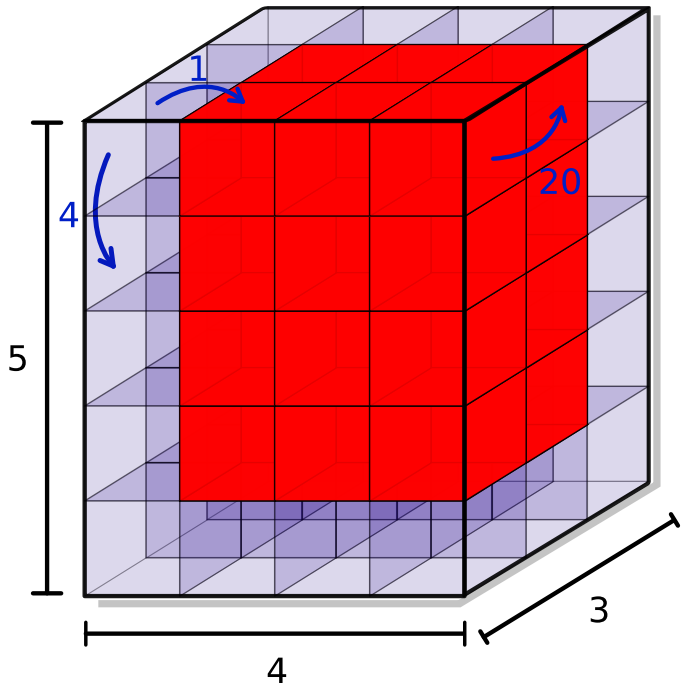


Figure 3: An example of strides in a three-dimensional array. The purple block of sizes of $3 \times 5 \times 4$ cells (the order of dimensions is Z,Y,X) is the "parent container". The red block, with $2 \times 4 \times 3$ cells, is a window overlay of the parent container. In both blocks, the memory strides between adjacent elements are (20, 4, 1) in corresponding dimension order.

```
// Create original CT volume.  The sizes are specified
// in the Z-Y-X order.  nsize_type is a "fixed-size" vector.
NArrayType::nsize_type originalSize(240, 512, 512);
NArrayType originalCTVolume(originalSize);

// load volume with CT data
/* ... */

// Create a permuted overlay of the original CT.
// Dimension 0 of the permuted array corresponds to dimension
// 2 of the original volume, that is, the X dimension, or sagittal
// cross sections.
NArrayType::nsize_type orderOfDimensions(2, 0, 1);
NArrayType::PermutationRefType sagittalOrientation;
sagittalOrientation.PermutationOf(originalCTVolume, orderOfDimensions);

// Select slice of interest in dimension 0.
NArrayType::size_type dimension = 0;
NArrayType::size_type index = 172;
NArrayType::SliceRefType sliceOfInterest;
sliceOfInterest.SliceOf(sagittalOrientation, dimension, index);

// Select region of interest.
NArrayType::SliceRefType::nsize_type regionSize(192, 192);
NArrayType::SliceRefType::nsize_type regionStart(10, 50);
NArrayType::SliceRefType::SubarrayRefType regionOfInterest;
regionOfInterest.SubarrayOf(sliceOfInterest, regionStart, regionSize);
```

Table 2: Code example of defining layout manipulation overlays.

only requires the user to select the appropriate expression structure and provide the relevant operators. For example, computing the maximum element, the sum of elements, and the sum of squares of an *nArray* all make use of the same engine because they all have the same expression structure: compute a scalar function from all the elements of one *nArray* . The three operations differ only in the operations to be performed on or between the elements. For example, in *sum of elements* and *maximum*, there is no operation on individual elements, but in *sum of squares*, the square of each element is computed; in *sum of elements* and *sum of squares*, there is also an addition operation between elements, and in *maximum* there is a *max* operation between elements. Beyond these differences, the structure of all three operations is identical, and this similarity is expressed in the engines. In practice, the operations (*add*, *max* ...) are passed to the engines as template parameters whereas the signature of the engine function is identical for all operations. In this way, the engines make the containers very straightforward to use and encourage the creation of easy-to-debug code.

The engines have several features that contribute to the overall efficiency and scalability of the package while hiding the details of the traversal algorithm from the caller function. They can be viewed as an abstraction of a multi-level nested loop.

3.2 Engine algorithm

The *nArray* engine uses a pointer, which we call the "current" pointer, to traverse an *nArray* container, much like an STL iterator, which traverses a container from beginning to end in sequential order. However, since the engine must operate on both memory-allocating and overlay *nArray* containers, the algorithm is not as simple as having the pointer run through the container's memory block from start to finish. Instead, when the pointer reaches the end of a dimension of the container, the engine must know by how many address spaces to increment the pointer in order to reach the next element in that dimension.

In designing the engine algorithm, we had to address three main issues. First, we could not use a nesting structure to loop through a container because there is an arbitrary number of dimensions in that container. We resolved this by using a vector of the same size as the number of dimensions of the container to store "target" pointers, which mark the end of each dimension. When the current pointer reaches a target pointer, the engine "wraps" the current pointer around this dimension by incrementing the pointer by the appropriate number of address spaces. The wrap-around test is performed in a recursive fashion, traversing down the list of targets until the appropriate one is found. This replaces the conventional nested loop code structure.

The second issue was how to efficiently wrap around the current pointer when it reaches a target pointer. This is resolved by having the engine precalculate the dimension offsets (the "stride to the next dimension", or STND, values) using the stride values.

Finally, the engines also have to update the target pointers accordingly when the wrap-around occurs. This is shown in the `IncrementPointers` method below. `IncrementPointers` is the equivalent of the "loop header", and it is called to move the current pointer to the next element. Notice that `IncrementPointers` returns the number of dimensions that have been exhausted and wrapped-around. We take advantage of this result to wrap-around the "current pointer" for other operand containers that may be involved. This topic, however, is not covered in this paper.

The result is an efficient and versatile engine algorithm that is capable of running on both memory-allocating and reference *nArray* containers. The source code for the algorithm is provided below. It is written in C-style notation.

```
// The PreProcess function computes the wrap-around strides (STND)
// and the target pointers before the engine loop is begun
void PreProcess(const unsigned int numDimensions,
  const unsigned int arraySizes[], const int arrayStrides[],
  const _elementPointer basePtr, int arraySTND[],
  _elementPointer arrayTargets[])
{
  unsigned int i;

  for (i = 0; i < numDimensions; ++i)
  {
    unsigned int span = arraySizes[i] * arrayStrides[i];
    /* calculate initial placement of target pointers */
    arrayTargets[i] = basePtr + span;
    /* calculate STND */
    arraySTND[i] = (i != 0) ? arrayStrides[i-1] - span : 0;
   }
}

// The function IncrementPointers checks which dimensions are
```

```
// exhausted, performs a wrap-around of the current pointer,
// and recomputes new targets if they need to be updated.  It
// returns the number of dimensions that were exhausted.
unsigned int IncrementPointers(const unsigned int numDimensions,
  _elementPointer targets[],  _elementPointer  & currentPointer,
  const int strides[],  const int stnd[])
  {
    unsigned int i = numDimensions - 1;
    unsigned int wrapCounter = 0;
    currentPointer += strides[numDimensions - 1];
    while (currentPointer == targets[i]) {  // the i-th dimension is exhausted
      currentPointer += stnd[i];
      ++wrapCounter;
      --i;
      if (wrapCounter == numDimensions) // exhausted all elements
        return wrapCounter;
    }

    // if no dimensions were exhausted, we can return immediately
    if (wrapCounter == 0)
      return wrapCounter;

    // now, update the targets forward from the current pointer
    ++i;
    do {
      targets[i] = currentPointer + (strides[i-1] - stnd[i]);
      ++i;
    } while (i < numDimensions);

    return wrapCounter;
  }
```

As an example, consider the *nArray* container in Figure 3. The red overlaying window container has size $2 \times 4 \times 3$ and stride values (20, 4, 1). The values the engine would calculate for this container are:

```
Sizes                2   4   3
Strides             20   4   1
STND                 0   4   1
Target offsets      40  16   3
```

3.3 Iterators

Iterators are a widely-used method of traversing all the elements of a container, utilizing an object to keep tabs on the location. We created the *nArray* iterators to conform with the Standard Template Library's specification for a random access iterator.

The *nArray* iterators are designed to handle complicated layouts such as dimension permutations and non-contiguous memory blocks. While a mechanism similar to the *nArray* engines could be encapsulated as an object, we believe this might be overkill for iterators. Instead, an *nArray* iterator keeps a "meta-index" internally, which indicates the sequential position of an element in the container. The iterator converts the meta-index to a "position index", which is a tuple of zero-based coordinates, similar to our intuitive notion

of a multidimensional index. The position index then is converted again to an address offset from the array's base pointer by computing its dot product with the strides of the *nArray* .

To explain this mechanism, let us first write down a recursive function, presented below, which takes as input the array of sizes (i.e., numbers of elements in each dimension) of the *nArray* , the array of strides in the respective dimensions, and a meta-index, and returns an offset from the base pointer and a small array of indices. For simplicity of presentation, the function is written in C-style notation.

```
(0) unsigned int MetaIndexToOffsetAndMultiIndex(const unsigned int numDimensions,
       const unsigned int arraySizes[], const int arrayStrides[],
       const unsigned int metaIndex, unsigned int multiIndex[])
    {
(1)    if (numDimensions == 0)
(2)        return 0;
(3)    const unsigned int d = numDimensions-1;
(4)    multiIndex[d] = metaIndex % arraySizes[d];
(5)    const unsigned int dContribution = strides[d] * multiIndex[d];
(6)    return dContribution +
           MetaIndexToOffsetAndMultiIndex(numDimensions-1,
           arraySizes, arrayStrides, metaIndex / arraySizes[d], multiIndex);
    }
```

As an example, let us follow the operation of this function on a three-dimensional flat container of size $3 \times 5 \times 4$ (see Figure 3) with corresponding strides of (20, 4, 1). We start with a meta-index `metaIndex=23`. Unknown values are written as question marks. The number preceding each line refers to the corresponding line number in the code above. For simplicity, we do not replicate the array parameters through the nesting of the recursion.

```
(0) numDimensions = 3, arraySizes = [3, 5, 4], arrayStrides = [20, 4, 1],
       metaIndex = 23, multiIndex = [?, ?, ?]
(3) d = 2
(4) multiIndex = [?, ?, 23%5] = [?, ?, 3]
(5) dContribution = arrayStrides[d] * multiIndex[d] = 1 * 3 = 3
      (0) numDimensions = 2, metaIndex = 23/5 = 4
      (3) d = 1
      (4) multiIndex = [?, 4%4, 3] = [?, 0, 3]
      (5) dContribution = arrayStrides[d] * multiIndex[d] = 4 * 0 = 0
            (0) numDimensions = 1, metaIndex = 4/4 = 1
            (3) d = 0
            (4) multiIndex = [1%3, 0, 3] = [1, 0, 3]
            (5) dContribution = arrayStrides[d] * multiIndex[d] = 20 * 1 = 20
                  (0) numDimensions = 0, metaIndex = 1 / 3 = 0
                  (2) return 0
            (6) return dContribution + 0 = 20
      (6) return dContribution + 20 = 20
(6) return dContribution + 20 = 23
```

Since the layout in this example is flat, the final offset is equal to 23, which is the original meta-index. However, applying this mechanism to a configuration with different strides still computes a correct offset, which may be different from the meta-index. The final values in `multiIndex` are (1, 0, 3), which are the zero-based "coordinates" of the element whose meta-index is 23 in an ordinary indexing mode.

It is worth noting that a memory vs. runtime tradeoff is expected when comparing the engines and the iterators, with the engines being the more runtime-efficient. In principle, either mechanism could replace the other, but in practice, the engines are better optimized for runtime than a replication of iterators is, even if a similar wrap-around mechanism were implemented for the iterators; this is because the wrap-around decision inside an engine needs only to be made once, while individual iterators must decide on the wrap-around independently. Therefore, most of the operations that need to be performed on an *nArray* should be done via the engines and not the iterators.

4 Performance Benchmarks

4.1 Performance discussion

As a general rule, more regular data layouts can be processed with faster algorithms. With regards to multidimensional arrays, a flat layout can be processed faster than another layout, such as a non-contiguous block or a layout manipulation overlay, for the same data size. This is because the overhead of keeping tabs on the pointer position is easier in the flat case.

When we compare the performance of *nArray* operations via expression engines with other software packages, we have to take care to compare features on level terms. If, for example, a certain library supports only flat layouts and has a fast algorithm for evaluating expressions on them, then it should be compared with the performance of the CISST package on flat containers, i.e., vectors. On the other hand, if a library supports certain layout manipulators, such as regions of interest, then we can compare them with the engine methods in the *nArray* package. In addition, we can compare the *nArray* engines with the expression engines for lower-dimension containers in the CISST package, namely, dynamic vectors. This comparison should provide an estimate of the bookkeeping overhead involved with traversing the *nArray* .

Considering this overhead, the efficiency of using overlay arrays can depend on the frequency of their use. Evaluating a single expression involving a layout manipulation can be faster using the overlay structure than if the manipulated data layout needs first to be copied to a second container before the expression is evaluated. However, if many expressions involving the same immutable dataset are considered, it is usually more efficient to copy the elements once into a flat container and evaluate all the expressions using the new block. An important advantage of the *nArray* overlays is that they provide flexibility to the user in choosing the preferred processing method.

Likewise, if we want to compare the performance of overlays with a library that only supports copy-and-evaluate implementations, the timing of one expression involving an overlay should be compared with the timing of copying and evaluating the expression, combined, and not just with the time of evaluating. If new memory allocation is needed before the copy, then the timing for memory allocation must be counted as well.

4.2 Benchmark specifications

We performed two kinds of benchmark runs. The first consisted of extracting a manipulated layout from a parent container and copying its elements into another container. Considering the different software architectures, this simple operation was chosen to highlight the performance of the array traversal algorithm in different configurations. Notice that only two data containers were involved: the parent and the destination, and no arithmetic operations were performed on the data elements.

We compared the CISST *nArray* package with ITK's Image class. Using both libraries, we created the following test cases: a four-dimensional array; a smaller region of interest ("window") which is also four-dimensional; a three-dimensional slice of the larger array; and an axis permutation, having the same number of elements as the parent container but in a different access order. In the *nArray* package, each layout was created as an overlay on the parent container. We called the `Assign()` method to read elements from an overlay array and write them to a memory-owning array. In ITK, each operation was represented as a "filter" object, which stores a copy of the outcome. The evaluation of the expression was triggered by calling the method `Update()` on the filter.

The second kind of benchmark involved container arithmetic, namely, adding two four-dimensional arrays into a third four-dimensional array. Here, the two input operands were "windows" or subregions of two larger parent containers. The output container was allocated independently. The purpose of this benchmark was to compare the *nArray*'s overlay approach with the more traditional copy-out approach in ITK. There are two ways to compute the sum of two *nArray*s, as can be seen in Table 1: (a) apply the method `SumOf` to a third *nArray* object (the result operand); or (b) use an overloaded `operator +`. We compared both methods to demonstrate the performance cost of using an overloaded operator. In ITK, we created an `AddImageFilter` object to compute and store the sum; its inputs were the outputs of two `RegionOfInterestImageFilter` objects, which in turn copied the contents of the regions to an internal storage.

The source files for the benchmarking programs are as follows.

Benchmark	***nArray***	**ITK**
Layout manipulations	`Subarray_nArray_Benchmark.cpp`	`Subarray_ITK_Benchmark.cpp`
Array sum	`ImageAdd_nArray_Benchmark.cpp`	`ImageAdd_ITK_Benchmark.cpp`

4.3 Performance evaluation

In the C++ programs listed above, the operations of interest were surrounded by calls to a "stopwatch" object with `Start()` and `Stop()` methods to measure the evaluation time. The stopwatch uses the high-frequency timer in either Windows or Linux. The output was rounded to a millisecond precision. Both the CISST -based and the ITK-based programs were compiled and run on the following systems.

- Windows XP workstation: Two dual-core Intel Xeon CPU, 3.06 GHz, 2.00 GB RAM. Compiler: Microsoft Visual Studio 7.1
- Linux server: Two dual-core Intel Xeon CPU 64 bit, 2.0 GHz, 6.0 GB RAM, 4 MB cache per CPU. Ubuntu Linux distribution. Compiler: gcc 4.1.2

The compilation on both systems was in "Release" mode, using compiler optimizations for speed. The accumulated times for 30 repetitions of the test are summarized in Table 3; all times are in milliseconds. The data sizes are as given in the source files listed above.

For the *slice* operation, the CISST implementation consistently performed 46% to 53% faster than ITK. In the other tests, however, it was more difficult to obtain a consistent comparison. For the *window* operation, CISST was about 41% faster on the Windows build but about 16% slower on Linux. The performance of the *permute* operation was also inconsistent, depending strongly on the specific reordering of the elements (possibly due to cache coherence or access-order optimizations); we present in the table two different permutations to demonstrate this. For example, in the tests that we performed, ITK showed a ratio of about 5.3 in the computation time for different permutation access orders on the Linux system. The CISST implementation also yielded significant time variations, making comparison between CISST and ITK difficult.

Similarly inconclusive results occurred in a few other tests of the *permute* operation, which are not shown here.

Note, on the other hand, that the CISST *nArray* arithmetic engine was consistently faster than a similar computation in ITK. If we account for the filter extraction time, CISST was 65% to 74% faster in computing the addition of two arrays; if we do not account for the extraction time, CISST was consistently 5% to 30% faster.

To obtain the time ITK took to add two arrays, we had to isolate the time the `AddImageFilter` operation took to execute from the time the entire pipeline took to execute. In ITK, the timing for `AddImageFilter` should normally include the time it takes to update the two `RegionOfInterestImageFilter` objects, which were its input sources in our tests. However, we needed to isolate these in order to consider the time it takes to add the outputs only. Therefore, we measured three different times with the `AddImageFilter` using the following computations: (a) `Update()` the two region of interest filters (extraction); (b) add the regions of interest once they were extracted; and (c) `Update()` the final sum image after the two input sets were `Modified()`, triggering an implicit `Update()` of the region of interest filters. The times (a) and (b) do not necessarily add up to the time (c), since all three times were measured as "atomic" operations. Note that the parent container and region of interest sizes for the arithmetic operation benchmark are slightly different from the ones we used for the overlay benchmarks.

The results show that for the isolated layout manipulations, it is hard to determine in advance what the computation time will be. There are dependencies on the sizes of the containers (the results from these tests are not included here) and on the order of element access. Nevertheless, CISST is not consistently outperformed by ITK. For more complex operations, such as array arithmetic, the CISST overlay structures can perform much faster than ITK's filters, even when narrowed down to the actual evaluation of the result. In addition, the overlays use memory more efficiently because they do not require storage space for the overlays.

As we also show in the last example of Table 3, the CISST package provides overloaded arithmetic operators, which to many users are more intuitive than methods or filters. The overloaded operators are generally less efficient than named methods, however, because they require the creation of a temporary object to hold the computational outcome which is assigned to the final container object after evaluation, while the named method directly stores its output to the final container. This is a general weakness of the C++ language which can be overcome using expression analysis structures (some examples are in the Blitz++ library [6]). Even so, evaluating an overloaded operator on an *nArray* container is more efficient than the full cycle of

Operation	**CISST time, Windows**	**ITK time, Windows**	**CISST time, Linux**	**ITK time, Linux**
Window	8,814	14,860	4,093	3,538
Slice	250	465	122	261
Permute (3, 1, 2, 0)	202,342	183,173	73,125	99,123
Permute (3, 0, 2, 1)	193,742	95,403	58,738	18,767
Add 4D arrays	12,203	(a) 28,674 (b) 17,501 (c) 46,262	3,757	(a) 6,875 (b) 3,950 (c) 10,848
Operator +	19,044	N/A	8,790	N/A

Table 3: Benchmark times for the CISST *nArray* and ITK operations on Windows and Linux systems. The times are in milliseconds for 30 repetitions of the operation.

evaluation in ITK.

Through benchmarking these two toolkits, we have found that the algorithms implemented in CISST *nArray* are comparable to, and in a wide variety of cases outperform, those of ITK. It is important to note, though, that this is not an exhaustive benchmark comparison of the two toolkits. Nevertheless, the overlay concept and the engine algorithms are powerful tools for improving the efficiency of managing multidimensional data sets, and their implementation may improve the performance of ITK in cases it sacrifices now.

5 Final Words

Our goal in designing the CISST *nArray* package was to provide a cross-platform software library for multidimensional arrays that is computationally efficient, easy to learn, and easy to extend. The performance of the traversal algorithms implemented in CISST is comparable to or better than another popular library used in medical image processing, ITK. The use of overlay arrays reduces the computational overhead incurred by ITK when subarray extraction is combined with other operations, such as array arithmetics.

Researchers at the ERC-CISST laboratory are already using the *nArray* library in medical image processing and statistical analysis of multidimensional data.

Future development plans of the CISST *nArray* library include: a redesign of the expression engines to optimize calculations involving containers with flat or partially-flat layouts; improving the interoperability of *nArray* and lower-dimension containers, i.e., 1-D vectors and 2-D matrices; and implementing the techniques developed for *nArray*s in those lower-dimension containers. Additionally, we are examining ways to create inter-opearbility between the CISST package and ITK.

6 Acknowledgments

This work is supported by NSF ERC Grant 9731748.

References

[1] Ibanez, Schroeder, Ng, Cates: The ITK Software Guide. Kitware, Inc. ISBN 1-930934-15-7. 1

[2] Kazanzides, P., Deguet, A., Kapoor, A., Sadowsky, O., LaMora, A., and Taylor, R.H.: Development of open source software for computer-assisted intervention systems. ISC/NAMIC/MICCAI Workshop on Open-Source Software (2005). 1.2

[3] The CISST Software Package. On the web: http://www.cisst.org/cisst. 1.2

[4] The VxL Libraries. On the web: http://vxl.sourceforge.net. 1.2

[5] Veldhuizen, T.: Using C++ template metaprograms. C++ Report 7 (1995)3643 Reprinted in C++ Gems, ed. Stanley Lippman. 1.2

[6] The Blitz++ library. On the web: http://www.oonumerics.org/blitz/. 1.2, 4.3

Diffeomorphic Demons Using ITK's Finite Difference Solver Hierarchy

Release 0.02

Tom Vercauteren[1,2], Xavier Pennec[1], Aymeric Perchant[2] and Nicholas Ayache[1]

September 10, 2007

[1]Asclepios Research Group, INRIA Sophia Antipolis, Sophia Antipolis, France
[2]Mauna Kea Technologies, Paris, France
Contact: tom.vercauteren@m4x.org

Abstract

This article provides an implementation of our non-parametric diffeomorphic image registration algorithm generalizing Thirion's demons algorithm. Within the Insight Toolkit (ITK), the demons algorithm is implemented as part of the finite difference solver framework. We show that this framework can be extended to handle diffeomorphic transformations. The source code is composed of a set of reusable ITK filters and classes. In addition to an overview of our implementation, we provide a small example program that allows the user to compare the different variants of the demons algorithm.

Contents

Forewords **2**

1 Introduction **2**

2 Overview of the Algorithm **2**
2.1 The Demons Algorithm 2
2.2 Newton Methods for Lie Groups 3
2.3 Diffeomorphic Demons 4

3 Brief Note on the Implementation **4**

4 Users' Guide **5**

5 Conclusion **6**

Forewords

This article is a companion paper to the authors MICCAI 2007 paper [15] entitled "Non-parametric diffeomorphic image registration with the demons algorithm". It is intended to share the source code of our algorithm. As such it provides only basic information about the theory and does not present an evaluation of the method. The reader is thus invited to refer to [15] for the theoretical aspects and for an evaluation of the algorithm.

1 Introduction

Since Thirion's seminal paper [13], the demons algorithm has become a popular method for the problem of intra-modality deformable image registration. The demons algorithm has successfully been used by several teams [16, 17] and an open source implementation of it is available in the Insight Toolkit [7]. The success of this method in the field of biomedical imaging can largely be explained by its efficiency. Thirion introduced *demons* that push according to local characteristics of the images in a similar way Maxwell did for solving the Gibbs paradox. The forces are inspired from the optical flow equations [2] and the method alternates between computation of the forces and their regularization by a simple Gaussian smoothing.

With the advent of computational anatomy and in the absence of a justified physical model of inter-subject variability, statistics on diffeomorphisms have become an important topic [1]. Diffeomorphic registration algorithms are at the core of this research field since they often provide the *input data*. They usually rely on the computationally heavy solution of a partial differential equation [3, 6, 8, 11, 12] or use very small optimization steps [5]. In [15], we proposed an efficient non-parametric diffeomorphic image registration algorithm based on an extension of the demons algorithm.

To the best of our knowledge, no diffeomorphic registration method has yet been integrated to the Insight Toolkit. The goal of this work is to introduce the algorithm of [15] into ITK to provide an open source implementation of an efficient diffeomorphic image registration method.

2 Overview of the Algorithm

2.1 The Demons Algorithm

It has been shown in [4] that the demons algorithm could be seen as an optimization of a global energy. The main idea is to introduce a hidden variable in the registration process: correspondences. We then consider the regularization criterion as a prior on the smoothness of the transformation s. Instead of requiring that point correspondences between image pixels (a vector field c) be exact realizations of the transformation, one allows some error at each image point.

Given a *fixed image* $F(.)$ and a *moving image* $M(.)$, we end-up with the global energy:

$$E(c,s) = \frac{1}{\sigma_i^2}\,\mathrm{Sim}\,(F, M\circ c) + \frac{1}{\sigma_x^2}\,\mathrm{dist}\,(s,c)^2 + \frac{1}{\sigma_T^2}\,\mathrm{Reg}\,(s)\,, \tag{1}$$

$$\mathrm{Sim}\,(F, M\circ s) = \frac{1}{2}\,\|F - M\circ s\|^2 = \frac{1}{2|\Omega_P|}\sum_{p\in\Omega_P} |F(p) - M(s(p))|^2\,, \tag{2}$$

where Ω_P is the region of overlap between F and $M\circ s$, σ_i accounts for the noise on the image intensity, σ_x accounts for a spatial uncertainty on the correspondences and σ_T controls the amount of regularization

we need. We classically have $\text{dist}(s,c) = \|c-s\|$ and $\text{Reg}(s) = \|\nabla s\|^2$ but the regularization can also be modified to handle fluid-like constraints [4].

Within this framework, the demons registration can be explained as an alternate optimization over s and c. It can conveniently be summarized into the algorithm below:

Algorithm 1 (Demons Algorithm)**.**

- Choose a starting spatial transformation (a vector field) s
- Iterate until convergence:
  - Given s, compute a correspondence update field u by minimizing $E_s^{\text{corr}}(u) = \|F - M \circ (s+u)\|^2 + \frac{\sigma_i^2}{\sigma_x^2}\|u\|^2$ with respect to u
  - If a fluid-like regularization is used, let $u \leftarrow K_{\text{fluid}} \star u$. The convolution kernel will typically be Gaussian
  - Let $c \leftarrow s+u$
  - If a diffusion-like regularization is used, let $s \leftarrow K_{\text{diff}} \star c$ (else let $s \leftarrow c$). The convolution kernel will also typically be Gaussian

In [14], we showed that a Newton method on $E_s^{\text{corr}}(u)$ provided us with the folowing optimization step:

$$u(p) = -\frac{F(p) - M \circ s(p)}{\|J^p\|^2 + \frac{\sigma_i^2(p)}{\sigma_x^2}} J^{pT} \tag{3}$$

where we use the local estimation $\sigma_i(p) = |F(p) - M \circ c(p)|$ of the image noise and where $J^p = -\nabla_p^T (M \circ s)$ with a Gauss-Newton method, $J^p = -\frac{1}{2}\left(\nabla_p^T F + \nabla_p^T (M \circ s)\right)$ with the efficient second-order minimization (ESM) method of [10] and $J^p = -\nabla_p^T F$ with Thirion's rule. Note that σ_x then controls the maximum step length: $\|u(p)\| \leq \sigma_x/2$.

2.2 Newton Methods for Lie Groups

The most straightforward way to adapt the demons algorithm to make it diffeomorphic is to optimize (1) over a space of diffeomorphisms. This can be done as in [9, 10] by using an intrinsic update step

$$s \leftarrow s \circ \exp(u), \tag{4}$$

on the Lie group of diffeomorphisms. This approach obviously requires an algorithm to compute the exponential for the Lie group of interest. Thanks to the scaling and squaring approach of [1], this exponential can efficiently be computed for diffeomorphisms with just a few compositions:

Algorithm 2 (Fast Computation of Vector Field Exponentials)**.**

- Choose N such that $2^{-N}u$ is close enough to 0, e.g. $\max \left\|2^{-N}u(p)\right\| \leq 0.5$
- Perform an explicit first order integration: $v(p) \leftarrow 2^{-N}u(p)$ for all pixels
- Do N (not 2^N!) recursive squarings of v: $v \leftarrow v \circ v$

2.3 Diffeomorphic Demons

By plugging the above Newton method tools for Lie groups within the alternate optimization framework of the demons, we proposed in [15] the following non-parametric diffeomorphic image registration algorithm:

Algorithm 3 (Diffeomorphic Demons Iteration)**.**

- Compute the correspondence update field u using (3)
- If a fluid-like regularization is used, let $u \leftarrow K_{\text{fluid}} \star u$.
- Let $c \leftarrow s \circ \exp(u)$, *where* $\exp(u)$ is computed using Algorithm 2
- If a diffusion-like regularization is used, let $s \leftarrow K_{\text{diff}} \star c$ (else let $s \leftarrow c$).

3 Brief Note on the Implementation

Our implementation tries to follow the style and design of the Insight Toolkit. All our filter are N-dimensional and are templated over the important types such as the pixel types. In most cases we tried to divide the algorithm into meaningfull and reusable classes.

As shown in Algorithm 3, several blocks can be distinguished. We can first see that a method is required to compute the Lie group exponential of Algorithm 2. This algorithm takes a speed vector field on input and provides on output a diffeomorphic deformation represented as a standard displacement vector field. A natural choice was thus to implemented this exponential as an ITK image filter: the `ExponentialDeformationFieldImageFilter` class. This filter can easily be reused in a different setting such as to compute statistics on diffeomorphisms.

In order to ease the creation of the `ExponentialDeformationFieldImageFilter` class, several usefull and reusable filter such as the `DivideByConstantImageFilter` are also provided.

Within the Insight Toolkit, the demons algorithm is implemented as part of the finite difference solver (FDS) framework. Our implementation of the diffeomorphic demons is also built on top of this framework by implementing a specialized version of a `PDEDeformableRegistrationFilter`: the `DiffeomorphicDemonsRegistrationFilter` class. The most important modification we did to the FDS pipeline, is to include this exponentiation step within the `ApplyUpdate` function of our specialized `PDEDeformableRegistrationFilter` class.

In addition to these main classes, our submission also includes a set of filters that are not fully part of the algorithm (e.g. `WarpJacobianDeterminantFilter`). These filters are meant to provide some statistics on the output of the algorithms. They should ease a quantitative comparison of the different variants of the demons algorithm.

Below is the list of classes, with brief descriptions, that we provide and use within our method:

- **itk::DiffeomorphicDemonsRegistrationFilter < TFixedImage, TMovingImage, TDeformationField >:** Deformably register two images using a diffeomorphic demons algorithm
- **itk::DivideByConstantImageFilter < TInputImage, TConstant, TOutputImage >:** Divide input pixels by a constant

- **itk::ESMDemonsRegistrationFunction < TFixedImage, TMovingImage, TDeformationField >:** Fast implementation of the symmetric demons registration force
- **itk::ExponentialDeformationFieldImageFilter < TInputImage, TOutputImage >:** Compute a diffeomorphic deformation field as the Lie group exponential of a vector field
- **itk::FastSymmetricForcesDemonsRegistrationFilter < TFixedImage, TMovingImage, TDeformationField >:** Deformably register two images using a symmetric forces demons algorithm
- **itk::GridForwardWarpImageFilter < TDeformationField, TOutputImage >:** Warp a grid using an input deformation field
- **itk::MultiplyByConstantImageFilter < TInputImage, TConstant, TOutputImage >:** Multiply input pixels by a constant
- **itk::MultiResolutionPDEDeformableRegistration2 < TFixedImage, TMovingImage, TDeformationField, TRealType >:** Framework for performing multi-resolution PDE deformable registration
- **itk::VectorCentralDifferenceImageFunction < TInputImage, TCoordRep >:** Calculate the derivative by central differencing
- **itk::VectorLinearInterpolateNearestNeighborExtrapolateImageFunction < TInputImage, TCoordRep >:** Linearly interpolate or NN extrapolate a vector image at specified positions
- **itk::WarpJacobianDeterminantFilter < TInputImage, TOutputImage >:** Compute a scalar image from a vector image (e.g., deformation field) input, where each output scalar at each pixel is the Jacobian determinant of the warping at that location
- **itk::WarpHarmonicEnergyCalculator < TInputImage >:** Compute the harmonic energy of a deformation field

4 Users' Guide

From a user's point of view the most important file of our submission is the example application provided in `DemonsRegistration.cxx`. The goal of this example is to provide a command-line tool to perform an intra-modality deformable registration with a chosen variant of the demons. This tool works in both 2D and 3D and can trivially be extended to other dimensions.

The user can choose the input images, the variant of the demons that should be used and the type of output that should be stored. The image IO operations use standard ITK filters meaning that all file formats suported by ITK can be used.

Below is the list of options of the command-line tool:

- **-f/–fixed-image=STRING:** Fixed image filename - mandatory argument
- **-m/–moving-image=STRING:** Moving image filename - mandatory argument
- **-b/–input-field=STRING:** Input field filename - default: empty
- **-o/–output-image=STRING:** Output image filename - default: output.mha

- **-O/–output-field(=STRING):** Output field filename, optional argument - default: OUTPUTIMAGENAME-field.mha
- **-r/–true-field=STRING:** True field filename, this is for controlled experiments only where we want to compare the results of the algorithm with a known true field - default: not used
- **-n/–num-levels=UINT:** Number of multiresolution levels - default: 3
- **-i/–num-iterations=UINTx...xUINT:** Number of demons iterations per level - default: [10 10 10]
- **-s/–def-field-sigma=FLOAT:** Smoothing sigma for the deformation field at each iteration - default: 3
- **-g/–up-field-sigma=FLOAT:** Smoothing sigma for the update field at each iteration - default: 0
- **-l/–max-step-length=FLOAT:** Maximum length of an update vector (0: no restriction) - default: 2
- **-a/–update-rule:** Type of update rule. (0: $s \leftarrow s \circ \exp(u)$ (diffeomorphic), 1: $s \leftarrow s + u$ (ITK basic), 2: $s \leftarrow s \circ (\mathrm{Id} + u)$ (Thirion))
- **-t/–gradient-type=UINT:** Type of gradient used for computing the demons force (0 is symmetrized, 1 is fixed image, 2 is warped moving image, 3 is mapped moving image) - default: 0
- **-e/–use-histogram-matching:** Use histogram matching (e.g. for different MRs)
- **-v/–verbose(=UINT):** Verbosity, if a verbose mode is used, the application will compute a set of statistics and write them to a text file - default: 0; without argument: 1
- **-h/–help:** Display an help message and exit

This command-line tool is used within a unit test triggered by CMake.

5 Conclusion

We have proposed an ITK implementation of our efficient non-parametric diffeomorphic registration algorithm. To the best of our knowledge, this is the first open-source implementation of a diffeomorphic registration tool within the Insight Toolkit. The design of our implementation tries to follow the design of ITK and thus provides templated N-dimensional filters. The code should be easily integrated to ITK and provide reusable blocks.

References

[1] Vincent Arsigny, Olivier Commowick, Xavier Pennec, and Nicholas Ayache. A Log-Euclidean framework for statistics on diffeomorphisms. In *Proceedings of the 9th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI'06)*, volume 4190 of *Lecture Notes in Computer Science*, pages 924–931. Springer-Verlag, 2006.

[2] John L. Barron, David J. Fleet, and Steven S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, February 1994.

[3] M. Faisal Beg, Michael I. Miller, Alain Trouvé, and Laurent Younes. Computing large deformation metric mappings via geodesic flows of diffeomorphisms. *International Journal of Computer Vision*, 61(2), February 2005.

[4] Pascal Cachier, Eric Bardinet, Didier Dormont, Xavier Pennec, and Nicholas Ayache. Iconic feature based nonrigid registration: The PASHA algorithm. *Computer vision and image understanding*, 89(2–3):272–298, February 2003.

[5] Christophe Chefd'hotel, Gerardo Hermosillo, and Olivier Faugeras. Flows of diffeomorphisms for multimodal image registration. In *Proceedings of the IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI'02)*, pages 753–756, 2002.

[6] Gary E. Christensen, Richard D. Rabitt, and Michael I. Miller. Deformable templates using large deformation kinematics. *IEEE Transactions on Image Processing*, 5(10), October 1996.

[7] Luis Ibáñez, Will Schroeder, Lydia Ng, and Josh Cates. *The ITK Software Guide*. Kitware, Inc., 2 edition, 2005.

[8] Sarang C. Joshi and Michael I. Miller. Landmark matching via large deformation diffeomorphisms. *IEEE Transactions on Image Processing*, 9(8):1357–1370, August 2000.

[9] Robert Mahony and Jonathan H. Manton. The geometry of the Newton method on non-compact Lie-groups. *Journal of Global Optimization*, 23(3):309–327, August 2002.

[10] Ezio Malis. Improving vision-based control using efficient second-order minimization techniques. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'04)*, April 2004.

[11] Stephen Marsland and Carole Twining. Constructing diffeomorphic representations for the group-wise analysis of non-rigid registrations of medical images. *IEEE Transactions on Medical Imaging*, 23(8):1006–1020, 2004.

[12] Michael I. Miller, Sarang C. Joshi, and Gary E. Christensen. Large deformation fluid diffeomorphisms for landmark and image matching. In Arthur Toga, editor, *Brain Warping*, pages 115–131. Elsevier, 1998.

[13] Jean-Philippe Thirion. Image matching as a diffusion process: An analogy with Maxwell's demons. *Medical Image Analysis*, 2(3):243–260, 1998.

[14] Tom Vercauteren, Xavier Pennec, Ezio Malis, Aymeric Perchant, and Nicholas Ayache. Insight into efficient image registration techniques and the demons algorithm. In *Proceedings of Information Processing in Medical Imaging (IPMI'07)*, volume 4584 of *Lecture Notes in Computer Science*, pages 495–506. Springer-Verlag, July 2007.

[15] Tom Vercauteren, Xavier Pennec, Aymeric Perchant, and Nicholas Ayache. Non-parametric diffeomorphic image registration with the demons algorithm. In *Proceedings of the 10th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI'07)*, volume 4792 of *Lecture Notes in Computer Science*, pages 319–326, Brisbane, Australia, October 2007. Springer-Verlag.

[16] He Wang, Lei Dong, Jennifer O'Daniel, Radhe Mohan, Adam S. Garden, K. Kian Ang, Deborah A. Kuban, Mark Bonnen, Joe Y. Chang, and Rex Cheung. Validation of an accelerated 'demons' algorithm for deformable image registration in radiation therapy. *Physics in Medicine and Biology*, 50(12), 2005.

[17] Jocasta A Webb, Alexandre Guimond, Neil Roberts, Paul Eldridge, David W Chadwick, Jean Meunier, and Jean-Philippe Thirion. Automatic detection of hippocampal atrophy on magnetic resonance images. *Magnetic Resonance Imaging*, 17(8):1149–1161, 1999.

An open, clinically-validated database of 3D+t cine-MR images of the left ventricle with associated manual and automated segmentations

Release 1.00

L. Najman[1], J. Cousty[1], M. Couprie[1], H. Talbot[1], S. Clément-Guinaudeau[2,3,4], T. Goissen[2,3,4] and J. Garot[2,3,4]

June 28, 2007

[1]Institut Gaspard-Monge, Laboratoire A2SI, Groupe ESIEE,
Cité Descartes, BP99, 93162 Noisy-le-Grand Cedex France
[2]Fédération de Cardiologie
[3] Unité INSERM 660, Faculté de Médecine de Créteil
[4]Mondor University Hospital, Assistance-Publique-Hôpitaux de Paris,
Créteil, France.

Abstract

In this paper, we describe a database of cine-MR (3D+t) images of the left ventricle. This database contains the voxel data, one automated and two manual segmentations for each sequence of images. The segmentations are validated from a clinical point of view. We detail how the images were obtained, as well as how the associated segmentations were performed. We also provide the data clinical validation process.

This database, including tools to compute quantitative measures and the software package used to obtain the automated segmentation, is freely available for research purposes.

The address of the site is `http://laurentnajman.org/heart`.

Contents

1 Introduction **2**

2 MR images acquisition **2**

3 Methodologies for segmentation **3**
3.1 Preprocessing 3
3.2 Automated 4D segmentation 3
3.3 Manual segmentation 4

4 Validation of the segmentations **4**
4.1 Quantitative assessment 4
Accuracy: point-to-surface measurement 4

Accuracy: False Negative/Positive Volume Fraction 5
Assessment of critical parameters 6
4.2 Qualitative assessment: manual segmentation and inter-expert variability 6

5 Conclusion **7**

Key Words. Cardiac magnetic resonance, left ventricular function, left ventricular mass, myocardial infarction, image processing.

1 Introduction

The assessment of left ventricular (LV) function is performed routinely in the clinical field. It provides important prognostic information among patients with various cardiomyopathies. Cardiac magnetic resonance (CMR) can image the heart in arbitrary direction with excellent spatial and temporal resolution. This procedure yields full anatomic coverage and dynamic assessment of the heart throughout the cardiac cycle. Thanks to good image quality and contrast, as well as complete anatomic coverage of the LV, CMR has become a gold standard for the clinical assessment of LV function [3]. However, dynamic CMR in the cine mode yields a large amount of data. As a consequence, the clinical analysis of LV function from cine-MRI requires an interactive segmentation of adjacent 2D short-axis locations, as frequent manual corrections of myocardial contours are required. This is especially true in patients with segmental wall motion abnormality or deformed LV. Numerous authors (see for instance [9, 14, 10, 6, 8, 13, 7, 11]) have contributed in the development of accurate methods that aim to allow fully automated segmentation of the whole 3D cine-MRI dataset over time (i.e., 3D+t or 4D) for the assessment of LV function, volumes and mass.

One of the main problems in the assesment of those methods is the obtention of a database of cine-MRI and associated ground-truth segmentations that are also validated from a clinical point of view. Until now, to our best knowledge, no such base was available. This not only prevents the clinical validation of some of the pre-cited methods, but also precludes a fair comparison between the existing validated methods.

The goal of this paper is to propose a database freely available on the internet for research purposes. It contains cine-MR images of the LV, together with three associated segmentations: two hand-made segmentations – each one of them performed by an independent and blinded expert cardiologist – and one 4D automated segmentation. We also provide scripts and programs that compute each of the quantitative measures described in this paper, and the software package used to obtain the automated segmentation. The web-address of this database is `http://laurentnajman.org/heart`.

The outline of the paper is the following: we first describe the acquisition of the images. We then detail the methodologies for the three segmentations, concentrating on the manual ones. Finally we discuss the validity of the different segmentations, both quantitatively and qualitatively, by comparing them together, each one of them against the others.

2 MR images acquisition

Screened population patients referred to our Institutions for recent acute myocardial infarction (AMI) were prospectively screened regardless of the treatment received at the acute phase. To be included, patients had to

exhibit symptoms of AMI – i.e., chest pain, ST segment elevation of more than 1 mm in at least 2 contiguous leads of the ECG, and greater than double the normal elevation of creatine-kinase MB subfraction. In addition, the diagnosis of AMI was required to be confirmed by invasive coronary angiography with a clearly documented culprit epicardial coronary artery. If no contraindication to CMR was found, patients were scheduled to have CMR examination between day 2 and day 4 after AMI. The study protocol was approved by the Joint Committee of the Henri Mondor Medical Institutions and informed written consent was obtained from all patients.

In fine, 18 out of 25 patients from routine clinical practice were screened according to these guidelines. These patients had experienced a first AMI and had agreed to undergo subsequent CMR examination.

The patients were examined on a 1,5 T MR scanner (Magnetom Symphony®, Siemens, Erlangen, Germany) using 6-channel anterior and posterior phased array surface coil technology. Following a 3D fast gradient-echo localizer sequence, the long axis of the heart was located and dynamic cine-MR images of the heart were acquired in 2-chamber, 4-chamber, and 3-chamber views. From these, the short axis of the heart was located perpendicularly to the long axis of the LV. Contiguous short-axis slices of the LV were acquired from base to the apex encompassing the entire LV, through the use of repeated breath-held ECG-gated steady-state free precession sequence (SSFP) with typical imaging parameters as follow: 300-360 mm field of view, 2.1 ms TR, 1.6 ms TE, 60° flip angle, 6 mm slice thickness, no gap, image matrix 256x160, 30-40 ms temporal resolution.

The number of LV short-axis locations required to cover the entire LV by cine-CMR ranged from 9 to 14. The number of frames acquired during the entire cardiac cycle ranged from 22 to 37, depending on heart rate (49-91 bpm). The most basal slice included in the analysis was located just above the mitral valve within the LV cavity. To be included, the basal myocardium had to be visible in the entire circumference at end-systole. The most apical slice was chosen as the one with the smallest visible LV cavity at end-systole. Since the sequences are ECG-gated, the end-diastolic frame corresponds to the first image of the sequence.

3 Methodologies for segmentation

3.1 Preprocessing

For each patient, the cine MRI dataset consisted of a succession of contiguous (no gap) LV short-axis 2D planes that were successively imaged over time (2D+t). The sequences were registered to the heart-cycle, and could therefore be stacked in order to construct 3D sequences. Taken together, these different planes from base to apex were considered a 3D representation of the LV. The succession of these, over time, is a 3D+t representation of the LV. Before applying any segmentation procedure, the images were over-sampled in order to provide isotropic voxels. For each sequence of 3D+t images, a single mouse click on the center of the LV cavity at end-systolic time was recorded, and the images were cropped centered on the corresponding location. Typically, the size of each volume of the sequence represents $100 \times 100 \times 40$ voxels. When a misalignment of the different sections of a same volume was observed, a translation-only registration procedure was applied.

3.2 Automated 4D segmentation

The method for the 4D segmentation of the LV was recently developed and is described elsewhere [4] (a journal paper is in preparation). It automatically detects both the endocardial and epicardial borders of the LV for the whole 4D dataset, based on prior knowledge of the shape and appearance of the LV. The 4D

object isolated between the 2 borders is labeled as LV myocardium (LVM) and the 4D object isolated inside the endocardial border is labeled as LV cavity (LVC).

The computation time to automatically segment a whole 4D sequence ranged from 2 to 5 minutes on a conventional personal computer.

3.3 Manual segmentation

A conventional 2D manual segmentation of the LV myocardium was performed by two independent and blinded expert cardiologists. They used a software package that is well established for the post-processing of medical images (Analyze®, Biomedical Imaging Resource, Mayo Clinic Foundation, Rochester, MN). These two experts are called e_1 and e_2 in the sequel. The cine-MR dataset was analyzed as a succession of 2D LV short-axis planes. For each slice location, the experts manually overlaid the endocardial and epicardial contours both at end-diastolic and end-systolic times. During manual tracing, papillary muscles and LV trabeculae were included within the LV myocardium. Then, the segmented slices were stacked to rebuilt a 3D object for quantification.

The time spent by the experts to manually segment one volume of a 3D+t sequence ranged from 15 to 20 minutes. For this reason, a manual segmentation is not available at every time-step.

4 Validation of the segmentations

In this section, we discuss the quality of the segmentations of the two experts and of the automated method, both from a quantitative and a qualitative point of view.

4.1 Quantitative assessment

In order to characterize the accuracy of all methods, we used two different kinds of measures. The first measure is relative to the mean distance between the surfaces extracted from the automated segmentations and from the manual segmentations. The second characterises the false positive and false negative volume of the segmentations. Then, we assess the ability of the automated method to produce reliable characteristics of the LV function via the computation of the ejection fraction and of the myocardium mass.

Accuracy: point-to-surface measurement

Given two surfaces ∂X and ∂Y represented by two sets of polygons, the *point-to-surface measurement* (P2S) between ∂X and ∂Y estimates the mean distance between the vertices of ∂X and ∂Y (see [1]). A symmetrical measure is obtained by taking the maximum from the P2S between ∂X and ∂Y and the P2S between ∂Y and ∂X.

On our dataset, the endocardial and the epicardial borders were extracted from the segmentations by a marching cube algorithm[5]. The P2S was computed from the segmentations obtained by the automated method and the two experts. In order to evaluate the inter-observer variability the P2S between the two experts is also provided. Table 1 presents the mean and standard deviation of these measures at end-diastolic time and end-systolic time. We note that, in all cases in Table 1, the P2S is less than 1 voxel. The automated method achieved a mean P2S of 1.51mm $\pm$ 0.38 for the endocardial border and a mean P2S of 1.81mm $\pm$ 0.43

Table 1: Details of the point to surface measurements from the results of the various segmentation methods (mean point to surface measurements expressed in mm $\pm$ standard deviation).

	soft. vs. e_1	soft. vs. e_2	e_1 vs. e_2
End-diastolic time			
Endocardial border	1.52 ± 0.35	1.67 ± 0.43	1.37 ± 0.47
Epicardial border	2.04 ± 0.35	1.68 ± 0.39	1.23 ± 0.41
End-systolic time			
Endocardial border	1.50 ± 0.41	1.35 ± 0.34	1.15 ± 0.41
Epicardial border	1.90 ± 0.56	1.61 ± 0.41	1.31 ± 0.83

for the epicardial border. These results compare favourably with those obtained by other groups on their own datasets. Furthermore, the P2S between automated and manual segmentations is in the same range as the inter-observer P2S. This is a strong indication that the automated method produces as satisfying a segmentation as either manual one.

Although the accuracy of the methods is assessed by the P2S, these measures do not precisely describe the quality of the produced segmentations. In particular, the relative importance of the misclassified objects – a parameter which becomes crucial while quantifying the volume of the different objects – is not handled by the point-to-surface measurements.

Accuracy: False Negative/Positive Volume Fraction

In order to better characterise the accuracy of the segmentation methods, we also used the two following measures preconized by J. Udupa *et al.* in [12]. Let Y be a subset of the image voxels considered as the reference segmentation and let X be the segmentation which is to be evaluated. We set

$$FNVF(X,Y) = \frac{|Y \setminus X|}{|Y|} \quad \text{and} \quad FPVF(X,Y) = \frac{|X \setminus Y|}{|Y|}.$$

These measures are expressed as a fraction of the volume of "true" delineation. The $FNVF$ (False Negative Volume Fraction) indicates the fraction of tissue that was missed and $FPVF$ (False Positive Volume Fraction) denotes the amount of tissue falsely identified as a fraction of the total amount in the "true" delineation.

For each of the 18 patients, we computed $FNVF$ and $FPVF$ for e_2 against e_1, automated method against e_1, e_1 against e_2, and automated method against e_2.

These measures were computed for three objects: LVC, LVM and for $LVCM$ – the union of LCM and LVM. Table 2 presents the mean and standard deviation of these measures at end-diastolic and end-systolic times for the 18 datasets. We remark that error rates between the two experts and between experts and software are in the same range. In the task of segmenting LVC, $LVCM$ and LVM, the inter-expert errors are comparable with the errors between software and expert segmentations. However, we observe a tendency of the automated method to underestimate LVC and $LVCM$ with respect to the experts. Indeed, the false negatives are 1.5 to 4.5 times greater than the false positives. From a qualitative study described later, the two experts came to the conclusion that the automated contours were generally better localized than the manual ones. The apparent under-estimations of LVC and $LVCM$ can, thus, be seen as a side effect of the manual segmentation process. In Section 4.2, we explain some of the bias due to the manual segmentation process.

Table 2: Mean and standard deviation of FNVP, FPVF for all segmentations of *LVC*, *LVCM* and *LVM* at end-diastolic and end-systolic-time [see text].

End-diastolic time

		e_2 vs e_1	soft. vs e_1
LVC	FNVF	0.06 ± 0.03	0.13 ± 0.05
	FPVF	0.07 ± 0.08	0.03 ± 0.02
LVCM	FNVF	0.06 ± 0.01	0.09 ± 0.02
	FPVF	0.03 ± 0.06	0.03 ± 0.02
LVM	FNVF	0.20 ± 0.04	0.22 ± 0.04
	FPVF	0.10 ± 0.03	0.20 ± 0.06

		e_1 vs e_2	soft. vs e_2
LVC	FNVF	0.07 ± 0.05	0.14 ± 0.06
	FPVF	0.07 ± 0.02	0.03 ± 0.01
LVCM	FNVF	0.02 ± 0.05	0.06 ± 0.03
	FPVF	0.07 ± 0.01	0.04 ± 0.02
LVM	FNVF	0.11 ± 0.03	0.15 ± 0.04
	FPVF	0.22 ± 0.06	0.25 ± 0.10

End-systolic time

		e_2 vs e_1	soft. vs e_1
LVC	FNVF	0.10 ± 0.04	0.16 ± 0.06
	FPVF	0.06 ± 0.03	0.05 ± 0.03
LVCM	FNVF	0.07 ± 0.01	0.07 ± 0.03
	FPVF	0.02 ± 0.02	0.05 ± 0.02
LVM	FNVF	0.14 ± 0.04	0.14 ± 0.04
	FPVF	0.08 ± 0.03	0.16 ± 0.07

		e_1 vs e_2	soft. vs e_2
LVC	FNVF	0.06 ± 0.03	0.13 ± 0.05
	FPVF	0.11 ± 0.05	0.06 ± 0.03
LVCM	FNVF	0.02 ± 0.02	0.04 ± 0.02
	FPVF	0.07 ± 0.02	0.06 ± 0.02
LVM	FNVF	0.09 ± 0.03	0.09 ± 0.02
	FPVF	0.15 ± 0.05	0.18 ± 0.06

Assessment of critical parameters

Left ventricular ejection fraction (EF) and left ventricular myocardium mass (MM) are critical parameters for cardiac diagnosis and remodeling prevention. Their estimation is routinely used by cardiologists. The EF is the amount of blood ejected during a heart cycle expressed as a fraction of the tele-diastolic volume. In our dataset the EF (resp. MM) range was 20-75% (resp. 94-197 g).

From the segmented images, the EF can be computed by $(|LVC_{\max}| - |LVC_{\min}|)/|LVC_{\max}|$, where $|LVC_{\max}|$ (resp. $|LVC_{\min}|$) is the maximal (resp. minimal) volume of the left ventricular chamber along the heart cycle. Let X_p^o denote the measure of the parameter X performed by operator o for patient p, where $X \in \{EF, MM\}$, $o \in \{e_1, e_2, s\}$, and $p \in [1 \ldots 18]$. We take $refX_p = (X_p^{e_1} + X_p^{e_2})/2$ as a reference value for the parameter X on patient p and we evaluate the deviation $\Delta X_p^o = |X_p^o - refX_p|/refX_p$. Notice that $\Delta X_p^{e_1} = \Delta X_p^{e_2}$. Over all 18 patients, the automated method achieved a mean deviation on the EF (resp. MM) of 0.032 (resp. 0.050) whereas the experts only achieved 0.055 (resp. 0.052). Furthermore, we observe that ΔEF_p^s (resp. ΔMM_p^s) is less than $\Delta EF_p^{e_1}$ (resp. $\Delta MM_p^{e_1}$) in 8 (resp. 9) of the 18 patients. In other words, the deviation on the EF (resp. MM) achieved by the automated method is less than the deviation achieved by the experts in 8 (resp. 9) of the 18 patients. We conclude that the automated tool produces reliable assessment of left ventricular functional parameters comparable to the experts'. A statistical analysis of these data – including linear regression and Bland-Altman plots – can be found on the web site http://laurentnajman.org/heart.

4.2 Qualitative assessment: manual segmentation and inter-expert variability

We now analyze some features of all segmentation methods which cannot be exhibited solely from a quantitative assessment. In Fig. 1, we present a view of the segmentations obtained by the two experts and by the automated method for a sample 3D image. We observe that the *LVM* segmented by e_1 is significantly thinner than the one segmented by e_2. We also remark that in location A, expert e_1 did not recognize a part

of the papillar muscle which was however segmented by e_2 and by the automated method. In location B, the myocardium segmented by e_1 presents a hole, which is not compatible with anatomy. We note that the automated method avoids such situations by construction and we observe that e_2's segmentation has no hole. This topological consideration has however no ill-effect on any of the quantitative measures.

Several explanations can be given. On the one hand, expert segmentations were realized exclusively on 2D images, which introduces a bias. Neither the spatial coherency between successive 2D sections of a 3D image, nor temporal coherency with the previous and next images of the sequence were taken into account. On the other hand, the precise delineation of a contour, pixel by pixel, is a very demanding task for human operators. It is well known that, in general, human operators can outperform computerized procedure in recognition tasks, whereas algorithms can often perform better than humans at delineation. Finally, we point out that there is no standardized procedure for manual segmentation of *LVM* in cardiac MR-images, contrary to what can be the case for other segmentation tasks in other modalities [2]. Therefore, the physicians who made this evaluation all believe that the automated method generally outperforms manual segmentation.

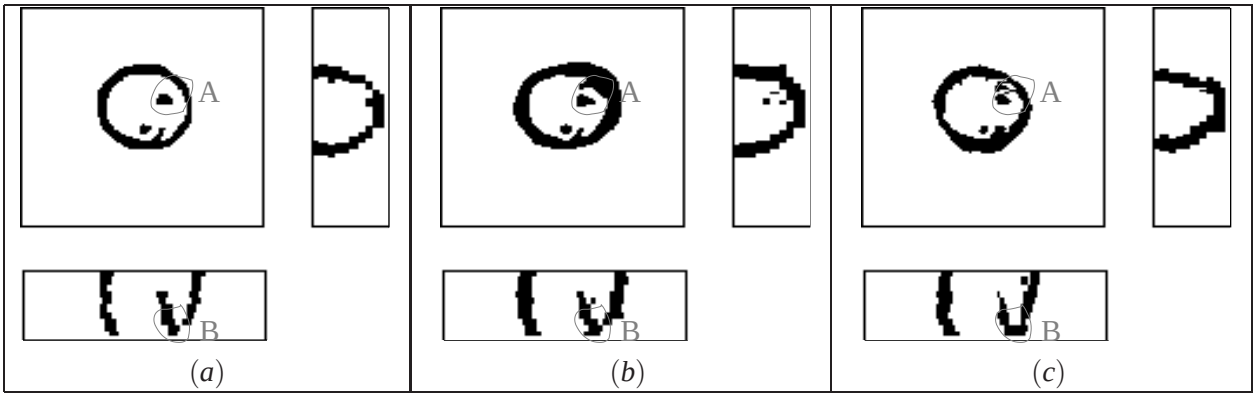


Figure 1: Examples of segmentations performed by e_1 (a), e_2 (b) and the automated method (c).

5 Conclusion

We describe in this paper the acquisition and analysis of a database of cine-MRI of the LV, with associated clinically-validated segmentations. The number of patients in the database may superficially appear to be small but, in actual fact, the number of contours processed by the tested techniques corresponds to 4752 endocardial and 4752 epicardial borders drawn from 216 LV short-axis slices, all performed by specialist cardiologists. The measured values obtained with the automated 4D analysis were found to be comparable with those obtain by the manual 2D+t methods. In this patient population, there was no other standard for comparison with true measurements of LV mass or volumes. The 4D analysis shows no significant difference in the clinical practice with values obtained from the interactive analysis of successive 2D locations.

This database is thus validated from a clinical point of view. Freely available for research purposes on http://laurentnajman.org/heart, it is a first step towards a fair evaluation and comparison of LV-segmentation methods. In the future, we plan to enrich this database with subsequent images and other modalities, as soon as these data are validated from a clinical point of view. We also plan to include anatomo-pathological animal data (using rabbits and pigs).

The authors would like to emphasize that such a work is only possible if cardiologists and computer scientists

are working together in close partnership.

References

[1] N. Aspert, D. Santa-Cruz, and T. Ebrahimi. MESH: measuring errors between surfaces using the Hausdorff distance. In *IEEE International Conference in Multimedia and Expo*, volume 1, pages 705–708, 2002. 4.1

[2] Leonardo Bonilha, Eliane Kobayashi, Fernando Cendes, and Li Min-Li. Protocol for volumetric segmentation of temporal structures using high-resolution 3-d magnetic resonance imaging. *Human Brain Mapping*, 2004. 4.2

[3] P. T. Buser, W. Auffermann, W. W. Holt, S. Wagner, B. Kircher, C. Wolfe, and C. B. Higgins. Noninvasive evaluation of global left ventricular function with use of cine nuclear magnetic resonance. *Journal of the American College of Cardiology*, 2(13):1294–1300, 1989. 1

[4] J. Cousty, L. Najman, M. Couprie, S. Clément-Guinaudeau, T. Goissen, and J. Garot. Automated, accurate and fast segmentation of 4d cardiac mr images. In F.B. Sachse and G. Seemann, editors, *FIMH: Procs. of Functional Imaging an Modeling of the Heart*, volume 4466 of *LLNCS*, pages 474–483, 2007. 3.2

[5] X. Daragon, M. Couprie, and G. Bertrand. Discrete frontiers. In *Discrete geometry for computer imagery*, volume 2886 of *Lecture Notes in Computer Science*, pages 236–245. Springer, 2003. 4.1

[6] Michael R. Kaus, Jens von Berg, Jürgen Weese, Wiro Niessen, and Vladimir Pekar. Automated segmentation of the left ventricle in cardiac MRI. *Medical Image Analysis*, 8:245–254, 2004. 1

[7] M. Lorenzo-Valdés, G. I. Sanchez-Ortiz, A. G. Elington, R. H. Mohiaddin, and D. Rueckert. Segmentation of 4D cardiac MR images using a probabilistic atlas and the EM algorithm. *Medical Image Analysis*, 10:286–303, 2006. 1

[8] J. Lötjönen, S. Kivistö, J. Koikkalainen, D. Smutek, and K. Lauerma. Statistical shape model of atria, ventricles and epicardium from short- and long-axis MR images. *Medical Image Analysis*, 8:371–386, 2004. 1

[9] Tim McInerney and Demetri Terzopoulos. A dynamic finite element surface model for segmentation and tracking in multidimensional medical images with application to cardiac 4D image analysis. *Computerized Medical Imaging and Graphics*, 19(1):69–83, 1995. 1

[10] Steven C. Mitchell, Boudewijn P. F. Lelieveldt, Rob J. van der Geest, Hans G. Bosch, Johan H. C. Reiber, and Milan Sonka. Multistage hybrid active appearance model matching: Segmentation of left and right ventricles in cardiac MR images. *IEEE Trans. on Medical Imaging*, 20:415–423, 1997. 1

[11] Johan Montagnat and Hervé Delingette. 4D deformable models with temporal constraints: application to 4D cardiac image segmentation. *Medical Image Analysis*, 9:87–100, 2005. 1

[12] J.K. Udupa, V.R. LeBlanc, Y. Zhuge, C. Imielinska, H. Schmidt, L.M. Currie, B.E. Hirsch, and J. Woodbrun. A framework for evaluating image segmentation algorithms. *Computerized Medical Imaging and Graphics*, 30:75–87, 2006. 4.1

[13] H. van Assen, M. G. Danilouchkine, A. F. Frangi, S. Ordàs, J. J. M. Westenberg, J. H. C. Reiber, and B. P. F. Lelieveldt. SPASM: A 3D-ASM for segmentation of sparse and arbitrarily oriented cardiac MRI data. *Medical Image Analysis*, 10:286–303, 2006. 1

[14] Rob J. van der Geest, Vincent G. M. Buller, Eric Jansen, Hildo J. Lamb, Leo H. B. Baur, Ernst E. van der Wall, Albert de Roos, and Johan H. C. Reiber. Comparison between manual and semiautomated analysis of left ventricular parameters from short-axis MR images. *Journal of Computer Assisted Tomography*, 21:756–765, 1997. 1

vtkINRIA3D: A VTK Extension for Spatiotemporal Data Synchronization, Visualization and Management

Release 1.00

Nicolas Toussaint[1], Maxime Sermesant[1] and Pierre Fillard[1]

July 6, 2007

[1]INRIA Sophia Antipolis, Asclepios Project-Team, France

Abstract

This paper presents an extension of the Visualization ToolKit dedicated to spatiotemporal data synchronization, visualization and management. It basically consists in a versatile library providing functionalities to help developers setting up sophisticated applications with minimal development effort. In the medical imaging context, various types of data are often encountered, which raises the need for adapted visualization and synchronization techniques. Moreover, the management of these data (organization, creation, deletion, access) can become a burden. We propose in vtkINRIA3D a strategy to synchronize interactions between datasets representations, to manipulate complex objects (e.g., neural fibers as obtained in DT-MRI), as well as a managing framework for organizing data (including temporal sequences). The efficiency of vtkINRIA3D is illustrated with two applications: MedINRIA (general medical image processing software) and CardioViz3D (cardiac image visualization). vtkINRIA3D is open-source, and comes with a set of examples, test data and softwares built upon it: http://www-sop.inria.fr/asclepios/software/vtkINRIA3D.

Contents

1 Introduction **2**

2 The vtkRenderingAddOn Library **3**

3 The vtkVisuManagement Library **5**

4 The vtkDataManagement Library **7**
4.1 The vtkDataManager and the vtkMetaDataSet Classes 8
4.2 Time Support 10

5 Applications **11**
5.1 MedINRIA 11
5.2 CardioViz3D 11

6 Conclusions and Future Work **12**

A Examples **13**
A.1 SynchronizedViews 13
A.2 The Data Manager 14

1 Introduction

The Visualization ToolKit (VTK) [2] offers an impressive set of comprehensive C++ classes for data representation and manipulation, and has become a standard in scientific visualization. Not only VTK provides state-of-the-art techniques for processing datasets and displaying meaningful information from them, but it also eases a lot the developer's pain by the object-oriented programming and clarity of the coding.

In medical image processing and visualization, one often has to deal with very specific types of data that require specialized visualization and interaction techniques. Moreover, the management of all these data by programmers who want to build a complete processing and visualization system can be a challenging task, due to the various forms they can take, and to the specific visualization strategies they require. In order to provide a more straightforward approach, we thought of three important features:

- Synchronization of interactions and visualization among windows. For instance, when one clicks on a window to position an axis in a slice of a 3D volume, one would like to have the other windows displaying this data (if any) to automatically set their axis at the exact same position. This feature is very desirable when comparing images or when looking at a 3D volume with different orientations. Another example is when one adjusts the contrast of a view: one would like the other views to adjust their contrast similarly.
- Adapted manipulation of complex data coming from the increasing diversification of the source of medical information. For instance, diffusion tensor MRI (DT-MRI or DTI) [4] is a very attractive modality since it allows to reconstruct white matter fibers from several MR measurements. Up to several thousands of fibers can be reconstructed, and obviously it requires adapted visualization and interaction techniques to give medical experts the chance to extract a specific fiber bundle of interest.
- Simple and efficient management of these data for programmers. In particular, we thought interesting to have a single object that reads, writes, allocates and deletes any sort of dataset (images, meshes, neural fibers, etc.), so that a complete visualization system of any type of data could be easily plugged in a homemade program. Moreover, support of temporal sequences of these data is extremely desirable, especially in cardiac research.

The `vtkINRIA3D` library is a concrete implementation of these features. There are numerous visualization projects built around VTK and ITK (Slicer, MITK,...) but the aim of `vtkINRIA3D` is to make available a simple and versatile library providing the described features and allowing developers to build their own software upon it. This is why it is basically an extension of VTK, i.e. a collection of new classes based on the VTK architecture. `vtkINRIA3D` is divided into three libraries following the three points presented above:

- The `vtkRenderingAddOn` library (Sec. 2) implements a strategy for synchronization of visualization and interactions among windows.

- The vtkVisuManagement library (Sec. 3) provides a set of classes that manages the visualization, interaction, ROI extraction of certain type of data, like tensor fields, VTK polylines, and isosurfaces.
- The vtkDataManagement library (Sec. 4) offers a framework to handle heterogeneous vtkDataSet objects by federating them into a single class named vtkDataManager and supporting time sequences of these objects.

In addition, we briefly present in Sec. 5 two softwares based on vtkINRIA3D: *MedINRIA*, which is a collection of graphic tools targetting the clinicians, and *CardioViz3D*, a platform for the processing and visualization of cardiac imaging.

Requirements: vtkINRIA3D is compiled with VTK 5.0.3, ITK 3.2.0 (for some optional components), CMake 2.4.6, and is open-source. Source code, doxygen files, dashboard, and examples data can be found at: http://www-sop.inria.fr/asclepios/software/vtkINRIA3D.

2 The vtkRenderingAddOn Library

The main purpose of the vtkRenderingAddOn library is to provide a framework to synchronize user interactions on a vtkRenderWindow. To do so, a cycled-tree structure is used. The base class, vtkView, is fed with a vtkRenderer, vtkRenderWindow and a vtkRenderWindowInteractor to display and interact with vtkActors. It has also a unique Parent (of the same type) and a set of children (of the same type as well). Then, when a synchronized method is called, the called vtkView transmits it to its children and so on. As this is a cycled structure, one should be careful that the first calling object is not called again, which would result in an infinite loop. To prevent this undesirable behavior, we implemented a Lock() and UnLock() methods that each vtkView must call before and after calling its children's method.

The class vtkView implements this strategy for the synchronization of user interactions. However, it does not provide any concrete feature, as this base class should remain as generic as possible. A concrete implementation is given by classes vtkViewImage2D and vtkViewImage3D which derive from the base class vtkViewImage (which itself derives from vtkView). The class vtkViewImage is the interface class for the functions shared by any view that displays an image. The specificity of vtkViewImage2D is to display an image slice by slice, while vtkViewImage3D displays an image in 3D using either multi-planar reconstruction (MPR) or volume rendering (VR) techniques. Moreover, in VR, a vtkBoxWidget allows the user to remove a volume of interest from the displayed image. This gives for instance a rapid insight into a patient's brain and is a very desirable feature. The methods that can be synchronized are:

- Adjust Window/Level: SyncSetWindow(), SyncSetLevel();
- Set the position: SyncSetPosition();
- Set a lookup table (SyncSetLookupTable()), a mask image (or ROI) (SyncSetMaskImage()), a vtkDataset (SyncAddDataSet()), or a vtkPolyData (SyncAddPolyData());

Notice that all synchronized functions start by Sync. Naturally, the "desynchronized" version of the same function has the same name without Sync, like SyncSetWindow() and SetWindow(). We illustrate the synchronization mechanism with the method SyncSetColorWindow(double w):

```
void vtkViewImage::SyncSetColorWindow (double w)
{
```

```
  if( !this->IsLocked() )
  {
    this->SetWindow (w); // actually change the window

    // The current view is now locked to prevent it to be called again and again...
    this->Lock();
    for( unsigned int i=0; i<this->Children.size(); i++)
    {
      vtkViewImage* view = dynamic_cast<vtkViewImage*> (this->Children[i]);
      if( view )
      {
        view->SetColorWindow (w);
        view->Render();
      }
    }
    this->UnLock();
 }
}
```

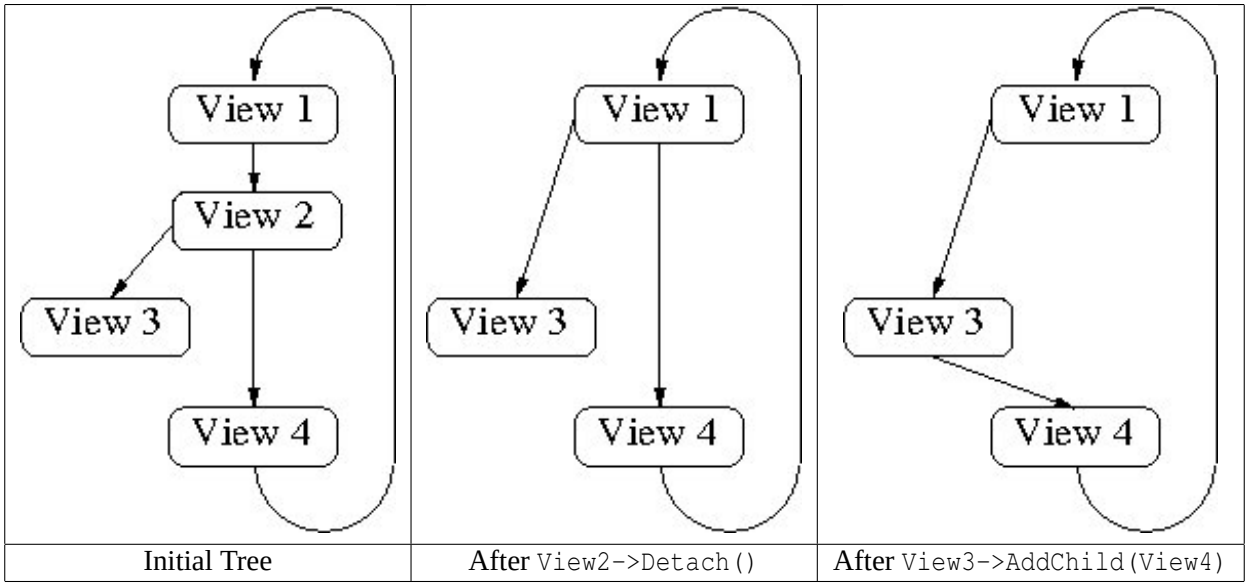
Obviously, the effect of the synchronized methods will be different depending on the implementation made in each class. For example, calling `SyncSetPosition()` on a `vtkViewImage2D` object will place 2D cursors on the exact required position, while a `vtkViewImage3D` object will intersect the 3 orthogonal planes at this position when MPR is selected, or position a 3D cursor at the exact same coordinate when VR is activated. Note that the two last methods (`SyncAddDataSet()` and `SyncAddPolyData()`) are particularly interesting since they allow to project any `vtkDataSet` (and derived classes) onto the 2D views: a slice of the dataset is extracted using either a `vtkCutter` (`SyncAddDataSet()`) or a `vtkClipDataSet` (`SyncAddPolyData()`). The former cuts the dataset with the current image slice, i.e., it returns the intersection of the dataset with a plane defined by the current slice displayed. As a consequence, it turns a N-Dimensional dataset into a N-1-Dimensional dataset (e.g., lines become points). The later extracts the polygonal data contained between two planes defined by the current image slice plus a user-provided thickness (generally the spacing between two consecutive slices). This has the advantage to preserve the dimensionality of the data (lines are still lines).

Extra classes are provided for further interactions: the class `vtkViewImage2DWithTracer` allows to manually trace on a 2D slice (using the VTK class `vtkImageTracerWidget`) and to transform the tracing into a binary image itself overlapped onto the views (and synchronized among them). The class `vtkViewImage2DWithOrientedPoint` gives the possibility to place a point and a direction on a slice to perform for instance a manual rigid transformation of this image.

When the user wants to remove a view from the tree, he should call the method `Detach()`: it alerts the `Parent`'s view that it is detaching and connects its own `Children` to its `Parent`'s ones. This prevents any loss in the synchronization mechanism (Fig. 1 left and middle). Moreover, when a view is "re-parented" (case of `View4` on Fig. 1 right), its `Parent` is automatically alerted and the detaching object is automatically removed from the `Children` list of its `Parent`. This insures that a view is the child of only one view, thus preventing to be called several times by several `Parents`.

An example of how to use these synchronized views is given in the folder `Example/SynchronizedViews` and described in Appendix A.1.

We included in `vtkINRIA3D` the source of a software, called *ImageViewer*, that uses the synchronized views described above (see Fig. 2). Moreover, it offers some interesting features that are not present in many medical image visualization softwares:



Initial Tree | After `View2->Detach()` | After `View3->AddChild(View4)`

Figure 1: **Left:** Initial tree. Four views are linked: `View1` has child `View2`, `View2` has children `View3` and `View4`, and `View4` is linked back to `View1`. `View3` has no child, meaning that it can only receive function calls but not transmit any. **Middle:** When one wants to remove a view, one should call the method `Detach()`: `View2->Detach()` in this example. This results in the removal of `View2`, but `View2`'s Parent is automatically re-linked to its children, avoiding a loss in the synchronization of the remaining views. **Right:** We now want to link `View3` with `View4`. In that case, `View4`'s parent is informed that it lost a child, thus preventing a single view to be the child of several others (then preventing to receive multiple function calls from their parents).

- Full DICOM support using ITK I/O factory mechanism [6];
- Tab-browsing of images;
- A preview screen (Fig. 2) to compare images: interactions can be synchronized, which allows to interact with many images by acting on only one of them;

The user-interface of *ImageViewer* is made in `wxWidgets` [3] in order to be integrated in *MedINRIA* (more details can be found in Sec. 5.1).
The next section depicts a library specialized in visualization of various data.

3 The vtkVisuManagement Library

The flavor of this library is to facilitate the task of programmers who want to quickly integrate nice visualization and interactions of scientific data into their software. This library is composed of a set of classes, each of them handling the rendering and interactions of specific data:

- `vtkIsosurfaceManager` takes as input a `vtkImageData` and generates one isosurface per label (the number and value of labels is automatically determined). This class uses a `vtkContourFilter` to generate the contours, a `vtkDecimatePro` filter to decimate the polygons up to 90% of the original number, and a `vtkSmoothPolyDataFilter` to smooth the final mesh. A `vtkLookupTable` controls the color and opacity of each isosurface. An example can be found in `Example/IsosurfacesManager/IsosurfacesManager.cxx`: It renders a segmentation of the BrainWeb in a few lines of code (Fig. 3 left):

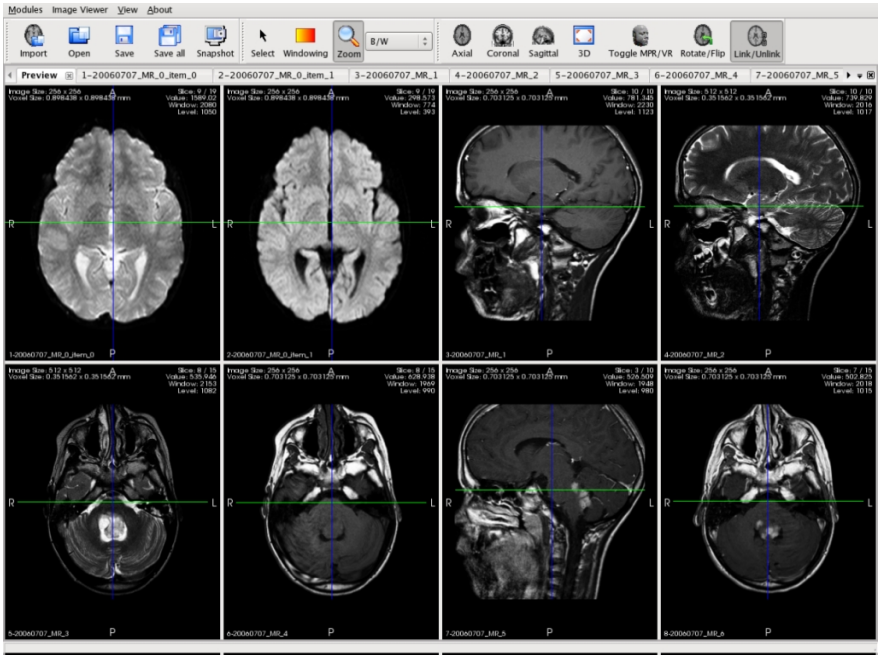


Figure 2: Snapshot of the *ImageViewer* application built with `vtkINRIA3D`. The snapshot represents the preview screen, where each loaded image is displayed, and interactions on windows are synchronized.

```
vtkIsosurfaceManager* manager = vtkIsosurfaceManager::New();
manager->SetRenderWindowInteractor ( iren );
manager->SetInput ( segmentation );
manager->GenerateData();
manager->SetOpacity (0.5);
```

- `vtkTensorVisuManager` takes as input a `vtkStructuredPoints` and renders the tensor data by extracting a slice in each of the orthogonal directions (axial, sagittal and coronal), just like 3D MPR image visualization does. Several glyph representations are available: ellipsoid, line, arrow, disk, cylinder, cube and even superquadric. The color coding of the glyphs can be set to either use a scalar value and a user-provided look-up table, a specific eigenvalue or eigenvector weighted by the fractional anisotropy as it is done in DT-MRI. Finally, the resolution and sampling of the glyphs can be controlled to fasten the rendering. An example can be found in `Examples/TensorManager/TensorManager.cxx`. Fig. 3 right was produced using the following lines of code:

```
vtkTensorManager* manager = vtkTensorManager::New();
manager->SetRenderWindowInteractor (iren);
manager->SetInput (tensors);
manager->SetGlyphShapeToCube();
manager->ResetPosition();
manager->Update()
```

- `vtkFibersManager` is a class that was first developed to interact with thousands VTK polylines produced in DT-MRI tractography: each line represents a single white matter fiber. The main contribution is the use of a `vtkBoxWidget` to limit the visualization to fibers that go through the box. The box can be scaled and translated. Extracted polylines can be displayed either by lines, ribbons or tubes. Recursive exploration of fibers is possible, i.e., one can take the subset of fibers currently displayed as a new starting point for exploration, and so on, to extract a fiber bundle of interest for example. An example can be found in `Examples/FibersManager/FibersManager.cxx`, and the sample of the code that generated Fig. 4 left is given below:

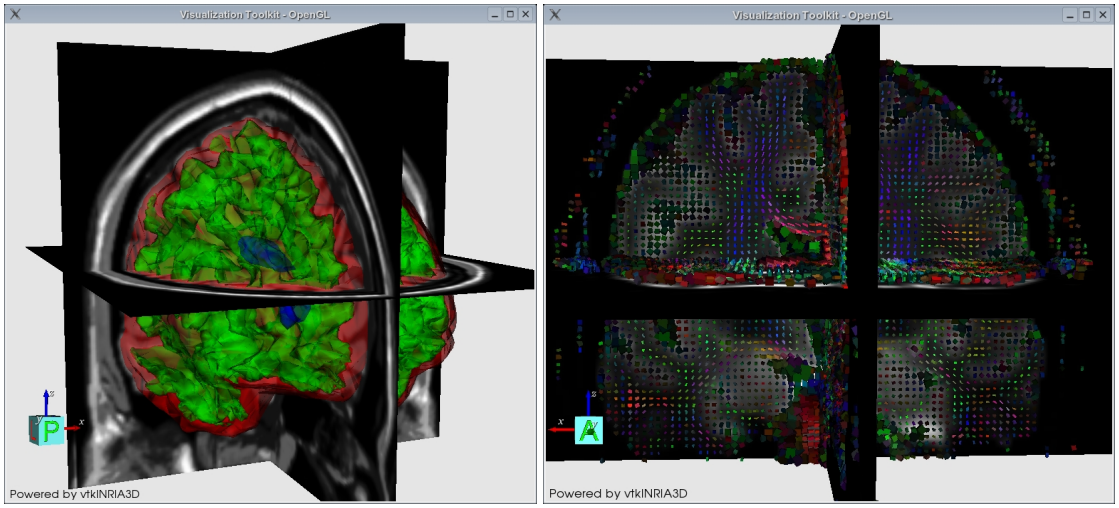


Figure 3: The left image was produced with the example in `Example/IsosurfacesManager/IsosurfacesManager.cxx` The right image was produced with the example in `Examples/TensorManager/TensorManager.cxx`. The test data are provided with vtkINRIA3D (image from BrainWeb, segmentation courtesy of Olivier Clatz, Ph.D.).

```
vtkFibersManager* manager = vtkFibersManager::New();
manager->SetRenderWindowInteractor (iren);
manager->SetInput (fibers);
manager->BoxWidgetOn();
```

- The `vtkCompareImageManager` takes two images as input and fuse them into a single image. Two fuse modes are available. The first one uses a `vtkImageBlend` to blend the two inputs according to an alpha parameter controlled by the user. The second one uses a `vtkImageCheckerboard` whose number of divisions is also controlled by the user. An example can be found in `Examples/CompareImageManager/CompareImageManager.cxx`. and is briefly described below (with Fig. 4 right):

```
vtkCompareImageManager* manager = vtkCompareImageManager::New();
manager->SetInput1 ( image1 );
manager->SetInput2 ( image2 );
manager->SetComparisonMode ( vtkCompareImageManager::COMPARE_GRID );
manager->SetNumberOfDivisions (10, 10, 10);
manager->GenerateData();
```

- `vtkLookupTableManager` is a simple but useful class that provides the user with some of the standard `vtkLookupTable`: Black&White, Hot Metal, Full Spectrum, etc.

To conclude, the `vtkVisuManagement` library is a combination of some of the VTK classes into new classes that simplify the incorporation of complex data manipulation and visualization into homemade softwares.

We propose in the next section an original way of managing VTK data, which facilitates input, output, memory allocation, and which also supports temporal sequences of data.

4 The vtkDataManagement Library

VTK provides classes for the manipulation of various types of data, such as polygonal data (`vtkPolyData`), unstructured grids (`vtkUnstructuredGrid`) and images (`vtkImageData`), all deriving from the base class

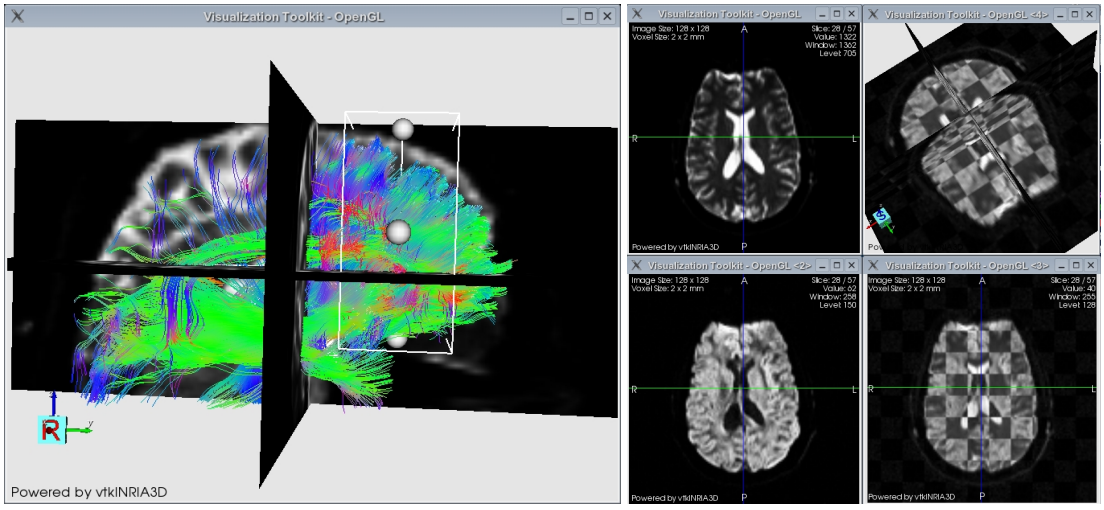


Figure 4: The left image was produced with the example in `Examples/FibersManager/FibersManager.cxx` The right image was produced with the example in `Examples/CompareImageManager/CompareImageManager.cxx`.

`vtkDataSet`. From a developer point of view, it would be interesting to have the possibility to:

1. Use and display any of these datasets with minimal effort,
2. Have access to each individual dataset, process it, and observe the result directly,
3. Be able to work with time-sequences, i.e., a sequence of objects of the same type describing a process evolving with time (e.g., cardiac imaging).

The library `vtkDataManagement` is a concrete implementation of these requirements and allows to rapidly and simply integrate a visualization system into a program. It consists of two main components:

- The class `vtkDataManager` is the core of the library. This class is the interface class between any `vtkDataSet` and the developer. It can be seen as a container for VTK objects.
- The class `vtkMetaDataSet` (and its derived classes). This class extends the `vtkDataSet` class by adding new methods and attributes, such as Input/Output, time flag, etc.

In the following, we first start by describing the two main components of the library, and then give more details on how time-sequences are supported in the `vtkDataManagement` library.

4.1 The vtkDataManager and the vtkMetaDataSet Classes

The purpose of the class `vtkDataManager` is to handle spatiotemporal data. It is basically a container for several `vtkDataSets`, and simplifies their creation (allocation) and manipulation by providing convenient methods such as read, write, and access. Moreover, it is able to automatically detect and create the right type of data when reading. An example of how to use this manager is given in folder `Examples/DataManager/`, illustrated in Fig. 7 and given in appendix A.2.

As shown in Figure 5, the `vtkDataManager` does not support directly instances of the class `vtkDataSet` but of the class `vtkMetaDataset` instead. The class `vtkMetaDataset` not only contains a reference to

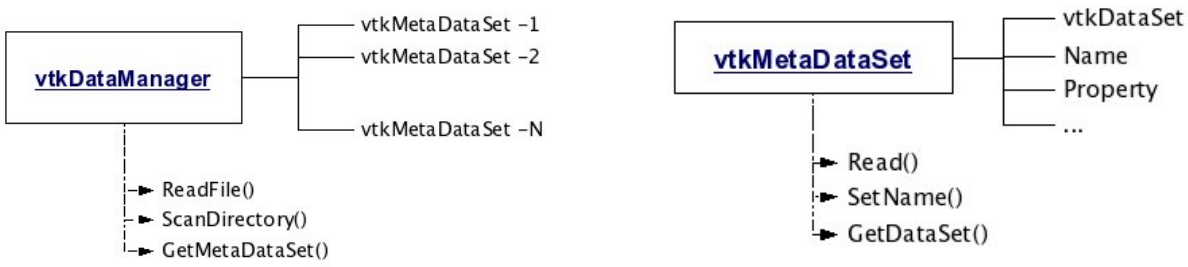


Figure 5: **Left:** The principal members of the class `vtkDataManager`. It contains a list of `vtkMetaDataSet`. Calling `ReadFile()` opens a file, creates the correct type of dataset, and adds it to its list. Calling `ScanDirectory()` performs a full scan of a directory, adds any available dataset to its list. `GetMetaDataSet()` allows to access any item of the list. **Right:** Principal members and methods of `vtkMetaDataSet`. This base class carries a `vtkDataSet` and some extra attributes, like a name, a `vtkProperty` and a time flag. I/O methods are provided. A concrete example can be found in `Examples/DataManager/DataManager.cxx` and is described in Sec. A.2.

a `vtkDataSet` instance (accessible by the function `GetDataSet()`), but it also provides extra attributes like a name, a time flag (used for time support, see Sec. 4.2) or a `vtkProperty` to be shared by several representations of the same dataset. From this base class derive four other classes:

- The class `vtkMetaImageData` represents a `vtkImageData`. It provides input/output methods using the ITK I/O factory mechanism, thus supporting a large spectrum of medical image formats. Calling `GetDataSet()` returns an instance of `vtkImageData`.
- The class `vtkMetaSurfaceMesh` carries a `vtkPolyData` object. It uses the input/output methods of VTK. Calling `GetDataSet()` returns an instance of `vtkPolyData`.
- The class `vtkMetaVolumeMesh` supports `vtkUnstructuredGrid` objects. The input/output functions are those of VTK. Calling `GetDataSet()` returns an instance of `vtkUnstructuredGrid`.
- The class `vtkMetaDataSetSequence` carries several `vtkMetaDataSet` of the same type (i.e., several `vtkMetaImageData` for instance). It allows to handle a temporal sequence of datasets. More details on this class can be found in Sec. 4.2. Calling `GetDataSet()` returns an instance of `vtkImageData`, `vtkPolyData` or `vtkUnstructuredGrid` depending on the type of the sequence (the type is determined by the first object inserted).

One last important feature is that for each `vtkMetaDataSet`, a list of `vtkActor` can be set as attributes. This is useful when synchronizing different representations of the same data. For example, one could imaging visualizing a mesh of the heart in both 3D and 2D using the library `vtkRenderingAddOn`: the mesh is cut to the current image slice displayed by a `vtkViewImage2D`. Then, one would like to color code the mesh by a field array. Instead of accessing each instance of `vtkActor` individually (the 3D and 2D actors in this case), getting their `vtkMapper` and calling the appropriate method (here `ColorByArray()`), one can pass once for all the generated actors to the corresponding meta dataset, and call this same method from the meta dataset directly if it exists: all actors will be updated with one function call.

In the following, we give an in-depth description of the time support in the `vtkDataManagement` library.

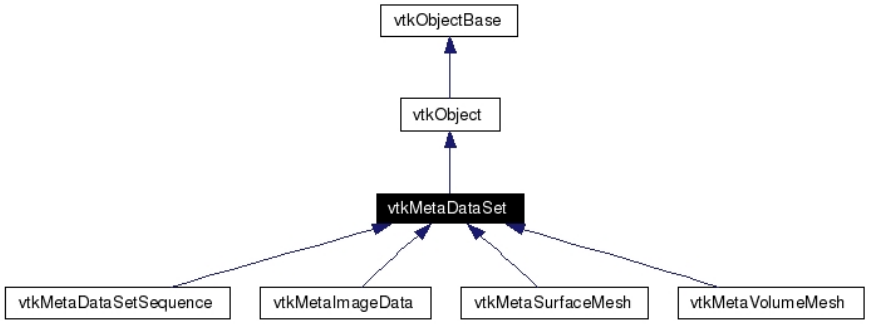


Figure 6: Hierarchy of the base class `vtkMetaDataSet` and its derived classes.

4.2 Time Support

Time support is achieved through the class called `vtkMetaDataSetSequence`. Basically, this class derives from a `vtkMetaDataSet` (Fig. 6 left) so that it is considered as a regular dataset by the `vtkDataManager`. The difference resides in the fact that it has a list of `vtkMetaDataSet` as attribute. When a sequence is read (via the `Read()` function which takes in this case a directory as parameter, and not a single file name), datasets are read and stored in its list according to their time flag. Then, the `vtkDataSet` member inherited from its base class (`vtkMetaDataSet`) is used as a buffer: the object is created (the memory address is set once for all), and the members (like point data, polygonal data, scalar arrays) will point towards the requested time point when the method `UpdateToTime(double)` is called (Fig. 8). Thus, no object is copied when `UpdateToTime()` is called (optimal memory use), and only the `vtkMetaDataSet` that serves as an interface will have its attributes changed. However, no temporal interpolation is made when the requested time point is not available in the list. Instead, the nearest temporal neighbor is chosen. This prevents memory copies but may not be the best solution when time points of the sequence are not regularly sampled: temporal interpolation would be necessary in this case and is part of the future work.

Several time sequences can be synchronized via the `vtkDataManager` by calling the function `UpdateSequencesToTime(double time)`. This is a very desirable feature in cardiac simulation, when

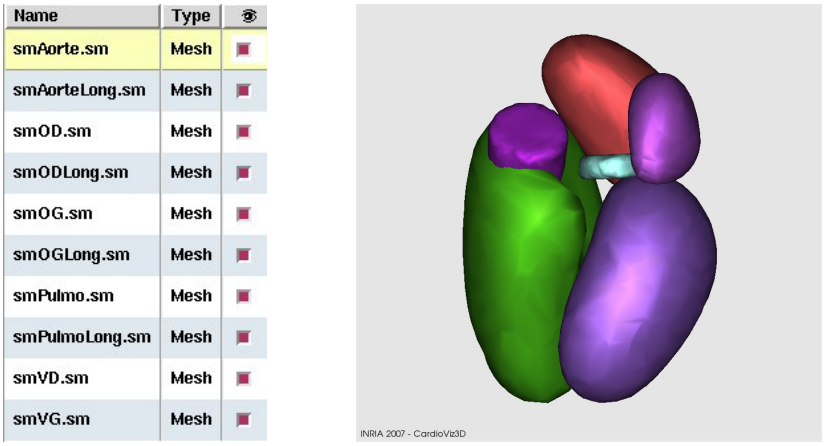


Figure 7: An example of the `vtkDataManager` in use. **Left:** A panel (built in KWWidget) summarizes the `vtkDataManager` content. The selected data are rendered in the 3D view on the right. The meshes here come from a heart segmentation (data courtesy of Thomas Mansi).

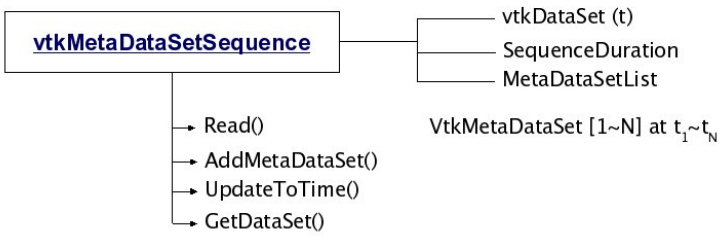


Figure 8: Structure of the `vtkMetaDataSetSequence` class. After adding several `vtkMetaDataSets` to this class with `AddMetaDataSet()` or `Read()`, the user may call `UpdateToTime(double time)` to see the attributes of the output dataset (`GetDataSet()`) change according to the requested time point. Attributes that are passed to the buffer vary with the type of `vtkMetaDataSet`. For instance, point coordinates and cell data are passed for a polygonal object while point scalars are passed for an image object.

an image plus a corresponding mesh sequence of the heart are displayed simultaneously.

The support of time sequences was very recently introduced in VTK, after the creation of the `vtkDataManagement` library. As the functionalities are not completely similar we chose to keep on developing our approach. However, the possibility to fuse with the latest VTK developments is still open.

5 Applications

We briefly describe in this section two applications that use vtkINRIA3D, VTK and ITK. The first one, called *MedINRIA*, is a collection of softwares for medical image processing and visualization and is dedicated to clinicians. The second one, called *CardioViz3D*, targets research in cardiac imaging.

5.1 MedINRIA

MedINRIA consists of a set of programs, each of them being dedicated to a specific application or MRI modality. For instance, a first application called *DTI Track* provides a processing and visualization pipeline for DT-MRI using Log-Euclidean metrics [5]. *RegistrationTool* allows to perform from manual rigid to fully automatic non-linear registration using state-of-the-art methods [8]. Other applications include a semi-automatic segmentation of MS lesions and a simple yet powerful *ImageViewer*.

It uses `wxWidgets` [3] for the user interface. `wxWidgets` has a proper interface with VTK (`http://wxvtk.sourceforge.net`) that we slightly modified and included in `vtkINRIA3D`.

MedINRIA is partly open-source, and binaries for Windows, Linux and MacOSX are freely available at: `http://www-sop.inria.fr/asclepios/software/MedINRIA`. The source of the *ImageViewer* application comes with `vtkINRIA3D`. Figure 9 left shows *DTI Track* running. A documentation is available on the same web site, as well as a set of test data.

5.2 CardioViz3D

CardioViz3D targets research in cardiac imaging. It is funded by the CardioSense3D project (`http://www-sop.inria.fr/CardioSense3D`) and aims at providing researchers with a set of tools for

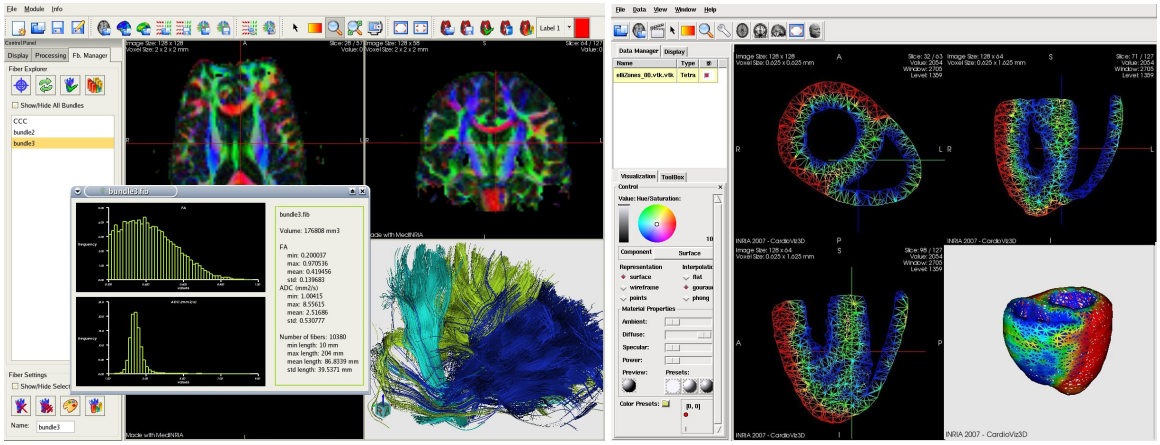


Figure 9: MedINRIA and CardioViz3D: Two applications built upon `vtkINRIA3D`. **Left:** MedINRIA running the application *DTI Track* with fiber bundles extracted from DT-MRI. **Right:** CardioViz3D is used to visualize the result of a cardiac simulation: a model of the heart is displayed in 3D (bottom right window) and in three 2D orthogonal views. The color codes for the propagation of the action potential.

the processing, simulation and visualization of cardiac data. At the current state of development (v. 1.2.7), the software proposes a visualization system of cardiac images and meshes, and supports time sequences.

CardioViz3D is built with ITK, VTK, vtkINRIA3D and KWWidgets [1]. The use of KWWidgets gives the possibility to researchers to write and run tcl scripts.

CardioViz3D is partly open-source, and a release of a beta version for Windows, linux and MacOSX is available at `https://gforge.inria.fr/projects/cardioviz3d/`. The core of CardioViz3D, called `KWAddOn`, is given with the source code of `vtkINRIA3D`.

6 Conclusions and Future Work

This paper presents `vtkINRIA3D`, a collection of new VTK classes to handle spatiotemporal datasets in terms of synchronization, visualization, and management. Practically, it consists of three libraries. First, the `vtkRenderingAddOn` library implements a strategy of synchronization of interactions and visualization of several `vtkRenderWindow`. It also provides a 2D and 3D visualization pipeline for volumes, with the possibility to use a 2D tracer and create interactively volume of interests in 3D. Second, the `vtkVisuManagement` library's goal is to facilitate the creation of complex visualization and interaction systems of datasets. It concerns tensor fields, large sets of `vtkPolyLines` (as produced in DT-MRI tractography), isosurfaces, and images. Third, the `vtkDataManagement` library aims at providing a single container of objects in an application to easily manage creation, deletion, and organization of datasets. Moreover, it supports temporal sequences of any type of data, e.g., images or polygonal data. Finally, we briefly described two applications based upon `vtkINRIA3D`: *MedINRIA*, a set of softwares for medical image processing with a clear user interface, and *CardioViz3D* a research software dedicated to cardiac imaging.

Future developments include new synchronized methods between views (such as the synchronization of 3D cameras), visualization techniques of other complex data (like ordinary distribution function as obtained in Q-ball imaging [7]), and temporal interpolation of datasets using the recent classes developed in VTK.

`vtkINRIA3D` is open-source, freely available at `http://www-sop.inria.fr/asclepios/software/vtkINRIA3D`.

A set of examples and test data can be downloaded as well.

A Examples

A.1 SynchronizedViews

The source code for this example can be found in `Examples/SynchronizedViews/SynchronizedViews.cxx`. The goal of this example is to synchronize the visualization of an image, i.e., synchronize the navigation into the slices and the window/level for contrast adjustment. We use subclasses of `vtkView`: `vtkViewImage2D` for 2D visualization and `vtkViewImage3D` and 3D visualization:

```
#include <vtkViewImage2D.h>
#include <vtkViewImage3D.h>
```

We create four of these: three `vtkViewImage2D` (one axial, sagittal and coronal view) and one 3D view of the image. The creation of one of the `vtkViewImage2D` is given below:

```
vtkViewImage2D* view1 = vtkViewImage2D::New();
vtkRenderWindowInteractor* iren1 = vtkRenderWindowInteractor::New();
vtkRenderWindow* rwin1 = vtkRenderWindow::New();
vtkRenderer* renderer1 = vtkRenderer::New();
iren1->SetRenderWindow (rwin1);
rwin1->AddRenderer (renderer1);
view1->SetRenderWindow ( rwin1 );
view1->SetRenderer ( renderer1 );
```

We now set some properties for the 2D/3D views: orientation, type of interaction, background color, etc.:

```
view1->SetInteractionStyle (vtkViewImage2D::SELECT_INTERACTION); // navigate through the
slices with the mouse
view1->SetOrientation (vtkViewImage2D::AXIAL_ID);
view1->SetBackgroundColor (0.0,0.0,0.0);
view1->SetAboutData ("Powered by vtkINRIA3D");

view4->SetRenderingModeToPlanar();
view4->SetCubeVisibility(1); // orientation cube
```

We finally link the views together for synchronization:

```
// view1 transmits to view2 which transmits to view3, etc.
view1->AddChild (view2);
view2->AddChild (view3);
view3->AddChild (view4);
view4->AddChild (view1);
```

When the program exits, or whenever one want to delete a view, one should call the method `Detach()`, so the synchronization mechanism is not stopped:

```
// Before exiting and deleting the VTK objects
view1->Detach();
view2->Detach();
view3->Detach();
view4->Detach();
```

A.2 The Data Manager

This example shows how to use the `vtkDataManager` class. The source code for this section can be found in `Examples/DataManager/DataManager.cxx`. This example scans a directory, reads any dataset available in it, and renders all of them in a `vtkView`. It shows how to scan a directory for a time sequence.

```
#include <vtkViewImage2D.h>
#include <vtkDataManager.h>
#include <vtkMetaImageData.h>
#include <vtkMetaSurfaceMesh.h>
#include <vtkMetaVolumeMesh.h>
#include <vtkMetaDataSetSequence.h>
```

We set the vtkView (visualization pipeline):

```
  vtkViewImage3D* view            = vtkViewImage3D::New();
  vtkRenderWindowInteractor* iren = vtkRenderWindowInteractor::New();
  vtkRenderWindow* rwin           = vtkRenderWindow::New();
  vtkRenderer* renderer           = vtkRenderer::New();

  iren->SetRenderWindow (rwin);
  rwin->AddRenderer (renderer);
  view->SetRenderWindow ( rwin );
  view->SetRenderer ( renderer );

  view->SetRenderingModeToPlanar();
  view->SetCubeVisibility(1);
  view->SetAboutData ("Powered by vtkINRIA3D");
```

We allocate a `vtkDataManager` and scan the user-provided directory:

```
  vtkDataManager* DataManager = vtkDataManager::New();
  DataManager->ScanDirectory(directoryname.c_str());
```

For reading a time sequence of data, we replace the above last line of code by:

```
  DataManager->ScanDirectoryForSequence(directoryname.c_str());
```

Then we just need to add the datasets in the view:

```
  for (unsigned int i=0; i<DataManager->GetNumberOfMetaDataSet(); i++)
  {
    vtkMetaDataSet* metadataset = DataManager->GetMetaDataSet (i);
    vtkProperty* prop = vtkProperty::SafeDownCast(metadataset->GetProperty());
    view->AddDataSet (metadataset->GetDataSet(), prop);
  }
```

Note that playing the sequence is done by calling `DataManager->UpdateSequencesToTime()` in a loop. It can be done in real time thanks to a `vtkTimerLog`.

References

[1] KWWidgets. http://www.kwwidgets.org/. 5.2

[2] VTK: The visualization toolkit. http://www.vtk.org. 1

[3] wxWidgets. http://www.wxwidgets.org. 2, 5.1

[4] P. Basser, J. Mattiello, and D. Le Bihan. MR diffusion tensor spectroscopy and imaging. *Biophysical Journal*, 66:259–267, 1994. 1

[5] Pierre Fillard, Vincent Arsigny, Xavier Pennec, and Nicholas Ayache. Clinical DT-MRI estimation, smoothing and fiber tracking with Log-Euclidean metrics. *IEEE Transactions on Medical Imaging*, 2007. In Press. 5.1

[6] L. Ibanez, W. Schroeder, L. Ng, and J. Cates. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, first edition, 2003. 2

[7] D. Tuch. Q-ball imaging. *MRM*, 52:1358–1372, 2004. 6

[8] Tom Vercauteren, Xavier Pennec, Aymeric Perchant, and Nicholas Ayache. Non-parametric diffeomorphic image registration with the demons algorithm. In *Proc. of MICCAI'07*, 2007. In press. 5.1

A Framework for Comparison and Evaluation of Nonlinear Intra-Subject Image Registration Algorithms

Release 0.1

Martin Urschler[1], Stefan Kluckner[1] and Horst Bischof[1]

July 15, 2007

[1]Institute for Computer Graphics and Vision, Graz University of Technology, Austria
Contact: urschler@icg.tu-graz.ac.at

Abstract

Performance validation of nonlinear registration algorithms is a difficult problem due to the lack of a suitable ground truth in most applications. However, the ill-posed nature of the nonlinear registration problem and the large space of possible solutions makes the quantitative evaluation of algorithms extremely important. We argue that finding a standardized way of performing evaluation and comparing existing and new algorithms currently is more important than inventing novel methods. While there are already existing evaluation frameworks for nonlinear inter-subject brain registration applications [8, 2], there is still a lack of protocols for intra-subject studies or soft tissue organs. In this work we present such a framework which is designed in an ”open-source” and ”open-data” manner around the Insight Segmentation & Registration Toolkit. The goal of our work is to provide the research community with the basis framework that should be extended by interested people in a community effort to gain importance for evaluation studies. We demonstrate our proposed framework on a sample evaluation and release its implementation and associated tools to the public domain.

Contents

1 Introduction **2**

2 Related Work **2**

3 A Nonlinear Intra-Subject Registration Evaluation Framework **3**
3.1 Compared Algorithms 4
3.2 Synthetic Deformations 6
3.3 Quantitative Evaluation Measures 7

4 A Sample Evaluation **9**
4.1 Discussion of the Evaluation Results 10

5 Conclusion & Future Work **11**

1 Introduction

Nonlinear (non-rigid) image registration is an important field for medical computer vision applications [4]. Since it resembles an ill-posed problem, it is also a very complex topic from the theoretical point of view. Unfortunately, validation of nonlinear image registration algorithms tends to be quite difficult as well, so we are facing a major challenge if quantitative statements about performance and comparison of algorithms are required.

On the one hand, the problem of validating nonlinear registration stems from the lack of ground-truth data which in general can not be derived from realistic, clinical data sets under investigation. On the other hand, the definition of suitable quantitative measures to compare different algorithms or an algorithm result to a synthetic ground truth also is a challenge. Many quantitative similarity metrics (which are frequently used in publications to compare registration results), like e.g. the sum-of-squared differences, are only judging outcomes of the registration process and they are often biased to algorithms which use the same or a related similarity measure in the cost function of their underlying optimization scheme.

Due to the problems outlined above and the inherently available influence of noise, partial volume effect, limited numerical precision, interpolation schemes, etc. on quantitative measurements we refer to algorithm evaluation (in accordance with [8]), i.e. measuring ”relative” algorithm performance, in contrast to validation or the measurement of ”absolute” performance.

We argue that the current state of the art in nonlinear registration algorithm evaluation is inadequate. For registration algorithms it is only possible to gain practical, clinical acceptance if the research community builds a standardized protocol for evaluation. Therefore, in this contribution we propose a modular evaluation framework for objectively comparing algorithms which is designed with the spirit of open-source, open-access, open-data and open protocols in mind, according to the basic ideas and principles from the Insight Segmentation & Registration Toolkit (ITK [9]) and the Insight Software Consortium.

The building blocks of our framework are:

- Freely available data sets, e.g. from the NLM data collection project [12], the MIDAS project from the Insight Software Consortium [3] or the NCIA project [11].
- A number of freely available algorithms for nonlinear registration, like e.g. the ITK is able to provide and continuously gathers.
- A set of quantitative measures on which the research community has agreed upon, with public domain implementations residing in a central repository.
- A pool of synthetic deformation models, again with the agreement of the research community, either defined analytically or derived by using the existing pool of registration algorithms.
- An evaluation framework in an easily customizable scripting language that forms the glue for the rest of the components.

2 Related Work

Given the current state of the art in nonlinear intra-subject registration we consider it more important to perform research on the evaluation of algorithms than inventing a novel one. In literature one finds many

publications on algorithms which do not focus on a thorough evaluation and there are only few publications which are entirely dedicated to evaluation.

A common approach to perform evaluation is to identify landmark or region correspondences independently of the registration and to measure how well they are aligned after registration. While this is a sufficient method for rigid registration [23] and also for some nonlinear registration applications [8] it obviously has drawbacks in the latter case since the error in regions far from the landmarks in general can not be measured.

In literature we can find several similar attempts to create evaluation projects. Examples are the retrospective rigid registration evaluation project from West et al. [22], the VALMET project for evaluation of segmentation results [6] or more recently the NIREP project for evaluation of nonlinear inter-subject registration algorithms [2]. Additionally the success of community-based evaluation efforts are also obvious in related computer vision fields like stereo correspondence algorithms [14] and multi-view 3D reconstruction [16]. However, currently there are no projects that focus on a comparison of the large number of available algorithms for intra-subject nonlinear registration.

Next we give some examples for works that deal with nonlinear registration evaluation. In [8] an evaluation study on brain data sets is presented where six inter-subject registration methods are investigated and compared using a number of local and global measures. The main focus lies on the correct alignment of those brain landmarks which are assumed to be present in all individuals. In [15] a finite element method is used to create a physically plausible synthetic deformation model that might be used for comparison with registration results. Their focus lies on the evaluation of B-Spline grid based methods [13] applied to mammography MR input data. The work of [21] presents an evaluation of the Demons [18] algorithm applied to prostate CT images. They use synthetic experiments to argue on the suitability of Demons for this specific application. The focus of [17] is on the comparison of different similarity measures in the context of nonlinear registration. Their protocol includes a number of optimization-independent, statistical measures describing capture range, number, location and extent of local optima as well as the accuracy and distinctiveness of optima of a cost function derived from a similarity measure. In [2] a framework for the evaluation and comparison of inter-subject brain registration algorithms is presented. Their project NIREP (Nonlinear Image Registration Evaluation Project) is most related to our work, however, its dedication to inter-subject registration and its incompatibility with the idea of ”open-source” and ”open-data” are the major restrictions.

3 A Nonlinear Intra-Subject Registration Evaluation Framework

The main problem in nonlinear image registration evaluation is the lack of ground truth information from clinical data sets. Researchers therefore have to restrict themselves to define practically relevant synthetic experiments. There are several groups of synthetic experiments possible, i.e. (1) synthetically deformed synthetic data, (2) synthetically deformed phantom data and (3) synthetically deformed real data. In this paper we concentrate on the third kind of experiments, however, the framework allows the usage of the first two input types as well. The choice of synthetic deformations is crucial in an evaluation procedure. If the chosen transformation is too simple or restrictive, then the derived quantitative measures are restricted to a very coarse approximation of the originally investigated problem. This fact especially complicates the task of nonlinear registration.

Our approach is to use a combination of simple synthetic transformation methods to derive quantitative measures on the performance of several nonlinear registration algorithms. We assume that the calculation of many different synthetic deformations, with each of them testing different behavior, along with a large number of different evaluation measures leads to a method for thorough evaluation and comparison. We further hypothesize in this paper that one can make use of a ”pool” of different, sufficiently dissimilar

methods to create a number of synthetic deformations. If this ”pool” of methods is large enough one can assume that the evaluation converges to a ”bronze standard” evaluation, a term which was borrowed from a publication of Glatard et al. [7] who describe a way of establishing a ”ground truth” in rigid registration evaluation by combining a large number of data sets with a number of algorithms and analyzing all possible combinations of registration results. One of our most important assumptions is, that a community effort is needed to increase the number of algorithms, measures and synthetic transformations in order to arrive at a practically relevant evaluation protocol.

We have built an evaluation framework to assist in the quantitative comparison of different algorithms over a variety of synthetic transformations. This framework uses Python as high-level scripting language to call the C++ methods that perform the synthetic transformation calculation, call the registration algorithms and the computation of the quantitative evaluation measures. Input images are synthetically transformed with the help of a configuration file. A list of nonlinear registration algorithms along with its parameterizations may be specified to perform the computations. Finally the synthetic ground-truth deformation fields and the original moving images are compared with the outcomes of the registration algorithms and log files with the quantitative evaluation results are written. Note, that all of these processes are performed in an easily customizable fully automatic fashion. Another important aspect this evaluation framework allows, is to automatically test algorithms with different parameterizations to determine optimal parameters with respect to the evaluation measures.

Our basic strategy for the evaluation is in all cases identical. We take an original image, apply a synthetic transformation and store the synthetically warped image and the resulting displacement field. The displacement field will be our ground truth to compare to. Now each of the investigated nonlinear registration algorithms gets the synthetically warped image as fixed input and the original image as moving input, i.e. we try to find a displacement field that warps the original image to the synthetically transformed. In this way we can finally compare the warped image with the synthetically warped image and the ground truth displacement field with the calculated displacement field. Note that only this way it can be guaranteed that the displacement fields represent transformations in the same directions (from fixed to moving image). Compare Figure 1 for an overview of this strategy. This figure also clearly shows that the nonlinear registration algorithms have no insights on the working of the synthetic transformations, making their behavior completely independent from these transformations. A possible bias that one method might have concerning a certain synthetic transformation may be removed by increasing the number of investigated synthetic deformations.

3.1 Compared Algorithms

In this section we list the nonlinear registration algorithms from the ITK that are currently used in the framework. We shortly describe each method and the parameters that have to be chosen for the evaluation procedure. There are two parameters that are required in all of the algorithms, the number of levels in the Gaussian pyramid approach and the number of iterations per pyramid level.

Demons Registration - ”demons” is an approximation to the elastic or fluid deformation model, where the computation of the similarity metric and the regularization of the displacement field are decoupled and the latter part is approximated by a Gaussian smoothing [18]. The only additionally required parameter is the standard deviation sigma of the displacement field smoothing step.

Symmetric Demons Registration - ”symdemons” builds upon the same principles as Demons, however, the gradient term in the similarity metric computation uses information from both images to be registered, thereby increasing robustness. The only required parameter is the standard deviation sigma of the displacement field smoothing step.

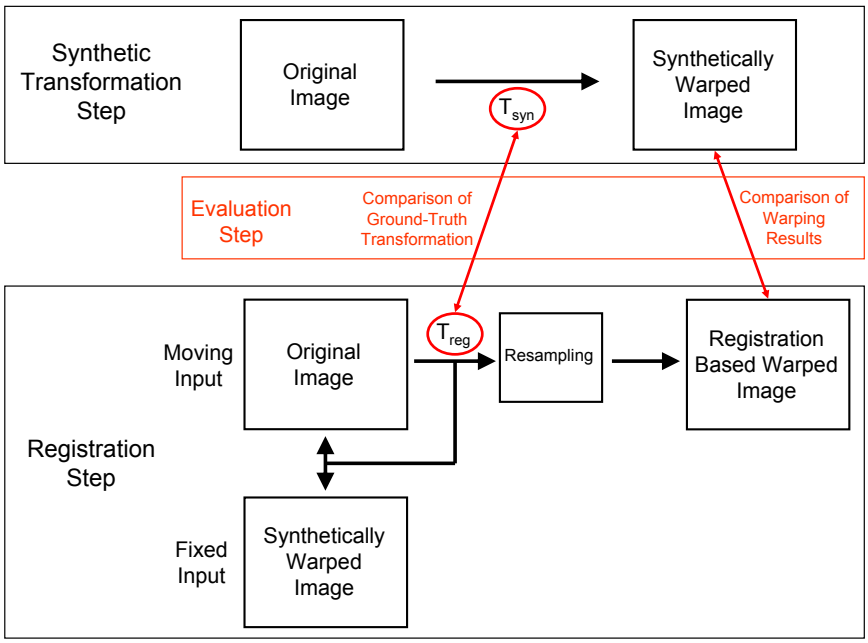


Figure 1: The basic setup for the synthetic transformation evaluation experiments. The synthetic transformations are applied to the original image, the nonlinear registration of original and synthetically transformed image leads to a transformation and a warped image which can be compared.

Level Set Motion Registration - "levelsetmotion" is a very efficient registration algorithm that interprets the image intensity as isocontours of a level set and tries to match those iso-contours [20]. It gains its efficiency from the fact that no regularization is used.

Curvature Registration - "curvature" uses a regularization term that penalizes the norms of the second derivatives of the displacement field components to perform registration [5]. In this implementation the variational formulation for combining similarity and regularization term is used [10] and numerically solved using a fast fourier transform. Additional parameters are the time-step τ of the semi-implicit scheme and the regularization weight α.

We propose to add two more algorithms to our framework, which currently are not included in the official ITK release, but were submitted to the 2007 MICCAI Open Science Workshop. Here obviously the open manner of this workshop shows its clear advantage compared to traditional workshops.

Fast Block Matching Registration - "fastblock" is a 2007 MICCAI Open Science workshop contribution by Suarez-Santana et al. It is a simple algorithm relying on a pyramidal block-matching scheme and a local entropy-based similarity measure. There are two parameters required in this algorithm which define regularization and similarity.

Diffeomorphic Demons Registration - "diffdemons" is another 2007 MICCAI Open Science workshop contribution by Vercauteren et al. It describes a diffeomorphic extension of the Demons algorithm in order to increase its suitability for inter-subject registration. Although inter-subject registration is not in the focus of this work, we chose to include this paper in order to enlargen the pool of available algorithms. There are two additionally required parameters, the standard deviation sigma of the displacement field smoothing and the maximum step length of displacement field updates.

Figure 2: Synthetic transformation - Simulated breathing (*simbr*). a) original data set, b) effect of $t_{vertical} = 25mm$ and $t_{inward} = 10mm$ and c) effect of $t_{vertical} = 55mm$ and $t_{inward} = 25mm$.

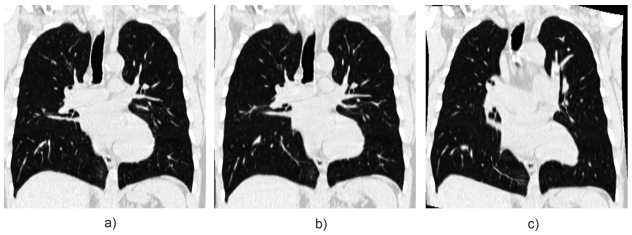


Figure 3: Synthetic transformation - grid based (*grid*). a) original data set, b) effect of $gridSize = 32$ and $maxDeviation = 2$ and c) effect of $gridSize = 32$ and $maxDeviation = 4$.

The reader may note that we decided not to include the B-Spline based algorithm using a MeanSquares metric or the Finite Element based elastic registration. The B-Spline method in our experience requires too much computation time, we have not yet managed to bring down the computational effort near the times of the other ITK algorithms. The decision to ignore the Finite Element based registration was made due to the very complex way this algorithm has to be set up. We invite the community to add these implementations to the framework.

3.2 Synthetic Deformations

We are using several different kinds of synthetic transformations. Note that in this paper the choice of synthetic deformations is guided by the intended application of evaluating thoracic soft-tissue data sets. However, the framework certainly allows to add and remove any kind of synthetic transformation to tailor the evaluation to the intended application area.

Our first synthetic model is a very simple simulated breathing transformation. It depends on two parameters $t_{vertical}$ and t_{inward} which provide a means for a simple approximation of diaphragm and rib-cage movement. Further, a slight intensity variation is applied to the interior of the lung, to simulate the partial volume effect. A more detailed description of this transformation can be found in [19]. We will denote it *simbr*. Examples for this transformation are given in Figure 2, for the experiments we use two instances of this transformation (*simbr-25-10*, *simbr-55-25*).

The second synthetic transformation consists of a regular grid with random deformations, interpolated by a thin plate spline. There are two parameters that may be varied, the grid size of the points to be placed and the possible extent of the random deformation that is applied per grid-point. It will be denoted *grid* from now on. The parameters of this transformation are *gridSize* and *maxDeviation*. Figure 3 gives some examples for this transformation, later on we will be using two instances of this transformation (*grid-32-4*, *grid-32-8*).

The third synthetic transformation is a uniform periodic one which we took from [24]. Here the displacement

field follows a cosinus distribution with a given amplitude and a given phase. We use a single instance of this transformation with amplitude 5 and phase 32 (*uniform-5-32*).

Our final form of synthetic transformations makes use of the pool of nonlinear registration algorithms described above. Here, given two input images that differ in a nonlinear fashion, we calculate the nonlinear registration of these two data sets with each of the algorithms. For n algorithms this gives us n displacement fields. Now it is possible to use each of these n displacement fields as synthetic deformations to synthetically warp one of the input images. Thus we get a pair of images to be registered by all of the available algorithms.

Note that we currently do not simulate noise in these synthetic deformation models, however, this could easily be incorporated by modifying the programs for calculating the synthetic deformations.

3.3 Quantitative Evaluation Measures

Two kinds of measures are used in our evaluation framework. The first one is used to compare displacement field ground truth data with displacement field results calculated during the registration. The second kind of measures compares two sets of images, before and after registration. This comparison step is performed on fixed and moving input image and on fixed and warped moving image. After successful registration, warping the moving image should result in an increase of similarity with the fixed image.

We always compute the evaluation measures only on the overlapping regions of the input image data sets and the deformation fields. This is especially important in the case of synthetic transformations, where certain areas might vanish due to a shrinking like behavior, we always mark these regions with special values during synthetic transformation and we omit those marked regions from the evaluation process.

We have to remark here that all of these measures possess one inherent shortcoming. Data sets with a size of $256 \times 256 \times 256$ consist of a total around 16 million voxels and each measure of interest will try to reduce this huge amount of information to one single number. Although some of our measures are inspired by robust statistics, their information value has to be at least discussed and also questioned. One should always keep this in mind when looking at quantitative evaluation results.

Displacement Field Measures

To assess the similarity of a synthetic ground-truth deformation field φ_{syn} and a deformation field computed by a nonlinear registration algorithm φ_{reg}, we use:

Root-Mean-Square of Displacement Field RMS_{disp} The Root-Mean-Square of the displacement field interprets the two displacement fields of interest as feature vectors of dimension $3 \times N_1 \times N_2 \times N_3$, where N_i is the number of voxels on the input image grid in the according dimension. The measure reads

$$RMS_{disp} := \sqrt{\frac{1}{3N_1N_2N_3} \sum_{\mathbf{x} \in \Omega} \sum_{i=1}^{3} \left(\varphi_{syn}(\mathbf{x}_i) - \varphi_{reg}(\mathbf{x}_i)\right)^2}$$

Median Absolute Deviation of Displacement Field MAD_{disp} This measure is similar to the Root-Mean-Square, however it is a more robust variant in the presence of outliers. The median absolute deviation is defined as

$$MAD_{disp} := \text{Median}\left(|m_i - \text{Median}\left(m_i\right)|\right)$$

with $m_i = |\varphi_{syn}(\mathbf{x}_i) - \varphi_{reg}(\mathbf{x}_i)|$ and $i = 1 \cdots 3$.

Maximum Deviation of Displacement Field MAX_{disp} The maximum deviation of the displacement field states the maximal difference of all components of the two displacement fields $\varphi_{syn}$and φ_{reg}. We use a more robust maximum, i.e. our maximum difference is defined as the difference that is larger than 95% of all other values.

Jacobian of Displacement Field JAC_{disp} The Jacobian of the displacement field gives information about the transformations consistency. Especially it identifies possible singularities of the field. The Jacobian is defined as the determinant of the first partial derivatives of the transformation (compare the definition in [8]), negative values of the determinant resemble singularities, i.e. foldings. The measure JAC_{disp}scans the deformation field for negative values and reports its number as a percentage of the total number of voxels, ideally it would be 0.

Image Similarity Measures

To assess the similarity of images (fixed image $I_F(\mathbf{x})$, moving image $I_M(\mathbf{x})$) before and after registration we use:

Root-Mean-Square of Intensity Differences RMS_{int} The Root-Mean-Square of the intensity differences needs as its input the pixel-wise intensity differences over the overlapping region of the image domain. It is defined as

$$RMS_{int} := \sqrt{\frac{1}{N_1 N_2 N_3} \sum_{\mathbf{x}\in\Omega} d(\mathbf{x})^2}$$

with

$$d(\mathbf{x})^2 := \begin{cases} 100 & \text{if } (I_F(\mathbf{x}) - I_M(\mathbf{x}))^2 > 100 \\ (I_F(\mathbf{x}) - I_M(\mathbf{x}))^2 & \text{otherwise} \end{cases}$$

We decided to clamp intensity differences that are larger than 100 Hounsfield Units and assign the absolute difference of 100 Hounsfield Units to all of these differences. This makes the measure more sensitive to lower intensity differences, the unmodified Root-Mean-Square measure would weight outliers too strong. Note that we perform this clamping consistently for comparing all image pairs, either original fixed and original moving or original fixed and warped moving.

Median Absolute Deviation of Intensity Differences MAD_{int} This measure is similar to the Root-Mean-Square, however it is a more robust variant in the presence of outliers. The median absolute deviation is defined as

$$MAD_{int} := \text{Median}\left(|d(\mathbf{x}) - \text{Median}\left(d(\mathbf{x})\right)|\right)$$

with $d(\mathbf{x}) = |I_F(\mathbf{x}) - I_M(\mathbf{x})|$.

Maximum Intensity Difference MAX_{int} The maximal intensity differences on the overlap region of the image grid. We use a robust maximum, i.e. our maximum intensity difference is defined as the intensity difference that is larger than 95% of all other values. Here of course no clamping (like for the Root-Mean-Square measure) of the intensity differences is performed.

Normalized Mutual Information NMI_{int} The normalized mutual information is a measure from information theory, which relates the information content of two images by probability distributions of the gray values. We use the formulation from the ITK normalized mutual information histogram metric

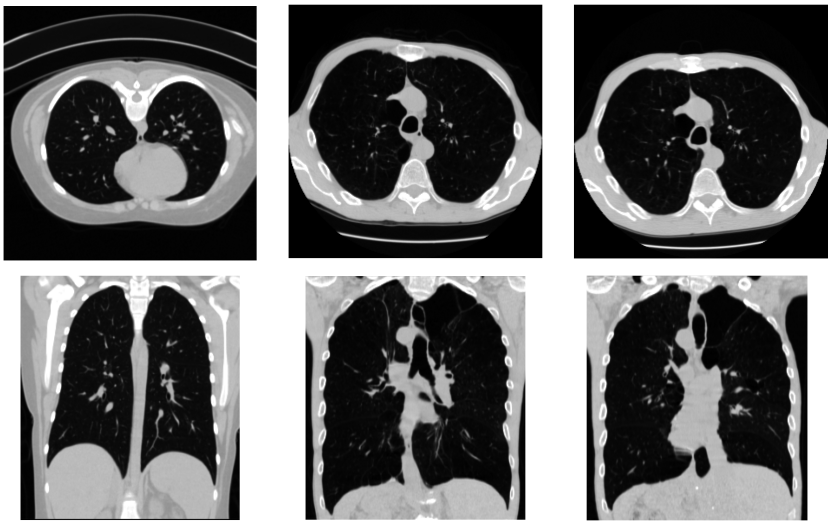


Figure 4: Our input data. a) Data set ”nlm”. b), c) Data set ”ncia” at two different breathing states.

that divides twice the mutual information by the sum of the individual entropies resulting in a measure between 0 and 1.

Edge Overlap $EDGE_{int}$ We are using a simple scheme to extract strong gradients from both images, based on the Canny edge detector [1]. The Canny edge detector is used with a σ of 3 times the voxel spacing, the result is a binary image with edges marked with a 1. The sum of the absolute differences divided by the number of voxels in the overlap region is used to compare the binary images. This function lies between 0 and 1 with 0 denoting optimal overlap.

4 A Sample Evaluation

To show the applicability of our evaluation framework we now present an evaluation example. The intention is to test different algorithms for the purpose of registering intra-modality thoracic CT data sets under the influence of breathing differences. Therefore, we are using two different publicly available data sets. First, we took the data set ”NormalChestCTNoContrast” from the NLM (National Library of Medicine) Data Collection Project [12]. This data set was used for the anlytically defined synthetic deformation experiments (see Figure 4a) and is abbreviated *nlm*. Second, we downloaded a pair of data sets from the NCIA (National Cancer Imaging Archives) repository [11] that differ by some breathing motion (see Figure 4b,c). We used it for the creation of synthetic deformation fields from the different registration algorithms. We refer to this data as *ncia*, it can be found in the LIDC section of the archives with the identifier ”30047”.

All of the data sets were resampled to a resolution of $256 \times 256 \times 256$ voxels to limit the run-time of some of the algorithms to a reasonable time-span. We performed our experiments on 64 bit AMD Opteron processors with 2.4GHz and 8GB of RAM running Linux. Since algorithms tend to need a large amount of system memory the choice to use a 64 bit system was necessary.

The necessary parameters for the different algorithms were chosen as follows. For the two algorithms that were taken from the MICCAI Open Science workshop contributions we directly took the proposed

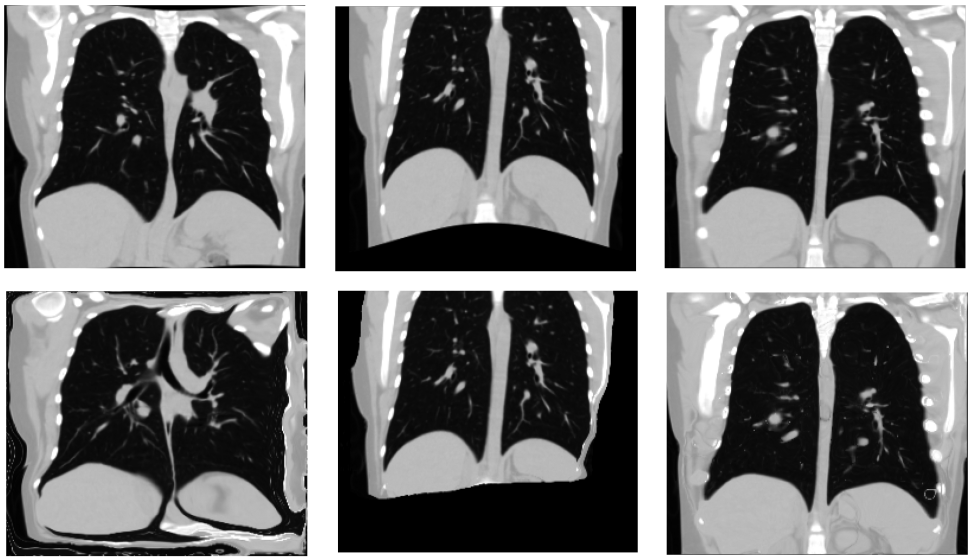


Figure 5: Problems of some registration algorithms. Top row is the target image and bottom row the registration result of a) fast block matching, b) symmetric Demons and c) level set motion algorithm.

default arguments without tuning. The other algorithms were set up with 4 shrink levels in the Gaussian pyramid, and a number of iterations that was calculated from two parameters $iter1 = 35$ and $iter2 = 25$ as $iter = iter1 * shrinkLevel + iter2$ in order to perform many more iterations on higher levels in the pyramid where calculation is cheap. The *sigma* parameter for demons and symmetric demons was chosen to be 1 voxel. τ and α in the curvatrue based algorithm were both set to 1.0. After this choice all of the parameters remained fixed during the evaluations.

4.1 Discussion of the Evaluation Results

The results of our evaluation can be found in Tables 2, 3, 4, 5. Due to the large number of quantitative measures resulting from the framework we decided to discuss it only in an overall manner.

What one can immediately see from these results, is that there are two algorithms which are not performing well on the synthetic data. These are the level set motion registration and the fast block matching registration. We assume that the level set motion method fails due to its lack of regularization and therefore do not suggest that it should be used. Especially the large percentage of negative values of the jacobian determinant of the displacement field are a problem. The problems are also visible on result images like in Figure 5a. The problems of the fast block matching algorithm are in the border region (see Figure 5b). We are not yet sure if this is an implementation problem or a problem of the method. This algorithm should be adapted and evaluated anew. The problems of these two algorithms also makes the synthetic deformations that use these two algorithms obsolete.

The overall best results on the evaluation experiments is given by the diffeomorphic demons algorithm. Also the Demons algorithm performs very well, however on the important displacement field comparisons diffeomorphic demons behaves better. The percentage of negative jacobian determinants is zero for diffeomorphic demons as could be expected. The curvature algorithm wins some of the performance measures as well, however there is no clear benefit compared to diffeomorphic demons and the run-times are approximately

Algorithm	demons	symdemons	levelset	curvature	fastblock	diffdemons
Runtime [s]	800	2800	850	3500	1700	3100

Table 1: Computational effort of the various investigated algorithms.

equal. As for the symmetric demons implementation, it performs quite well, however, there are some situations where the quantitative measures grow very high. This corresponds to misregistered data sets (see Figure 5c), we suppose that there might be an implementation problem in the itkFastSymmetricDemonsFunction.

We omit presenting output images of well registered data sets, since the difference to the original image is seldom visible with the eye.

The run-times of the different algorithms on the $256 \times 256 \times 256$ can be found in Table 1. Note that these are only approximate values. We can see that the practically quite well performing Demons algorithm also offers fast computation times.

To conclude we currently would suggest using Demons for time-critical registration of thoracic CT data and diffeomorphic Demons for the most accurate registration.

5 Conclusion & Future Work

In this paper we have presented an evaluation framework for evaluating and comparing nonlinear registration algorithms with the special intent on intra-subject applications. We have successfully shown how it is possible to automatically compare open-source algorithms on public domain data using our tools which we provide in conjunction with this submission. Our main goal is to create a standardized way of evaluating algorithms maintained at a centralized and open repository, e.g. the Insight Consortium server structure. In order to gain importance we invite the research community to contribute and/or discuss algorithms, quantitative measures and synthetic transformations. We know that our sample evaluation is not yet representative, but we hope to converge towards some kind of ”bronze standard” when collecting more and more modules.

Further work beside the implementation of the framework on a central repository should definitely include to upgrade the Python based framework to automatically distribute the computations among different computers in a cluster.

Measure			simbr-25-10	simbr-55-25	grid-32-4	grid-32-8	uniform-5-32
RMS_{disp}	*initial*	[mm]	7.937	18.40	1.930	3.782	3.848
	demons	[mm]	1.642	7.880	0.915	1.748	2.811
	symdemons	[mm]	7.470	21.52	1.686	3.968	3.153
	levelset	[mm]	18.52	25.30	11.86	12.61	10.63
	curvature	[mm]	5.731	10.29	3.450	4.052	2.666
	fastblock	[mm]	65.89	70.67	56.42	nan	75.26
	diffdemons	[mm]	1.017	7.757	0.222	0.638	1.689
MAD_{disp}	*initial*	[mm]	2.479	5.945	0.806	1.524	2.541
	demons	[mm]	0.055	0.500	0.043	0.076	0.187
	symdemons	[mm]	0.081	0.967	0.054	0.114	0.165
	levelset	[mm]	4.670	7.764	2.486	2.983	3.115
	curvature	[mm]	0.048	0.079	0.020	0.027	0.037
	fastblock	[mm]	11.26	19.61	2.706	5.929	16.33
	diffdemons	[mm]	0.024	0.310	0.054	0.116	0.365
MAX_{disp}	*initial*	[mm]	18.34	41.39	3.628	7.386	6.608
	demons	[mm]	3.596	22.16	0.956	3.525	6.112
	symdemons	[mm]	7.004	47.51	2.517	8.836	7.205
	levelset	[mm]	41.27	55.28	26.08	27.44	24.03
	curvature	[mm]	7.281	16.95	4.113	5.728	6.168
	fastblock	[mm]	166.6	168.7	153.3	152.2	178.8
	diffdemons	[mm]	0.658	21.97	0.452	1.267	4.200
JAC_{disp}	*demons*	[%]	0	0	0	0	4.58956e-06
	symdemons	[%]	0.0282316	0.123886	0.00223255	0.00272346	0.011709
	levelset	[%]	0.271759	0.365262	0.195936	0.20556	0.191423
	curvature	[%]	0.0569056	0.224993	0.00568753	0.0122414	0.0112995
	fastblock	[%]	0.445634	0.529313	0.391153	0.377419	0.475234
	diffdemons	[%]	0	0	0	0	0

Table 2: Registration results for the analytically defined synthetic experiments. Displacement field measures.

Measure			demons	symdemons	levelset	curvature	fastblock	diffdemons
RMS_{disp}	*initial*	[mm]	10.68	11.36	20.00	11.76	64.96	10.23
	demons	[mm]	2.115	3.166	22.04	6.182	61.74	1.424
	symdemons	[mm]	6.152	6.589	22.30	8.190	61.81	5.259
	levelset	[mm]	17.89	17.91	15.40	17.93	66.34	17.39
	curvature	[mm]	7.160	5.966	21.47	6.804	60.82	6.298
	fastblock	[mm]	71.44	73.48	77.31	74.63	36.82	71.15
	diffdemons	[mm]	1.563	3.355	21.53	5.154	59.61	0.716
MAD_{disp}	*initial*	[mm]	3.810	4.216	6.486	4.434	14.19	3.799
	demons	[mm]	0.222	0.657	7.192	1.026	9.844	0.077
	symdemons	[mm]	0.318	0.554	7.916	0.743	7.469	0.106
	levelset	[mm]	5.598	5.680	4.111	5.900	16.14	5.456
	curvature	[mm]	0.325	0.588	7.290	0.270	6.391	0.073
	fastblock	[mm]	13.58	14.64	24.10	15.45	2.977	13.78
	diffdemons	[mm]	0.306	0.732	6.622	1.022	6.853	0.125
MAX_{disp}	*initial*	[mm]	22.80	23.87	43.97	24.94	152.2	21.99
	demons	[mm]	4.020	7.388	48.74	14.29	147.7	2.883
	symdemons	[mm]	13.04	14.89	47.91	20.07	149.1	11.77
	levelset	[mm]	37.87	37.86	34.95	38.24	154.1	37.05
	curvature	[mm]	14.83	13.75	47.07	16.80	146.4	13.02
	fastblock	[mm]	174.3	178.9	180.6	181.2	87.78	173.3
	diffdemons	[mm]	2.802	6.873	48.26	11.23	144.0	1.359
JAC_{disp}	*demons*	[%]	0.00162363	0.0216962	0.00681263	0.0185669	0.0226597	8.34465e-06
	symdemons	[%]	0.0601146	0.127387	0.117009	0.129581	0.078536	0.0621149
	levelset	[%]	0.240135	0.237093	0.209323	0.246824	0.348112	0.234164
	curvature	[%]	0.06316	0.120942	0.123614	0.131058	0.142232	0.0565321
	fastblock	[%]	0.437119	0.432593	0.432847	0.436029	0.284698	0.440297
	diffdemons	[%]	3.54052e-05	2.98023e-05	4.79817e-05	1.20997e-05	1.29938e-05	0

Table 3: Registration results for the synthetic experiments using a pool of algorithms. Displacement field measures.

Measure			simbr-25-10	simbr-55-25	grid-32-4	grid-32-8	uniform-5-32
RMS_{int}	*affine*	[HU]	60.77	73.82	43.97	50.87	52.32
	demons	[HU]	10.57	22.05	7.496	9.133	16.18
	symdemons	[HU]	26.29	43.07	13.94	17.23	13.47
	levelset	[HU]	34.63	36.47	32.97	33.44	31.52
	curvature	[HU]	21.22	28.70	13.47	18.30	13.13
	fastblock	[HU]	60.16	67.61	51.62	54.28	65.38
	diffdemons	[HU]	8.022	24.80	8.817	15.00	23.60
MAD_{int}	*affine*	[HU]	22	76	7	10	10
	demons	[HU]	1	1	1	1	1
	symdemons	[HU]	1	1	1	1	1
	levelset	[HU]	7	8	6	7	6
	curvature	[HU]	1	1	1	1	1
	fastblock	[HU]	17	43	8	12	31
	diffdemons	[HU]	1	2	1	1	3
NMI_{int}	*affine*		0.212	0.131	0.365	0.293	0.301
	demons		0.861	0.781	0.873	0.842	0.753
	symdemons		0.761	0.622	0.835	0.797	0.772
	levelset		0.418	0.393	0.446	0.438	0.454
	curvature		0.765	0.684	0.801	0.753	0.761
	fastblock		0.231	0.212	0.318	0.284	0.195
	diffdemons		0.860	0.676	0.796	0.721	0.599
$EDGE_{int}$	*affine*		0.199	0.192	0.173	0.184	0.180
	demons		0.012	0.038	0.012	0.016	0.037
	symdemons		0.016	0.035	0.018	0.022	0.033
	levelset		0.125	0.123	0.122	0.122	0.116
	curvature		0.014	0.019	0.019	0.021	0.019
	fastblock		0.132	0.150	0.120	0.130	0.150
	diffdemons		0.010	0.052	0.019	0.030	0.068
MAX_{int}	*affine*	[HU]	835	923	361	571	661
	demons	[HU]	5	11	5	7	13
	symdemons	[HU]	8	21	7	10	14
	levelset	[HU]	98	105	92	94	81
	curvature	[HU]	15	22	16	17	20
	fastblock	[HU]	994	998	923	928	1020
	diffdemons	[HU]	8	34	15	29	61

Table 4: Registration results for the analytically defined synthetic experiments. Intensity difference measures.

Measure			demons	symdemons	levelset	curvature	fastblock	diffdemons
RMS_{int}	*affine*	[HU]	62.16	61.04	61.20	61.09	72.07	62.15
	demons	[HU]	11.06	23.92	40.99	21.78	51.01	6.429
	symdemons	[HU]	16.34	15.99	30.40	13.47	41.33	13.35
	levelset	[HU]	32.77	32.28	29.20	32.27	37.83	33.41
	curvature	[HU]	25.31	25.49	38.13	16.73	40.68	17.69
	fastblock	[HU]	59.17	59.38	62.71	59.37	47.75	58.24
	diffdemons	[HU]	28.81	36.06	47.16	33.59	51.80	19.23
MAD_{int}	*affine*	[HU]	27	24	26	24	65	27
	demons	[HU]	1	2	7	1	5	1
	symdemons	[HU]	1	1	4	1	2	1
	levelset	[HU]	9	8	7	8	13	9
	curvature	[HU]	2	4	7	2	4	1
	fastblock	[HU]	23	23	30	22	12	21
	diffdemons	[HU]	4	6	12	5	11	1
NMI_{int}	*affine*		0.204	0.215	0.208	0.212	0.126	0.201
	demons		0.789	0.667	0.439	0.700	0.382	0.841
	symdemons		0.749	0.730	0.522	0.753	0.494	0.800
	levelset		0.471	0.479	0.519	0.462	0.407	0.458
	curvature		0.638	0.606	0.463	0.714	0.454	0.806
	fastblock		0.201	0.188	0.160	0.189	0.318	0.211
	diffdemons		0.609	0.519	0.389	0.537	0.350	0.740
$EDGE_{int}$	*affine*		0.181	0.185	0.187	0.184	0.190	0.185
	demons		0.045	0.088	0.138	0.083	0.102	0.041
	symdemons		0.055	0.073	0.124	0.074	0.084	0.049
	levelset		0.133	0.143	0.135	0.139	0.123	0.136
	curvature		0.068	0.094	0.137	0.079	0.094	0.052
	fastblock		0.164	0.172	0.182	0.167	0.136	0.161
	diffdemons		0.086	0.116	0.150	0.110	0.126	0.056
MAX_{int}	*affine*	[HU]	1004	1007	995	1007	1060	999
	demons	[HU]	13	30	128	22	843	4
	symdemons	[HU]	16	21	88	16	204	6
	levelset	[HU]	87	85	73	85	106	91
	curvature	[HU]	49	60	137	27	245	13
	fastblock	[HU]	1057	1056	1055	1056	817	1051
	diffdemons	[HU]	86	104	202	89	661	38

Table 5: Registration results for the synthetic experiments using a pool of algorithms. Intensity difference measures.

References

[1] John Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[2] G. E. Christensen, X. Geng, J. G. Kuhl, J. Bruss, T. J. Grabowski, I. A. Pirwani, M. W. Vannier, J. S. Allen, and H. Damasio. Introduction to the Non-rigid Image Registration Evaluation Project (NIREP). In *Proc Workshop on Biomedical Image Registration*, volume LNCS 4057, pages 128–135. Springer Verlag, 2006.

[3] Insight Software Consortium. MIDAS - Multi Format Image and Data Assimilation System. http://www.insight-journal.org/dspace/community-list/, 2007.

[4] W. R. Crum, L. D. Griffin, D. L. G. Hill, and D. J. Hawkes. Zen and the Art of Medical Image Registration: Correspondence, Homology, and Quality. *NeuroImage*, 20(3):1425–1437, 2003.

[5] B. Fischer and J. Modersitzki. Curvature based image registration. *Journal of Mathematical Imaging and Vision*, 18(1):81–85, 2003.

[6] G. Gerig, M. Jomier, and M. Chakos. Valmet: A new validation tool for assessing and improving 3D object segmentation. In *Proc Intern Conf on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume LNCS 2208, pages 516–528, 2001.

[7] T. Glatard, X. Pennec, and J. Montagnat. Performance Evaluation of Grid-Enabled Registration Algorithms Using Bronze-Standards. In R. Larsen, M. Nielsen, and J. Sporring, editors, *Proc Intern Conf on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 4191 of *LNCS*, Copenhagen, Denmark, 2006. Springer.

[8] P. Hellier, C. Barillot, I. Corouge, B. Gibaud, G. Le Goualher, D. L. Collins, A. Evans, G. Malandain, N. Ayache, G. E. Christensen, and H. J. Johnson. Retrospective Evaluation of Intersubject Brain Registration. *IEEE Transactions on Medical Imaging*, 22(9):1120–1130, 2003.

[9] ITK. Insight Software Consortium Segmentation and Registration Toolkit. http://www.itk.org, 2006.

[10] J. Modersitzki. *Numerical Methods for Image Registration*. Oxford University Press, 2004.

[11] National Cancer Institute (NCI). National Cancer Imaging Archives. https://imaging.nci.nih.gov/ncia/faces/baseDef.tiles, 2007.

[12] NLM. National Library of Medicine Image Data Collection Project. http://nova.nlm.nih.gov/Mayo/, 2006.

[13] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes. Nonrigid Registration Using Free-Form Deformations: Application to Breast MR Images. *IEEE Transactions on Medical Imaging*, 18(8):712–721, August 1999.

[14] D. Scharstein and R. Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1/2/3):7–42, 2002.

[15] J. A. Schnabel, C. Tanner, A. D. Castellano-Smith, A. Degenhard, M. O. Leach, D. R. Hose, D. L. G. Hill, and D. J. Hawkes. Validation of Nonrigid Image Registration Using Finite-Element Methods: Application to Breast MR Images. *IEEE Transactions on Medical Imaging*, 22(2):238–247, 2003.

[16] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. In *Proc Conf on Computer Vision and Pattern Recognition (CVPR)*, pages 519–526, 2006.

[17] D. Skerl, B. Likar, and F. Pernus. Evaluation of Similarity Measures for Non-Rigid Registration. In J. P. W. Pluim, B. Likar, and F. A. Gerritsen, editors, *Workshop on Biomedical Image Registration*, volume LNCS 4057, pages 160–168. Springer Verlag, 2006.

[18] J.-P. Thirion. Image matching as a diffusion process: An analogy with Maxwell's demons. *Medical Image Analysis*, 2(3):243–260, 1998.

[19] M. Urschler, C. Zach, H. Ditt, and H. Bischof. Automatic Point Landmark Matching for Regularizing Nonlinear Intensity Registration: Application to Thoracic CT Images. In R. Larsen, M. Nielsen, and J. Sporring, editors, *Proc Intern Conf on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 4191 of *LNCS*, pages 710–717, Copenhagen, Denmark, 2006. Springer.

[20] B. C. Vemuri, J. Ye, Y. Chen, and C. M. Leonard. Image registration via level-set motion: Applications to atlas-based segmentation. *Medical Image Analysis*, 7(1):1–20, 2003.

[21] H. Wang, L. Dong, J. O'Daniel, R. Mohan, A. S. Garden, K. K. Ang, D. A. Kuban, M. Bonnen, J. Y. Chang, and R. Cheung. Validation of an accelerated 'demons' algorithm for deformable image registration in radiation therapy. *Phys. Med. Biol.*, 50(12):2887–2905, 2005.

[22] J. West, J. M. Fitzpatrick, M. Y. Wang, B. M. Dawant, C. R. Maurer Jr., R. M. Kessler, and R. J. Maciunas. Retrospective Intermodality Registration Techniques for Images of the Head: Surface-Based Versus Volume-Based. *IEEE Transactions on Medical Imaging*, 18(2):144–150, February 1999.

[23] R. P. Woods, S. T. Grafton, J. D. G. Watson, N. L. Sicotte, and J. C. Mazziotta. Automated Image Registration: II. Intersubject Validation of Linear and Nonlinear Models. *Journal of Computer Assisted Tomography*, 22(1):153–165, 1998.

[24] Z. Zhang, Y. Jiang, and H. Tsui. Consistent multi-modal non-rigid registration based on a variational approach. *Pattern Recognition Letters*, 27:715–725, 2006.

A White Matter Stochastic Tractography System

Release 1.00

Tri M. Ngo[1], Carl-Fredrik Westin[2] and Polina Golland[3]

July 17, 2007

[1]MIT, Cambridge
[2]Harvard, Cambridge
[3]MIT, Cambridge

Abstract

White matter tractography enables studies of fiber bundle characteristics. Stochastic tractography facilitates these investigations by providing a measure of confidence regarding the inferred fiber bundles. This article presents a multithreaded ITK stochastic tractography filter that will enable novel studies of fiber tract abnormalities. Additionally, we provide an easy to use command line interface for the filter that is compatible with the 3D Slicer visualization environment.

Contents

1 Introduction **1**

2 Algorithm Overview **2**

3 Implementation Details **4**

4 User's Guide **7**
4.1 ITK Stochastic Tractography Filter 7
4.2 Command Line Module Interface 9
4.3 3D Slicer Interface 10

5 Conclusion **10**

1 Introduction

DTI data sets provide information about the diffusion of water at each voxel, or volume element, in the form of diffusion tensors. This information can be used to extract the underlying fiber tracks which produced the observations. This technique is known as DTI Tractography.

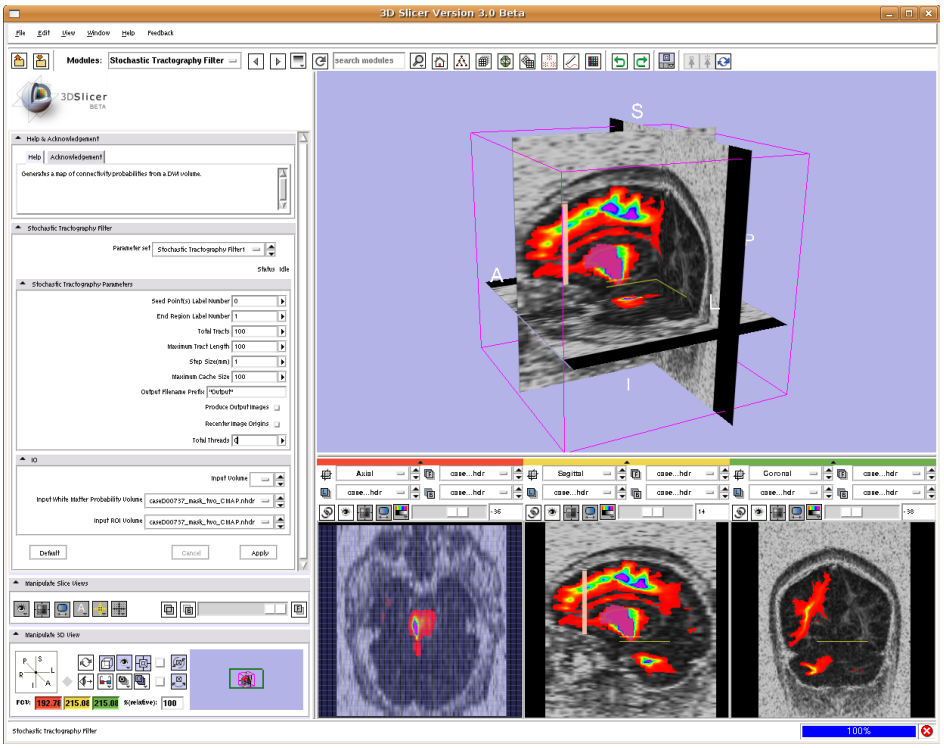


Figure 1: 3D Slicer environment displaying the Stochastic Tractography Module interface and a connectivity map overlaid on a fractional anisotropy image.

One possible method of performing tractography is to generate tracts which follow the direction of maximal water diffusion of the voxels they pass through [9, 1]. This method is sometimes called streamline tractography. Unfortunately streamline tractography does not provide information about the uncertainty of the generated tracks caused by noise or insufficient spatial resolution. Bayesian white matter tractography methods try to address this problem by performing tractography under a probabilistic framework. Several formulations of stochastic/probabilistic tractography have been suggested [3, 2, 10, 7, 8], however tools which enable widespread adoption of stochastic tractography in clinical studies are not currently available.

This paper presents an easy to use, multithreaded ITK filter for performing stochastic white matter tractography based on the algorithm described by Friman et al. [5, 6]. Additionally, we present a 3D Slicer [4] graphical user interface module which uses the stochastic tractography filter, further increasing its ease of use and further encouraging its application in clinical research (figure 1).

2 Algorithm Overview

The stochastic tractography algorithm implemented in this paper is based on Friman's [6] approach with some modifications to the stopping criteria. This paper provides a brief overview of the algorithm. Please refer to Friman's [6] original paper for a more complete explanation of the theory. Figure 2 provides a flow chart demonstrating key steps in the algorithm.

A fiber tract is modeled as a sequence of unit vectors. The orientation of these unit vectors is determined by sampling a posterior fiber orientation distribution which is dependent on the local diffusion data as well as the orientation of the unit vector in the previous step. The posterior distribution is a normalized product

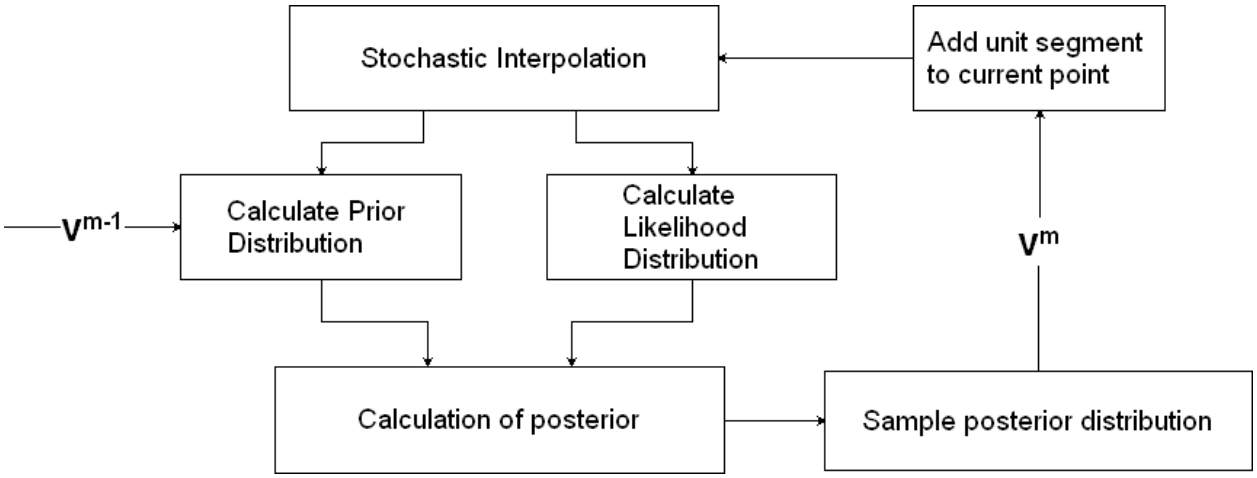


Figure 2: A flow chart demonstrating key steps in the stochastic tractography algorithm

of the prior likelihood of the fiber orientation and the likelihood of that fiber orientation given the local diffusion data.

Friman uses a subset of the tensor model which is called a constrained diffusion model. In this model, the two smallest eigenvectors of diffusion tensor are equal, constraining the shape of the diffusion tensor to be linearly anisotropic. The constrained model rules out the possibility of nonlinear, or non-cylindrical anisotropic diffusion distributions. Deviations from linearly anisotropic diffusion distributions are captured as uncertainty in the fiber orientation. The constrained model is combined with a Gaussian DWI noise model to obtain a fiber orientation likelihood function. The parameters for the constrained model are derived from a weighted least squares estimation of the parameters for the log tensor model.

The orientation of each vector depends only on the previous vector. This dependency is formulated in the prior on the fiber orientation. Prior knowledge about the regularity of the fiber tract can encoded in this prior probability. The prior also serves to prevent the fiber from backtracking, since the likelihood distribution alone is axially symmetric.

Friman's approach is a Bayesian inference algorithm similar to Behrens's but with some important optimizations [6]. In contrast with Behrens's two-compartment observation model, the constrained model used by Friman is derived from the thoroughly studied tensor model of diffusion. The advantage of using the constrained model is that it is relatively easy to estimate the parameters for the model. The parameters for the constrained model are obtained after the tensor model has been fit to the diffusion data. Since the parameters for the tensor model are easily obtained through many computationally efficient ways, the constrained model's parameters are likewise easy to obtain. The constrained model can be fit to every voxel within a matter of seconds whereas Behrens's model takes a couple of hours [6]. Additionally Friman avoids using MCMC techniques by assuming that parameters other than the principle diffusion direction take on their ML estimates with certainty within each voxel. Friman demonstrates that eliminating this source of uncertainty has little effect on the resulting posterior fiber orientation distribution.

In Friman's paper, tracking is terminated when an encountered voxel's diffusion distribution anisotropy is below some threshold. However, since the stochastic tractography algorithm takes into account this uncertainty with an increase in the spatial variance of sampled fibers, this termination criterion seems arbitrary and contradictory with the goals of stochastic tractography, which is to enable sampling of white matter fiber tracts in regions of uncertainty. Thus we replace this termination criterion with one which terminates tractography based on the posterior probability that a fiber tract exists within the current voxel. This probability

is equivalent to the probability that the current voxel is in white matter. A map of white matter posterior probability can be obtained through a soft segmentation of an anatomical image of the brain co-registered with the DWI data. Alternatively, the soft segmentation can also be performed on the B0 image of the DWI data set, thus eliminating the need for additional data. Although this alternative termination criteria may seem equivalent to using an anisotropy threshold criterion, because white matter generally has higher anisotropy than gray matter, the alternative method does not exclude regions of white matter which have low anisotropy due to crossing fibers. This criteria should enable the algorithm to detect more tracts in white matter than under the anisotropy termination criteria.

3 Implementation Details

The stochastic tractography algorithm is a Monte Carlo algorithm which samples the high dimensional parameter space of fiber tracts. This is a large space because fiber tracts are characterized by a sequence of segment orientations, each of which is a separate parameter describing the fiber tract. As such, it may take many samples to accurately approximate the posterior distribution of these parameters. However, since these samples are IID, they can be generated in parallel. Implementing the filter in a multithreaded fashion enables parallel sampling of the tract distribution.

ITK provides a framework for implementing multithreaded algorithms. The ITK multithreading framework assumes that the output region can be divided into disjoint sections with each thread working exclusively on their own section of the output image. This design prevents threads from simultaneously writing to the same memory region, which may cause unexpected results. However, since the stochastic tractography algorithm generates tracts that may span the entire output image, dividing the output region into disjoint sections is not possible. Additionally, in order to obtain statistics on these tracts, we need to output the generated tracts as well as the resultant connectivity image. Thus the existing ITK framework for implementing multithreaded filters is not very useful for our stochastic tractography filter. Fortunately ITK also provides basic multithreading functions which allowed us to create a custom multithreaded design that is still within the ITK framework.

Each thread of stochastic tractography filter is an instance of the stochastic tractography algorithm. The block diagram in figure 3 demonstrates graphically the architecture of the ITK stochastic tractography filter. Every thread allocates it own independent memory for the tract that it is currently generating. Once the tract has terminated, the thread stores a memory pointer to the completed tract in a tract pointer container that is shared among all threads. The tract pointer container is protected by a mutex, which serializes write operations so that only one thread can store its completed tract in the vector at a time. Once the filter has generated enough samples, the tracts can be transferred to an output image to create a connectivity map. Additionally other statistics can be computed on the tracts. In essence, we divide the process into two sections, a multithreaded portion that samples the tracts and a single threaded portion which accumulates the tracts and calculates relevant statistics on them.

The most computationally expensive part of the algorithm is the calculation of the likelihood distribution. The algorithm must compute probabilities for 2,562 possible fiber orientations in a voxel. Fortunately, this likelihood distribution is a deterministic function of the diffusion observations within that voxel. The filter runs much faster, at the cost of additional memory, by caching the generated likelihood distribution for later access. Caching is effective because in highly anisotropic regions of the brain, the sampled tracts are expected to be dense causing many of the sampled tracts to visit the same voxels many times.

The cache is implemented as an image whose voxels are re-sizable arrays. ITK's optimized pixel access capabilities enable quick access to the likelihood distribution associated with any voxel in the image. On

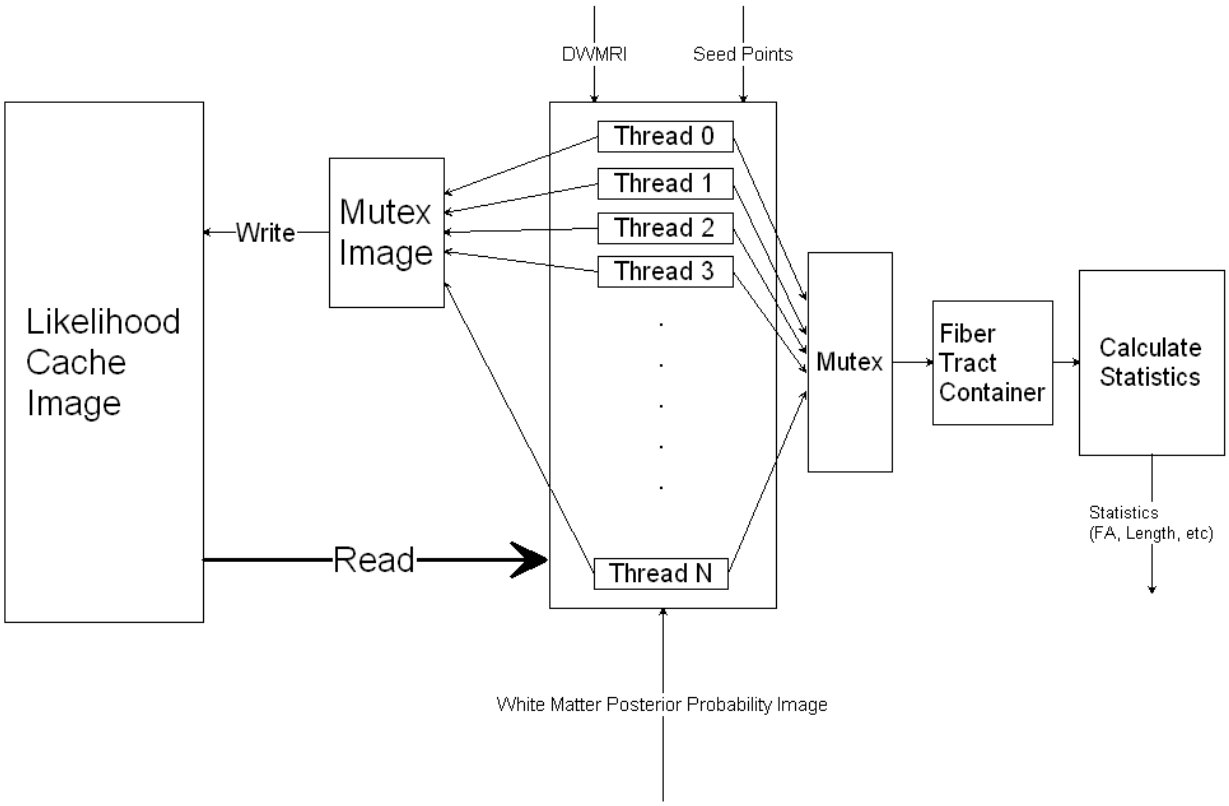


Figure 3: A block diagram of the filter showing its shared likelihood cache and multithreaded architecture.

creation, every voxel in the likelihood cache image is initiated to a zero length array. Whenever the algorithm encounters a voxel, it first checks to see if the likelihood cache contains this voxel by testing if associated array is zero length. If the voxel has never been visited, the associated array is resized and the computation of the likelihood distribution associated with this voxel is stored inside the newly resized array.

Using a shared likelihood cache between multiple concurrent threads creates additional complexities. Simultaneous writes to the cache would cause unexpected behavior. Additionally there is the possibility of one thread reading an incomplete cache entry while another thread is trying to write it. One possible solution is to ensure that only one thread can read or write to the likelihood cache at a time. This is easily implemented by serializing access to the likelihood cache using a mutex. A mutex serves as a lock on data. A thread will wait to obtain a lock on the data before it proceeds to the next section of code. Inside this section, which is called the critical section, the thread holds the lock ensuring exclusive access to the otherwise shared data. All other threads must wait and idle while the thread which owns the lock finishes it operations. Since threads must access the likelihood cache very often, this results in a situation where many threads are waiting for other threads to finish accessing the likelihood cache. The serialized access to the likelihood cache creates a bottleneck, which in the worse case would result in performance that is only marginally better than a single threaded version of the filter.

Access collisions to the likelihood cache can be reduced if we increase the granularity of the lock. Instead of using one large lock for the entire likelihood cache image, we use a lock for each voxel. The probability of two threads accessing the same voxel simultaneously is much less than the probability of two threads accessing any part of the likelihood cache. These per voxel locks are conveniently constructed using an ITK

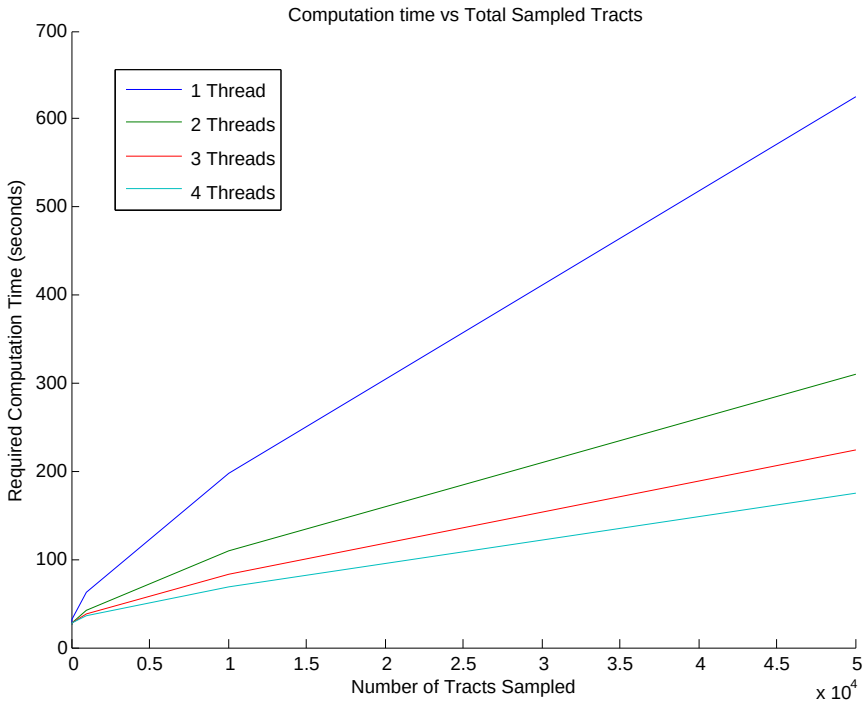


Figure 4: A graph displaying the amount of time needed to sample a number of tracts. Each line represents the algorithm's performance using different numbers of threads. This test was run on a 4 processor machine.

image whose voxel data type is a mutex. Similar to the likelihood cache, this collection of mutexes is indexed by coordinates which correspond to the coordinates of the voxels in the DWI input data. Again, access to the mutex image is fast due to ITK's optimized access operators for data types indexed by coordinates. The only cost to using this high resolution mutex image is the additional memory required to store pixel mutexes. However this cost is small since a mutex is essentially a Boolean variable. The mutex image allows different voxels in the likelihood cache image to be updated simultaneously, increasing the rate that the likelihood cache is filled. The advantage of using a mutex image is most evident when tracking in highly isotropic regions, where collisions are very unlikely to occur, since the sampled paths are very dispersed. Even on uni-processor systems, using multiple threads may improve performance since the rate of encountering an unvisited voxel may be higher, thus filling the likelihood cache faster. Figure 4 demonstrates the required computation time for a given number of tracts under different number of threads.

Additionally, to compute the weighted least squares estimates for the log tensor model parameters, we must first estimate the weights. These weights are found by calculating a least squares estimate of the true intensities of each voxel. The design matrix used in this least squares estimation is a function solely of the magnetic gradient directions and associated b-values, which are the same for every voxel in the image. Since the same design matrix is used for each voxel in the least squares calculation, a common optimization is to orthogonalize the design matrix by computing its QR decomposition. While this operation is computationally expensive, it is performed only once for the entire DWI image. The orthogonalized design matrix reduces the cost of computing the weights for every voxel.

4 User's Guide

4.1 ITK Stochastic Tractography Filter

The Stochastic Tractography Filter is implemented as a multithreaded image filter in ITK under the class name itk::StochasticTractographyFilter. The filter is templated over the DWI and white matter probability map input image types and also on the connectivity map image type. The filter expects the DWI input image type to be an ITK VectorImage Type. The code below demonstrates how to instantiate the Stochastic Tractography Filter.

```
//Define Types
typedef itk::VectorImage< unsigned short int, 3 > DWIVectorImageType;
typedef itk::Image< float, 3 > WMPImageType;
typedef itk::Image< unsigned int, 3 > CImageType;
typedef itk::StochasticTractographyFilter< DWIVectorImageType, WMPImageType,
  CImageType > PTFilterType;
//Allocate Filter
PTFilterType::Pointer ptfilterPtr = PTFilterType::New();
```

The filter's required inputs and parameters must be set before it can be run. Table 1 lists filter methods that should be called to set the required inputs and parameters, and a short description of what each methods expects as arguments.

The code below is a continuation of the demonstration above and shows how to setup the filter's required inputs and parameters. The inputs to these methods are provided by ITK's image readers.

```
ptfilterPtr->SetInput( dwireaderPtr->GetOutput() );
ptfilterPtr->SetWhiteMatterProbabilityImageInput( wmpreader->GetOutput() );
ptfilterPtr->SetbValues(bValuesPtr);
ptfilterPtr->SetGradients( gradientsPtr );
ptfilterPtr->SetMeasurementFrame( measurement_frame );
ptfilterPtr->SetMaxTractLength( maxtractlength );
ptfilterPtr->SetTotalTracts( totaltracts );
ptfilterPtr->SetMaxLikelihoodCacheSize( maxlikelihoodcachesize );
ptfilterPtr->SetSeedIndex( seedindex );
```

The filter can then be run by calling the Update method.

```
ptfilterPtr->Update();
```

For the specified seed voxel, the filter outputs a connectivity map and a container holding all of the sampled tracts used to generate the connectivity map. The container of sampled tracts can be further processed outside of the stochastic tractography filter to obtain various statistic on the sampled tracts. Additional seed voxels can be included in the seed region by changing the seed voxel index and rerunning the filter. The statistics for a multi-voxel seed region can be analyzed by accumulating statistics for all seed voxels within the seed region. These outputs can be accessed by calling the `GetOutput` and `GetOutputTractContainer` methods after calling the `Update` method. The code below continues the example above and demonstrates how to obtain the filter's outputs.

Filter Member Method	Description
SetInput	DWI Image: An ITK VectorImage consisting of a vector of DWI measurements including the baseline b0 measurements, at each voxel.
SetWhiteMatterProbabilityImageInput	White Matter Probability Input: An ITK image whose voxel values range from 0 and 1 representing the posterior probability that the voxel is a white matter.
SetbValues	b-Values: An ITK VectorContainer whose elements are the corresponding b-values for the DWI input image. The b0 measurements must have a 0 b-value.
SetGradients	magnetic gradient directions: An ITK VectorContainer whose elements are 3 dimensional vnl vectors. These vectors should be unit length.
SetMeasurementFrame	DWI Measurement Frame: A 3x3 vnl matrix which transforms the gradient directions to the physical reference frame of the image. For instance multiplying a magnetic gradient direction vector by the Measurement Frame Matrix will take the vector to the RAS reference frame if RAS is the physical frame of the DWI image.
SetMaxTractLength	Maximum Tract Length: A positive integer that sets the maximum length of a sampled tract. This can also be interpreted as the number of segments which comprise the tract when using the default step size of 1 unit in the physical frame of the DWI image.
SetTotalTracts	Total Sampled Tracts: A positive integer that sets the total number tracts to sample from the seed voxel.
SetMaxLikelihoodCachSize	Maximum Likelihood Cache Size(MB) A positive integer that sets the maximum size of the Likelihood Cache in megabytes.
SetSeedIndex	Seed Voxel Index: The discrete index of the seed voxel, in the (IJK) reference frame of the image to start tractography.

Table 1: ITK Stochastic Tractography Filter Required Inputs and Parameters

```
PTFilterType::TractContainerType::Pointer tractcontainer =
  ptfilterPtr->GetOutputTractContainer();
CImageType::Pointer cmap = ptfilterPtr->GetOutput();
```

4.2 Command Line Module Interface

The command line module interface provides an easy to use method of performing common tasks which use the ITK stochastic tractography filter. The command line module takes as input a DWI volume, a white matter probability map and a label map to produce a connectivity probability map and fractional anisotropy and length statistics for a selected seed region in the label map. Currently the command line module is designed to work only with NRRD formatted volumes due to its support of the diffusion measurement frame, but future revisions of the software will extend support to other formats.

The command line module is named `StochasticTractographyFilter`. Calling the executable with the `--help` flag will list all available inputs and options as well as a short description of each item. This section will demonstrate two typical usages of the command line module interface.

Given a DWI volume, an associated white matter probability map and a label map, the command line module can be used to generate an image that provides the probability of connectivity from an ROI in the label map to all other voxels in the DWI. The label map is an integer valued image that segments voxels into different classes or labels.

Let `case24` be the name of the subject we are interested in analyzing. The directory `case24` contains all relevant files for that subject. It will also hold the output files generated by the command line module. Before executing the command line module interface, a possible list of files in the `case24` directory may include:

```
case24_DWI.nhdr (DWI NRRD header)
case24_DWI.raw (DWI NRRD data)
case24_whitematterPB.nhdr (White Matter Probability Map NRRD header)
case24_whitematterPB.raw (White Matter Probability Map NRRD data)
case24_labelmap.nhdr (Label Map NRRD data)
case24_labelmap.raw (Label Map NRRD data)
```

Assuming that the starting ROI is labeled 15 inside the labelmap, the stochastic tractography filter can be run by executing the command below within the `case24` directory: To run the command line module, execute the command:

```
StochasticTractographyFilter -c 6500 -m 500 -t 200 -e 15 -l 15
-r -o case24_RUN0 case24_DWI.nhdr case24_whitematterPB.nhdr
case24_labelmap.nhdr
```

After the command completes, the `case24` directory will contain the following additional files:

```
case24_RUN0_CMAP.nhdr (Connectivity Map NRRD header)
case24_RUN0_CMAP.raw (Connectivity Map NRRD data)
case24_RUN0_TENSOR.nhdr (Tensor image NRRD header)
case24_RUN0_TENSOR.raw (Tensor image NRRD data)
case24_RUN0_COND.nhdr (Conditioned Connectivity Map NRRD header)
```

```
case24_RUN0_COND.raw (Conditioned Connectivity Map NRRD data)
case24_RUN0_CONDFAValues.txt (Conditioned Tract-Averaged FA values)
case24_RUN0_CONDLENGTHValues.txt (Conditioned Tract Length values)
```

The conditioned connectivity map is identical to the normal connectivity map when the start label and end labels are the same. However, if the label map contains ROIs designated by two labels, the conditioned connectivity map will be generated using only fibers which start in the start ROI and also pass through the second ROI. Assuming the second ROI is labeled 2 in the labelmap, the following command will isolate tracts which start in the start ROI and pass through the end ROI:

```
StochasticTractographyFilter -c 6500 -m 500 -t 200 -e 2 -l 15
-r -o case24_RUN0 case24_DWI.nhdr case24_whitematterPB.nhdr
case24_labelmap.nhdr
```

Now the conditioned connectivity maps and statistics are generated using only tracts which fulfill the condition of passing through both ROIs. This feature allows us to analyze the particular bundle of tracts which connect two regions.

4.3 3D Slicer Interface

To encourage the algorithm's adoption in clinical studies, we created an interactive GUI module for the 3D Slicer medical image visualization program was created which interfaces with the ITK stochastic tractography filter.

The module was implemented using the command line module interface provided by the 3D Slicer environment. This interface greatly eased the adaption of the command line interface into a graphical interface that could be included with 3D Slicer. The command line interface and the graphical interfaces are both completely described using an XML file. This XML file description is then parsed by a program provided by 3D Slicer which generates code that can be included with the command line interface to create what 3D Slicer refers to as a command line module. Command line modules can be run independently of 3D Slicer but can also be incorporated in to the 3D Slicer graphical interface. This enables the stochastic tractography algorithm to function as an easy to use extension in the 3D Slicer program as well as a stand alone program suitable for processing a large numbers of data sets non-interactively.

5 Conclusion

In this paper we have implemented a stochastic tractography system that can be used to analyze white matter architecture from DTI images. The multithreaded stochastic tractography ITK filter allows parallel sampling of fiber tracts, reducing computation time on multi-processor systems. Finally, we have created easy to use command line and 3D Slicer graphical interfaces to facilitate algorithm's use in clinical studies.

References

[1] PJ Basser and S. Pajevic. In vivo fiber tractography using dt-mri data. *Magn Reson Med*, 44:625–632, 2000. 1

[2] T.E.J Behrens, M.W. Woolrich, M. Jenkinson, H. Johansen-Berg, R. G. Nunes, S. Clare, P.M. Matthews, J.M Brady, and S.M. Smith. Characterization and propagation of uncertainty in diffusion-weighted mr imaging. *Magnetic Resonance in Medicine*, 50:1077–1088, 2003. 1

[3] M. Björnemo, A. Brun, R. Kikinis, and C.-F. Westin. Regularized stochastic white matter tractography using diffusion tensor MRI. In *Fifth International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'02)*, pages 435–442, Tokyo, Japan, 2002. 1

[4] Brigham and Women's Hospital. 3d slicer medical visualization and processing environment for research. *http://www.slicer.org/*. 1

[5] O. Friman and C.-F. Westin. Uncertainty in fiber tractography. In *Eighth International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'05)*, Lecture Notes in Computer Science 3749, pages 107–114, Palm Springs, CA, USA, October 2005. 1

[6] Ola Friman, Gunnar Farneback, and Carl-Fredrik Westin. A Bayesian approach for stochastic white matter tractography. *TMI*, 25(8):965–978, 2006. 1, 2

[7] Derek K. Jones and Carlo Pierpaoli. Confidence mapping in diffusion tensor magnetic resonance imaging tractography using a bootstrap approach. *Magnetic Resonance in Medicine*, (5):1143–1149, 2005. 1

[8] M. Lazar and A. Alexander. Bootstrap white matter tractography (boot-trac). *NeuroImage*, 24:524–532, 2005. 1

[9] S. Mori, B. Crain, V.P. Chacko, and PCM van Zijl. Three dimensional tracking of axonal projections in the brain by magnetic resonance imaging. *Ann Neurol*, 45:265–269, 1999. 1

[10] D.S. Tuch, M.R. Wiegell, T.G. Reese, J.W. Belliveau, and V.J. Weeden. Measuring cortico-cortical connectivity matrices with diffusion spectrum imaging. In *Proc. of the International Society for Magnetic Resonance Medicine*, page 502, Glasgow, Scotland, 2001. 1

Non-rigid Groupwise Registration using B-Spline Deformation Model

Release 0.00

Serdar K. Balci[1], Polina Golland[1] and William M. Wells[2]

July 17, 2007

[1]MIT CSAIL, Cambridge, MA, USA
[2]Brigham & Women's Hospital, Harvard Medical School, Cambridge, MA, USA

Abstract

In this work, we extend a previously demonstrated entropy based groupwise registration method to include a non-rigid deformation model based on B-splines. We describe an open source implementation of the groupwise registration algorithm using the Insight Toolkit ITK www.itk.org. We provide the source code, parameters, input and output data that we used for validation.

We describe an efficient implementation of the algorithm by using a stochastic optimization scheme embedded in a multi-resolution setting. The objective function is optimized using gradient descent algorithm combined with line search for the step size. The derivative of the objective function is evaluated efficiently by computing Jacobian of B-spline deformation field locally.

We demonstrate the algorithm in application to different imaging modalities including proton density, FA, T1 and T2 MR images. We validate the algorithm on synthetic datasets varying from 2 to 30 images by recovering randomly applied affine and B-spline transforms.

Contents

1 Introduction **2**

2 Algorithm **2**
2.1 Stack Entropy Cost Function . . . 3
2.2 Free-Form B-spline Deformation Model . . . 3

3 Implementation **4**
3.1 Metric . . . 5
3.2 Optimization . . . 5
Efficient B-spline Implementation . . . 5
3.3 Component Interaction . . . 6
3.4 Summary of New Classes . . . 6

4 Results **7**

5 Running Tests **8**

6 Conclusion **8**

A Test Results **10**

B Registration Example **11**

C Using Command Line Modules **15**
C.1 Registration Parameters 15

1 Introduction

Groupwise registration is an important tool in medical image analysis for establishing anatomical correspondences among subjects in a population [10, 2]. A groupwise registration scheme can be used to characterize anatomical shape differences within and across populations [9].

Miller et al. [7] introduce an efficient groupwise registration method in which they consider sum of univariate entropies along pixel stacks as a joint alignment criterion. They provide a template-free approach to groupwise registration by simultaneously driving all subjects to the common tendency of population. Zollei et al. [11] successfully applied this method to groupwise registration of medical images using affine transforms and used stochastic gradient descent algorithm for optimization.

In our work, we extend Miller et al.'s [7] method to include free-form deformations based on B-splines in 3D. We describe an efficient implementation using stochastic optimization in a multi-resolution setting. To optimize the objective function we use gradient descent algorithm combined with line search for the step size. We validate the algorithm on synthetic datasets by recovering randomly applied affine and B-spline transforms. All experiments that we present are easily reproducable as we provide our source code, parameters, input data and an automated procedure to obtain the results.

This paper is organized as follows. In the next section, we describe the groupwise registration algorithm by describing the stack entropy cost function and B-spline based deformation model. In Section 3, we present our registration framework and describe implementation of the metric, deformation model and optimization. In Section 4, we demonstrate the groupwise registration algorithm in application to different sets of input data. In Section 5, we describe an automated procedure to reproduce the test results. In Appendix A, we give all test results and in Appendix B we go over a groupwise registration example. In Appendix C we discuss important registration parameters.

2 Algorithm

Given a set of images $\{I_1, \ldots, I_N\}$, each described by intensity values $I_n(\mathbf{x}_n)$ across the image space $\mathbf{x}_n \in \mathcal{X}_n$, we can define a common reference frame $\mathbf{x}_R \in \mathcal{X}_R$ and a set of transforms $\mathcal{T}$ which maps points from the reference frame to points in the image space

$$\mathcal{T} = \{T_n : \mathbf{x}_n = T_n(\mathbf{x}_R), n = 1, \ldots, N\} \tag{1}$$

In the following section, we describe the stack entropy cost function as introduced by Miller et al. [7] and applied to groupwise registration of medical images using affine transforms by Zollei et al. [11]. We

continue by describing the transformation model and extend Miller et al.'s [7] method to include free-form deformations based on B-splines.

2.1 Stack Entropy Cost Function

In order to align all subjects in the population, we consider sum of univariate entropies as a joint registration criterion. We let $x_v \in \mathcal{X}_R$ be a sample from a spatial location in the reference frame and $H(I(T(x_v)))$ be the univariate entropy of the stack of pixel intensities $\{I_1(T_1(x_v)),\ldots,I_N(T_N(x_v))\}$ at spatial location x_v. The objective function for the groupwise registration can be given as follows

$$f = \sum_{v=1}^{V} H(I(T(x_v))). \tag{2}$$

We employ a Parzen window based density estimation scheme to estimate univariate entropies [3]:

$$f = -\sum_{v=1}^{V} \frac{1}{N} \sum_{i=1}^{N} \log \frac{1}{N} \sum_{j=1}^{N} G_\sigma(d_{ij}(x_v)) \tag{3}$$

where $d_{ij}(x) = I_i(T_i(x)) - I_j(T_j(x))$ is the distance between intensity values of a pair of images evaluated at a point in the reference frame and G_σ is a Gaussian kernel with variance σ^2. Parzen window density estimator allows us to obtain analytical expressions for the derivative of the objective function with respect to the transformation parameters

$$\frac{\partial f}{\partial T_n} = \sum_{v=1}^{V} \frac{1}{\sigma^2 N} \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{G_\sigma(d_{ij}(x_v)) d_{ij}(x_v)}{\sum_{k=1}^{N} G_\sigma(d_{ik}(x_v))} \frac{\partial d_{ij}(x_v)}{\partial T_n}. \tag{4}$$

2.2 Free-Form B-spline Deformation Model

For the nonrigid deformation model, we define a combined transformation consisting of a global and a local component

$$T(\mathbf{x}) = T_{local}(T_{global}(\mathbf{x})) \tag{5}$$

where T_{global} is a twelve parameter affine transform and T_{local} is a deformation model based on B-splines.

Following Rueckert et al.'s formulation [8], we denote $\Phi_{i,j,k}$ an $n_x \times n_y \times n_z$ grid of control points with uniform spacing. The free form deformation can be written as a 3-D tensor product of 1-D cubic B-splines.

$$T_{local}(\mathbf{x}) = \mathbf{x} + \sum_{l=0}^{3} \sum_{m=0}^{3} \sum_{n=0}^{3} B_l(u) B_m(v) B_n(w) \Phi_{i+l,j+m,k+n} \tag{6}$$

where $\mathbf{x} = (x,y,z)$, $i = \lfloor x/n_x \rfloor - 1$, $j = \lfloor y/n_y \rfloor - 1$, $k = \lfloor z/n_z \rfloor - 1$, $u = x/n_x - \lfloor x/n_x \rfloor$, $v = y/n_y - \lfloor y/n_y \rfloor$, $w = z/n_z - \lfloor z/n_z \rfloor$ and where B_l is l'th cubic B-spline basis function. Using the same expressions for u,v and w as above, the derivative of the deformation field with respect to B-spline coefficients can be given by

$$\frac{\partial T_{local}(x,y,z)}{\partial \Phi_{i,j,k}} = B_l(u) B_m(v) B_n(w) \tag{7}$$

where $l = i - \lfloor x/n_x \rfloor + 1$, $m = j - \lfloor y/n_y \rfloor + 1$ and $n = k - \lfloor z/n_z \rfloor + 1$. We consider $B_l(u) = 0$ for $l < 0$ and $l > 3$. The derivative terms are nonzero only in the neighborhood of a given point. Therefore, optimization of the objective function using gradient descent can be implemented efficiently.

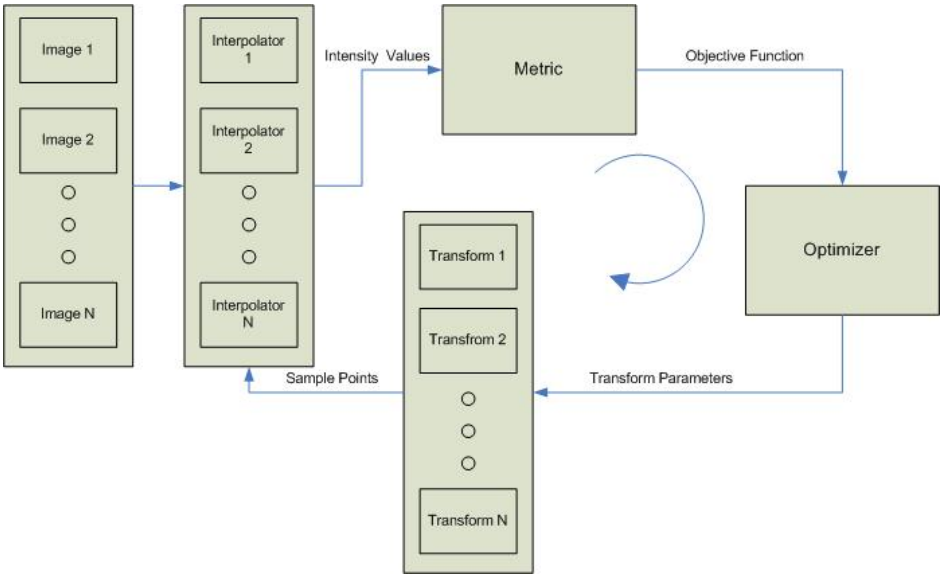


Figure 1: The basic components of the registration framework are a metric, an optimizer and arrays of input images, transforms and interpolators.

As none of the images are chosen as an anatomical reference, it is necessary to add a geometric constraint to define the reference coordinate frame. Similar to Bhatia et al. [1], we define the reference frame by constraining the average deformation to be the identity transform:

$$\frac{1}{N}\sum_{n=1}^{N} T_n(\mathbf{x}) = \mathbf{x} \tag{8}$$

This constraint assures that the reference frame lies in the center of the population. In the case of B-splines, the constraint can be satisfied by constraining the sum of B-spline coefficients across images to be zero. In the gradient descent optimization scheme, the constraint can be forced by subtracting the mean from each update vector [1].

3 Implementation

Figure 1 displays the basic components of a groupwise registration framework and the interactions between them. The framework is an extension of ITK's pairwise registration framework described in ITK software guide [4].

The basic input to the registration process is an array of images. In our setting all images are considered to be moving images. None of the images are chosen as a fixed image to avoid anatomical bias to the chosen reference. Instead, the reference frame is defined by constraining the sum of all transforms to be the identity transform as shown in equation 8. The constraint is forced inside the metric classes by subtracting the mean from transform parameter updates.

Every image is associated with an interpolator and a transform. Each transform map points from the common coordinate frame to the corresponding space of the input image. Interpolators allow to evaluate intensity values at non-grid locations. The metric computes an objective function which measures how well the

group of images are registered. The optimizer component optimizes this objective function by searching over the parameters of the transforms.

In following sections, we introduce registration components and the interactions between them.

3.1 Metric

We extended `itk::ImageToImageMetric` base class to `itk::MultiImageMetric` to compute an objective function on a group of images. The base class is multi-threaded and provides generic methods to be used by the optimizer.

`itk::UnivariateEntropyMultiImageMetric` derive from this class and compute the sum of univariate entropies as given in equation 3. This metric uses a stochastic subsampling scheme to efficiently evaluate the objective function [6]. The function `SampleFixedImageDomain` takes random samples from the image domain in a multi-threaded fashion. In each iteration of the registration process, random samples are taken and the objective function is evaluated only on this sample set. The metric makes use of the B-spline optimization for Jacobian computations if a B-spline transform is connected to it.

`itk::VarianceMultiImageMetric` computes sum of variances along pixel stacks. This class matches each image to a mean template image using sum of squared errors as an objective function similar to Joshi et al. [5]. This class also follows a stochastic subsampling procedure and performs multi-threaded execution.

3.2 Optimization

We provide an efficient optimization scheme by using line search with the gradient descent algorithm. For each iteration, the class `itk::GradientDescentLineSearchOptimizer` performs a line search to determine the step size of the gradient descent algorithm. Empirically we observed that this optimization method is more robust to optimization parameters and increases the convergence rate.

As in every iterative search algorithm, local minima pose a significant problem. To avoid local minima we use a multi-resolution optimization scheme. The registration is first performed at a coarse scale by downsampling the input. Results from coarser scales are used to initialize optimization at finer scales.

Efficient B-spline Implementation

To compute B-spline deformation fields efficiently we modified `itk::BSplineDeformableTransform` to take into account the locality of B-splines for Jacobian computations. The new class `itk::BSplineDeformableTransformOpt` computes the Jacobian field locally by using the function `GetJacobian( point, indexes, weights )`. This function returns the nonzero `indexes` of the Jacobian field and the `weights` associated with them. The metric class can make use of this optimization in `GetValueAndDerivative()` method with an `if` statement for B-splines. `itk::MultiImageMetric` has a member `bool m_BsplineDefined`, which is turned on if the metric class is used with `itk::BSplineDeformableTransformOpt`. As only a fixed number of control points in the Jacobian field are nonzero, the computational gain is significant, especially if the deformation field is dense. The changes we made to `itk::BSplineDeformableTransform` are backward compatible as we keep the default implementation of the member function `GetJacobian( point)`.

3.3 Component Interaction

Interconnections between the components should be handled before starting the registration. We modified the pairwise registration method `itk::MultiResolutionImageRegistrationMethod` to create a groupwise registration method `itk::MultiResolutionMultiImageRegistrationMethod` that automatically initializes connections between registration components and provides a multiresolution optimization scheme. The Registration method has several functions to describe a multiresolution groupwise registration setting.

- `SetNumberOfImages()`: Sets number of images used in registration
- `SetNumberOfLevels()`: Sets number of multiresolution levels
- `SetOptimizer()`: Sets the optimizer
- `SetMetric()`: Sets the registration metric
- `SetImagePyramidArray(int, imagePyramid)`: Uses the given image pyramid in the registration
- `SetTransformArray(int,transform)`: Connects the given transform to corresponding images
- `SetInterpolatorArray( int,interpolator)`: Connects the given interpolator to corresponding images
- `SetTransformParametersLength()`: Allocates space for transform parameters
- `StartRegistration()`: Connects the components and starts the registration

Using the functions above the registration method initializes registration components, handles the interconnections between them and performs a multi-resolution optimization. Appendix B provides a registration example using this class.

3.4 Summary of New Classes

We provide the following classes as part of the groupwise registration framework.

- `itk::MultiImageMetric<TImage>`
  A multi-threaded base class templated over the input image for groupwise registration metrics.
  - `itk::UnivariateEntropyMultiImageMetric<TImage>`
    Computes sum of univariate entropies along pixel stacks.
  - `itk::VarianceMultiImageMetric<TImage>`
    Computes sum of variances along pixel stacks.
- `itk::MultiResolutionMultiImageRegistrationMethod<TImage>`
  Provides a generic interface for multi-resolution registration using components of the registration framework.
- `itk::GradientDescentLineSearchOptimizer`
  Gradient descent optimizer combined with line search for determining the step size.
- `itk::BSplineDeformableTransformOpt<TScalarType, NDimensions, VSplineOrder >`
  B-spline deformable transform optimized for Jacobian computations.

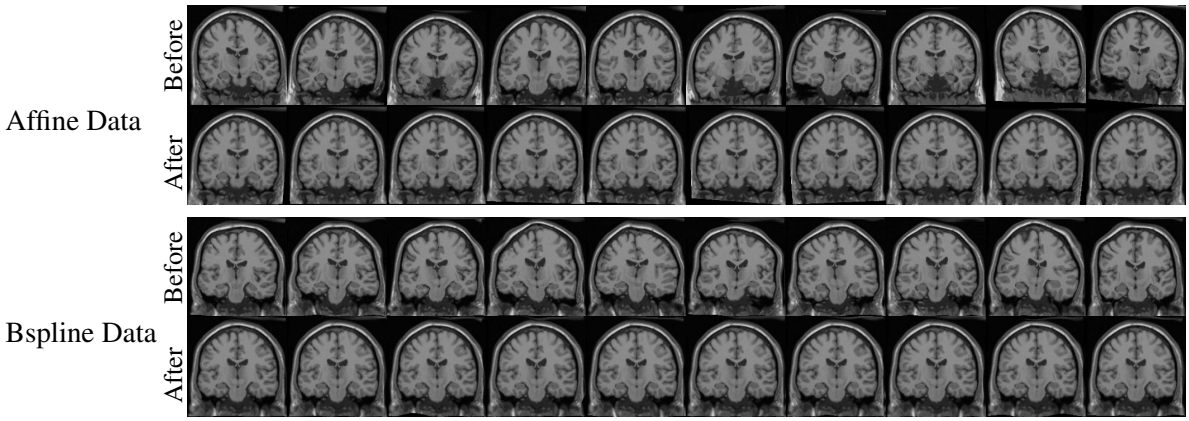


Figure 2: Central slices of synthetic affine and B-spline 3D volumes before and after registration.

4 Results

To validate the algorithm we run synthetic experiments using four different imaging modalities: proton density data with $181 \times 217 \times 180$ voxels and $1 \times 1 \times 1$ mm spacing, T1 data with $181 \times 217 \times 180$ voxels and $1 \times 1 \times 1$ mm spacing, T2 data with $181 \times 217 \times 180$ voxels and $1 \times 1 \times 1$ mm spacing, FA data with $256 \times 256 \times 50$ voxels and $0.9375 \times 0.9375 \times 2.5$ mm spacing. The input data can be obtained from ITK's Data directory at http://public.kitware.com/pub/itk/Data/.

For each sample medical data we perform two different sets of experiments. First, we apply random affine transforms to input data and register this synthetic dataset using global affine registration. In the second setting, we apply random B-spline transforms to input data and recover applied transforms using B-spline registration. Figure 2 show the central slices of 10 random 3D images from the T1 MR image before and after registration. The figure indicate that the images are well aligned after registration.

To evaluate registration accuracy visually, we compute mean and standard deviation images before and after registration. Figure 3 show the central slices of mean and standard deviation images for all four modalities using an input data of 10 images. Visually we can observe that the mean images get sharper and the standard deviation images get darker. In a perfect registration, the standard deviation images should be all zero images.

From the images in figure 3, we can note some registration artifacts at image boundaries. These occur because during random image generation images get cropped at boundaries. For datasets generated with affine transforms, standard deviation images get close zero after affine registration. Therefore, we can state that for all imaging modalities transform components are recovered successfully.

For datasets generated with B-spline transforms, we note some residuals at central locations in standard deviation images after non-rigid registration. This occur because we used the same transform complexity with $8 \times 8 \times 8$ B-spline control points for both image generation and registration. The residuals can be made smaller by using B-splines with higher number of control points, as a dense deformation field can capture larger shape variations. The main reason for the residuals is that the inverse of a B-spline transform is generally not a B-spline transform and the registration results show the best fit to the inverse transforms.

To examine the performance of the algorithm with respect to number of input images we run experiments with 2, 10 and 30 images. For the results of these experiments see Appendix A. The results in Appendix A suggest that the groupwise registration algorithm is robust to varying number of input images. Visually, standard deviation values are close to zero which indicates successful recovery of applied transforms.

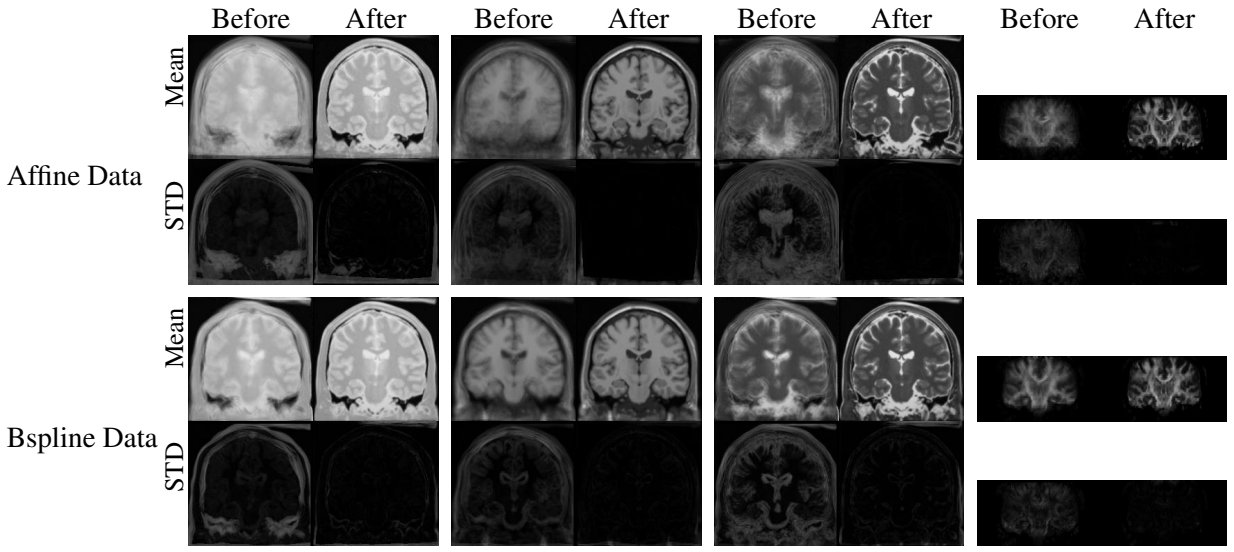


Figure 3: Central slices of 3D volumes before and after registration for (from left to right) PD, T1, T2 and FA synthetic images. Inside a block top row shows the mean images and bottom row standard deviation images before and after registration. Blocks in the top row show the results for synthetic data generated with 10 random affine transforms and blocks in the bottom row show results for synthetic data generated with 10 random B-spline transforms.

The experiments show that the groupwise registration algorithm can handle few input images including a minimum of two input images.

5 Running Tests

To reproduce the results presented in this paper please follow the directions below:

1. Compile ITK 2.8 or higher (http://www.itk.org).
2. Install CMake 2.4 or higher (http://www.cmake.org).
3. Download Groupwise Registration Project from Insight Journal (http://insight-journal.org).
4. Configure and compile Groupwise Registration Project using CMake. Point the folder containing brain MR images to Brainweb_DATA_ROOT and the folder containing FA images to FAImage_Data_ROOT. To avoid timeout errors set DART_TESTING_TIMEOUT parameter to a large value(e.g. 36000).
5. To reproduce the results presented in this document, run CTest from the project folder.
6. Check test results from the corresponding folders in Testing/Temporary. Examine registration parameters and the example code `GroupwiseRegistrationExample.cxx`.

6 Conclusion

In this paper, we describe an open source implementation of a non-rigid groupwise registration algorithm using the Insight Toolkit ITK `www.itk.org`. We provide the source code, parameters, input and output data that we used for validation.

We demonstrated the algorithm in application to different imaging modalities including proton density, FA, T1 and T2 MR images. We validated the algorithm on synthetic datasets varying from 2 to 30 images by recovering randomly applied affine and B-spline transforms.

Appendix A show the results for all experiments Appendix B present a groupwise registration example using the implemented classes and Appendix C explain how to use command line modules.

A Test Results

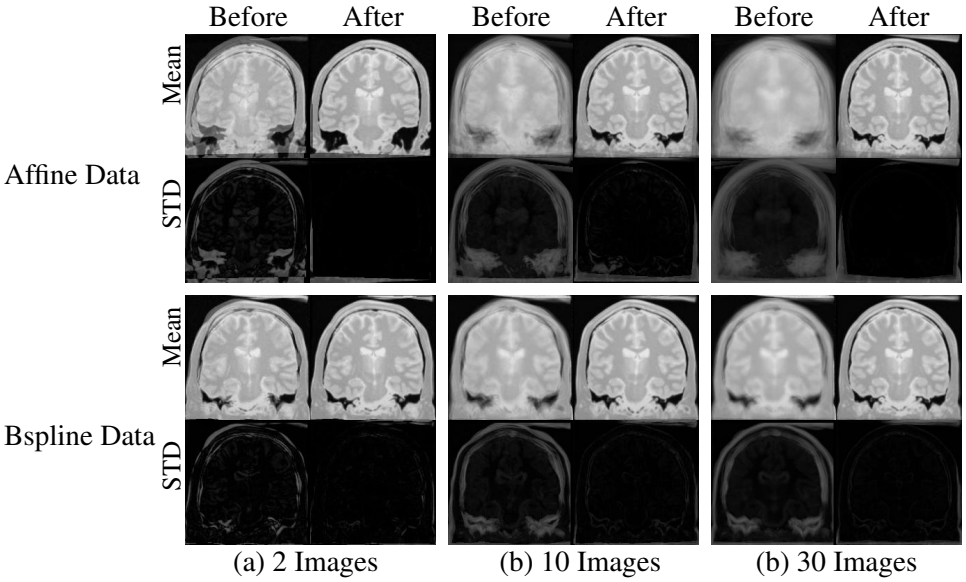


Figure 4: Central slices of 3D volumes before and after registration for proton density images with varying number of input images.

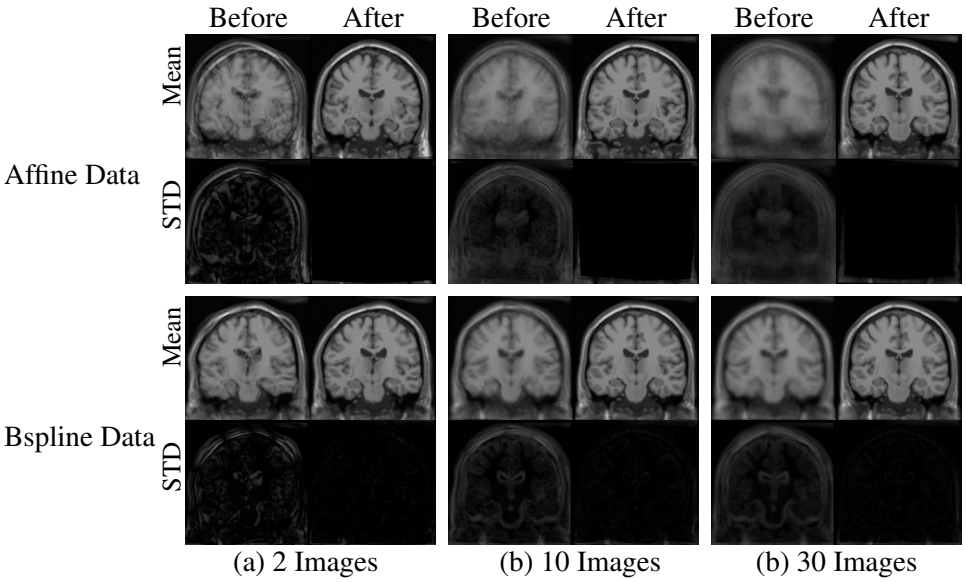


Figure 5: Central slices of 3D volumes before and after registration for T1 MR images with varying number of input images.

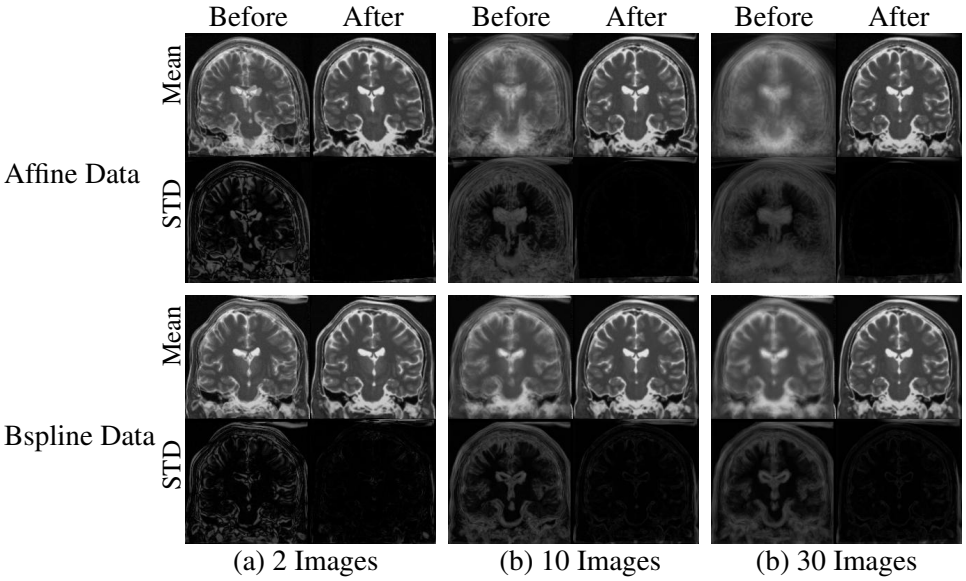


Figure 6: Central slices of 3D volumes before and after registration for T2 MR images with varying number of input images.

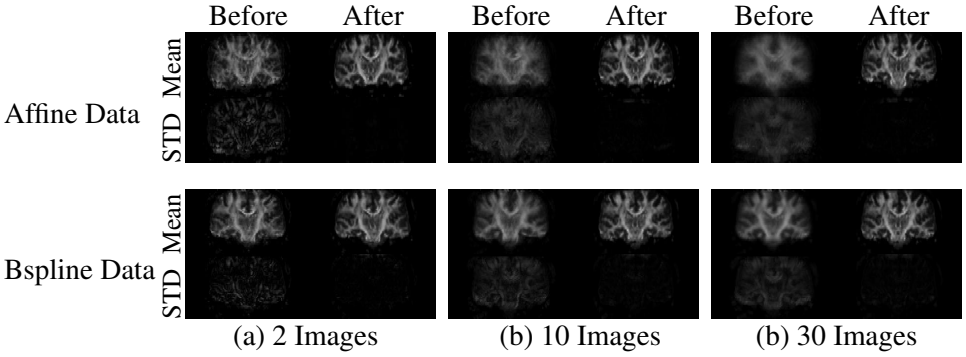


Figure 7: Central slices of 3D volumes before and after registration for FA images with varying number of input images.

B Registration Example

The source code for this example can be found in `Source/GroupwiseRegistrationExample.cxx`

In this example we show how to use the groupwise registration framework. The example code performs groupwise registration on an input data and outputs the resulting transform parameters. The resulting transform parameters can be used to transform input images, calculate mean and standard deviation images. As the size of the example code is relatively large to explain all in detail here, we will briefly mention main parts and show how to set up main components.

We first include project specific headers and define the pixel types using `#define` statements.

```
#include "MultiResolutionMultiImageRegistrationMethod.h"
#include "VarianceMultiImageMetric.h"
#include "UnivariateEntropyMultiImageMetric.h"
#include "BSplineDeformableTransformOpt.h"
#include "GradientDescentLineSearchOptimizer.h"
```

```
#define Dimension         3
#define InternalPixelType float
```

As itk::ImageFileReader casts input images to specified internal type we only define an internal image type. For metric calculations pixel type should have real value (float or double). To decrease the memory requirements we use float type.

```
typedef itk::Image< InternalPixelType, Dimension >      InternalImageType;
```

We declare the registration method type by using the internal image type as a template argument. This class handles the interconnection between registration components and uses a multi-resolution optimization.

```
typedef itk::MultiResolutionMultiImageRegistrationMethod< InternalImageType >
                                                          RegistrationType;
RegistrationType::Pointer   registration  = RegistrationType::New();
```

First we set the number of images to be used in the registration and the number of multi-resolution levels. The number N is parsed from the input parameters.

```
registration->SetNumberOfImages(N)
registration->SetNumberOfLevels( multiLevelAffine );
```

To optimize the objective function we use gradient descent algorithm combined with line search. We declare the optimizer type and connect to the registration using SetOptimizer() method.

```
typedef itk::GradientDescentLineSearchOptimizer  LineSearchOptimizerType;
LineSearchOptimizerType::Pointer lineSearchOptimizer;
lineSearchOptimizer     = LineSearchOptimizerType::New();

registration->SetOptimizer(     lineSearchOptimizer     );
```

Next, we define the metric type by passing internal image type as a template argument. To avoid if statements later in the code we assign metric pointers to the base class typedef itk::MultiImageMetric. The metric pointer is connected to the registration using SetMetric() method. The type of the metric to be used can be set through parameters in the input files.

```
typedef itk::MultiImageMetric< InternalImageType>                   MetricType;
typedef itk::VarianceMultiImageMetric< InternalImageType>           VarianceMetricType;
typedef itk::UnivariateEntropyMultiImageMetric< InternalImageType>  EntropyMetricType;
MetricType::Pointer                 metric;
if(metricType == "variance")
{
  metric        = VarianceMetricType::New();
}
else
{
  EntropyMetricType::Pointer entropyMetric        = EntropyMetricType::New();
  entropyMetric->SetImageStandardDeviation(parzenWindowStandardDeviation);
  metric = entropyMetric;
}

registration->SetMetric( metric  );
```

After instantiating the metric and the optimizer, we connect images to the registration method. Each image is associated with an interpolator and a transform. Therefore, we start by declaring interpolator and transform arrays.

```
typedef itk::AffineTransform< ScalarType, Dimension >   TransformType;
typedef std::vector<TransformType::Pointer>             TransformArrayType;
TransformArrayType                          affineTransformArray(N);

typedef itk::LinearInterpolateImageFunction<InternalImageType,ScalarType>  InterpolatorType;
typedef vector<InterpolatorType::Pointer>                              InterpolatorArrayType;
InterpolatorArrayType      interpolatorArray(N);
```

We instantiate interpolator and transforms in a for loop and connect to the registration method

```
affineTransformArray[i]     = TransformType::New();
registration->SetTransformArray( i, affineTransformArray[i]);

interpolatorArray[i]  = InterpolatorType::New();
registration->SetInterpolatorArray( i,interpolatorArray[i]);
```

To perform a multi-resolution registration we build image pyramids and connect to the registration method. We start by declaring image pyramid filters.

```
typedef itk::RecursiveMultiResolutionPyramidImageFilter<InternalImageType,InternalImageType>
                                                          ImagePyramidType;
typedef  vector<ImagePyramidType::Pointer>                ImagePyramidArray;
ImagePyramidArray imagePyramidArray(N);
```

In a for loop we construct image pyramid and connect them to registration using `SetImagePyramidArray` function.

```
imagePyramidArray[i] = ImagePyramidType::New();
imagePyramidArray[i]->SetNumberOfLevels( multiLevelAffine );
imagePyramidArray[i]->SetInput( imageReader->GetOutput() );
imagePyramidArray[i]->Update();

registration->SetImagePyramidArray(i, imagePyramidArray[i]);
```

We now set up the initial parameters of the registration. Registration method uses a parameters array which is formed by the concatenation of individual transform parameters.

```
typedef RegistrationType::ParametersType ParametersType;
ParametersType initialAffineParameters( affineTransformArray[0]->GetNumberOfParameters()*N );
initialAffineParameters.Fill(0.0);

registration->SetInitialTransformParameters( initialAffineParameters );
```

Next, we specify the region from which the samples are taken. We specify the sample region to be the whole image region.

```
InternalImageType::RegionType fixedImageRegion;
imagePyramidArray[0]->GetOutput(imagePyramidArray[0]->GetNumberOfLevels()-1)->GetBufferedRegion();
registration->SetFixedImageRegion( fixedImageRegion );
```

In the following line we set the number samples to be used in the metric. For choosing the parameters of the algorithm see the section C.1 on registration parameters.

```
metric->SetNumberOfSpatialSamples( numberOfSamples );
```

Now, we all components are instantiated and the registration is ready to be started

```
registration->StartRegistration();
```

We use the results of global affine registration above to initialize B-spline transforms. We start by declaring the B-spline transform type.

```
const unsigned int SplineOrder = 3;
typedef double CoordinateRepType;
typedef itk::BSplineDeformableTransformOpt<  CoordinateRepType,Dimension,SplineOrder >
                                              BSplineTransformType;
typedef vector<BSplineTransformType::Pointer> BSplineTransformArrayType;
BSplineTransformArrayType      bsplineTransformArrayLow(N);
```

Next, we set the total parameters length of B-splines. This is necessary as ITK's B-spline implementation only holds pointers to actual transform parameters. We calculate length of the B-spline parameters explicitly from the input parameters.

```
registration->SetTransformParametersLength(
             static_cast<int>( pow( static_cast<double>(bsplineInitialGridSize+SplineOrder),
                                    static_cast<int>(Dimension))*Dimension*N ) );
```

To initialize B-splines using the results of the global affine registration we get the latest parameters from the registration.

```
ParametersType affineParameters = registration->GetLastTransformParameters();
ParametersType affineCurrentParameters(affineTransformArray[0]->GetNumberOfParameters());

for( int i=0; i<N; i++)
{
  for(unsigned int j=0; j<affineTransformArray[0]->GetNumberOfParameters(); j++)
  {
    affineCurrentParameters[j]=
                affineParameters[i*affineTransformArray[0]->GetNumberOfParameters()+j];
  }
  affineTransformArray[i]->SetParametersByValue(affineCurrentParameters);
}
```

In a for loop, we initialize B-splines and connect them to registration method.

```
bsplineTransformArrayLow[i] = BSplineTransformType::New();
bsplineTransformArrayLow[i]->SetBulkTransform(affineTransformArray[i]);
bsplineTransformArrayLow[i]->SetParameters( bsplineParametersArrayLow[i] );

registration->SetInitialTransformParameters
                         (i, bsplineTransformArrayLow[i]->GetParameters());
registration->SetTransformArray(i, bsplineTransformArrayLow[i]);
```

To make use of the B-spline optimization, we explicitly connect B-spline transforms to the metric.

```
metric->SetBSplineTransformArray(i, bsplineTransformArrayLow[i]);
```

Now, B-spline registration can be started

```
registration->StartRegistration();
```

After each stage of the algorithm, e.g. affine and B-spline registration, transform parameters are written to files using `itk::TransformFileReader`.

To obtain a dense deformation field capturing variations at different scales, we gradually increase the complexity of the deformation field by refining the grid of B-spline control points. See the example code on how to initialize B-spline control points at finer grids using results at coarser grids.

C Using Command Line Modules

We provide the modules `CreateImageSetAffine` and `CreateImageSetBspline` to generate synthetic datasets from a sample data. These modules apply random transforms to the input image, generate a dataset with the specified number of images and write the results to an output folder. The usage of the command line modules is as follows

```
CreateImageSetAffine inputImage outputFolder numberOfImages
```

`GroupwiseRegistration` is the main command line tool to perform groupwise registration. The parameters of the registration algorithm are passed to the binary using two text files. Folders under `Testing/Temporary` contain the text files that we used for each experiment. The following line shows the usage of the binary

```
GroupwiseRegistration filenames.txt parameters.txt
```

The first text file contains the paths of the input folder, output folder and the file names. The second text file supplies the parameters of the registration algorithm and is explained more in detail in the next section. A sample `filenames.txt` for setting up a registration with two input images is as follows

```
-i inputFolder/
-o outputFolder/

-f filename1
-f filename2
```

The registration code only outputs transform parameters. These transform parameters can be used to visualize registration results. We provide `ComputeOutputs` command line module to visualize registration results in 3D. This module outputs transformed images, mean and standard deviation images along with central slices. `ComputeOutputs` takes the same input arguments as `GroupwiseRegistration` and can be used as follows

```
ComputeOutputs filenames.txt parameters.txt
```

C.1 Registration Parameters

Registration parameters for each experiment can be found under corresponding folders in `Testing/Temporary`. Here we briefly go over the registration parameters used in a non-rigid regis-

tration setting.

The metric type can be specified using the option `-metricType`. `entropy` option computes sum of univariate entropies and `variance` option computes sum of variances along pixel stacks.

```
-metricType entropy
```

By default, global affine registration is performed. B-spline registration can be turned on by using the `-useBspline on` option. `-useBsplineHigh` option specifies whether to use mesh refinement in the registration.

```
-useBspline on
-useBsplineHigh off
```

The initial size of the B-spline deformation field can be set using the following option. To capture anatomical variations at different scales we start with a coarse size of 8 points and gradually increase the number of control points.

```
-bsplineInitialGridSize  8
```

If `-useBsplineHigh` is turned on, the number of mesh refinements can be specified using the `-numberOfBsplineLevel` option. After each level of B-spline registration the number of B-spline control points are doubled, e.g. $8 \rightarrow 16 \rightarrow 32$ for a two level mesh refinement starting from an initial size of 8.

```
-numberOfBsplineLevel 2
```

For stochastic optimization we randomly subsample the image domain. Increasing the percentage of samples increases the registration accuracy; however, it also increases the registration time. Empirically, we found the following parameters as a good trade-off between registration accuracy and run-time.

```
-numberOfSpatialSamplesAffinePercentage      0.0025
-numberOfSpatialSamplesBsplinePercentage     0.0050
-numberOfSpatialSamplesBsplineHighPercentage 0.0200
```

For a successful registration, multi-resolution optimization plays a key role. For all experiments we used a three level multi-resolution optimization. If the resolution of the input images are smaller than the data we used ($\approx 200 \times 200 \times 200$) the number of multi-resolution levels can be decreased to two.

```
-multiLevelAffine 3
-multiLevelBspline 3
-multiLevelBsplineHigh 3
```

For optimization we used fixed number of iterations as the stopping criteria. The command line module `GroupwiseRegistration` outputs metric values every ten iterations. These outputs can be used to check the convergence of the algorithm. The number of iterations can be increased for higher registration accuracy. During the tests we observed that the number of iterations should be increased with an increase in the number of input images.

```
-optAffineNumberOfIterations 50
-optBsplineNumberOfIterations 75
-optBsplineHighNumberOfIterations 50
```

The initial step size of the gradient descent algorithm can be set using the following options. As we use line search for the actual step size, the registration accuracy is relatively robust to step size.

```
-optAffineLearningRate  1e-4
-optBsplineLearningRate 1e4
-optBsplineHighLearningrate 1e4
```

An important parameter for `itk::UnivariateEntropyMultiImageMetric` is the width of the Gaussian kernel to compute the entropy. We observed that approximately five percent of the range of the intensity values work in practice. For intensity values ranging between 0-255 we used a value of 10.

```
-parzenWindowStandardDeviation 10
```

References

[1] K K Bhatia, J V Hajnal, B K Puri, A D Edwards, and D Rueckert. Consistent groupwise non-rigid registration for atlas construction. In *IEEE ISBI*, 2004. 2.2, 2.2

[2] W R Crum, T Hartkens, and D L G Hill. Non-rigid image registration: Theory and practice. *The British Journal of Radiology*, 77(140–153), 2004. 1

[3] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973. 2.1

[4] L. Ibanez, W. Schroeder, L. Ng, and J. Cates. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, first edition, 2003. 3

[5] S Joshi, Brad Davis, Matthieu Jomier, and Guido Gerig. Unbiased diffeomorphic atlas construction for computational anatomy. *NeuroImage*, 23:151–160, 2004. 3.1

[6] Stefan Klein, Marius Staring, and Josien P.W. Pluim. A comparison of acceleration techniques for nonrigid medical image registration. In *WBIR*, pages 151–159, 2006. 3.1

[7] E. Miller, N. Matsakis, and P. Viola. Learning from one example through shared densities on transforms. In *IEEE CVPR*, pages 464–471, 2000. 1, 2

[8] D. Rueckert and et al. Nonrigid registration using free-form deformations: Application to breast mr images. *IEEE TMI*, 22:120–128, 2003. 2.2

[9] A Toga and P Thompson. The role of image registration in brain mapping. *Image and Vision Computing*, 19(3–24), 2001. 1

[10] Barbara Zitova and Jan Flusser. Image registration methods: A survey. *Image and Vision Computing*, 21(977–1000), 2003. 1

[11] Lilla Zollei, E. Learned-Miller, E. Grimson, and W. Wells. Efficient population registration of 3d data. In *Computer Vision for Biomedical Image Applications, ICCV*, pages 291–301, 2005. 1, 2

An accessible, hands-on tutorial system for image-guided therapy and medical robotics using a robot and open-source software

Release 0.00

Danielle F. Pace[1], Ron Kikinis[1] and Nobuhiko Hata[1]

September 1, 2007

[1]Surgical Planning Laboratory, Brigham and Women's Hospital and Harvard Medical School
Boston, MA, USA

Abstract

This paper describes a new open-source tutorial for image-guided therapy (IGT) and medical robotics that is both accessible and hands-on using the LEGO Mindstorms NXT (a commercially available robotics kit) and 3D Slicer (an open-source application for medical image processing). The tutorial covers all stages of a typical IGT or medical robotics procedure, including the concepts of imaging, preoperative planning, targeting and tracking, navigation and registration, by using the LEGO robot to perform a "needle biopsy" on a phantom (anatomical model) made of traditional LEGO pieces. In addition, this paper describes a C++ library that allows direct control of a LEGO Mindstorms NXT robot from a Linux computer over a USB connection.

Contents

1 Introduction **2**

2 Materials and Methods **3**

2.1 The tutorial supplies 3

2.2 Building a robot using LEGO 3

2.3 Robotic control using 3D Slicer 4

2.4 The tutorial task: a simulated needle biopsy 5

2.5 The basic tutorial covers the typical IGT and medical robotics workflow 7

2.6 The advanced tutorial introduces registration 9

2.7 Assessment of targeting accuracy 11

3 Results **13**

3.1 The IGT and medical robotics tutorial is highly accessible 13

3.2 Assessment of targeting accuracy 13

4 Discussion 15

5 Conclusions 16

6 Acknowledgement 16

A Functions of our robotic control library 16

B Example use of our robotic control library 17

1 Introduction

Image-guided therapy (IGT) and medical robotics compose a modern interventional approach that uses technology to enable new minimally invasive therapies, improve postoperative outcomes, increase the quality and speed of interventional procedures, shorten hospital stays and decrease long-term hospital costs. It is because of these notable advantages that IGT and medical robotics have gained significant interests in research, clinics and industry and have made major advances in recent years. Open-source software projects such as ITK [1], VTK [2] and IGSTK [3] are major contributors to this growth, as are multi-center collaborations such as those described by Hata *et al.* in [4]. The current expansion of IGT and medical robotics means that both academia and industry require specialized personnel in order to continue the forward momentum. Therefore educational materials that both attract people to IGT and medical robotics and provide them with the necessary theoretical knowledge and technical skills must be widely distributed. In order to reach as many people as possible, these tools must be open-source, low-cost and widely available for purchase. This goal has been partially met by tutorials, books and seminars associated with the aforementioned open-source software packages [5].

However, current educational tools for IGT and medical robotics are insufficient because the field's intrinsic reliance on equipment means that tutorials must be hands-on in order to be truly effective. Since tracking devices are expensive and phantoms (anatomical models) are time-consuming to construct, this requirement directly conflicts with the equally crucial requirements for educational materials to be open-source, low-cost and widely available for purchase. The vast majority of tutorials on IGT and medical robotics are unfortunately not hands-on. A very comprehensive practical tutorial system is provided in [6], however it is not accessible because it requires the purchase of a tracking device. To the best of our knowledge, there does not exist a sub-$500 USD, practical tutorial system for IGT and medical robotics available today. Instead, the only options available for many newcomers to the field are to merely read about the subject or, if they are lucky, attend a brief conference workshop that may or may not have a practical component. These educational experiences do not come close to conveying the dynamic and exciting world of IGT and medical robotics: without practical and accessible educational tools, the field of image-guided therapy and medical robotics will fail to capture the best and brightest minds that it needs to continue its rapid growth.

The objective of this paper is to describe a tutorial for image-guided therapy and medical robotics that is open-source, accessible and hands-on. The tutorial uses the LEGO Mindstorms NXT, an inexpensive and widely available robotics kit. The software architecture chosen for this project was 3D Slicer (Massachusetts Institute of Technology Artificial Intelligence Lab and Brigham and Women's Hospital,

Boston, Massachusetts), a comprehensive open-source software package for medical image processing and image-guided therapy [7]. 3D Slicer was selected for its very unrestrictive license and its extensive built-in image processing and IGT functionality. The LEGO robot and 3D Slicer are used to simulate a needle biopsy with a preoperative target (such as that which may be done in potential cases of breast or prostate cancer) on a phantom created out of standard LEGO pieces. In this way, tutorial participants work with both hardware and software while being exposed to all of the typical steps of an IGT or medical robotics procedure, including imaging, preoperative planning, targeting and tracking, navigation and registration.

2 Materials and Methods

2.1 The tutorial supplies

Table 1 lists the materials used in our image-guided therapy and medical robotics tutorial. Tutorial participants are required to supply a LEGO Mindstorms NXT kit, a LEGO Deluxe Brick Box of traditional LEGO pieces, various other small components needed to build the phantom, and a Linux computer with root access. We provide open-source tutorial software in the form of a specialized module in 3D Slicer version 3, a high-resolution CT volume of the phantom (scanned at 3 mm with no overlapping slices), tutorial slides, assembly instructions for the LEGO robot and the phantom, and a "Phantom Placement Guide" that aids in positioning the LEGO robot and the phantom during the tutorial. Before beginning the tutorial, participants use our provided instructions and the LEGO Mindstorms NXT kit to construct the tutorial robot that will perform the simulated needle biopsy. Users also build the phantom (anatomical model) using the LEGO Deluxe Brick Box, pom-poms, paper and tape.

2.2 Building a robot using LEGO

The LEGO Mindstorms NXT is a basic robotics kit manufactured and sold by The LEGO Group. Although originally commercialized based on research by the MIT Media Lab [8] as a toy for children,

Table 1 The materials used in our image-guided therapy and medical robotics tutorial. The left column lists materials that must be supplied by users of the tutorial, while the right column lists open-source materials that we provide as part of the tutorial package.

Materials provided by the tutorial participant	Materials provided as part of the tutorial package
• One LEGO Mindstorms NXT robotics kit (Item #8527) • One LEGO Deluxe Brick Box (Item #6167) • Two pom-poms of diameter 2.5 cm (1 inch) • Two pieces of paper and tape • A Linux computer with root access	• Specialized tutorial module in 3D Slicer v. 3 • CT volume of the phantom • Tutorial slides containing background information and instructions on how to follow the tutorial • Assembly instructions for the LEGO robot and the phantom • Phantom placement guide

the potential of the LEGO robot as a tool for research and education was quickly taken advantage of by the scientific community. LEGO robots have been used to investigate prey retrieval in collective robotics [9] and are also useful for rapid-prototyping of interactive robots [10]. In the educational domain, LEGO robots have been integrated into undergraduate computer science courses in artificial intelligence [11, 12] while LEGO robot competitions provide popular introductions for children to robotics [13, 14]. In particular, our IGT and medical robotics tutorial was inspired by the work of the Computer-Integrated Surgery Student Research Society (CISSRS), which runs weekend-long CISSRS Surgical LEGO Robot Competitions to expose high-school students to medical robotics in particular [15]. The ease with which robots can be quickly constructed and programmed, as well as the extensive open-source software and documentation provided by both The LEGO Group and the extremely active LEGO Mindstorms NXT community, makes LEGO an ideal system with which to build a tutorial for IGT and medical robotics.

Detailed hardware and software specifications of the LEGO Mindstorms NXT can be found in The LEGO Group's description of the product [16] and in several books on the system (such as [17]). Provided in the LEGO Mindstorms NXT kit is the "NXT intelligent brick", containing a 32-bit ARM7 microcontroller, a 8-bit AVR microcontroller, 256 + 4 Kbytes of FLASH memory and 64 Kbytes + 512 bytes of RAM. Also included are four sensors: the touch sensor provides button-press functionality, the sound sensor detects decibels up to 90 dB, the light sensor measures light intensity and the ultrasonic sensor detects objects by measuring the distance to the nearest object in front of it (in units of one centimeter). Three servo motors each have an adjustable power setting to provide rotational movement and a built-in rotation sensor that returns the number of degrees that its motor has rotated through since the sensor's last reset. Finally, 519 LEGO TECHNIC pieces form the basic construction materials used to build a robot. LEGO Mindstorms NXT robots are typically smaller than 2' x 2' x 2' although this depends on the design. Many scientific and educational projects using LEGO Mindstorms use multiple kits to build a single robot, however we constrained ourselves to a single kit in order to minimize cost to the user and therefore satisfy our goal of accessibility.

Typically, when programming a LEGO Mindstorms NXT one writes and compiles the code on a personal computer and then transfers the executable to the robot's intelligent brick using its USB 2.0 or Bluetooth capability. The robot then acts autonomously from the computer when the program is run on the brick. Although LEGO ships its own graphical programming software with the product, most users with programming experience prefer to use an alternative open-source language that is text-based. Of particular note is Not eXactly C (NXC) [18], a C-like language built on top of the affiliated Next Byte Codes (NBC) compiler, and Bricx Command Center [19], a popular integrated development environment (IDE). Both the official graphical LEGO Mindstorms NXT programming software and Bricx Command Center run on Windows and Macintosh systems only. However, UNIX users can write NXC code in any text editor, compile using the command line NBC compiler and transfer the resulting executable to the robot using the open-source LiNXT [20]. Finally, The LEGO Group also provides documentation for advanced users to send direct commands covering all robotic functions from a personal computer to the NXT brick, to create their own sensors, and more [21].

2.3 Robotic control using 3D Slicer

In contrast to the typical programming procedure described above, in our IGT and medical robotics tutorial the LEGO robot does not act autonomously. Instead, all visualization, preoperative planning, robotic control, registration and miscellaneous computation is performed using a specialized 3D Slicer v. 3 module that participants download, build and install. Thus this tutorial builds upon the work done by CISSRS by 1) integrating an advanced image processing and IGT software package currently used for

research and development and 2) covering more advanced topics such as preoperative planning and registration.

Our need to send direct commands to the LEGO robot from 3D Slicer led to the development of a C++ robotic control library that enables the control of an LEGO Mindstorms NXT robot from a computer over a USB connection. In particular, one can read sensor values, move the motors and read from the motors' rotation sensors within any C++ program that includes our library. The USB functionality of the LEGO NXT intelligent brick was chosen over Bluetooth to maximize the reliability of the connection, especially in a classroom setting where multiple robots may be in use at the same time. Please see Appendices A and B for a list of the functions provided by our C++ library and a simple example of their use.

Our C++ library was developed based on the open-source projects NXT++ [22] and Device::USB [23]. Although NXT++ provides the same functionality as our library, it was not used in this project because it unfortunately did not work properly on our 64-bit Linux platform. Our robotic control library also uses the open-source library libusb [24] to access the USB device. At present our robotic control library is available only for the Linux platform, although support for Windows and Macintosh is planned for the near future. Please note that programs using our library should be executed as root, since root access guarantees the permissions required to access the USB device using libusb.

2.4 The tutorial task: a simulated needle biopsy

As previously mentioned, our tutorial uses a needle biopsy as a model of a typical IGT or medical robotics intervention. Equipped with a "needle" that can move up and down, the tutorial robot is used to "biopsy" a "tumour" target on the phantom by aiming the end of the needle actuator at the target (Figure 1).

The LEGO robot designed for our tutorial is shown in Figure 1A. All three motors provided with the LEGO Mindstorms NXT kit are utilized in the design: one to swing the robotic arm from side to side (motor A), one to move the robotic arm forwards/down and backwards/up (motor B), and one to move the needle up and down (motor C). In addition, one of the four LEGO sensors is used: the ultrasonic sensor measures the distance to the nearest object in front of it and is used to get fiducial coordinates on the phantom in the registration process (as described extensively in Section 2.6).

Figure 1B shows a photograph of the phantom created for the IGT and medical robotics tutorial. Using LEGO pieces to create the phantom gave us maximum design flexibility while maintaining affordability for the user. In addition, the dimensions of LEGO pieces are so exceptionally precise that we were able to define a coordinate system in the robot's physical space, known as the patient (robot) coordinate system, or *PCS*, in terms of "LEGO units" of length 0.8 cm: the distance between the centers of adjacent LEGO studs. A red and a green pom-pom (which are small fuzzy balls typically used in arts-and-crafts projects) are visible in Figure 1B and are referred to as such in the remainder of this paper. The centers of these pom-poms are the two tumour targets. The phantom is composed of a central box containing two smaller boxes into which the pom-pom tumour targets are placed. Four "registration pillars" surround the central box (two tall ones at the back of the phantom and two shorter ones in front). The registration pillars and the ultrasonic sensor are used to find fiducial coordinates in the *PCS* in the registration process. The top halves of the registration pillars are wrapped with paper that is secured with tape so that they can be better "seen" by the ultrasonic sensor. Finally, a T-shaped spacer at the bottom of the phantom aids with its positioning during the hands-on components of the tutorial.

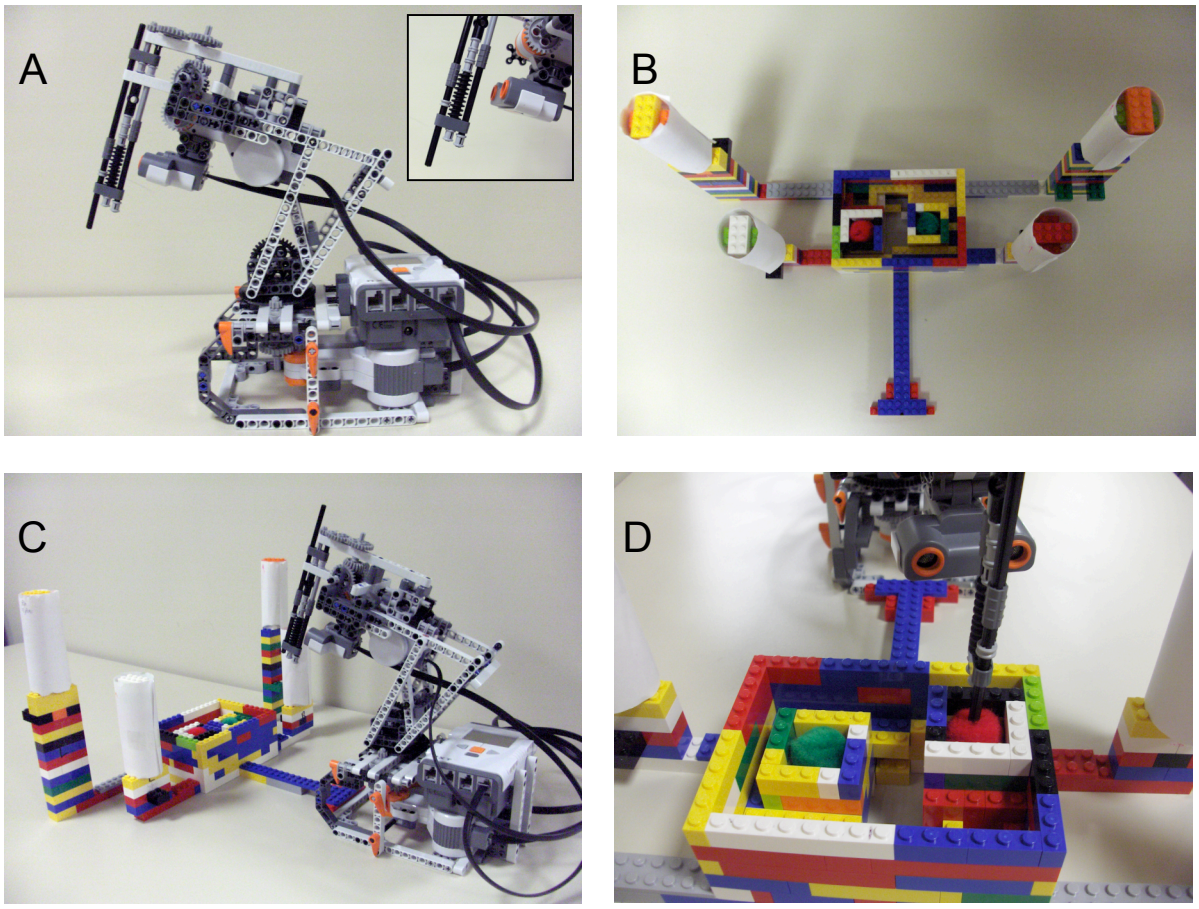


Figure 1 The tutorial task is to have the LEGO robot perform a "needle biopsy" on the phantom. A) The tutorial robot (the inset highlights the needle mechanism and the ultrasonic sensor); B) The phantom (note the red and green pom-pom targets and the registration pillars); C) An example of the tutorial setup; D) A successful needle biopsy of the red pom-pom target.

Our tutorial is composed of three sections. The Background and Materials section provides theoretical information about image-guided therapy and medical robotics, with a focus on the five tutorial concepts that we aim to cover: imaging, preoperative planning, targeting and tracking, navigation and registration. The next two sections cover the material in a hands-on fashion using the LEGO robot and the phantom (Table 2). The Basic Tutorial is the first of the two hands-on tutorial sections and provides education on all but the last of the tutorial topics listed above, while the Advanced Tutorial reinforces the concepts of imaging, preoperative planning, targeting and tracking and navigation and also incorporates a registration procedure. In total, the estimated time to complete all three sections is approximately two hours: one hour to read the material in the Background and Materials section and one hour to complete the hands-on Basic and Advanced tutorial sections.

Table 2 The five tutorial concepts to be covered in a hands-on manner in the basic and advanced tutorial sections.

Tutorial concept	Basic	Advanced	How?
Imaging	✓	✓	CT volume of the phantom
Preoperative planning	✓	✓	Target selection on the CT volume by clicking
Targeting and tracking	✓	✓	Robotic control by the 3D Slicer tutorial module
Navigation	✓	✓	Report of the final needle position after the biopsy is executed
Registration		✓	Using eight fiducials in the image and the patient (robot) coordinate systems and a rigid landmark registration algorithm

2.5 The basic tutorial covers the typical IGT and medical robotics workflow

In the basic tutorial, participants use the spacer at the base of the phantom and the phantom placement guide to put the phantom in a predefined position and orientation with respect to the LEGO robot (see Figure 2A). The LEGO robot is also initially positioned such that the robotic arm is centered and pushed as far back as possible, and the needle is in its highest position. We must specify the robot's initial configuration because there is no way to automatically sense the initial configuration at the beginning of the tutorial. The user then follows the tutorial slides to move through the typical image-guided therapy / medical robotics workflow. He or she will upload the CT volume of the phantom in 3D Slicer and establish the USB connection between the LEGO robot and the 3D Slicer tutorial module. The user then selects the target coordinate in the image coordinate system, or *ICS*, by clicking on the appropriate point in the CT image volume (Figure 3). Typically this target coordinate will be the center of one of the pom-poms on the phantom, however the user is free to try to have the robot's needle reach additional targets if desired.

Once the target has been defined in the *ICS* of the CT volume, the user can instruct the LEGO robot to "execute the biopsy", that is, to have the robotic arm and needle move so that the needle tip reaches the target. Since the physical relationship between the LEGO robot and the phantom is predefined, the rigid transformation that transforms image coordinates into coordinates in the patient (robot) coordinate system within which the robot operates is approximately constant for all executions of the basic tutorial. The target coordinate in the *ICS* is therefore transformed into a target coordinate in the *PCS* by multiplying it by a hard-coded registration matrix. This registration matrix was the result of running a rigid landmark registration algorithm using corresponding pairs of image and patient (robot) coordinates from 72 fiducial points distributed over the phantom as input. Thus in the basic tutorial no registration procedure is explicitly performed by the user.

Finally, in order to "execute the biopsy" and move the robot's needle to the target coordinate in the *PCS* we need to calculate the number of rotations that each of the three motors must execute (i.e. we must solve the robot's inverse kinematics problem). We identified three key angles on the robot's frame and

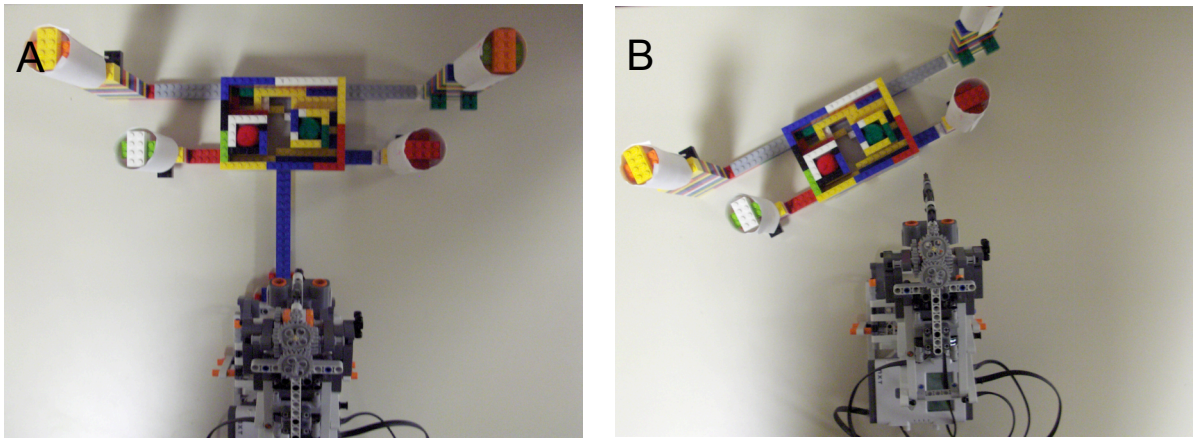


Figure 2 Placement of the phantom relative to the LEGO robot for the A) basic tutorial; B) advanced tutorial.

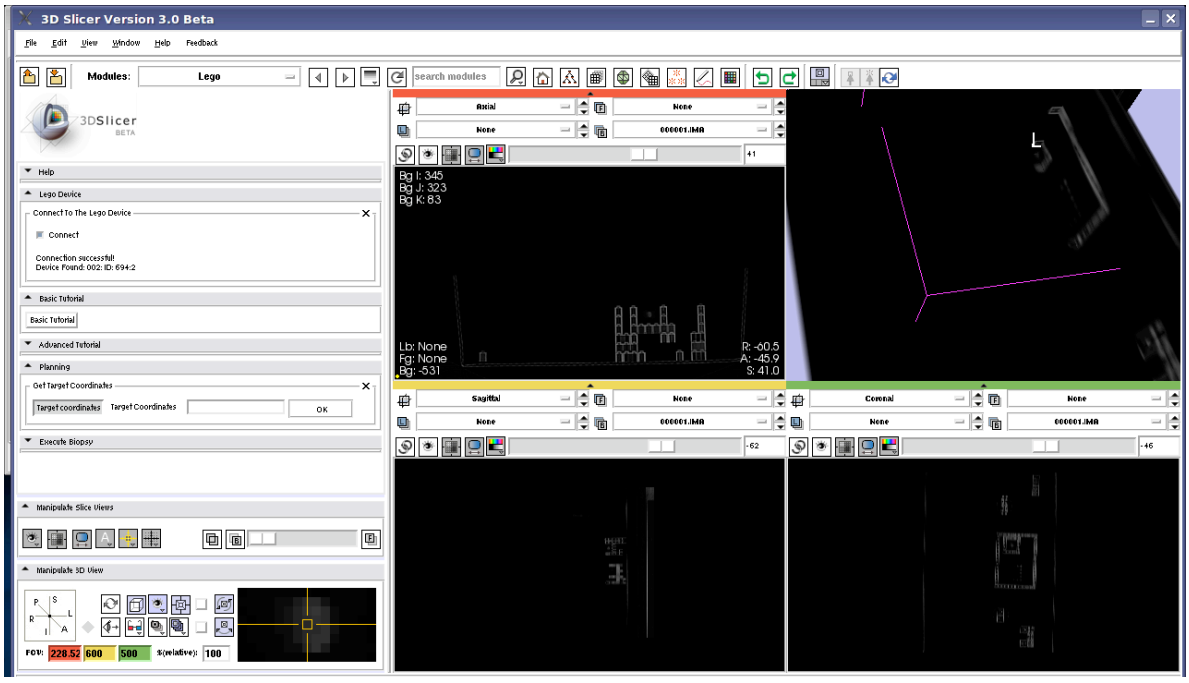


Figure 3 Screenshot of the 3D Slicer tutorial module during the preoperative planning (target selection) phase. The user selects the target by clicking on the appropriate point in the CT volume. Note that both the LEGO bricks and the pom-pom targets show up well in the CT volume.

empirically determined interpolating polynomial relationships between these angles and the degrees of rotation of each motor. The angle values that will lead to the needle hitting the target are found geometrically; these angles are then converted into motor rotations using the interpolating polynomial relationships described above. In this section of the tutorial the user physically views the robot's needle reach the target. The target coordinate in the patient (robot) coordinate system is displayed, as is the coordinate in the *PCS* that the needle tip actually reached. The later is calculated by geometrically solving the LEGO robot's forward kinematics problem using the interpolating polynomials and the degrees of motor rotations that were actually executed by each motor (which are close to, but not exactly, the degrees of motor rotation commanded, due to slight imprecision within the servo motors).

2.6 The advanced tutorial introduces registration

The advanced tutorial builds upon the concepts demonstrated in the basic tutorial by introducing the topic of registration to the user. The user can place the phantom in any position with respect to the LEGO robot as long as 1) the tip of the robot's needle can reach both pom-poms and 2) the phantom's four registration pillars will be within range of the LEGO robot's ultrasonic sensor as its robotic arm scans from side to side. The phantom placement guide shows one example of how to place the phantom relative to the LEGO robot so that these two conditions are met, however the user is encouraged to repeat the advanced tutorial section with different phantom positions.

The steps of the advanced tutorial are equal to those of the basic tutorial with the exception of the additional registration section. The user first loads the CT volume, establishes the connection between the LEGO robot and the 3D Slicer tutorial module, and selects a target coordinate in the image coordinate system by clicking on a point in the CT volume.

Figure 4 illustrates the registration process used. We perform registration between the image coordinate system, or *ICS*, and the patient (robot) coordinate system, or *PCS* (shown in Figure 4A), using a rigid landmark registration algorithm on the *ICS*-*PCS* pairs of coordinates corresponding to the eight fiducial points on the phantom. The locations of these fiducials are shown in Figure 4B: four are centered on the front side at the top surface of each registration pillar, and four are located on the top surface of each corner of the phantom's central box. We followed the suggestions of West *et al.* when choosing the number and location of the fiducials: 1) the fiducial points are in a nonlinear configuration, 2) their center lies close to the two pom-pom targets (the critical regions), 3) the distance between them is relatively large and 4) we use many of them [25].

In theory, one could use the ultrasonic sensor to generate the lengths of the four line segments from the origin of the *PCS* to the closest edge of the registration pillars along with the angles between them and the y-axis of the *PCS*. Together with prior knowledge of the phantom's structure, this information would be sufficient to calculate the fiducial coordinates in the *PCS* regardless of the position and orientation of the phantom with respect to the LEGO robot. Unfortunately though, the accuracy of the ultrasonic sensor (0 to 255 cm ± 3 cm [26]) is not high enough for these required distance measurements to be sufficiently accurate.

Instead, our requirement that the robot's needle must be able to reach both pom-pom targets constrains both the distance *d* from the origin of the *PCS* to the phantom (shown in Figure 4C) and the orientation of the phantom relative to the LEGO robot. If we define α as the angle between the y-axis of the *PCS* and the line perpendicular to the front of the phantom that passes through the origin of the *PCS*, the eight fiducial coordinates in the *PCS* can be calculated using α and prior knowledge of the phantom's structure.

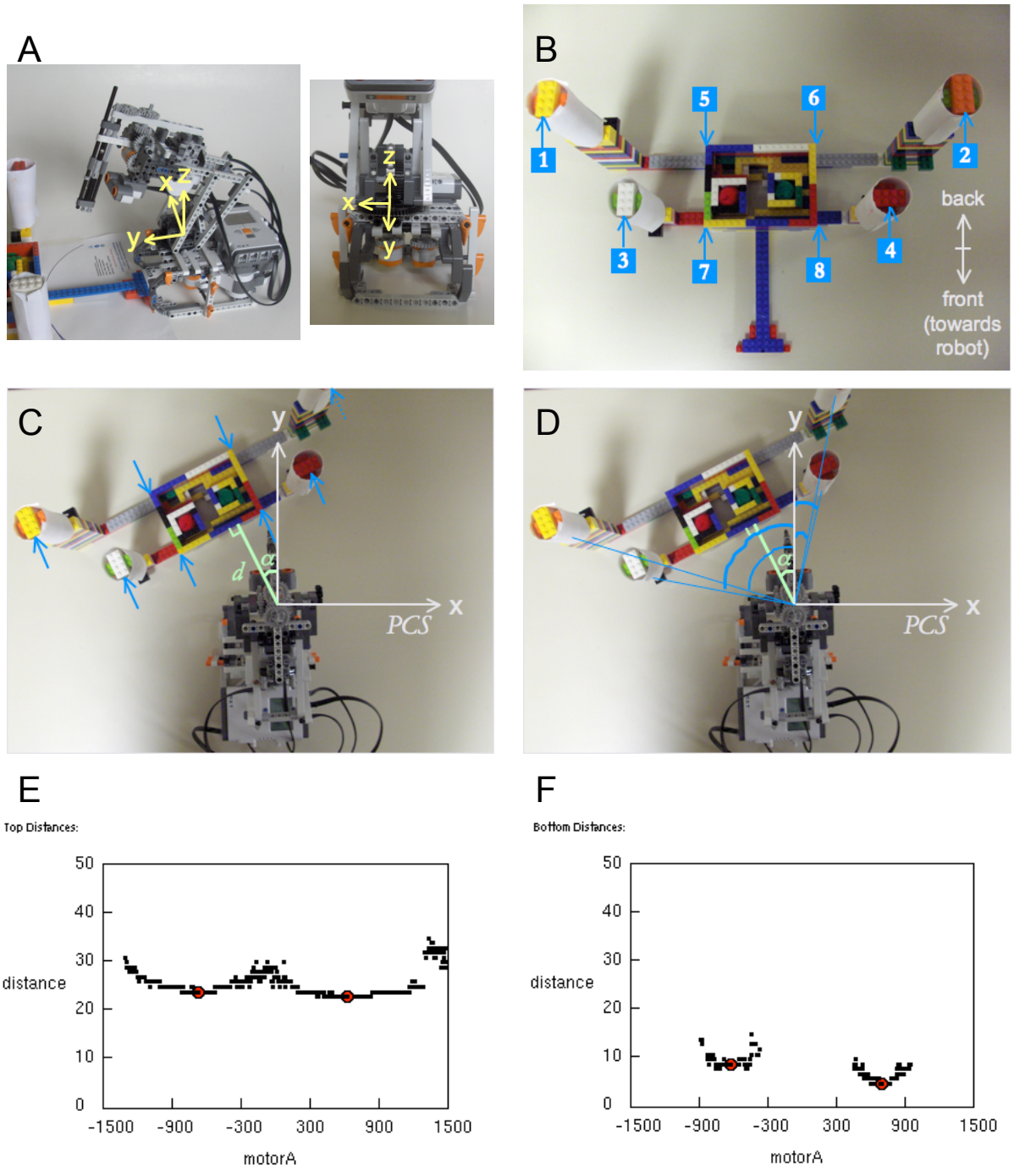


Figure 4 Registration in the advanced tutorial. A) Two views of the patient (robot) coordinate system, or *PCS*; B) The eight fiducial points on the phantom; C) Knowledge of the distance *d* and the angle α are sufficient along with knowledge of the phantom's structure to calculate the eight fiducial coordinates in the *PCS*; D) The average of the four angles between the y-axis of the *PCS* and the line segments from the origin of the *PCS* to the registration pillars approximates α; E, F) Sample distance profiles of *motorA* (the number of degrees through which motor A has turned to move the robotic arm from left to right) versus the distances (cm) returned by the ultrasonic sensor for the top and bottom swipes, respectively. The red dots show the automatically detected local minima.

We approximate this angle α by averaging the four angles between the y-axis of the *PCS* and the line segments from the origin of the *PCS* to the closest edge of the registration pillars (Figure 4D). These four angles are found by using the ultrasonic sensor to generate a distance profile at two different vertical levels: a "top" profile for the tall registration pillars at the back of the phantom and a "bottom" profile for the shorter registration pillars at the front. The robotic arm scans slowly from left to right while continuously polling the ultrasonic sensor for distance measurements and motor A's rotation sensor for the current number of degrees rotated. Figures 4E and 4F show examples of these profiles. Each of the two local minima of each profile corresponds to the number of degrees by which motor A must be rotated so that the center of the ultrasonic sensor faces the associated registration pillar. These *motorA* values can be transformed into the four angles required to calculate α using additional interpolating polynomials that were empirically determined.

Both the top and bottom distance profiles have a characteristic shape that is relatively constant between trials of the advanced tutorial and merely shifts from light to right with different legal positionings of the phantom. It is therefore simple to automatically find the two local minima on each profile. Although a very wide median filter is first used on both scans to reduce noise, a different algorithm is used to find the local minima on the top and bottom profiles in order to take advantage of their different characteristic shapes. Following the median filter, the top profile typically has only two sections that are local minima, that is, there are two stretches in the array where a section of equal distances is preceded and succeeded by sections of larger distances. The middles of these two such sections represent the local minima. If there is only one local minimum (for example if the phantom is positioned at a sharp angle to the robot so that only one and a half curves are present in the top profile), the one local minimum is found as described above and the other is extrapolated using it and the width of the one complete curve. The first and last local minima are chosen if three or more local minima are found, which gives reasonable results in our experience. The bottom profile is even more consistent than the top profile. Two disconnected curves result from filtering the bottom profile with the median filter, and so the middles of these curves give an excellent approximation to the local minima. We do not have to accommodate the possibility of there being one or 3+ curves because this virtually does not happen so long as the user positions the phantom according to the tutorial instructions.

Thus in order to find the fiducial coordinates in the patient (robot) coordinate system the user simply instructs the 3D Slicer tutorial module to scan for them. The two distance profiles are displayed with the automatically detected local minima highlighted, as are the eight calculated fiducial coordinates in the patient (robot) coordinate system. If there was a problem with the scan or with the automatic local minima detection the user has the opportunity to repeat the scan until satisfied with the results.

In the advanced tutorial, tutorial participants select the eight fiducial coordinates in the image coordinate system by clicking on the points in the CT volume of the phantom. The registration matrix is then found using a landmark rigid registration algorithm on the eight pairs of corresponding fiducials in the *ICS* and the *PCS*. Once the target in the *PCS* is determined by multiplying the target in the *ICS* by the registration matrix, the robot "executes" the biopsy in the same way as in the basic tutorial. Displayed to the user following the biopsy is the registration matrix that was used, the target in the *PCS* and the final needle position in the *PCS*.

2.7 Assessment of targeting accuracy

The goal of this project is to create an educational, accessible and practical tutorial, not to create a needle biopsy system with sub-millimeter accuracy. However, targeting accuracy remains important because

continued failure of the robot to hit the target will only frustrate and distract the user. Keeping this in mind, the best way to evaluate targeting accuracy is to determine the percentage of trials in which the LEGO robot's needle reaches the pom-pom target.

There are two sources of error that accumulate to form the *total targeting error*, defined as the Euclidean distance in the patient (robot) coordinate system between the true target on the phantom (the pom-pom center) and the final position of the robot needle tip. First, we define *registration error* as the Euclidean distance in the *PCS* between the true target coordinate and the target in the *PCS*, that is, the coordinate that the robot aims at following registration. Registration error includes the user's error in selecting the target in the image coordinate system as well as the target registration error (TRE), which in our case is the Euclidean distance between the point in the *PCS* that actually corresponds to the user-selected target in the *ICS* and the target in the *ICS* multiplied by the registration matrix (see [25] for additional information about target registration error and its multiple components). Second, we define *execution error* as the distance between the point in the *PCS* that the robot aims for and the final position of the robot needle tip. Execution error comprises inexact initial positioning of the robot, error in the polynomial relationships between robot beam angles and motor rotations (composed of both error in the interpolation and measurement error in the sample points used), differences between the degrees of motor rotation commanded and the degrees executed, and finally any shakiness in the robot's movement.

In order to evaluate the total targeting error of our tutorial system, we purchased a LEGO Mindstorms NXT kit and a LEGO Deluxe Brick Box and built the tutorial robot and phantom described above. For each of the two targets in the image coordinate system corresponding to the centers of the red and green pom-pom targets (as shown in Figure 1B), we completed the basic tutorial five times. The LEGO robot was manually returned to the correct initial position between trials in cases when the robotic arm did not move completely back to the initial position following the biopsy. We considered the target to be "hit" if the final needle position fell into the small 2.4 cm x 2.4 cm x 1.9 cm box surrounding the pom-pom. The coordinates of the actual centers of both pom-pom targets in the *PCS* were determined by counting the number of LEGO units in all three dimensions between the pom-pom centers and the origin of the *PCS*; the accuracy of these true target coordinates is extremely high due to the precision of the LEGO manufacturing process. We recorded the displayed target in the *PCS* (the coordinate in the *PCS* that the robot was aiming for) and determined the final needle position in the *PCS* by visual inspection using the studs on the phantom's LEGO bricks as markers (we estimate that the accuracy of this visual inspection is approximately 0.25 LEGO units, which equals 0.2 cm). The total targeting error, registration error and execution error were calculated as described above, as were the means for the "red" trials, the "green" trials and the total number of trials.

Similarly, we performed an accuracy assessment of the advanced tutorial section using three different phantom positions corresponding to $\alpha = 0^\circ$, $\alpha = -14.5^\circ$ (a typical angle where the phantom is to the left of the robot) and $\alpha = 25^\circ$ (an extreme angle where the phantom is to the right of the robot). Once again, for each phantom position we targeted the centers of the red and green pom-poms in five trials. As in the accuracy assessment of the basic tutorial, we repeatedly used the same target coordinates for the red and green pom-poms in the *ICS* and performed a manual reset of the robotic arm between trials if it did not return exactly to the initial position. We also had the tutorial software output the value of α determined by the registration algorithm specifically for this accuracy assessment. The true coordinates of the centers of the pom-pom targets were calculated algebraically using the true α value, the estimated value of d and knowledge of phantom's structure. The target in the *PCS* after registration was recorded from the display on the screen. Finally, we calculated the final needle position in the *PCS* using the distance in all three dimensions between the needle tip and the center of the pom-pom (once again visually determined with an accuracy of approximately 0.2 cm), the true value of α, the estimated value of d and knowledge of the

phantom's structure. We counted the number of target "hits" and calculated the total targeting error, registration error and execution error as well as the "red", "green" and total means.

3 Results

3.1 The IGT and medical robotics tutorial is highly accessible

Our tutorial is unique because it provides education on advanced topics in image-guided therapy and medical robotics while remaining accessible and hands-on. One of the most important requirements of our tutorial is accessibility, as it will not be widely used if the required materials are not widely available and affordable. All tutorial materials not provided by us are available in both stores and online (the LEGO Mindstorms NXT and Deluxe Brick box are available from [27], pom-poms can be found in craft stores or from [28]). Although future plans to provide the 3D Slicer tutorial module for Windows and Macintosh systems will increase accessibility, most users in university computer science and engineering departments will have access to Linux computers and would therefore be able to use our tutorial. The total cost of our tutorial (using the sources in [27] and [28] and ignoring the negligible cost of paper and tape) is $308.47 USD plus applicable taxes and shipping charges. This cost is well below our goal of $500 USD and represents an affordable cost to university departments. In addition, many computer science and engineering departments already own LEGO Mindstorms NXT kits because they themselves run educational programs for younger students. It is clear that the tutorial remains accessible to departments, surgeons and any other individuals interested in image-guided therapy and medical robotics.

3.2 Assessment of targeting accuracy

The results of the accuracy assessment are summarized in Table 3: please recall our previous argument that the percentage of times in which the target is hit is the best indicator of sufficient accuracy, as this is the goal that the user will see. In addition, although total targeting error is composed of registration error and execution error, the sum of registration error and execution error is not expected to equal the total targeting error because execution error may be partially compensated for by registration error in an opposing direction.

Our results show that the basic tutorial is highly reliable: the target was hit in 100% of the trials, while the mean total targeting error is relatively low. The registration error is minor as expected since 72 fiducials were used to create the registration matrix used in the basic tutorial, while the execution error is much larger. Thus the total targeting error in the basic tutorial is primarily due to execution error: we can calculate the goal target with relatively high accuracy, but have difficulty getting the robot's needle there. Errors related to misplacement of the robotic arm initially and differences between the degrees of motor rotation commanded and those executed are expected to be small and are not easily controlled. Therefore the main factors influencing execution error, and therefore total targeting error, are errors in the relationships between beam angles and motor rotations and shakiness of the robot's movement.

100% of the targets were hit in the advanced tutorial for typical values of α, while the percentage of times in which the target was hit decreases with extreme values of α. However, even for $\alpha = 25^{\circ}$ the percentage of times in which the target was hit was a reasonable 70%, meaning that the user could potentially have to repeat the advanced tutorial a couple of times at most. As expected, registration error in the advanced

Table 3 Results of the accuracy assessment for the basic and advanced tutorials. The total targeting error, registration error and execution error values are Euclidean distances as described in Section 2.7, while the absolute error in α is the difference between the known value of α and that determined by the registration procedure in the advanced tutorial. Means in the "Red target" and "Green target" columns are means over five trials, whereas means in the "Both targets" columns are means over all ten trials. All values are rounded to the nearest tenth of a centimeter or degree.

Basic tutorial:

	Red target	Green target	Both targets
# Times target hit	**5/5**	**5/5**	**10/10**
Mean total targeting error	1.2 cm	1.2 cm	1.2 cm
Registration error	0.2 cm	0.2 cm	0.2 cm
Mean execution error	1.2 cm	1.2 cm	1.2 cm

Advanced tutorial:

	$\alpha = 0^\circ$			$\alpha = -14.5^\circ$			$\alpha = 25^\circ$		
Mean absolute error in α	1.3°			1.3°			1.9°		
	Red target	Green target	Both targets	Red target	Green target	Both targets	Red target	Green target	Both targets
# Times target hit	**5/5**	**5/5**	**10/10**	**5/5**	**5/5**	**10/10**	**3/5**	**4/5**	**7/10**
Mean total targeting error	1.4 cm	1.3cm	1.3 cm	1.2 cm	1.0 cm	1.1 cm	1.6 cm	1.9 cm	1.7 cm
Mean registration error	0.4 cm	0.6 cm	0.5 cm	0.7 cm	0.4 cm	0.6 cm	0.7 cm	0.8 cm	0.8 cm
Mean execution error	1.6 cm	1.2 cm	1.4 cm	1.4 cm	0.9 cm	1.2 cm	1.8 cm	1.9 cm	1.8 cm

tutorial is higher than registration error in the basic tutorial. This is due to both the fact that we use eight fiducials rather than 72 and increased fiducial localization error for both the fiducials in the image coordinate system and the patient (robot) coordinate system. Users may not be accurate in clicking on the fiducials in the *ICS*, whereas the fiducials found in the *PCS* are influenced by error in finding the local minima of the *motorA* versus ultrasonic distance curves and in converting the *motorA* values of the detected local minima into robot beam angles. In addition, the distance between the phantom and the origin of the *PCS* might not equal the estimated distance *d* used, the average of the registration pillar angles might not be exactly equal to the true value of α and, even if they could be found exactly, the local minima might not correspond exactly to the registration pillar angles. Yet despite these multiple sources of error, the mean absolute error in α (which encapsulates most of the sources of registration error described above) was in all cases less than two degrees. Finally, as in the basic tutorial, execution error is larger than registration error. Since execution error is the Euclidean distance between the actual position of the target and the coordinate that was aimed for after registration, the differences in mean execution error between the basic and advanced tutorials and between trials of different α values in the advanced tutorial have nothing to do with the quality of the registration. Instead, they are primarily related to the fact that the accuracies of the interpolated motor rotation / robot beam angle relationships and the shakiness of the robotic arm both vary when the robotic arm moves to targets corresponding to different α values (where α in the basic tutorial is 0° by definition).

In passing, please note that the error values shown in Table 3 are not completely accurate. In the basic tutorial data, the total targeting error and the execution error are based on visual inspection of the final needle position and will invariably contain some inaccuracies. However, registration error can be calculated with high accuracy because the actual coordinates of the pom-pom centers and the target that the robot was aiming for in the *PCS* are known. In the advanced tutorial, the actual coordinates of the pom-pom centers are calculated using the estimate for *d* and will therefore have some error. The final needle position is once again based on visual inspection but also has additional error compared to the basic tutorial because the estimate for *d* is used in its calculation. Therefore in the advanced tutorial data, error is now introduced into the measurements of registration error, and error is increased compared to the basic tutorial in the measurements of total targeting error and execution error.

4 Discussion

Currently it is next to impossible for newcomers to IGT and medical robotics to find hands-on educational tools that encapsulate the true nature of the field. Our tutorial on image-guided therapy and medical robotics can be used by anyone with elementary knowledge of algebra to understand the fundamental steps of such procedures in a practical way. We foresee the use of our tutorial as part of labs in undergraduate medical informatics courses, by beginning graduate students in the field, by medical students as an introduction to image-guided therapies and medical robotics, by high-school students in workshops encouraged to increase interest in computer science and engineering, and by various others interested in IGT and medical robotics.

That being said, there is an important fundamental difference in approach between this work and that of other LEGO educators, such as the competitions organized by CISSRS. Whereas the focus of the CISSRS competitions is to get students excited in computer science and engineering, the goal of our tutorial is to provide education on the main steps of a typical IGT or medical robotics procedure. In the former case creativity in robotic design and in programming are emphasized, while in the current version of our tutorial the robot design is preset and there is no programming done by the user. This is not to say

that our tutorial system cannot be used in student competitions or should not inspire interest in image-guided therapy and medical robotics. Instead, our tutorial would be an excellent introduction to IGT and medical robotics that students in such competitions could complete before working on their own creations. However, our tutorial has primarily been designed for use by more "serious" students of IGT and medical robotics in a classroom or laboratory setting.

Future work in this project includes enhancing the 3D Slicer tutorial module so that it can be run under Windows and Macintosh systems in addition to the Linux platform. This will increase accessibility and widespread use of our tutorial. In addition, we plan to enhance the navigation provided to the user in the "execute biopsy" phase by providing real-time visualization of the current needle position overlaid onto the CT volume of the phantom. Finally, we would like to encourage more interactivity into the tutorial by having the user play with different targets, registration algorithms, phantom designs and more.

5 Conclusions

In this paper we have described a tutorial on image-guided therapy and medical robotics using the LEGO Mindstorms NXT robotics kit and 3D Slicer. This tutorial both uses open-source software packages and will be supplied in an open-source fashion itself in order to meet the currently unsupplied need for accessible and hands-on educational tools in IGT and medical robotics. In addition, we describe a C++ library that can be used to control a LEGO Mindstorms NXT robot from a personal computer.

6 Acknowledgement

This publication was made possible by grant numbers 5U41RR019703, 5P01CA067165 and 5U54EB005149 from NIH. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the NIH. This study was also in part supported by NSF 9731748 and CIMIT. Thanks also to Terry Peters, Ph.D of the Robarts Research Institute and The University of Western Ontario; Steven Canvin of The LEGO Group; Cory Walker, NXT++ developer; G. Wade Johnson, Device::USB developer; and Steve Pieper, Ph.D, Haiying Liu, Junichi Tokuda, Ph.D, Christoph Ruetz and Philip Mewes of the Surgical Planning Laboratory.

A Functions of our robotic control library

Below is a list of the public functions offered by our C++ library enabling control of a LEGO Mindstorms NXT robot. More extensive documentation on the use of these functions has been included with this submission.

```
// Constructor and destructor
vtkLegoUSB();
~vtkLegoUSB();

// Open and close the USB connection to the LEGO robot
int OpenLegoUSB();
int CloseLegoUSB();
```

```
// Set up the sensors before their use
void SetSensorLight(int port, bool active);
void SetSensorTouch(int port);
void SetSensorSound(int port, bool dba);
void SetSensorUS(int port);
void SetUSOff(int port);
void SetUSSingleShot(int port);
void SetUSContinuous(int port);
void SetUSEventCapture(int port);
void SetUSContinuousInterval(int port, int interval);

// Read from the sensors
int GetLightSensor(int port);
bool GetTouchSensor(int port);
int GetSoundSensor(int port);
int GetUSSensor(int port);

// Move and stop the motors
void SetMotorOn(int port, int power);
void SetMotorOn(int port, int power, int tachoCount);
void MoveMotor(int port, int power, int tachoCount);
void StopMotor(int port, bool brake);

// Read the current degrees of motor rotation from the motors
int GetMotorRotation(int port, bool relative);
void ResetMotorPosition(int port, bool relative);

// Play a tone on the LEGO robot
void PlayTone(int frequency, int duration);

// Get the connection status of the LEGO robot
char * GetStatus();

// Get information about the LEGO robot
char * GetDeviceFilename();
int GetIDVendor();
int GetIDProduct();

// Send direct commands to the LEGO robot – advanced users only
void SendCommand (char * outbuf, int outbufSize, char * inbuf, int inbufSize);
// Get the LS status of the LEGO robot – advanced users only
int LSGetStatus(int port);
```

B Example use of our robotic control library

In order to use our library to control a LEGO Mindstorms NXT robot, simply include `NXT_USB.h`. Installation instructions are provided in the README file included with this submission.

```
#include "NXT_USB.h"

int main (int argc, char* argv[])
{
    // Create the NXT_USB object
    NXT_USB* legoUSB = new NXT_USB();

    // Open the connection to the LEGO robot
```

```
    int open = legoUSB->OpenLegoUSB();

    // Return if we cannot open the connection
    if (open == 0)
    {
        std::cerr << "Could not connect: " << legoUSB->GetStatus() << std::endl;
        return 1;
    }

    // Setup:  the touch sensor should be attached to input port 1
    legoUSB->SetSensorTouch(IN_1);

    // Move the motor attached to output port A (by 180 degrees at +50% power) once
    // the button on the touch sensor has been pressed
    while (!legoUSB->GetTouchSensor(IN_1)) {};
    legoUSB->SetMotorOn(OUT_A, 50, 180);

    // Close the connection to the LEGO robot
    legoUSB->CloseLegoUSB();
    delete legoUSB;

    return 0;
}
```

References

[1] L. Ibanez and W. Schroeder. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, 2003.

[2] W. Schroeder, K. Martin, and W. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 2nd ed*. Kitware, Inc. ISBN 1-930934-12-2. Prentice-Hall, Old Tappan, N.J., 1998.

[3] K. Gary, L. Ibanez, S. Aylward, D. Gobbi, M.B. Blake, and K. Cleary. IGSTK: An open source software toolkit for image-guided surgery. *Computer*, 39(4):46-53, 2006.

[4] N. Hata *et al*. Software Engineering Methods for Multicenter Clinical Application of Computer-enhanced MR-guided Therapies. In *Proceedings of the 10th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI'07)*, 2007. To appear.

[5] National Alliance for Medical Image Computing. Training, available at http://wiki.na-mic.org/Wiki/index.php/Training:Main, 2007.

[6] H. Liu and N. Hata. Slicer User Training 101: IGT Edition, available at http://wiki.na-mic.org/Wiki/images/8/89/Igt-tutorial-updated.ppt, 2006.

[7] D. Gering, A. Nabavi, R. Kikinis, N. Hata, L. O'Donnell, W. Eric L. Grimson, F. Jolesz, P. Black, and W. Wells III. An Integrated Visualization System for Surgical Planning and Guidance Using Image Fusion and an Open MR. *Journal of Magnetic Resonance Imaging,* 13(6): 967-975, 2001.

[8] M. Resnick. Behavior Construction Kits. *Communications of the ACM,* 36(7):64-71, 1993.

[9] T.H. Labella, M. Dorigo, and J.L. Deneubourg. Efficiency and Task Allocation in Prey Retrieval. In *Proceedings of the Frist International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, Lecture Notes in Computer Science, 3141:274-289, 2004.

[10] C. Bartneck and J. Hu. Rapid Prototyping for Interactive Robots. In *Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8)*, 2004.

[11] F. Klassner. A Case Study of LEGO Mindstorms'™ Suitability for Artificial Intelligence and Robotics Courses at the College Level. *ACM SIGCSE Bulletin,* 34(1):8-12, 2002.

[12] A.N. Kumar. Using Robots in an Undergraduate Artificial Intelligence Course: An Experience Report. In *Proceedings of the 31st Annual ASEE/IEEE Frontiers in Education Conference*, 2001.

[13] D. Opplinger. Using First Lego League to Enhance Engineering Education and to Increase the Pool of Future Engineering Students. In *Proceedings of the 32nd Annual ASEE/IEEE Frontiers in Education Conference*, 2002.

[14] H.H. Lund and L. Pagliarini. RoboCup Jr. with LEGO MINDSTORMS. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1(1):813-819, 2000.

[15] O. Gerovich, R.P. Goldberg, and I.D. Donn. From Science Projects to the Engineering Bench. *IEEE Robotics & Automation Magazine*, 10(3):9-12, 2003.

[16] The LEGO Group. NXT Technology Overview, available at http://mindstorms.lego.com/eng/Overview/default.aspx, 2007.

[17] B. Bagnall. *Maximum Lego NXT: Building Robots with Java Brains*. ISBN 0973864915. Variant Press, Winnipeg, M.B., Canada, 2007.

[18] Not eXactly C is available for download at http://bricxcc.sourceforge.net/nbc

[19] Bricx Command Center is available for download at http://bricxcc.sourceforge.net

[20] LiNXT is available for download at http://sourceforge.net/projects/linxt

[21] The LEGO Group. Mindstorms NXT'REME, available at http://mindstorms.lego.com/Overview/NXTreme.aspx, 2007.

[22] NXT++ is available for download at http://nxtpp.sourceforge.net

[23] Device::USB is available for download at http://search.cpan.org/~gwadej/Device-USB-0.21

[24] libusb is available for download at http://libusb.sourceforge.net

[25] J.B. West, M.J. Fitzpatrick, S.A. Toms, C.R Maurer, and R.J. Maciunas. Fiducial Point Placement and the Accuracy of Point-based, Rigid Body Registration. *Neurosurgery*, 48(4):810-817, 2001.

[26] The LEGO Group. LEGO Mindstorms User Guide, 2006 (included with the purchase of a LEGO Mindstorms NXT kit)

[27] The LEGO Mindstorms NXT robotics kit (#8527) and the LEGO Deluxe Brick Box (#6167) are available for purchase at http://shop.lego.com

[28] Pom-poms are available for purchase at http://www.amazon.com/PAC-1859614-Poms-ClassPack-Sizes/dp/B0006HXNRY/ref=sr_1_12/104-9511701-9925562?ie=UTF8&s=office-products&qid=1187791961&sr=8-12

Tagged Volume Rendering of the Heart: A Case Study

Release 1.1

Dan Mueller[1]

July 17, 2007

[1]Queensland University of Technology, Brisbane, Australia

Abstract

This is a companion paper describing the process and parameters for tagged volume rendering of the heart. We firstly review the relevant concepts and consider the problem of visualising the coronary arteries from computed tomography angiography (CTA) images. We then discuss the implementation using the SharpImage prototyping environment. Finally, we list the set of commands capable of reproducing our results using the accompanying dataset.

Keywords: *tagged volume rendering, heart, coronary arteries, ITK, SharpImage*

Contents

1 Introduction **2**

2 Implementation **3**
2.1 Vessel Segmentation 3
2.2 Fragment Shader 5

3 Commands **7**
3.1 Orient and Resize 7
3.2 Speed Function 7
3.3 Initial Contour 8
3.4 Fast Marching 8
3.5 Vesselness 9
3.6 Combine Tags 9
3.7 Histogram 9
3.8 Volume Render 9

4 Results **10**

5 Conclusion **11**

1 Introduction

For diagnostic and treatment planning purposes, radiologists and surgeons require means to visualise the coronary arteries from computed tomography angiography (CTA) images. However, this task is non-trivial for various reasons: (a) unwanted structures (such as the thoracic cage) clutter the regions of interest; (b) the contrast agent highlights the coronary arteries (as desired), but also portions of the ventricles, atria, aorta, and pulmonary arteries, and (c) the coronary arteries can exhibit high-intensity calcification artefacts. Clinicians currently have a number of tools at their disposal, including: thin slab maximum intensity projection [1], curved planar reformatting [3], and direct volume rendering (DVR).

In the case of direct volume rendering, the dataset is segmented during the rendering process ('online') via a lookup table ('transfer function'). Various online segmentation algorithms for DVR have been discussed in the literature [8, 4, 12]. Kniss et al. [4] advocated the use of two dimensional transfer functions dependent on pixel value and gradient magnitude (termed 'value-magnitude' transfer functions), in which the user interactively specifies the function using a number of widgets. This method is simple, fast and effective for basic datasets; yet, it is insufficient for delineating objects of interest sharing similar characteristics. Tzeng et al. [12] proposed an approach using an online machine learning algorithm (such as an artificial neural network or support vector machine), with the current pixel value, gradient magnitude, location, and neighbourhood values as inputs. Because of their online nature, both of these methods are constrained by the graphics hardware on which they are implemented.

A number of more recent methods combine both offline and online segmentation. Hadwiger et al. [2] proposed tagged volume rendering which uses a number of *a priori* segmented binary masks ('tags'), each assigned separate transfer functions. Kniss et al. [5] introduced a probabilistic method which decouples classification from colour-mapping, simplifying the transfer function interface. Weber et al. [13] proposed the use of contour-trees for classifying objects based on topology (the trees are computed and simplified offline before being uploaded the the graphics hardware). Of these methods, tagged volume rendering is the most flexible as it does not restrict the choice of segmentation algorithm.

In this paper we apply tagged volume rendering to the problem of visualising the coronary arteries. We utilise Hessian-based line filters for segmenting the coronary arteries [9] and use the Fast Marching active contour method [10] for segmenting the pericardial cavity. A suitable speed function is presented for controlling the expansion of the active contour. The exact set of SharpImage [7] commands for reproducing the results on the accompanying dataset are given. For your reference, a pre-print of the main MICCAI paper can be found here: http://eprints.qut.edu.au/archive/00008421/

2 Implementation

This section details the implementation of the method. It requires built versions of ITK 3.2, ManagedITK 3.2.0.3 [6], and SharpImage 0.9.2 [7]. Before tackling the case study, we must extend the SharpImage environment to handle vessel segmentation and implement a fragment shader for tagged transfer functions. Pre-built versions are available if you wish to skip this step (see below).

2.1 Vessel Segmentation

SharpImage does not have an explicit vessel segmentation command, however it is quite easy to add one. The first step is to wrap the required filters using a ManagedITK external project (see `source/vesselness/CMakeLists.txt`):

```
PROJECT(WrapVesselness)

# Find required packages
FIND_PACKAGE(ITK REQUIRED)
FIND_PACKAGE(ManagedITK REQUIRED)

# Use required packages
INCLUDE(${ITK_USE_FILE})
INCLUDE(${MANAGED_ITK_USE_FILE})

# Wrap the project
BEGIN_MANAGED_WRAP_EXTERNAL_PROJECT("Filtering" "Vesselness")
  SET(MANAGED_WRAPPER_OUTPUT "${CMAKE_BINARY_DIR}")
END_MANAGED_WRAP_EXTERNAL_PROJECT()
```

We need to wrap two filters, `itkHessianRecursiveGaussianImageFilter` and `itkHessian3DToVesselnessMeasureImageFilter`:

```
# Begin the wrapping
WRAP_CLASS("itk::HessianRecursiveGaussianImageFilter")

  # Wrap the class for INT and REAL types
  WRAP_IMAGE_FILTER_INT(1)
  WRAP_IMAGE_FILTER_REAL(1)

  # Wrap the Sigma property
  BEGIN_MANAGED_PROPERTY("Sigma" SET)
    SET(MANAGED_PROPERTY_SUMMARY   "Set the Gaussian variance (in image spacing).")
    SET(MANAGED_PROPERTY_TYPE      "double")
    SET(MANAGED_PROPERTY_SET_BODY  "m_PointerToNative->SetSigma( value );")
  END_MANAGED_PROPERTY()

  # Wrap the NormalizeAcrossScale property
  BEGIN_MANAGED_PROPERTY("NormalizeAcrossScale" GETSET)
    SET(MANAGED_PROPERTY_SUMMARY   "Get/set whether normalization should occur.")
    SET(MANAGED_PROPERTY_TYPE      "bool")
    SET(MANAGED_PROPERTY_GET_BODY  "return m_PointerToNative->GetNormalizeAcrossScale( );")
    SET(MANAGED_PROPERTY_SET_BODY  "m_PointerToNative->SetNormalizeAcrossScale( value );")
  END_MANAGED_PROPERTY()

# End the wrapping
END_WRAP_CLASS()
```

```
# Begin the wrapping
WRAP_CLASS("itk::Hessian3DToVesselnessMeasureImageFilter")

  # Wrap the template arguments
  WRAP_TEMPLATE("${ITKM_F}" "${ITKT_F}")

  # Wrap the Alpha1 property
  BEGIN_MANAGED_PROPERTY("Alpha1" GETSET)
    SET(MANAGED_PROPERTY_SUMMARY   "Get/set alpha1.")
    SET(MANAGED_PROPERTY_TYPE      "double")
    SET(MANAGED_PROPERTY_GET_BODY  "return m_PointerToNative->GetAlpha1( );")
    SET(MANAGED_PROPERTY_SET_BODY  "m_PointerToNative->SetAlpha1( value );")
  END_MANAGED_PROPERTY()

  # Wrap the Alpha2 property
  BEGIN_MANAGED_PROPERTY("Alpha2" GETSET)
    SET(MANAGED_PROPERTY_SUMMARY   "Get/set alpha2.")
    SET(MANAGED_PROPERTY_TYPE      "double")
    SET(MANAGED_PROPERTY_GET_BODY  "return m_PointerToNative->GetAlpha2( );")
    SET(MANAGED_PROPERTY_SET_BODY  "m_PointerToNative->SetAlpha2( value );")
  END_MANAGED_PROPERTY()

# End the wrapping
END_WRAP_CLASS()
```

We point CMake at this folder, choose a build directory, and configure the project for Visual Studio 8[1]. ManagedITK should create the relevant files, including a `FinishCMake.bat` file which must be run *before* opening the solution file and compiling the project. The output will be a single executable `ManagedITK.Filtering.Vesselness.dll`.

The next step is to create a Python script for use with SharpImage (see `source/vesselness/Vesselness.py`):

```
#=========================================================================
# Module:    Vesselness.py
#=========================================================================

# Import the base script class
import ImageToImageScript
from ImageToImageScript import *

# Add CLR reference and import required libraries
clr.AddReference("ManagedITK.Filtering.Vesselness")
from itk import *

class VesselnessScript(ImageToImageScriptObject):
    # ------------------------------------------------------------------------
    Name = "Vesselness"
    Help = """Implements a Hessian-based line enhancement filter for segmenting\r\
              tubular objects. The input image must be a 3-D image."""
    Sigma = 1.0
    Alpha1 = 0.5
    Alpha2 = 2.0
    NormalizeAcrossScale = False
    # ------------------------------------------------------------------------

    def ThreadedDoWork(self):
        """ Perform the main functions of the script on a background thread. """
        try:
            # Start
```

[1] We have not tried Visual Studio Express Edition, so there may be issues if you are using that specific compiler.

```
            self.StartedWork()
            self.WriteInputName()

            # Compute hessian
            filterHessian = itkHessianRecursiveGaussianImageFilter.New( self.Input )
            self.AddEventHandlersToProcessObject( filterHessian )
            filterHessian.SetInput( self.Input )
            filterHessian.Sigma = self.Sigma
            filterHessian.NormalizeAcrossScale = self.NormalizeAcrossScale

            # Compute vesselness
            filterVesselness = itkHessian3DToVesselnessMeasureImageFilter.New( itkPixelType.F )
            self.AddEventHandlersToProcessObject( filterVesselness )
            filterVesselness.SetInput( filterHessian.GetOutput() )
            filterVesselness.Alpha1 = self.Alpha1
            filterVesselness.Alpha2 = self.Alpha2
            filterVesselness.UpdateLargestPossibleRegion( )
            filterVesselness.GetOutput( self.Output )

            # Clean up and finish
            self.DisconnectInputAndOutput()
            self.DisposeOfObject( filterHessian )
            self.DisposeOfObject( filterVesselness )
            self.WriteOutputName()
            self.FinishedWork( True )

        except Exception, ex:
            self.HandleException( ex )
            self.FinishedWork( False )
            self.Finalise( )
```

These files (`ManagedITK.Filtering.Vesselness.dll` and `Vesselness.py`) must be copied to the SharpImage Scripting folder (we recommend creating a subfolder called `SharpImage/Scripting/Custom` or similar). A pre-compiled version of the ManagedITK assembly is provided with this article if you wish to skip this step (see `results/ManagedITK.Filtering.Vesselness.dll`).

2.2 Fragment Shader

We implement a derivation of the tagged volume rendering method described by Hadwiger et al. [2]. In their original method tags represent exact objects, which requires tri-linear interpolation and complex boundary filtering to avoid artefacts and improve resolution. Our application allows us to make a different assumption: a tag is a mask guaranteeing to *include* structures of interest, but not *exactly* delineate them. We can therefore use nearest neighbour interpolation and a stack of value-magnitude transfer functions to refine the structures inside the tags (ie. a 3-D texture, similar to Svakhine et al. [11, Sec. 3]). The tagged transfer function is implemented using OpenGL Shading Language (GLSL) (see `source/fragment-shader/shader-vmt-phong.frag`):

```
//==============================================================================
// Module:    shader-vmt-phong.frag
//==============================================================================
uniform sampler3D sam3Tex0;     // Sampler for transfer function
uniform sampler3D sam3Tex1;     // Sampler for value image
uniform sampler3D sam3Tex2;     // Sampler for gradient image
uniform sampler3D sam3Tex3;     // Sampler for tags image
varying vec3 pos3Tex1;          // Current coord of value image
```

```
23 varying vec3 pos3Tex2;          // Current coord of gradient image
24 varying vec3 pos3Tex3;          // Current coord of tags image
96
97 void main()
98 {
99     // Interpolate images
100    float value = vec4( texture3D(sam3Tex1, pos3Tex1) ).a;
101    float gradmag = vec4( texture3D(sam3Tex2, pos3Tex2) ).a;
102    float tags = vec4( texture3D(sam3Tex3, pos3Tex3) ).a;
103
104    // Interpolate transfer function
105    const float fNumLayers = 4.0;
106    vec3 pos3TfAll = vec3( value, 1.0-gradmag, 0.0 );
107    vec3 pos3TfTag1 = vec3( value, 1.0-gradmag, 1.0/(fNumLayers-1.0) );
108    vec3 pos3TfTag2 = vec3( value, 1.0-gradmag, 2.0/(fNumLayers-1.0) );
109    vec4 val4TfAll  = texture3D( sam3Tex0, pos3TfAll );
110    vec4 val4TfTag1 = texture3D( sam3Tex0, pos3TfTag1 );
111    vec4 val4TfTag2 = texture3D( sam3Tex0, pos3TfTag2 );
112
113    // Adjust alpha for sampling rate
114    val4TfAll.a = adjustAlphaForSamplingRate( val4TfAll.a );
115    val4TfTag1.a = adjustAlphaForSamplingRate( val4TfTag1.a );
116    val4TfTag2.a = adjustAlphaForSamplingRate( val4TfTag2.a );
117
118    // Mix tags
119    bool light, enhance;
120    vec4 val4Mix;
121    if (val4TfAll.a == 0 && tags > 0.5)
122    {
123        // Tag 2
124        val4Mix = val4TfTag2;
125        enhance=true; light=true;
126        if (val4TfTag2.a <= 0.0) discard;
127    }
128    else if (val4TfAll.a == 0 && tags > 0.25)
129    {
130        // Tag 1
131        val4Mix = val4TfTag1;
132        enhance=true; light=true;
133        if (val4TfTag1.a <= 0.0) discard;
134    }
135    else
136    {
137        // All
138        val4Mix = val4TfAll;
139        enhance=true; light=true;
140        if (val4TfAll.a <= 0.0) discard;
141    }
142
143    // Compute the normal
144    vec4 vec4G = gradientFromCentralDifferences( );
145    vec4 vec4N = normalize( vec4G );
155
156    // Apply lighting
157    if (light)
158        gl_FragColor = val4Mix * phong( vec4N, gl_LightSource[0] );
159    else
160        gl_FragColor = val4Mix;
161 }
```

3 Commands

This section lists the SharpImage commands required to reproduce our results. You will need a good desktop computer with *at least* 2 GB RAM and a recent graphics card with *at least* 512 MB VRAM (you will receive memory allocation errors otherwise). For this experiment we used an Intel Pentium D, 2 $\times$ 3.0 GHz processors, 2 GB RAM, NVIDIA GeForce 8800 GTX, 768 MB VRAM. We assume the data has been unzipped to `C:/Temp`, along with `source/commands/A-SPHERES.txt`. The full list of commands can be found in `source/commands/commands.txt`.

3.1 Orient and Resize

Because our data came from various sources, we found it desirable to orient the images to the same anatomical position. Much of the processing is undertaken on a subsampled image, so we also resize the image at this point.

```
> Open "C:/Temp/A.mhd#SS3"
> Orient Given="RPI" Desired="ASL"
> Properties Origin=itkPoint(0,0,0)
> Save "C:/Temp/A-ASL.mhd"
> CastToF
> Resize OutputSize=itkSize(256,256,256) Interpolator="Linear"
> Save "C:/Temp/A-ASL-SMALL.mhd"
> Close
```

3.2 Speed Function

We now compute the edge-based speed function:

```
> Open "C:/Temp/A-ASL-SMALL.mhd#F3"
> CurvatureFlow TimeStep=0.1 Iterations=10
> Threshold Lower=-2000 Upper=100 OutsideValue=100
> ConnectedThreshold Lower=60 Upper=100
> Rename "Sternum"
> BinaryPixelMath Operation="Add" Input1="Threshold" Input2="Sternum"
> LevelSetSpeed OutputMinimum=0.0
> Save "C:/Temp/A-ASL-SMALL-SPEED.mhd"
> Close
```

The seed for the region growing can be anywhere in the sternum (we used [50, 90, 100]) and the parameters for the speed function were as follows:

```
Gaussian Normalize=False
Gaussian Sigma=0.5
Sigmoid Alpha=-10.0
Sigmoid Beta=10.0
```

```
Threshold Lower=-250.0
Threshold Upper=355.0
Rescale OutputMinimum=000.000
Rescale OutputMaximum=001.000
```

3.3 Initial Contour

The initial contour could be computed from a set of points or — as we prefer — from a set of spheres generated using a 3-D interface:

```
> Open "C:/Temp/A-ASL-SMALL.mhd#SS3"
> BinaryThreshold Lower=-250 Upper=2000
> CastToUC
> GenerateMaskFromFile "C:/Temp/A-SPHERES.txt"
> BinaryPixelMath Operation="Mask" Input1="Cast" Input2="Mask"
> MorphologicalOpen Operation="Binary" KernelRadius=itkSize(4,4,4)
> Save "C:/Temp/A-ASL-SMALL-INITIAL.mhd"
> Close
```

Notice the morphological opening which removes unwanted pulmonary vessels. The list of spheres is defined in `source/commands/A-SPHERES.txt`:

```
Sphere: Center=119,062,100 Radius=48
Sphere: Center=095,085,076 Radius=45
Sphere: Center=068,106,055 Radius=25
Sphere: Center=112,018,126 Radius=54
Sphere: Center=102,075,126 Radius=50
```

3.4 Fast Marching

We now evolve the active contour, extract the contour at a suitable arrival time, and return the tag to full size:

```
> Open "C:/Temp/A-ASL-SMALL.mhd#SS3"
> Open "C:/Temp/A-ASL-SMALL-INITIAL.mhd#UC3"
> Open "C:/Temp/A-ASL-SMALL-SPEED.mhd#F3"
> FastMarching InitialImage="Initial" StoppingValue=100.0
> MorphologicalErode KernelRadius=itkSize(2,2,2)
> BinaryThreshold Lower=0 Upper=14
> CastToUC
> Resize OutputSize=itkSize(512,373,512) Interpolator="Nearest"
> Save "C:/Temp/A-ASL-HEART.mhd"
> Close
```

3.5 Vesselness

We now segment the vessels using the script created earlier:

```
> Open "C:/Temp/A-ASL-SMALL.mhd#F3"
> Vesselness Sigma=0.7
> ImageRange Minimum=0.0 Maximum=100.0
> ConnectedThreshold Lower=40 Upper=1000
> CastToUC
> Resize OutputSize=itkSize(512,373,512) Interpolator="Nearest"
> MorphologicalDilate Operation="Binary" KernelRadius=itkSize(2,2,2)
> Save "C:/Temp/A-ASL-VESSELS.mhd"
> Close
```

The `ImageRange` script is required to increase the contrast for selecting the seeds. We specified two seeds for the region growing: one in the left coronary artery (LCA) [95, 122, 169], and one in the right coronary artery (RCA) [126, 86, 96].

3.6 Combine Tags

We are now ready to combine our two tags into a single image:

```
> Open "C:/Temp/A-ASL-VESSELS.mhd#UC3"
> Open "C:/Temp/A-ASL-HEART.mhd#UC3"
> ShiftScale Shift=-150
> BinaryPixelMath Operation="Max" Input1="Heart_Shift" Input2="Vessels"
> Save "C:/Temp/A-ASL-TAGS.mhd"
> Close
```

3.7 Histogram

This step is purely optional, but it is often desirable to have a histogram to aid transfer function specification:

```
> Open "C:/Temp/A-ASL-SMALL.mhd#F3"
> MorphologicalGradientMagnitude KernelRadius=itkSize(1,1,1)
> ValueEdgeHistogram Value="Small" Edge="Magnitude" NumberOfBins=[512,128]
> Save "C:/Temp/A-ASL-HISTOGRAM.png"
> Close
```

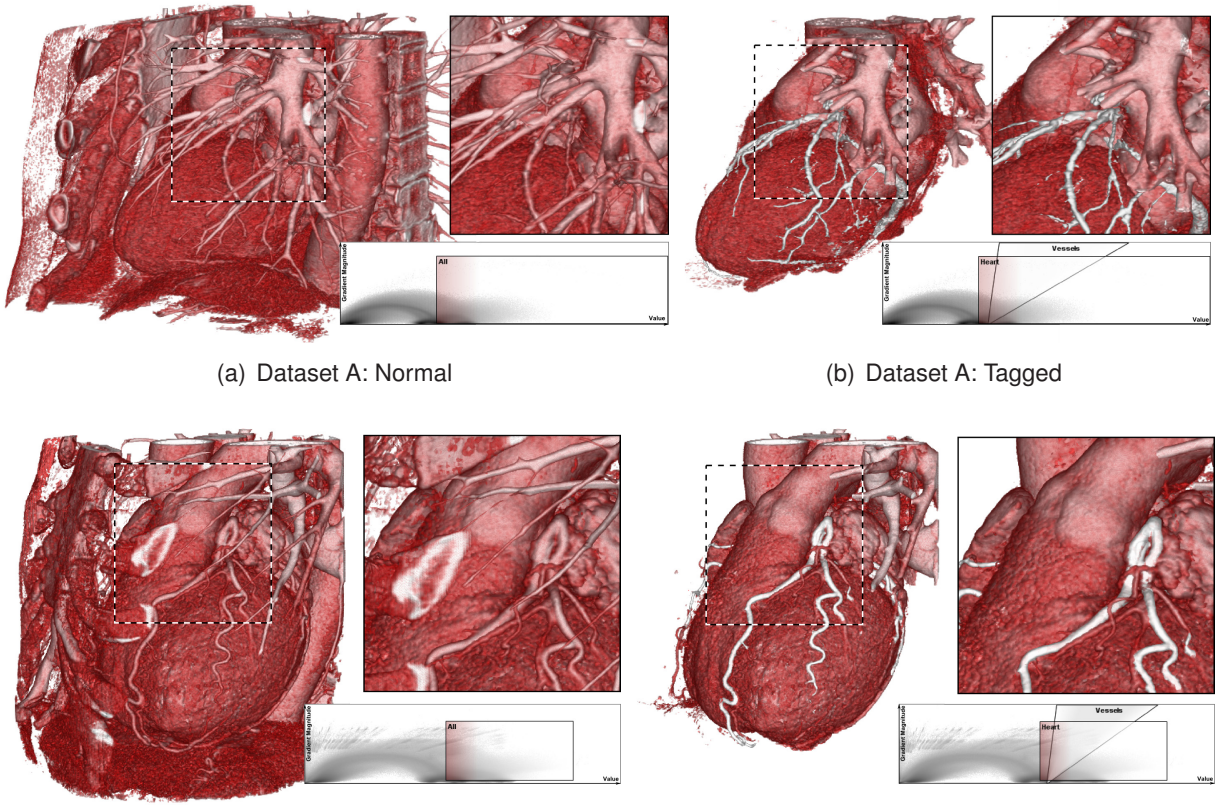
3.8 Volume Render

The preparation is now complete and we are ready to render the volumes:

```
> Open "C:/Temp/A-ASL-HISTOGRAM.png#UC2"
> Open "C:/Temp/A-ASL-TAGS.mhd#UC3"
> Open "C:/Temp/A-ASL.mhd#SS3"
> MorphologicalGradientMagnitude KernelRadius=itkSize(1,1,1)
> RescaleIntensityToUC
> VolumeRender Value="A-ASL.mhd" Gradient="Rescale" Tags="Tags"
        Tf=siTransferFunction(itkSize(512,128),"All","Heart","Vessels")
```

Select the “Renderer Editor” and configure the “FragmentProgramPath”. Select the “Transfer function editor” and right-click on the transfer function to add components similar to those shown in the results section.

4 Results



(a) Dataset A: Normal

(b) Dataset A: Tagged

(c) Dataset C: Normal

(d) Dataset C: Tagged

Figure 1: Resultant images using the normal and tagged volume rendering approaches. Dataset A is provided with this article.

5 Conclusion

We have presented a method for tagged volume rendering of the heart. The method is comprised of two stages: offline segmentation of the coronary arteries and pericardial cavity; followed by the online application of value-magnitude transfer functions to refine the tags. The method was presented using the SharpImage prototyping environment which made the steps somewhat involved, but easily reproduced. Feel free to contact us with any questions or suggestions[2].

[2]Corresponding author: Dan Mueller: d.mueller@qut.edu.au or dan.muel@gmail.com.

References

[1] E. Fishman, D. Ney, D. Heath, F. Corl, K. Horton, and P. Johnson. Volume rendering versus maximum intensity projection in CT angiography: What works best, when, and why. *Radiographics*, 26(3):905–922, 2006. 1

[2] M. Hadwiger, C. Berger, and H. Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Visualization*, pages 301–308. IEEE, 2003. 1, 2.2

[3] A. Kanitsar, R. Wegenkittl, D. Fleischmann, and M. Gröller. Advanced curved planar reformation: flattening of vascular structures. In *IEEE Visualization*, pages 43–50, 2003. 1

[4] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002. 1

[5] J. Kniss, R. Van Uitert, A. Stephens, G. Li, and T. Tasdizen. Statistically quantitative volume visualization. In *Visualization*, pages 287 – 294. IEEE, 2005. 1

[6] D. Mueller. ManagedITK: .NET wrappers for ITK. *The Insight Journal*, January–June, 2007, Available online: http://insight-journal.org/dspace/handle/1926/501. 2

[7] D. Mueller. SharpImage: An image processing prototyping environment. *The Insight Journal*, July–December, 2007, Available online: http://insight-journal.org/dspace/handle/1926/556. 1, 2

[8] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. Avila, K. Raghu, R. Machiraju, and J. Lee. The transfer function bake-off. *IEEE Computer Graphics and Applications*, 21(3):16–22, 2001. 1

[9] Y. Sato, S. Nakajima, N. Shiraga, H. Atsumi, S. Yoshida, T. Koller, G. Gerig, and R. Kikinis. Three-dimensional multi-scale line filter for segmentation and visualization of curvilinear structures in medical images. *Medical Image Analysis*, 2(2):143–168, 1998. 1

[10] J. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge Press, 2nd edition, 1999. 1

[11] N. Svakhine, D. Ebert, and D. Stredney. Illustration motifs for effective medical volume illustration. *IEEE Computer Graphics and Applications*, 25(3):31–39, 2005. 2.2

[12] F. Tzeng, E. Lum, and K. Ma. An intelligent system approach to higher-dimensional classification of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):273–284, 2005. 1

[13] G. Weber, S. Dillard, H. Carr, V. Pascucci, and B. Hamann. Topology-controlled volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):330–341, 2007. 1

Vessel Enhancing Diffusion Filter

Release 2.00

Andinet Enquobahrie [1], Luis Ibanez[1], Elizabeth Bullitt[2] and Stephen Aylward[1]

September 13, 2007

[1]Kitware Inc.
[2]CASILab, The University of North Carolina.

Abstract

This paper describes vessel enhancing diffusion (VED) filters implemented using the Insight Toolkit (ITK) [2]. The filters are implementation of the VED algorithm developed by Manniesing et al [4]. The VED algorithm follows a multiscale approach to enhance vessels using an anisotropic diffusion scheme guided by a vesselness measure at the pixel level. Vesselness is determined by geometrical analysis of the Eigen system of the Hessian matrix. For this purpose, a smoothed version of the Frangi's vesselness function [1] is formulated. Experiments were conducted to evaluate the performance of the VED filters in enhancing vessels in lung CT scans.

Contents

1 Introduction **2**
1.1 Overview of VED algorithm 2

2 VED Filters Design in ITK **3**
2.1 Smoothed Frangi's Vesselness Measure Computing Filters 4
2.2 Anisotropic Diffusion Filters for Vessel Enhancement 4

3 Experiments and results **4**

4 Conclusions **6**

5 Acknowledgment **9**

A Example 1 - Computing vesselness measure **9**

B Example 2 - Vessel enhancement using VED filter **11**

1 Introduction

Accurate quantification and visualization of vascular structures is critical in diagnosis and treatment of vascular diseases. Successful interventional clinical procedures such as bypass surgery and coronary artery stenting require the accurate vascular structure visualization during the planning stages. Similarly, effective diagnostic procedures such as stenosis grading depend on the accurate vascular structure quantification.

Various vascular imaging techniques are deployed in clinical practices. Among two dimensional techniques, Digital Subtraction Angiography (DSA) is one of the most commonly used technique for the visualization of blood vessels. Three dimensional imaging techniques such as CTA (Computed Tomography Angiography) and MRA (Magnetic Resonance Angiography) are also common in the clinical setting.

Vessel segmentation algorithms can be applied to 2D and 3D vascular images. Several segmentation techniques have been developed. A review of several techniques is given in [3].

To increase the effectiveness of segmentation algorithms, vessel enhancement procedures are often first applied as a preprocessing step [1]. The performance of the enhancement algorithm has been shown to tremendously impact the results of the segmentation algorithm.

Vessel enhancement algorithms are also useful for the visual interpretation of 3D vascular images. For example, clinicians often generate maximum intensity projects (MIP) images for the visual analysis of the massive amount of data produced by 3D imaging techniques. However, the occurrence of overlapping non-vascular anatomical structures greatly affects vascular visualization in MIP images. Additionally, small blood vessels with low contrast edges are often not clearly visible in MIP images. To alleviate these problems, enhancement algorithms can be first applied tn the vascular images to suppress non-vascular structures and improve the delineation of small blood vessels.

This paper presents an open-source implementation of a vessel enhancement algorithm called VED [4].

1.1 Overview of VED algorithm

The VED algorithm is based on anisotropic diffusion scheme guided by vessel-likelihood at pixel level. It is basically a smoothing filter with the strength and direction of diffusion is determined by a "'vesselness"' measure. Vesselness is measured by analyzing the eigen system of the Hessian matrix. Several vesselness functions have been proposed. Manniesing's VED algorithm ([4])is based on Frangi's vesselness function. For increasing-magnitude eigen values of a Hessian matrix

$$|\lambda_1| \leq |\lambda_2| \leq |\lambda_3|$$

Frangi's vesselness function is composed of three components formulated to discriminate tubular structures from blob-like and/or plate-like structure as shown in Equation 1

$$V_F(\lambda) = \begin{cases} 0 & \text{if } \lambda_2 > 0 \text{ or } \lambda_3 > 0 \\ (1 - e^{-\frac{R_A^2}{2\alpha^2}}) \, . \, e^{-\frac{R_B^2}{2\beta^2}} \, . \, (1 - e^{-\frac{S^2}{2\gamma^2}}) & \text{otherwise} \end{cases} \quad (1)$$

With

$$R_A = \frac{|\lambda_2|}{|\lambda_3|} \quad (2)$$

$$R_B = \frac{|\lambda_1|}{\sqrt{|\lambda_2 \lambda_3|}} \tag{3}$$

$$S = \sqrt{\lambda_1^2 + \lambda_2^2 + \lambda_3^2} \tag{4}$$

α, β, γ are thresholds which control the sensitivity of the vesselness measure.

However, Frangi's vesselness function is not continuous and can't be used in the diffusion process. Hence, Manniesing et al proposed a smoothed version of Frangi's vesselness function as shown below.

$$V_S(\lambda) = \begin{cases} 0 & \text{if } \lambda_2 \geq 0 \text{ or } \lambda_3 \geq 0 \\ (1 - e^{-\frac{R_A^2}{2\alpha^2}}) \, . \, e^{-\frac{R_B^2}{2\beta^2}} \, . \, (1 - e^{-\frac{S^2}{2\gamma^2}}) \, . \, e^{-\frac{2c^2}{|\lambda_2|\lambda_3^2}} & \text{otherwise} \end{cases} \tag{5}$$

For a multiscale analysis, the vesselness function is computed for a range of scales and the maximum response is selected.

$$V = max_{\alpha_{min} \leq \alpha \leq \alpha_{max}} \; V_s(\lambda) \tag{6}$$

Next, a diffusion tensor is defined in such a way that diffusion is promoted along the vessel but prohibited perpendicular to the vessel.

$$D = Q \lambda' Q^T \tag{7}$$

Where Q is a matrix containing eigen vectors of the Hessian matrix and λ' is a diagonal matrix containing the following elements

$$\lambda_1' = 1 + (w - 1) \, V^{\frac{1}{S}}$$
$$\lambda_2' = \lambda_3' = 1 + (\varepsilon - 1) \, . \, V^{\frac{1}{S}}$$

Where ε, w and S are algorithm parameters.

Using this tensor definition, a diffusion equation is formulated as follows

$$L_t = \nabla \, . \, (D \nabla L) \tag{8}$$

Vascular structures will be enhanced by evolving the image according to the above diffusion equation.

2 VED Filters Design in ITK

VED algorithm implementation consists of two main parts. The first part involves implementation of filters to compute smoothed Frangi's vesselness measure for a given image. The second part involves implementation of the anisotropic diffusion filters for vessel enhancement.

2.1 Smoothed Frangi's Vesselness Measure Computing Filters

Two filters were implemented to evaluate Frangi's vesselness measure in a multiscale framework. The first filter, HessianSmoothed3DToVesselnessMeasureImageFilter computes vesselness measure at a single scale. The second filter, MultiScaleHessianSmoothed3DToVesselnessMeasureImageFilter computes the maximum vesselness response from a range of scales. To use the filter for a multiscale analysis, a user has to specify minimum, maximum sigma values and number of scales. The filter computes and selects the maximum vesselness response at exponentially distributed number of scales between the specified minimum and maximum sigma values.

These filters are derived from the `itk::ImageToImageFilter`. For each pixel, Hessian matrix is computed using the `itk::HessianRecursiveGaussianImageFilter` filter. This filter computes Hessian matrix by convolving the input image with second and cross derivatives of the Gaussian function.

The multiscale filter has the following main public methods.

1 To set minimum sigma value

```
SetSigmaMin  ( double );
```

2 To set maximum sigma value

```
SetSigmaMax ( double );
```

3 To set the number of sigma steps

```
SetNumberOfSigmaSteps( int );
```

Appendix A shows an example on how to use this filter.

2.2 Anisotropic Diffusion Filters for Vessel Enhancement

The implementation of the anisotropic filters follow the finite difference solver framework. The implementation consists of two components: solver object (itkAnisotropicDiffusionVesselEnhancementImageFilter) and a specialized finite difference function object (itkAnisotropicDiffusionVesselEnhancementFunction). The solver object establishes the infrastructure for accepting input image and producing output image. In addition, the solver object uses the specialized finite difference function object to perform the diffusion equation computation at each pixel for several iterations. The solver object and the the function objects are derived from `itk::FiniteDifferenceImageFilter` and `itk::FiniteDifferenceFunction` respectively. Figure 1 shows the flowchart for the vessel enhancement diffusion algorithm. An example program that demonstrates how to use this filter is provided in appendix B.

3 Experiments and results

Experiments were conducted to test the effectiveness of the VED algorithm in enhancing vessels in a whole lung CT scan. Figure 2 shows lung CT scan used to test the algorithm.

The testing dataset is distributed as part of the source code submission to the Insight Journal. Readers are encouraged to build the source code and run the algorithm on the testing dataset.

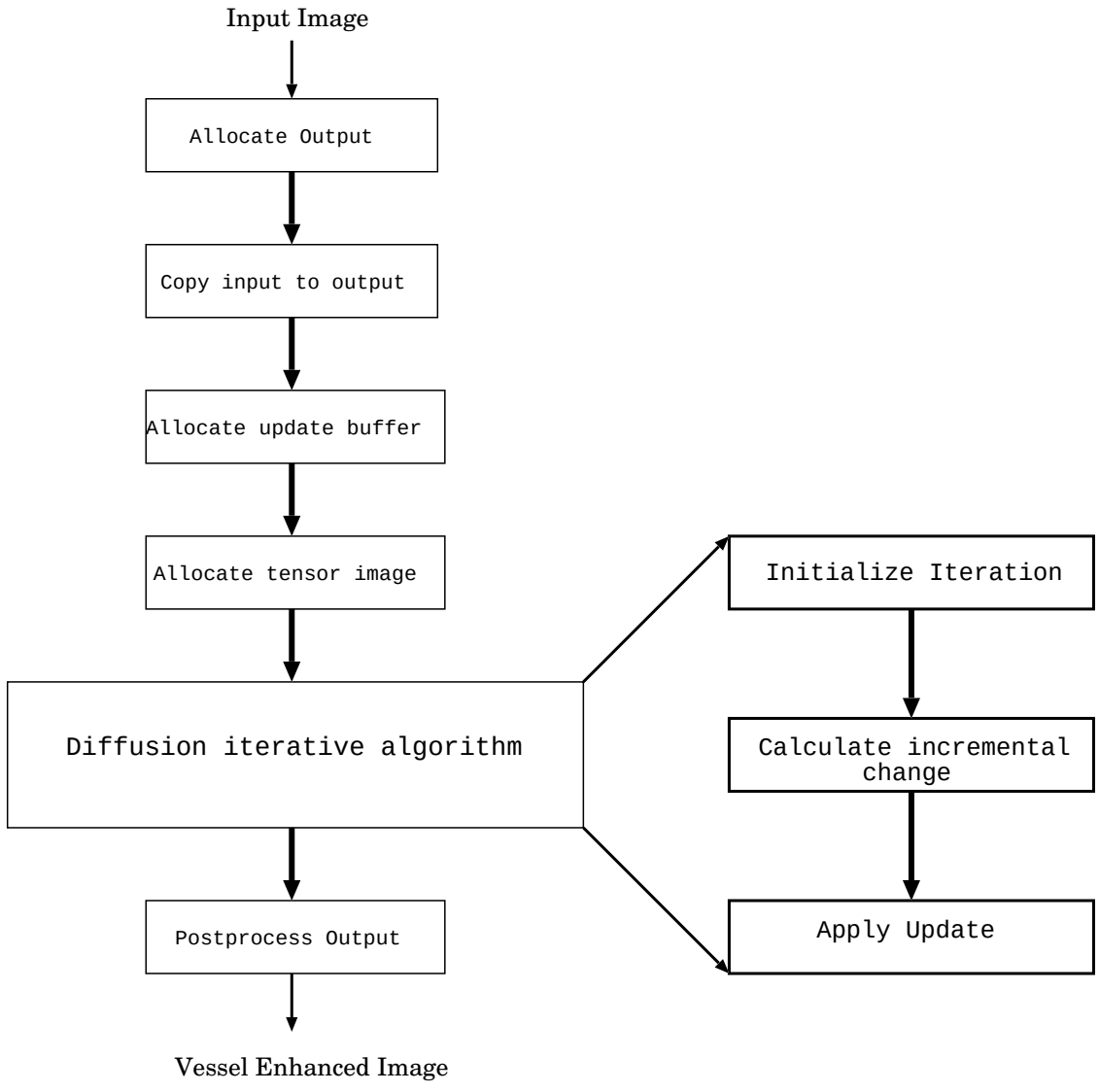


Figure 1: VED ITK filter flowchart

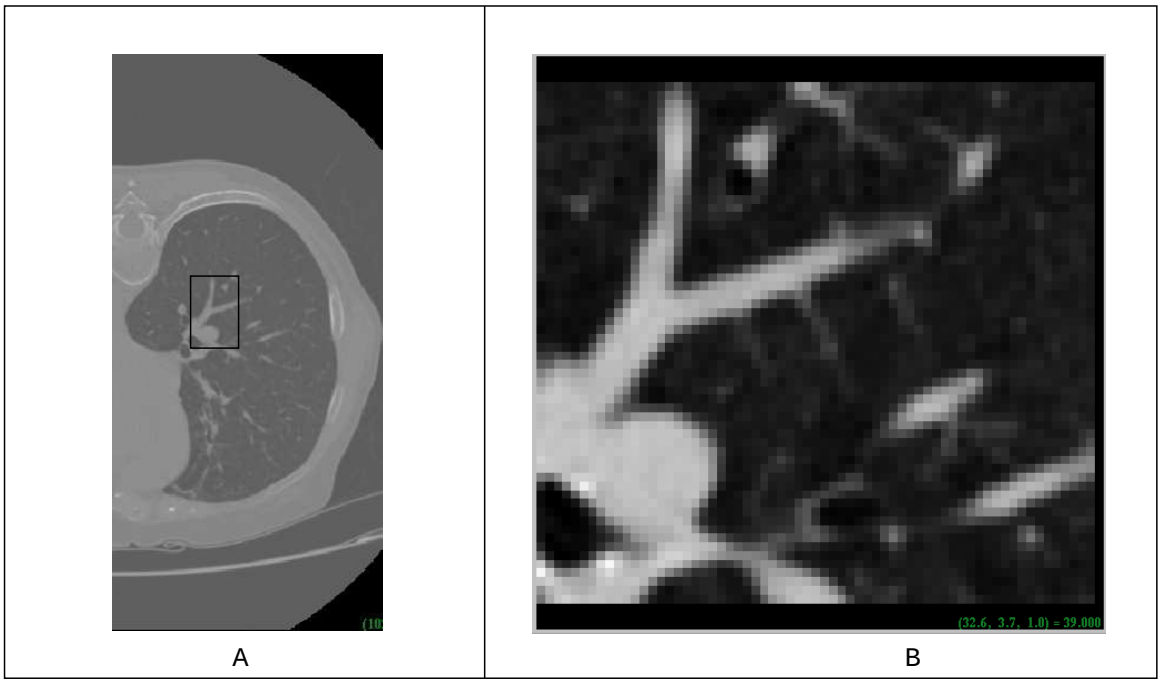


Figure 2: Testing dataset: A) Lung CT scan B) Cropped ROI

In the first experiment, Frangi's vesselness filters were run on the testing dataset to evaluate the performance of the filter on a single scale and a range of scales. The results are shown in Figure 3. With a sigma value of 0.5, small size vessels are enhanced as shown in Figure 3B. If the scale is increased to a sigma value of 4.0, large size vessels are enhanced (Figure 3C). To enhance vessels with varying size, the image was run through Frangi's multiscale filter with a sigma range of [0.5 4.0]. The result is shown in Figure 3D. As clearly evident from the result, multi scale analysis is useful in enhancing various size vessels available in the scan. This is very useful for a complete vascular structure enhancement and reconstruction. Although the vascular structures are generally enhanced, the results show that the vesselness measure is highly sensitive to noise pixels.

In the second experiment, the performance of the VED filters was evaluated. VED filter was run on the same Lung CT scan. The results are shown in Figure 4. The VED filter was operated in a multiscale framework with computations at ten sigma values exponentially distributed in sigma value range [0.5 4.0]. The diffusion process was analyzed for different number of iterations. The results for 10, 25 and 50 number of iterations is shown in Figure 4B, 4C and 4D respectively. Overall, the results show that VED filters enhance the vascular structures better than Frangi's vesselness measure. In addition, VED filters are less affected by noise pixels. Furthermore, with the increase in the number of iterations, an increased smoothing effect was observed.

4 Conclusions

In this paper, we have described VED filters implemented using ITK. The implementation is based on the algorithm by Manniesing et al [4]. VED algorithm uses vesselness measure based on Hessian Eigen system to guide a diffusion process. Experiments were conducted to evaluate the performance of the filters in enhancing vessels in Lung CT scans. Visualizing the enhanced images showed that VED algorithm performs better than Frangi's vesseleness measure.

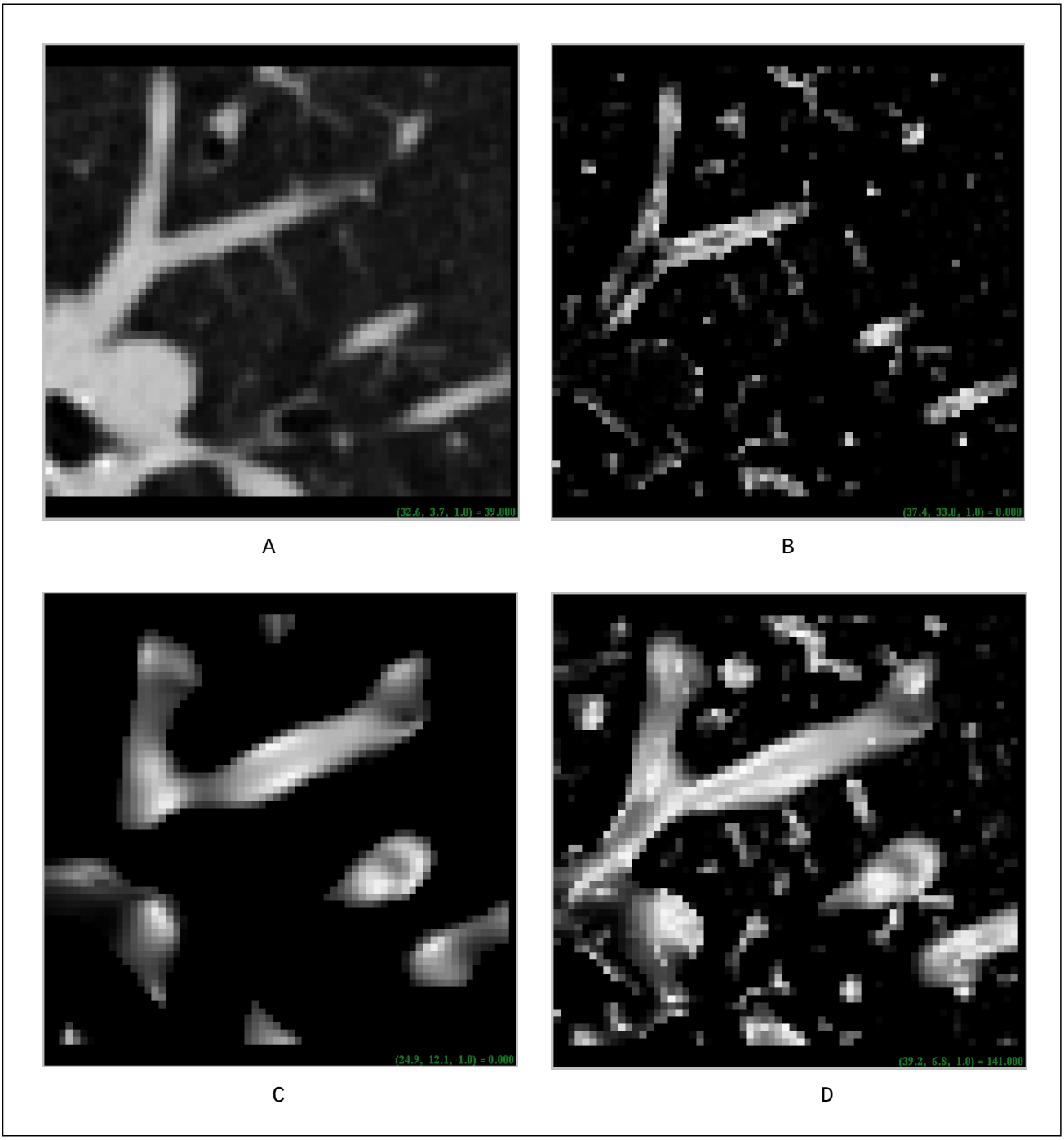


Figure 3: Frangi vesselness results: A) Original image B) sigma=0.5 C) sigma=4.0 D) Multiscale analysis sigma=[0.5 4.0]

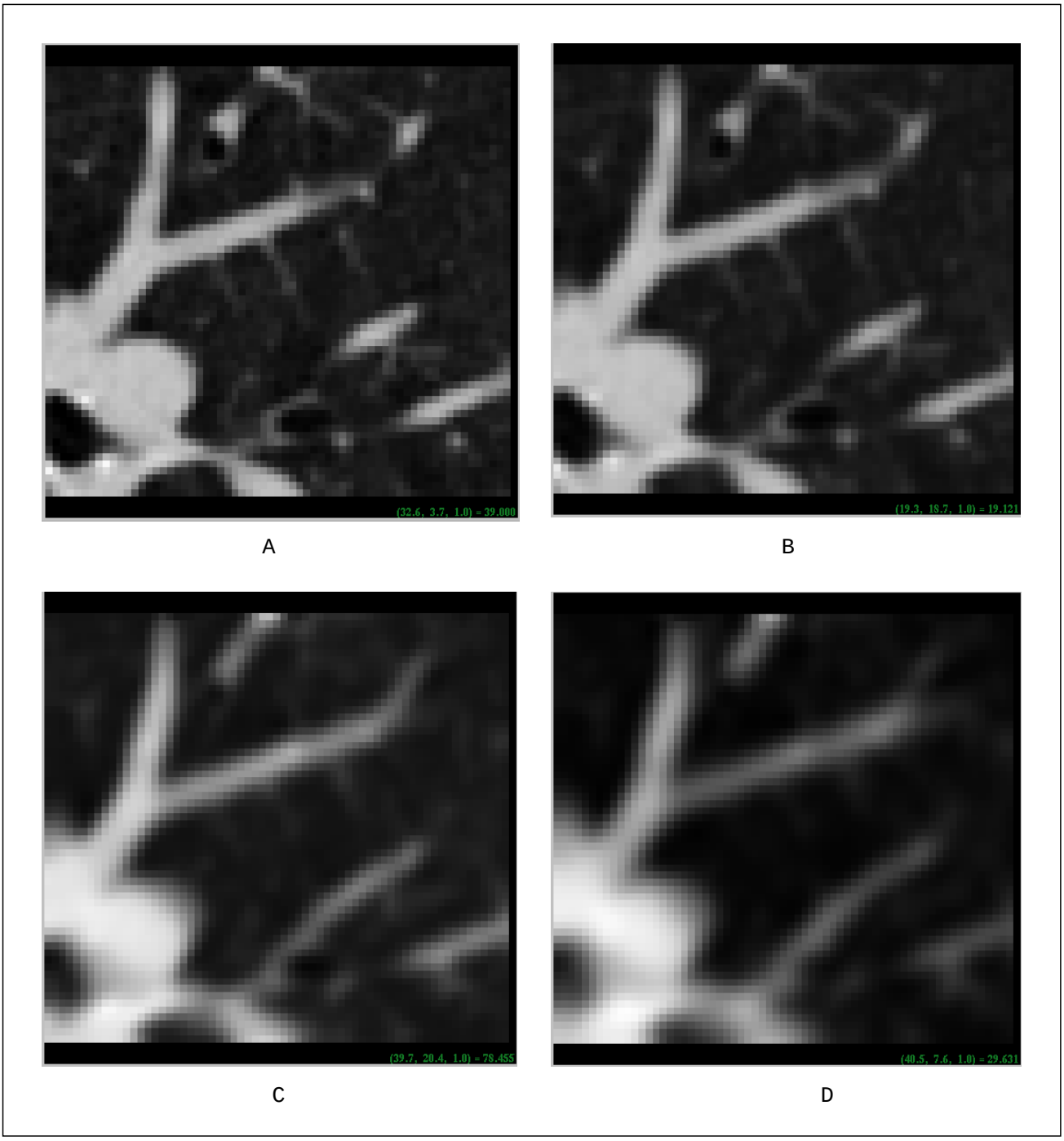


Figure 4: VED algorithm results: A) Original image B) Number of iterations = 10 C) Number of iterations = 25 D) Number of iterations = 50

5 Acknowledgment

This work was supported by NIH R01 EB000219 (PI: Bullitt, UNC) and NIH R01 HL69808 (PI: Bullitt, UNC).

A Example 1 - Computing vesselness measure

```
#include "itkMultiScaleHessianSmoothed3DToVesselnessMeasureImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "itkRescaleIntensityImageFilter.h"

int main(int argc, char* argv [] )
{
  if ( argc < 3 )
    {
    std::cerr << "Missing Parameters: "
              << argv[0]
              << " Input_Image"
              << " Vessel_Enhanced_Output_Image"
              << [SigmaMin SigmaMax NumberOfScales]" << std::endl;
    return EXIT_FAILURE;
    }

  // Define the dimension of the images
  const unsigned int Dimension = 3;
  typedef short      InputPixelType;
  typedef double     OutputVesselnessPixelType;

  // Declare the types of the images
  typedef itk::Image< InputPixelType, Dimension>            InputImageType;

  typedef itk::Image< OutputVesselnessPixelType, Dimension> VesselnessOutputImageType;

  typedef itk::ImageFileReader< InputImageType  >      ImageReaderType;

  ImageReaderType::Pointer   reader = ImageReaderType::New();
  reader->SetFileName ( argv[1] );

  std::cout << "Reading input image : " << argv[1] << std::endl;
  try
    {
    reader->Update();
    }
  catch ( itk::ExceptionObject &err )
```

```
  {
  std::cerr << "Exception thrown: " << err << std::endl;
  return EXIT_FAILURE;
  }


// Declare the type of multiscale vesselness filter
typedef itk::MultiScaleHessianSmoothed3DToVesselnessMeasureImageFilter<
                                          InputImageType,
                                          VesselnessOutputImageType>
                                          MultiScaleVesselnessFilterType;

// Create a vesselness Filter
MultiScaleVesselnessFilterType::Pointer MultiScaleVesselnessFilter =
                                    MultiScaleVesselnessFilterType::New();

MultiScaleVesselnessFilter->SetInput( reader->GetOutput() );

if ( argc >= 4 )
  {
  MultiScaleVesselnessFilter->SetSigmaMin( atof(argv[3])  );
  }

if ( argc >= 5 )
  {
  MultiScaleVesselnessFilter->SetSigmaMax( atof(argv[4]) );
  }

if ( argc >= 6 )
  {
  MultiScaleVesselnessFilter->SetNumberOfSigmaSteps( atoi(argv[5]) );
  }

try
  {
  MultiScaleVesselnessFilter->Update();
  }
catch( itk::ExceptionObject & err )
  {
  std::cerr << "Exception caught: " << err << std::endl;
  return EXIT_FAILURE;
  }

std::cout << "Writing out the enhanced image to " <<  argv[2] << std::endl;

//Rescale the output of the vesslness image
typedef itk::Image<unsigned char, 3>              OutputImageType;
typedef itk::RescaleIntensityImageFilter< VesselnessOutputImageType,
```

```
                                          OutputImageType>
                                          RescaleFilterType;

  RescaleFilterType::Pointer rescale = RescaleFilterType::New();
  rescale->SetInput( MultiScaleVesselnessFilter->GetOutput() );
  rescale->SetOutputMinimum(   0 );
  rescale->SetOutputMaximum( 255 );
  rescale->Update();

  typedef itk::ImageFileWriter< OutputImageType  >      ImageWriterType;
  ImageWriterType::Pointer writer = ImageWriterType::New();

  writer->SetFileName( argv[2] );
  writer->SetInput ( rescale->GetOutput() );

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

  return EXIT_SUCCESS;

}
```

B Example 2 - Vessel enhancement using VED filter

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

int main(int argc, char* argv [] )
{
  if ( argc < 3 )
    {
    std::cerr << "Missing Parameters: "
              << argv[0]
              << " Input_Image"
              << " Vessel_Enhanced_Output_Image [SigmaMin SigmaMax NumberOfScales NumberOf
    return EXIT_FAILURE;
    }
```

```
// Define the dimension of the images
const unsigned int Dimension = 3;
typedef double      InputPixelType;
typedef double      OutputPixelType;

// Declare the types of the images
typedef itk::Image< InputPixelType, Dimension>           InputImageType;
typedef itk::Image< InputPixelType, Dimension>           OutputImageType;

typedef itk::ImageFileReader< InputImageType  >      ImageReaderType;

ImageReaderType::Pointer   reader = ImageReaderType::New();
reader->SetFileName ( argv[1] );

std::cout << "Reading input image : " << argv[1] << std::endl;
try
  {
  reader->Update();
  }
catch ( itk::ExceptionObject &err )
  {
  std::cerr << "Exception thrown: " << err << std::endl;
  return EXIT_FAILURE;
  }

// Declare the anisotropic diffusion vesselness filter
typedef itk::AnisotropicDiffusionVesselEnhancementImageFilter< InputImageType,
                                          OutputImageType>  VesselnessFilterType;

// Create a vesselness Filter
VesselnessFilterType::Pointer VesselnessFilter =
                                    VesselnessFilterType::New();

VesselnessFilter->SetInput( reader->GetOutput() );

if ( argc >= 4 )
  {
  VesselnessFilter->SetSigmaMin( atof(argv[3])  );
  }

if ( argc >= 5 )
  {
  VesselnessFilter->SetSigmaMax( atof(argv[4]) );
  }

if ( argc >= 6 )
  {
```

```
    VesselnessFilter->SetNumberOfSigmaSteps( atoi(argv[5]) );
    }

  if ( argc >= 7 )
    {
    VesselnessFilter->SetNumberOfIterations( atoi(argv[6]) );
    }

  VesselnessFilter->SetSensitivity( 5.0 );
  VesselnessFilter->SetWStrength( 25.0 );
  VesselnessFilter->SetEpsilon( 10e-2 );

  std::cout << "Enhancing vessels.........: " << argv[1] << std::endl;

  try
    {
    VesselnessFilter->Update();
    }
  catch( itk::ExceptionObject & err )
    {
    std::cerr << "Exception caught: " << err << std::endl;
    return EXIT_FAILURE;
    }

  std::cout << "Writing out the enhanced image to " <<  argv[2] << std::endl;

  typedef itk::ImageFileWriter< OutputImageType  >      ImageWriterType;
  ImageWriterType::Pointer writer = ImageWriterType::New();

  writer->SetFileName( argv[2] );
  writer->SetInput ( VesselnessFilter->GetOutput() );

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

  return EXIT_SUCCESS;

}
    std::cerr << "Exception caught: " << err << std::endl;
    return EXIT_FAILURE;
    }
```

```
  return EXIT_SUCCESS;

}
```

References

[1] A. F. Frangi, W. J. Niessen, K. L. Vincken, and M. A. Viergever. Multiscale vessel enhancement filtering. In W. M. Wells, A. Colchester, and S. Delp, editors, *MICCAI'98 Medical Image Computing and Computer-Assisted Intervention*, Lecture Notes in Computer Science, pages 130–137. Springer Verlag, 1998. (document), 1

[2] L. Ibanez and W. Schroeder. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, 2003. (document)

[3] C. Kirbas and F. Quek. A review of vessel extraction techniques and algorithms. *ACM Computing Surveys*, 36(2):81–121, 2004. 1

[4] R. Mannieshing, M.A. Viergever, and W. J . Niessen. Vessel enhancing diffusion: A scale space representation of vessel structures. *Medical Image Analysis*, 2006. (document), 1, 1.1, 4

Spherical Wavelet ITK Filter

Release 0.00

Yi Gao[1], Delphine Nain[1,2], Xavier LeFaucheur[1,3], Allen Tannenbaum[1,2,3]

June 13, 2007

[1]Department of Biomedical Engineering, Georgia Institute of Technology
[2]College of Computing, Georgia Institute of Technology
[3]Department of Electrical and Computer Engineering, Georgia Institute of Technology

Abstract

This paper describes the Insight Toolkit (ITK) Spherical Wavelet object: itkSWaveletSource. This ITK object is an implementation of a paper by Schröder and W. Sweldens, "Spherical Wavelets: Efficiently Representing Functions on the Sphere" [8], with pseudo-code given in their paper entitled "Spherical wavelets: Texture processing" [7]. In these papers, Swelden et. al. show how to do decompose a scalar signal defined on a spherical mesh into spherical wavelet coefficients (analysis step, also called forward transform), and vice-versa (synthesis step, also called inverse transform). We have implemented the spherical wavelet transform in ITK entitled itkSWaveletSource object, which will take the scalar function defined on a spherical mesh as input and apply spherical wavelet analysis and synthesis on it. In this paper, we describe our code and provide the user with enough details to reproduce the results which we present in this paper. This filter has a variety of applications including shape representation and shape analysis of brain surfaces, which was the initial motivation for this work.

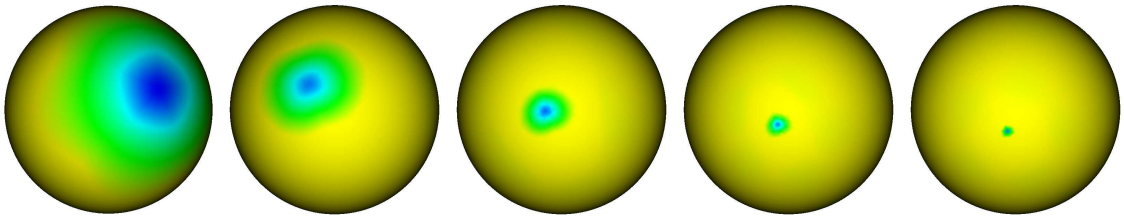
This paper is accompanied with the source code, input data, parameters and output data that the authors used for validating the spherical wavelet transform described in this paper.

Contents

1 Algorithm **2**
1.1 Spherical Wavelet Transform . . . 2
1.2 Discrete Fast Spherical Wavelet Transform . . . 3
Multi-resolution Analysis Mesh . . . 3
Fast Spherical Wavelet Transforms . . . 4

2 User's Guide **5**
2.1 Software Requirements . . . 5
2.2 Visualizing the Analysis Mesh . . . 5
2.3 Basic Usage: Forward and Backward Transform . . . 7
2.4 Advanced Usage: Forward Transform, Coefficient Modification/Filtering, Backward Transform 7
Visualization of a Basis Function . . . 8

3 Applications **8**



(a) Scaling function, scale 0 (b) Wavelet function, scale 1 (c) Wavelet function, scale 2 (d) Wavelet function, scale 3 (e) Wavelet function, scale 4

Figure 1: Visualization of selected basis function. The color corresponds to the value of the functions. (a) Visualization of a scaling function at scale 0 on the sphere (b-e) Visualization of a wavelet function for various scales.

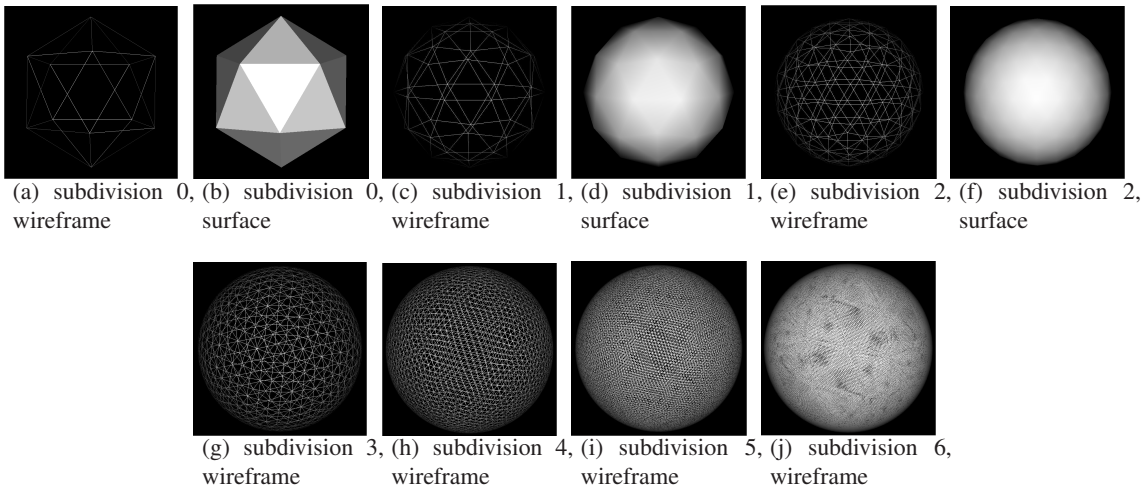
4 Conclusions **9**

Spherical wavelets are used in various fields such as computer graphics for texture analysis applications [7], in computer vision for data compression applications [11], or astronomy for data analysis applications [10]. In citesch-swe:env, Sweldens et. al describe a fast spherical wavelet transform on the discrete sphere. The forward transform takes as input a scalar function defined on the discrete sphere and outputs coefficients that describe the features of the signal in a local fashion in both frequency and space. Conversely, the backward transform reconstructs a signal on the sphere from a set of spherical wavelet coefficients. This transform is computationally attractive as the associated transform is linear in the number of vertices of the sphere. Once in the wavelet domain, filtering operators such as smoothing, enhancement, edge finding, and noise removal can be performed in a straightforward fashion. Additionally, coefficient analysis reveals at what frequency and spatial location the signal content lies. Recently, the fast spherical wavelet transform has been used in medical imaging for shape representation [2], shape analysis [13, 12], segmentation [3, 4] and statistical shape analysis []. In this paper, we present an ITK implementation of the fast spherical wavelet transform from [5]. We present applications and code of interest to the medical imaging community.

1 Algorithm

1.1 Spherical Wavelet Transform

A spherical wavelet basis is composed of functions defined on the sphere that are localized in space and characteristic scales and therefore match a wide range of signal characteristics, from high frequency edges to slowly varying harmonics [1]. The basis is constructed of scaling functions defined at the coarsest scale and wavelet functions defined at subsequent scales. A *scaling function* is a function on the standard unit sphere ($\mathbb{S}$) denoted by $\varphi_{j,k} : \mathbb{S} \rightarrow \mathbb{R}$ where j is the scale of the function and k is a spatial index that indicates where on the surface the function is centered. A usual shape for the scaling function is a hat function defined to be 1 at its center and to decay linearly to 0. Figure 1(a) shows a scaling function for resolution $j = 0$ and a select k. The color on the sphere indicates the value of the function $\varphi_{0,k}$ at every point on the sphere. As the resolution j increases, the support of the scaling function decreases. A wavelet function is denoted by $\psi_{j,m} : \mathbb{S} \rightarrow \mathbb{R}$. At a particular scale j, wavelet functions are combinations of resolution j and $(j+1)$ scaling functions. Figures 1(b)- 1(e) show wavelet functions for different values of j and a select m. Note that the



(a) subdivision 0, wireframe (b) subdivision 0, surface (c) subdivision 1, wireframe (d) subdivision 1, surface (e) subdivision 2, wireframe (f) subdivision 2, surface

(g) subdivision 3, wireframe (h) subdivision 4, wireframe (i) subdivision 5, wireframe (j) subdivision 6, wireframe

Figure 2: Visualization of the Recursive Partitioning of an icosahedron

support of the functions becomes smaller as the resolution increases. Together, the coarsest level scaling function and all wavelet scaling functions construct a basis for the function space L^2:

$$L^2 = \{\varphi_0, k | k \in N_0\} \bigcup \{\psi_j, m | j \geq 0, m \in N_{j+1}\}. \tag{1}$$

A given function $f : \mathbb{S} \rightarrow \mathbb{R}$ can be expressed in the basis as a linear combination of basis functions and coefficients:

$$f(\mathbf{x}) = \sum_k \lambda_{0,k} \varphi_{0,k}(\mathbf{x}) + \sum_{0 \leq j} \sum_m \gamma_{j,m} \psi_{j,m}(\mathbf{x}). \tag{2}$$

Scaling coefficients $\lambda_{0,k}$ represent the low pass content of the signal f, localized where the associated scaling function has support, whereas wavelet coefficients $\gamma_{j,m}$ represent localized band pass content of the signal, where the band pass frequency depends on the resolution of the associated wavelet function and the localization depends on the support of the function.

1.2 Discrete Fast Spherical Wavelet Transform

In this paper, we present the implementation of the discrete biorthogonal spherical wavelets functions defined on a 3D mesh proposed by Schröder and Sweldens [8, 7]. These are second-generation wavelets adapted to manifolds with non-regular grids. The main difference with the classical wavelet is that the filter coefficients of second generation wavelets are not the same throughout, but can change locally to reflect the changing (non translation invariant) nature of the surface and its measure. This means that wavelet functions defined on a mesh are not scaled and shifted versions of the function on a coarser grid, although they are similar in shape, in order to account for the varying shape of mesh triangles.

Multi-resolution Analysis Mesh

The second-generation spherical wavelets that we use are defined on surfaces which are topologically equivalent to the unit sphere ($\mathbb{S}$) and equipped with a multi-resolution mesh. A multi-resolution mesh is created

by recursively subdividing an initial polyhedral mesh so that each triangle is split into 4 "child" triangles at each new subdivision (resolution) level. This is done by adding a new midpoint at each edge, and connecting midpoints together. This process is shown on Figures 2. The starting shape is an icosahedron with 12 vertices and 20 faces, and at the fourth subdivision level, it contains 5120 faces and 2562 vertices. Table xx summarizes the number of vertices and triangles for a given subdivision level.

Any shape (not necessarily a sphere) that is equipped with such a multi-resolution mesh can be used to create a spherical wavelet basis and perform the spherical transform of a signal defined on that mesh. The number of basis functions and coefficients will equal the number of vertices of the mesh of analysis, which in turn depends on the number of subdivisions of the icosahedron used to create the analysis mesh. Table xx summarizes the number of scaling functions and wavelet functions for a given level of subdivision.

Fast Spherical Wavelet Transforms

The algorithm for the fast discrete spherical wavelet transform (FSWT) is given in [7]. In our implementation of the algorithm, we followed the notations of the pseudo-code as closely as possible.

Here, we sketch the transform algorithm in matrix form which gives a more compact and intuitive notation to understand the transform. In our implementation, we use the FSWT given in the pseudo-code of [7] in our implementation.

If there exist N vertices on the mesh, a total of N basis functions are created, composed of N_0 scaling functions (where N_0 is the initial number of vertices before the base mesh is subdivided, with $N_0 = 12$ for the icosahedron) and N_r wavelet functions for $r = 1,...R$ where N_r is the number of new vertices added at subdivision r ($N_1 = 30, N_2 = 120, N_3 = 480, N_4 = 1920$, etc for the subdivision of the icosahedron). In this paper, we will refer to the set of scaling and wavelet basis functions as basis functions as a shorthand.

In matrix form, the set of basis functions can be stacked as columns of a matrix Φ of size $N \times N$ where each column is a basis function evaluated at each of the N vertices. The basis functions are arranged by increasing resolution (subscript j) and within each resolution level, by increasing spatial index (subscript k). Since the basis functions are biorthogonal, $\Phi^T\Phi \neq Id$ (the identity matrix), so the inverse basis Φ^{-1} is used for perfect reconstruction, since $\Phi^{-1}\Phi = Id$.

Any finite energy scalar function evaluated at N vertices, denoted by the vector F of size $N \times 1$, can be transformed into a vector of basis coefficients C of size N using the *Forward Wavelet Transform*:

$$C = \Phi^{-1}F, \tag{3}$$

and recovered using the *Inverse Wavelet Transform*:

$$F = \Phi C. \tag{4}$$

The vector of coefficients C is composed of coefficients associated with each basis function in Φ. It contains scaling coefficients as its first N_0 entries, then wavelet coefficients associated with wavelet functions of resolution 1 for the next N_1 entries, and so forth, all the way to wavelet coefficients of resolution R for the last N_R entries. Therefore in total the vector contains N entries.

2 User's Guide

2.1 Software Requirements

Insight Toolkit(ITK) is needed to compile and run the code. The spherical wavelet coefficients and/or reconstructed scalar function values are in plain text files. The subdivided sphere, if written to file by the function provided, is of ITK meta file type.

Both KWVisu[6] and the dispMetaWithColor can be used to display the spherical mesh meta file.

2.2 Visualizing the Analysis Mesh

The following code is given in the file `itkSWaveletTest1.cxx`

```
SphereMeshSourceType::Pointer  mySphereMeshSource = SphereMeshSourceType::New();
int n = atoi(argv[1]);
if (n>=7 && n<0)
{
  std::cerr<<"Finest resolution can't be less than 0, and best
              to be no larger than 6..."<<std::endl;
  exit(-1);
}
mySphereMeshSource->SetResolution( n );
```

The first line creates the spherical wavelet object. This object will contain the finest multi-resolution analysis mesh (an icosahedron that has been divided n times), the function to be analyzed and the coefficients that result from the transform. Line 2 reads the level of subdivision `n` from the command line. Lines 3-8 check that the level of subdivision is in the right bounds[1]. Line 9 sets the total number of icosahedron subdivisions that will produce the finest resolution mesh.

```
PointType center;
center.Fill( 0.0 );
mySphereMeshSource->SetCenter( center );

VectorType scale;
scale.Fill( 1.0 );
mySphereMeshSource->SetScale( scale );
```

Lines 10-12 center the initial icosahedron at $(0,0,0)$ and lines 13-15 scale the vertices of the initial icosahedron to lie on the unit sphere (radius is 1.0).

```
mySphereMeshSource->Modified();
try
{
  mySphereMeshSource->Update();
}
```

[1] We constrained the subdivision level to be no larger than 6 due to a high level of vertices that will increase considerably the amount of memory and time needed for the transform.

```
catch( itk::ExceptionObject & excp )
{
  std::cerr << "Error during Update() " << std::endl;
  std::cerr << excp << std::endl;
}
```

Lines 16-25 update the object. At that point the initial icosahedron is subdivided `n` times, and the object keeps track of the parent relationships between vertices.

```
std::cerr<<"Testing task 1:"<<std::endl;
std::cerr<<"    Given different resolution in the command line, generate and
             visulize subdivided mesh at different resolutions."<<std::endl;

MeshType::Pointer myMesh = mySphereMeshSource->GetOutput();
displayITKmesh(myMesh);
```

In line 29, `mySphereMeshSource->GetOutput()` returns the finest resolution mesh (an icosahedron subdivided `n` times). It's not called here but by `mySphereMeshSource->WriteFinestSubdivisionMeshToMetaFile(fileName)`, the finest subdivision mesh is saved into .meta file. Line 30 displays the ITK mesh by using the VTK renderer.

Figure 2(b) was created by running:

```
itkSWaveletTest1 0
```

Figure 2(d) was created by running:

```
itkSWaveletTest1 1
```

Figure 2(f) was created by running:

```
itkSWaveletTest1 2
```

Figure 2(g) was created by running:

```
itkSWaveletTest1 3
```

Figure 2(h) was created by running:

```
itkSWaveletTest1 4
```

Figure 2(i) was created by running:

```
itkSWaveletTest1 5
```

Figure 2(j) was created by running:

```
itkSWaveletTest1 6
```

2.3 Basic Usage: Forward and Backward Transform

The scalar function defined on the mesh is represented as a vector of scalars, each element associated with one vertex. As the example, here we use the z coordinate of the vertex as the scalar associated with it. This is done by the following piece of codes:

```
std::vector< PointType > v = mySphereMeshSource->GetVerts();
std::vector< PointType >::const_iterator itv = v.begin();
std::vector< PointType >::const_iterator itvEnd = v.end();

std::vector< double > f( v.size() );
std::vector< double >::iterator itf = f.begin();
for ( ; itv != itvEnd; ++itv, ++itf ) *itf = (*itv)[2];
mySphereMeshSource->SetScalarFunction( f );
```

The last line above is to "push" the vector into the wavelet object. Next by

```
mySphereMeshSource->SWaveletTransform();
```

the wavelet coefficents are generated. In this part we omit the manipulation of the coefficients but immediately inverse transform to complete the procedure, which is done by:

```
mySphereMeshSource->inverseSWaveletTransform();
```

For further process, the reconstructed function can be retrieved by the following code:

```
std::vector< double > reF =  mySphereMeshSource->GetReconstructedScalarFunction();
```

The immediately reconstructed function should be a copy of the original one. Here, to see the fidelity of the transformation pair, we calculated the $\infty-$norm of the difference between original and reconstructed function. The result is around 10^{-16}.

2.4 Advanced Usage: Forward Transform, Coefficient Modification/Filtering, Backward Transform

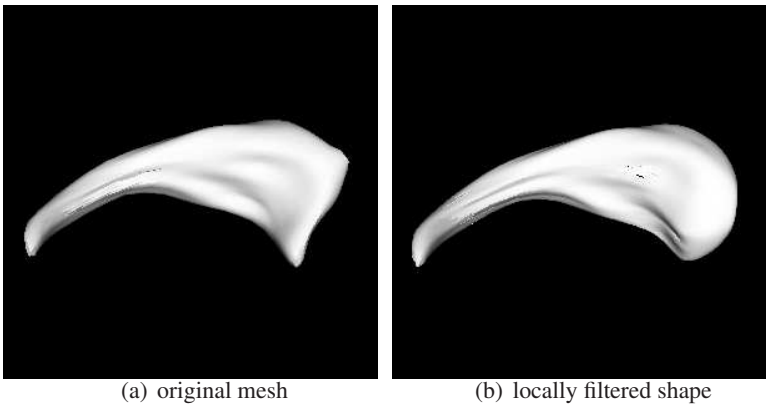
Such technique is analogous to frequency domain signal filtering. For instance, one can filter out those high frequency component in the function, leaving a denoised version of the signal. The advantage of spherical wavelet analysis lies in that the filtering can be done both locally in frequency and spatial domain. This is one of the advantages of wavelet over Fourier based methods.

The coefficients have to be first retrieved then filtered. This can be done by

```
std::vector< std::vector< double > > GetCoefficients( void );
```

It returns all the coefficients in one big structure. The elements of the structure are vectors. The first vector stores all the scaling coefficients while the wavelet coefficients of different resolutions, are in following vectors. Users can modify the returned value and then feed it back for reconstruction, or build a new one of the same structure. Also, several convenient ways are provided to modify the coefficients directly, they are:

```
void SetAllCoarsestScalingCoefficients( std::vector< double > );
void SetAllCoarsestScalingCoefficients( double );
void SetScalingCoefficientAtCoarsestScale( int, double );
void SetAllWaveletCoefficientsOfAllScales( double );
void SetAllWaveletCoefficientsAtOneScale( int scale, double dWaveletCoeff );
void SetWaveletCoefficientAtScale( int scale, int idx, double waveletCoeff );
```



(a) original mesh (b) locally filtered shape

Figure 3: Locally filtering the shape

One can tell the functions and usage by their names and arguments.

After manipulation in the wavelet domain, users can reconstruct the signal according to the procedures mentioned in the previous section.

Visualization of a Basis Function

Till now we already have all we need to visualize the basis functions. Simply set the coefficient we want to one and all the others to zero, followed by inverse transform and we'll have the basis function defined on the mesh.

Two remarks are to be made here. First, the object keeps an internal flag indicating whether the wavelet transform has been done. Only when it's true, further modification of coefficients or reconstruction can be applied. Thus we have to first decompose an (arbitrary)signal to make this flag true and then modify the coefficients and reconstruct.

Secondly, there are two formats for the output scalar function. One is just a list of floating numbers and the other format is preceded by some header information for KWVisu[6], an open source mesh visualization software, to load in and to display. Figure 1(a) is an instant of scaling function at the coarsest level with the highest level to be 5.

By similar approach we can visualize wavelet functions at different resolution level. `itkSWaveletTest4.cxx` gives and example for this.

3 Applications

As an example, we show how the spherical wavelet transform object can be applied to the shape analysis area. The surface of Caudate Nucleus as the original shape.

A shape with spherical topology can be mapped to a sphere thus having the spherical parameterization [9]. After mapping, the x, y and z coordinates of the shape can be treated as three separate scalar functions defined on the sphere and thus can be analyzed by the technique presented here. This can be found in the `filteringShape.cxx`.

In the example, to highlight the local processing property of wavelet transform, we only smooth the scalar functions, the x, y and z coordinates of the caudate surface, around the head region of the shape. The tail part of the shape is kept intact. Thus the head becomes blunt but the sharp tail is preserved. The original shape is shown in Figure 3(a) and its locally filtered version is in Figure 3(b).

4 Conclusions

The spherical wavelet transformation is a powerful tool for signal processing on the sphere. The paper introduced an open source object for carrying out such transformation and analysis. Hopefully it can help in certain areas of signal processing, computer graphics and shape analysis [5].

Currently the code is not fully templated, e.g. the scalar function is defined to be double type since in most cases we need high accuracy for the the scalar function. However, if needed, this can be a direction for further improvement.

References

[1] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1998. 1.1

[2] D. Nain, S. Haker, A. Bobick, and A. Tannenbaum. Multiscale 3d shape analysis using spherical wavelets. In *MICCAI*, LNCS 3750, pages 459–467, 2005. (document)

[3] D. Nain, S. Haker, A. Bobick, and A. Tannenbaum. Shape-driven surface segmentation using spherical wavelets. *MICCAI*, pages 66–74, 2006. (document)

[4] D. Nain, S. Haker, A. Bobick, and A. Tannenbaum. Multiscale 3d shape representation and segmentation using spherical wavelets. *IEEE Trans. Med. Imaging, Special Issue on Computational Neuroanatomy*, 2007. Accepted, publication date: April 2007. (document)

[5] D. Nain, M. Styner, M. Niethammer, J. J. Levitt, M. E. Shenton, G. Gerig, A. Bobick, and A. Tannenbaum. Statistical shape analysis of brain structures using spherical wavelets. In *IEEE International Symposium on Biomedical Imaging (ISBI)*, 2007. (document), 4

[6] I. Oguz, G. Gerig, S. Barre, and M. Styner. Kwmeshvisu: A mesh visualization tool for shape analysis. *MICCAI 2006 Open Source Workshop. Insight Journal*, 2006. Insight Journal DSpace link: http://hdl.handle.net/1926/220. 2.1, 2.4

[7] P. Schröder and W. Sweldens. Spherical wavelets: Texture processing. In *Rendering Techniques '95*. Springer Verlag, 1995. (document), 1.2, 1.2

[8] Peter Schröder and Wim Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. *Computer Graphics Proceedings (SIGGRAPH 95)*, pages 161–172, 1995. (document), 1.2

[9] M. Styner, I. Oguz, S. Xu, C. Brechbuehler, D. Pantazis, J. Levitt, M. Shenton, and G. Gerig. Framework for the statistical shape analysis of brain structures using spharm-pdm. *MICCAI 2006 Open Source Workshop. Insight Journal*, 2006. Insight Journal DSpace link: http://hdl.handle.net/1926/215. 3

[10] L Tenorio, A. H Jaffe, S Hanany, and C. H Lineweaver. Applications of wavelets to the analysis of cosmic microwave background maps. *Monthly Notices of the Royal Astronomical Society*, 310(3):823, December 1999. (document)

[11] Ze Wang, Chi-Sing Leung, Yi-Sheng Zhu, and Tien-Tsin Wong. Data compression with spherical wavelets and wavelets for the image-based relighting. *Comput. Vis. Image Underst.*, 96(3):327–344, 2004. (document)

[12] P. Yu, X. Han, F. Segonne, R. L. Buckner, R. Pienaar, P. Golland, P. E. Grant, and B. Fischl. Cortical surface shape analysis based on spherical wavelet transformation. In *IEEE Computer Society Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA)*, June 2006. (document)

[13] P. Yu, F. Segonne, X. Han, and B. Fischl. Shape analysis of neuroanatomical structures based on spherical wavelets. In *Human Brain Mapping (HBM)*, 2005. (document)

Fast BlockMatching Registration with Entropy-based Similarity

Release 0.00

Eduardo Suárez-Santana[1], Rafael Nebot[1],
Carl-Fredrik Westin[3] and Juan Ruiz-Alzola[1,2]

June 25, 2007

[1]Canary Islands Institute of Technology
[2]Department of Signals & Communications, University of Las Palmas de Gran Canaria
[3]Laboratory of Mathematics in Imaging, Brigham and Women's Hospital and Harvard Medical School

Abstract

This paper describes the implementation of a multidimensional block-matching nonrigid registration algorithm. The main features of the algorithm are its simplicity, its free form nature, the modularity of the similarity measure, which makes it possible using local entropy-based similarity measures and the avoidance of the optimization module. The algorithm implementation described in this paper is based on the method by Suarez et al. [5, 3]. This paper, which has already been submitted to the Insight Journal, is accompanied with the source code, input data, parameters and output data used for validating the algorithm described in it.

Contents

1 The Algorithm **2**

2 The Implementation **3**
2.1 Source code . . . 3
2.2 Use of Not-A-Number . . . 4

3 Future Work **5**
3.1 The implementation . . . 5
3.2 The algorithm . . . 5

4 Results **6**

5 Conclusions **6**

6 Acknowledgements **6**

Nonrigid registration using entropy-based similarity measures is most often accomplished in three ways: (1) using global optimization, (2) using a differential similarity measure and (3) using a block-matching scheme. Examples and implementations of (1) and (2) are found in the Insight Toolkit [1]. However, block-matching schemes are rarely found in literature to perform this task.

Global optimization facilitates the implementation of a wide range of registration models. However, it makes finer meshes computationally more expensive, it requires the right choice of parameters and optimization algorithm, and it becomes difficult to parallelize.

Differential similarity measures is another good approach. It can be fast and accurate. Nevertheless, the numerical approach and its implementation is usually complex for similarity measures based on entropy, the number of iterations may be large because of the propagation of the displacement information, it needs the computation of derivatives which may need smoothing steps for medical images, and boundary conditions may lead to non-free-form implementations.

Block-Matching is probably less elegant from a theoretical point of view, but it shows several nice features:

- It is simple to formulate
- It is simple to implement
- It is difficult to get a numerical instability
- It allows parallelization
- It is robust against noise
- Few parameters are in general required
- It is suitable for hardware implementations

Perhaps these are some of the reasons why the MPEG standard for video uses a block-matching scheme as well. However, some tricks are needed in order to make it fast and to compute local estimations of entropy-based similarity measures.

The algorithm implemented includes several contributions already published by Suarez et al. [5, 3]. A review of this algorithm is given in section 1, and section 2 describes the details of the implementation. Future work follows after this section in order to introduce the improvements of next releases. Section 4 presents the experiment and its result attached to the code.

1 The Algorithm

The registration algorithm consists of a pyramidal block-matching scheme [5]. A block is defined as a neighborhood of voxels around a selected one. For the explanation of the algorithm, we will follow figure 1, which is the representation of one level of the pyramid.

First, *Image B* is sampled in the positions of the samples of *Image A* to get *Image B Transformed*. Then, in the *Data Matching* step, for every voxel in *Image B Transformed*, a block is taken which then is matched against a set of test blocks in *Image A*. The displacement that achieves the best similarity between blocks, is stored as the displacement for the original voxel. With the multi-resolution scheme used, it is only necessary to test displacements of one voxel. It can be shown that given a point-wise similarity measure, it is equivalent to switch between slow block matching per voxel and a faster convolution for every displacement.

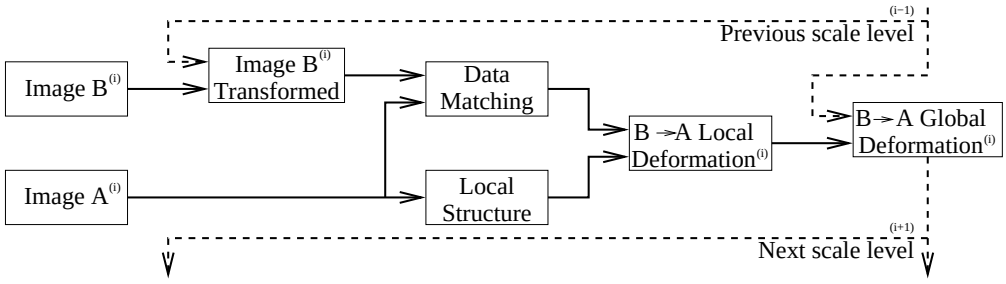


Figure 1: Algorithm pipeline for pyramidal level (i).

Next, the discrete vector field is regularized weighted by *Local Structure* in *Image A*, in order to speed up the propagation of the displacement information (*B→A Local Deformation*). Finally, composition of local deformation field for the working level with the global deformation field for the previous level gives the global deformation field for this level (*B→A Global Deformation*).

A point-wise similarity measure based on entropy is calculated by estimating the joint probability density function (pdf) of the whole images $\mathsf{p}(i_A, i_B)$. Then, for every pair of matching voxels $(A(x^1,x^2,x^3), B(y^1,y^2,y^3)))$, we can measure its point-wise joint density as $\mathsf{p}(i_A(x^1,x^2,x^3), i_B(h(x^1,x^2,x^3)))$. In this way, the mutual information on a block of N matched samples $(i_A[k], i_B[k])$ would be:

$$MI_{block}(I_A, I_B) \backsimeq \sum_{k=1}^{N} \log \frac{\mathsf{p}(i_1[k], i_2[k])}{\mathsf{p}(i_1[k])\mathsf{p}(i_2[k])} \quad . \tag{1}$$

This is similar to divide the number of samples in two groups: the former used to estimate the pdf and the latter to evaluate it. In this case, the former would be the whole matched image and the latter the samples in the block. A full explanation of this technique can be found in [4].

2 The Implementation

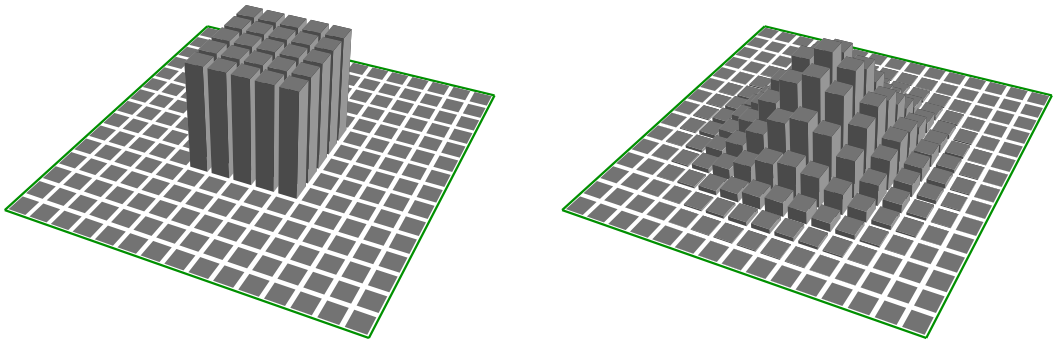
The algorithm has been implemented using the Insight Toolkit [1]. Hence, you need to have the following software installed:

- Insight Toolkit 2.8
- CMake 2.4

To help the readers of the source code, some notation will be introduced. We are registering images A attached to the reference system X, and B attached to the reference system Y. X is the reference system where the matching is done.

2.1 Source code

Every iteration (shown in figure 1) is performed by the method `UpdateDeformation`. As input we receive the deformation field `initialDeformationInX`, and the images `ImageAInX` and `ImageBInY`. There, the method `LinearResampleIntoX` moves B to the reference system X (*Image B Transformed* in the figure).



(a) Square kernel (w=2) (b) Gaussian kernel (σ_{sim}=2.2)

Figure 2: Kernels for convolution with point-wise similarity images.

Data Matching takes place inside `Update Deformation` as well. There, for every unitary test displacement, is carried out point-wise similarity. We have implemented Mutual Information in a method called `SimilarityMI`. Block Matching, or its equivalent implementation as a convolution with a square window, has been carried out with Gaussian smoothing, because it is faster and has less aliasing (see figure 2). *Local Structure* is implemented in the method `Structure`. Weighting with local structure is done with the method `SmoothWithCertainty`. The typical deviation (σ) of these two Gaussian smoothing steps, similarity and regularization, are the only parameters needed for the algorithm. Default values are used in the constructor to facilitate the use of the class.

Composition of local with initial deformation to get the global deformation is done by sampling the initial deformation field. To resample this field it is necessary to transform the displacement vectors into mesh positions:

```
vect = initialDeformationInXIterator.Get();
initialDeformationInX->TransformIndexToPhysicalPoint(
  initialDeformationInXIterator.GetIndex(),
  position );

for ( int coordinate = 0; coordinate < ImageDimension; ++coordinate )
  vect[ coordinate ] += position[ coordinate ];

initialDeformationInXIterator.Set( vect );
```

Then, method `SmoothWithCertainty` is used before transforming the mesh back to displacement vectors.

2.2 Use of Not-A-Number

This algorithm makes intensive use of the signal/certainty philosophy and of the *Normalized Convolution* algorithm [2]. Representation of the unknown voxel values has been done using the value NaN. Some modifications to `itkWarpImageFilter` have been necessary to reuse this class. Hence, using NaNs to pad images will make consistent the representation of NaN values for samples where no information is available. This trick allows saving memory and the free-form behaviour of the deformation field. To use the normalized convolution algorithm, it is necessary to decouple signal and certainty from these images. Hence, the `StripNanImage` has been implemented.

3 Future Work

This is a first release of the algorithm. Some optimizations can still be done, both in the algorithm and in the implementation.

3.1 The implementation

We have done our best to implement the algorithm using the Itk Coding Style. However, Itk kernel developers will need to have a look at it to integrate it with the rest of Itk. Next there are some items to take into account:

- **Use of an initial transformation**. The algorithm is almost prepared for that, but this feature is not available at the moment.
- **Efficient use of symmetric matrices**. At the moment, there is not such implementation in Itk.
- **Efficient use of image adaptors**. The similarity measure, as an example, could be coded using them.
- **Independence of the normalized convolution**. It has been coded inside the class. It should be coded as a separate one.
- **Independence of the similarity measure**. It has been coded inside the class as well and they should be coded as separate classes.
- **Adaptation for interaction**. The algorithm needs to be adapted to be suitable for user interaction. In that way, a user would set some control points that would modify signal/certainty in the regularization step. This would lead the registration process, following the indications of a physician.

3.2 The algorithm

Despite the algorithm may be modified to be more efficient, there are some direct optimizations to be made in following releases:

- **Use of another pyramidal scheme**. The algorithm makes use of the `MultiResolutionPyramidImageFilter` algorithm already implemented in Itk. In this algorithm, the resolution of each level is reduced an integer factor from the original data. Having a float factor would allow more levels and it would increase the quality of the registration.
- **Use of a reduced neighborhood**. Neighborhood information is needed in differential registration algorithms to compute gradients. This algorithm makes use of a neighborhood of radius one. In 3-D that means to test 27 possible displacements to get the right one. This could be reduced to 7 test displacements (two in each direction plus the center), and get the final displacement as a composition, in contrast to gradient estimation, that would need at least test 4 positions (one in each direction plus the center).

4 Results

Because of the size of medical data, only a 2-D example has been uploaded to the Insight Journal. The original image (`data_original_eduardo256.jpg`) is a picture of the first author.

The experiment consists of deforming the original image with a synthetic deformation field (`data_input1_odraude256.jpg`) and then registering them back with the registration algorithm. To test the Mutual Information similarity measure, the intensity of the original image has been modified by a sin function and some noise has been added (`data_input0_fevbsep256.jpg`). So that, `data_input0_fevbsep256.jpg` and `data_input1_odraude256.jpg` are the images to be registered. Images are shown in figure 3.

The registration process took, on a 3.4 GHz intel Pentium running linux, about eight seconds. With the output deformation field, the moving image (`data_input1_odraude256.jpg`) has been moved back with the deformation field, resulting in the registered image `data_outputAuthor_odraude256def.mha`. Simple visible inspection is the guarantee of the success of the algorithm.

5 Conclusions

This paper has presented the implementation of a block matching algorithm for registration of medical images. Since there is a lot of improvements and future work, we are on an early stage to compare it with other registration algorithms. Our experience is that block matching can be fast, and thus may have an advantage in certain applications. Furthermore, the use of a certainty function would make it possible to take care of the user interactions to lead the registration process by setting points of anatomic relevance or homologous points.

6 Acknowledgements

This work has been partially supported by the grant from the Spanish Ministry of Science and Technology PTQ2004-1444. This work is part as well of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149, and Neuro Analysis Center (NAC) NIH P41-RR13218. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics. Special thanks to Dan Blezek and Luis Ibañez for their hints to this implementation.

References

[1] L. Ibanez, W. Schroeder, L. Ng, and J. Cates. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, first edition, 2003. (document), 2

[2] H. Knutsson and C-F. Westin. Normalized and differential convolution: Methods for interpolation and filtering of incomplete and uncertain data. In *Proceedings of Computer Vision and Pattern Recognition*, pages 515–523, New York City, USA, June 1993. IEEE. 2.2

(a) Target image:
data_input0_fevbsep256.jpg

(b) Moving image:
data_input1_odraude256.jpg

(c) Original image:
data_original_eduardo256.jpg

(d) Registered image:
data_outputAuthor_odraude256def.mha

Figure 3: Images in the experiment.

[3] Eduardo Suárez. *Un marco general para la normalización geométrica de datos tensoriales y su aplicación al registrado de imágenes médicas*. PhD thesis, University of Las Palmas de Gran Canaria, jul 2003. (document)

[4] Eduardo Suárez, Jose A. Santana, Eduardo Rovaris, Carl-Fredrik Westin, and Juan Ruiz-Alzola. A fast mutual information estimation for nonrigid registration. In *Proceedings of the 9th Internation Workshop on Computer Aided Systems Theory (EUROCAST '03)*, Cast and Tools for Complexity in Biological, Physical and Engineering Systems, pages 89–93, Las Palmas de Gran Canaria, Spain, February 2003. 1

[5] Eduardo Suárez, Carl-Fredrik Westin, Eduardo Rovaris, and Juan Ruiz-Alzola. Nonrigid registration using regularized matching weighted by local structure. In *Proceedings of the Fifth International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI '02)*, number 2 in Lecture Notes in Computer Science, pages 581–589, Tokyo, Japan, September 2002. (document), 1

Optimizing ITK's Registration Methods for Multi-processor, Shared-memory Systems

Release 3.0

Stephen Aylward, Julien Jomier, Sebastien Barre, Brad Davis, and Luis Ibanez

September 10, 2007

Kitware, Inc.
http://www.kitware.com

Abstract

This document describes work-in-progress for refactoring ITK's registration methods to exploit the parallel, computation power of multi-processor, shared-memory systems. Refactoring includes making the methods multi-threaded as well as optimizing the algorithms. API backward compatibility is being maintained. Helper classes that solve common registration tasks are also being developed.

The refactoring has reduced computation times by factors of 2 to 200 for metrics, interpolators, and transforms computed on multi-processor systems. Extensive sets of tests are being developed to further test operation and backward compatibility.

More information on this project is available at:
http://www.na-mic.org/Wiki/index.php/ITK_Registration_Optimization

NOTE #1: Recent changes to ITK and BatchMake in preparation for the release of ITK v3.4 and in support of Slicer have caused the build of this project to fail. Please visit this project's dashboard (available via the link above) prior to downloading the code to ensure that the code is working.

NOTE #2: The software is being incorporated directly into ITK. It should be available via ITK's CVS approximately one month after the release of version 3.4 of the Insight Toolkit (October/November 2007). The above wiki page will contain up-to-date information.

Contents

1 Introduction **2**

2 Background and Significance **3**

3 Techniques Used in Development and Testing **3**

4 Results **7**

5 Discussion **12**

6 Acknowledgements **13**

1 Introduction

The work-in-progress described in this paper focuses on refactoring ITK's registration framework [Ibanez 2002] to take advantage of the parallel, computation capabilities of multi-processor, shared-memory systems. The refactoring consists of multi-threading the methods as well as optimizing the algorithm implementations. Simultaneously, backward compatibility with ITK's existing classes must be maintained.

Regarding multi-threading, while ITK's execution pipeline for filters supports multi-threading, ITK's registration framework is not multi-threaded and several of its methods are not thread safe. Many of the registration modules presented in this paper, on the other hand, make heavy use of multi-threading, and all are thread safe.

Regarding algorithm optimization, it should be noted that ITK's registration methods were built to support research, so their implementations emphasized easy-to-understand code and a plug-and-play framework. ITK's registration framework is illustrated in Fig. 1. Modules can be substituted into the transform, interpolator, metric, and solver components. In contrast, the work presented in this paper emphasizes computational speed, i.e., certain implementations have been specialized to speed specific combinations of modules. For example, all image-to-image metrics now detect if they are being combined with a cubic b-spline transform, and in those situations, appropriate pre-computations and algorithmic shortcuts are taken. Nevertheless, in keeping with the policies of the Insight Software Consortium, backward compatibility with ITK's existing API was maintained.

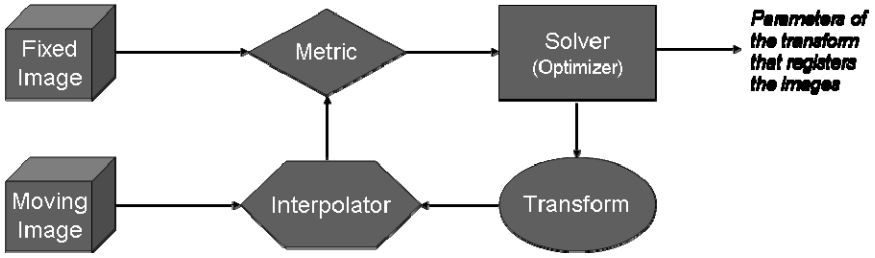


Figure 1. *ITK's registration framework. Modularity promotes research and experimentation. For optimization, specific combinations of modules are being integrated to speed specific applications.*

ITK's registration methods needed to be refactored because of the growing role of registration in medical image analysis and the extensive registration computation time (over 15 hours) that has been anecdotally reported when using ITK. As driving problems, we selected registration for atlas formation and registration for atlas-based segmentation, particularly as featured in the EMSegmenter algorithm developed by Dr. Kilian Pohl [Pohl 2006]. Dr. Pohl's algorithm is widely used as the EMSegmenter module in Slicer (http://www.slicer.org). His algorithm relies on the inter-subject mapping of cortical MRIs. An analysis of this driving problem led us to focus on the following ITK modules and their combinations:

Transforms:	Rigid, affine, and BSplines
Interpolators:	Linear and cubic-BSplines
Metrics:	Mattes mutual information [Mattes 2003] and mean-squares error

Solvers: One-plus-one evolutionary optimization [Styner 2000] and LBFGSB (Limited memory Broyden Fletcher Goldfarb Shannon minimization with simple Bounds) optimization [Zhu 1997]

Beyond the above driving problem, the work presented in this paper includes updates to every transform, interpolator, and image-to-image metric in ITK due to base-class modifications needed to support thread safety. Ongoing work is focusing on further speed improvements and extensive testing.

Further details on the background and significance for the work are given next. Then the optimization techniques and the measures used to quantify their performance are presented. The performance improvements gained on two different multi-processor platforms are then presented. The conclusion discusses future work. Additional and updated details are available on the web at the following link:

http://www.na-mic.org/Wiki/index.php/ITK_Registration_Optimization.

2 Background and Significance

By focusing on multi-processor, shared-memory systems, the work presented herein addresses the current trend in computer hardware for laptops, desktops, workstations, computer servers, and grid nodes. In those systems, the trend is to use multi-core processors such as Intel's Core2Duo chip. Intel estimates that over 85% of the processors it is producing are multi-core processors. Today, nearly every new desktop contains a multi-core processor, and mid-range ($4,000) compute servers typically have eight dual-core processors. On a server with eight dual-core processors, 16 independent tasks can simultaneously process a common dataset – thereby potentially generating a result 16 times faster. Intel predicts that within three years, 32-core processors will be the norm – thereby suggesting desktops with 32 cores and compute servers with 256+ cores.

Distributing a task across the processors and cores of a system is accomplished using multi-threading. Typically, threads are sections of a program that can be computed independently and simultaneously. Defining a multi-threaded implementation of an algorithm requires consideration of how information will be allocated to, computed on, and recombined from the threads. Some algorithms are easy to implement in a multi-threaded manner while other algorithms cannot be effectively multi-threaded. Given the multi-core future of computer hardware, algorithm researchers as well as algorithm implementers need to understand multi-threading.

The techniques used to develop optimized registration modules for ITK are summarized next.

3 Techniques Used in Development and Testing

This ongoing project involved code optimization and threading, process prioritization and timing, and comparative quantification of performance gains.

3.1 Optimizations and Threads

The work began by focusing on non-rigid registration and then spread to include other registration challenges. Numerous small changes were made to the code. The major changes are as follows:

1) Developed multi-threaded versions of ITK's image-to-image metrics. Mattes mutual information metric, the mean-squared error metric, and similar voxel-matching metrics now derive from a common image-to-image-metric base class that provides:

    a. Sampling strategies for computing the metric using a (sub-)sample of points in the images: (i) random sub-sampling, (ii) user-specification of the sub-sample points, or (iii) using every voxel in the images. Furthermore, (iv) a mask can be used to limit the domain of (i) random sub-sampling and (iii) using every voxel.

    b. Distributing the (sub-)sampled points across threads when computing the metric,

    c. Threading the pre-computation of a gradient image to speed metric derivative computation,

    d. Caching the affine pre-transform of the (sub-)sampled points when a metric is combined with a BSpline transform,

    e. Pre-computing the BSpline coefficients for the (sub-)sampled points when a metric is combined with a BSpline transform,

    f. Providing "hooks" so that derived classes can conduct threaded computations in three synchronized phases. This approach was chosen to avoid the use of mutex locks – additional details are given below.

2) Made ITK transforms thread safe. While most ITK transforms do not need to use member variables to transform a point, kernel-based transforms (such as the b-spline and thin-plate-spline trasnforms) benefit from using member variables to avoid having to repeatedly allocate and free large data structures that hold intermediate values. ITK's original implementation of select transforms, such as many of the kernel-based transforms, were not thread safe because the use of those intermediate-value member variables was not thread-safe. In the presented implementations, the base itkTransform class was modified so that all transforms may have thread-specific data.

3) Identified and optimized frequently used code segments. Several commonly used and computationally expensive routines were identified using Lightweight Technologies' LTProf (http://www.lw-tech.com/), Linux's Valgrind (http://valgrind.org) and Intel's VTune (http://www.intel.com) profiling tools. For example, ITK's image Fill() method uses a pixel-by-pixel iterative assignment technique. In registration, images are often used to store intermediate values such as joint histograms. As a result, every value and derivative computation was making multiple calls to Fill(), typically to set those values/images to zero. Given that derivatives of joint histograms equate to images with 30,000+ pixels, the Fill() method was accounting for nearly 10% of the time spent on a metric value computation. Replacing Fill() with a single system-level call (memset) reduced computation time by nearly 10%. Similar savings were discovered for other frequently called routines.

Mutex lock checking was determined to be extremely time consuming on SunOS and other platforms. The initial implementations made use of mutex locks. Experiments revealed, however, that even when mutex collisions did not occur, the computation time of a single mutex check per sample exceeds the per voxel time for computing a Mattes Mutual Information metric value.

Instead of using mutex locks, the current implementations create unique copies of relevant variables for each thread and then join their results after thread completion. The challenge is to manage the allocation of data to those variables and the integration of the results from those variables. If that allocation and integration is conducted in the main thread, those costs (i.e., computation times) can exceed the benefits of threading. To alleviate those costs, the implementations provide the "hooks" for three, synchronized sets of threads, as discussed above in Optimization, 3.1.1f. Typically, first one set of threads is run to initialize the thread specific variables. The second set of threads is then run to process the samples allocated to each thread and accrue their results in their thread-specific variables. The third set of threads is then run to combine the results from the thread-specific variables into the final result. A set of threads is not begun until the prior set of threads completes. Consider, for example, that for Mattes Mutual Information computation there is a joint histogram variable. To partition the samples to different threads, each thread is given its own copy of the joint histogram variable. During the first set of threads, each thread initializes its joint histogram to zero. During the second set of threads, the each thread's samples are inserted into that thread's histogram. During the third set of threads, the summation of the total joint histograms is threaded – that is, each thread is responsible from summing a different region of the total joint histogram from the thread-specific histogram variables. Different initialization, allocation, and integration strategies are used for different variables.

Our work also revealed that certain changes and parameterizations could improve performance on one platform and degrade performance on others. For example, one surprising observation was that using more threads than processors had negligible effect on computation speed on certain machines, while on other platforms using more threads than available processes resulted in significant degradation in performance. Additionally, the cost to initiate a thread on certain platforms is much greater than on other platforms.

The complexity of the optimization task necessitated establishing an infrastructure that verified backward compatibility and monitored the effects of the day-to-day code and parameter changes for multiple platforms. That infrastructure is discussed next.

3.2 Backward Compatibility and Performance Monitoring

The registration methods presented were implemented to maximize their backward compatibility with and speed relative to the standard ITK implementations. Backward compatibility was judged by providing consistent results and maintaining a consistent API with respect to the standard ITK implementations. Speed was quantified using real-world ("wall clock") time. These criteria were tested as follows.

Backward Compatibility: To ensure consistent software, i.e., having a backward compatible API and generating similar results, the infrastructure included iterative, interleaved testing of standard ITK methods and the optimized methods. The consistency of these tests was controlled by writing them as C++ frameworks that used symbolic placeholders where the registration methods were left unspecified. CMake's CONFIGURE_FILE command was then used to substitute calls to the standard and optimized methods at the symbolic placeholders. Tests of the standard ITK methods were interleaved with tests of the optimized methods to quantify and thereby enable compensation for workload fluctuations on the testing platforms. The standard tests also served to verify that the optimized methods produced similar results, as explained next.

The optimized methods, for any given number of threads, produce consistent results, and in most cases, when run using a single thread, produce the same results as the standard methods. However the optimized methods' results may vary as the number of threads is varied. Result variation by number of

threads arises from across-thread problem partitioning. As mentioned previously, threading often necessitates the use of additional intermediate values for each additional threads used. Such changes in intermediate processing may have an effect on the accumulated values produced, in part due to machine precision. Such variations are likely to have a negligible effect on the result from a single registration optimization iteration, but their cumulative effect over many iterations may be greater. Nevertheless, the implementations, computations, and final outcomes are still valid.

For reasons of consistency as explained above, and to ensure backward compatibility for user-derived (potentially not thread-safe) code that is not distributed with ITK, the optimized methods default to running single threaded. The user must call `metric->RunMultiThreaded(void)` to run with the ITK defined default number of threads. The ITK defined default number of threads is the lesser of 8 or the number processors on the system, see Insight/Code/Common/itkMultiThreader.h. The user may also set the number of threads by calling `metric->SetNumberOfThreads(int)`. Either call must be made prior to calling `metric->Initialize(void)`.

Performance Monitoring: There were two components to process monitoring: speed quantification and centralized reporting. This project made significant developments in each area.

For this project, speed quantification involves code instrumentation, process prioritization, and workload compensation. Code instrumentation refers to the explicit use of start and stop timing calls in the tests. Time for loading/creating the fixed and moving images and for initializing the registration framework is not included in the timing results. The timers are provided by ITK's high performance, cross-platform timing routines (Insight/Code/Common/itkRealTimeClock.h). Since the tests used real-world timers and since the workload placed by other users on the test platforms could not be controlled and varied day to day, process prioritization and workload compensation are used. Specifically, to mitigate some of the effects of concurrent users, an itkHighPriorityRealTimeClock class (located in BWHITKOptimization/Code/Utilities) is defined. When an instance of that class is instantiated in a program, the process' priority is increased. When destructed, the process' priority is returned to normal. Furthermore, as previously mentioned, the standard ITK methods' tests are interleaved with the optimized methods' tests. The speed of the standard ITK methods is thus available to normalize the speed of the optimized methods. Thereby, three measures of performance are reported.

The first measure is the absolute (real-world) run time of the method, Equ. 1. In addition to the number of threads, if a metric is being measured then its time is typically parameterized by the number of samples used to compute the metric, the type of interpolation used, the transform used, and the number of metric value or derivative computations performed. If a transform is being measured, then its time is typically parameterized by the type of interpolation used and the number of point transformations performed.

$$T_n = A + B/n \tag{1}$$

In Equ. 1, A is the portion of the method that cannot be threaded and B is portion of the method that can be threaded.

The second measure is often called *speedup* or *Amdahl's law* [Amdahl 1967].

$$C_n = T_1 / T_n = (A + B) / (A + B/n) \tag{2}$$

Amdahl's law states that the speedup of a threaded algorithm will eventually begin to decrease as additional threads are used, since the serial component remains constant and since the above equation does not account for the cost of distributing data to and integrating data from the parallel processes. Such behavior is revealed in the results presented in this paper.

The third measure is called *optimization ratio*, Equ. 3. It compares the absolute run time of the unoptimized (single-threaded) version of a method with its optimized version running in *n* threads. This metric compensates for workload variations and partially compensates for cross-platform differences in processor type and speed.

$$R_n = T_1(\text{unoptimized}) / T_n(\text{optimized}) \quad (3)$$

The above three metrics only require the computation of the real-world run times of the unoptimized and optimized methods on the testing platforms. The challenge is managing the variety of tests and their parameters. We developed a novel system for managing these tests, as explained next.

Centralized Performance Reporting: To compute and collect the above measures on the testing platforms, BatchMake (http://www.batchmake.org) and CMake (http://www.cmake.org) are integrated. BatchMake is a cross-platform system for scripting parameter space explorations and batch image processing. BatchMake also includes a web-based reporting system, where results from BatchMake scripts can be collected, monitored, graphed, and compared over the web. By integrating BatchMake with CMake, a cross-platform testing system with centralized reporting is established. Using this system, a cmake build/test sequence now automatically initiates the following:

a. BatchMake's implementation of the Whetstones benchmark is run to compute the speed, i.e., MFLOPS and MIPS ratings, of a single core on each testing platform.

b. BatchMake's system information library is used to determine the number of cores as well as the physical and virtual memory space of each testing platform.

c. Based on the *n* cores on the testing platform as well as its total physical memory, ctest is used to run a set of relevant tests, e.g., using *n* or fewer threads and image sizes less than a pre-defined limit based on total physical memory.

d. Results from each test are automatically sent to a central BatchMake server. The server then generates a multitude of web-based graphs and links those graphs with the CMake dashboard. These graphs reveal the complexities of the high-dimensional parameter space of algorithm parallelization. Linking with CMake dashboards reveals how changes in performance are tied to changes in the code. New graphs can be interactively specified.

A subset of those tests, platforms, and results are presented next.

4 Results

Space and time limit the breadth and depth of the results that can be presented herein. This paper instead focuses on a popular set of registration modules and two mid-range compute servers. The registration module configurations reported in this paper are the following:

1. Mattes mutual information metric with cubic B-Spline transform and cubic B-Spline interpolation

2. Mattes mutual information metric with linear transform and linear interpolation

3. Mean squared error metric with linear transform and linear interpolation

Table 1 lists the machines on which the tests are computed nightly. The tests that are run on each machine vary based on the number of simultaneous threads supported, i.e., number of CPUs x number of cores per CPU = number of simultaneous threads supported. Tbl. 1 reveals that the code works on Apple Mac OSX, Linux 32bit, Linux 64bit, and SunsOS platforms. Furthermore, experimental builds routinely (albeit not nightly) test the code on Microsoft Windows.

Table 1. *Machines on which nightly tests are conducted to track the ITK optimizations being developed.*

Site	Buildname	CPU Speed (MHz)	Number of CPUs	Number of cores per CPU	MIPS	MFLOPS	Total Physical Memory (MB)	Processor Size	Processor Name	Processor Vendor	OS Name	OS Release	OS Version	OS Platform
kw.panzer	MacOSX-gcc4.0-rel-static	1670.0	2	1	1738.3	290.62	1024	32	Unknown P6 family	Intel Corporation	Darwin	8.9.1	Darwin Kernel Version 8.9.1: Thu Feb 22 20:55:00 PST 2007; root:xnu-792.18.15~1/RELEASE_I386	i386
spl.b2_d6_1	Linux64-gcc4.1-rel-static	2792.9	4	1	2270.3	204.37	7900	64	Pentium	Intel Corporation	Linux	2.6.20-1.2316.fc5	#1 SMP Fri Apr 27 19:19:10 EDT 2007	x86_64
John	Linux64-gcc4.1-rel-static	2411.1	8	2	2351.2	330.99	128738	64	Unknown AMD family	Advanced Micro Devices	Linux	2.6.20-1.2316.fc5	#1 SMP Fri Apr 27 19:19:10 EDT 2007	x86_64
kw.fury	Linux-gcc4.1-rel-static	2784.8	1	1	811.38	103.39	1010	32	Pentium	Intel Corporation	Linux	2.6.17-1.2174_FC5	#1 Tue Aug 8 15:30:55 EDT 2006	i686
spl.vision	SunOS-gcc3.0-rel-static	900.00	6	1	255.27	76.413	23984	32	sparcv9	Sun Microelectronics	SunOS	5.8	Generic_117350-46	sun4u
Forest	SunOS-gcc3.0-rel-static	750.00	10	1	218.37	63.870	9896	32	sparcv9	Sun Microelectronics	SunOS	5.8	Generic_117350-46	sun4u

The two machines focused upon in this paper are highlighted: John and Forest. These machines reside within the Surgical Planning Laboratory of the Brigham and Women's Hospital.

- John is a 64-bit machine with 8 dual-core, 2.4, GHz AMD processors; 128 GB of shared memory; and 64-bit Linux kernel 2.6. The tests were compiled using GCC version 4.1 and CMake's "Release" build options. Tests were performed using 1, 2, 4, 8, and 16 threads on this platform.

- Forest is a 32-bit machine with 10 single-core, 750 MHz, Sparc processors; 9 GB of shared memory; and SunOS 5.8. the tests were compiled using GCC 3.0 and CMake's "Release" build options. Tests were performed using 1, 2, 4, and 8 threads on this platform.

Select results from these two test platforms are summarized next. The graphs shown below are taken directly from the nightly "Batchboards" for this work. Additional Batchboard results can be viewed at

http://www.insight-journal.org/batchmake/
(click on the "BWH-ITK Optimization" public project link)

4.1 Mattes Mutual Information Metric with Cubic B-Spline Transform and Cubic B-Spline Interpolation

One view of Mattes mutual information metric run times (Equ. 1) is given in Fig. 2. For these tests, the metric is paired with a B-Spline transform consisting of a 12x12x12 grid of control points and a cubic B-Spline interpolator. These run times are for 5 calls to the `metric->GetValue()` function using 50,000 (red line) and 100,000 (blue line) samples per metric evaluation.

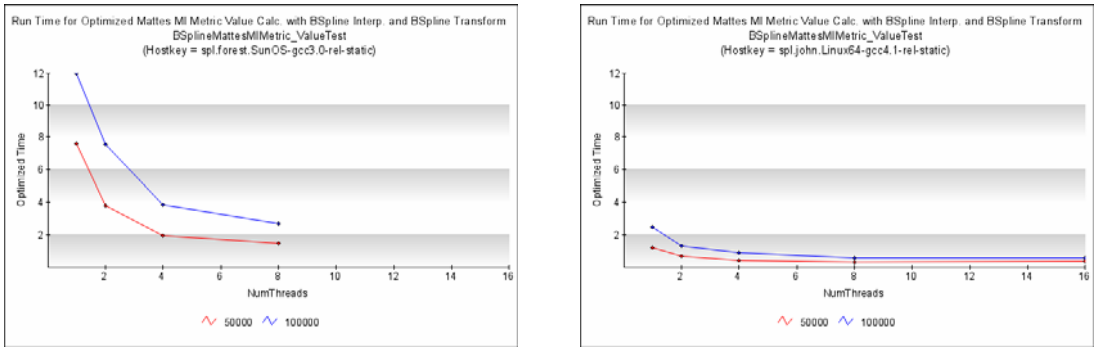


Figure 2. *Mattes mutual information metric evaluation run times for 5 calls to* `metric->GetValue()` *using a B-Spline transform and B-Spline interpolation, for various number of threads. Results for the machine Forest are given on the left and for the machine John on the right.*

The corresponding speedup (Equ. 2) for the optimized metric with 100,000 samples using 8 threads on Forest is C_8 = 12.02 / 3.10 = 3.99. The speedup for the optimized metric using 16 threads on John is C_{16} = 2.22 / 0.65 = 3.42. By Amdahl's law we can expect speedup to eventually decrease as more threads are added.

In Fig.3 the optimization ratios (Equ 3), that compare the optimized method with the standard ITK method, are given. Values greater than one, when one thread is used, indicate that the optimized method is faster than the standard ITK method when run single threaded. On both machines the optimized method is on average approximately 6 times faster than the standard ITK method when 8 threads are used.

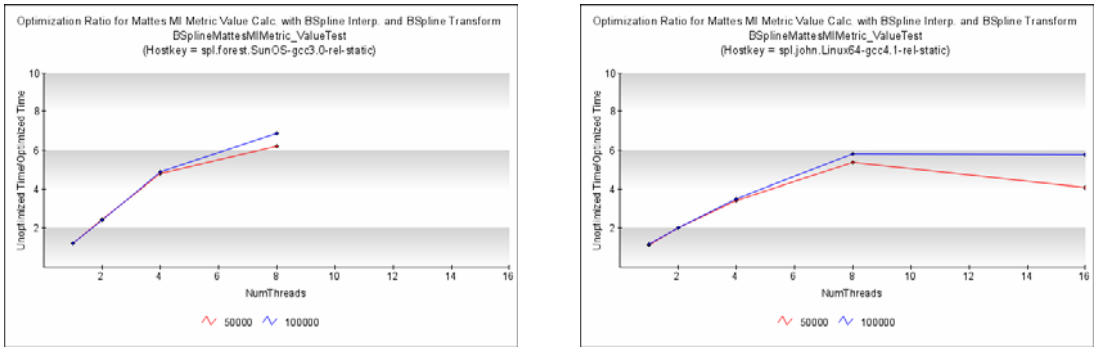


Figure 3*. Optimization ratios (Equ. 3) for Mattes mutual information metric with a B-Spline transform and B-Spline interpolation. Improvement compared to the standard ITK method is better than linear on Forest (left) and only slightly less than linear on John (right).*

A view of Mattes mutual information metric run times and optimization ratios, for five calls to the `metric->GetDerivative()` function, are given in Fig. 4. In these tests, as in the previous set, a 16x16x16 grid of control points is used with the B-Spline transform and linear interpolation is used. These graphs reveal the challenges associated with metric optimization as well as the costs associated with distributing data to and integrating results from multiple threads. Specifically, they show that run times may actually increase as more threads are used.

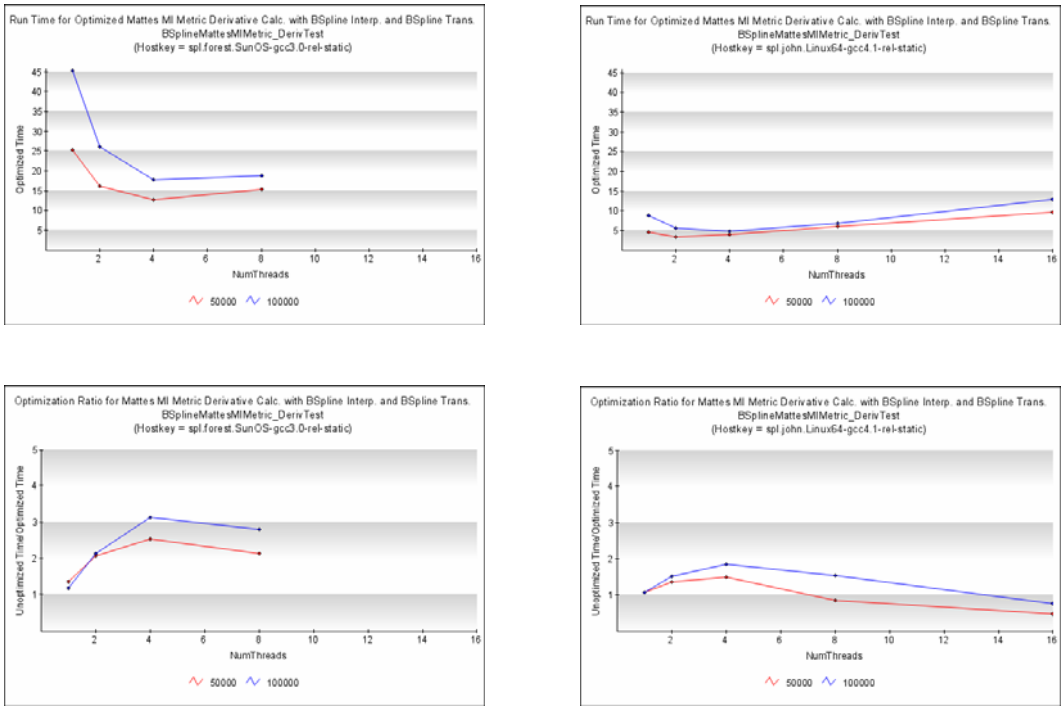


Figure 4. *Run times (top row) and optimization ratios (bottom row) for the* `metric->GetDerivative()` *function of the optimized Mattes mutual information metric using B-Splines for the transform and the interpolator. On both testing platforms (left = Forest, right = John) the best speedup for 100,000 samples was achieved using four threads, e.g., C_4 = 47.0 / 17.8 = 2.6 for Forest. Using additional threads increases run times.*

4.2 Mattes Mutual Information Metric with Linear Transform and Linear Interpolation

The run times in Fig. 5 correspond to tests using the Mattes mutual information metric using a linear (matrix and offset) transform and linear interpolation. The optimization ratios (Equ. 3) were on average approximately 4.0 when 8 threads were used on either platform.

In these tests, unlike the previously reported tests, 10 calls were made to each function, i.e., run times are for conducting twice as much work. While changing the number of calls per test confounds the comparison across methods, that changed was deemed necessary. The motivation is that the runtimes were excessive when using Mattes Mutual Information metric with BSpline transforms and interpolators on certain platforms. Therefore, to reduce testing time, we reduced the number of calls for that setup. However, we also determined that using fewer than 10 calls for the other tests caused those tests to be very susceptible to minor changes in platform workload since those tests using 10 calls already completed in less than one second on most platforms. We regret the confusion, but judged the presented situation to offer an acceptable balance.

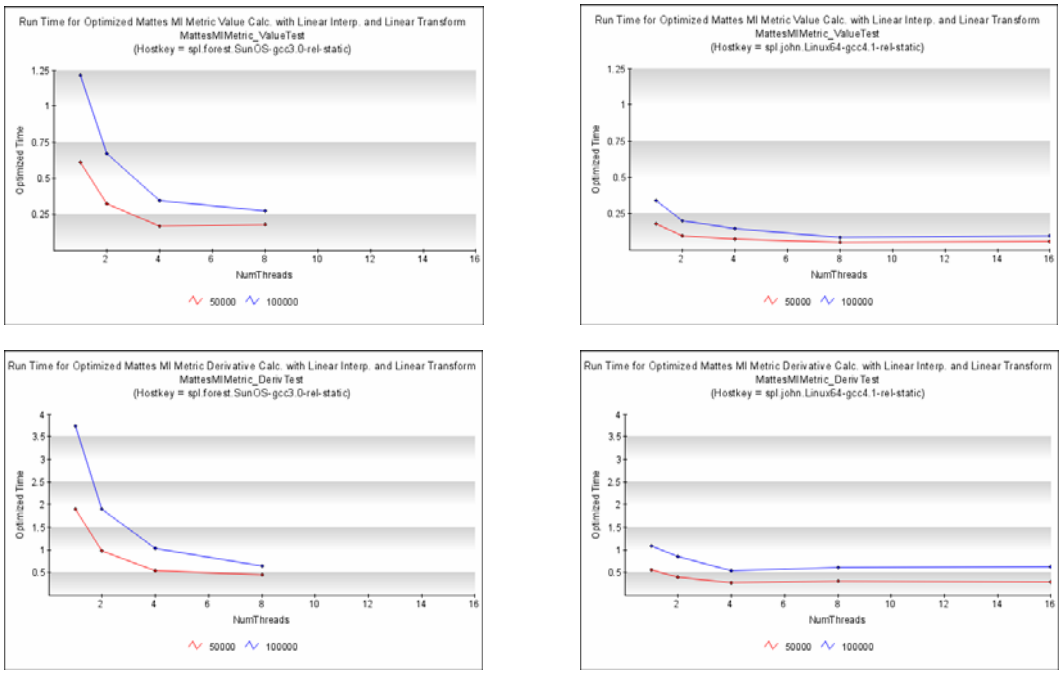


***Figure 5**. Run times for ten calls to `metric->GetValue()` (top row) and ten calls to `metric->GetDerivative()` (bottom row) for Forest (left column) and John (right column). These tests paired Mattes mutual information metric with a linear transform and a linear interpolator.*

4.3 Mean Squared Error Metric with Linear Transform and Linear Interpolation

Fig. 6 presents the run-times for ten calls to `metric->GetValue()` and `metric->GetDerivative()` for the Mean Squared Error metric using a linear transform and linear interpolation.

The standard ITK implementation of the Mean Squared Error metric required the use of every sample in the image when computing the `GetValue()`and the `GetDerivative()` functions. The optimizations performed and the use of a subset of the voxels (sub-sampling) enabled the new metric's optimization ratio (Equ. 3) to typically exceed 300 or more, based on the image size. That is, while ITK's standard implementation prohibited the use of this metric in most situations, the optimized version now makes this metric a viable alternative.

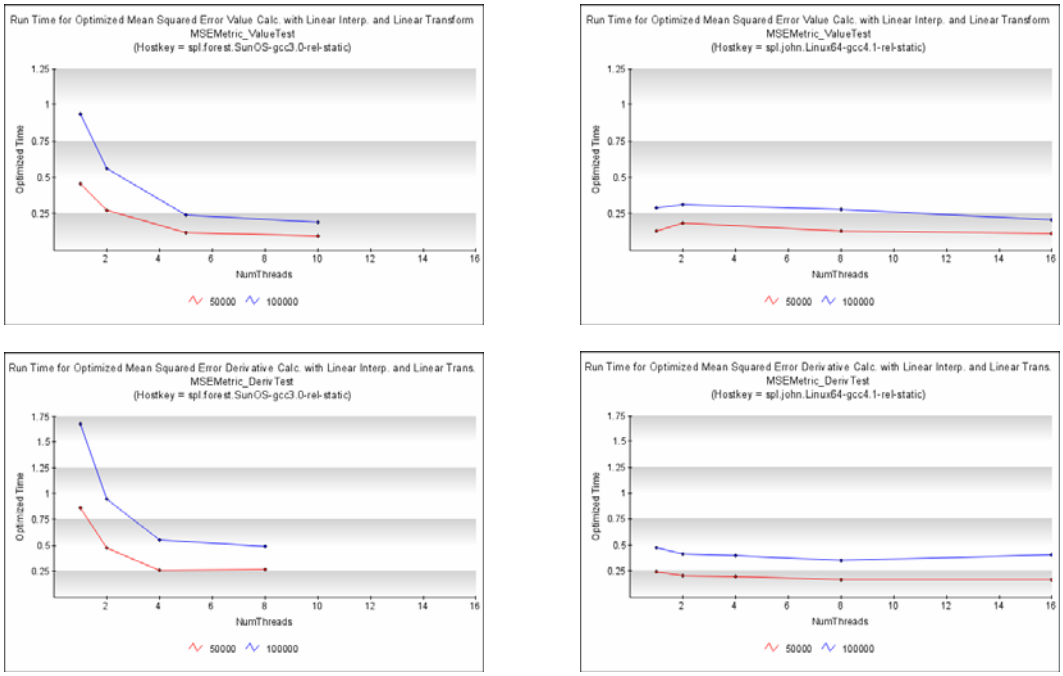


***Figure 6**. Mean Squared Error metric run times for ten calls to the `metric->GetValue()` (top row) and `metric->GetDerivative()` (bottom row) for Forest (left column) and John (right column). Linear interpolation and linear transforms were used with the metric. 50,000 (red line) and 100,000 (blue line) samples were used for the metric computations.*

5 Discussion

The work-in-progress presented in this paper spans a multitude of registration modules in ITK. Only a subset of the results could be presented in this paper. Much additional work remains.

The results indicate that the optimizations employed can decrease the run-time of select existing ITK registration programs by a factor of 6 or more. Run-time improvement ratios will vary significantly if the registration optimizer uses derivatives more heavily than value calls, if the number of threads used isn't optimal, or if the problem size is too small.

The problem sizes used in the tests presented in this paper are small compared to real-world problems and thereby the potential speedup offered by the optimized methods is under estimated. Typically when using 512x512x400 images, more that 100,000 samples should be used to drive the deformable registration optimization process. Problem sizes were kept small in the tests presented in this paper to allow for the nightly testing of multiple parameterizations. Using larger sample sizes will increase the parallelizable component of the method (i.e., the term *B* in Equ 1) – thereby resulting in more speedup for any given number of threads and continued speedup as more threads are used.

Future work will borrow from several of the parallelization techniques presented in [Rohlfing 2003]. In particular, we are extending the metrics and the B-Spline transform to produce sparse Jacobian matrices that exclude B-Spline control points that do not containing meaningful image information (e.g., contain only background voxels). This will reduce the computation time per iteration and should simplify the solution space considered during registration optimization.

6 Acknowledgements

This work is funded in-part by the National Alliance for Medical Image Computing (NAMIC) with the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics.

This work is also funded in-part by a supplemental grant from NAC P41 RR013218-09 for ITK parallelization and grid computing.

Reference

[Amdahl 1967] G. M. Amdahl, “Validity of the single-processor approach to achieving large scale computing capabilities,” in Proc AFIPS Conf., Vol 30, Reston , VA, 1967, pp. 483-485

[Ibanez 2002] L. Ibanez, L. Ng, J. Gee, and S. Aylward, “Registration patterns: the generic framework for image registration of the Insight toolkit.” IEEE International Symposium on Biomedical Imagine, pp. 345-348, 2002

[Matsumoto 1998] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator", ACM Transactions on Modeling and Computer Simulation, Vol. 8, No. 1, January 1998, pp 3-30.

[Mattes 2003] D. Mattes, D. R. Haynor, H. Vesselle, T. Lewellen and W. Eubank, "PET-CT Image Registration in the Chest Using Free-form Deformations." IEEE Transactions in Medical Imaging. Vol.22, No.1, January 2003. pp.120-128.

[Pohl 2007] K.M. Pohl, S. Bouix, M. Nakamura, T. Rohlfing, R.W. McCarley, R. Kikinis, W.E.L. Grimson, M.E. Shenton, and W.M. Wells. A hierarchical algorithm for mr brain image parcellation. IEEE Transactions on Medical Imaging, 2007. In Press.

[Rohlfing 2003] T. Rohlfing, D. B. Russakoff, C. R. Maurer Jr., “Expectation Maximization Strategies for Multi-atlas Multi-label Segmentation.” IPMI 2003, pp 210-221

[Styner 2000] M. Styner, G. Gerig, C. Brechbuehler, and G. Szekely, "Parametric estimate of intensity inhomogeneities applied to MRI," IEEE Transactions in Medical Imaging.; 19(3), 2000, pp. 153-165

[Wells 1996] W. M. Wells, P. Viola, H. Atsumi, S. Nakajima, R. Kikinis, “Multi-modal volume registration by maximization of mutual information,” Medical Image Analysis, 1:53-51, 1996.

[Zhu 1997] C. Zhu, R.H. Byrd, and J. Nocedal, “L-BFGS-B, Fortran routines for large scale bound constrained optimization.” ACM Transactions on Mathematical Software, 23(4):550-560, 1995

GoFigure and The Digital Fish Project: Open tools and open data for an imaging based approach to systems biology

Release 0.91

Alexandre Gouaillard, Titus Brown, Marianne Bronner-Fraser,
Scott E. Fraser, and Sean G. Megason

September 11, 2007

Center of Excellence in Genomic Science and Division of Biology,
California Institute of Technology, Pasadena, CA 91125

Abstract

As part of the Center of Excellence in Genomic Science at Caltech, we have initiated the Digital Fish Project. Our goal is to use *in toto* imaging of developing transgenic zebrafish embryos on a genomic scale to acquire digital, quantitative, cell-based, molecular data suitable for modelling the biological circuits that turn an egg into an embryo. *In toto* imaging uses confocal/2-photon microscopy to capture the entire volume of organs and eventually whole embryos at cellular resolution every few minutes in living specimens thoughout their development. The embryos are labelled such that nuclei are one color and cell membranes another color to allow cells to be segmented and tracked as they move and divide. The use of a transgenic marker in a third color allows a variety of molecular data to be marked. *In toto* imaging generates 4-d image sets (xyzt) which can contain 100,000 to 1,000,000 images per experiment. We are developing a software package called GoFigure to visualize, segment, and analyze these very large image sets. GoFigure uses a MySQL database back end for managing storage of images and segmented objects and uses VTK and ITK for visualization and segmentation. We plan to use *in toto* imaging to digitize the complete expression and subcellular localization patterns of thousands of proteins throughout zebrafish embryogenesis. This genomic data, our zebrafish lines, and GoFigure will be distributed following the Open Data/Open Source model.

Contents

1 Introduction **2**
1.1 Systems Biology, Imaging, and Image Analysis 2
1.2 CEGS and the Digital Fish Project 3

2 *In toto* imaging **3**
2.1 Embryo labeling 4
2.2 Image acquisition 5
2.3 Image processing 5
2.4 Significance 5

3 GoFigure **7**
3.1 Goal . . . 7
3.2 Technologies . . . 7
3.3 Client-server architecture . . . 8
3.4 GUI . . . 8
3.5 Figures, meshes, tracks, and lineages . . . 9
3.6 Image segmentation . . . 9
3.7 Visualization . . . 9
3.8 Plans for future development . . . 13

4 Conclusions **13**

5 Links **14**

1 Introduction

1.1 Systems Biology, Imaging, and Image Analysis

With the completion of the genome projects, the stage is now set for a more complete understanding of how animals are created. The genome projects have given us the complete sequence of DNA that encodes the blueprint of life for many animals including human, mouse, zebrafish, and fruit fly[2][3][4][5]. The challenge is now understanding how this code functions. The problem however is that the only thing that we can currently reliably predict from the genomic code is which parts can be transcribed and translated into protein. We cannot accurately predict when and where a protein will be expressed, we cannot predict how a protein will fold and function once it is translated, and we cannot predict the interaction between expressed proteins that allow them to form functional genetic circuits. Thus, although the genome projects have given us the complete code for constructing many organisms, the only part of the code we can currently read is the parts list. The challenge ahead is understanding how all these parts fit together to form molecular circuits, how these circuits process information to regulate the behavior of cells, and how the behavior of cells is orchestrated to generate functional form as in embryogenesis. This post-genomic endeavor has come to be called systems biology.

Although systems biology grew out of the "-omics" field which made heavy use of in vitro biochemical approaches (e.g. sequencing, microarrays, and proteomics), in vivo imaging is becoming an increasingly powerful tool for systems biology. In vivo imaging is advantageous over biochemical approaches for doing systems biology because of 4 considerations about how biological circuits function[1]. First, biological circuits function at the single cell level. Microscopic imaging easily achieves this resolution while in vitro techniques typically do not. Second, biological circuits function over time. With imaging, different components of a biological circuit can be labeled using different colors of fluorescent proteins and the dynamics of the circuit monitored non-invasively with time-lapse fluorescent imaging as the circuit functions in an intact system. In vitro biochemical approaches typically require the tissue to be destroyed in order to be assayed which precludes longitudinal analysis. Thirdly, the quantitative amounts of components in a biological circuit are important for its function. Fluorescent imaging can accurately quantitate the levels of molecular components even at the protein level through the use of fluorescent protein (e.g. GFP) fusions. Omic approaches are typically less quantitative and focus at the DNA or RNA level which is less relevant.

And finally, the anatomical context of biological circuits is essential for determining their function; the use of spatial cues to generate different cell types in different places is a fundamental aspect of development. In vivo imaging can capture data from intact animals preserving its anatomical context. In vitro approaches typically grind up the tissue to assay it, thus destroying its anatomy.

As you will see reading this paper, the use of imaging for systems biology opens up a number challenges in image analysis. Some of these challenges such as visualization of large image sets have close parallels in other areas such as medical imaging. Other challenges such as segmenting closely packed cells in space, across time, and across cell division can benefit from techniques borrowed from other areas but have unique differences requiring novel approaches. We believe that imaging based systems biology provides a novel growth opportunity for the image analysis field, and likewise that the sucessful development of image analysis tools is absolutely essential for the progress of systems biology

1.2 CEGS and the Digital Fish Project

We recently received a grant from the National Human Genome Research Institute to establish a Center of Excellence in Genomic Science (CEGS) at Caltech. The CEGS program was established around 5 years ago to support high-risk/high-reward research with the potential for making a substantial impact on genomics. Around 1 CEGS has been funded each year since its inception. The focus of research of each CEGS is investigator led but taken together they span the range of genomics, nanotechnology, and systems biology. Our CEGS is more focussed on imaging, systems biology, and embryogenesis relative to the others and is titled "*In toto* genomic analysis of vertebrate development". Our CEGS has 4 specific aims: 1) To develop *in toto* imaging to allow us to image and track every cell in developing embryos and digitize fluorescently marked data at the level of the cell; 2) develop our FlipTrap technology to allow us to generate transgenic animals that fluorescently mark a range of biological data including protein expression and subcellular localization as well as mutant phenotype; 3) develop our Hybridization Chain Reaction (HCR) technology to detect a wide range of biological substrates including RNA, protein, and metabolites; and 4) to integrate and scale-up the above aims to create a Digital Fish by using in toto imaging to digitize expression patterns and mutant phenotypes at cellular reolution in living embryos on a genomic scale and use this data to construct computer models of developmental processes.

Zebrafish is the main model system we are employing with a lesser focus on quail. Zebrafish (*Danio rerio*) is a small, freshwater, tropical fish commonly available in pet stores. It has become a popular model system for the study of genetics and development in the last 2 decades and there are now many labs worldwide that study it. There have been several large-scale mutagenesis screens caried out in zebrafish and its genome has been largely sequenced. Zebrafish has the advantages of fruit fly in that it is amenable to forward genetic screens, but unlike fruit fly, zebrafish is a vertebrate so is much more relevent to humans. Another huge advantage of zebrafish is its suitability for imaging. Zebrafish embryos are transparent, small, develop freely outside their mother, and develop directly from egg to adult without any larval stages. This means that a zebrafish egg can be placed under a microscope and continuously imaged throughout embryogenesis. This would be impossible with a mouse which develop inside its mother, with a frog (Xenopus) egg which is opaque, or with a fruit fly (Drosophila) which has several larval/pupal stages and is opaque.

2 *In toto* imaging

The goal of *in toto* imaging is to image and track every single cell in a developing tissue, organ, or eventually whole embryo[6]. There are several steps to *in toto* imaging. The embryos must first be labeled with

"segmentation markers" which allow all the cells to be segmented. Additionally embryos can be labelled with additional markers to reveal RNA or protein expression. The next step is to acquire xyzt image sets using confocal or 2-photon fluorescent microscopy. And the final step is processing the often very large image sets to segment out all the cell trajectories and to extract quantitative, cell-centric data. This final image analysis step is the main focus of this paper and the goal of GoFigure.

2.1 Embryo labeling

Since we use fluorescence microscopy, the embryos must be labeled with fluorescent markers and since we are doing time-lapse imaging of living embryos, we must used vital labels. Fluorescent proteins such as GFP (Green Fluorescent Protein) are the perfect choice. Fluorescent proteins (FPs) are proteins cloned from a number of marine invertebrates including jellyfish and coral that have the unique property of generating a signal (fluorescence) without the addition of any exogenous substrate. The proteins can be expressed in a wide variety of animals and will fluoresce light after being excited with light of a shorter wavelength. There are now FPs available across the visible spectra including blue, cyan, green, yellow, orange, red, and far red. Since FPs are proteins rather than organically synthesized small molecule dyes, FPs can be endogenously expressed by transgenic organisms and are open to all the power of genetic engineering.

There are 2 basic uses of markers for *in toto* imaging: segmentation and expression. Segmentation markers allow the cells to be segmented in space, across time, and across cell-division. We typically use a histone-FP of one color (e.g. histone-cerulean) and a membrane-localized FP of another color (e.g. membrane-mCherry). This combination would generate embryos in which every nuclei is cyan and all the cell membranes are red. Histone-FP is a fusion protein between an FP and histone2B one of the core proteins that DNA is wound around to form chromatin. This marker is nice because it not only marks the nuclei during interphase but also during mitosis. Many other nuclear markers will become diffuse during cell divsion, but since the histone-FP is stuck to the DNA, it marks the condensed chromosomes of both daughter nuclei thoughout mitosis allowing the mother cell to be linked to its 2 daughters.

We key on nuclei rather than whole cells for the initial segmentation because nuclei have a simpler and more uniform shape than whole cells. Nuclei generally have some kind of ovoid shape (ball, coin, sausage) whereas whole cells can have a wide diversity of shape including spheres, columns, spindles, stars, or in the extreme case of neurons highly branched trees. The histone-FP marker alone can be used for nuclear based segmentation. This can work well on cultured cells, but in tissue of intact animals the nuclei are often very close together and appear to be touching in our fluorescent micrographs. For this reason, we use a second color to label the cell membranes. By subtracting the membrane channel from the nuclear channel, neighboring nuclei can be made more distinct and easier to segment. The membrane label has the aditional advantage that it defines the extent and shape of the whole cell. Cell morphology can be very useful for determining cell type and tissue type in fluorescent micrographs. The membrane marker also provides the opportunity to segment out the whole cell. For exampe, once all the nuclei have been segmented, regions can be grown outward until they hit the cell membrane marker to segment whole cells. If this dual segmentation is done for all cells, then it can be used to define the basic compartments of a cell: cell membrane, cytoplasm, and nucleus. This informations is very useful for cellular approaches because it defines the cellular geometry as well as molecular approaches since it can be used to define the subcellular localization of proteins.

In addition to segmentation markers, the other use of markers for *in toto* imaging is expression markers. This is a bit of a catch all term but refers to whatever else is being marked such that it can be digitized with *in toto* imaging. For example, transgenic fish can be made that express a fluorescent protein wherever some gene is normally expressed during development. Once all the cells have been segmented using the segmentation markers, the amount of the expression marker in each cell can then be quantitated at cellular resolution.

Our FlipTrapping approach is particularly useful in this regard because it generates fusions of endogenously produced proteins to YFP (yellow fluorescent protein). Since the YFP fusion protein is expressed from the same genetic locus as the endogenous protein, it should be expressed at the same levels and in the same tissues as the endogenous protein, and thus in toto imaging can be used to quantitate protein expression on a cell by cell level. FlipTrap fusion proteins also reveal the subcellular localization of the trapped protein.

2.2 Image acquisition

In toto imaging tracks cells during development only by the correspondence of cells from frame to frame. This is quite different than traditional fate mapping in developmental biology where only a small subset of cells is labeled at one stage of development and the location of their progeny is observed at a later stage. In toto imaging thus requires images that have high enough resolution in space to resolve all the cells (we typically aim for 1um resolution) and high enough resolution in time to not lose track of moving and dividing cells (this requires around 2 minute time resolution). In addition to high resolution, in toto imaging also requires complete coverage. All the cells of interest must be continuously imaged in order to be tracked; if cells leave the field of view then their lineage cannot be completely tracked.

Achieving both high spatial and temporal resolution as well as complete coverage can be achieved through the use of time-lapse confocal or 2-photon microscopy. Confocal microscopy uses a pinhole to eliminate out of focus light whereas 2-photon microscopy only excites fluorescence in the focal plane. Thus both techniques allow thin (˜1 um) optical sections of fluorescent samples to be captured. By imaging at a series of focal planes, stacks of optical sections can be captured to generate a volumetric image and this can be repeated over time to generate xyzt images. The axial (z) resolution with these techniques is typically around 2-fold less than the planar (xy) resolution so the volumetric images are anisotropic. Confocal imaging provides 2-fold higer resolution than 2-photon imaging but has a depth penetration limit of ˜150 um. 2-photon imaging can image all the way through a zebrafish embryo and has reduced phototoxicity to the embryo and photobleaching of the fluorescent labels (see image 1).

For in toto imaging, we mount zebrafish embryos in custom developed chambers that hold the embryos stationary for imaging while allowing them to grow and develop normally. We mount the embryos shortly after fertilization and can continuosly image them throughout embryogenesis (3 days). The use of a motorized stage permits tiling xyzt images across a single embryo and imaging multiple embryos.

2.3 Image processing

After the embryos have been labeled and imaged, the final challenge of in toto imaging is image processing. The goal of image processing is to track all the cell movements and divisions to generate cell lineage trees, to define the boundaries of all cells and their compartments (nucleus, cytoplasm, membrane, extracellular space), and to quantitate the level of fluorescence within each cell and subcellular compartment. We are developing a software package called GoFigure for handling all of the image processing needs of in toto imaging so this step will be further elaborated in the next section.

2.4 Significance

There are a number of significant outcomes that we hope to acheive with the Digital Fish Project using in toto imaging. The first is to obtain a complete understanding of the cellular basis of development through the construction of complete lineage trees. We plan to use in toto imaging to image and digitize all the cell

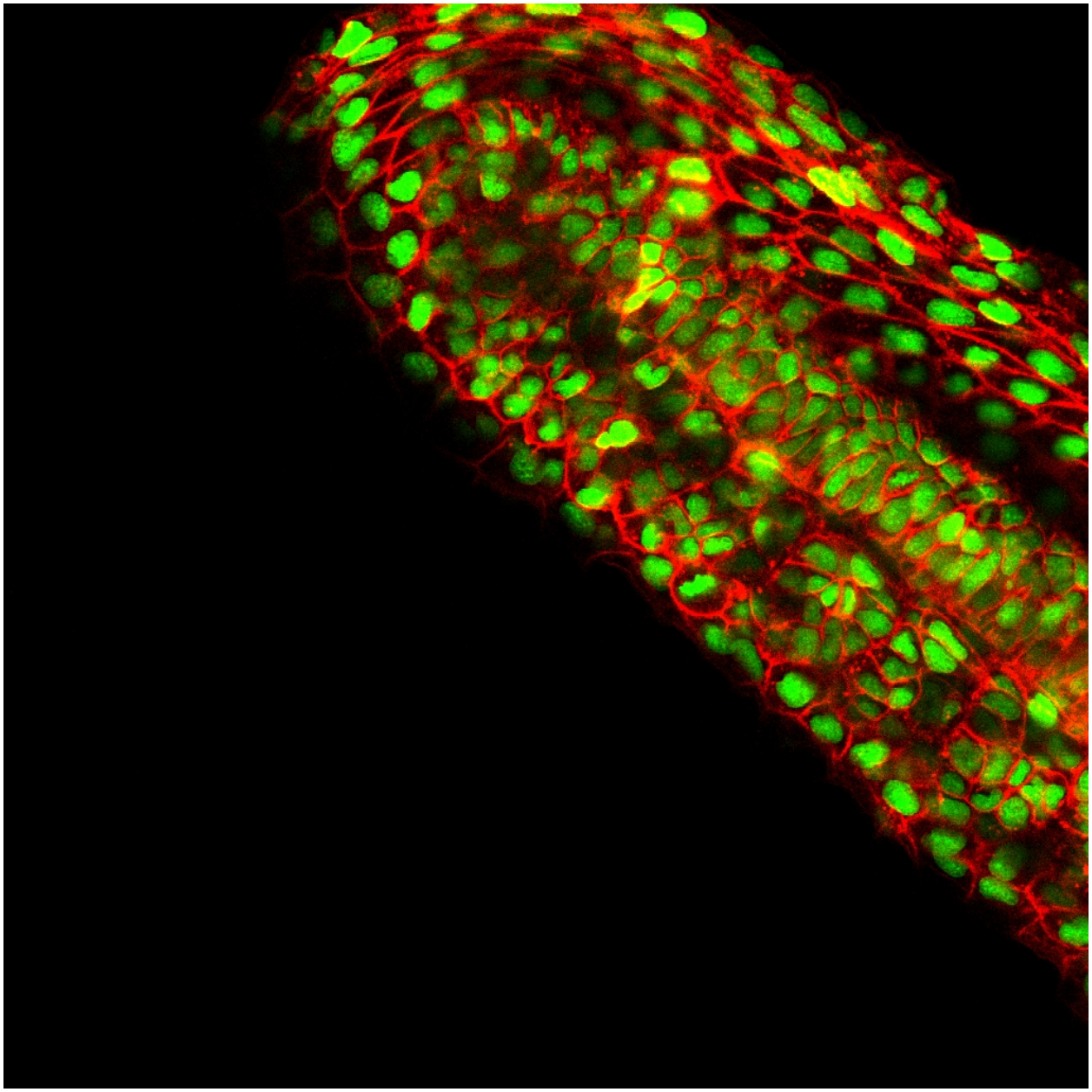


Figure 1: This represents the tailbud of the zebrafish embryo. The image has been taken using a 2-poton confocal microscope.

movements, cell divisions, cell deaths, and cell shape changes that are used to construct organs such as the inner ear, eye, and spinal cord and we hope to eventually apply this approach to the whole embryo beginning with the fertilized egg. This type of approach will give us a complete understanding of morphogenesis at a cellular/morphological level (but not molecular).

A second use of in toto imaging is to digitize molecular data for use in systems biology. Transgenic fish such as FlipTraps can be in toto imaged and the segmentation masks used to quantitatively define the fluorescence in each cell and subcellular compartment over time. Since FlipTraps generate fluorescent fusion proteins, the levels of fluorescence directly correlate with the number of molecules of the tagged protein. Thus with the proper controls and corrections, in toto imaging can be used to get accurate measurements of the number of molecules of a protein in all cells during a developmental process. Such data is extremely useful for doing systems biology. For example, by using a few colors to mark the critical nodes of a biological circuit, it is possible to monitor the function of that circuit dynamically in live embryos[1]. In toto imaging can also be used on a large scale in a high-throughput fashion to digitize molecular data. We hope to use such an approach to capture near single cell resolution, 4-d representations of protein expression data using our FlipTraps. This data will be integrated to form the Digital Fish.

3 GoFigure

3.1 Goal

GoFigure is a software application we are developing to serve as the principle image analysis tool for in toto imaging. The goal of GoFigure is to digitize image based data at the cellular level. Confocal/2-photon microscopy generates image sets comprised of pixels but it must be translated into cells for data analysis for biologists. The cell is the basic building block of embryos as well as a fundamental unit of computation in biological circuits since most molecular components cannot pass the cell membrane. While humans have an easy time spotting cells in microscopic images, computers have a notoriously difficult time. One of the most significant challenges of GoFigure is cell segmentation. The datasets can be several terabytes in size which pauses computational issues. It is 3D+t datasets, whose visualization can be challenging. It can also be very noisy and/or presents (interlacing) acquisition artifacts. Once the cells are segmented in space, time, and once cell divisions have been taken into account, , it is straightforward to visualize the data and quantify the data. We intend GoFigure to be an integrated interface for segmentation, visualization, and cell-based analysis. It should be an user-friendly tool for biologists to use in their research. Although GoFigure is being created to serve the needs of in toto imaging and the Digital Fish Project, we hope that this software will also be used by other scientists doing imaging-based systems biology.

3.2 Technologies

The curent version is a windows-only research prototype. Coded entirely in C++, it uses high-level OO design, design patterns and other recent component oriented theories to assemble the 100+ classes into a large, but flexible application.

it is based on Microsoft Foundation Classes (MFC) for the Graphical User Interface (GUI) and database accesses. It is using a MySQL database for segmentation results and cell information storage. Links to datasets are stored in the database, but the datasets are actually stored outside of the database. VTK is used for data visualization.

3.3 Client-server architecture

GoFigure uses a client server architecture. The server hosts a MySql database which stores all of the segmentation results and cell analysis. Links to the images are stored, but the image themselves are kept on a different location. The client is the GoFigure software itself. GoFigure interacts with the (possibly remote) server during image analysis. Typically, GoFigure will load data from the server, allow analysis, processing and visualization of the data, and then store any new data in the database.

the client server architecture has several advantages. Most importantly, the database server allows very large data sets to be worked on. The MySql database server can manage data sets many terabytes in size. It also stores the data as it is created in the possibility the client crashes. The client-server architecture also allows several people to interact with the same server and dataset using different clients.

The server and the client can reside on different computers or can be located on one computer. For most applications, all of the components required for GoFigure can simply be installed on one computer which can serve as both client and server.

3.4 GUI

The GUI is based on MFC classes that give GoFigure the qualities of a professional applications. Based on the MDI design, it is composed of a main window that allows standard processes (loading new files, quiting the application, ...) while no child window is open. When opening a new document (i.e. a dataset stored in a database) a child window will appear, enabling menu bars and options depending on which data are available and / or visible.

An important concept in GoFigure are the “views”. Views are subwindows within the MDI child window that are dock-able and tab adjustable. GoFigure has quite a few of these available: XYZ or XYT composite viewers, slice informations viewer, dataset slice / file viewer and database table viewer. The end user can define the layout of the views according to his own preference or the experiment he is working on. GoFigure stores the curent layout in the registry and thus, “remembers” the layout across sessions.

As most people working on real cases know, there is no such thing as an overall general segmentation algorithm. Whatever algorithm we choose, it will never give perfect results. It is thus important to add user interaction to the segmentation aspect of GoFigure. It will not only impact the GUI layer, but also the segmentation layer. We will only adress the GUI layer here.

1. GoFigure allows you to interactively draw a Region Of Interest (ROI) that will be used as a constraint by the segmentation algorithms. If you define the ROI to be an anatomical feature, that can lead to tracing the partial lineage for that specific features (the ear, for example)

2. It is possible to run the algorithm on a single cell instead of running it on an entire image or on all a ROI. In that case a slightly different algorithm is used. It allows one to complete the results by manually targeting the cells that the global algorithm would have missed (false negatives). Aiming for the lineage, we need to insure that all cells are extracted.

3. GoFigure also allows you, after segmentation, to individually pick any result (any cell) and remove it, thus dealing wth the false positives. You can either pick the results in the Table viewer ,which in turn will colored the representation of the corresponding cell on the XYZ / XYT viewer, or vice-versa, as all the views are linked.

3.5 Figures, meshes, tracks, and lineages

GoFigure defines and handle four kinds of objects: figures, meshes, tacks and lineages.

Figures are the 2D contours of the nucleus of a cell within a 2D XY slice of a dataset, at a given depth (Z) and time (T).

Meshes are the 3D surfaces of the nucleus of a cell within a 3D XY-Z stack of 2D XY slices for all depths (Z) at a given time. T. It can also be seen as a collection of figures, and in GoFigure, all the figures have a link to their corresponding mesh. Depending on the segmentation algorithm, we can start from either figures or meshes.

Tracks are the continuation of meshes in time. You can display it as a "tube" representing all the positions of a given mesh during a certain amount of time. Accordingly, all meshes in GoFigure have a link to their corresponding track.

Lineages are the goal of the segmentation process. It is composed of the addition of all tracks, plus their branching (where cells subdivide). Again, each track in GoFigure has a link to its lineage. Note that lineages are, by definition binary trees.

3.6 Image segmentation

Currently, we use a segmentation algorithm to obtain the figures on a dataset, and we then group them into meshes, tracks, and lineage using a propagation algorithm that searches the local neighborhood (with respect to the dimension of the space) for the next object. For example, once the figures are available, we search within the Z stack for other figures belonging to the same mesh. Similarly, given the meshes, we look across time for its next and previous positions.

The current segmentation algorithm for figures is quite accurate (85 percents or more of the cells are found) and has the advantage of directly extracting all the biological values of interest, but unfortunately is too slow (more than a minute per 2D image) to accommodate with the size of the dataset we are expecting. We are experimenting with other algorithms, possibly working directly at the 3D, mesh, level.

The sampling in time is good enough for simple neighborhood search algorithm to be used. This supposes that the figure segmentation is complete and accurate to begin with, since errors are cumulative. We are working on tracking algortihms, possible guided by a motion estimation of the cells, that can enhance the results of a previous segmentation. The mesh segmentation algorithm could for example expect the cell to exist across several slices, and if a figure is missing between two existing related, figures, the algorithm can fill in the gap.

3.7 Visualization

All the visualization is based on VTK. We, at most, visualize a Z-stack and a T-stack for a given common XY slice at a time. We do not really suffer, as long as visualization is concerned, from the size of the dataset. Moreover, the depth and resolution in Z are usually limited in confocal imaging, thus the amount of memory needed for vizualization depends only on the time range and sampling.

The two main data visualization "views", respectively XYZ and XYT viewers, are made of 4 render windows which are resizable. One window is used for visualizing the curent XY slice, two side windows are used for the projection on Z or T, and the last windows is used for a 3D volumetric representation of the curent Z/T stack. All 4 windows are linked, and any modification affects them all.

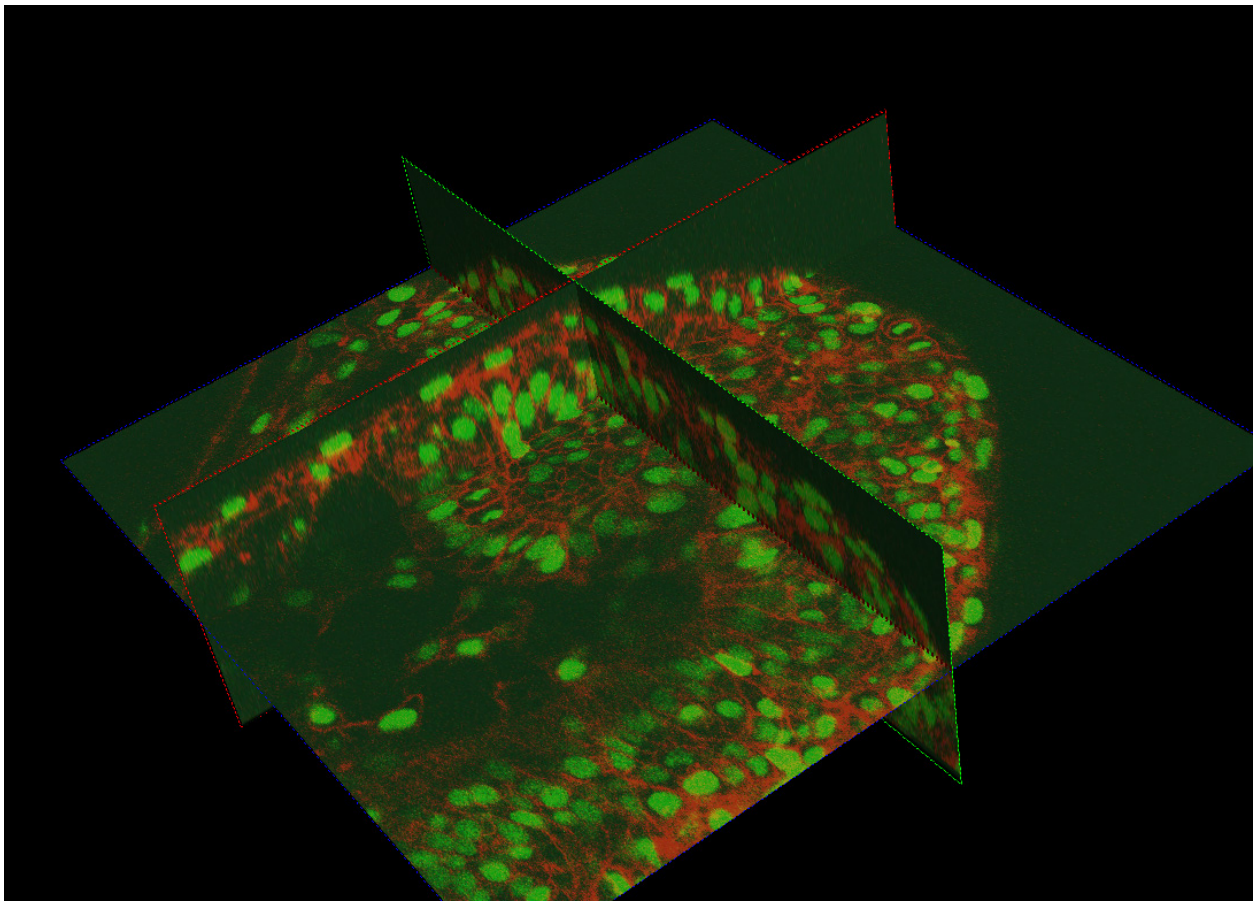


Figure 2: One of the four rendering wndows composing our XYZ view. Each plane is linked with the three other representations (see **??**). If the user modify the position of one plane, the four renderer are updated.

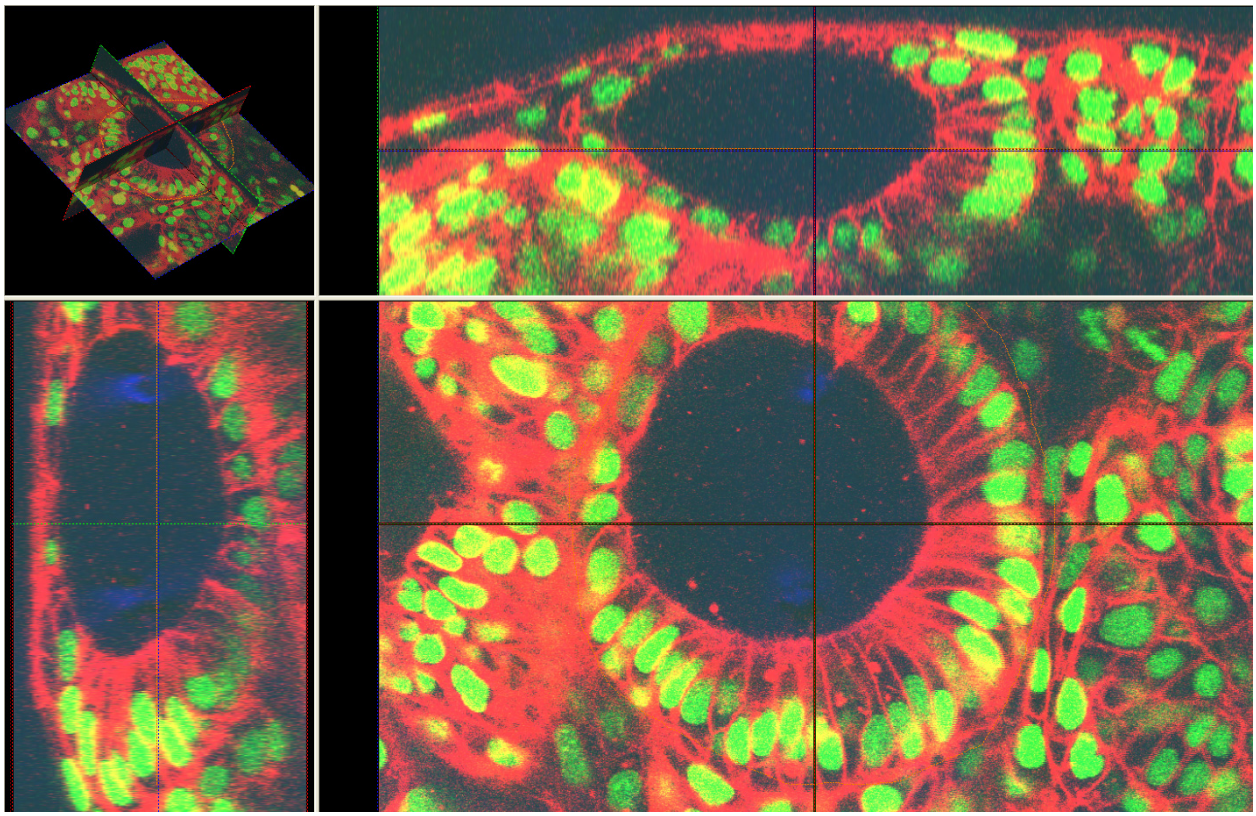


Figure 3: Our XYZViewer.

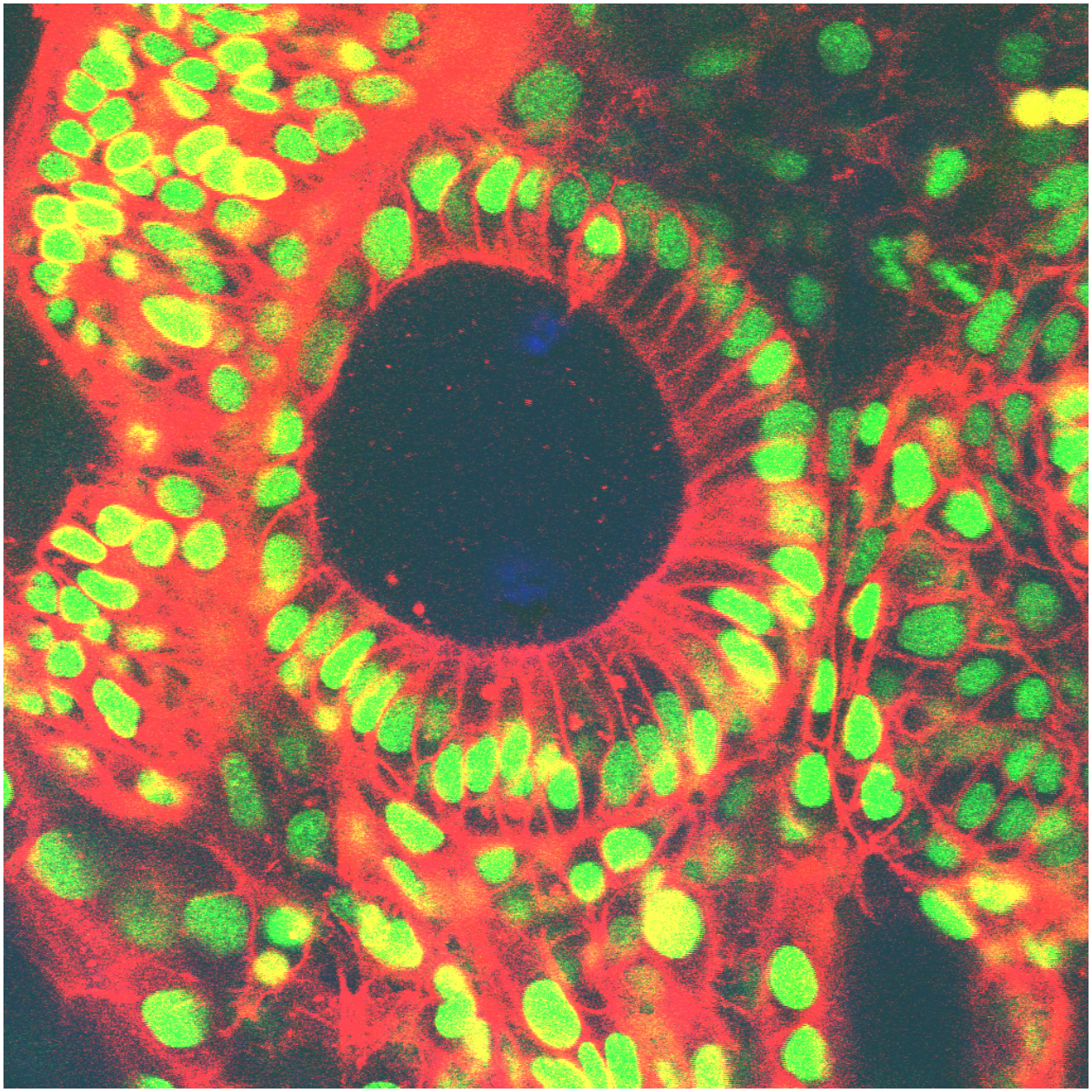


Figure 4: A high resolution zoom on the ear area of the zebrafish embryo (unfortunatly, the image had to be scaled down for the paper).

The result of the segmentations are added to the views as discrete meshes whose dimension depends on the type of object. We use the vtkPolyData structure for that.

3.8 Plans for future development

This paper is more a work in progress report than a technical report. We are willing to announce to the community that an open-source, open-data project is starting so that interested individuals or research group cam follow or even join the effort. As the software is based on NAMIC toolkits we thought that the Insight Journal, and MICCAI Open-source workshop was the best place to make such an announcement.

There is still a lot of work on the segmentation part and we are looking forward having motivated individuals join the development effort to work on that specific subject. As the project is Open Source, we are also hoping that other researchers from other institution will join the effort and help us develop these tools.

To ease the transfer, configuration and installation of GoFigure, we are planning to move from a windows-only code to a cross-platform code. Right now, we have already made one step toward it by moving the configuration process to CMake, one of the NA-MIC tool. CMake can handle cross-plateform configuration. CMake also comes with CPack, that we are using right now to create packages for GoFigure. We have set up a dashboard (Using Dart 2) to continuously build and monitor our code on different flavors of windows, using different versions and/or parameters of MSVC++. As for the compilation, to be truly cross-plateform, we need to replace MFC classes.

We already use VTK for visualization, CMake for configuration, and CPack for packaging. All those are part of the NA-MIC Toolkits. We are planing to replace the MFC-based GUI we have right now by a KWStyle GUI, and we are looking within the other toolkits for a replacement for the MFC-based database access management.

We are planning to improve the quality of our code by developping a test suite and benchmarks, and by monitoring the code coverage, the memory leaks. All those things will be directly available to us once we will have cross platform code with a test suite thanks to hte NA-MIC Tollkits integration (CMake / CTest / Dart 2 / gcov / valgrind).

The images we are acquiring are tiled, and so are our segmentation algorithms. We are planning to developp multithreaded versions of our algorithms to take advantage of the multi-core processors or to be run on clusters. We are planning to develop command line version of each feature of our code to be able to run them in batch mode.

4 Conclusions

As part of the CEGS based Digital Fish Project we intend to scale up our FlipTrap screen to generate ¿1000 transgenic lines of fish and to use in toto imaging to to digitize their 4d protein expression patterns. We will register these 4d data sets onto a common framework to form the Digital Fish, a vast anatomically-based warehouse of molecular data. This data will be made publically accesible via the web as it becomes available in accordance witht the Bermuda Principles which have been successful for genome sequence data. We also intend the Digital Fish to be more than just a data warehouse, but to form a framework for constructing cell-based computer models of developmental processes such that we can move towards the ability to actually compute phenotype from the code of genotype.

5 Links

Both Data and code are available under Creative Commons by attribution licence.

Data are available at the following link. More dataset will be added later.
http://www.insight-journal.org/midas/view_collection.php?collectionid=37
http://www.insight-journal.org/dspace/handle/123456789/580

The Research Prototype is available here:
http://gofigure.caltech.edu/beta/

The current work is documented here:
http://iorich.caltech.edu/gofigure/

The current installation process is documented here:
http://iorich.caltech.edu/gofigure/wiki/Installation

Access to the svn repository is documented here:
http://iorich.caltech.edu/gofigure/wiki/CodeVersioning

The doxygen-generated documentation of the source code is available here:
http://iorich.caltech.edu/gofigure-doc/doxygen/html/classes.html

The dart2 dashboard is visible here:
http://cegs-dart-server.caltech.edu/GOFIGURECore/Dashboard/

References

[1] Megason SG and Fraser SE. (2007). "Watching biological circuits function: The role of imaging in systems biology", Cell, in press 1.1, 2.4

[2] Venter JC et al. (2001). The sequence of the human genome. Science. 291, 1304-51 . 1.1

[3] Lander ES et al. (2001). Initial sequencing and analysis of the human genome. Nature 409, 860-921. 1.1

[4] Waterston RH et al. (2002). Initial sequencing and comparative analysis of the mouse genome. Nature 420, 520-62. 1.1

[5] Jaillon O et al. (2004). Genome duplication in the teleost fish Tetraodon nigroviridis reveals the early vertebrate proto-karyotype. Nature. 431, 946-57. 1.1

[6] Megason SG, Fraser SE. (2003). Digitizing life at the level of the cell: high-performance laser-scanning microscopy and image analysis for in toto imaging of development. Mech Dev.120:1407-20.

2

Data, data everywhere, nor an image to read - Finding open image databases.

Release 0.00

David R. Holmes, III and Richard A. Robb

July 15, 2007

Biomedical Imaging Resource, Mayo Clinic College of Medicine, Rochester, MN

Abstract

Open Science includes access to both open source software/methodologies and open data. While there has been progress in open image databases, the results of these efforts are under-reported. As such, imaging scientists are unaware of the available data. In addition, many researchers are interested in providing their data to the greater research community, but may be unaware of the process to release the data. The purpose of this paper is to describe our efforts in developing an open website which includes information on accessible medical image databases as well as some of the logistics for providing an open image database. Most importantly, the authors are requesting participation from the community to contribute to the Medical Image Database Repository (http://midr.org) in order to consolidate the collect knowledge of the community.

Contents

1 Implementation **2**

2 Medical Image Database Repository (http://www.midr.org) **2**

3 The Database Wiki **2**

4 The Logistics Wiki **4**

5 Discussion **4**

At the MICCIA 2005 Open Source Workshop, the keynote speaker, Dr. Michael Vannier, challenged the open source community to develop new image processing methodologies which are clinically translatable. During his talk, Dr.Vannier observed that there are massive amounts of medical image data but the research community is not utilizing it. This led the authors to consider a quote from “The Rime of the Ancient Mariner” – *Water, water everywhere, nor a drop to drink.* In the context of Dr Vannier’s comments, we would say ”Data, data everywhere, nor an image to read.” The primary challenge is that most acquired image data is clinical and often unavailable to the imaging scientist. In addition, the data that is available

through open databases is often under-promoted and overlooked by researchers. The purpose of this paper is to describe the development of an open website where the image analysis community can contribute and consolidate knowledge on open image databases as well as approaches to providing open data in the future.

A recent article in Biomedical Computation Review [4] provides a nice overview of some of the issues relating to image databases. Appropriately, the article begins with a description of the Visible Human Project [2]. The VHP has been a very successful image collection with wide-spread acceptance throughout the field. In additional, it is spawned the development of several other VHP-like image collections[12, 7]. The VHP project has been successful for many reasons; however, we suggest that the particular strength of the VHP is that the project was specifically targeted as an open image database with clear specifications for the outcome. There have been several other image databases [6, 8, 10] which have shown similar success. In most cases, the intended use of the databases is unique, therefore providing an important and novel contribution to the community. These examples have all been well-accepted and utilized by the research community. In order to best inform the imaging community, we suggest that the consolidation of the communities' knowledge of medical image databases will help to further disseminate both existing and future image databases.

1 Implementation

Our approach to consolidating and disseminating information on medical image databases is to develop a wiki-based website which will store meta-information about existing medical image databases. The wiki-based approach was chosen over other, more traditional, approaches such as a conventional website or newsgroup, because it provides structured access to all of the data while maintaining an open process to add/edit content by the community.

2 Medical Image Database Repository (http://www.midr.org)

The website has been built with conventional web development tools. The front matter is standard html with links into the wiki(See Figure 1). MediaWiki [5] , a popular wiki engine, was used in the development of the wiki pages. The wiki content is split into two major categories - "databases" and "logistics." The Database Wiki is organized with a page per database. Each database page includes database contact and content information. The Logistic Wiki is a more general wiki will included other information related to image databases such as HIPPA requirements [9], suggestions for text on open databases to be included in IRB or Animal Care Protocols, and links to relevant software.

3 The Database Wiki

The intent of the Database Wiki is to provide basic meta-information about various open image databases available online; at this time, neither the wiki nor the website will provide the data from an image database. In order to provide a common look-and-feel across the database descriptions, a template has been built which contains all of the proposed content headers. The content is divided into four major categories - introduction and text description of the database, contact information, database details, and reference material. A screenshot of a wiki page is shown in Figure 2. Details on the categorical data are shown in Table 1.

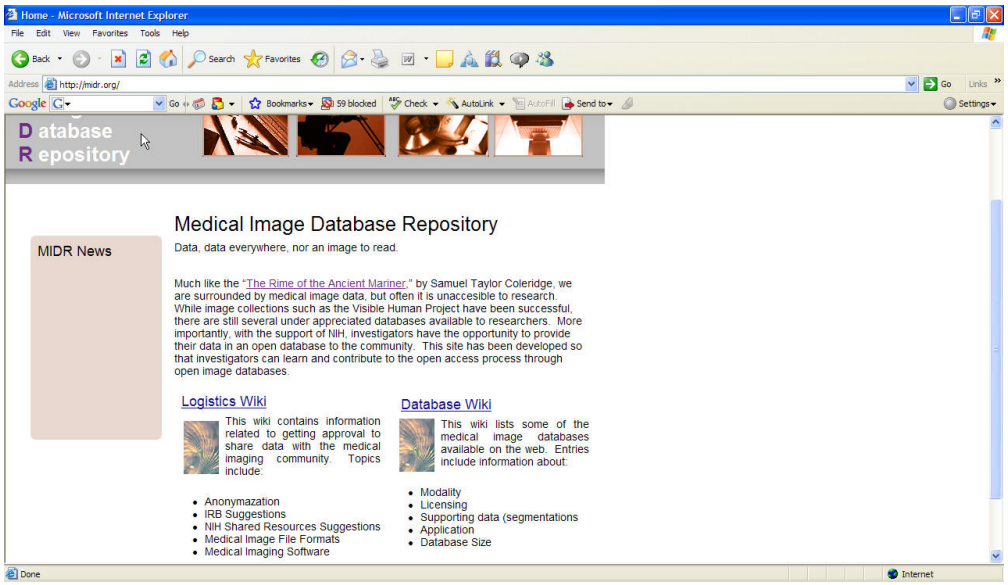


Figure 1: Snapshot from the Medical Image Database Repository website (http://midr.org)

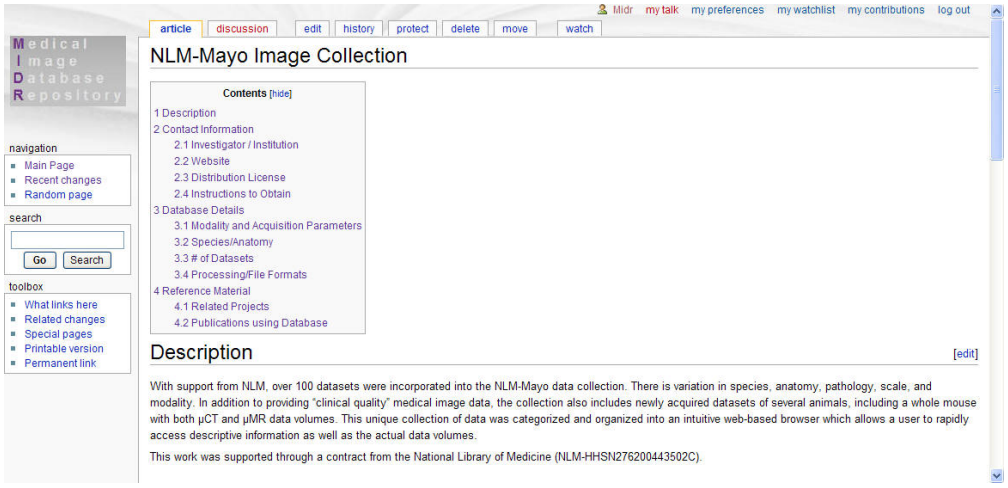


Figure 2: Snapshot from the Database Wiki using [3] as an example.

Table 1: Proposed content for image database entries.

Abstract
Contact Information
Investigator / Institution
Website
Distribution License
Instructions to Obtain
Database Details
Modality and Acquisition Parameters
Species/Anatomy
# of Datasets
Processing / File Formats
Reference Material
Related Projects
Publications using Database

4 The Logistics Wiki

The Logistics Wiki serves as the primary repository of information relating to the creation, management, distribution, and processing of medical image databases. Articles on the regulatory aspects of image data dissemination will be particularly important for researchers interested in disseminating data. It is the hope of the authors that boilerplate text can be developed for insertion into IRB protocols based on examples of previously successful IRB protocols. A small section of the wiki will be devoted to medical image software for anonymization, file format conversion, and image processing; however, it is understood that there are already successful examples of wiki-style software lists on the web [1].

5 Discussion

All too often, research is hindered by inadequate tools and resources. Open science, if embraced, promises to enable the research community through the sharing of knowledge, tools, and resources. There have been very successful examples of both open source software [11] and open data collections [2]; however, there are many other resources available to researcher in the community. While search engines such as Google, provide a cursory approach to find out about open image databases, the time required to parse through false hits or unmaintained websites is substantial.

We are hopeful that a community-maintained resource, such as midr.org, will provide a common repository of information on open medical image databases. The continuing challenge, however, will be to active engage members of the research community to effectively contribute. While the authors will be actively updating and supporting this website, it is essential that other provide content as well to ensure broad recognition and coverage of available databases. Unmaintained or incomplete data will hinder future efforts.

References

[1] A Crabb. idoimaging.com: Free medical imaging software. http://idoimaging.com. 4

[2] MJ Ackerman. Accessing the visible human project. *D-lib Magazine*, Oct, 1995. (document), 5

[3] DR Holmes III, EL Workman, and RA Robb. The nlm-mayo image collection: Common access to uncommon data. *ISC/NA-MIC/MICCAI Workshop on Open Source Software*, 2005. 2

[4] MA Kunz. Imaging collections: How they're stacking up. *Biomedical Computation Reivew*, Summer, 2007. (document)

[5] MediaWiki. Mediawiki. http://mediawiki.org. 2

[6] Montreal Neuologic Institute. Brainweb. http://www.bic.mni.mcgill.ca/brainweb/. (document)

[7] JS Park, MS Chung, and et al. Visible korean human: Improved serially sectioned images of the entire body. *IEEE Trans Med Imaging*, 24(3):352–360, 2005. (document)

[8] UC Davis. The brain atlas project. http://nir.cs.ucdavis.edu/. (document)

[9] US Department of Health and Human Services. Medical privacy - national standards to protect the privacy of personal health information. http://www.hhs.gov/ocr/hippa. 2

[10] J West, JM Fitzpatrick, MY Wang, and et al. Comparison and evaluation of retrospective intermodality brain image registration techniques. *Journal of Computer Assisted Tomography*, 21(4):554–566, 1997. (document)

[11] TS Yoo, MJ Ackerman, WE Lorensen, W Schroeder, V Chalana, S Aylward, D Metaxes, and R Whitaker. Engineering and algorithm design for an image processing api: A technical report on itk - the insight toolkit. *In Proc. of Medicine Meets Virtual Reality*, pages 586–592, 2002. 5

[12] SX Zhang, PA Heng, and et al. Creation of the chinese visible human data set. *The Anatomical Record*, 275B:190–195, 2003. (document)

Workshop on Open-Source and Open-Data for MICCAI

Brisbane, Australia
November 2, 2007

Schedule

9:00 – 9:20 Introduction

9:20 – 9:40 Efficient Implementation of Kernel Filtering
Beare R; Lehmann G
Rank Filtering, Efficient Methods

9:40 – 10:00 Radial Thickness Calculation and Visualization for Volumetric Layers
Wang D; Shi L; Heng P
Volumetric Layers, Radial Thickness

10:00 – 10:20 Multidimensional Arrays and the nArray Package
Sadowsky O; Li D; Deguet A; Kazanzides P
Vectors and Matrices, Multidimensional Arrays

10:20 – 11:20 Posters and Break

11:20 – 11:40 Diffeomorphic Demons Using ITK's Finite Difference Solver Hierarchy
Vercauteren T; Pennec X; Perchant A; Ayache N
Diffeomorphisms, Demons

11:40 – 12:00 An Open, Clinically-Validated Database of 3D+t cine-MR Images of the Left Ventricle with Associated Manual and Automated Segmentation
Najman L; Cousty J; Couprie M; Talbot H; Clément-Guinaudeau S; Goissen T
Image Processing, Myocardial Infarction

12:00 – 12:20 vtkINRIA3D: A VTK Extension for Spatiotemporal Data Synchronization, Visualization and Management
Toussaint N; Sermesant M; Fillard P
Spatiotemporal, Synchronization

12:20 – 12:40 A Framework for Comparison and Evaluation of Nonlinear Intra-Subject Image Registration Algorithms
Urschler M; Kluckner S; Bischof H
Evaluation, Nonlinear Registration

12:40 – 2:00 Lunch

2:00 – 3:00 Invited Speaker

3:00 – 3:20 A White Matter Stochastic Tractography System

Ngo T; Westin C; Golland P
DTI, Tractography

3:20 – 3:40 Non-rigid Groupwise Registration using B-Spline Deformation Model
Balci S; Golland P; Wells W
Non-rigid Registration, Groupwise Registration

3:40 – 4:40 Posters and Break

4:40 – 5:00 An Accessible, Hands-on Tutorial System for Image-Guided Therapy and Medical Robotics Using a Robot and Open-Source Software
Pace D; Kikinis R; Hata N
3D Slicer, LEGO Mindstorms NXT

5:00 – 5:20 Tagged Volume Rendering of the Heart: A Case Study
Mueller D
Tagged Volume Rendering, Coronary Arteries

Poster Presentations

Vessel Enhancing Diffusion Filter
Enquobahrie A; Ibanez L; Bullitt E; Aylward S
Hessian, Vesselness

Spherical Wavelet ITK Filter
Gao Y; Nain D; LeFaucheur X; Tannenbaum A
Shape Analysis, Spherical Wavelet

Fast BlockMatching Registration with Entropy-based Similarity
Suarez-Santana E; Nebot R; Westin C; Ruiz-Alzola J
Nonrigid Digistration, Multimodal Registration

Optimizing ITK's Registration Methods for Multi-processor, Shared-Memory Systems
Aylward S; Jomier J; Barre S; Davis B; Ibanez L
ITK, Parallel

GoFigure and The Digital Fish Project: Open tools and open data for an imaging based approach to system biology
Gouaillard A; Brown T; Bronner-Fraser M; Fraser S; Megason S
Confocal Imaging, Cell Tracking

Data, Data Everywhere, nor an Image to Read - Finding Open Image Databases
Holmes D; Robb R
Image Databases

Open Source and Open Data for MICCAI

Want to share, replicate, extend, or compare software and data presented at the main conference? Want to publish or learn about the full set of parameters and processes that were needed to achieve results presented at the main conference? This workshop features peer-reviewed submissions that describe the available data and software of MICCAI. We encourage submissions that are complimentary to papers being presented at the main conference, as well as submissions on novel topics.

Important dates for the workshop:

- July 1: Deadline for submissions
- September 1: Deadline for reviews
- September 5: Notification of acceptance
- October 29: MICCAI conference begins
- November 2: Workshop at MICCAI

The receipt, review, and publication of submissions to this workshop will use the open-access Insight Journal. Authors are encouraged to include other files with their submission, particularly videos, code, and data. Also, submissions are immediately available to any registered member of the Insight Journal (registration is free); and any member may contribute a public, peer-review of any submission. Authors may also post responses to reviews and upload revisions to their submissions.

On September 1, the workshop committee will consider the posted reviews, as well as the intent of the workshop, to select papers for oral and poster presentation. Those decisions will be announced on or about September 5.

When appropriate, authors will be offered the opportunity to have their code distributed with the Insight Toolkit (ITK). However, the use of ITK is not a requisite for submitting to the workshop; featuring other toolkits is strongly encouraged.

Visit the workshop website at
http://www.insightsoftwareconsortium.org/wiki/index.php/2007_MICCAI_Open_Workshop