A lightweight rapid application development framework for biomedical image analysis

Shekhar S. Chandra, Jason A. Dowling, Craig Engstrom, Ying Xia, Anthony Paproki, Aleš Neubert, David Rivest-Hénault, Olivier Salvado, Stuart Crozier, Jurgen Fripp

PII: S0169-2607(16)30578-8
DOI: 10.1016/j.cmpb.2018.07.011
Reference: COMM 4757

To appear in: *Computer Methods and Programs in Biomedicine*

Received date: 2 June 2016
Revised date: 11 July 2018
Accepted date: 24 July 2018

Please cite this article as: Shekhar S. Chandra, Jason A. Dowling, Craig Engstrom, Ying Xia, Anthony Paproki, Aleš Neubert, David Rivest-Hénault, Olivier Salvado, Stuart Crozier, Jurgen Fripp, A lightweight rapid application development framework for biomedical image analysis, *Computer Methods and Programs in Biomedicine* (2018), doi: 10.1016/j.cmpb.2018.07.011

Highlights

- The work presents a new biomedical image analysis and visualization framework for multiple platforms with an open-source license suitable for commercial use.
- This Simple Medical Imaging Library Interface (SMILI) is designed for creating easy-to-use applications for clinical end-users ideal for deploying biomedical image analysis algorithms to clinical partners for evaluation.
- This submission is also part of an open-source release of the above library at SourceForge (http://smili-project.sourceforge.net/) and GitHub (https://github.com/shakes76/smili).
- The library is compared to other state-of-the-art biomedical image analysis software packages and validated for clinical usability in the context of measuring distances and angles for bone deformities of the hip joint.

A lightweight rapid application development framework for biomedical image analysis

Shekhar S. Chandra[c,*], Jason A. Dowling[a], Craig Engstrom[b], Ying Xia[a], Anthony Paproki[a,c], Aleš Neubert[a], David Rivest-Hénault[a], Olivier Salvado[a], Stuart Crozier[c], Jurgen Fripp[a]

[a]*Australian e-Health Research Centre, CSIRO, Australia.*
[b]*School of Human Movement Studies, The University of Queensland, Australia.*
[c]*School of Information Technology and Electrical Engineering, The University of Queensland, Australia.*

Abstract

Biomedical imaging analysis typically comprises a variety of complex tasks requiring sophisticated algorithms and visualising high dimensional data. The successful integration and deployment of the enabling software to clinical (research) partners, for rigorous evaluation and testing, is a crucial step to facilitate adoption of research innovations within medical settings. In this paper, we introduce the Simple Medical Imaging Library Interface (SMILI), an object oriented open-source framework with a compact suite of objects geared for rapid biomedical imaging (cross-platform) application development and deployment. SMILI supports the development of both command-line (shell and Python scripting) and graphical applications utilising the same set of processing algorithms. It provides a substantial subset of features when compared to more complex packages, yet it is small enough to ship with clinical applications with limited overhead and has a license suitable for commercial use. After describing where SMILI fits within the existing biomedical imaging software ecosystem, by comparing it to other state-of-the-art offerings, we demonstrate its capabilities in creating a clinical application for manual measurement of cam-type lesions of the femoral head-neck region for the investigation of femoro-acetabular impingement (FAI) from three dimensional (3D) magnetic resonance (MR) images of the hip. This application for the investigation of FAI proved to be convenient for radiological analyses and resulted in high intra (ICC=0.97) and inter-observer (ICC=0.95) reliabilities for measurement of α-angles of the femoral head-neck region. We believe that SMILI is particularly well suited for prototyping biomedical imaging applications requiring user interaction and/or visualisation of 3D mesh, scalar, vector or tensor data.

1. Introduction

The eventual success of algorithms within the field of biomedical image analysis and visualisation depends on their widespread adoption within clinical research and practice. Although validation of these algorithms on large datasets is essential, this alone is generally insufficient (for large scale implementation) because of the challenges associated with clinical scenarios, such as limited or variable operator training levels, time pressures and large variations of anatomical and pathological features across patients. Thus, applications deploying a package of such algorithms to clinical research and practice environments usually require very specific user interface design and/or have unique processing requirements that are not easily generalisable, resulting in complicated software designs. Moreover, analysis of biomedical images tends to be a complex task, consisting of sophisticated algorithms, whose re-implementation can be time consuming if the source code is not available or not commercially viable due to restrictive software licensing.

In this work, we present a lightweight biomedical imaging analysis and visualisation library called Simple Medical Imaging Library Interface (SMILI)[1], which has a liberal open-source license suitable for commercial use. With a compact suite of core objects in the library, biomedical imaging developers can harness the features listed in table 1, which demonstrates SMILIs support of a substantial subset of features when compared to a representative sample of other leading contemporary and commercial biomedical image analysis software. By light-weight, we mean that SMILI has a high ratio between features available to the number of class objects required compared to the state-of-the-art. It is a very high-level biomedical imaging interface with a minimal footprint (in terms of memory and computation overhead) and small number of objects with a simple structure (see Classes Defined in table 1 for example), while still making sophisticated features and algorithms more readily accessible.

SMILI achieves compactness in terms of number of classes by utilising a data driven object oriented design mirrored at two layers. The first layer is geared to be deployed via command line applications and scripts by providing image processing features. It is a high level (non-Graphical User Interface (GUI)) Application Programming Interface (API) wrapping to the popular Insight Toolkit (ITK) [1] and the Visualisation Toolkit (VTK) [2] libraries. The second layer provides very high level pre-built GUI objects via the open-source version of the Qt software framework (qt-project.org). This layer is geared for GUI application development with pre-built components that can be extended upon to rapidly create custom research oriented (cross-platform) applications suitable for clinical end-users. This overall design facilitates readable code, high level

*Principal corresponding author
Email address: Shekhar.Chandra@uq.edu.au (Shekhar S. Chandra)
[1]http://smili-project.sourceforge.net/

Slicer	SMILI	ITK-SNAP	MITK	MeVisLab
Classes Defined (Total/Core): 1000+/500+	43/27	500+/300+	2000+/1000+	2000+/1000+
Licensing: Open-Source (BSD)	Open-Source (BSD)	Open-Source (GPL)	Open-Source (BSD)	Commercial
Installer Size: 170 MB	60 MB	20 MB	60 MB	1 GB
DICOM Reading/Convert/Anonymise	✓	✗	✓	✓
Image Processing	✓	✗	✓	✓
Image Measurement/Annotation	✓	✗	✓	✓
Interactive Segmentation	2D Image, Surfaces	3D Image	3D Image	3D Image
Scripting	Python	✗	Python	Python
Volume Rendering	✓	✗	✓	✓
Screen Capture Functionality	✓	✓	✓	✓
Rigid and Non-rigid Registration	✓	✗	✓	✓
Landmark Registration	✓	✗	✓	✓
Semi-Automatic Segmentation	✗	✓	✓	✓
4D Image Viewer	✓	✗	✓	✓
Neuro-Imaging (Diffusion etc.)	Diffusion, FODs	✗	✓	✓
Cardiac-Imaging	✗	✗	✓	✓
Shape Analysis/Visualisation	✓	✗	✗	✗
Plugins/Extensions	✓	✗	✓	✓
Plugins/Extensions AppStore	✗	✗	✗	✗
Auto Window Levelling	✓	✓	✓	✓
Flexible Layouts and Slice Viewers	Slice & Four Views only	Slice & Four Views only	Slice & Four Views only	✓

Table 1: Comparison of features between 3D Slicer, SMILI and other leading contemporary open-source or commercial software packages.

scripting and reduces pre-requisite knowledge requirements of image processing and computer graphics for biomedical imaging developers. Details of SMILI will be described in the design section of the paper (see section 2). We first review other work in the area, especially in relation to the image analysis (and visualisation) software packages presented in table 1.

1.1. Background

In recent years, a number of biomedical image analysis libraries have been developed to address many of the challenges associated with deploying algorithms for clinical research. One of the most complete is the open-source Medical Imaging Interaction Toolkit (MITK) [3], which provides a wrapping of ITK and VTK [1, 2] with the goal of extending both libraries for biomedical image analysis. The MITK library is mature, feature rich and is rapidly becoming a defacto standard for both libraries. It provides only limited support for deformable models however, a valuable tool in medical image analysis [4–7]. Instead, it consists of a large number of classes based on a similar design structure as these libraries and its use is dependent on prior knowledge and/or experience of the ITK, VTK libraries (such as filter names, pipelines etc.), as well as image processing and computer graphics principles.

A similar open-source project with a liberal open-source license is 3D Slicer [8], which delivers mature, feature complete (see table 1) and stable medical image processing. However, this comprehensive feature set results in a GUI that might be complex to grasp and learn for some, while also requiring knowledge of underlying libraries such as the ITK to fully utilise its capabilities as a library. Even though projects such as SimpleITK [9] have made efforts to provide a high level interface for ITK, they are implemented for use in interpreted languages only, such as Java and Python, are intended for users new to ITK and do possess singular easy-to-use objects within GUI applications.

Another promising software package is the commercially developed MeVisLab [10], which provides a large range of processing and visualisation algorithms in an easy-to-use manner via a visual programming interface. It also has tools, such as 'ToolRunner' to easily deploy custom applications for clinical partners once custom algorithms have been imported into MeVisLab. However, it is only available via a non-commercial license for non-commercial entities and the software development kit is of significant size (1000+ classes, 1 GB). It is also optimised for constructing imaging pipelines rather than for end users such as clinicians without investing significant development resources. That is, it is not built to directly provide an extensible GUI application, but rather is a powerful software development kit (SDK) that has a comprehensive set of tools necessary to build one if experienced in developing MeVisLab modules. For example, to visualise a medical image with a custom algorithm applied to it, one has to create an 'ImageLoad' module, connect it to an appropriate viewer module and a MeVisLab module developed for your custom algorithm, and then deploy the application using the 'ToolRunner' application. The user interface for this application might not be intuitive for use by clinical researchers unless specific efforts are made to ensure proper design.

An open-source alternative to MeVisLab [10] is the DeVIDE Python distribution. DeVIDE is a highly portable distribution (also featuring ITK and VTK) for rapid prototyping of medical imaging pipelines via a visual programming interface. This versatile distribution however, requires porting existing libraries to the Python programming language or providing Python wrappers of existing algorithms. The distribution provides no

wrapping of GUI libraries such as the Qt framework to aid in user interface design of any clinical application that may be required. Furthermore, the ability to rapidly prototype may not necessarily translate well to large projects and commercial or clinical deployment.

Further examples of image visualisation software include the parallel visualisation package called Paraview [11, 12]. It provides ways to visualise large datasets and polygonal surfaces with a liberal open-source license, although applying image processing algorithms to surfaces or to open complex medical image formats such as DICOMs are not immediately apparent. Visualising images as slices together with polygonal surfaces also requires knowledge of the underlying visualisation library and involves a series of steps that require knowledge of computer graphics. OsiriX [13, 14] provides a Mac OSX only open-source GUI environment optimised for picture archiving and communication systems (PACSs) and DICOM viewing and processing. ITK-SNAP [15] and MRIcro provide applications specialised for viewing, segmenting and contouring medical images across platforms. In addition to image viewing and contouring, ImageJ offers image processing, but has limited support for three dimensional (3D) data and models (such as processing and animation).

1.2. Overview

In this paper, we present SMILI, a freely available, lightweight library for (cross-platform) biomedical image analysis application development. An example of an imaging application constructed out of the main SMILI classes (one that also ships with the library) called Simple Medical Imaging Library X Viewer (sMILX) is shown in figure 1. The foundation of this sMILX application is the *milxQtMain* object, one of the core GUI classes in the compact suite of objects required for SMILI based rapid application development. These GUI classes have a very high level interface and provide essentially pre-built GUI components that can be extended upon to rapidly create custom research oriented (cross-platform) applications suitable for clinical end-users. A custom main window object can be inherited from the *milxQtMain* object to instantly create a fully featured application as shown in figure 2. This has been recently demonstrated by deploying a High Dynamic Range (HDR) algorithm for magnetic resonance (MR) images [16]. The cross-platform nature of SMILI allows the imaging data to be visualised and processed identically across Windows, Linux and Mac OSX operating systems. This very high level design also facilitates a naturally scriptable API currently available for Python via a plugin, which are also supported with SMILIs lightweight plugin interface. These GUI components and scripting are discussed in greater detail in section 2.2.

In the subsequent sections, we describe the dual layer (data driven) object oriented design of SMILI (see sections 2.1 and 2.2) and demonstrate its capabilities in common imaging tasks at the command-line and GUI applications built from its main classes (see section 4.1). We then compare SMILI to state-of-the-art biomedical image analysis software packages (see for example table 1) and discuss advantages and limitations of SMILI (see section 4.2). We then utilise these GUI components to show how clinical research applications can be developed rapidly, i.e. within 10 hours of programming and testing (given a biomedical imaging algorithm has already been developed), using SMILIs generality and extensibility (see section 3.3). We validate this custom application in a MR study of the hip focusing on manual intra- and inter-rater measurements related to femoro-acetabular impingement (FAI) (see sections 3.3 and 4.3).

2. Design

The overall design philosophy adopted for SMILI aims to maintain as few objects as possible, whilst having a data driven approach to the main high level objects that are to form the core of the library. These objects would then implement biomedical image processing as operations on data and act as pre-built components ready-to-use by biomedical imaging application developers in their research areas. The goal is to create a library that is lightweight in terms of the number of objects and is straight-forward to learn and grasp conceptually. The resulting code is made up of a series of high level operations, therefore easy to construct and in more human readable form.

For example, listing 1 shows example API of the GUI file and model objects for pre-processing surfaces as part of a larger medical imaging processing pipeline. Surface data is opened

Listing 1: Example of the *milxQt* API for the *milxQtModel* class.

```
QPointer<milxQtFile> reader = new milxQtFile;
QPointer<milxQtModel> model = new milxQtModel;
success = reader->openModel(filename, model);
model->clean();
model->smoothSinc(20); //windowed sinc 20 iterations
model->decimate(0.5); //50% decimation
model->generateModel(); //this is required before display
model->colourMapToJet(); //colourmap
model->viewToCoronal(); //view
model->show();
```

and then maintained in the GUI model class, namely the *milxQtModel* object, and operated on by its methods such as smoothing and decimation, as well as options for visualisation (such as the sub-windows in figure 1). This single class is the only object that is required to be learned and utilised for all aspects related to models. The steps within listing 1 are easy to follow and understood because of the high level nature of the interface making constructing solutions simpler. A similar class *milxQtImage* exists for images. Together with the already mentioned *milxQtMain* and the *milxQtFile* object, these classes are the only objects required to accomplish most of the major features compared to other state-of-the-art packages (see table 1). The remaining parts of this section discusses the relevant design areas that result in the API within the aforementioned listing. It is intended that the outcomes of such a philosophy results in the development of a library that is easily ported to existing projects, extensible to different use cases, yet maintaining a set of core features necessary for developing the most common types of biomedical image analysis applications.

To reflect the design philosophy, SMILI is designed to facilitate biomedical imaging research and development mirrored at two levels. Firstly, researchers often utilise different command-line applications to process specific part(s) of medical imaging

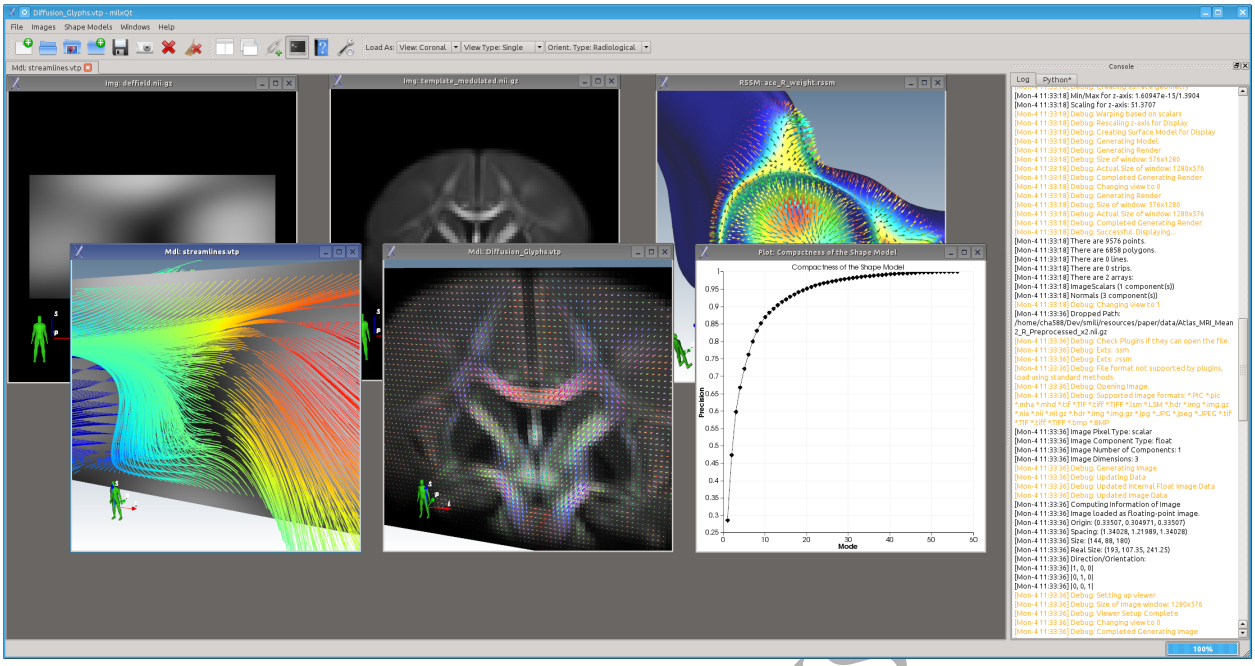


Figure 1: A screenshot of the sMILX GUI application for biomedical image analysis that can display assorted types of data in a variety of different representations, such as *n*-D images, surfaces/models, fields and 2D/3D plots. This screenshot shows (from left to right in pairs) vector image with subsequent streamlines, visualisation of orientation distribution functions (ODFs) for diffusion MRI and shape analysis with vector fields and plots.

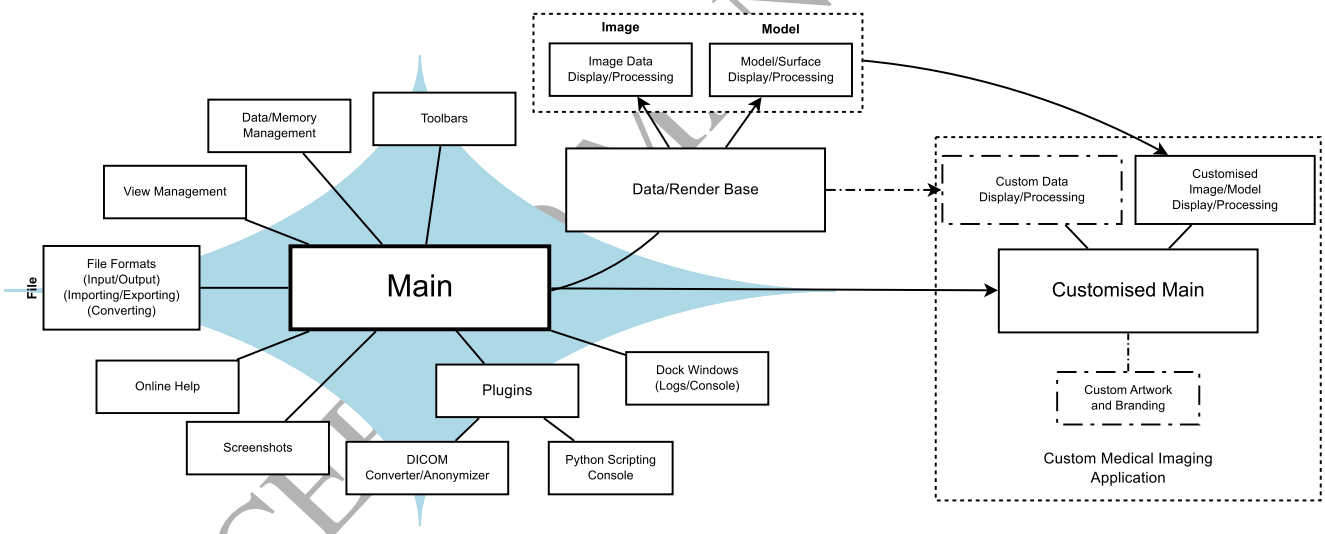


Figure 2: A schematic representation of SMILI and what it provides for medical imaging application development. The blue diamond region lists a selection of the main features available at the top-level in SMILI and the arrows indicate the small number of derivations (dashed lines for optional) required to generate a custom medical imaging application with all these functionalities.

data and script them together to create an imaging pipeline. The resulting pipeline is then run on cloud computing infrastructure with clinical researchers interacting with a web interface to process their data and with the purpose of obtaining clinically relevant finding(s) (see for example [17]). Secondly, the majority of biomedical imaging research requires interaction with clinical research, as well as feedback, such as making manual measurements or rapidly visualising results easily interpretable in clinical practice. The latter often requires deploying custom applications with GUI interfaces for clinical research sites to evaluate.

As a result, the overall design of SMILI consists of processing and GUI layers whose objects exist as duals at each other, namely the *milxSMILI* and *milxQt* sub-libraries respectively, to share common processing algorithms as summarised in figure 3. This design features three main objects, namely the *Image*, *Model* and *File* objects and their duals of the GUI layer, namely the *milxQtImage*, *milxQtModel* and *milxQtFile* objects, together with a main window object for applications (discussed in section 2.2).

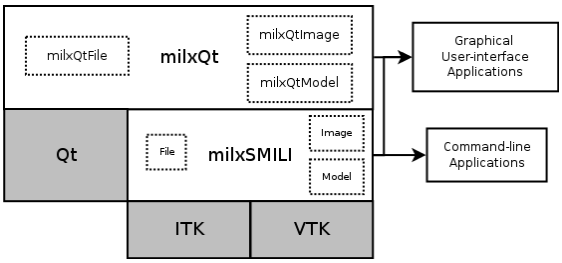


Figure 3: The internal design of SMILI (white blocks) to facilitate code reuse within both GUI and processing applications. The 'milx' designation is used by convention as a prefix for all classes. The *milxSMILI* layer is GUI independent and ideally suited for command-line applications. Both layers possess 'Swiss-army knife' like image and model classes capable of processing their data with a multitude of different algorithms. The *milxQt* layer allows for higher level use either via the API or in GUI applications.

2.1. Processing Layer

The processing layer of SMILI, called *milxSMILI*, features three main objects and an additional class for interaction between models and images. Firstly, the *File* class wraps both ITK and VTK to provide a unified interface for supporting a variety of different file formats for images and polygonal/model data. This is particularly useful for VTK, which does not have a single mechanism to load various formats for polygonal models, such as the Standford polygonal (PLY), Wavefront (OBJ), VTK polygonal (XML and legacy) and Stereolithography (STL) data files, the way ITK does for image formats using object factories.

Secondly, the *Image* object of *milxSMILI* is a single templated class designed to be used with the itk::Image class by both experienced and new users to ITK with over fifty supported image processing algorithms. For example, to open via the *File* and then threshold a labelled image and produce a signed distance-map, one would just issue the code in listing 2 (where template arguments and parameters are omitted for brevity). Members

Listing 2: *milxSMILI Image* class usage example.

```
itk::SmartPointer<> labelImage; //smart deletion
itk::SmartPointer<> threshImage; //smart deletion
itk::SmartPointer<> distMap; //smart deletion

success = milx::File::OpenImage<>(labelName, labelImage);

threshImage = milx::Image<>::BinaryThresholdImage<>(...);

distMap = milx::Image<>::DistanceMap<>(...);
```

of *Image*, such as 'DistanceMap' in listing 2, reduce the elements of ITK directly required by wrapping different algorithms available in a single interface.

Lastly, the *Model* class has a similar design to the *Image* object, but also maintains the current and previous states of the model during processing and does not require templates because it encapsulates a vtkPolyData object from VTK and its operations. Listing 3 shows how the model objects can be used in just several lines of code to clip, voxelise, convert and compute a distance map of a surface inside the given image. Listing 3 also shows how the clip member is very useful and

Listing 3: *milxSMILI Model* class usage example.

```
vtkSmartPointer<vtkPolyDataCollection> collection;
//Can use File::OpenModel() here too if opening a single model
milx::File::OpenModelCollection(filenames, collection);

vtkPolyData *surface = collection->GetNextItem();

milx::DeformableModel model(surface);
model.RemoveScalars(); //avoid causing problems with
//meshes having scalars already
//set weights as scalars from image
model.MarkSurfaceInsideImage<>(model.GetOutput(), ...);
//MarkSurface sets inside parts of surface as 1.0
model.Clip(1.0, 1.0);

//Voxelise surface
vtkSmartPointer<vtkImageData> voxelModel = model.Voxelise(...);
LabelImageType::Pointer modelLabel =
milx::Image<>::ConvertVTKImageToITKImage(voxelModel);
//We convert to ITK image to use ITK distance map algorithms

itk::SmartPointer<> modelDistanceMap =
milx::Image<>::DistanceMap<>(modelLabel, ...);
```

straight-forward to use, but is actually non-trivial to achieve in VTK unless one has in-depth knowledge of the library[2].

In addition to the processing layer classes, the capabilities of these objects are available via two command-line 'Swiss-army knife' like applications to apply operations to multiple medical images, namely the *milxImageApp* and *milxModelApp* applications. Examples of their use include those in the following listings:

Listing 4: Multiple Otsu thresholding (four levels) of a set of Nifti images.

```
milxImageApp --Otsu 128 *.nii.gz --labels 4 -p Otsu_128bins_
```

Listing 5: Concatenating all surfaces in the current directory.

```
milxModelApp *.vtk --cat -o initial.vtk
```

By ensuring that the command-line applications only depend on non-GUI medical imaging libraries, these applications are ideal for scripted processing pipelines and/or for deployment to the cloud. Here they can be run on computing clusters accessed by much less powerful machines via a web-interface or as part of a cloud infrastructure, such as the Galaxy cloud platform [18]. SMILI already (intrinsically) supports Central Processing Unit (CPU) multi-threading algorithms via implementations in ITK. As such, SMILI will utilise as many threads as specified via command-line arguments, usually half the number of available threads by default. Similar porting of ITK related command-line applications was recently demonstrated by Dowling *et al.* [17] for this cloud platform.

In SMILI, an additional layer is built upon *milxSMILI* that incorporates GUI capability. This GUI layer is discussed in the next section and consists of objects that are duals of the processing layer, while ensuring the library is structured clearly, making incorporating SMILI into user projects straight-forward.

[2]In this case, one has to utilise the vtkThreshold object and pipe the result into a vtkGeometryFilter object in order to obtain a clipped mesh.

2.2. GUI Layer

The *milxQt* sub-library provides a higher level GUI layer in SMILI by incorporating the Qt software framework (qt-project.org) into the image and model classes for visualisation and processing of their respective data. The primary objects in *milxQt* mirror those of *milxSMILI*, providing *milxQtImage*, *milxQtModel* and *milxQtFile* objects, augmented with two additional classes engineered for creating applications.

In particular, the *milxQtImage* object instantiates standard image pixel types (8-bit, RGB and float pixel types) using the *Image* object, while having additional support for vector images allowing loading and visualisation of *n*-D imaging data through vector image representation of the data. The *milxQtModel* wraps the *Model* object, while incorporating interactions with *milxQtImage* and other *milxQtModel* objects allowing overlaying of slices and meshes in respective windows (see the two middle sub-windows in figure 1). Listing 1 shows example usage of the *milxQtFile* and *milxQtModel* objects for pre-processing surfaces as part of a larger medical imaging processing pipeline.

Both objects share common scientific visualisation display options such as scalar bars, contouring and measuring quantities like angles and distance from the view ports. These capabilities are bundled into the base class for both objects called *milxQtRenderWindow*, which can be thought of as *milxQt*'s common data object. The overall design of the classes is summarised in figure 4. The *milxQtRenderWindow* class provides the basis

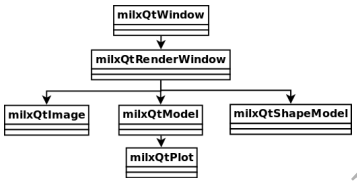


Figure 4: The class design structure of *milxQt*, the Qt based sub-library higher-level layer in SMILI.

of extending the data model based design of SMILI from images and models to other medical imaging data, such as shape models (implemented as *milxQtShapeModel* objects), user specified data or data types of the future.

Each *milxQt* object not only uses the relevant *milxSMILI* classes for processing, but uses the *QVTKWidget* object from VTK to display the data appropriately given the type of data and the parameters. Thus, each of the *milxQtImage* and *milxQtModel* objects manifest to the user as a GUI window that holds data, processes this data and all members of the classes are available as context menu options. A schematic representation of this object oriented design with example MR hip image is shown in figure 5. Crucially, with such a high level design, internal uses of ITK and VTK are hidden as much as possible, allowing the end-developer the freedom in writing applications for their research with a reduced learning curve (such as listing 1).

The design also naturally provides a Python scripting interface that is exposed automatically by the PythonQt library (pythonqt.sourceforge.net) accessible through a plugin for SMILI. The result is that GUI elements are drivable by Python scripts such as those shown in listing 6. In this listing, the entire sMILX

Listing 6: Python scripting example for scripting SMILI GUI elements.

```
# Load the joints and mydata module into smilx first
execfile("filenames.py") #module shipped with SMILI

meshPath = "parameterisations_surf/"
outputExt = ".vtk"
outputPrefix = "prostate_"

dirs = os.listdir(meshPath)
for file in dirs:
  MainWindow.loadFile(meshPath+file) #load mean mesh

  currentModel = MainWindow.activeModel() #get loaded mesh

  #find case id in filename and remember it
  case = getCaseID(file, 0) #filenames module

  outputName = outputPrefix+str(case)+outputExt

  #process model
  currentModel.clean()
  currentModel.smoothSinc(20) #windowed sinc
  currentModel.decimate(0.25)
  currentModel.smoothSinc(20) #windowed sinc
  currentModel.decimate(0.25)
  currentModel.smoothSinc(20) #windowed sinc

  milxQtFile.saveModel(outputName, currentModel) #save result

  currentModel.close() # close (since its delete on close)
```

application (via the MainWindow variable) is driven by the script to load all meshes in a folder, process each mesh and save it as a separate mesh. Each result can be loaded and viewed on-the-fly or hidden as required, as well as adjusting the camera and other visualisation options. The *milxQtFile* object is automatically loaded into the Python environment to allow file I/O.

Of the two additional classes for application creation, the first is a plotting class called *milxQtPlot* derived from *milxQtModel* that supports volume (rendering), surface and scatter plots (see figure 4). The other is the *milxQtMain* class designed for supporting all the aforementioned data classes in a user friendly main window for image viewer applications for developers. This class is the primary class for rapidly developing clinical applications and is discussed in detail in the next section in creating a prototype viewer for general biomedical imaging end-users.

2.3. sMILX - A Simple Medical Imaging Viewer

The primary viewer shipped with SMILI is the sMILX application built solely out of the *milxQtMain* class. This class is shown as the main window in figure 1 and effectively brings together the multi-instance display of the *milxQtImage*, *milxQtModel* and *milxQtPlot* objects, their interactions, as well as providing a GUI frontend for end-users to use for general medical image visualisation and processing. The viewer can also be Python scripted as shown in listing 6.

A custom main window object can be inherited from the *milxQtMain* object to instantly create a fully featured application as shown in figure 2. This object has mechanisms for maintaining tabbed workspaces, each one allowing the display of multiple windows, which can be *milxQtImage*, *milxQtModel* objects or any derivative of their base object if implementing one's own window. It implements interfaces, such as buttons, toolbars and dock windows, for loading, saving and converting

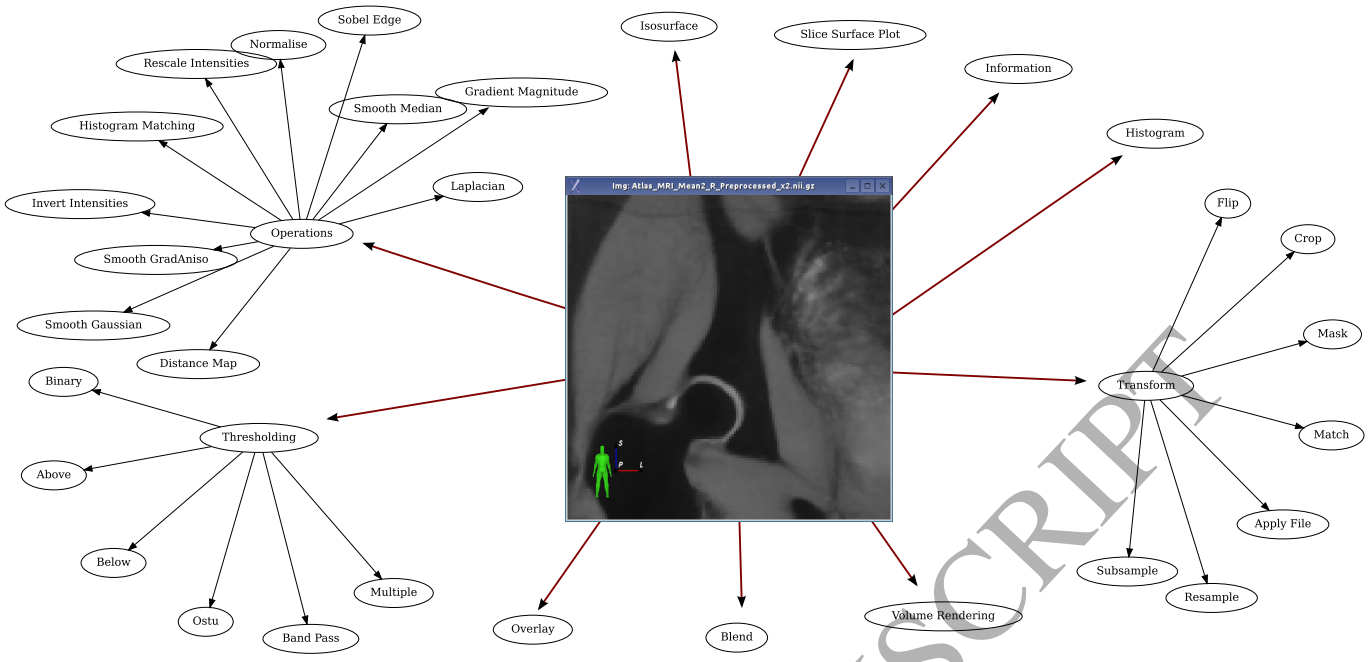


Figure 5: A schematic of the display and processing capabilities of SMILI for *n*-D images represented as *milxQtImage* object, which are also available as right click (context menu) options. Each GUI object in SMILI manifests as a window with the show() member using the Qt framework and all members mirror those of the right click options.

imaging data, as well as various view options such as taking screenshots, tiling and linking views of windows. The class also has mechanisms to load plugins via a compact plugin interface implemented in the *milxQtPluginInterface* class.

Figure 6 shows the result of window interaction (by double clicking and dragging) between the data objects, such as *milxQtImage* and *milxQtModel*, to create overlays containing multiple data combined into a single window. In this case, the windows are aware of what type of display they should transfer to other windows (such as an image slice for images) and (automatically) update other windows if the view is changed accordingly. Two additional *milxQt* command-line applications, namely the *milxOverlay* and *milxAnimate*, also offer the same capability for overlaying multiple data for (off-screen rendered) screenshots and movies respectively.

The additional *milxQtPluginInterface* provides the ability to extend *milxQt* and the *milxQtMain* based applications with plugins, but is not essential to SMILIs core functionality. A number of plugins are already provided for DICOM viewing/conversion (including DICOM-RT), animation, Python scripting and image registration.

3. Method

To demonstrate (the functionality of) SMILI and validate the framework in a clinical use case, we conducted three experiments. To show the code re-use, flexibility in user workflows and duality in object oriented design, we show and compare medical imaging use cases of processing MR images via the command-line and GUI applications. We then do a feature set comparison with some popular medical imaging software packages to show how SMILIs light-weight design permits a rich feature set. Finally, we discuss the design and implementation of a custom clinical application in manual image annotation and measurement within musculoskeletal use case and describe these results in section 4.

3.1. Duality in Object Oriented Design

In a typical medical imaging use case, a researcher needs to convert a DICOM series and pre-process the resulting MR image(s). In the following experiment, we show how these operations are achieved using the same algorithms for two workflows incorporating the command-line and GUI applications, mainly through the sMILX viewer, respectively. The data set chosen for this experiment comprise shoulder MR images utilised in [19]. In another typical medical imaging use case, we evaluate the computational and time performance of SMILI command-line and sMILX applications in handling large studies consisting of hundreds of MR images by processing and visualising this data. The data set chosen for this experiment comprised pelvic MR images (including the prostate) utilised in [20] of 38 patients each having eight timepoints (i.e. 304 3D MR Images totalling 4.5 GB of compressed storage and approximately 9.5GB of uncompressed storage at 256x256x128 resolution per image). A similar experiment was also conducted with sMILX. High resolution multi-channel data from a Sampling Perfection with Application optimized Contrasts using different flip angle Evolution (SPACE) knee scan at 7T (total of 28 images, each approximately 115 MB each in size compressed, 3.2 GB on compressed storage at 464x640x192 resolution and 0.5 mm isotropic acquisition per channel) were simultaneously visualised in sMILX and the performance of the visualisation was noted.

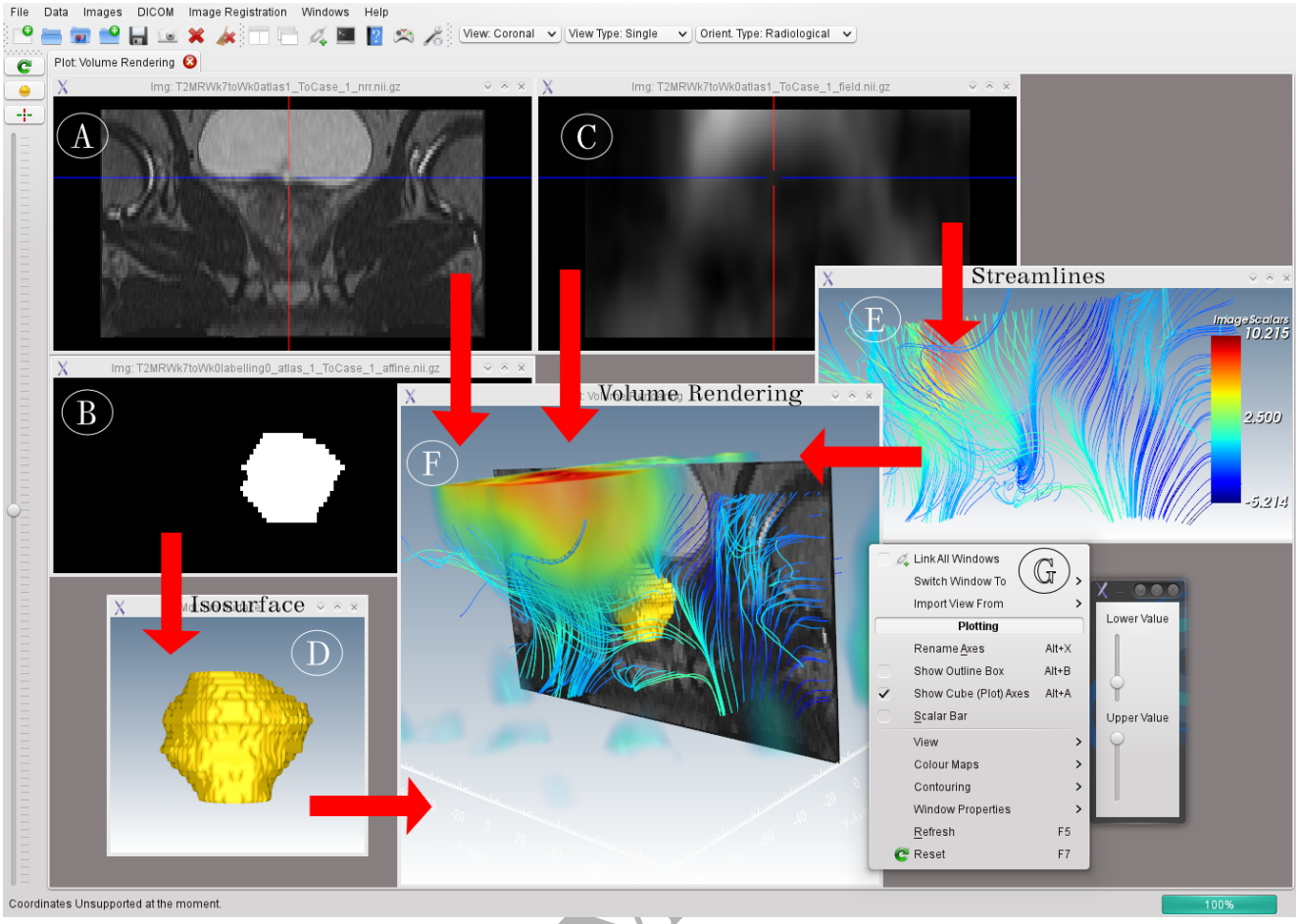


Figure 6: The overlay capability of the *milxQtMain* object (and hence the sMILX application) for supported data forms. The above schematic shows an example of an MR prostate dataset (A), its corresponding manual segmentation (B) (and surface representation (D)) and the non-rigid registration deformation field (C) (and streamlines seeded from the slice in view (E)) visualised in sMILX. The red arrows indicate the operations applied and direction of data placed into the central window (F) (with a volume rendering of (C)) for overlay. The data is transferred between windows by a double click and drag mouse operation or via context menus such as (G).

We then discuss the advantages and limitations of each approach and comment on which code executed is common within each workflow.

3.2. Feature Set Comparisons

To compare SMILIs feature set to the state-of-the-art, each major feature of 3D Slicer [8] was compared to the following popular software packages: ITK-SNAP [15], MITK [3], MeVisLab [10], as well as SMILI. The packages were chosen to represent a feature complete medical imaging library and viewer, a popular fast medical image viewer, a feature complete biomedical imaging library and a commercial package, respectively. Table 1 was constructed to show current features in 3D Slicer, which is mature and feature complete at the time of writing, and compared with these software packages. The classes in these packages were analysed using Visual Studio Code Metrics (Microsoft) and SourceMonitor (Campwood Software LLC). The numbers for defined classes were computed from respective software package sources without extensions/examples/applications and then for a more minimal set of core libraries and modules that still comprise the feature set within 3D Slicer. The advantages and limitations of SMILI was then discussed when compared to these software packages.

3.3. Custom Clinical Application

To demonstrate the implementation of SMILIs applied to a practical clinical use case, a custom end-user application was constructed. The application allowed measurements related to FAI, where bone lesions of the femur (cam-type) and acetabulum (pincher-type) have been associated with osteoarthritis (OA) in the hip joint of young adults [21]. Specifically, manual measurements of α-angles between the femoral neck axis and the most cephalic point of femoral head asphericity determined from multiple planes through the femoral head (see figure 7) [22, 23]) were to be calculated from automatically generated standardised radial 2D slices from 3D MR images. Cam-lesion severity is commonly determined using a two-dimensional (2D) α-angle with a threshold value of 50° as an indicator for asphericity of the femoral head-neck junction [24]. A schematic representation of FAI and α-angle calculation is given in figure 7.

The dedicated application, called "*Impinge*", was developed to standardise manual assessment of α-angles of the femoral

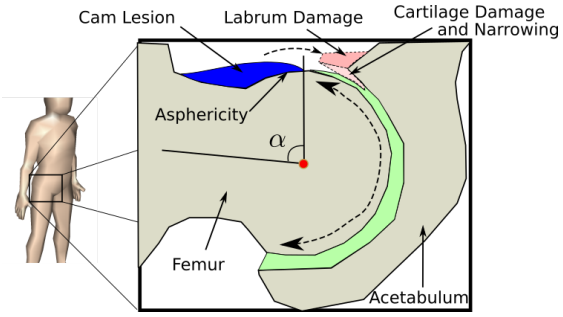


Figure 7: A schematic of a bony cam lesion of the femoral head-neck junction, which may predispose patients with FAI to hip OA [21]. The blue region indicates a cam deformity impinging into structures such as the acetabular labrum and joint cartilage. This can restrict movement (see dashed arrows), frequently becoming symptomatic and may be associated with chronic degeneration of the cartilage (in pink). The red dot indicates the femoral head centre that can be used to characterise the asphericity due to the lesion as an α-angle from the femoral neck axis.

head-neck region from 3D MR images from 31 volunteers (including healthy active individuals and high performance athletes involved in sports such as rugby and water polo). *Impinge* provided a streamlined alternative method to a previous cumbersome multi-software approach, which involved extracting radial views using the OsiriX software [13, 14] and then annotating these views using the Synedra View Personal software (non-commercial and non-medical use license), while switching between the Mac OSX and Windows platforms.

In the current work, an experienced musculoskeletal anatomist used *Impinge* to manually measure the α-angles from T2-weighted water-excited Double-Echo Steady State (weDESS) MR images acquired with a Siemens 3T Trio scanner [25] in a study approved by the medical research ethics committee of the University of Queensland. The MR images were automatically segmented using an algorithm previously developed by the authors [25] to obtain the relevant regions of the bone surface of the proximal femur.

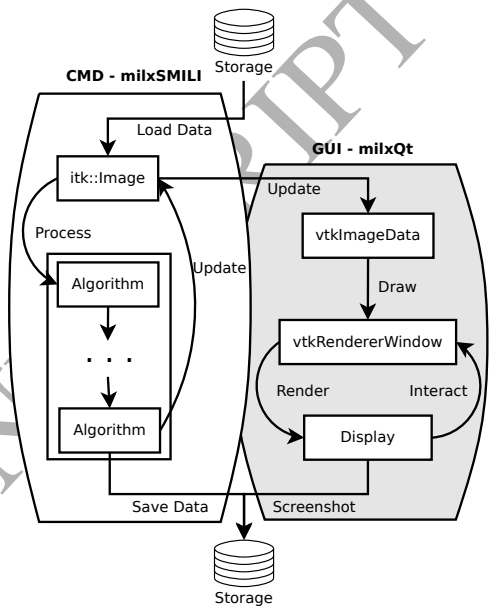
The *Impinge* application provided the necessary radial views of the 3D MR image data and GUI (on-slice) measurement instruments to allow clinical researchers to easily and quickly make α-angle measurements for quantification of cam lesion severity. Once the views are extracted, GUI instruments for image annotation, such as distance, angle and circle widgets, are available to the researcher(s) from a menu to make cam lesion related measurements.

4. Results

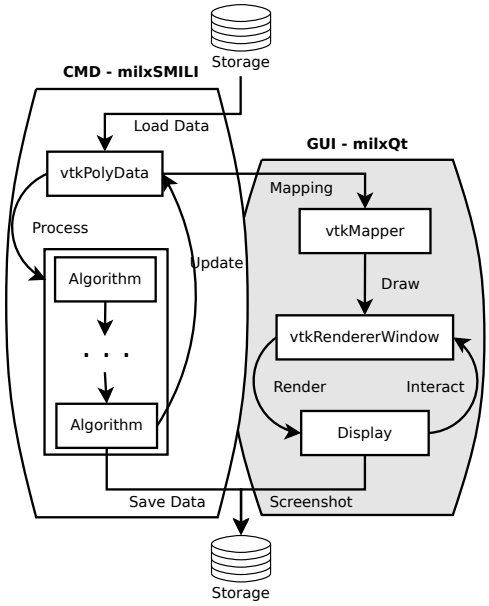
In the following subsections, we show how SMILI supports multiple workflows and maximal code reuse (in section 4.1), how SMILIs feature set is a substantial subset of features available relative to its size compared to much larger biomedical imaging packages (in section 4.2) and how it can be used to build a fully functional clinical research application in less than 10 hours of development (in section 4.3) for manual image annotation tasks from radiological experts.

4.1. Duality in Object Oriented Design

The duality of object oriented design and internal state are summarised using diagrams for the image and model (lines, surfaces etc.) data types in figure 8. These diagrams visually represent the separation of visualisation (via *milxQt*) and processing (via *milxSMILI*) layers for imaging and model data in command-line and GUI applications, but allowing the same interface and algorithms. To the best of our knowledge, no other software package for biomedical imaging offers a similar capability.



(a) Image States



(b) Model States

Figure 8: The state diagrams for (a) image and (b) model (lines, surfaces etc.) data represented in SMILI. Imaging data utilises ITK to ensure medical images are maintained in their correct patient spaces to prevent loss of information. Model data natively utilises VTK for rapid processing and display.

The processing steps necessary in a typical medical imaging use case were performed using the SMILI command-line applications in the following way:

- Shoulder MR data were converted to Nifti images using the *milxDICOMApp* as listing 7. Options such as image info and the option of exporting DICOM tags is also available.
- A high resolution scan was then selected and smoothed using anisotropic diffusion [26] as listing 8 with the −l option used to auto select the timestep. Other options for smoothing include bilateral and median denoise algorithms.

Listing 7: DICOM conversion with SMILI via the command-line.

```
milxDICOMApp --convert shoulder\Subject2 -p output\shoulder_
```

Listing 8: Image denoising with SMILI via the command-line.

```
milxImageApp --smooth -l shoulder_2.nii.gz -o preproc_2.nii.gz
```

The same operations were achieved with the GUI application sMILX or via the GUI API made available with *milxQt* in the following way:

- Shoulder MR data were converted to Nifti images using the DICOM plugin in sMILX or the *milxQtMain* openSeries() member via the *milxQt* API.
- The high resolution scan was then smoothed using anisotropic diffusion [26] with the sMILX context menu for images (Right Click → Operations → Smoothing via Anisotropic Diffusion). The same option is available via the *milxQtImage* anisotropicDiffusion() member via the *milxQt* API.

Both command-line and GUI instances of DICOM conversion and image smoothing utilise the same ITK code present in *milxSMILI*. For example, the DICOM was opened using the `milx::File::OpenDICOMSeries<>(...)` member in the *File* class in both workflows. This allows all the members to be utilised in command-line applications with or without GUI elements, Python scripting via the sMILX application Python plugin or in custom GUI applications, thus ensuring maximal code reuse while supporting multiple workflows, all the while with a cohesive interface. Indeed, the Python plugin allows direct access to all the *milxQt* classes and their members, since it is built using the PythonQt library that provides automatic access to Qt derived classes within a Python environment (see listing 6 for example). This allows users to automate all aspects of sMILX, much like defining macros, as they would for a word processing application, but using Python scripting.

For the other typical medical imaging use case for a large study, a total of 304 3D MR images (38 by 8 weeks, approximately 26 MB in size compressed each utilising 4.5 GB of disk space when compressed) were processed at once using the *milxImageApp* with median denoising (see listing 9). The

Listing 9: Denoising a large study with SMILI via the command-line.

```
milxImageApp *_Week?_LFOV.nii.gz --median 1 -p output/
```

command-line application was found to use a maximum of 10.2 GB of memory when loading all the data and taking a total time of 12 minutes to process all 3D images. The majority of the time was found to be taken in writing out the processed images to storage. The visualisation of the multi-channel data was found to take up 20.1 GB of RAM and 2 minutes to load. All image sub-windows could be tiled within the sMILX and viewing done as per usual, including the linking of the views of each sub-window while traversing the 3D imaging data. The processing was done on a 3.6GHz Intel Xeon desktop computer running Ubuntu Linux with 32 GB RAM.

In terms of time complexity of loading and maintaining data in SMILI applications, the *milxSMILI* layer is $O(N)$, where N is the total number of voxels, while the visualisation layer *milxQt* is of $O(2N)$ as the imaging data needs to be updated to VTK image structures for visualisation (such as *vtkImageData*, see image state diagram in figure 8). This can be seen in the memory usage for the experiments for large studies using *milxImageApp* and multi-channel data using sMILX, where memory usage is double in the latter, while noting that imaging data in memory must be in uncompressed form. The time complexities for the processing is dependent on the algorithms utilised.

The duality approach has two main limitations however. Firstly, only those members implemented at all levels are available for multiple workflows. For example, the Gaussian smoothing algorithm is available in *milxSMILI* and *milxQt*, and thus in sMILX, but an option is not implemented in the *milxImageApp* as of yet. This is because the filter is rarely used for medical images, but can be made available with relatively little effort. The second limitation involves the number of members in the *Image* and *Model* classes. As the number of algorithms supported increases, the classes grow in size. This can be overcome in the future by treating these classes as base classes implementing the core algorithms required for biomedical imaging and other derived classes implementing less often used algorithms can be included.

4.2. Feature Set Comparisons

Table 1 shows that although SMILI does not have all the features that larger libraries such as MITK, MeVisLab and 3D Slicer offer, it does have a large number of important medical imaging features with two orders of magnitude less number of classes defined. We also found that the start-up times were also notably different between Slicer and sMILX with the former taking approximately 6 seconds on an idle start (no image loaded) while the latter taking only 1 second using the same computer as the previous subsection. For example, tasks such as semi-automated or automated segmentation algorithms are often sophisticated and generic implementations are not often useful, so that their

inclusion is not essential for biomedical imaging frameworks. They are also the types of algorithms that often need deployment in a clinical setting, for which SMILI has been designed to augment. This re-enforces the potential of SMILI having a faster uptake and a reduced learning curve, while still allowing the construction of feature rich clinical applications deploying biomedical imaging algorithms.

4.3. Custom Clinical Application

A screenshot of the *Impinge* application that is derived from sMILX, running on the Mac OSX platform, as well as a α-angle measurement, is shown in figure 9. The application was

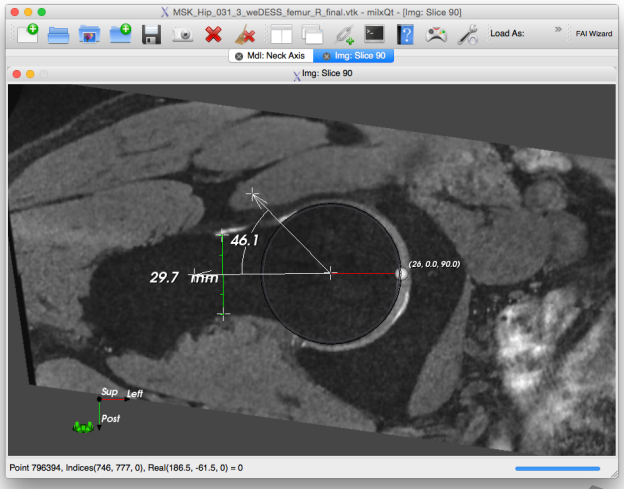


Figure 9: The *Impinge* Mac OSX application demonstrating an α-angle measurement of 46.1° on a radial MR view commonly utilised for investigating cam-type FAI.

developed in under 10 hours of development time and featured very little additional code. Due to the design of SMILI and object oriented design of *Impinge*, very little maintenance was required of the single function housing the FAI related code, namely the 2D radial orientation view generation from 3D MR image inputs. The *Impinge* application was also successfully run and packaged for multiple platforms without any issues encountered.

The validation of the *Impinge* application was made by analysing α-angle data from the weDESS MR images for the anterior-superior (45°) and anterior (90°) positions by an expert rater C.E. and a trainee S.C. The expert rater repeated α-angle measurements on the Mac OSX platform. The intra-rater reliability was ($ICC(1,1) = 0.98$; 95% CI: 0.96-0.99, $p < 0.01$) for the anterior-superior position and ($ICC(1,1) = 0.97$; 95% CI: 0.95-0.98, $p < 0.01$) for the anterior position. This intra-rater reliability was found to be consistent with that of the previous (cumbersome) approach of using two different applications on two different platforms [27]. A similar trend in agreement with the view orientations was found with the independent measurements of the trainee S.C. The inter-rater correlations of the trainee and the expert rater are summarised in the plots shown

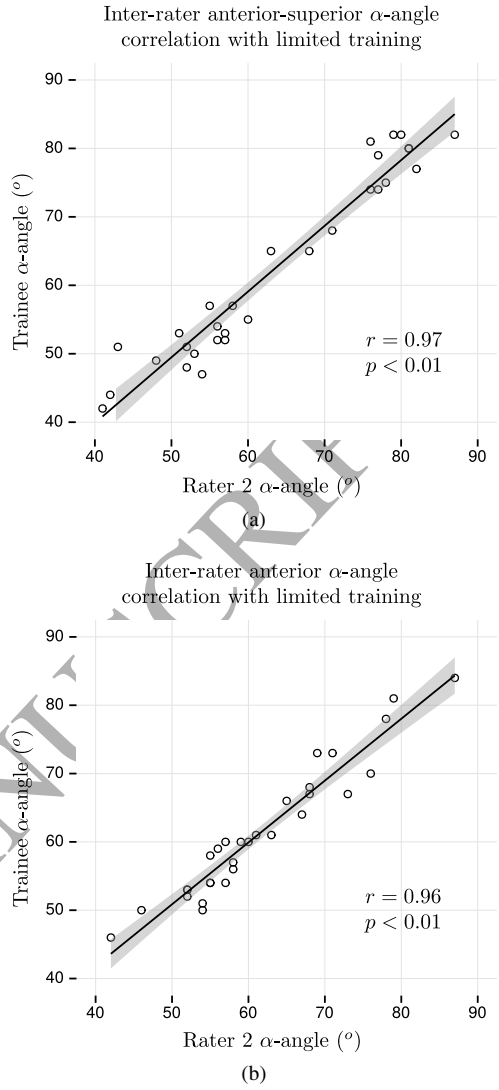


Figure 10: Correlations between manual α-angle measurements of an expert and a trainee at the (a) anterior-superior and (b) anterior positions of the femoral head-neck junction. The manual measurements show minor differences between the expert rater and the trainee based on the automatic view orientation processing within *Impinge* for bilateral weDESS images of both sides of 31 participants.

in figures 10. The measurements of the trainee had strong correlations (statistically significant) with the instructor and expert rater C.E. It suggests that reliable α-angle measurements can be made with limited training when using *Impinge* and demonstrates that SMILI is able to create clinical applications with minimal effort suitable for use with limited training, while available open-source.

The versatility of SMILI makes it ideally suited for a range of biomedical studies and has already been successfully utilised for a number of other applications since its release. It has been employed for advanced 3D *in vivo* visualisation of structures and models in musculoskeletal radiology [28, 29], computing and visualising surface distances between models and bone shape [30, 31], as well as in recent work on deformation fields in radiotherapy treatment planning [32, 33]. The latter has clinical value in not only understanding the effects of

distortion for the purposes of MR alone treatment planning, but also demonstrates that end-users can utilise SMILI built applications, such as sMILX, for clinical research. Recent work has been completed in using SMILI and its *milxQtMain* class (as described in figure 2) to deploy HDR algorithms for multi-channel and multi-sequence MR images [16, see also GitHub].

Finally, an example of integrating SMILI into a biomedical processing workflow is given by Chandra *et al.* [20]. A number of key steps are required during the segmentation and visualisation stages of the workflow. The segmentation is performed in stages by hierarchically weighting the surface fitting of the organs from largest, most easily discernible, to smallest, most difficult, progressively until the prostate is obtained in isolation. The algorithm depends on a series of preset weights for each organ at each stage (see figure 2 of [20]) and the organs are represented by a single hybrid (combined) surface (see figure 1 of [20]). A brief summary of the overall multi-object workflow is shown in figure 11 with the key components provided by SMILI shown in bold black boxes. The workflow was implemented using a Python script that ties all the executables together. This facilitates easier execution and maintenance on a cloud based platform. The entire workflow could also have been implemented directly as a C++ executable by calling the relevant members of SMILI and custom algorithms, but would have required handling the multi-threaded queueing of individual stages (that is blocks in figure 11) within this executable rather than via Python, which has a much simpler interface.

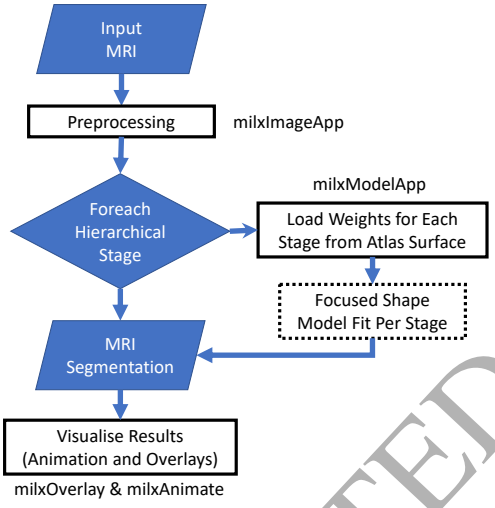


Figure 11: The workflow of Chandra *et al.* [20], where bold black boxes show the elements utilising SMILI and the corresponding application(s). Every input MR image is preprocessed using image smoothing algorithms available in *milxImageApp*. During multi-object segmentation via a hybrid surface made of multiple objects, weights are transferred from atlas surfaces representing each stage via 'scalarcopy' in *milxModelApp*. A custom application does the weighted shape learning segmentation fit as described in [20]. The resulting segmentation and the surfaces (per iteration of the fit) are visualised using the animate and overlay applications available in SMILI.

In summary, we believe SMILI is well suited in developing biomedical applications requiring the use of new algorithms or graphical user improvements in a timely and developmentally efficient manner for clinical interaction. For example, this could include (but not limited to) deploying new image analysis or image fusion techniques to clinical research sites for evaluation. Or new manual assessment techniques and algorithms as GUI applications to clinical sites. These applications will immediately have cross platform and cross desktop manager support. For example, see figures 1, 6, 10 and [16, figure 2] for Ubuntu Unity (Linux), Ubuntu KDE4 (Linux), Mac OSX and Windows 10 environments respectively. Applications not well suited to be developed with SMILI are those requiring new visualization techniques in the realm of computer graphics. This will require implementing new classes and structures in VTK, which is not trivial.

For datasets that cannot fit into RAM, SMILI currently does not provide such capability directly yet. Streaming off storage devices is supported in ITK and VTK libraries and could be enabled within the processing and reading filters that are utilised in SMILI. Libraries built on VTK that support parallel and networked streaming of storage devices, such as Paraview [11, 12], could also be integrated as a plugin or derived classes to stream very large datasets for processing and visualisation, but this would be future work on our proposed framework.

Further work is required so that the majority of ITK and VTK features are available at all layers of SMILI, including streaming large datasets. More plugins for cloud interaction and specific anatomy (such as Neuro-imaging) is also under development for broader appeal. The authors have secured a grant for the long-term development of SMILI and are committed to maintaining it for the duration of its software lifetime.

Conclusion

This paper presented a lightweight open-source library for (cross-platform) biomedical imaging application development entitled SMILI. This framework aims to allow rapid construction of biomedical imaging applications (see figure 2) by providing just a few main pre-built processing and GUI components with very high level interfaces suitable also for Python scripting. SMILI also provides a substantial subset of features to existing open-source and commercial alternatives, while having a commercially viable license and being substantially more compact (see table 1). It supports different user workflow types through the duality of objects at multiple API layers built into its object oriented design. SMILI was validated for clinical research application developed for the measurement of α-angles of the femoral head-neck region for assessment of cam-type lesions via image annotation. This application not only had comparable accuracy as a previous more cumbersome approach, but also resulted in time savings and greater convenience important for clinical users.

Acknowledgments

We would like to thank Dr. Duncan Walker for assisting in the development of *Impinge* and for his manual measurements of α-angles. We also thank the ITK and VTK development teams for their work on their respective libraries.

References

[1] L. Ibáñez, W. Schroeder, L. Ng, J. Cates, The ITK software guide: the insight segmentation and registration toolkit, Kitware, Inc., 2nd Edition (2005).

[2] W. Schroeder, K. Martin, B. Lorensen, Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th Edition, Kitware, 2006.

[3] I. Wolf, M. Vetter, I. Wegner, T. Böttger, M. Nolden, M. Schöbinger, M. Hastenteufel, T. Kunert, H.-P. Meinzer, The medical imaging interaction toolkit, Medical Image Analysis 9 (6) (2005) 594–604. doi:10.1016/j.media.2005.04.005.

[4] Y. Zheng, A. Barbu, B. Georgescu, M. Scheuering, D. Comaniciu, Four-Chamber Heart Modeling and Automatic Segmentation for 3-D Cardiac CT Volumes Using Marginal Space Learning and Steerable Features, IEEE Transactions on Medical Imaging 27 (11) (2008) 1668–1681. doi:10.1109/TMI.2008.2004421.

[5] J. Fripp, S. Crozier, S. Warfield, S. Ourselin, Automatic segmentation and quantitative analysis of the articular cartilages from magnetic resonance images of the knee, Medical Imaging, IEEE Transactions on 29 (1) (2010) 55–64. doi:10.1109/TMI.2009.2024743.

[6] S. Martin, J. Troccaz, V. Daanen, Automated segmentation of the prostate in 3D MR images using a probabilistic atlas and a spatially constrained deformable model, Medical Physics 37 (4) (2010) 1579–1590. doi:10.1118/1.3315367.

[7] J. Schmid, J. Kim, N. Magnenat-Thalmann, Robust statistical shape models for MRI bone segmentation in presence of small field of view, Medical Image Analysis 15 (1) (2011) 155– 168. doi:DOI:10.1016/j.media.2010.09.001.

[8] D. T. Gering, A. Nabavi, R. Kikinis, N. Hata, L. J. O'Donnell, W. E. L. Grimson, F. A. Jolesz, P. M. Black, W. M. Wells, An integrated visualization system for surgical planning and guidance using image fusion and an open MR, Journal of Magnetic Resonance Imaging 13 (6) (2001) 967–975. doi:10.1002/jmri.1139.

[9] D. Blezek, L. Ibáñez, H. Johnson, B. Lowekamp, SimpleITK Tutorial: Image processing for mere mortals, Medical Image Computing and Computer-Assisted Intervention - MICCAI Workshop.

[10] A. Medical Solutions, M. Fraunhofer, MeVisLab - development environment for medical image processing and visualization, Bremen, Germany (2013).
URL http://www.mevislab.de/

[11] J. Ahrens, K. Brislawn, K. Martin, B. Geveci, C. Law, M. Papka, Large-scale data visualization using parallel data streaming, IEEE Computer Graphics and Applications 21 (4) (2001) 34–41. doi:10.1109/38.933522.

[12] J. Ahrens, B. Geveci, C. Law, Paraview: An end-user tool for large data visualization.

[13] A. Rosset, L. Spadola, O. Ratib, OsiriX: An Open-Source Software for Navigating in Multidimensional DICOM Images, Journal of Digital Imaging 17 (3) (2004) 205–216. doi:10.1007/s10278-004-1014-6.

[14] A. Rosset, L. Spadola, L. Pysher, O. Ratib, Informatics in radiology (infoRAD): navigating the fifth dimension: innovative interface for multidimensional multimodality image navigation, Radiographics: A Review Publication of the Radiological Society of North America, Inc 26 (1) (2006) 299–308. doi:10.1148/rg.261055066.

[15] P. A. Yushkevich, J. Piven, H. C. Hazlett, R. G. Smith, S. Ho, J. C. Gee, G. Gerig, User-guided 3D active contour segmentation of anatomical structures: significantly improved efficiency and reliability, NeuroImage 31 (3) (2006) 1116–1128. doi:10.1016/j.neuroimage.2006.01.015.

[16] S. S. Chandra, C. Engstrom, J. Fripp, D. Walker, S. Rose, C. Ho, S. Crozier, Local Contrast Enhanced MR Images via High Dynamic Range Processing, Magnetic Resonance in Medicine 80 (3) (2018) 1206–1218. doi:10.1002/mrm.27109.

[17] J. A. Dowling, N. Burdett, P. B. Greer, J. Sun, J. Parker, P. Pichler, P. Stanwell, S. Chandra, D. Rivest-Hénault, S. Ghose, O. Salvado, J. Fripp, Automatic Atlas Based Electron Density and Structure Contouring for MRI-based Prostate Radiation Therapy on the Cloud, Journal of Physics: Conference Series 489 (1) (2014) 012048. doi:10.1088/1742-6596/489/1/012048.

[18] J. Goecks, A. Nekrutenko, J. Taylor, Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences, Genome Biology 11 (8) (2010) R86. doi:10.1186/gb-2010-11-8-r86.

[19] Z. Yang, J. Fripp, S. S. Chandra, A. Neubert, Y. Xia, M. Strudwick, A. Paproki, C. Engstrom, S. Crozier, Automatic bone segmentation and bone-cartilage interface extraction for the shoulder joint from magnetic resonance images, Physics in Medicine and Biology 60 (4) (2015) 1441. doi:10.1088/0031-9155/60/4/1441.

[20] S. S. Chandra, J. A. Dowling, P. Greer, J. Martin, C. Wratten, P. Pichler, J. Fripp, S. Crozier, Automatic fast automated segmentation of multiple objects via spatially weighted shape learning, Physics in Medicine and Biology 61 (22) (2016) 8070. doi:10.1088/0031-9155/61/22/8070.

[21] R. Ganz, J. Parvizi, M. Beck, M. Leunig, H. Nötzli, K. A. Siebenrock, Femoroacetabular impingement: a cause for osteoarthritis of the hip, Clinical Orthopaedics and Related Research (417) (2003) 112–120. doi:10.1097/01.blo.0000096804.78689.c2.

[22] R. Sutter, T. J. Dietrich, P. O. Zingg, C. W. Pfirrmann, How Useful Is the Alpha Angle for Discriminating between Symptomatic Patients with Cam-type Femoroacetabular Impingement and Asymptomatic Volunteers?, Radiology 264 (2) (2012) 514–521. doi:10.1148/radiol.12112479.

[23] C. Zilkens, F. Miese, R. Krauspe, B. Bittersohl, Symptomatic Femoroacetabular Impingement: Does the Offset Decrease Correlate With Cartilage Damage? A Pilot Study, Clinical Orthopaedics and Related Research 471 (7) (2013) 2173–2182. doi:10.1007/s11999-013-2812-2.

[24] M. Tannast, K. A. Siebenrock, S. E. Anderson, Femoroacetabular Impingement: Radiographic Diagnosis-What the Radiologist Should Know, American Journal of Roentgenology 188 (6) (2007) 1540–1552. doi:10.2214/AJR.06.0921.

[25] Y. Xia, J. Fripp, S. S. Chandra, R. Schwarz, C. Engstrom, S. Crozier, Automated bone segmentation from large field of view 3D MR images of the hip joint, Physics in Medicine and Biology 58 (20) (2013) 7375. doi:10.1088/0031-9155/58/20/7375.

[26] P. Perona, J. Malik, Scale-space and edge detection using anisotropic diffusion, Pattern Analysis and Machine Intelligence, IEEE Transactions on 12 (7) (1990) 629–639. doi:10.1109/34.56205.

[27] P. Melville, Magnetic Resonance (MR) Imaging Investigation of the Alpha Angle and Cam-Type Femoroacetabular Impingement in High-Performance Male Water Polo Players, Master's thesis, School of Human Movement Studies, University of Queensland (2013).

[28] S. S. Chandra, R. Surowiec, C. Ho, Y. Xia, C. Engstrom, S. Crozier, J. Fripp, Automated analysis of hip joint cartilage combining MR T2 and three-dimensional fast-spin-echo images, Magnetic Resonance in Medicine 75 (1) (2016) 403–413. doi:10.1002/mrm.25598.

[29] A. Neubert, K. J. Wilson, C. Engstrom, R. K. Surowiec, A. Paproki, N. Johnson, S. Crozier, J. Fripp, C. P. Ho, Comparison of 3d bone models of the knee joint derived from CT and 3t MR imaging, European Journal of Radiology 93 (2017) 178–184. doi:10.1016/j.ejrad.2017.05.042.

[30] S. Chandra, I. Svalbe, Exact image representation via a number-theoretic radon transform, IET Computer Vision 8 (4) (2014) 338–346. doi:10.1049/iet-cvi.2013.0101.

[31] A. Neubert, Z. Yang, C. Engstrom, Y. Xia, M. W. Strudwick, S. S. Chandra, J. Fripp, S. Crozier, Automatic segmentation of the glenohumeral cartilages from magnetic resonance images, Medical Physics 43 (10) (2016) 5370–5379. doi:10.1118/1.4961011.

[32] A. Walker, G. Liney, L. Holloway, J. Dowling, D. Rivest-Hénault, P. Metcalfe, Continuous table acquisition MRI for radiotherapy treatment planning: Distortion assessment with a new extended 3d volumetric phantom, Medical Physics 42 (4) (2015) 1982–1991. doi:10.1118/1.4915920.

[33] A. Walker, P. Metcalfe, G. Liney, V. Batumalai, K. Dundas, C. GlideÃćâĆñï£¡Hurst, G. P. Delaney, M. Boxer, M. L. Yap, J. Dowling, D. RivestÃćâĆñï£¡Henault, E. Pogson, L. Holloway, MRI geometric distortion: Impact on tangential whole-breast IMRT, Journal of Applied Clinical Medical Physics 17 (5) (2016) 7–19. doi:10.1120/jacmp.v17i5.6242.