MATLAB-ITK Interface for Medical Image Filtering, Segmentation, and Registration

Vincent Chu, Ghassan Hamarneh

School of Computing Science, Simon Fraser University,
Burnaby, BC, V5A 1S6, Canada

ABSTRACT

To facilitate high level analysis of medical image data in research and clinical environments, a wrapper for the ITK toolkit is developed to allow ITK algorithms to be called in MATLAB. ITK is a powerful open-source toolkit implementing state-of-the-art algorithms in medical image processing and analysis. However, although ITK is rapidly gaining popularity, its user base is mostly restricted to technically savvy developers with expert knowledge of C++ and advanced programming concepts. MATLAB, on the other hand, is well-known for its easy-to-use, powerful prototyping capabilities that significantly improve productivity. Unfortunately, the 3D image processing capabilities of MATLAB are very limited and slow to execute. With the help of the wrapper we introduce in this paper, biomedical computing researchers familiar with MATLAB can harness the power of ITK while avoiding learning C++ and dealing with low-level programming issues. We strongly believe this functionality will be of considerable interest to the medical image computing community. In this paper we provide details about the design and usage of this interface in medical image filtering, segmentation, and registration.

Keywords: ITK, MATLAB, Medical image analysis, filtering, segmentation, registration, MATITK

1. INTRODUCTION

MATLAB,[1] short for MATrix LABoratory, is an environment developed by the Mathworks, Inc. that facilitates matrix computations, numerical analysis and graphics viewing. MATLAB is often used by scientific researchers and biomedical engineers as it provides a high-level programming language that alleviates the users from low-level programming details such as memory management and pointer handling. Complicated peripheral tasks, such as GUI creation, graph plotting, statistical analysis, and 2D image acquisition, can be readily handled by the wide variety of toolboxes available. ITK,[2] short for the Insight ToolKit, originally developed to support the Visible Human project,[3] is a free, open-source toolkit written and used in a C++ environment. The toolkit contains various filtering, segmentation and registration algorithms designed for medical image analysis. While many researchers and engineers are comfortable with the one dimensional biomedical signal processing capabilities of MATLAB, as the dimensionality of the data increases (to two and three dimensional images) both the unavailability of advanced algorithms and slow processing speed quickly become a bottleneck. The latter mainly attributed to the interpreted nature of the MATLAB programming language. It is therefore desirable to facilitate the use of the state-of-the-art, compiled, fast, 3D (and higher) medical image processing capabilities of ITK while working in the fast-prototyping, high-level, environment of MATLAB that doesn't require intimate knowledge of C++, generic programming, and other advanced ITK programming concepts.

MATLAB has the functionality to access dynamically linked libraries compiled in another language such as C and Fortran, given that it conforms to certain criteria. Such a library, also referred to as MEX file (for MATLAB EXecutables), can be run from the MATLAB environment like MATLAB M-functions or built-in functions.[4] With MATLAB's large user-base in the medical image analysis community, it is therefore desirable to allow researchers to employ external medical image processing libraries in the MATLAB environment. Common tasks in medical image computing are already simplified in MATLAB by existing libraries such as MATLAB's own image processing toolbox and DICOM readers, in addition to other third party toolboxes including the Statistical Parametric Mapping (SPM),[5] the Extensible MATLAB Medical Analysis (EMMA),[6] Scientific Image Processing Toolbox (DIPimage),[7] the Image Fusion Toolbox,[8] the Computer Vision and Image Analysis libraries,[9] Geometric modeling toolboxes (e.g. NURBS[10]), Pattern Recognition Toolbox (PRTools),[11] and SDC Morphological Image Processing Toolbox.[12]

Medical Imaging 2006: Image Processing, edited by Joseph M. Reinhardt, Josien P. W. Pluim,
Proc. of SPIE Vol. 6144, 61443T, (2006) · 0277-786X/06/$15 · doi: 10.1117/12.652628

The work presented is a MEX that serves as a wrapper, or an interface between MATLAB computation environment and ITK, hereafter referred to as MATITK. Because image data is represented differently, the wrapper provides the necessary translations in an efficient manner. Notably, because medical image data volumes are often huge, it is impractical to write the image volumes to disk in a suitable format and read the volumes back with pre-compiled ITK algorithms. The wrapper allows the translation to be done in memory, which often provides an order of magnitude increase in speed. As an alternative, we considered using CableSwig to serve our purpose. CableSwig is a tool that specifically designed for generating interfaces for interpreted languages with the current support limited to Tcl and Python. However, extending CableSwig to support MATLAB is not straightforward and our simpler approach achieves the desired goal within the appropriate time frame.

The remainder of the paper describes in detail the setup of the MATLAB environment (section 2.1), the architectural design of MATITK (section 2.2), how calls to filtering, segmentation, and registration methods are enabled (section 2.3) and how their creation is automated (section 2.4). The simple procedure for *using* MATITK is described (section 2.5) and demonstrated with examples (section 3), and finally conclusions are drawn (section 4).

2. METHODS

2.1. MATLAB Setup for Building MEX

We used MATLAB 7.0 on MS Windows as our testing environment. The first step for creating the interface is to configure a MATLAB compatible compiler.[13] We use MS Visual Studio 2003 C++ compiler. This step is accomplished via the MATLAB command $mex - setup$. Automatically-generated options file *mexopts.bat* is replaced with *mexopts.bat* provided with MATITK package containing the ITK header and library paths*. The ITK toolkit v1.80 is used and the wrapper code is written in C++.

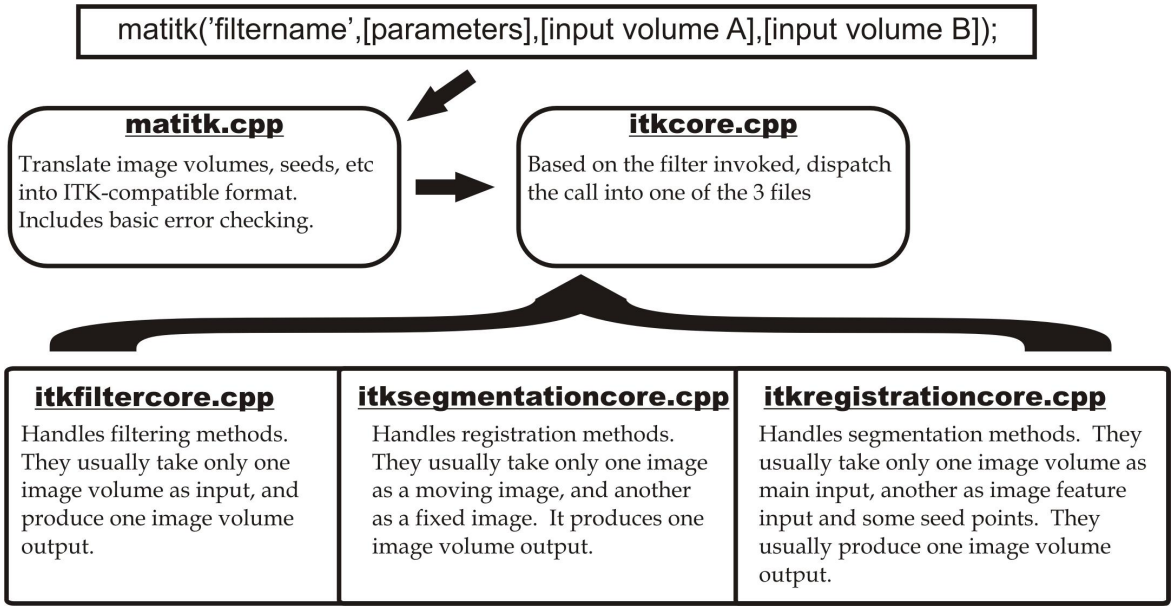

Figure 1. MATITK Execution Flowchart.

*Can be obtained by building a C++ ITK project file with CMake[14] and noting these paths as viewed from project properties in Visual Studio.

2.2. Architectural Design

When MATITK command is issued in MATLAB the code compiled from *matitk.cpp* is executed. *matitk.cpp* handles error handling and translation of image data passed from MATLAB into ITK-compatible format. This includes dealing with indexing differences between MATLAB and C++ arrays (ordering of dimensions and zero- vs. one-based array indexing). The resulting image will be stored in an ITK Image container accessible by other parts of the pipeline. Based on the MATITK command invoked, *itkcore.cpp* calls one of three procedures: *itkfiltercore.cpp*, *itksegementationcore.cpp* or *itkregistrationcore.cpp*, depending on whether the command invoked is a medical image filtering, segmentation, or registration command, respectively (Figure 1). Helper classes *seedcontainer.cpp* and *parametercontainer.cpp* contain the user-supplied seeds and parameters respectively. ITK methods can access the necessary seed points and parameters in the three ITK core files.

2.3. Addition of New ITK Methods

In the following we summarize the procedure for adding ITK filtering, segmentation and registration methods to MATITK. Readers interested in using MATITK only can skip this section.

2.3.1. Filtering

The following is a typical filtering code derived from ITK-provided example. ITK filtering code usually takes one input image volume and a few parameters. When a new ITK method is added, its respective *#include* must be added to the beginning of *itkfiltercore.cpp*.

```
   #include "itkDiscreteGaussianImageFilter.h"
   ...
 0 void filterGaussian(){
 1   const char* PARAM[]={"gaussianVariance","maxKernelWidth"};
 2   const char* SUGGESTVALUE[]={"","0"};
 3   const int nParam = sizeof(PARAM)/sizeof(*PARAM);
 4   ParameterContainer paramIterator(PARAM,SUGGESTVALUE,nParam);
 5   double gaussianVariance=paramIterator.getCurrentParam(0);
 6   unsigned int maxKernelWidth=(unsigned int)paramIterator.getCurrentParam(1);
 7   typedef itk::DiscreteGaussianImageFilter<InternalImageType,InternalImageType> FilterType;
 8   FilterType::Pointer filter = FilterType::New();
 9   filter->SetInput(importFilter[IMPORTFILTERA]->GetOutput());
10   filter->SetVariance( gaussianVariance );
11   filter->SetMaximumKernelWidth( maxKernelWidth );
12   filter->Update();
13   pixelContainer=filter->GetOutput()->GetPixelContainer(); }
```

The beginning of the function defines the parameters required by the filter. Line 1 and line 2 define the human-readable names of the parameters, and the corresponding suggested values of the parameters respectively. The suggested values will be listed along with the human-readable names to the users when the wrapper is invoked without supplying the required parameters. Currently 3D double image type is the only supported input/output data volume format. Line 9 exhibits how the input image volume passed from MATLAB can be accessed by the filter being added. Parameters required by the filter can be accessed by $paramIterator.getCurrentParam(i)$;. The output of the filter will be passed back to MATLAB environment via the exit statement as on line 13. Hence, every ITK code in *itkfiltercore.cpp* should end with the same exit statement. To allow the newly added filter to be called, the following additional step has to be taken:

```
const char* OPCODE[]={"FGA"};
const char* OPNAME[]={"filterGaussian"};
pt2Function OPFCNARRAY[]={&filterGaussian};
```

A meaningful opcode should be invented for the newly added ITK method and inserted at the end of the opcode array. All filtering opcodes must begin with the character 'f'. The opcode will be entered by the user to invoke this added ITK method. *opname* array stores the human-readable name that will be listed when the help mode is invoked. $opfcnarray$ stores the function pointer of C++ method.

2.3.2. Segmentation

The following is a typical segmentation code (edited for brevity) derived from ITK-provided example. ITK segmentation code usually takes two input image volume, a few parameters and an array of seed points.

```
void segmentationGeodesicActiveContourLevelSet(){
    const char* PARAM[]={"propagationScaling", .../*some more parameters*/};
    const char* SUGGESTVALUE[]={"","1.0","1.0","0.02","800"};
    const int nParam = sizeof(PARAM)/sizeof(*PARAM);
    ParameterContainer paramIterator(PARAM,SUGGESTVALUE,nParam);
    if (emptyImportFilter[IMPORTFILTERB]){mexErrMsgTxt("...")}
    mexPrintf("\nThis method requires two image volumes....\n");
    ////////////////////Begin Core Filter Code//////////////////////
    double propagationScaling=paramIterator.getCurrentParam(0);
    //... edited for brevity.  The other 4 parameters can be accessed in a similar fashion
    typedef itk::GeodesicActiveContourLevelSetImageFilter<InputImageType,OutputImageType>...
    GeodesicActiveContourFilterType::Pointer filter = GeodesicActiveContourFilterType::New();
    filter->SetPropagationScaling( propagationScaling );
    //... edited for brevity.  The other 4 parameters are set in a similar fashion as the line above.
    filter->SetInput(importFilter[IMPORTFILTERB]->GetOutput());
    filter->SetFeatureImage( importFilter[IMPORTFILTERA]->GetOutput());
    filter->Update();
    //...omitted code for setting up and connecting additional filters ..
    pixelContainer = thresholder->GetOutput()->GetPixelContainer();
    ////////////////////End Core Filter Code/////////////////////////
}
```

Lines that also appear in the filtering code excerpt serve identical purposes in the segmentation code. If the segmentation method requires more than one input image volume, the second input image volume can be access with $importFilter[IMPORTFILTERB]$. Before the use of $IMPORTFILTERB$, it is important to check whether a second input volume has been specified in the method using $emptyImportFilter[IMPORTFILTERB]$. This is designed because not all segmentation methods require a second input volume. Every ITK code in *itksegmentationcore.cpp* should end with an exit statement similar to the one in $itkfiltercore.cpp$ to pass the result back to MATLAB. Currently, only segmentation methods that produce one output image volume are supported. All segmentation opcodes must start with character 's'.

2.3.3. Registration Methods

The procedure for addition of registration methods is similar to that of segmentation methods. Refer to section 2.3.2 for details. All registration opcodes must start with character 'r'.

2.4. Automatic Generation of Filtering Code

Although this version of MATITK includes over 30 ITK methods, it is desirable to add filters with relative ease as new ITK filter come out. Due to the highly structured nature of most ITK filtering code, an automation perl script,[15] *matitkcode.pl*, is created to facilitate the conversion of ITK example files into MATITK methods with little human intervention. Upon successful execution of the script, an output file is created containing the generated code for every example file in a folder. The generated code below is an excerpt from a sample output. The script derives this section of code based on the example file, $DiscreteGaussianImageFilter.cxx$.

```
 void DiscreteGaussianImageFilter(){
   const char* PARAM[]={"gaussianVariance","maxKernelWidth"};
   const char* SUGGESTVALUE[]={"",""};
   const int nParam = sizeof(PARAM)/sizeof(*PARAM);
   ParameterContainer paramIterator(PARAM,SUGGESTVALUE,nParam);
   ///////////////begin core filter code//////////////////////
   typedef itk::DiscreteGaussianImageFilter<InputImageType, OutputImageType>  FilterType;
   FilterType::Pointer filter = FilterType::New();
   filter->SetInput( importFilter[IMPORTFILTERA]->GetOutput() );
   filter->SetVariance( gaussianVariance );
   filter->SetMaximumKernelWidth( maxKernelWidth );
   filter->Update();
   rescaler->SetInput( filter->GetOutput() );
   pixelContainer = rescaler->GetOutput()->GetPixelContainer();
   ///////////////end core filter code//////////////////////
```

Unfortunately, as for any code generation application, it is not perfect. Hence, it is important to proof read the code. The required *#include* directives can be found in the beginning of the code generation output file. Recommended *opcodes*, *opnames*, and function pointer entries can be located at the end of file and can be inserted readily to $itkfiltercore.cpp$.

2.5. Using the wrapper

Installing and using MATITK is extremely simple. *matitk.dll* is available for download.[16] Simply copy *matitk.dll* to the desired location and the wrapper is ready to be used. For convenience, the location should be added to MATLAB's search path (or change the current working directory). MATITK commands can then be invoked in MATLAB environment by simply typing *matitk*; which writes the following to MATLABs window:

```
matitk(operationName,[parameters],[inputArray1],[inputArray2]
,[seed(s)Array],[Image(s)Spacing])
```

This help information states that the first argument to *matitk*, $operationName$, specifies the opcode of the ITK method to be invoked. To list out implemented methods, type $matitk('?')$;. To list out only methods belonging to filtering, segmentation or registration, type $matitk('f')$;, $matitk('s')$;, and $matitk('r')$; respectively. The following are the currently supported MATITK methods.

The second argument to *matitk*, parameters, specifies the required parameters of the ITK method to be invoked (specified by operationName). To find out what parameters are required for a particular method, type $matitk(operationName)$; For example to perform anisotropic diffusion filtering on a 3D image, the user types $matitk('FCA')$ and the following will be written to MATLAB's window that lists the required parameters:

```
FCA is being executed... You must supply parameters for this
function in an array, with the elements in this order:
numberOfIterations, timeStep (which usually has value equal to
0.0625), conductance (which usually has value equal to 3.0) 3
parameters must be supplied.  You supplied 0.
```

The third and fourth arguments to *matitk*, inputArray1 and inputArray2, specify the input image volumes. They must be three dimensional and contain double data type elements. In the case where a second image volume is not required for the method being invoked, provide [] (open and close square brackets) as the fourth argument. The fifth argument, seedsArray, specifies the seed points (in MATLAB coordinate system) in the following order: [x_1, y_1, z_1, ..., x_n, y_n, z_n]. Because it is three dimensional, the number of elements in seedsArray should be a multiple of three. In the case where seeding is not required for the method, provide [] as the fifth argument. The last optional argument specifies the spacing of the supplied image volume. The performance of certain ITK methods may be affected by the spacing. If this argument is omitted, an isotropic spacing of $[1, 1, 1]$ is assumed.

3. RESULTS

Filtering, segmentation and registration methods in MATITK are included in table 1. Figures 2a-c present the result of applying anisotropic diffusion (FCA) filtering followed by confidence connected segmentation (SCC) to a sample image by issuing the following commands in MATLAB:

```
>> load mri; D=squeeze(D);
>> b=matitk('FCA',[5 0.0625 3],double(D));
FCA is being executed... FCA has completed.
>> c=matitk('SCC',[1.4 10 255],double(b),[],[102 82 25]);
SCC is being executed... SCC has completed.
```

Figures 2d-f present the result of another example. The purpose here, of course, is not to optimally analyze medical images but rather to demonstrate the use of MATITK.

Opcode	**Corresponding filter name**
FGA	filterGaussian
FCA	filterCurvatureAnsio
FCF	filterCurvatureFlow
FMMCF	filterMinMaxCurvatureFlow
FGM	filterGradientMagnitude
FGMS	filterGradientMagnitudeWithSmoothing
FSN	filterSigmoidNonlinearMapping
FBD	filterDilate
FBE	filterErode
FDM	filterDanielssonDistanceMapImageFilter
FDMV	filterDanielssonDistanceMapImageFilterGetVoronoiMap
FBL	filterBilateral
FBT	BinaryThresholdImageFilter
FBB	BinomialBlurImageFilter
FD	DerivativeImageFilter
FDG	DiscreteGaussianImageFilter
FF	FlipImageFilter
FGAD	GradientAnisotropicDiffusionImageFilter
FGMRG	GradientMagnitudeRecursiveGaussianImageFilter
FLS	LaplacianRecursiveGaussianImageFilter
FMEANF	MeanImageFilter
FMEDIANF	MedianImageFilter
SCC	segmentationConfidenceConnected
SIC	segmentationIsolatedConnected
SNC	segmentationNeighbourhoodConnected
SCT	segmentationConnectedThreshold
SFM	segmentationFastMarch
SOT	segmentationOtsuThreshold
SGAC	segmentationGeodesicActiveContourLevelSet
SLLS	segmentationLaplacianLevelSetLevelSet
RTPS	registerThinPlateSpline
RD	registerDemon

Table 1. MATITK available opcodes and the corresponding opnames.

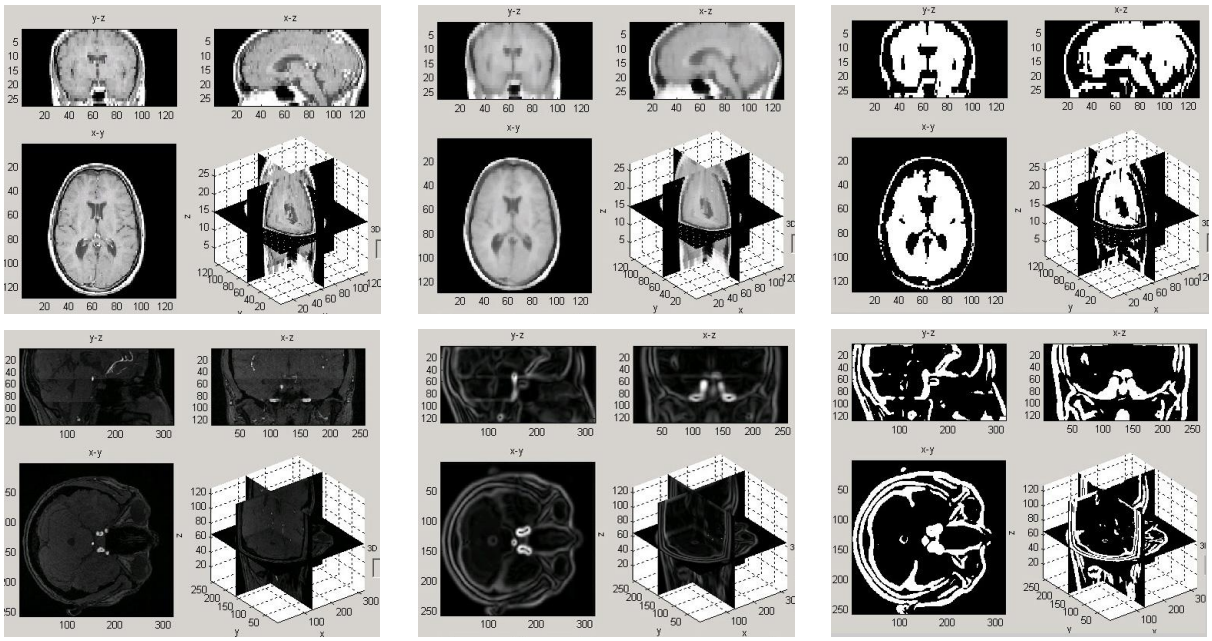

Figure 2. Example MATITK results. Left to right, top row: Original 3D image, anisotropic smoothing, connected component segmentation. Bottom row: original image, gradient magnitude, thresholding. Images are visualized using Orthoview (View3D)[17]

4. CONCLUSIONS

We presented MATITK, an easy to install, use, and extend, MATLAB-ITK interface. MATITK enables researchers and scientists to easily and efficiently access advanced medical image processing and analysis methods of ITK from MATLAB. Previously, users needed to become familiar with advanced C++ programming concepts to use ITKs methods, build customized MEX files for each method, or save an image volume from MATLAB to disk and open it up in ITK. With the framework architecture of this wrapper, additional filters can be added without worrying about the cumbersome data passing and translation from MATLAB to C++. ITK methods with structure similar to existing ones can be added without changes to the data passing mechanism, and can be directly inserted into the appropriate core files without affecting other modules. For this purpose, a script file written in Perl can be employed to translate a highly structured filtering example file into code segments that can be readily inserted into this wrapper framework. Possible future work includes developing a more comprehensive list of ITK methods, extending the current framework to accommodate ITK methods with different calling sequences or architecture, and dealing with meta image data such as origin, voxel size, and direction cosines.

REFERENCES

1. M. Natick, *MATLAB Reference Guide*, MathWorks, 1992.
2. L. Obez and W. Shchroeder, *ITK software guide : ITK 1.4 : the Insight segmentation and registration toolkit*, Kitware Inc., 2003.
3. M. J. Ackerman, "Accessing the visible human project," *D-Lib Magazine* , 1995.
4. "Mex-files guide," *Mathworks support* **1605**. http://www.mathworks.com/support/technotes/1600/1605.html [Accessed Nov. 20, 2005].
5. K. Friston, R. Dolan, and R. Frackowiak, *Statistical parametric mapping*, Functional Neuroimaging: Technical Foundations, 1994.
6. M. Wolforth, G. Ward, and S. Marrett, *EMMA: Extensible MATLAB Medical Analysis*, 1995.
7. C. Hendriks, L. van Vliet, B. Rieger, and M. van Ginkel, *DIPimage: a scientific image processing toolbox for MATLAB*, Pattern Recognition Group, TU Delft, 2001.

8. Rockinger, "Image fusion toolbox." http://www.metapix.de/toolbox.htm [Accessed Nov 20, 2005].
9. P. D. Kovesi, "MATLAB and Octave functions for computer vision and image processing." School of Computer Science & Software Engineering, The University of Western Australia. http://www.csse.uwa.edu.au/∼pk/research/matlabfns/ [Accessed Nov 20, 2005].
10. M. Spink, "The matlab nurbs toolbox." http://www.aria.uklinux.net/nurbs.php3 [Accessed Nov 20, 2005].
11. F. van der Heijden, R. P. Duin, D. de Ridder, and D. M. Tax, *Classification, parameter estimation and state estimation - an engineering approach using Matlab*, John Wiley & Sons, 2004.
12. E. R. Dougherty and R. A. Lotufo, *Hands-on Morphological Image Processing*, SPIE Tutorial Texts in Optical Engineering Vol. TT59, 2003.
13. "Matlab supported compilers," *Mathworks support* **1601**. http://www.mathworks.com/support/technotes/1600/1601.html [Accessed Nov. 20, 2005].
14. K. Martin and B. Hoffman, *Mastering CMake*, Kitware Inc., 2003.
15. L. Wall, *The PERL Programming Language*, O'Reilly & Associates, 1990.
16. V. Chu and G. Hamarneh, "MATITK." School of Computer Science, Simon Fraser University. Available from: http://www.cs.sfu.ca/∼hamarneh/software/matitk/.
17. G. Hamarneh, "Orthoview." School of Computer Science, Simon Fraser University. Available from: http://www.cs.sfu.ca/∼hamarneh/software/orthoview/.