SOCIETY FOR SCIENCE AND EDUCATION UNITED KINGDOM

JBEMi JOURNAL OF BIOMEDICAL ENGINEERING AND MEDICAL IMAGING

ISSN: 2055-1266
VOLUME 3 ISSUE 6

Advantages and Disadvantages of using Third-party software in the development of the CAS_Annotate and CAS_Navigate Medical Applications

[1]João Fradinho Oliveira
[1]C3i/Instituto Politécnico de Portalegre, Portalegre, Portugal;
jfoliveira@estgp.pt

ABSTRACT

This paper address the main design decision issues taken when using third party libraries in the creation of two medical applications [1] that specifically require editing or creating geometry from CT images (CAS_Annotate) and interactive 3D visualization (CAS_Navigate). Whilst the purpose of the first application was to research different 3D reconstruction algorithms, the second application was created to research different visual metaphors and the reconstructions themselves. This paper weights aspects such as the learning curve time versus coding in-house time, robustness and possible customization. In theory both applications could have been developed within the same IGTSK [2] framework, but the available project time and the development of different phases of the project made that impossible, instead a black box approach of using IGSTK's 3D Msh format was crucial to import algorithm results tested with a simple GLUT application, thus allowing development to be made in parallel.

Keywords: Image guided surgery; 3D reconstruction; IGSTK.

9 Introduction

Writing applications with third party software has always had many benefits such as access to functonality that would be prohibitive to implement in the time frame of a project, but also the known drawbacks regarding documentation and indeed the learning curve to be able to master and change those solutions at the required level for specific project needs. This paper outlines the requirements of two medical applications [1] CAS_Annotate (a tool that allows one to manually segment/edit contours of objects of interest in CT images and test-bed different 3D reconstruction algorithms) and CAS_Navigate (a tool that provides a road-map of pre-operative geometry of organs and vascular structures intraoperatively). The advantages and disadvantages found when using third party software at key implementation decisions in the project are discussed. This article is organized as follows. In the next section the main requirements are listed and discussed. In section 3 the system overview highlighting the various chosen third party toolkits and high-level processes is presented. Finally results are presented in section 4, and conclusions are presented in section 5.

DOI: 10.14738/jbemi.36.2596
Publication Date: 27th December 2016
URL: http://dx.doi.org/10.14738/jbemi.36.2596

10 Background

In this section we outline and discuss the main requirements for both applications (see Table 1), before listing third part software potential functionality (see Table 2).

Table 1. Main functional and non-functional requirements

#	Requirements	CAS_Annotate	CAS_Navigate
1	Read/Display DICOM	X	X
2	Spline Read/Write/editing	X	
3	Read/Display 3D models		X
4	Surgical Trackers		X
5	C++	X	X
6	Graphical User Interface	X	X
7	Fast to render	X	X
8	Reliability	X	X

The goal of CAS_Annotate was to create different reconstruction algorithms to supply 3D models that could be tested effectively with different visual metaphors in CAS_Navigate. To achieve this, basic DICOM reading and display of CT images would be required together with the ability to store and edit multiple spline contours per slice for geometry extraction. On the other hand CAS_Navigate required the use of surgical trackers, and since it was to be used intraoperatively stability/reliability and rendering performance was paramount.

For reading and displaying DICOM images several licensed or free third party software are available, such as VTK [3], ITK [4], MITK [5](medical imaging interaction toolkit, whose framework builds on VTK and ITK) and IGSTK [2, 6]. For editing splines over a DICOM CT image, the VTK distribution includes the TestImageActorContourWidget example project, that uses a vtkContourWidget coupled with a vtkSliderWidget which allows to change the slice being viewed with vtkImageViewer2, Figure 1 shows all three in action in a preliminary version of CAS_Annotate.

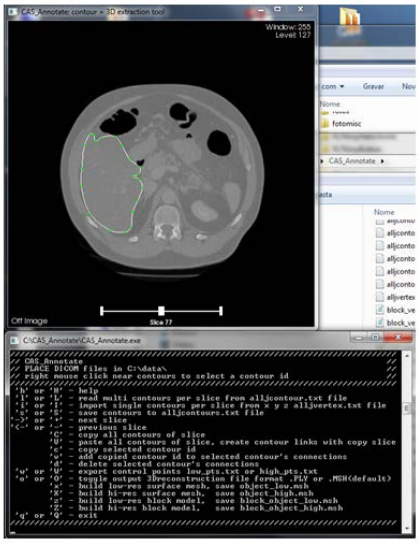

Figure 1. CAS_Annotate preliminary version with modified vtkImageViewer2, vtkContourWidget and vtkSliderWidget.

In addition, the IGSTK toolkit has several other advantages: it supports a wide range of trackers (AXIOS3D, ArucoTracker, Ascension3DGTracker, AtracsysEasyTrack500, InfiniTrack, MicronTracker, NDICertusTracker), it is built with a state machine design which is important for reliability, and finally it allows reading of the .Msh 3D file format.

C++ support was an important requirement for two reasons: the first reason was to enable the use efficient standard template library datastructures to hold the control points of several vtkSliderWidgets, thus enabling write/read for future spline editing, and the second reason was to facilitate the development of the different robust 3D reconstruction approaches within CAS_Annotate (please refer to [1] for specific algorithm details).

Graphical user interface menu support is available through Qt [7] or FLTK [8] in all the third party software listed in Table II, in addition mouse and keyboard callbacks can be customized.

Table 2. Third party software potential functionality

#	Functionality	VTK	ITK	IGSTK
1	Read/Display DICOM	X	X	X
2	Spline Read/Write/editing	X		
3	Read/Display 3D models	X	X	X
4	Surgical Trackers			X
5	C++	X	X	X
6	Graphical User Interface	X	X	X
7	Fast to render	X	X	X
8	Reliability	X	X	X

11 System Architecture

The following diagram (Figure 2) shows the Third-party toolkit relation with the two applications that were developed, with the addition of a third interim OpenGL 3D viewer used for fast debugging of the reconstruction algorithms used in CAS_Annotate.

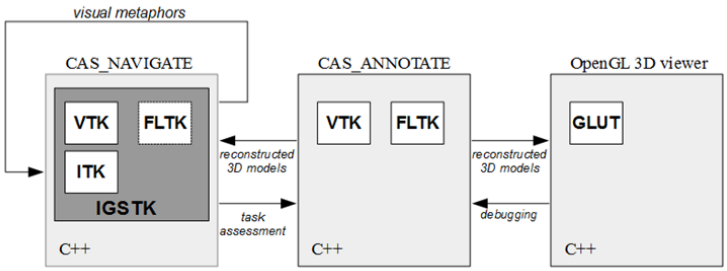

Figure 2. Toolkit and application overview diagram.

As mentioned in the previous section, VTK contained an example named TestImageActorContourWidget that fullfiled most of the requirements of CAS_Annotate, except for a problem on the access of the correct DICOM slice and the correct 3D coordinates of the mouse pick. On one hand VTK [9] offers plenty of examples in the installation, that enable one to quickly grasp the way it deals with geometric entities (actors), geometry (vtkPolyData, vtkPoints) and how they are indexed (vtkCellArray) and accessed (vtkPolyDataMappers), or read (vtkPLYReader), and how to create a camera (vtkCamera) and

add it to a rendering window (vtkRender) with mouse interaction support (vtkRenderWindowInteractor), but on the other hand this rather useful, perhaps too specialized example is regarded in developer pages as having a "broken" vtkImageViewer2. Eventually a solution was created [1] that accesses the correct slices using the DICOM direction cosines [10] from reader->GetImageOrientationPatient(), hides the incorrect text label and computes the 3D coordinates taking into account the inversion of imageViewer->getImageActor()->GetDisplayBounds(). Multiple vtkContourWidgets per slice are readily stored in C++ container classes, keyboard callbacks are defined to call the different reconstruction algorithms and functionality.

One negligible downside of CAS_Annotate, is the temporary memory duplication for holding extracted geometry from the contours, when passing them to the reconstruction algorithms before saving the models to file. Writing the algorithms directly on custom "in-house" datastructures proved to be time-saving in terms of development time. Similarly the saved exported models from CAS_Annotate were read and rendered in a third simple OpenGL viewer application for fast debugging (Figure 3 and Figure 4).

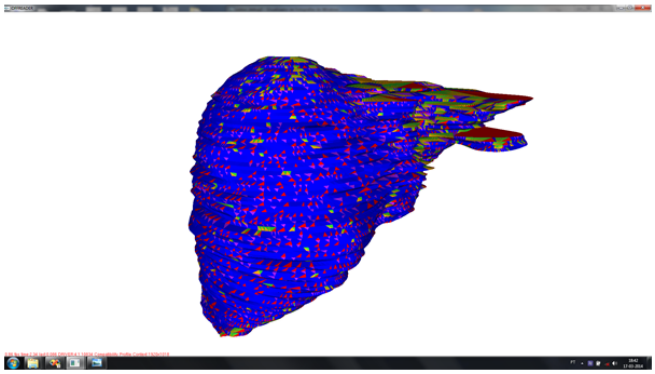

Figure 3. GLUT/OpenGL debugging of surface reconstruction algorithm result (liver).

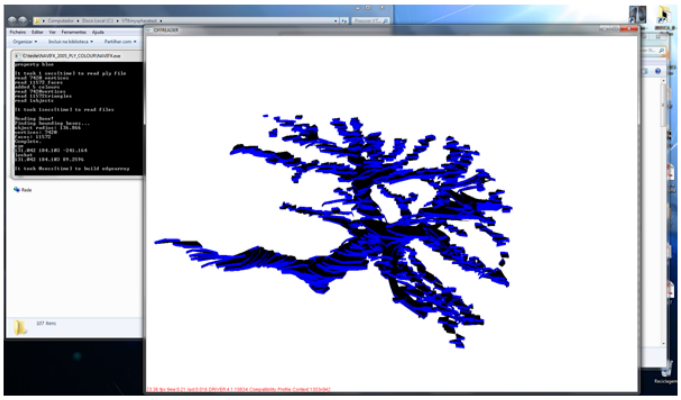

Figure 4. GLUT/OpenGL debugging of block reconstruction algorithm result (vascular network).

Finally, once algorithms were checked for correct operation, the 3D models were read into IGSTK's modifed Navigator example. Learning how to modify the rendering options (visual metaphors) and callbacks was fast, but on the other hand documentation for the Msh file layout was not readily

available (trial and error) and learning how to sucessfully compile the required libraries was not trivial (the reader is refered to the Appendix section).

12 Results

Figure 5 and Figure 6 show CAS_Annotate and CAS_Navigate applications respectively. Third party toolkits VTK, ITK, FLTK and IGSTK were essential to be able to complete this project. The project code will soon be placed in a Google project.

Figure 5. CAS_Annotate using VTK's DICOM reading and rendering capability together with the modified vtkSliderWidget and vtkContourWidget to annotate vascular contours. User commands and menus were created with FLTK

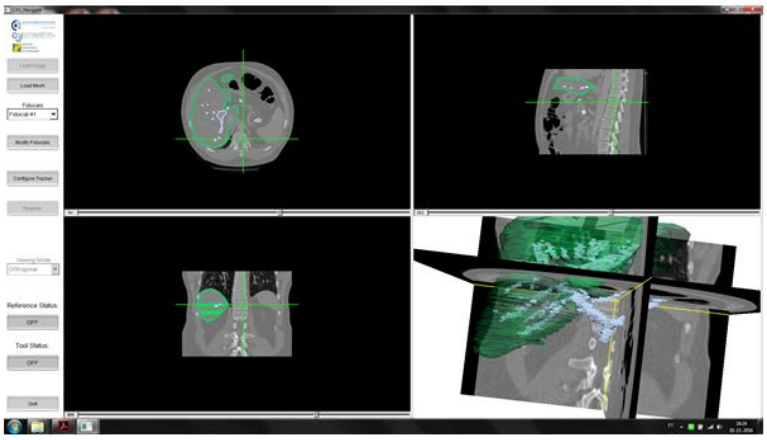

Figure 6. CAS_Navigate with modified IGSTK Navigator example showing a surface based reconstuction method (liver in green) combined with a block based reconstruction (vascular network in grey) being rendered with different oppacity levels over CT images.

13 Conclusions

In theory, CAS_Annotate and the OpenGL 3D viewer used for algorithm debugging could have been part of the CAS_Navigate application, since IGSTK uses the VTK library and has access to it's rendering capabilities, however the time available for the project and the associated learning curve mandated a

black box approach of reading 3D model results through the Msh format, separate contour annotation, and separate interim 3D visualization for rapid algorithm debugging.

ACKNOWLEDGMENT

This project was funded by Programa de Cooperación Transfronteriza España-Portugal (POCTEP), Fondo Europeo de Desarrollo Regional (FEDER) 0401_RITECA_II_4_E.

REFERENCES

[1] Oliveira, J. F. , J. L. Moyano-Cuevas, J. Blas, H. Capote, and F. M. S. Margallo, *Preoperative and Intraoperative Spatial Reasoning Support with 3D Organ and Vascular Models: Derived from CT Data using VTK and IGSTK*, International Journal of Creative Interfaces and Computer Graphics, vol. 6(2), pp. 56–82, July-December, 2015.

[2] The Image-Guided Surgery Toolkit, http://www.igstk.org, 2016.

[3] The Visualization Toolkit, http://www.vtk.org, 2016.

[4] Insight Segmentation and Registration Toolkit (ITK), https://itk.org, 2016.

[5] The Medical Imaging Interaction Toolkit (MITK), http://mitk.org/wiki/MITK, 2016.

[6] Gary, K. , M. B. Blake, S. R. Aylward, J. Jomier, D. Gobbi, H. Kim, R. Avila, L. Ibanez and Kevin Cleary, *IGSTK: Development Process and Project Management Best Practices for an Open Source Software Toolkit for Image-Guided Surgery Applications*, in MICCAI Open-Source Workshop, 2005.

[7] Qt, https://www.qt.io/, 2016.

[8] The Fast Light Toolkit (FLTK), www.fltk.org, 2016.

[9] Schroeder, W. J. , K. Martin, and B. Lorensen, *The Visualization Toolkit*, 4th edition, Kitware, 2006.

[10] Pianykh, O. S., *Digital Imaging and Communications in Medicine (DICOM) A Practical Introduction and Survival Guide*, 2nd Edition, Springer, ISBN 978-3-642-10849-5, 2012.

[11] Git, https://git-scm.com/, 2016.

[12] CMake, https://cmake.org/, 2016.

[13] Schroeder, W. J. and L. Ibanez, *Software process: the key to developing robust, reusable and maintanable open-source software*, DOI: 10.1109/ISBI.2004.1398621, IEEE Xplore, May 2004.

[14] The Image-Guided Surgery Toolkit, *How to build IGSTK*, https://public.kitware.com/IGSTKWIKI/index.php/How_to_build_IGSTK, 2016

[15] The Image-Guided Surgery Toolkit, *Download IGSTK*, https://public.kitware.com/IGSTKWIKI/index.php/Download_IGSTK#IGSTK_5.0_Requirements, 2016.

[16] Cleary, K. and Insight Software Consortium, *IGSTK: The Book*, p. 13, 2007.

APPENDIX – LIBRARY COMPILATION GUIDE

During the development of CAS_Annotate and CAS_Navigate the following aspects were found that could induce compilation errors of the libraries:

a) Compiler and compiler version

b) git (a free and open source distributed version control system) [11] "clone" or web download of the library

c) CMake [12, 13] version

d) library version

e) CMake project options

The first aspect, aspect a) occurs because different compilers and indeed different compiler versions have different assumptions and defaults. For example Microsoft Visual Studio Professional 2008 was used with no problem, but Microsoft Visual Studio Professional 2013 will give an error when compiling the smart pointer definition in VTK.

Aspect b) is quite unfortunate, as the only way found for VTK, ITK or IGSTK to compile correctly was by using git cloning rather than http download. On the other hand Git, CMake and FLTK can be downloaded via http without problem.

Aspect c) the latest version of CMake (3.7.1) does not find the VTK_bin and ITK_bin library directories when configuring IGSTK. When manually locating the directories and files, an include error related to the same libraries remains. On the other hand CMake 2.8.11.2 finds the same libraries without assistance or problem.

Aspect d); as stated by the IGSTK build guide, the use of the appropriate version of toolkit libraries is "critical" [14].

IGSTK5.2 requires[15] the following library versions:

- VTK 5.10.0 or later
- ITK 4.2.0 or later
- FLTK 1.1 or Qt 4.0 (or later)
- CMake 2.8 or later

This project used:

- git 2.11.0 (but other versions "should" work)
- CMake 2.8.11.2
- FLTK1.1.10 (FLTK-1.1.10-source.zip)
- VTK5.10.0
- ITK 4.4.1

Regarding aspect e), please refer to the CMake options (Figure 7, 8, 9, 10). However it should be noted that, although the build guide refers that the VTK and ITK should be built both as shared libraries if one wishes IGSTK to have shared libraries [16], and both as static otherwise, it was not possible with the

settings above to build the shared library version, in this project the shared library CMake option was turned off. In addition to avoid errors in CMake, the IGSTK CMake option for testing was set to off. Perl does not need to be installed, and only an informative message is shown in CMake.

Finally, the following steps should be performed:

1) create a FLTK_bin, VTK_bin, ITK_bin, and IGSTK_bin directories. This is where CMake will later copy the generated Visual Studio project files.
2) download git 2.11.0, and CMake 2.8.11.2 and fltk-1.1.10-source.zip
3) run CMake, set the browse source directories and browse build directories of FLTK (Figure 7)
4) press configure, it is critical that one presses the configure option again if checkboxs are set or unset, so that the project files can reflect the changes.
5) press generate and choose the target compiler
6) use the target compiler to open the FLTK project file inside FLTK_bin, choose Release, and build.
7) run the git-bash console, type “cd ..” until the /c directory is reached
8) type git clone git://vtk.org/VTK.git
9) still in the git-bash console, type cd VTK
10) type git tag -l (will provide the precise name of all versions of the library that was cloned)
11) type git checkout -b v5.10.0 v5.10.0
12) perform steps 3 to 6 following the respective Figure settings (Figure 8-10)
13) perform steps 7 to 12 with git clone git://itk.org/ITK.git with step 11 with -b v4.4.1 v4.4.1
14) perform steps 7 to 12 with git clone git://igstk.org/IGSTK.git with step 11 with -b v5.2 v5.2

Configuring, generating the project files and compiling FLTK, VTK, ITK and IGSTK with the settings of Figure 7, 8, 9, and 10 took in total 29 minutes with an i5 3470, 3.2 Ghz processor.

By default the generated INSTALL projects will not be built when the solution is compiled, however, after compilation, one can right click on the INSTALL project, and choose Project Only, Build Only INSTALL. Install will copy the library and header files of separate modules and directories into a common include or lib directory inside the Windows C:\Programs folder, this is especially useful when developing new projects, as only one folder needs to be set, rather than having to set each individual module folder.

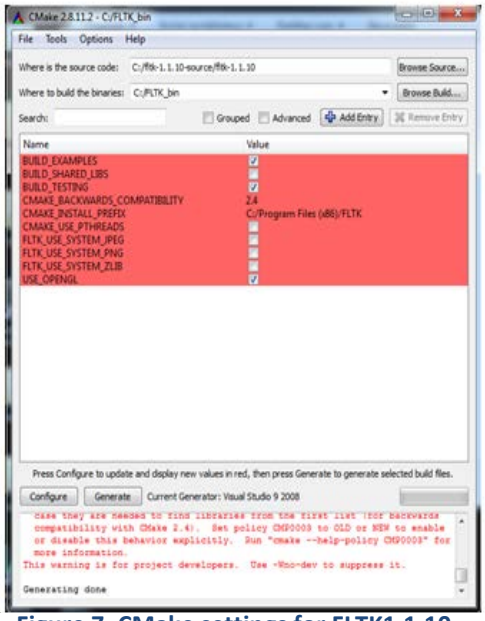

Figure 7. CMake settings for FLTK1.1.10

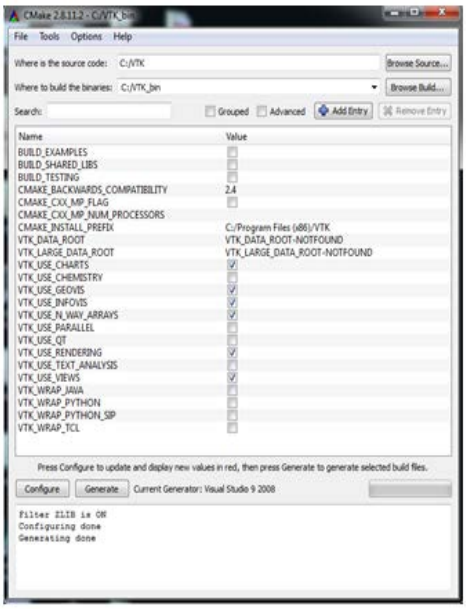

Figure 8. CMake settings for VTK5.10

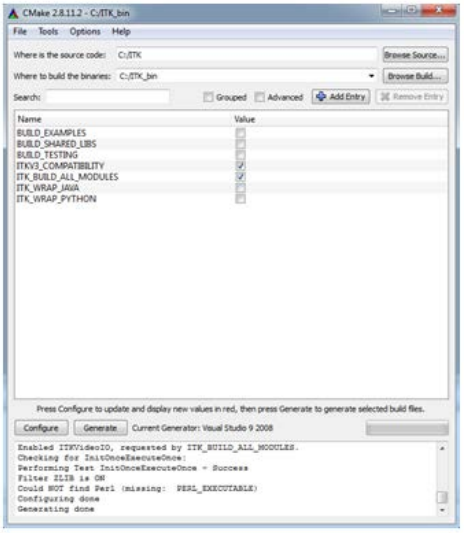

Figure 9. CMake settings for ITK4.4.1.

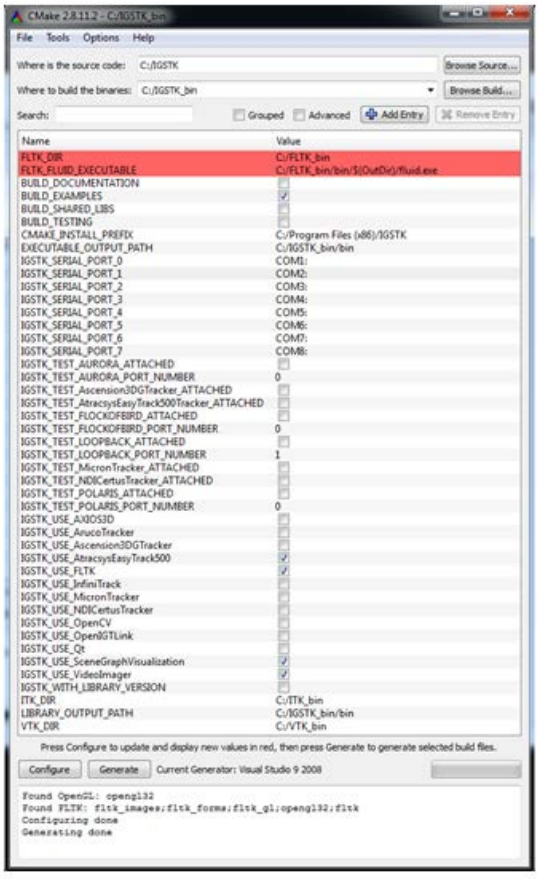

Figure 10. CMake settings for IGSTK5.2.