Simulink Libraries for Visual Programming of VTK and ITK

Release 1.0

D. G. Gobbi[1], P. Mousavi[1], K. M. Li[1], J. Xiang[1], A. Campigotto[1], A. LaPointe[1], G. Fichtinger[1] and P. Abolmaesumi[1,2]

July 21, 2008

[1]School of Computing, Queen's University, Kingston, Ontario, Canada
[2]Department of Electrical and Computer Engineering, Queen's University, Kingston, Ontario, Canada
email:dgobbi@cs.queensu.ca, purang@cs.queensu.ca

Abstract

We have created open-source Simulink block libraries for ITK and VTK that allow pipelines for these toolkits to be built in a visual, drag-and-drop style within MATLAB. Each block contains an instance of an ITK or VTK class. Any block connections and parameters that the user makes within MATLAB's Simulink visual environment are converted into connections and parameters for the ITK and VTK pipelines. In addition, we provide conversion of images to and from MATLAB arrays to allow MATLAB image processing blocks to be mixed with ITK and VTK blocks. The code for our block libraries is generated automatically from XML descriptions of the inputs, outputs, and parameters of the ITK and VTK classes. We have used these block libraries to build some example pipelines and believe that they will be useful for developing applications in image analysis and image-guided therapy.

Contents

1 Introduction **2**

2 Architecture **2**
2.1 SimITK Architecture . . . 3
2.2 SimVTK Architecture . . . 4
2.3 Code Generation . . . 6

3 Results **7**

4 Discussion and Conclusion **7**

5 Acknowledgements **8**

1 Introduction

Both the Insight Toolkit (ITK) and the Visualization Toolkit (VTK) have established themselves as invaluable software packages for image analysis, visualization, and image-guided surgery applications [4],[5]. Although these packages are often used directly as C++ libraries, higher-level interfaces can be useful where the toolkit user either lacks knowledge of C++, or requires a more rapid development schedule than C++ allows. Such high-level interfaces come in three varieties. First, scripting languages such as Python and Tcl have been interfaced to the toolkits in order to simplify the programming [8]. Second, general purpose open-source applications have been written that provide GUIs that expose either a broad range of the toolkits' functionality, such as Slicer [5], or a range of functionality that pertains a certain task, such as ITK-SNAP [12]. Third, visual programming environments are available in which the toolkit classes are represented as graphical "blocks" that can be connected with the mouse. The latter category includes MeVisLab [6] and SciRun [10], which are reviewed in detail by Bitter *et al.* [2], as well as ITKBoard [7] and XIP [11]. It is to this category that our contributions, which we call SimITK and SimVTK, belong.

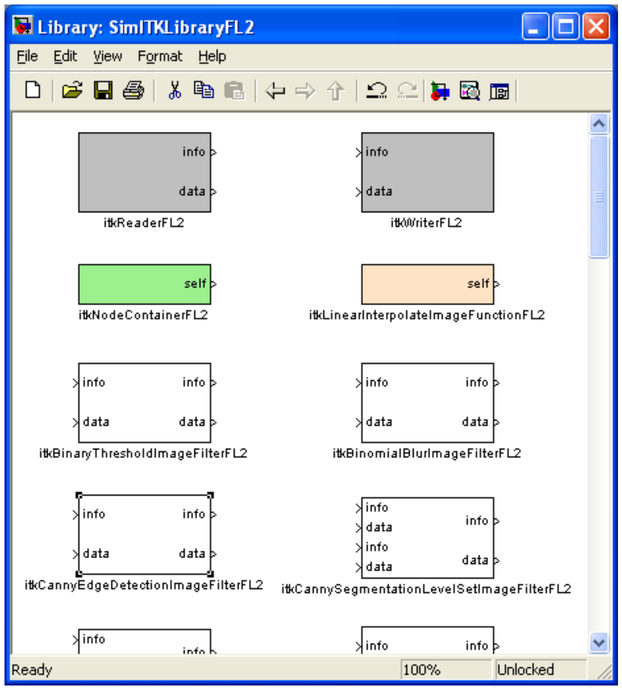
Our choice was to use MATLAB® (MathWorks, Natick, MA, USA) as our platform, due to its ubiquity and its capacity to provide a broad range of computational functionality complementary to ITK and VTK. The ability to utilize ITK from MATLAB has been demonstrated in the MATITK package of Chu and Hamarneh [3], which provides a large collection of ITK filters as MATLAB functions. SimITK and SimVTK, in contrast, provide the ability to use ITK and VTK within MATLAB's powerful Simulink visual programming environment. With Simulink, the algorithmic component of blocks can be written in either MATLAB or C/C++, which are the most commonly used computer languages for image analysis. We have the following design requirements for our software:

1. It must be possible to mix SimITK, SimVTK, and other Simulink blocks in a pipeline.
2. It must be possible to set the ITK and VTK parameters from Simulink.
3. The SimVTK and SimITK blocks must be automatically generated from XML.
4. The full package must be open source, apart from MATLAB and Simulink themselves.

We have collected our blocks into libraries, where each library consists of the loadable modules (i.e. DLLs) that contain the code for each block, plus a text-based .mdl file that specifies how the blocks will be displayed within Simulink. The block code and the .mdl files are generated from XML descriptions of the inputs, outputs, and parameters for each ITK and VTK class (Figure 1). Since the ITK classes are templated over data type and dimensionality, several SimITK libraries are created, each of which is dimension-specific and type-specific. When these block libraries are loaded into MATLAB, the user can drag-and-drop the blocks into a Simulink model to create his or her desired processing pipeline. If a block is double-clicked with the mouse, a dialog box appears that allows any parameters for the underlying ITK or VTK object to be set.

2 Architecture

The way in which blocks are created for Simulink is through files called "S-functions," which can either be MATLAB code or loadable modules (i.e. DLLs) compiled from C or C++, that implement a set of callback functions that Simulink uses to execute the code in the block. Although both ITK and VTK utilize pipeline architectures with many similarities, there are two fundamental differences between these two toolkits which made us choose to utilize a different S-function architecture for each. First, only ITK filters are templated


Library: SimITKLibraryFL2
File Edit View Format Help
itkReaderFL2
itkWriterFL2
itkNodeContainerFL2
itkLinearInterpolateImageFunctionFL2
itkBinaryThresholdImageFilterFL2
itkBinomialBlurImageFilterFL2
itkCannyEdgeDetectionImageFilterFL2
itkCannySegmentationLevelSetImageFilterFL2
info
data
self
Ready
100%
Unlocked

```
<Filter>
  <Filter_Name>BinaryThresholdImageFilter</Filter_Name>
  <Template_Parameters>2</Template_Parameters>
  <Filter_Inputs>
    <Input>
      <Input_Name>Input</Input_Name>
      <Input_Type>InputPixelType</Input_Type>
    </Input>
  </Filter_Inputs>
  <Filter_Parameters>
    <Parameter>
      <Parameter_Name>InsideValue</Parameter_Name>
      <Parameter_Type>OutputPixelType</Parameter_Type>
      <Parameter_Size>1,1</Parameter_Size>
    </Parameter>
    <Parameter>
      <Parameter_Name>LowerThreshold</Parameter_Name>
      <Parameter_Type>InputPixelType</Parameter_Type>
      <Parameter_Size>1,1</Parameter_Size>
    </Parameter>
    <Parameter>
      <Parameter_Name>OutsideValue</Parameter_Name>
      <Parameter_Type>OutputPixelType</Parameter_Type>
      <Parameter_Size>1,1</Parameter_Size>
    </Parameter>
    <Parameter>
      <Parameter_Name>UpperThreshold</Parameter_Name>
      <Parameter_Type>InputPixelType</Parameter_Type>
      <Parameter_Size>1,1</Parameter_Size>
    </Parameter>
  </Filter_Parameters>
  <Filter_Outputs>
    <Output>
      <Output_Name>Output</Output_Name>
      <Output_Type>OutputPixelType</Output_Type>
    </Output>
  </Filter_Outputs>
</Filter>
```

Figure 1: A library of SimITK filter blocks, and the XML description used to generate one of the blocks. The suffix FL2 indicates the type and dimensionality: this library is for operations on floating-point, two-dimensional images.

over data type and dimensionality, and second, ITK parameters are usually class-specific defined types, e.g. itk::AnisotropicDiffusionFunction::Radius, while VTK parameters are fundamental C types such as float, int, or arrays. As a result, the architecture for SimITK is significantly more complex than for SimVTK.

2.1 SimITK Architecture

In SimITK, data is transferred between blocks via two connections, each of which carries a MATLAB array. The first connection carries information about the image (i.e. pixel origin and spacing), while the second connection carries the image data itself. The central feature of the SimITK architecture are “VirtualBlock” helper classes that receive the contents of the MATLAB arrays, and convert those contents to the ITK-specific types used by the ITK filters (see Figure 2). By placing this conversion in a helper class, we keep the S-function source code more clean and understandable.

The way in which the S-function utilizes a VirtualBlock to convert MATLAB arrays to ITK data types is as follows. First, the S-function reads the image origin and spacing from the “information” connections and sets it in the VirtualPorts. Next, the MATLAB arrays that carry the image data are retrieved from each input and output connection, and the memory address of their contents is set in the corresponding VirtualPort. Then, inside of the VirtualBlock, itkImage objects are created and given these memory addresses to use as their own storage space, so that the memory is shared between MATLAB and ITK and no data copying is necessary. After this, Simulink reads the parameter values that were set by the user, and sets the parameter variables in the VirtualBlock to the same values. Finally, the Run() method of the VirtualBlock is executed, causing the ITK filter to process the data.

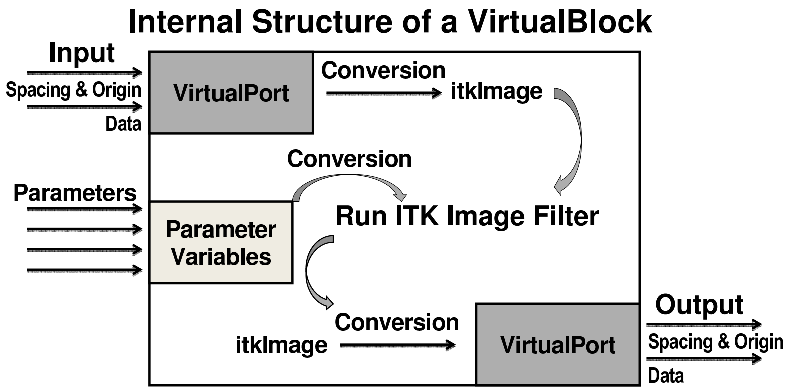


Figure 2: Internal structure of a VirtualBlock. The primary purpose of the block is to convert between the standard C data types used by Simulink, and the special data types used by ITK. No ITK-specific data structures are exposed to Simulink.

Certain ITK filter parameters are special ITK types that need to be treated differently from other parameters. An example is the NodeContainer type that is used in level set segmentation algorithms to store the values and positions of seed points. In SimITK, we handle the NodeContainer by including it in the library as a separate block. This block takes MATLAB arrays containing the node values and positions as parameters, and produces a “self” reference (i.e. a pointer) as output that can be passed as an input to a SimITK filter block. These NodeContainer blocks are generated by the perl scripts directly and do not utilize an XML description. We have created Transform and Interpolator blocks with a similar approach, and will continue to add support for more special blocks.

Since we do not yet have support for all the special types that are required to build a component-wise image registration pipeline in SimITK, we use the ITK Image Registration Helper classes of Aylward *et al.* [1] instead. Whereas image registration in ITK typically requires several C++ objects to be connected together (an ImageToImageMetric object, a Transform object, an Optimizer object, and an Interpolator object), their ImageRegistrationHelperClass encapsulates all of these and we provide it as a SimITK block.

2.2 SimVTK Architecture

In SimVTK, data is passed between blocks by reference, and is not converted into MATLAB arrays as for SimITK. This allows a uniform approach to all varieties of VTK data sets, including images, polydata, and the various grid formats. The tradeoff to using this approach is that transferring data from SimVTK to MATLAB arrays, or vice versa, requires the use of special import and export blocks at either end of the VTK pipeline. We currently provide such blocks for image data but not yet for other data types.

The way in which data is passed by reference, is that the pointer to the vtkObject containing the data is stored in a 1×1 MATLAB array that is passed between blocks. When Simulink runs, these pointers are used by the S-functions to connect the VTK pipeline such that it exactly mirrors the connections between the SimVTK blocks, and any necessary type checking is done via VTK’s run-time type information methods.

While the only way that parameters can be set for SimITK is through the block dialog boxes, for SimVTK it is also possible to set parameters via input connections to the blocks. This provides a great deal of flexibility in constructing pipelines. The manner in which a particular parameter is handled depends on the names of the interface methods for that parameter as they appear in the C++ header file for the VTK class. The

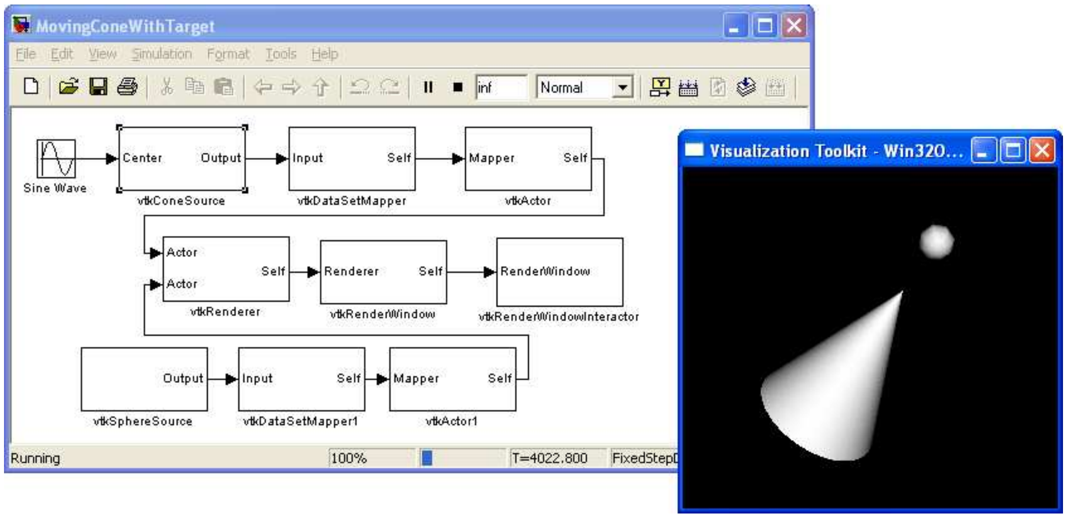


Figure 3: A SimVTK model that connects a cone to a sinusoid source. When the model runs, the sinusoid source generates a vector signal that varies between [-1 0 0] and [+1 0 0], causing the cone to oscillate back and forth and hit the target sphere.

following rules apply for SimVTK, we intend to develop a similar set of rules for SimITK:

"Set" and "Get" methods with scalar and array parameters, e.g. SetPoint(double p[3]). The dialog box allows the user to select from four options for the parameter. *As Parameter:* use an entry box to set the parameter, *As Input:* add a block input to set this parameter, *As Output:* add a block output to get this parameter, *Use Default:* don't call either method, use the default parameter value.

"Set" and "Get" methods with object parameters, e.g. SetMapper(vtkMapper *). For each Set method and Get method there is a checkbox for adding an input, or output, to the block.

"Add" methods with an object parameter, e.g. AddActor(vtkActor *). The dialog has an entry box to select how many inputs the block should have corresponding to this method.

Outputs for algorithms, e.g. GetOutputPort(). All algorithm outputs are included as block outputs.

Inputs for algorithms, e.g. SetInputConnection(). All mandatory algorithm inputs are included as block inputs. We will also add a checkbox for each optional input, and an entry box to select the number of inputs for each repeatable input. The latter will support AddInputConnection() methods.

"Self" as an output. All blocks have a checkbox to select whether the object pointer itself should be included as a block output, so that it can be used as an input to another block.

While Simulink is running the pipeline, all RenderWindows and RenderWindowInteractors are re-rendered at each time step, so that user interactions with the RenderWindow behave as usual. Furthermore, if the values change for any inputs change while Simulink is running, then the result of the change will be displayed when the next render occurs.

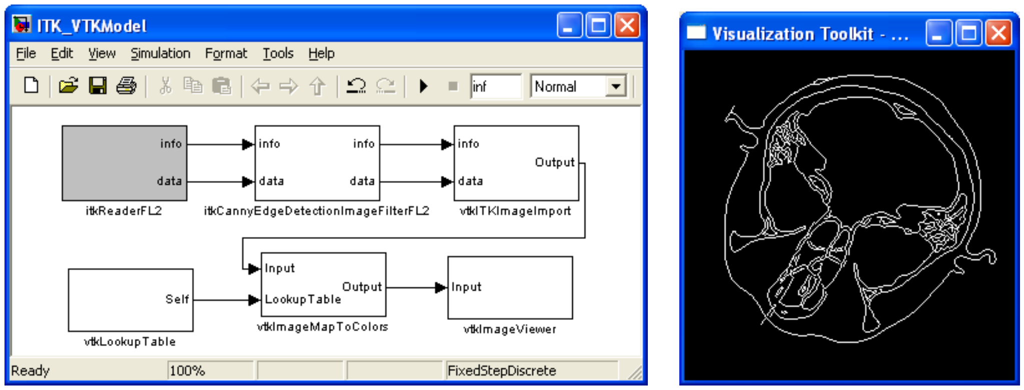
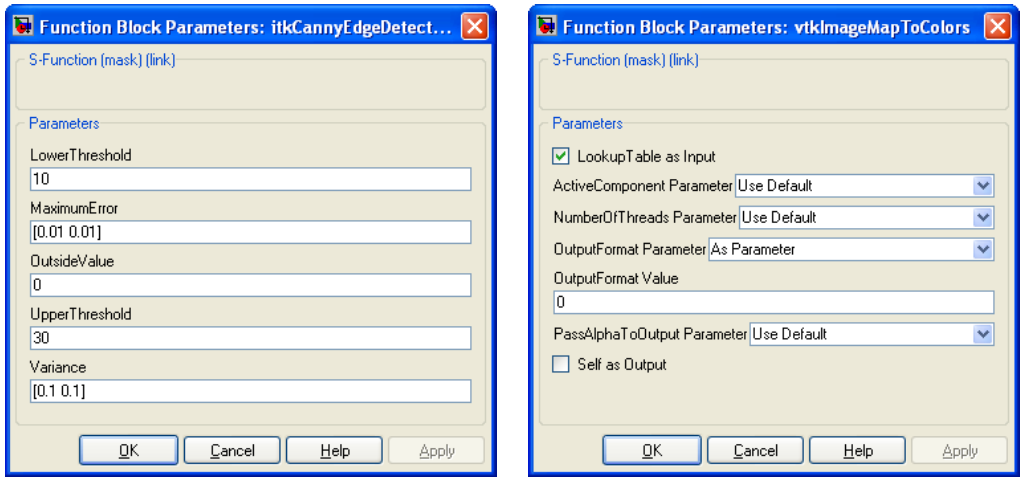


Figure 4: A Simulink model that combines ITK and VTK. This pipeline performs ITK Canny edge detection, applies a greyscale lookup table, and displays the result with vtkImageViewer.


Function Block Parameters: itkCannyEdgeDetect...

S-Function (mask) (link)

Parameters

LowerThreshold: 10

MaximumError: [0.01 0.01]

OutsideValue: 0

UpperThreshold: 30

Variance: [0.1 0.1]

Function Block Parameters: vtkImageMapToColors

S-Function (mask) (link)

Parameters

[✓] LookupTable as Input

ActiveComponent Parameter: Use Default

NumberOfThreads Parameter: Use Default

OutputFormat Parameter: As Parameter

OutputFormat Value: 0

PassAlphaToOutput Parameter: Use Default

[ ] Self as Output

Figure 5: ITK and VTK parameter dialog boxes. In the VTK dialog box (right), the checkboxes and selection boxes indicate parameters that are to appear as inputs or outputs of the block.

2.3 Code Generation

For both VTK and ITK, we have developed a straightforward XML description of the classes that provides the following necessary information:

- the class name
- inputs and outputs of the class
- image data types and dimensionalities that are supported (ITK only)
- names of all parameters
- the parameter size, for vector and matrix parameters
- the parameter type, and if it is a typedef, the C equivalent

These XML files are processed by a perl script which generates the following: 1) the C/C++ source code for the Simulink S-function that will call VTK or ITK from the blocks, and 2) the Simulink library files that describe a suitable graphical user interface for each block. For the ITK filters, several blocks are generated from each XML description since ITK objects are templated over data type and dimensionality. Certain special VTK and ITK classes such as vtkITKImageImport, itkImageFileReader/Writer and itkLevelSetNode do not utilize XML descriptions and are instead generated directly by the perl script. The entire process of generating the code and compiling the libraries is driven by CMake [9]. Our ultimate goal is to generate the libraries from the information in the ITK and VTK C++ header files, with the XML descriptions as only an intermediate step. For ITK, we have investigated WrapITK [8] and intend to filter its gccxml output to create our own XML. For VTK, we have written an XML generator based on the parser for VTK-Python, and we have fully automated SimVTK generation with a CMake build script.

3 Results

We have built and tested our libraries under MATLAB versions 2007a and 2008a, with Visual Studio 2005 as the compiler on Windows XP, and gcc 3.4 as the compiler on Linux and Linux64 (Ubuntu 6.06 and 8.04). ITK 3.6 and VTK 5.2 were used, as these were the latest versions available at the time.

The SimVTK model shown in Figure 3 demonstrates a simple VTK pipeline that includes a Simulink vector input (in this case, a Sine source) that is used for animation, and a vtkRenderWindowInteractor block that allows user interaction with the 3D scene. Such a pipeline is an example of how SimVTK could be utilized for image-guided surgery, given a Simulink block that interfaces to a surgical tracking system to provide position and rotation information. We have also built pipelines that combine ITK and VTK blocks, for instance Figure 4 demonstrates the use of ITK to read and process an image that is then sent to VTK for display. Note that the SimITK blocks are connected by two lines, one for the image information (the origin and spacing) and another for a MATLAB array that holds the pixel data values. The SimVTK blocks are connected by a single line which represents the transfer of a vtkObject pointer from one block to the next. Representative dialog boxes for SimITK and SimVTK are shown in Figure 5.

4 Discussion and Conclusion

We have presented a practical approach to visual programming with ITK and VTK, and anticipate that it will be suitable for a range of image computing applications. The advantage of our visual programming approach to ITK and VTK, as compared to scripting languages or straight C++ applications, are short development times and high maintainability due to the existence of SimITK/VTK models as, essentially, block diagrams that express the connections between components. Furthermore, if a pure C++ application was necessary, it would be possible to write a utility which could read a SimITK/VTK model file and automatically produce the C++ code that will connect ITK and VTK objects in a way that mirrors that model.

Our particular interest is image-guided therapy. To support this, we will create SimVTK blocks from our VTK code for surgical tracking devices and ultrasound video acquisition, and will directly use classes derived from ITK and VTK for the necessary calibration, image analysis, and visualization components of our image guidance system. Furthermore, since Slicer uses a VTK class as its plug-in module interface, a SimVTK block can be created that links Simulink to Slicer. This will allow us to use Slicer as the front-end for SimITK/VTK image-guided therapy applications.

In the future, we will fully automate the generation of Simulink blocks from ITK header files, similar to what we achieved for VTK. There are also several ITK classes which require special considerations in order to be usefully represented by Simulink blocks, and we will continue to improve support for these classes.

5 Acknowledgements

The authors wish to thank the Natural Sciences and Engineering Research Council (NSERC), the Canadian Institutes of Health Research (CIHR), and the National Alliance in Medical Image Computing (NIH 5U54EB005149-03) for funding this project.

References

[1] S. Aylward, J. Jomier, S. Barre, B. Davis, and L. Ibanez. Optimizing ITK's registration methods for multi-processor, shared-memory systems. *Insight Journal (www.insight-journal.org)*, paper 172, 2007. 2.1

[2] I. Bitter, R. Van Uitert, I. Wolf, L. Ibáñez, and J.-M. Kuhnigk. Comparison of four freely available frameworks for image processing and visualization that use ITK. *IEEE Trans. Vis. Comp. Graph.*, 13:483–493, 2007. 1

[3] V. Chu and G. Hamarneh. MATLAB-ITK interface for medical image filtering, segmentation, and registration. In *Medical Imaging 2006: Image Processing*, Proc. SPIE, 6144:3T, 2006. 1

[4] K. Gary, L. Ibáñez, S. Aylward, D. Gobbi, M.B. Blake, and K. Cleary. IGSTK: An open source software toolkit for image-guided surgery. *IEEE Computer*, 39:46–53, 2006. 1

[5] N. Hata, S. Piper, F.A. Jolesz, C.M.C. Tempany, P. Black, S. Morikawa, H. Iseki, M. Hashizume, and R. Kikinis. Application of open source image guided therapy software in MR-guided therapies. In *MICCAI 2007*, pages 491–498, 2007. 1

[6] M. Koenig, W. Spindler, J. Rexilius, J. Jomier, F. Link, and H.-O. Peitgen. Embedding VTK and ITK into a visual programming and rapid prototyping platform. In *Medical Imaging 2006: Image-Guided Procedures and Display*, Proc. SPIE, 6141:20, 2006. 1

[7] H.E.K. Le, R. Li, and S. Ourselin. Towards a visual programming environment based on ITK for medical image analysis. In *Digital Image Computing: Techniques and Applications (DICTA)*, pages 558–565, 2005. 1

[8] G. Lehmann, Z. Pincus, and B. Regrain. WrapITK: Enhanced languages support for the Insight Toolkit. *Insight Journal (www.insight-journal.org). January–June*, paper 85, 2006. 1, 2.3

[9] K. Martin and B. Hoffman. *Mastering CMake*. Kitware, Inc., Clifton Park, New York, 2006. 2.3

[10] S.G. Parker and C.R. Johnson. SCIRun: A scientific programming environment for computational steering. Proc. Supercomputing, 1995. 1

[11] XIP: Extensible Imaging Platform, https://collab01a.scr.siemens.com/xipwiki/. 1

[12] P.A. Yushkevich, J. Piven, H. Cody Hazlett, R. Gimpel Smith, S. Ho, J.C. Gee, and G. Gerig. User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability. *Neuroimage*, 31(3):1116–1128, 2006. 1