SimITK: Model Driven Engineering for Medical Imaging

Melissa Trezise[1], David Gobbi[2], James Cordy[1], Purang Abolmaesumi[1,3], Parvin Mousavi[1]

[1] School of Computing, Queen's University, Kingston, Ontario, Canada
[2] University of Calgary, Calgary, Alberta, Canada
[3] Dept. of Electrical & Computer Engineering, University of British Columbia, Vancouver, Canada

ABSTRACT

The Insight Segmentation and Registration Toolkit (ITK) is a highly utilized open source medical imaging library providing chiefly the functionality to register, segment, and filter medical images. Although extremely powerful, ITK has a steep learning curve for users with little or no background in programming. It was for this reason that SimITK was developed. SimITK wraps ITK into the model driven engineering environment Simulink, a part of the Matlab development suite. The first released version of SimITK was a proof of concept, and demonstrated that ITK could be wrapped successfully in Simulink. In this paper a new version of SimITK is presented where ITK classes are wrapped using a fully automated process. In addition, SimITK is transitioned to successfully support ITK version 4, in order to remain current with the ITK project. SimITK includes thirty-seven image filters, twelve optimizers, and nineteen transform classes from ITK version 4 which are successfully wrapped and tested, and can be quickly and easily combined to perform medical imaging tasks. These classes were chosen to represent a broad range of usability, and to allow for greater flexibility when creating registration pipelines. SimITK has the potential to reduce the learning curve for ITK and allow the user to focus on developing workflows and algorithms. A release of SimITK along with tutorials and videos is available at *www.simitkvtk.com*.

Keywords: SimITK, ITK, Simulink, Medical Imaging, Registration, Model Driven Engineering

1. INTRODUCTION

The Insight Registration and Segmentation Toolkit (ITK)[1] is a comprehensive open-source image segmentation, registration, and filtering toolkit that is popular and widely used in the area of medical image processing. However, many potential users do not attempt to take advantage of this toolkit due to their lack of programming background and the steep learning curve that is associated with ITK. It is for this reason that SimITK was developed[2,3,4].

Originally created as a proof of concept to demonstrate that the complexity of ITK could be reduced by wrapping classes into Simulink, SimITK is now being presented as a system that is well on the way to becoming a usable research tool. SimITK is a domain specific language for ITK that takes advantage of the model driven engineering (MDE) capabilities of Simulink[5]. For each ITK class, there is a specific Simulink block representation, and all written programming code is abstracted away and replaced with a visual model. A simple workflow for an ITK image reader is shown in Figure 1. Several lines of code are required to create the reader in ITK, while a simple block in SimITK can be used to provide the same functionality with no code whatsoever. Blocks can be selected and dropped onto a Simulink canvas, and combined to create a model to represent medical image processing workflows or algorithms. For example, the model shown in Figure 2 (top) will read an image, apply a filter to flip it vertically, and write the image as an output. To perform the same ITK functions in C++, about 100 lines of code would be required.

Several ITK-based systems exist that have attempted to alleviate the learning curve associated with ITK by abstracting away some of its more complex aspects. Examples of these systems include SimpleITK[6], MatITK[7], MeVislab[8], MITK[9], IGSTK[10], ITK-SNAP[11], and 3D Slicer[12]. These systems offer a simplified environment for ITK programming; some use a wrapped version of ITK to allow easier programming in the form of a toolkit (SimpleITK, MatITK, IGSTK, MITK). Others offer a software application with the ability to program visually through pipelines or GUI commands (ITK-SNAP, 3D Slicer, MeVisLab). Many provide a broad range of ITK functionality (amongst other

```
// Define and create CT data reader
typedef short  InputPixelType_CT;
typedef itk::Image< InputPixelType_CT,  3 >   InputImageType_CT;
typedef itk::ImageFileReader< InputImageType_CT >  ReaderType_CT;
ReaderType_CT::Pointer readerCT = ReaderType_CT::New();
readerCT->SetFileName( inputFilenameCT );
```

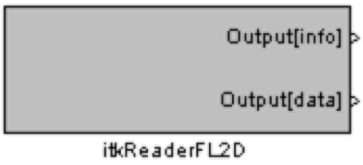
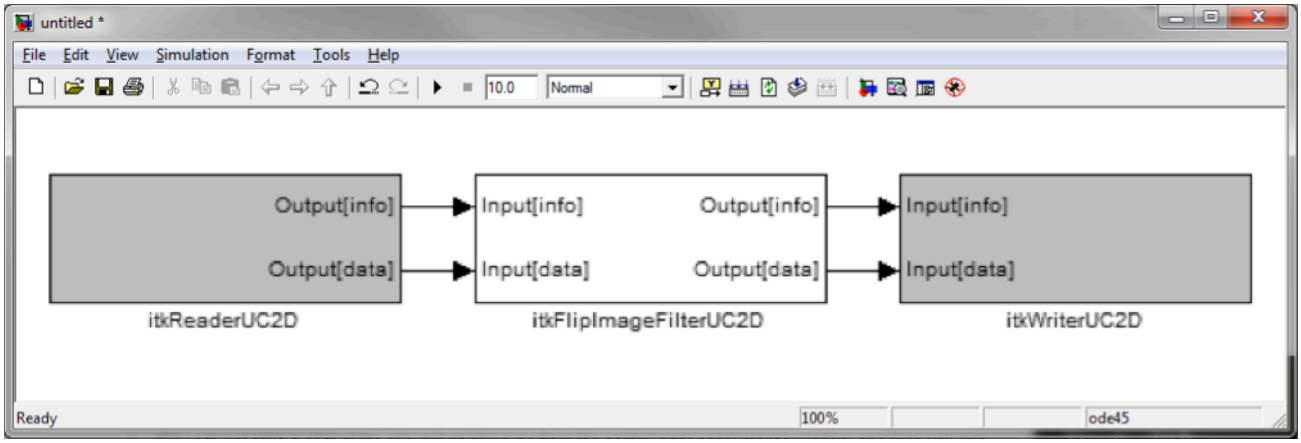


Figure 1: ITK code to generate a reader block (top); SimITK reader block (bottom).



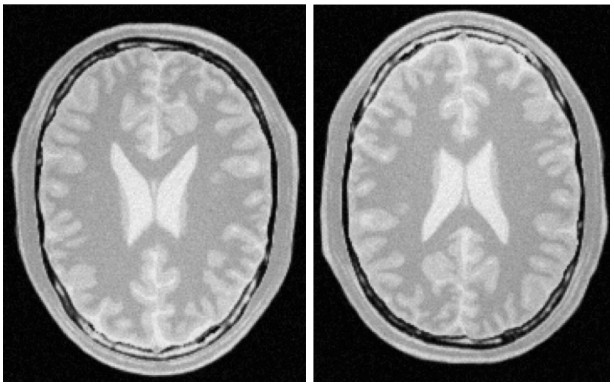


Figure 2: An example SimITK model to read, flip vertically, and write an image (top); the input and out images are shown on the left and right hand side, respectively (bottom).

functionalities) while others focus on specific tasks such as segmentation (e.g., ITK-SNAP). The main difference between all of these systems and SimITK is their choice of platform. With the exception of MatITK, each of these systems requires the user to learn a new application or toolkit. For a novice user, learning a new toolkit can often be time consuming.

Our objective is to integrate ITK with Matlab's Simulink, a toolkit widely used for research and teaching. This integration has the added advantage that it can seamlessly incorporate in-house medical imaging pipelines with ITK, Simulink and Matlab functions. This paper presents the recent advancements in SimITK development including support for ITK version 4, fully automated Simulink wrapper generation for ITK, and increased availability of registration-related ITK classes.

2. METHODOLOGY

An overview of the process for creating SimITK is shown in Figure 3. This process is fully automated and includes a series of source code transformations of input artifacts into output artifacts (arrows). The first step to create SimITK is to automatically convert the ITK header files into an easy-to-parse XML representation of each ITK class[13,14]. The XML captures information about the inputs, outputs, properties, and types of the ITK class in the header file. Once the XML files are automatically generated, a complex templating process occurs which ultimately generates a set of block libraries that can be used in Simulink to create models. An overview of this process can be seen in Figure 4. In this process, Perl scripts are used to automatically populate templates of what we call "Matlab glue files". These glue files, some of which are computer-generated C++ code and others of which consist of Matlab code and data structures, provide the links that tie ITK and Simulink to each other and are necessary to create and initialize SimITK blocks. They are the interface between Matlab (in which Simulink is implemented) and ITK, and they establish the back end connection needed to communicate between them, and also to provide the user-interface controls of the SimITK blocks. The class header file and system function file (Figure 4), for example, contain the C++ code that is responsible for the back end of the block that communicates with ITK. The outputs of the templating process in Figure 4 are a Matlab Executable Interface file (*mex* file), a block property file, and a Simulink library. The *mex* file represents the back end, the block property file is the appearance of the block and the library is loaded when a user accesses SimITK in Simulink. Simulink blocks and block libraries for SimITK are created from these many files and can be combined to create SimITK process models.

Each generated block represents a different ITK class and can have inputs, outputs, and properties that can be modified by the user. These blocks are organized in groups called block libraries based on their function, examples of which can be seen at the far right of Figure 4. Once the function blocks are available, SimITK process models can be created. Blocks can be joined together in the SimITK workspace within Simulink to create pipelines to perform medical imaging tasks, and these pipelines can furthermore be used as subsystems within a Simulink-based application, so the complexity of the application is not limited to what can be seen on a single canvas. The Simulink interface, along with SimITK blocks joined together in an example pipeline, can be seen in Figure 5.

Figure 3: The SimITK development process.

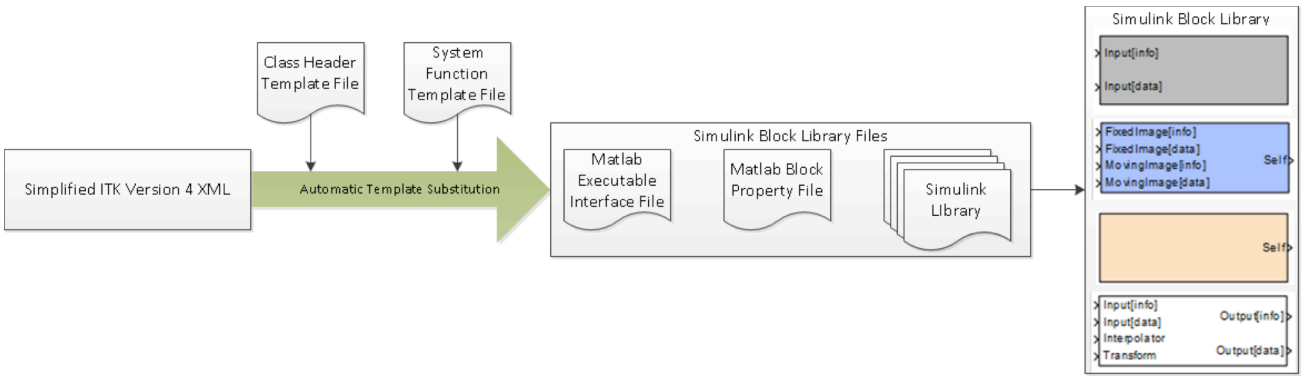


Figure 4: SimITK templating process.

3. EXAMPLE APPLICATION

A registration workflow for 3D CT to MRI registration using SimITK is shown in Figure 5 (top). This workflow loads CT and MR image volumes, performs a registration using the "MattesMutualInformationImageToImageMetric" with the "RegularStepGradientDescent" optimizer, and writes the output image to disk. The specific registration model shown, registers the CT of the head to its corresponding MRI. The two input images and the fused registration results are shown in Figure 5 (bottom). In addition, the user has the ability to monitor the metric value output from the optimizer, and the output of the registration block, as registration proceeds. These output values are also plotted using a Simulink scope block and can also be saved to the Matlab workspace for analysis. The graphs of the metric value demonstrate that registration has converged and has therefore been successful.

In the SimITK registration model presented in Figure 5, the user has the option to substitute the optimizer, transform, or metric block with other types of the operations based on the registration algorithm picked, and the dimensionality of the images. The user can choose from 12 optimizers, 15 2D transform classes, 14 3D transform classes, and five metrics that are available. In addition, the parameters of algorithms (such as the number of bins in the joint histogram of the metric) can be adjusted by double-clicking on the blocks and adjusting the values in the provided block interface. Since the algorithms are executing as compiled C++ code, via the SimITK glue code that ties Simulink to the ITK libraries, the execution time is similar to that of a native ITK pipeline.

4. DISCUSSION

SimITK provides a visual programming environment for ITK, in which image processing applications can be written without the need of a textual programming language. By providing a domain-specific language for image processing that incorporates the functionality of ITK, but without the steep learning curve of a C++ development environment, the SimITK project aims to make it possible for research groups to exploit the talents of scientists and students who have solid knowledge of image processing methodologies but limited software development experience.

Our overall goal of the presented work is to transition SimITK from a proof of concept to a usable research tool. At this time, the transition has begun by speaking with instructors who used SimITK in their class and for research to determine necessary improvements. A full usability study is needed to further our knowledge of the requirements of our user base in a systematic manner. This study would allow us to determine which classes need to be wrapped that are not currently available and which functionality needs to be improved or implemented.

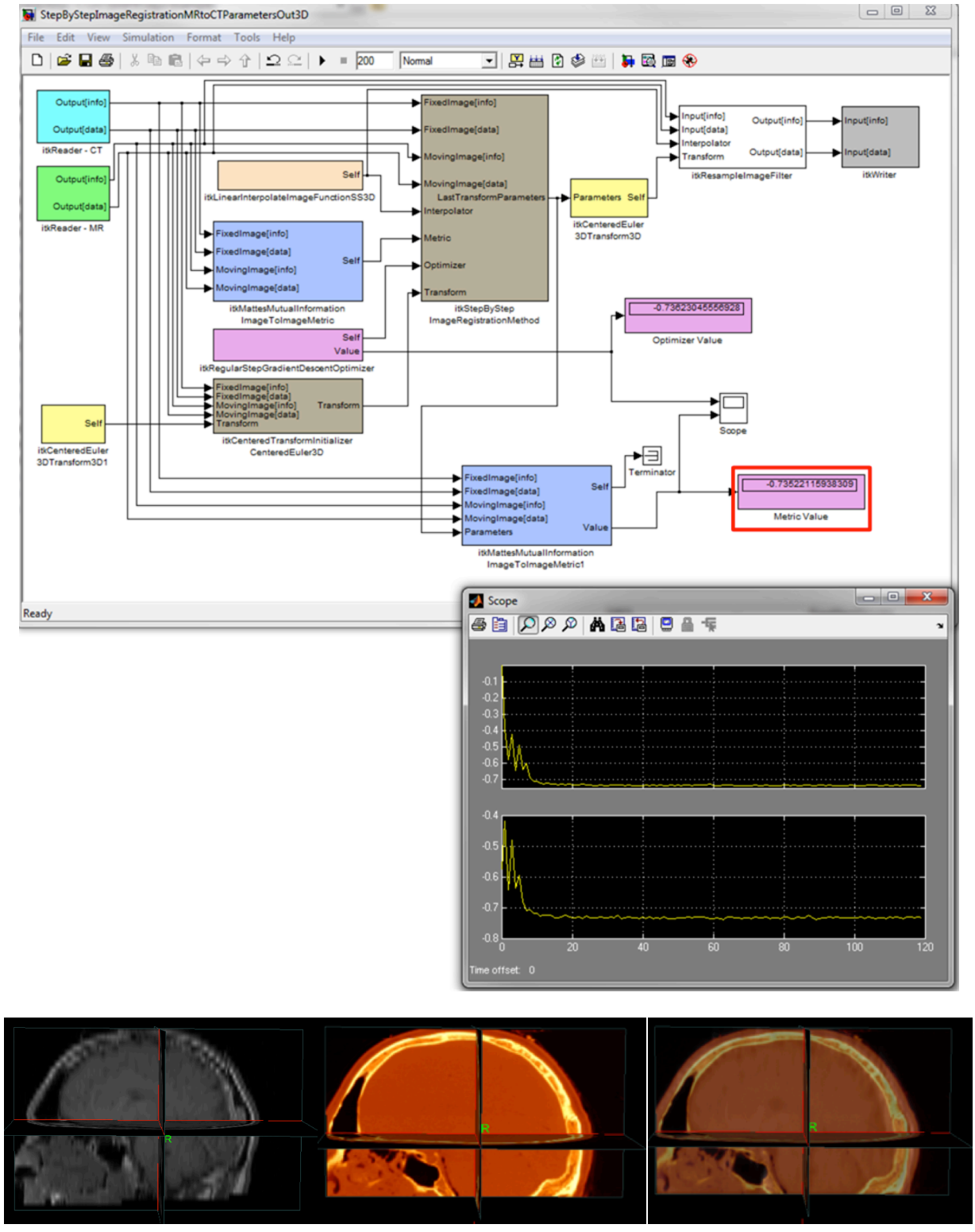


Figure 5. Example of a 3D registration model wrapping ITK version 4 with metric value output monitored via a scope (top); the MRI and CT images and their registration results overlaid (bottom).

5. CONCLUSIONS

This new release of SimITK includes usability improvements through transitioning to ITK version 4 and improving the registration pipeline. The software, tutorials, videos and information for developers are all available from *www.simitkvtk.com*. Development focused on support for more ITK types, registration-specific classes, and the creation of an automatic process to create XML files within the SimITK CMake process to enable the seamless transition needed to keep SimITK current. We believe the ability to create complex ITK pipelines by connecting Simulink blocks in SimITK, allows the users to focus on algorithm and workflow development for learning and teaching ITK.

For future work, it would be beneficial to expand the number of classes included in SimITK. In order to facilitate this improvement, the templating process needs to be expanded to support and resolve the ITK types that are not currently supported. Another barrier to wrapping more classes is the set up of the block inputs and outputs that assumes all filters have one input and one output. Generalizing this assumption would greatly increase the number of classes that could be supported. In SimITK's sister project SimVTK[4], the documentation for each block is available to the user with a click. This information is obtained from the header file of the class in VTK and is automatically added during the SimVTK wrapping process. This is another potential addition to SimITK that would make the classes easier to use and understand by eliminating the need to search for native ITK documentation.

REFERENCES

[1] L. Ibanez, W. Schroeder, L. Ng, and J. Cates, [ITK Software Guide: The Insight Segmentation and Registration Toolkit (Version 2.4)], Kitware Inc., Clifton Park, NY (2003).

[2] A.W.L. Dickinson, D.G. Gobbi, P. Abolmaesumi, P. Mousavi, "SimITK: Visual Programming of the ITK Image Processing Library within Simulink," Journal of Digital Imaging, accepted for publication (2013).

[3] A.W.L. Dickinson, P. Mousavi, D.G. Gobbi, and P. Abolmaesumi, "SimITK: Rapid ITK prototyping using the Simulink visual programming environment," Proc. SPIE Medical Imaging 7964, 796438, Lake Buena Vista, FL (2011).

[4] D. Gobbi, P. Mousavi, K. Li, J. Xiang, A. Campigotto, A. LaPointe, G. Fichtinger, and P. Abolmaesumi, "Simulink libraries for visual programming of VTK and ITK," MIDAS Journal-Systems and Architectures for Computer Assisted Interventions (MICCAI 2008 Workshop), New York, NY (2008).

[5] Mathworks Inc., Simulink, http://www.mathworks.com/products/simulink/, (accessed 13 December 2013).

[6] SimpleITK, http://www.simpleitk.org/, (accessed 10 December 2013).

[7] V. Chu and G. Hamarneh. MATLAB-ITK interface for medical image filtering, segmentation, and registration, Proc. SPIE Medical Imaging 6144, 61443T, San Diego, CA (2006).

[8] F. Ritter, T. Boskamp, A. Homeyer, H. Laue, M. Schwier, F. Link, and H.-O. Peitgen, "Medical Image Analysis: A Visual Approach," IEEE Pulse 2(6), 60-70 (2011).

[9] I. Wolf, M. Vetter, I. Wegner, T. Bttger, M. Nolden, M. Schbinger, M. Hastenteufel, T. Kunert, and H.-P. Meinzer. "The Medical Imaging Interaction Toolkit," Medical Image Analysis 9(6), 594-604 (2005).

[10] A. Enquobahrie, P. Cheng, K. Gary, L. Ibanez, D. Gobbi, F. Lindseth, Z. Yaniv, S. Aylward, J. Jomier, and K. Cleary, "The image-guided surgery toolkit IGSTK: an open source C++ software toolkit," Journal of Digital Imaging 20, 21-33 (2007).

[11] P. A. Yushkevich, J. Piven, H. Cody, S. Ho, J. C. Gee, and G. Gerig, "User-guided level set segmentation of anatomical structures with ITK-SNAP," Insight Journal 1 (2005).

[12] S. Pieper, M. Halle, and R. Kikinis. 3D Slicer, Proc. IEEE International Symposium on Biomedical Imaging: Nano to Macro, 632-635, Arlington, VA (2004).

[13] D. Gobbi. "VTK wrapper overhaul," Kitware Source 15, 9-11, Kitware Inc., Clifton Park, NY (2010).

[14] D.G. Gobbi, P. Mousavi, A. Campigotto, Dickinson A. W.L., and P. Abolmaesumi, "Visual programming of VTK pipelines in Simulink," The VTK Journal (2010).