DTP Computational Physiology Documentation

Release 0.1

University of Auckland

March 08, 2016

Contents

Welcome to the Computational Physiology module of the Doctoral Training Programme from the MedTech CoRE. In this series of tutorials, you will be exposed to a selection of the methods used by scientists in the MedTech CoRE working in the field of computational physiology - the application of engineering and mathematical sciences to the study of physiology.

This series of tutorials will first guide you through the main steps in a "standard" clinical workflow that takes advantage of computational physiology: image processing, geometric model building, simulation, and visualisation. You will be led through the development and implementation of a complete clinically focused workflow which will take advantage of the skills you have developed in the earlier modules. Finally, the course ends with an introduction to some of the key tools that are used in this field locally.

Note: Anonymous feedback on any part of this module can be left at: https://goo.gl/6XpHxp

Contents:

CHAPTER 1

DTP CP Virtual Machine

For the May and July 2015 trial runs of the Computational Physiology module, we provide a VirtualBox virtual machine that contains all the tools and documentation required to participate in the module. Getting participants up and running with the virtual machine requires you to *Machine* → *Add* the provided virtual machine - but the tutors are on hand to help with this.

Some useful things to note regarding the provided virtual machine are provided below.

- The default username is `abi` with the password `abipassword`.
- In all the task and example descriptions, *HOME* is used to indicate the folder `/home/abi/`.
- OpenCOR is installed and can be launched via the shortcut on the desktop.

Todo

- add in more useful links in the virtual machine
- make sure the password is what is specified above.
- check consistent use of HOME

CHAPTER 2

Clinical Workflows

In Fig. 2.1 you can see an example of a clinical workflow that adopts a computational physiology approach. In summary: 1) clinical observations are made (in this example, images from a mamographic scan); 2) a model is customised to those observations (a geometric model specific to the given patient); 3) simulations are performed as part of the investigation (simulating mamographic compression on the geometrical model to determine the affect on internal structures); 4) some kind of visualisation is performed to help interpret the simulation reuslts in regard to the clinical observations (registering mamographic and MRI images to highlight potential calcifications in the breast tissue); and 5) the preceeding steps are wrapped into a customised user interface which can be provided to the clincians for them to execute these steps in the clinc.

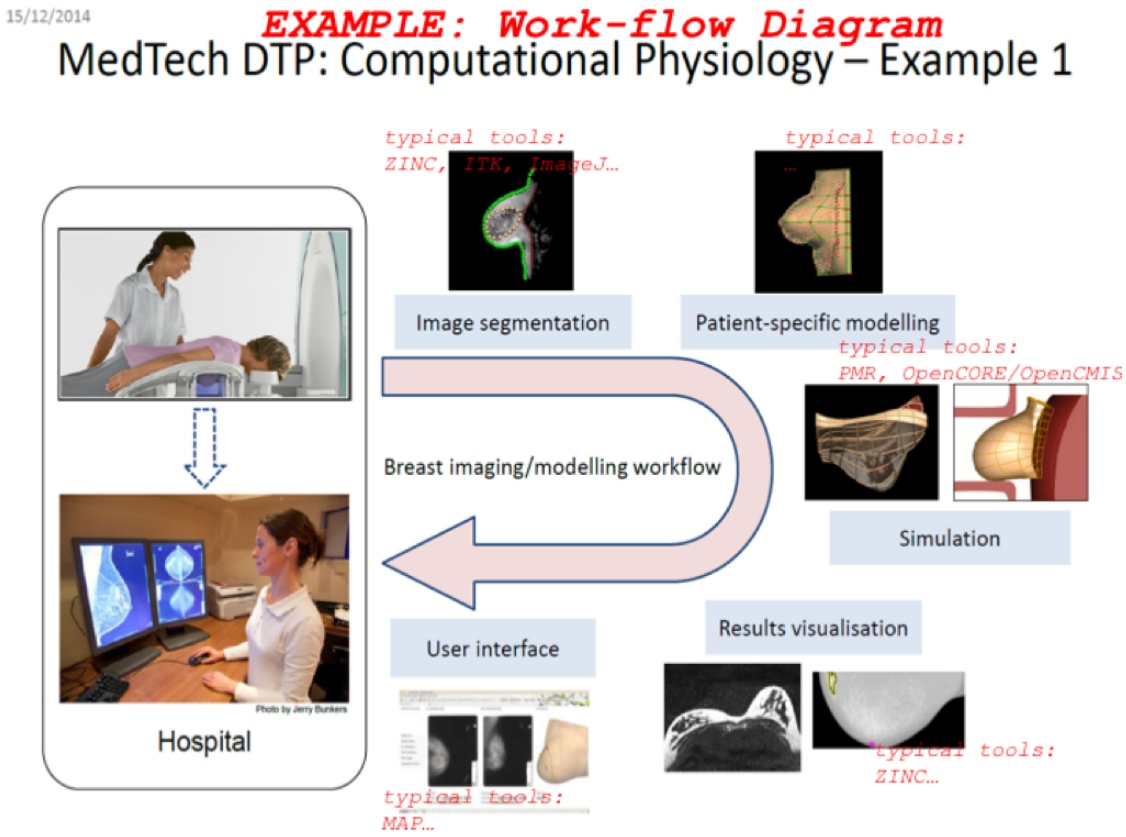

Fig. 2.1: An example of a clinical workflow, showing the steps in the process which form the basis for the Computational Physiology module.

The goal of these tutorials is to make you aware of the key skills required at each step in the process described above and in Fig. 2.1. We achieve this here by leading you through some examples of the various skills and demonstrate their applicability across a range of organ systems and spatial scales. The final step (5) described above is part of the *final module* in the Computational Physiology module, but the remainder of the steps are described here.

2.1 Computational Physiology - Segmentation

Contents:

2.1.1 Image Segmentation

This tutorial is on image segmentation and image data post-processing. The objectives are to gain a basic understanding of the different types of images frequently acquired from medical devices. Understand some of the DICOM standard and the information that is useful to image segmentation. Gain some knowledge on manual and semi-automatic image segmentation and be aware of the reasons for post-processing segmented data.

Overview

Segmentation begins with the acquisition of images. The images used in our domain originate from medical devices. The Types of images that we come across can be roughly divided into two groups macroscopic anatomical images and microscopic anatomical images. The macroscopic image modalities that we typically come across are Magnetic Resonance Images (MRI), Computed Tomography (CT) images and Ultrasound (US). The microscopic images that we use are confocal images.

Most medical devices store their image output using the the Digital Imaging and Communications in Medicine (DICOM) standard. The DICOM standard is a standard that covers the handling, storing, printing and transmitting of information in medical imaging. It includes a file format definition and a network protocol definition. We shall look at the information that can be stored in the DICOM file format and look at how we can make use of it.

With the information garnered from the DICOM file we will orient and display the images for visualisation, ready for segmentation. In the task 2 the segmentation of the images will use a technique called point-and-click digitisation, in task 3 we will perform semi-automatic segmentation using edge detection and edge erosion. In task 4 we will perform some post-processing on the segmented data to transform the data from the machine or magnet coordinates to heart coordinates. Finally, we will put it all together to create a segmentation workflow that will produce two point clouds suitable for left ventricle heart model fitting.

Task 1

Task one is to investigate the DICOM headers from a set of tagged MR images stored using the DICOM standard. It is quite common to see the DICOM standard used interchangeably with the DICOM image format it is important to remember that the DICOM standard is not only for the storing of images. We will use the MAP Client application and load the 'DTP Segmentation Task 1' workflow, with the workflow loaded you should see something like Fig. 2.2.

In this workflow there are three steps, the first step is the image source step from which we will load the DICOM images. The second step is the DICOM header query step which will be used to query tags from the DICOM image. The third stick is a simple Python dict serialisation step so that we may store queries for future reference.

When we execute this workflow we are presented with Fig. 2.3 an interface for querying DICOM headers.

In Fig. 2.3 we can see a number of the GUI elements. On the left of the screen [1] we can see the DICOM image that we have chosen to query and two combo boxes below from which we can select one to perform a query the identified tag. The query button [2] when clicked will query the selected DICOM image with the header tag from the

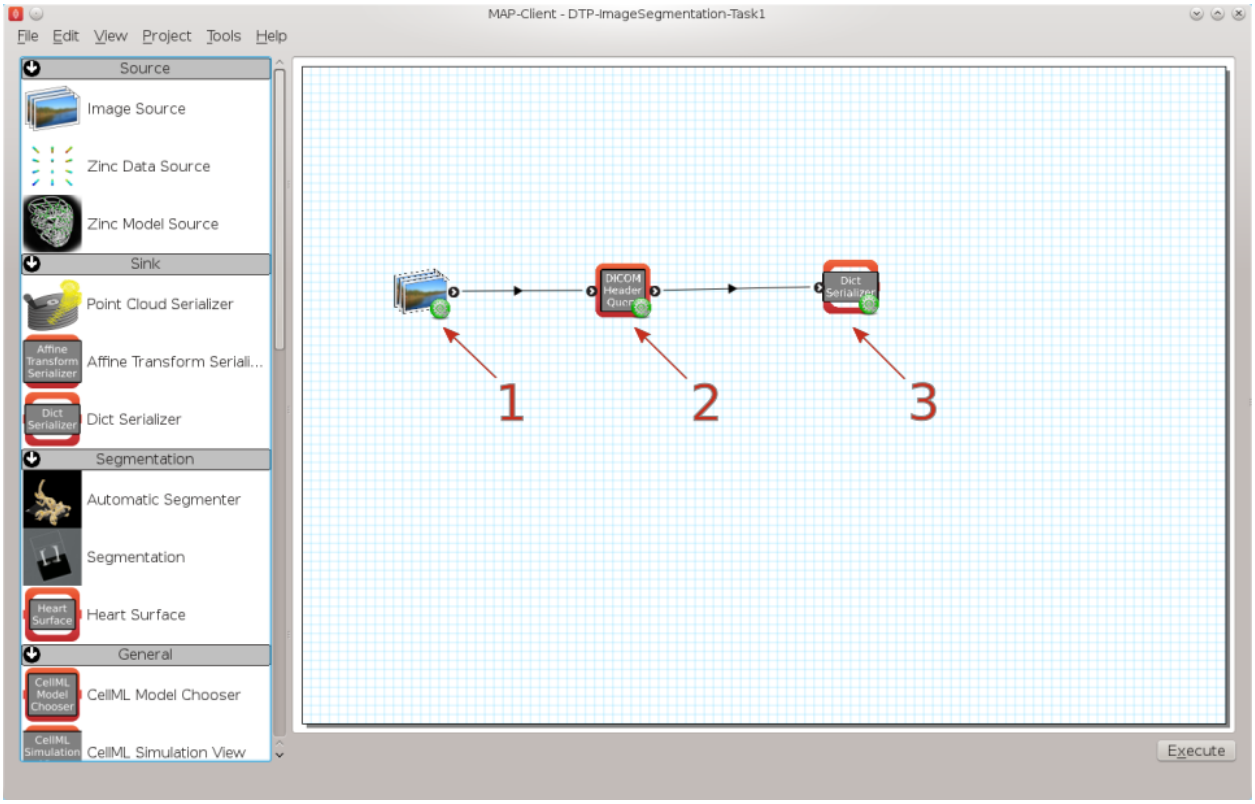

Fig. 2.2: Task 1 workflow

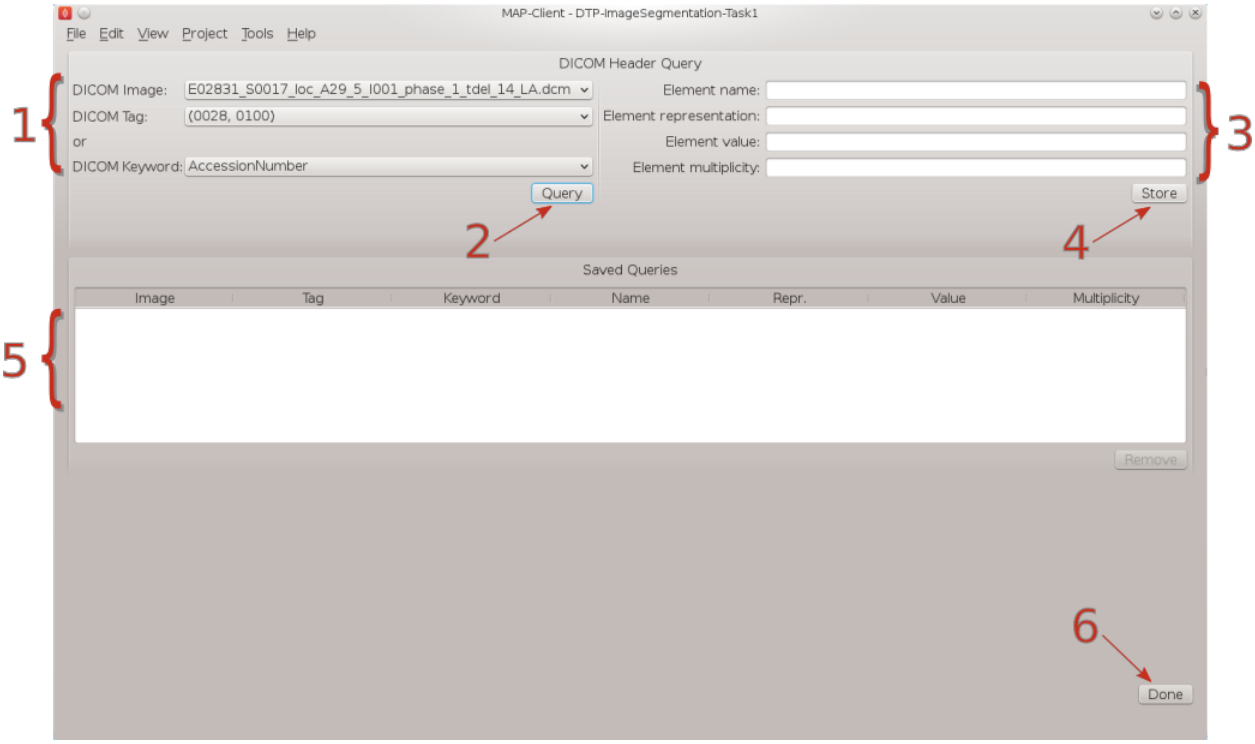

Fig. 2.3: DICOM header query interface

last activated combo box. The results of the query will appear on the right-hand side [3]. On the right-hand side we have four text boxes that will be populated with the result of our query. The element name is related to the DICOM keyword but it is not an exact match, the element representation is defined by the standard and is used to determine the format of the element value. The element value is the actual value of tag queried and the element multiplicity is the number of values in the value element usually the element multiplicity is one. When the store button [4] is clicked the result of the query edit to the saved queries table [5], Rows in this table maybe deleted by selecting the row to be deleted with the mouse and clicking the remove button. When the Done button [6] is clicked the workflow will finish and return to the workflow edit screen.

The DICOM standard is a rather large and ungainly document freely available on the web, of interest to us here is part three of the standard dealing with Information Object Definitions and part six of the standard dealing with the Data Dictionary in particular table 6-1 which relates Tags, Names, Keywords, Element Representation and Element Multiplicity. If you take a look at table 6–1 you will see that it is it defines a great number of terms and in any given DICOM file most of these terms will not be defined. What is of interest here though are the tags relating to the image position in relation to the patient, position of the patient, the pixel spacing, the size of the image and the image data itself.

It is the values taken from these tags that will enable us to correctly orient the images of the patient when we come to segment the left ventricle in task two.

Task 2

Task two is to segment the left ventricle of the heart. Using the MAP Client application again, load the 'DTP Segmentation Task 2' workflow. In this workflow we see five steps there are two image source steps, the heart segmentation step and two point cloud serialisation steps. In this workflow we have two image source steps one for the long axis images and another for the short axis images. We will also differentiate between the endocardial surface and the epicardial surface of the heart which will result in two separate point clouds.

When we execute this workflow we are presented with Fig. 2.4.

In Fig. 2.4 we can see on the left a toolbox that allows us to change the state of this segmentation tool, on the right hand side we can see a three-dimensional view of the two sets of DICOM images. To create this view we have used the

- Pixel spacing
- Image orientation patient
- Image position patient
- Rows
- Columns

information from the DICOM header. This has placed each image plane in the machine or magnet coordinate system. In the images we are using you will see lines across the image picture, this comes from the saturated MR signals so that we can track myocardial motion. In the images that we see we have straight saturated bands indicating that these are the reference images.

From the view toolbox on the left-hand side we can show the image planes and from the file for box we can load and save our progress. The done button is also in the file toolbox for when we are finished segmenting.

Using the view toolbox first hide all the image planes and then make the 13th short axis image plane visible. You should now be looking at something very similar to Fig. 2.5.

See the 3D View Help for help on manipulating the view. Move the image plane to a more suitable view for segmentation. We wish to segment both the endocardial and epicardial surfaces of the left ventricle. In the segmentation toolbox we can see which surface of the heart we are set up to segment. In this view the control key is used as a modifier for the left mouse button to add segmentation points to the scene. With the left mouse button held down we can drag the segmentation points to the desired location. We can also click on existing segmentation points to adjust their position

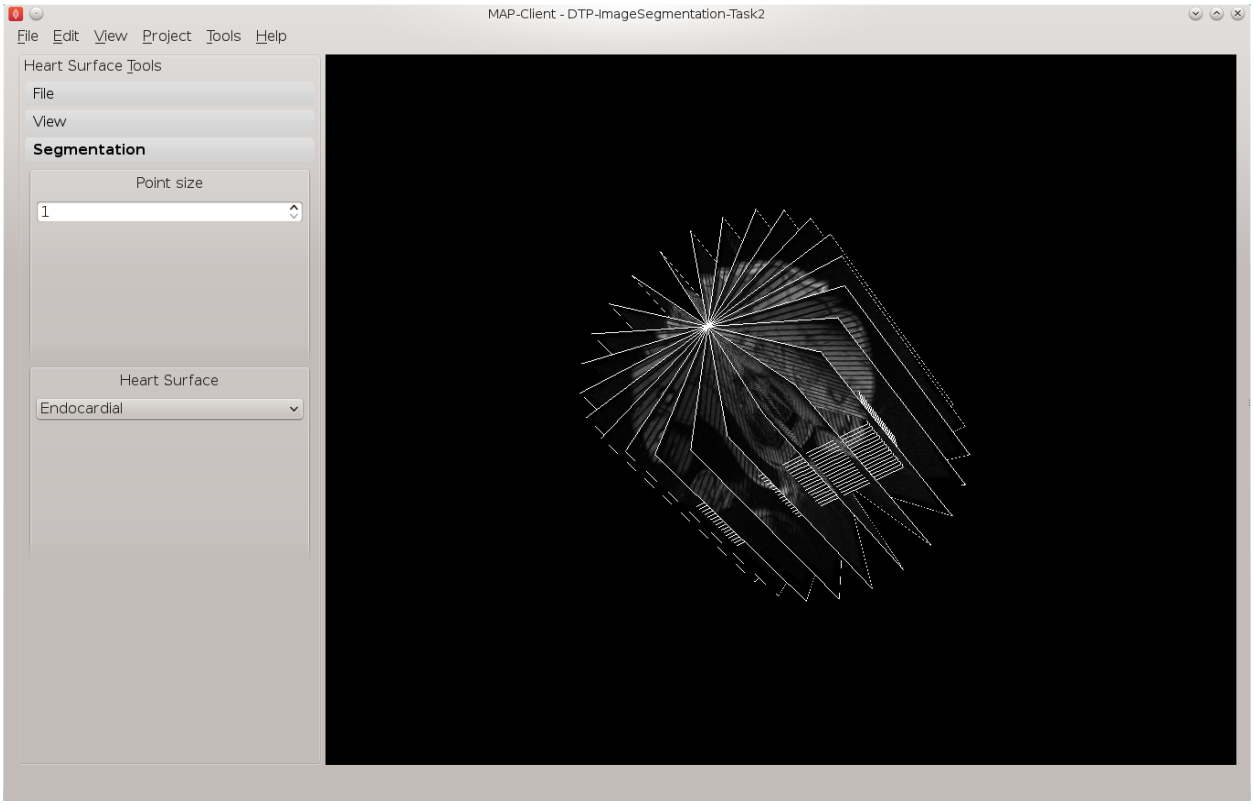

Fig. 2.4: Heart segmentation interface initial state

at a later time. Segmentation points coloured red will be put into the endocardial set of points, segmentation points coloured green will be put into the epicardial set of points. Use the heart surface combo box in the segmentation toolbox to change the current point set.

Segmenting this image should result in Fig. 2.6.

Continue segmenting the left ventricle using the long axis images to check for consistency. The end result should look like Fig. 2.7.

Using the save button from the file toolbox save your progress and click the done button to write the two point clouds to disk.

Task 3

In this task we will use image processing techniques such as edge detection and edge erosion to automatically segment regions of interest. It is often necessary to correct this type of segmentation due to errors in the edge detection or edge errosion process.

Task 4

In this task we want to transform the data created in tasks 2 and 3 from machine coordinates to heart or model coordinates. Open the MAP Client workflow 'DTP Segmentation Task 4' and execute it. You should see the image planes as before. In this task we need to define the heart coordinate system so that we may contstruct the transformation from machine coordinates to heart coordinates. We can do this by selecting three landmark points; the Base point, the Apex point, and the RV point. This will define our heart coordinate system.

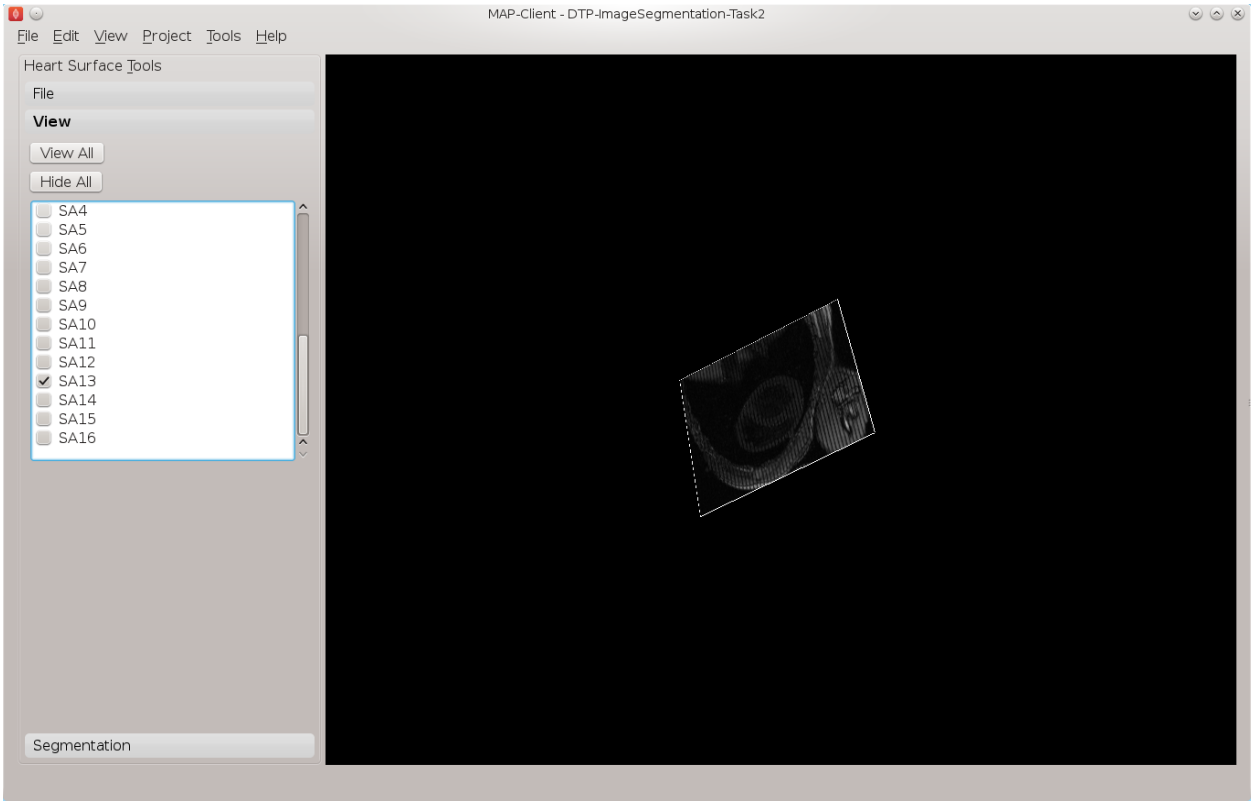

Fig. 2.5: View of the thirteenth short axis image plane

From the transform toolbox we can set the current point we are positioning. Starting with the apex point find the location at the lower pointed end of the heart which defines the bottom of the left ventricle volume. This can be seen the clearest on the 3rd short axis image plane, Fig. 2.8 shows the apex point.

Make only the 13th image plane visible, on this image plane place the landmarks for the base point and the RV point. The base point is the centre of mass of the left ventricle and the RV point is the centre of mass of the right ventricle. See Fig. 2.9 for an example of these locations.

With these three landmarks set we can determine the heart coordinate system. The origin of this system is one third of the way down the base to apex line. The X axis for the system is increasing from the base point to the apex point the, Y axis is increasing from the base point to the RV point and the cross product of these two vectors defines the Z axis. We make this coordinate system orthogonal by projecting the RV-base line onto the base-apex line.

In Fig. 2.10 we can see an axes glyph to represent the heart coordinate system. This glyph should be consistent with the definition from the previous paragraph.

From the file toolbox use the save button to save the location of these points then click the done button to complete this workflow.

Finish

To compete this tutorial we shall put together a complete workflow that will start from DICOM images and result in segmented points of the left ventricle in model coordinates.

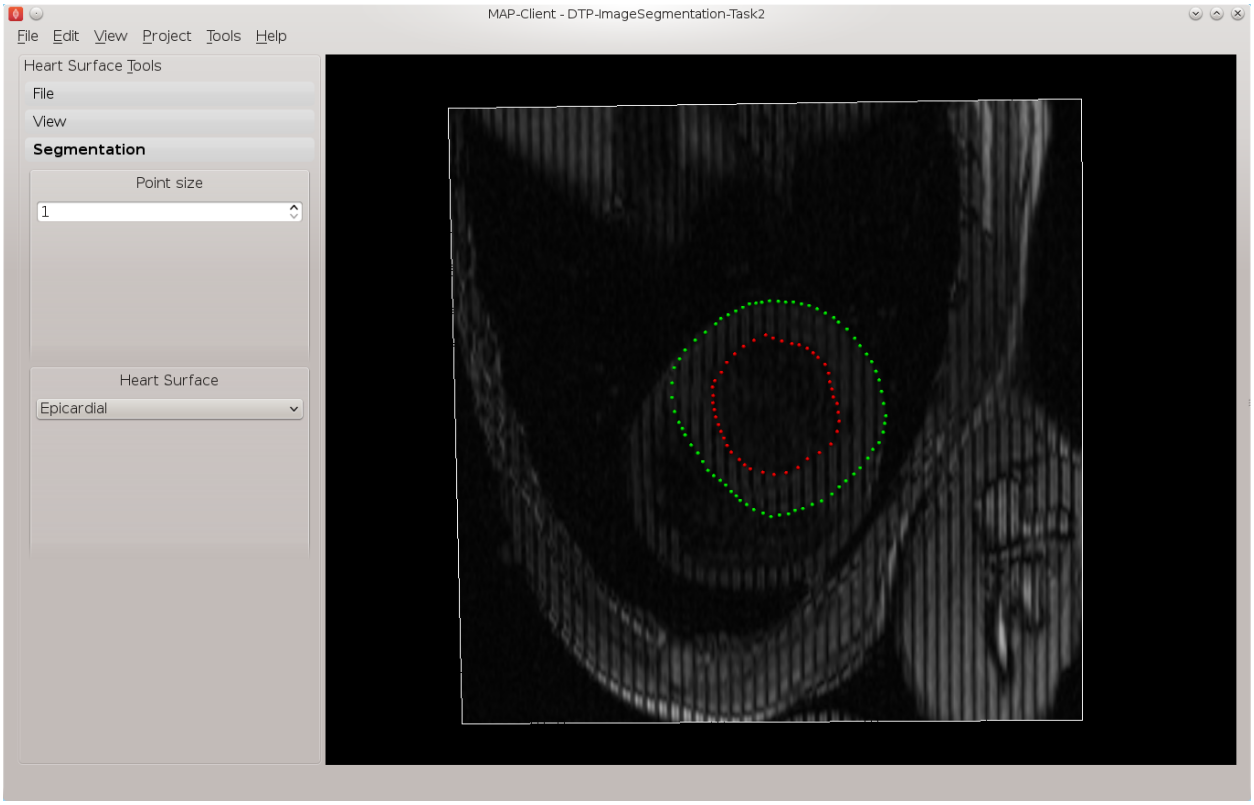

Fig. 2.6: View of the segmented thirteenth short axis image plane

2.1.2 3D-View Help

The section describes how you can control the three-dimensional view. The three-dimensional view can be manipulated with the mouse and some modifier keys. The following sections describe the effect of the different mouse buttons and modifier keys on manipulating the view.

Left Mouse Button

If you can imagine a sphere around the scene and when you click that you're placing a point on the sphere so that when you drag the mouse it as if you are pulling the sphere around buy a piece of string attached to that point. This means that when the whole scene is visible and we click on the edge of the sphere containing the scene we should be able to achieve a pure roll movement of the scene.

By default there are no modify buttons used in this mode, however modifier key functionality is often added to suit a particular application.

Middle Mouse Button

The middle mouse button can be used to pan the scene. Panning the scene results in a translation from the current viewpoint. By default there are no modified buttons used in this mode.

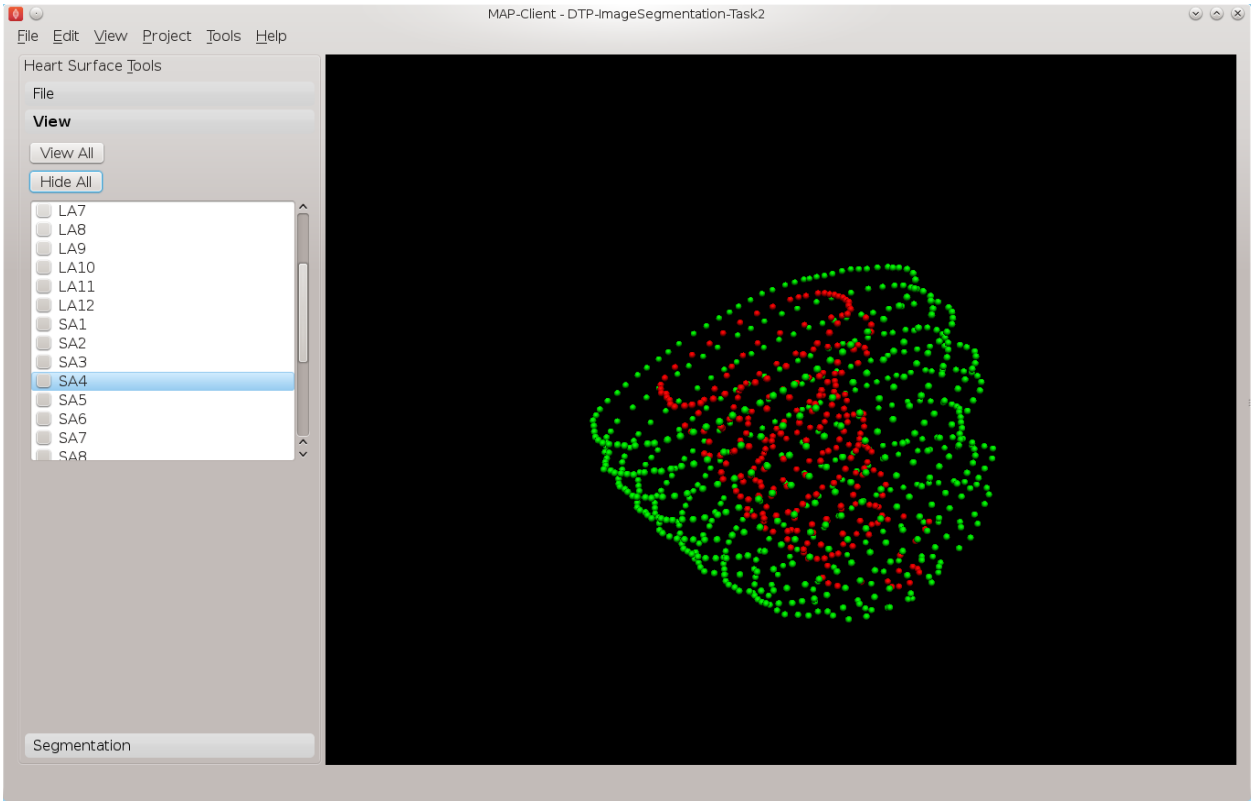

Fig. 2.7: View of the segmented left ventricle

Right Mouse Button

The right mouse button is used for zooming. The zoom applied is what is known as a fly-by-zoom where it appears as if the viewer is walking towards or away from the scene. Clicking and dragging down the screen will make it appear as if you are walking away from the scene and clicking and dragging up the screen will make it appear as if you are walking towards the scene. Using the shift key modifier we can change the type zoom. The shift key changes the zoom to a telephoto lens type of zoom in this mode it is as if the viewer is standing still and using the zoom on a camera or pair of binoculars to change the view.

2.2 Computational Physiology - Model Construction

Contents:

2.2.1 Model Construction

Model construction consists of:

1. *Mesh Creation*: creating a finite element mesh with a topology and interpolation suited to describing the body of interest to the level of detail required.
2. *Geometric Fitting*: customising the geometry of the mesh to be physically realistic, or tailored to an individual.
3. *Material Field Definition*: describing and fitting additional spatially-varying fields which affect behaviour of the body e.g. material properties such as fibre orientations for anisotropic materials.

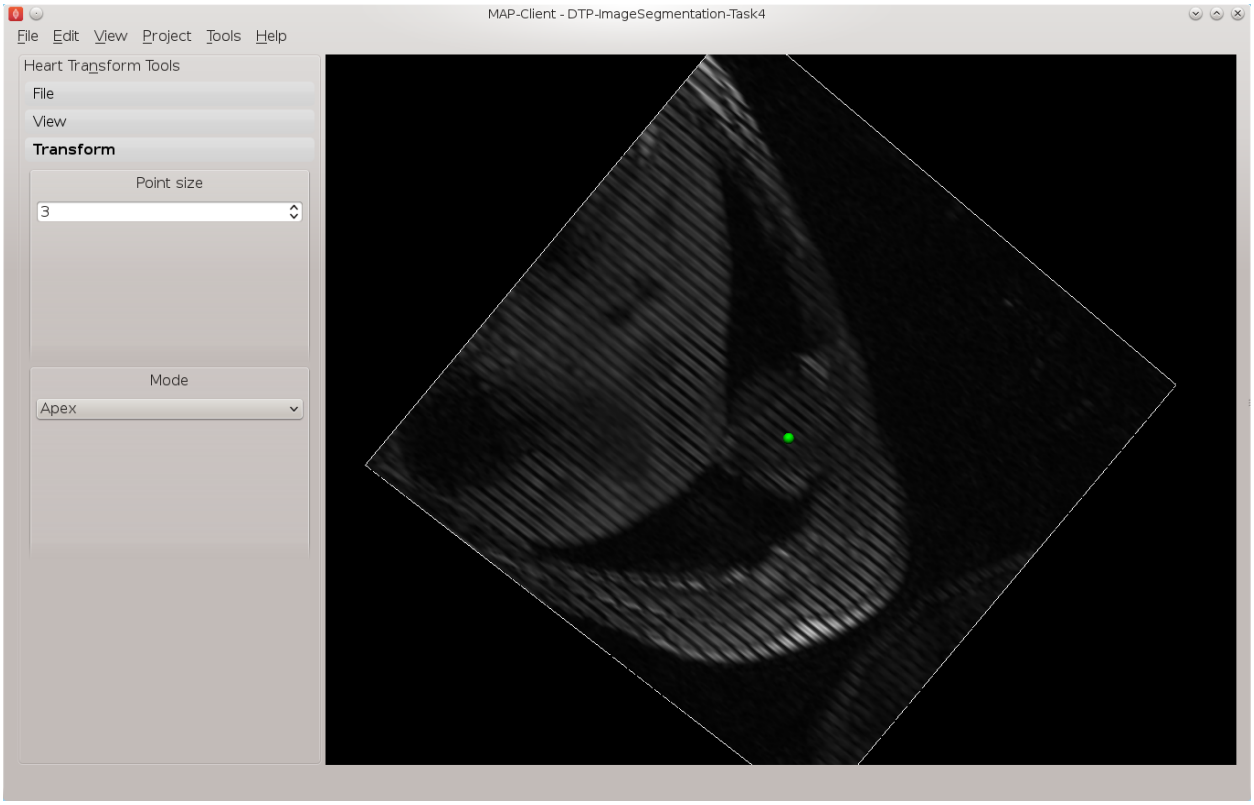

Fig. 2.8: Apex point position in the left ventricle

This tutorial currently concentrates on geometric fitting, but gives overviews of the other topics. It uses the breast surface as the model for fitting, which is appropriate as customising breast models to individuals is needed to simulate deformation with change in pose, and this in turn is helps co-locate regions of possible cancerous tissue from multiple medical images each made with different imaging devices which necessarily use different body poses.

Mesh Creation

We are concerned with constructing models consisting of 'finite elements' – simple shapes such as triangles, squares, cubes etc. – which join together to form a 'mesh' which covers the body and describes its topology. Over the elements of the mesh we interpolate coordinates (and eventually other fields of interest) to give the model its 3-D shape and location. Usually mesh creation involves creating the elements and specifying at least initial coordinates for the model.

Common methods for creating the finite element mesh include:

1. Automatic mesh generation to boundaries described by segmented edges, point clouds or CAD models. Automated algorithms are usually limited to creating triangle (in 2-D) or tetrahedron (3-D) elements, with linear or quadratic basis functions only.
2. Generating part or all the topology from simple, standard shapes such as plates, blocks and tubes, then relying on fitting to the geometry. The models used in the geometric fitting steps below were all generated from a plate topology and the tutorial involves fitting their geometry.
3. Manually building elements by selecting/creating corner points ('nodes') in the correct order. To ease creating a valid model, tools can assist by locking on to points from a surface or point cloud segmented from real medical images.

Surface model can be automatically extruded to 3-D models (triangles become wedges, squares become cubes). Also,

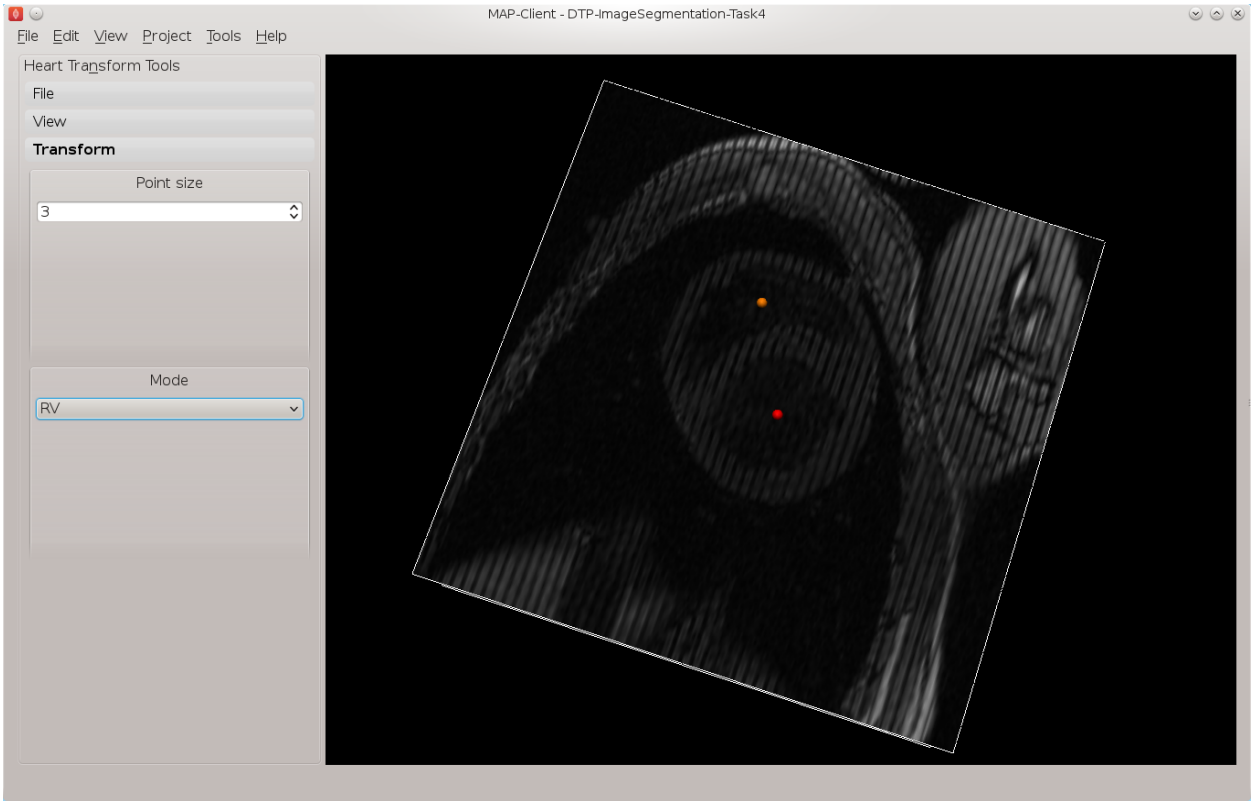

Fig. 2.9: Placement of the base point and RV point

different, higher-order basis functions can be used over the same topology to give more degrees of freedom in the model. This is particularly important for studying the human body as it contains few straight lines. In many models of body parts we employ C_1-continuous cubic Hermite basis (interpolation) functions to model their inherent smoothness, and the following breast fitting examples demonstrate this. There is a downside to using Hermite bases: it is painstaking work to properly connect the mesh in areas where the topology is non-trivial, such as apex points, armpits etc.

Mesh creation is a very involved topic; one needs to consider favourable alignment of elements with expected material behaviour, having sufficient density of elements to describe the problem with sufficient accuracy (the fitting examples below employ 2 different sized meshes to give some indication of the importance of mesh refinement) but overall one wishes to minimise the total number of degrees of freedom to reduce computation time. This tutorial does not go further into mesh creation at this time.

Geometric Fitting

This tutorial concentrates on directly fitting generic models to data point clouds obtained from an earlier segmentation or other digitisation step. Other types of fitting not covered include:

- Fitting to modes from a Principal Component Analysis, where the variation in geometry over a population is reduced to combinations of a small number of significant mode shapes and lesser modes are discarded;
- Host-mesh fitting where the body is embedded in a coarse, smooth 'host' mesh, data is used to morph the host mesh and the embedded 'slave' mesh is moved with it.

In many cases the above methods are used as a first step to get a close approximation before direct geometric fitting.

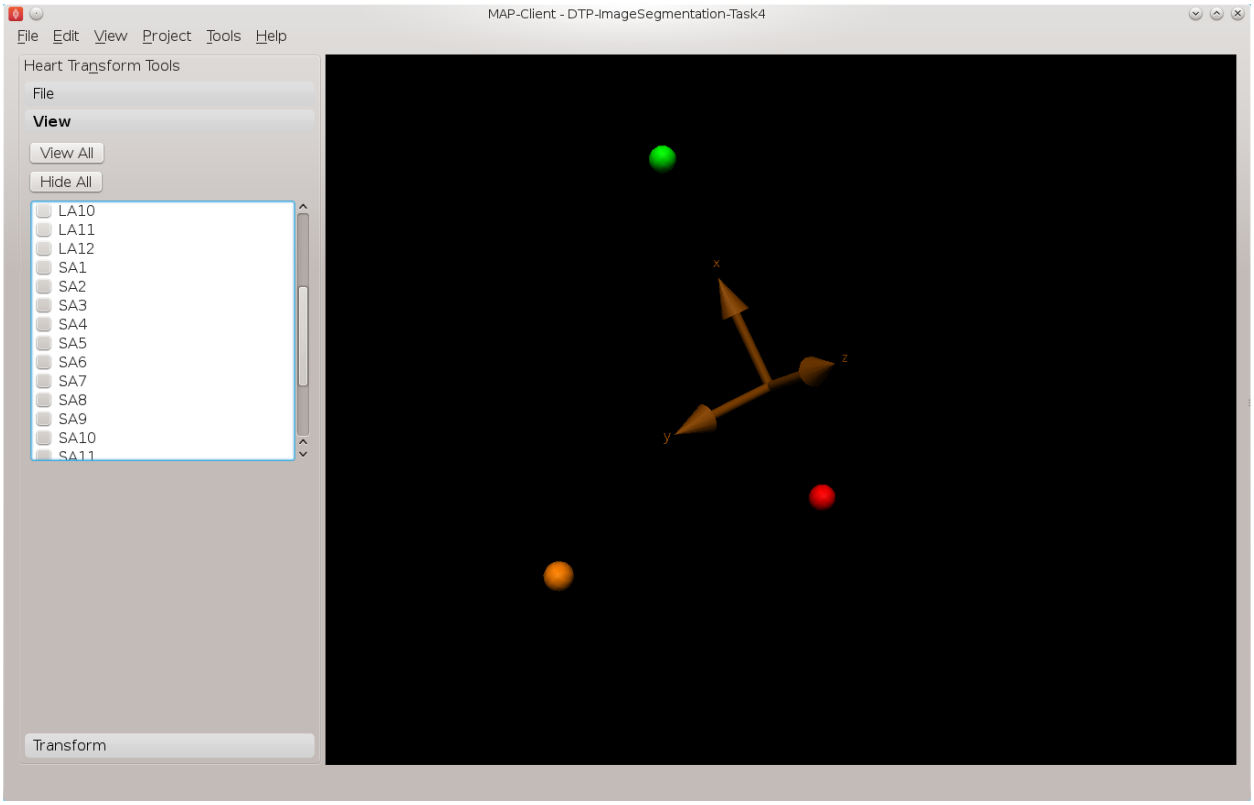

Fig. 2.10: Axes glyph representing the heart coordinate system

Smoothfit Tool

Fitting the geometry of a model to a point cloud consists of 3 steps:

1. Model alignment
2. Data point projection onto the model
3. Performing the fit

This tutorial uses the 'smoothfit' MAP client plugin for interactive fitting with these 3 steps. The inputs to smoothfit in a workflow are a model file and a point cloud file (each currently limited to EX or FieldML formats that can be read by OpenCMISS-Zinc). The workflow in the MAP client is shown in Fig. 2.11, and requires only the input files to be specified (and workflow step identifiers to be named):

When the workflow is executed, the smoothfit interface is displayed showing the model as a semi-transparent surface and the point cloud as a cloud of small crosses. The initial view is of the '1. Align' step, shown in Fig. 2.12.

In any of the views you may rotate, pan and zoom the view using the standard OpenCMISS-Zinc controls of left, middle and right mouse button drag, click 'View All' to recentre the view and click 'Done' to close the workflow step.

The first step in fitting is to scale the model and bring it into alignment with the point cloud; this is done so that the projections are as close and consistent as possible, as described below. To scale and align the model in this step, hold down the Ctrl key as you left, middle and right mouse button drag in the window: this moves the model relative to the data cloud. Be aware that rotation is a little difficult and may take practice. Other controls include alignment reset, auto centre and the Load button which will load a saved alignment. (The Save button can be disabled in the smoothfit configuration so tutorialsters don't accidentally wipe the good one that is saved for progressing to the next step!)

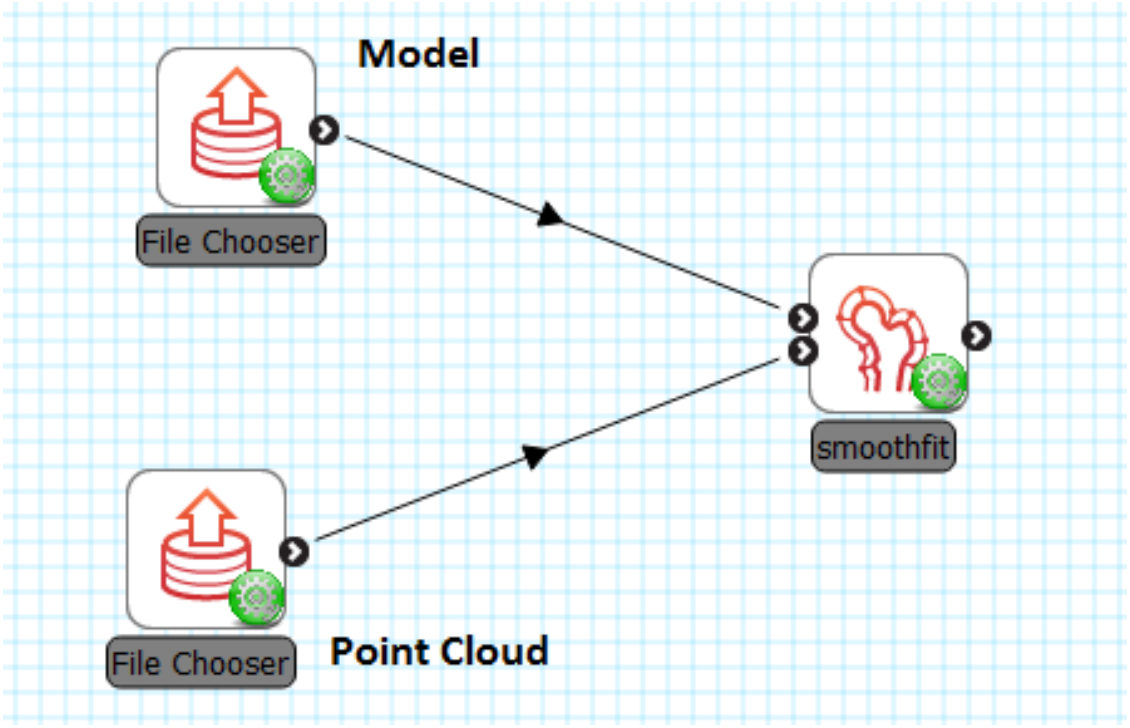

Fig. 2.11: Geometric fitting workflow in the MAP client framework.

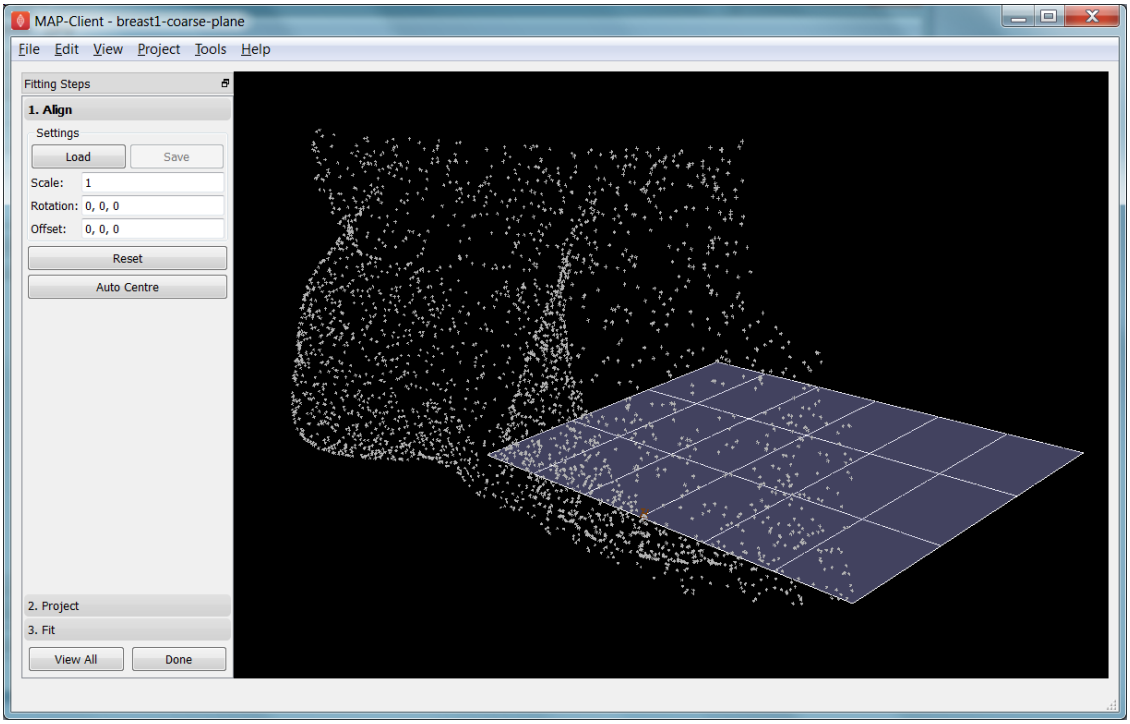

Fig. 2.12: Fitting step 1: Model alignment.

Often the shape of the model and point cloud make it pretty clear where to align to. Note that this tool uses manual alignment, but other tools may make it automatic (based on shape analysis) or semi-automatic (e.g. by identifying 3 or more points on the data cloud as being key points on the model, and automatically transforming to align with them).

The second step in fitting is to project the data points onto the nearest location on the aligned mesh. Switch to the '2. Project' page of the tool bar, then click on the 'Project Points' button to get the view in Fig. 2.13. When projections have been calculated, the view changes to show error bars between the data points and their projections, coloured by magnitude, plus the on-screen display of mean and maximum error.

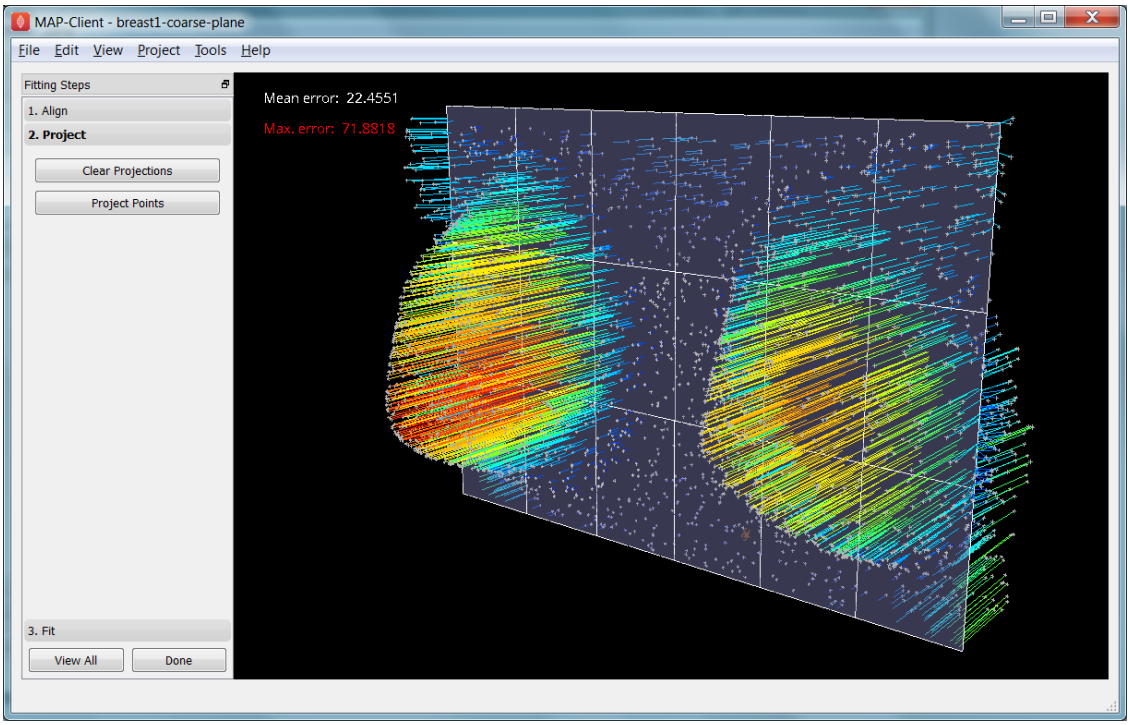

Fig. 2.13: Fitting step 2: Projecting data points onto the model.

The key point is that the projections are what the fitting aims to minimise, and if they don't agree on where a point on the mesh should move to, the fit will have problems. It's good if the projection lines are short and/or near parallel, and it's bad if they cross over each other. Often you will see some projections that are very long and probably erroneous; in fitting applications these may be eliminated or weighted lower to have less effect on the solution, but this is not offered in smoothfit yet.

Switch to the next step '3. Fit' to configure and perform the fit. This is where fitting becomes less a science and more a dark art. The normal fit adjusts the coordinates to minimise the error bars; clicking the 'Perform Fit' button performs a single iteration and it may take multiple iterations to get close to the data. Fig. 2.14 shows what the view looks like after a couple of iterations of fitting.

Note that the projections are not recalculated during the fitting, but you can switch back to step 2, reproject and then fit again. Switching back to the Align step clears the fitted solution.

The penalty values allow you to smooth the fit by penalising particular deformations. The strain penalty limits excessive strain in the model so in the absense of data (or in the presence of noisy data) the solution errs towards one with minimal deformation from the initial aligned state. The edge discontinuity penalty is only useful for non-C_1-continuous coordinate fields such as the linear dome example later. Penalties always increase the data point projection error (in a least squares sense, which is the solution method used in the fitting), but generally give a much more attractive result. Penalty values should be adjusted in orders of magnitude until a likeable result is obtained, then fine-tuned. It is often better to use stiffer (higher penalty) values for initial iterations to prevent waviness from developing in the

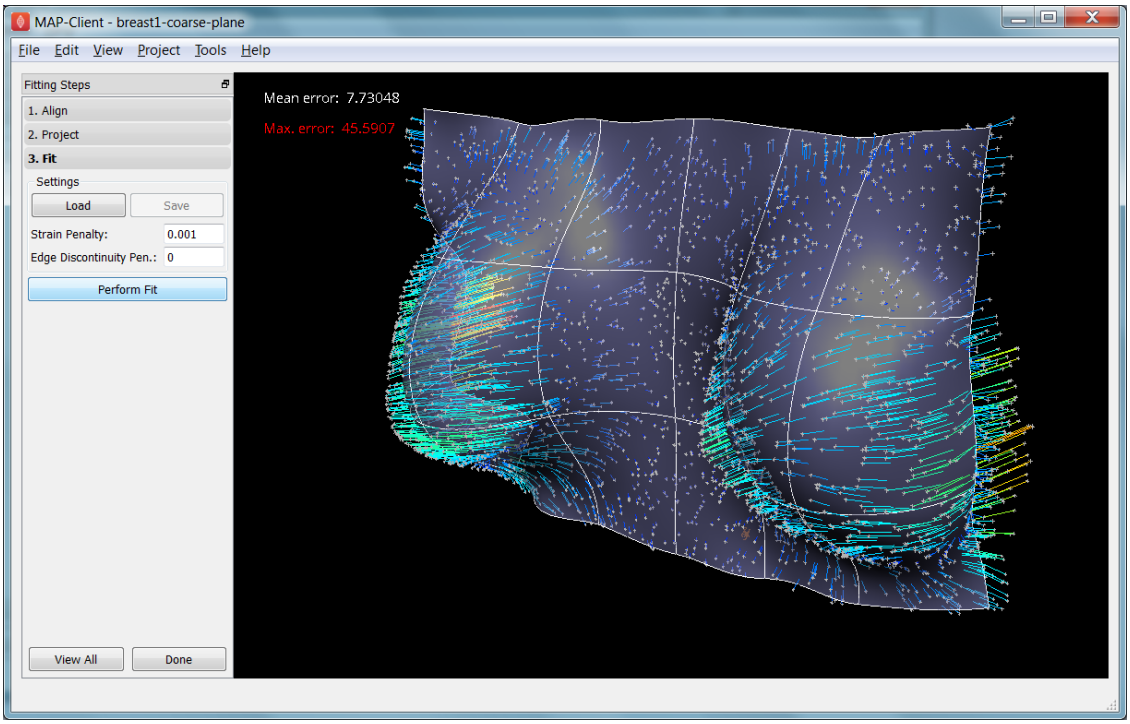

Fig. 2.14: Fitting step 3: Perform the fit

mesh, then dropping for a final iteration. As for the align step, you can load and save (latter if enabled) the fitting options.

Note that smoothfit does not yet offer a curvature penalty which is one of the most powerful tools for dealing with noisy or sparse data. Using the strain penalty is the next best thing but isn't as good at dealing with excessive waviness in the solution, particularly since higher values capable of helping the waviness may considerably reduce the accuracy of the fit. This shortcoming will hopefully be rectified soon.

The following tutorial tasks each have a workflow associated with them which should be run in the usual way.

Task 1: Coarse plate model fitted to breast data

Open the *DTP-ModelBuilding-Task1* workflow and execute it. The breast data was obtained in 'prone' pose (hanging down) as done in MRI scans; this is also the simplest pose to digitise and fit to. Try manually aligning the surface with the breast data by Ctrl-clicking the left, middle or right mouse button and dragging to rotate, pan or scale the model. Project points and attempt to fit without any smoothing parameters. It takes several seconds to perform the fit: be patient! Try multiple fit iterations until the solution is stable. Re-project and try again.

The result without smoothing even for this example with a coarse mesh and a relatively large number of high quality data points is quite wavy, particularly around the edges. It also has some unusual depressions about the front of the breasts which is not really representative of the data cloud in general.

For a second exercise we'll use a set sequence to obtain a good fit. Switch to the Align page to reset the fit, click on 'Load' to load a good alignment, project points, switch to the Fit page and click 'Load' to load a strain penalty of 0.001. Perform the fit 2 times which gets fairly close to the data points. Next switch to the Project page and reproject points. On the fit page, lower the strain penalty to 0.0001 and fit once more. The error bars almost disappear over most of both breasts (but can't over the edges where the outlying points are - ignore these). **Write down the mean error for comparison later.** Have a look at the tips of the breast to see where the fit has not been great - this is where

the limited number of elements may make the model unable to fit the data (but it is also not helped by the 'pull' of the outlier points).

Next go back to the Align page to reset the fit, then re-project points. Try a much higher strain penalty (e.g. 0.01) and see how it limits the possible deformation (after several iterations) - this is what is considered a 'stiff' model.

Try fitting with poor initial alignment to see what happens.

Task 2: Fine plate model fitted to breast data

Open the *DTP-ModelBuilding-Task2* workflow and execute it. It has the same data point cloud as the first task, but has a mesh with more than twice as many elements and approx. twice as many parameters, so it is more able to attain a close fit with the data, but takes longer to solve.

Try some of the exercises from Task 1 with this model. With more elements the model is more susceptible to wavy solutions so applying appropriate smoothing penalties is more critical.

When performing the second exercise from Task 1, iterate 3 times with the initial strain penalty of 0.001, then re-project points and fit with a strain penalty of 0.0001. Note down the mean error which should be about 2/3 of the value from Task 1. More importantly, zoom in on the tips of the breasts to see that the fit is much better there. (Remember the actual mean error is swayed by the outlier points around the edge of the model.)

Task 3: Coarse breast model fitted to breast data

Open the *DTP-ModelBuilding-Task3* workflow and execute it. In this example the initial model is more breast-like in shape so when well-aligned the amount of fitting needed is reduced. You should be able to fit it with the lower strain penalty of 0.0001 directly in 2 iterations. Since the initial model is already so close, deformations will not be as great to get a close fit.

Task 4: Fine breast model fitted to noisy data

Open the *DTP-ModelBuilding-Task4* workflow and execute it. This example uses a fine model with a breast-like shape, however random offsets up to +/- 5mm have been added to all data points. With a large enough number of data points the effect of randomness is diminished however in small areas the randomness can introduce waviness to the solution, so smoothing penalties must be applied.

Try fitting the model without any strain penalty, and fit with several iterations to see the waviness. Reset the fit and try with the regime from task 1: 2 iters at strain penalty 0.001, re-project, 1 iter at strain penalty 0.0001. The overall result is a good fit but there is unattractive waviness on the chest area. If a curvature penalty were available, these issues with noisy data could be better controlled.

Because of the random noise the mean error will never get very low, but the average fit of the breast surface can be a reasonable 'best fit'.

Task 5: Fine breast model fitted to sparse, noisy data

Open the *DTP-ModelBuilding-Task5* workflow and execute it. This example uses only 10% of the data points from the previous tasks and adds +/- 3mm error to each point.

Try fitting as before. The effect of sparse data with random noise makes it even harder to obtain a close fit. Using the successful regime from Task 1 gives a result that is quite wavy after the final less-stiff fitting. Curvature penalties would greatly assist such models.

Task 6: Bilinear model fitted to point cloud

Open the *DTP-ModelBuilding-Task6* workflow and execute it. This example has a bilinear mesh and needs no alignment with the data point cloud.

Project points and fit with all smoothing penalties set to zero. Rotate the result to see that it has developed a 'ridge' along one side, and the under-constrained corner elements distort unacceptably. Reset the fit, reproject and fit with the 'edge discontinuity penalty' set to 1. The result is much smoother. This penalty discourages solutions with differences in surface normals across edges of the mesh. Since the mesh uses bilinear interpolation, exact satisfaction of this condition cannot be met, nevertheless it minimises it as much as possible, and in particular it evens out this discontinuity since it is minimised in a 'least squares' sense.

Experiment with different (much lower, much higher) edge discontinuity penalties to see how the fit is affected. Try combining with strain penalty values.

Material Field Fitting

In addition to geometry, bioengineering models often need to include spatially varying data describing the alignment of tissue microstructures, concentrations of cell types, or other differences in material properties. In heart and skeletal muscle, fibre orientations must be described over the body to orient their anisotropic material properties. Similarly, Langer's lines affect properties of the skin, and collagen orientations within other tissue can affect material behaviour.

Each of these properties can be described by spacially-varying fields which interpolate the property of interest over the same elements the coordinates are defined on.

This topic is not covered further in this example, but the concepts of creating and fitting such fields are similar to geometry: one must define the interpolation of the values over the mesh, and fit the field to data obtained from imaging or other techniques. The difference lies mainly in that the data is not coordinates, but orientations when fitting fibres, known concentrations at points when fitting points etc.

A similar process is often used to obtain solution fields from results. Often the solution technique produces outputs with high accuracy only at certain points in the model. With the Finite Element Method, for example, stress is of highest accuracy at the Gauss points, and fitting can be used to give a better idea of these solution field values away from Gauss points.

2.3 Indices and tables

- genindex
- modindex
- search

2.4 Computational Physiology - Simulation

This tutorial was created as part of the Computational Physiology module in the MedTech CoRE Doctoral Training Programme. The tasks presented in this tutorial are designed to make the reader aware of common key skills required for the application of mathematical models in computational simulation experiments in the context of computational physiology. We will demonstrate these skills across a range of spatial scales and numerical methods.

Contents:

2.4.1 Integrating systems of differential equations

In this tutorial we look into the integration of systems of differential equations, using example models from the domain of computational physiology. We also examine the the main control parameters that can have a significant impact on the performance and accuracy of a given method.

Numerical integrators

Euler method

The Euler method to integrate a system of ordinary differential equations is probably the most well known method, particularly popular given its simplicity. Due to its simplicity it is, however, usually not the most appropriate method to use when performing simulation experiments with complex models of biological systems. It is, however, a very useful example to use to demonstrate key concepts that apply to most numerical integration methods.

Given the initial value problem

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0, \tag{2.1}$$

we now want to approximate $y(t)$ over some interval $t_0 \rightarrow t_{final}$. The Euler method approximates $y(t)$ by dividing the interval into a series of steps, of size h, and stepping through the interval. One step of the Euler method from t_n to $t_{n+1} = t_n + h$ is given by

$$y_{n+1} = y_n + hf(t_n, y_n). \tag{2.2}$$

The value of y_n is an approximation of the solution of the ODE (2.1) at time t_n: $y_n \approx y(t_n)$. The Euler method proceeds through the interval in constant steps of size h until $t_n = t_{final}$ and the integration is complete.

Task 1 - the effect of step size As you'd imagine, the size of h in the Euler method is crucial to the successful application of the method. For the successful (although perhaps inaccurate) integration of a typical physiological model (see the Physiome Repository for a collection of examples), h can be so small that the computational cost of performing the simulation is very high. In some cases, mathematical models may be unsuitable for integration by Euler method, regardless of how small the step size is reduced to.

In the follow task, we investigate the effect of altering the size of h on two separate simulation experiments. The first looks at a simple mathematical model where the experiment could be replicated with pen and paper, while the second looks into the application of Euler's method to a biophysical model of cellular electrophysiology.

The following task assumes familiarity with the *virtual machine*. In this task we use the trivial model:

$$\begin{aligned} x(t) &= sin(t), \\ y'(t) &= cos(t) \quad \text{with} \quad y(0) = 0. \end{aligned} \tag{2.3}$$

As you can see from (2.3), if correctly integrated $x(t)$ and $y(t)$ should be identical. We now use this model to demonstrate the effect of step size (h) on simulating this model using Euler integration over the interval $0 \rightarrow 2\pi$.

1. Run MAP Client, choose *File* → *Open* and select `HOME/projects/mapclient-workflows/DTP-Simulation-Task1`
2. This simple workflow should look similar to Fig. 2.15. The workflow is pre-configured so there is no configuration required.

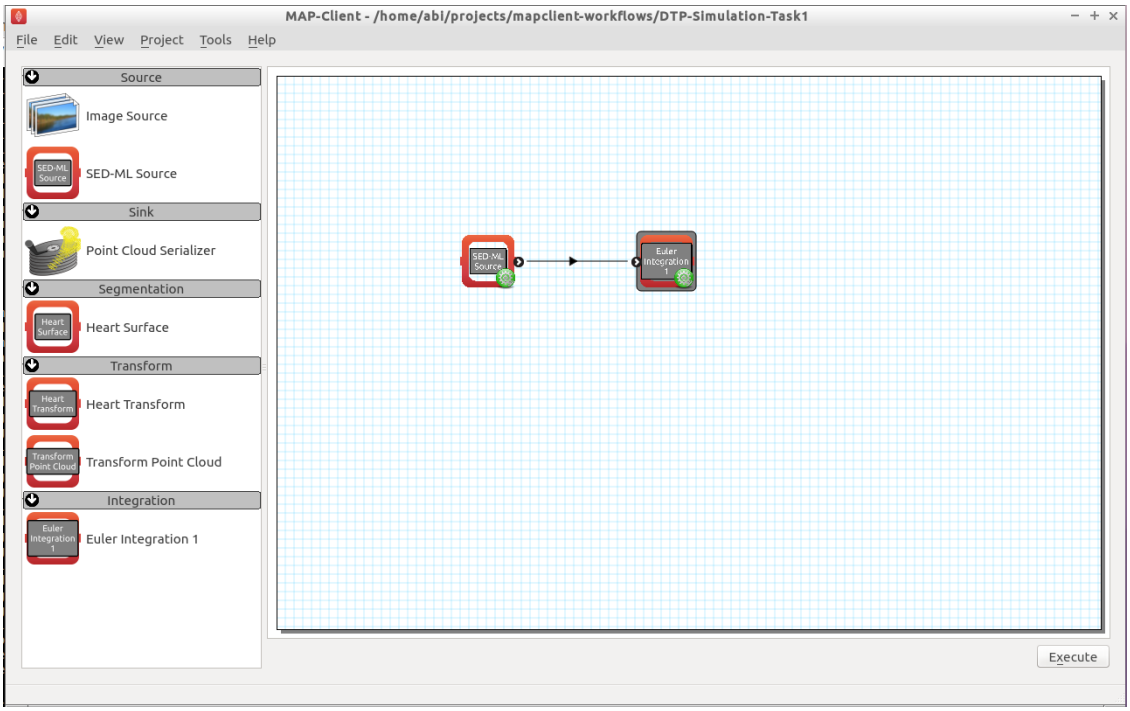

Fig. 2.15: The first Euler example as loaded.

3. Click the *Execute* button and you should get a widget displayed as per Fig. 2.16.
4. As described in Fig. 2.16, multiple simulations can be performed with varying values for the step size, h. Shown in Fig. 2.17 you can see that as h reduces in size, the approximation of the model (2.3) by integration using the Euler method gets more accurate.

CVODE

From the Sundials suite of tools, CVODE is a solver for stiff and nonstiff ordinary differential equation (ODE) systems (initial value problem) given in explicit form in (2.1) above. CVODE is widely regarded as one of the gold standard implementations of a robust and flexible numerical integrator. One of the advantages of CVODE over Euler's method is that it makes use of adaptive stepping over the interval of integration - rather than taking fixed sized steps through time, for example, CVODE will determine how quickly things are changing and adjust the size of the step accordingly.

Task 2 - fixed vs adaptive stepping In this task we examine the limitations and the computational costs associated with a fixed step method (Euler) compared to an adaptive step method (CVODE). We use the recent biophysically based mathematical model of unitary potential activity in interstitial cells of Cajal. The interstitial cells of Cajal (ICC) are the pacemaker cells of the gastrointestinal tract and provide the electrical stimulus required to activate the contraction of smooth muscle cells nescessary for the correct behaviour of the GI tract. This particular model was developed by scientists at the Auckland Bioengineering Institute to investigate a specific hypothesis regarding the biophysical mechanism underlying the pacemaker function of ICCs.

1. Run MAP Client, choose *File* → *Open* and select *HOME*`/projects/mapclient-workflows/DTP-Simulation-Task2`
2. This simple workflow should look similar to that used in task 1 above (Fig. 2.15). The workflow is pre-configured so there is no configuration required.
3. Click the *Execute* button and you should get a widget displayed as per Fig. 2.18.

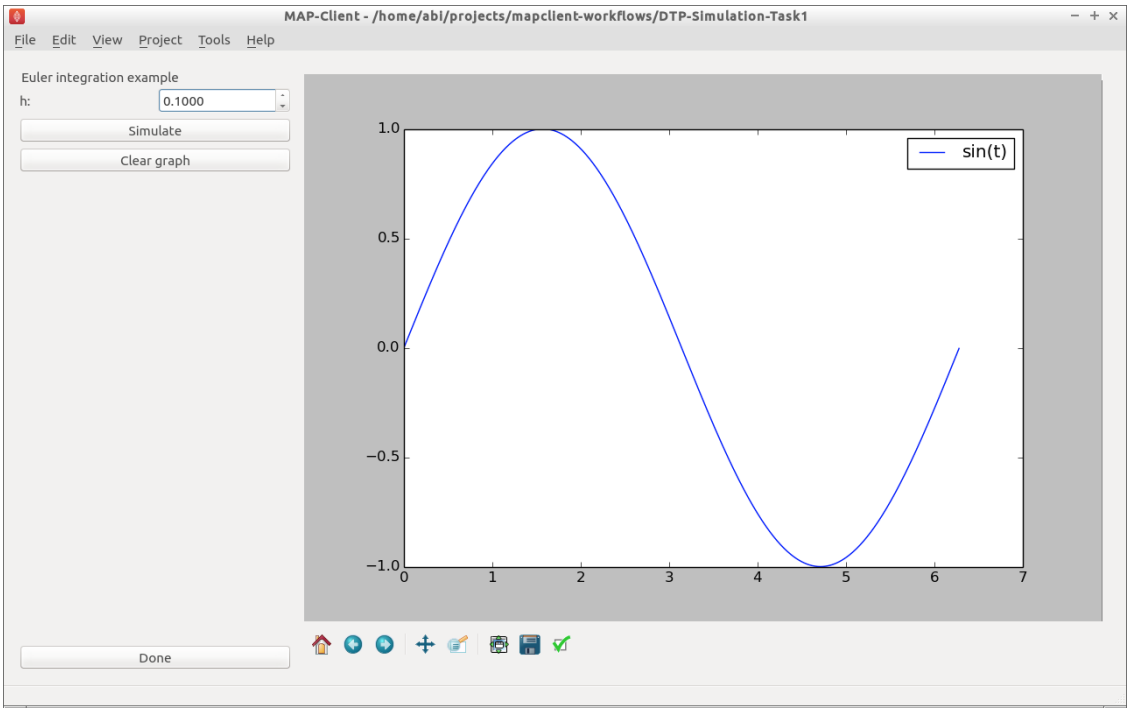

Fig. 2.16: The cool Euler integrator interface. In this simple interface, you will see the standard sine function, $sin(t)$, plotted in the right hand panel. The toolbar under the plot is self-explanatory, but provides access to some nifty features. At the top of the left hand panel you will see the control to set the Euler step size for this model, h. The *Simulate* button will execute the Euler integration of the model (2.3) and plot the result on the plot to the right. This can be repeated with various values of h. The *Clear graph* button will, surprisingly, clear the current simulation results from the plot panel. The *Done* button will drop you back to the work-flow diagram, where you can get back to the plot by executing the work-flow once more.

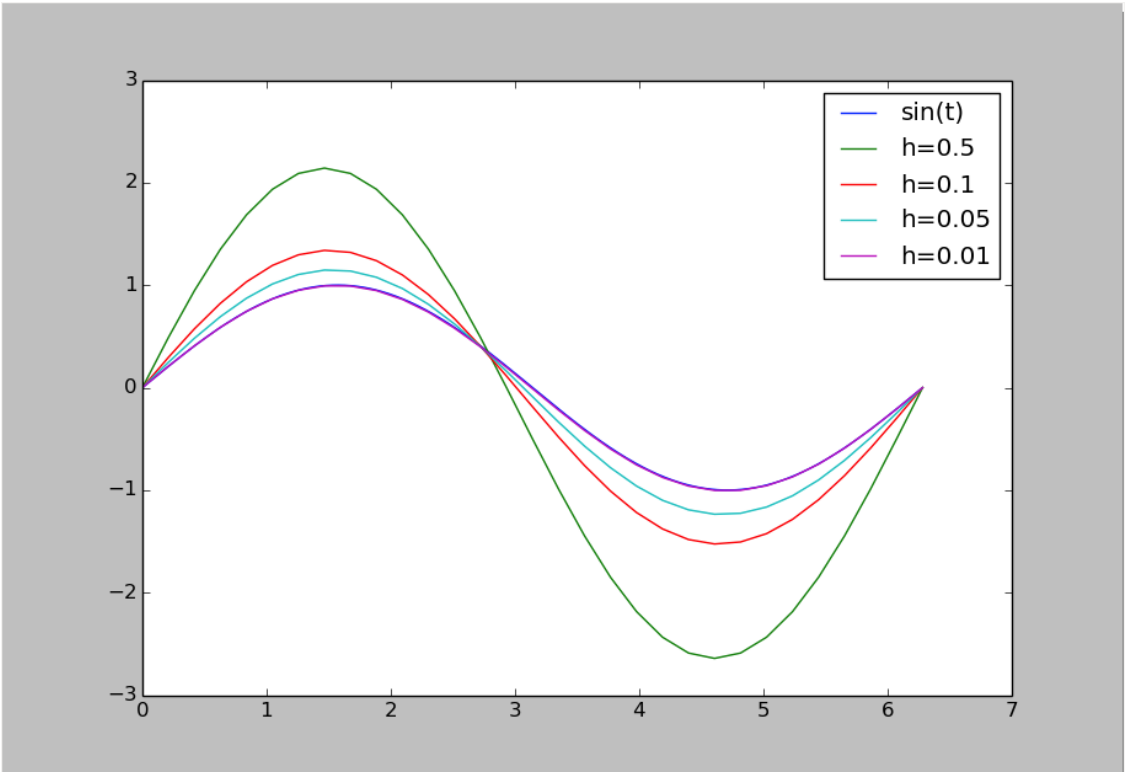

Fig. 2.17: Simulation results demonstrating the effect of step size, h, on the accuracy of Euler's method in approximating the solution of (2.3).

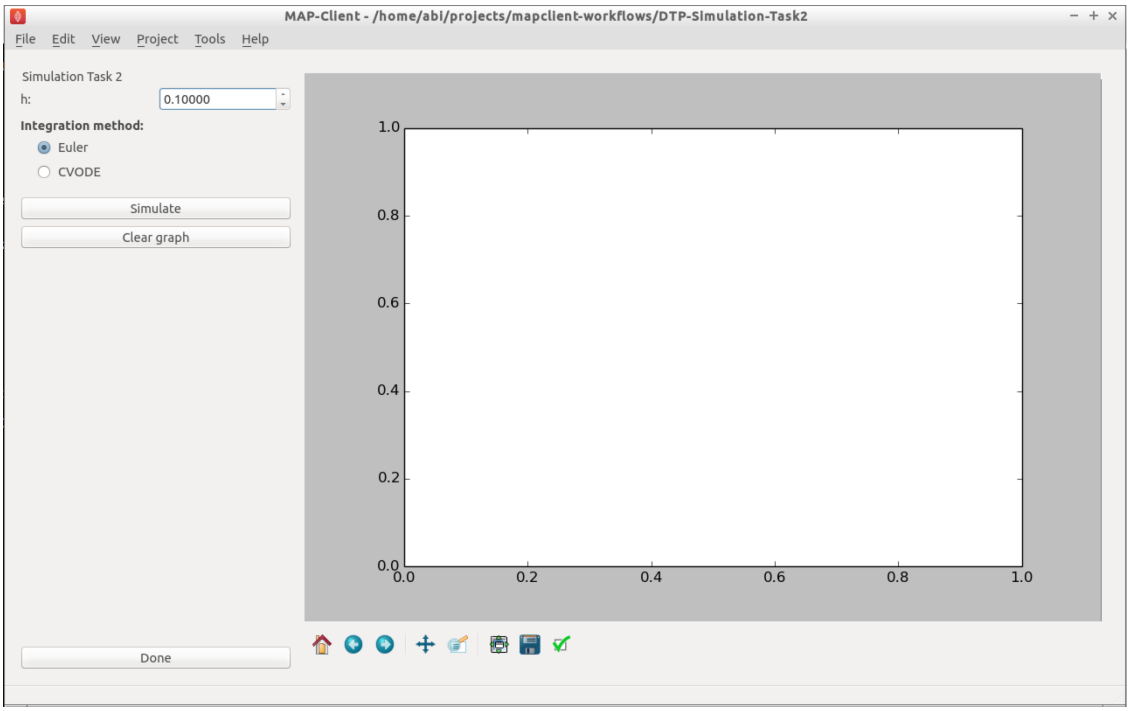

Fig. 2.18: The user interface in this task is similar to that described in Fig. 2.16, and the common elements behave the same. In addition there is the ability to choose either the Euler or CVODE numerical integration methods. As the CVODE method is an adaptive stepping method, the value of h is used to limit the maximum step size that the algorithm will use, with $h = 0$ indicating the maximum step size is unlimited.

4. You can now attempt to find the value of h that will enable the Euler integration method to successfully reproduce the result shown in Fig. 2.19. Hint: this particular model has characteristics that make it very difficult to simulate using a fixed method like Euler.
5. You can now compare the results obtained using the CVODE integration method with those from your successful Euler simulation. In the plot legend you will see the `time` value - this is a rough measure of the length of time taken for the given simulation. Hopefully you will see that the adaptive stepping CVODE method performs better than the Euler method.

Task 3 - error control In addition to providing adaptive stepping, CVODE is also a very configurable solver. Beyond the maximum step size explored above, a further control parameter of that is often of interest are the tolerances used to control the accumulation of error in the numerical approximation of the mathematical model. This tolerance specifies how accurate we require the solution of the integration to be, and the value used can be very specific to the mathematical model being simulated. In task 2 above, we used a tolerance of 1.0e-7. Depending on the behaviour of your mathematical model, you may need to tighten (reduce) or loosen (increase) the tolerance values, depending on the specific application the model is being used for. Here we explore the effect of the tolerance value on the ICC model introduced above.

1. Run MAP Client, choose *File* → *Open* and select *HOME*`/projects/mapclient-workflows/DTP-Simulation-Task3`
2. This simple workflow should look similar to that used in task 1 above (Fig. 2.15). The workflow is pre-configured so there is no configuration required.
3. Click the *Execute* button and you should get a widget displayed as per Fig. 2.20.
4. You can now investigate the effect of changing the tolerance value and maximum step size on the simulation result. Not all combinations will successfully complete. Example results are shown in Fig. 2.21.

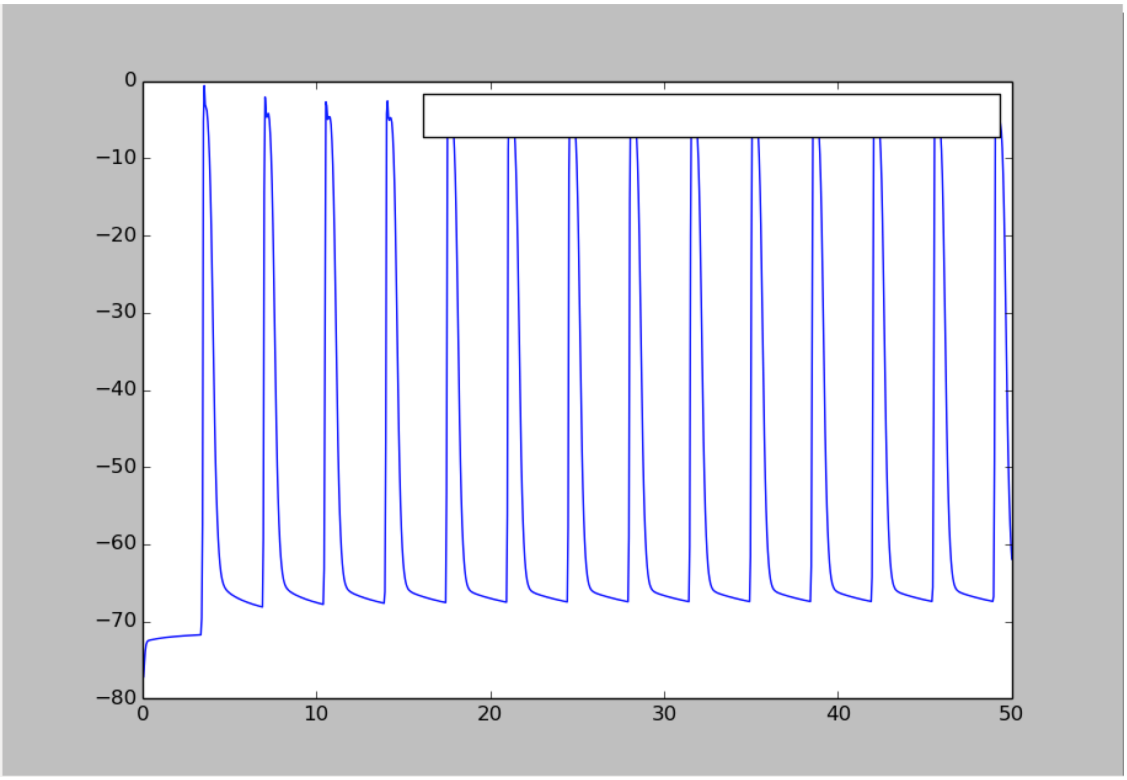

Fig. 2.19: Simulation results for a successful integration of the ICC model using the Euler integration method. Determining a suitable Euler step size, h, to recreate these results is an exercise for the reader.

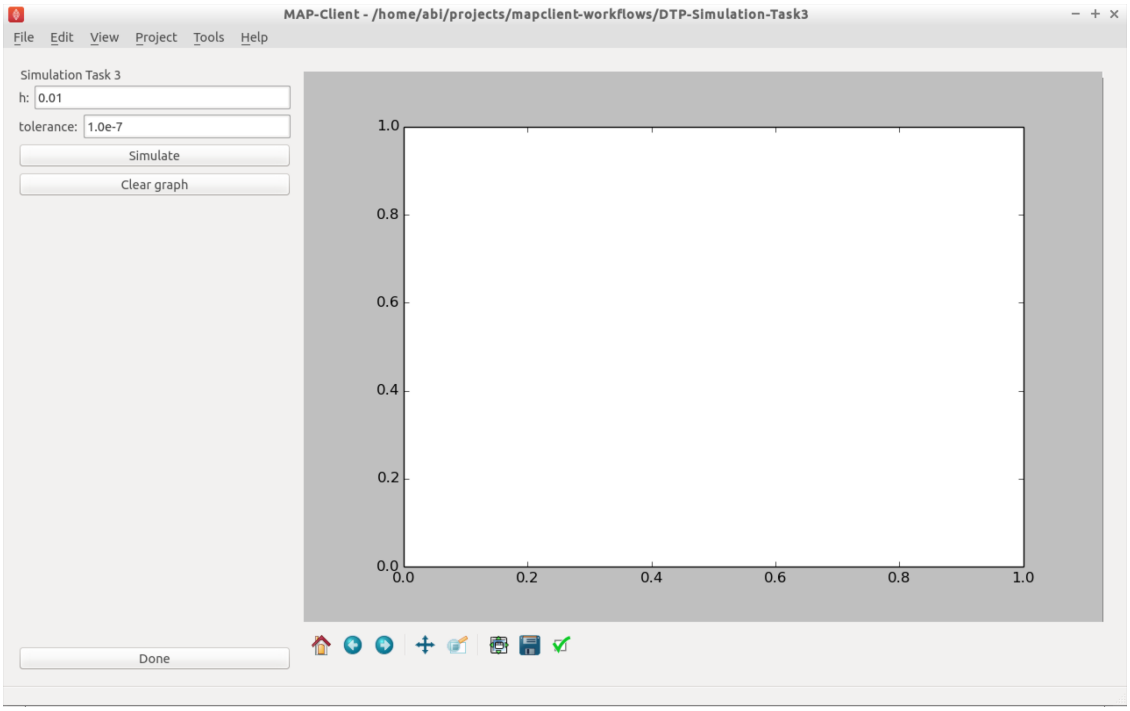

Fig. 2.20: The user interface in this task is similar to that described in Fig. 2.16, and the common elements behave the same. We now are only using the CVODE integration method so h is the maximum step size with $h = 0$ indicating an unlimited step size. The tolerance value for the simulation can also be edited in this interface.

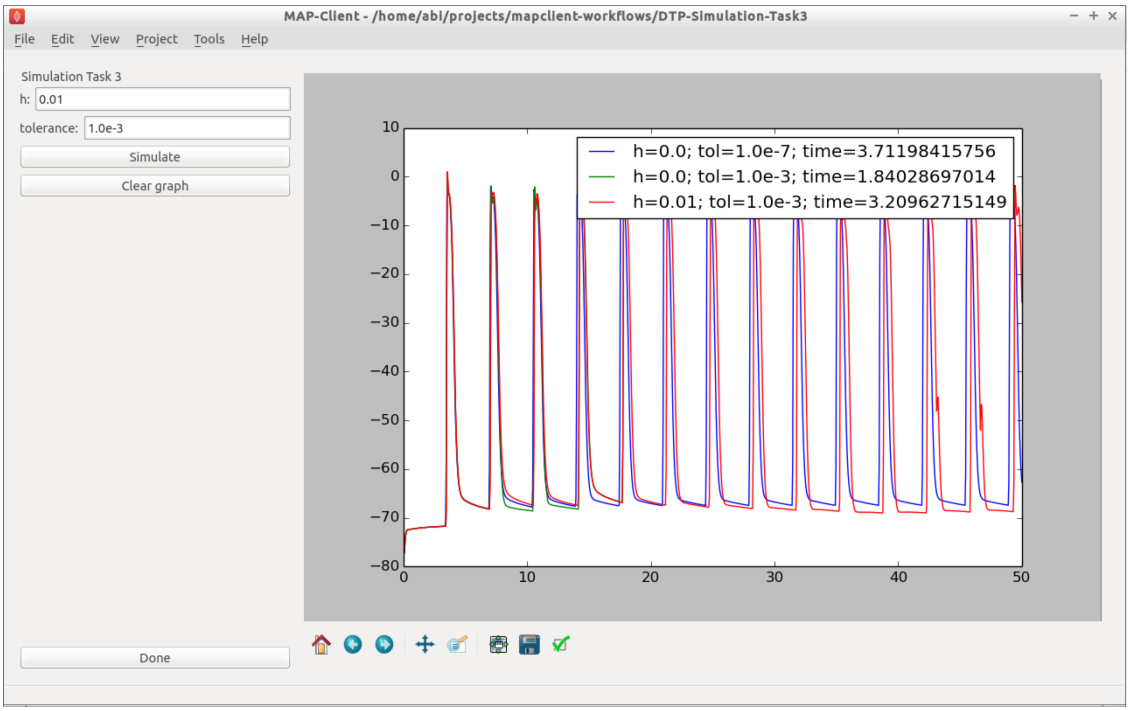

Fig. 2.21: Simulation results for a selection of simulations of the ICC model using various configurations of the CVODE integratior.

2.4.2 Biological Complexity

Mathematical models are a useful tool for investigating biological systems, but such models only ever approximate the actual biological system and are tuned to specific applications. Depending on your specific requirements, the inclusion of more or less biological complexity may be necessary.

Switch to Powerpoint slides :)

2.4.3 Multiscale simulation

Switch to Powerpoint slides :)

2.4.4 TL;DR

The quick highlights in this tutorial.

1. Introduce the concept of numerical integration of systems of differential equations.
   (a) Two specific algorithms presented: *Euler's method* and *CVODE*.
   (b) Investigate the impact of choosing the correct algorithm control parameters (*Task 1 - the effect of step size*).
   (c) Observe the limitations and the computational cost associated with numerical method choices (*Task 2 - fixed vs adaptive stepping*) and algorithm parameterisation (*Task 3 - error control*).
2. How much biological detial is too much?
3. Integrating cellular models into organ models.

2.5 Computational Physiology - Visualisation

Contents:

2.5.1 Visualisation

This tutorial was created as part of the Computational Physiology module in the MedTech CoRE Doctoral Training Programme. The tasks presented in this tutorial are designed to make the reader aware of common key visualisation skilss used in the context of computational physiology. We will demonstrate these skills across a range of spatial scales and visualisation techniques.

Overview

A picture is worth a thousand words, as the saying goes so a four-dimensianal visualisation should be worth somewhere near one billion if not more. That many words is difficult to comprehend let alone worry about their meaning. So in visualisation we are trying to illustrate the complexities of an underlying biophysical data in a manner that is consumable by the target audience. This tutorial is about the visualisations that we are currently able to produce and the aspects that are relevant to creating a visualisation from general or specific comsumption.

Task 1

In this task we shall investigate the capabilities of the visualisation tool on some existing models. First we shall start with a script that will define a model and some initial graphics so that we can actually see something. Start the view application with the following command (from the HOME directory):

```
python ~/projects/opencmiss-software/zinc-software/zincv/src/zincview.py
```

From the 'Model' toolbox click the load button and choose the heart.zincview.py file from the data directory. The result should be something similar to Fig. 2.22.

Manipulating the View

We can manipulate the view with mouse actions, clicking and dragging with the mouse will transform the view. The following table describes the mouse button to the transformation applied.

Mouse Button	Transformation
Left	Tumble
Middle	Pan/Translate
Right	fly-zoom
Shift+Right	Camera zoom

When we transform the view with the mouse you can see from the view pane the change in the camera controls (see Fig. 2.23).

Everything that can be seen is set inside what is known as the viewing frustrum. The frustrum is a pyrimad shape with the top chopped off. We look at the frustrum looking from the top through to the base of the pyramid. The top of the pyramid is known as the near plane and the bottom is known as the far plane. Ideally we want to position the near plane just in front of everything that should be visible and position the far plane just behind everything that should be visible. The better job we do of this the better the hidden graphics removal will work, this is important when making large high-quality images.

Position the mouse over the near/far sliders [1] and use the mouse wheel to affect it - this will move the near/far planes in the scene. When the near plane passes past the nearest object in the scene you will see part of it disappear, this is

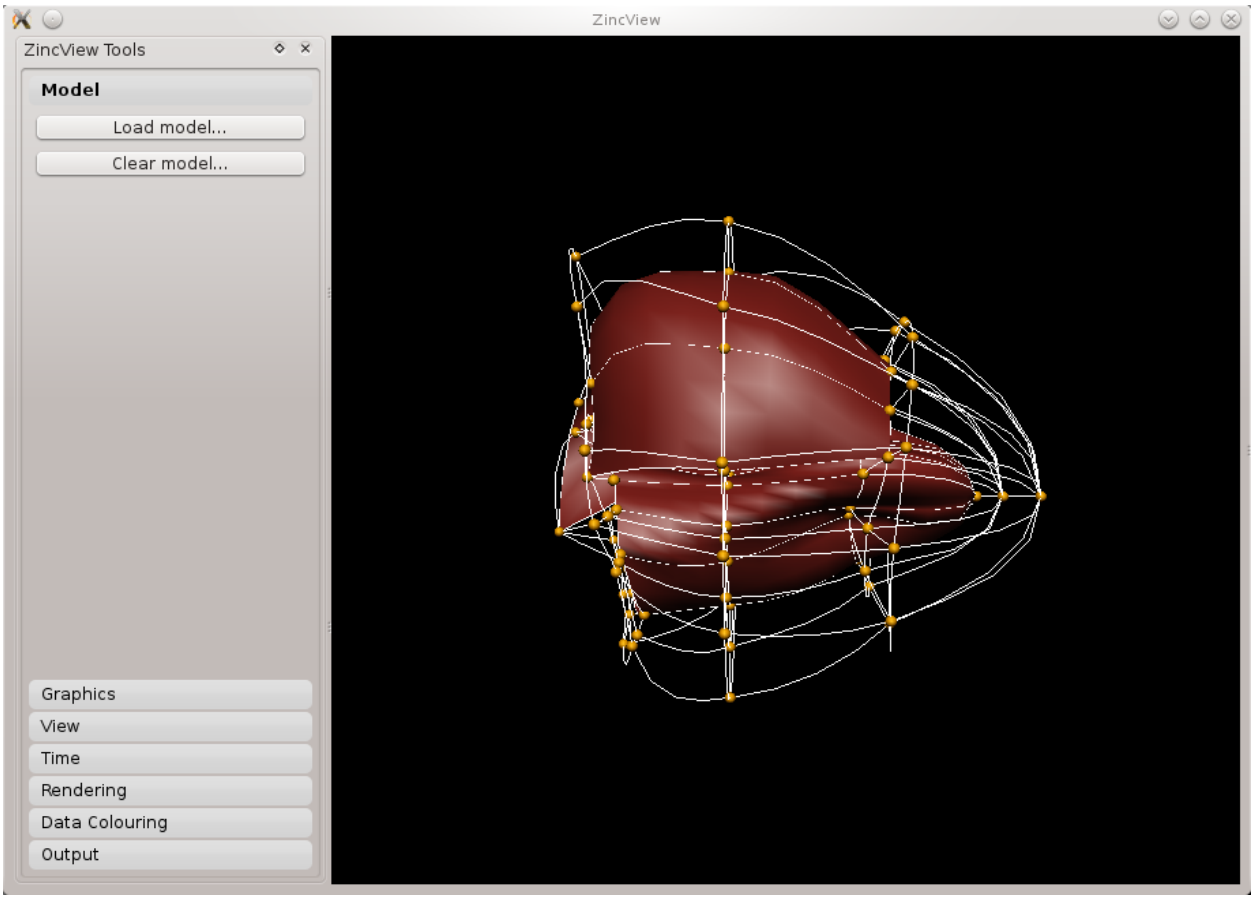

Fig. 2.22: Visualisation of the Heart Model

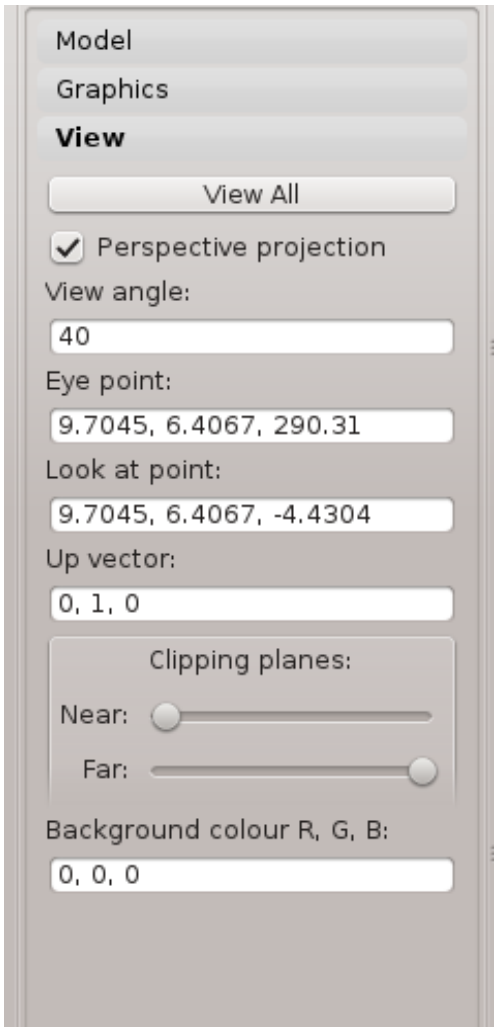

Fig. 2.23: View pane

known as clipping. The same phenomenon happens when the far plane passes the furtherest most object in the scene. Adjusting the near and far plane can be used to see inside a closed surface or to create an effect when producing a movie.

We can also manipulate the viewing frustrum in other ways. We have spoken about the viewing frustrum being a chopped pyramid, and this is true when we are viewing the scene in perspective projection. But we can change this to an orthographic projection (sometimes called parallel projection) where the chopped pyramid becomes a cuboid. Using the parallel/perspective control [2] observe what effect this has on the scene. Using the camera zoom zoom out in perspective mode and then fly-zoom back in to get an ultra-wide angle lens view. At extremes the scene can look a little crazy, use the 'view all' button [3] to reset the view.

From the 'Graphics' toolbox (see Fig. 2.24) we can change the properties of the graphics. We can change the visibility for instance, this can be done by checking and unchecking the check box next to the graphic name [1]. Another graphical aspect that we can easily change is the material used to colour the graphic. The material can be changed with the material combobox [2], try selecting the surfaces in the graphics list [3] and setting the material to blue.

Point graphics are drawn using glyphs. There are pre-defined glyphs that have already been created that can be used to produce interesting visualisations, sometimes a cube glyph is more appropriate representation of the data. Select the 'node points' from the graphics list [3] and change the glyph to 'cube_solid'. We can use the scaling to change the size of the glyph (and other graphics). The final size of the scaled object is defined as:

```
size = base_size + scaling*scale_field
```

If no scale field is set and the base size is zero then the graphics will be not be visible.

We can also add new graphics with the *add button* [4]. Add a 'point graphic', change the glyph for the new graphic to 'axes_xyz' and set the base size to 50. This point graphic is representing the global x, y, z axes in the current scene.

The heart model contains data on the direction of the fibres within the heart wall, to visualise this we can add some more graphics. Using the following instructions we can visualise the heart fibre direction:

```
1. Add 'streamlines' using the Add combobox [3]
2. Set the Streamlines vector field to 'fibres' using the Vector field combobox [8]
3. Using the Shape combobox [9] set the shape to 'square extrusion'
4. Set the base size to 1*0.2
5. Set the sampling divisions to 1*3*1
6. Set the Time length [10] value to 50
```

Here we have set the base size and sampling divisions using a special notation. This notation allows us to set different values for different components, we can also just set one value which will be propagated across all components automatically.

It is often desirable to view the contours of the data, a contour is where the function has a constant value. We can show contours through the heart wall volume. To do this:

```
1. Add a 'contour' graphic using the Add button [4]
2. Set the value field to 'lambda'
3. Set the iso value to 0.75
```

We can see now that the visualisation is getting quite busy. To reduce some of the graphics visible we can try setting the exterior checkbox [5] on the surfaces. The exterior checkbox allows us to only view the exterior surfaces of a volume. This can be very useful especially when using transparent materials where we do not wish to show the construction of the mesh used for the model.

We can colour the graphics according to some data available in the model. We will colour the surfaces with the lambda field to do this:

```
1. Select the surfaces in the graphics list
2. From the data combobox [6] choose the 'lambda' field
```

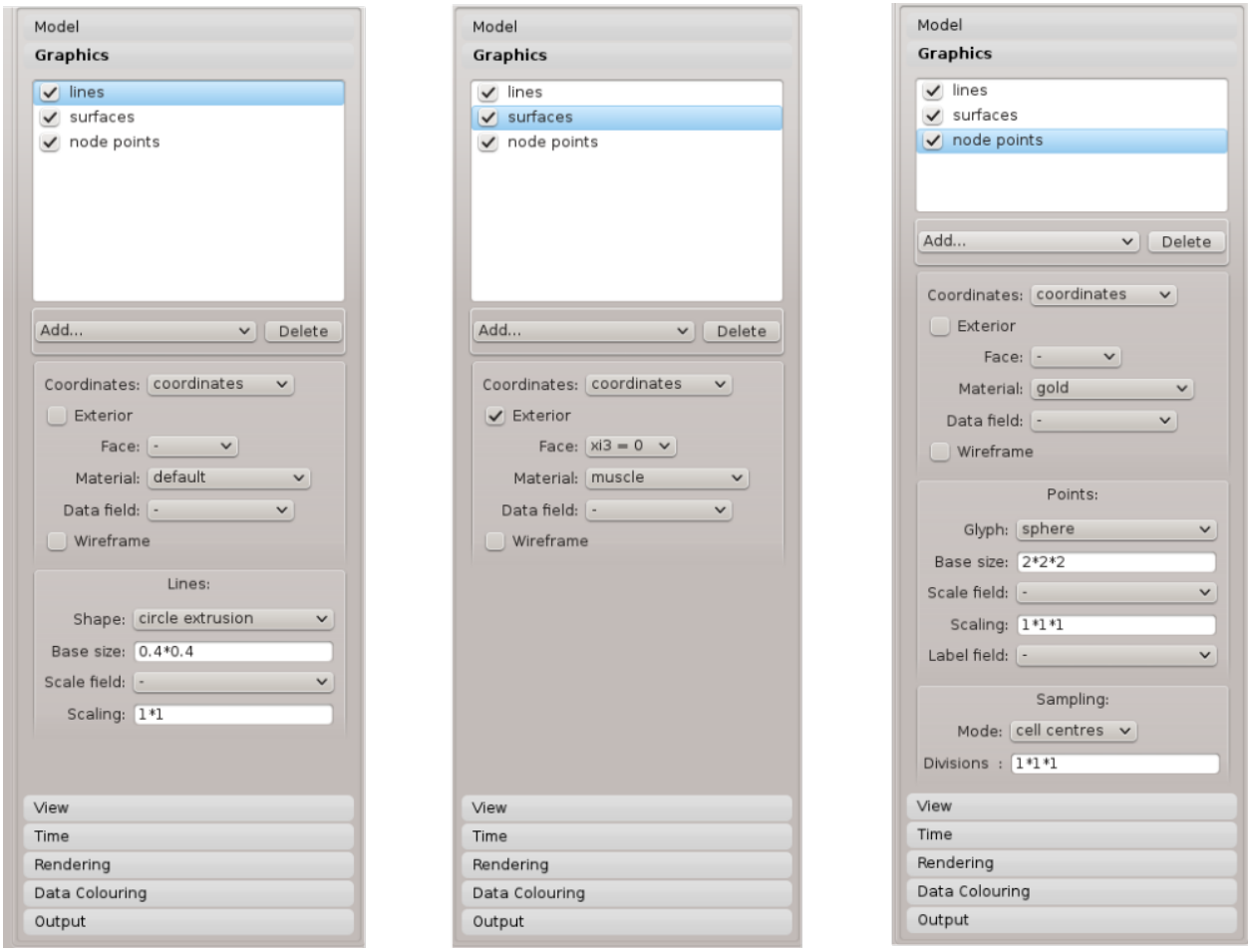

Fig. 2.24: Graphics pane

For the final rendering before we produce a publishable image we may decide that the background colour is not suitable for our target medium. We can change the background colour one the view pane. Using the view pane thingy box set the RGB values for the background colour, 1,1,1 will set the background colour to white for example.

We can also control the quality of the rendering via the refinement option on the rendering pane (Fig. 2.25). Use this control carefully it can take a long time to render highly refined graphics. The circle divisions option controls the quality of spheres and cylinders. Set the refinement factor to 10 and see the result.

All of this visualisation is done through OpenGL and we can see what is actually being rendered by using the wireframe option [7] on the graphics pane (Fig. 2.24).

Task 2

data/airways/AirwaysLobes.ex{node|elem}.

Steps:

1. Load nodes then elements (it's a large model so doesn't load instantly) and get attendees to create lines. They need to do view-all on the view tab to see them.
2. change the line shape to circle extrusion with scale by the 'general' field (a radius) with scaling 2 to make it a diameter.
3. zoom in to see the gaps between the line segments. Add nodes with sphere glyphs scaled by the same fields to close off the gaps.
4. colour by radius by picking general as the data field
5. on the data colouring tab. Click on autorange spectrum, try different ranges to make the image pretty. Add a colour bar. The colour bar appears in the list of graphics, but it uses some hidden attributes (not editable) to make it appear on top where it is. You can change the colour of the point graphics for the colour bar which affects its shininess and the colour of the labels. The colour bar is actually just a glyph, but it's pretty silly to plot it at every node, for example, but it works!
    - (Alan may be able to get another model with dependent fields by the time of the course, which will make it more interesting to visualise.)
6. [Alan needs to implement output of images here] From the output tab output a [hopefully hires] image of what's on the screen, including colour bar, suitable for putting in your report.

Task 3

data/deforming_heart/deforming_heart.zincview.py

1. load the model which is similar to the heart in part 1 but has twice as many elements. It also defines strain fields and creates element point graphics which visualise mirrored glyphs to show principal strains: inward and red for compression, outward and blue for extension.
2. on the time pane, adjust the time slider to animate the model
3. zoom in and look at the deformation of parts of the tissue, the twisting of the ventricle etc.
4. change the glyph to arrow_solid (on all 3 element points) and line to see the difference.
5. advanced users may look at the script to see how these additional field are created by expressions; interest them in the possibilities of what they could visualise. They can also see how the time-varying model was loaded.
6. [Alan would have to implement]. Output to ThreeJS an animating, beating heart (probably no glyphs?)

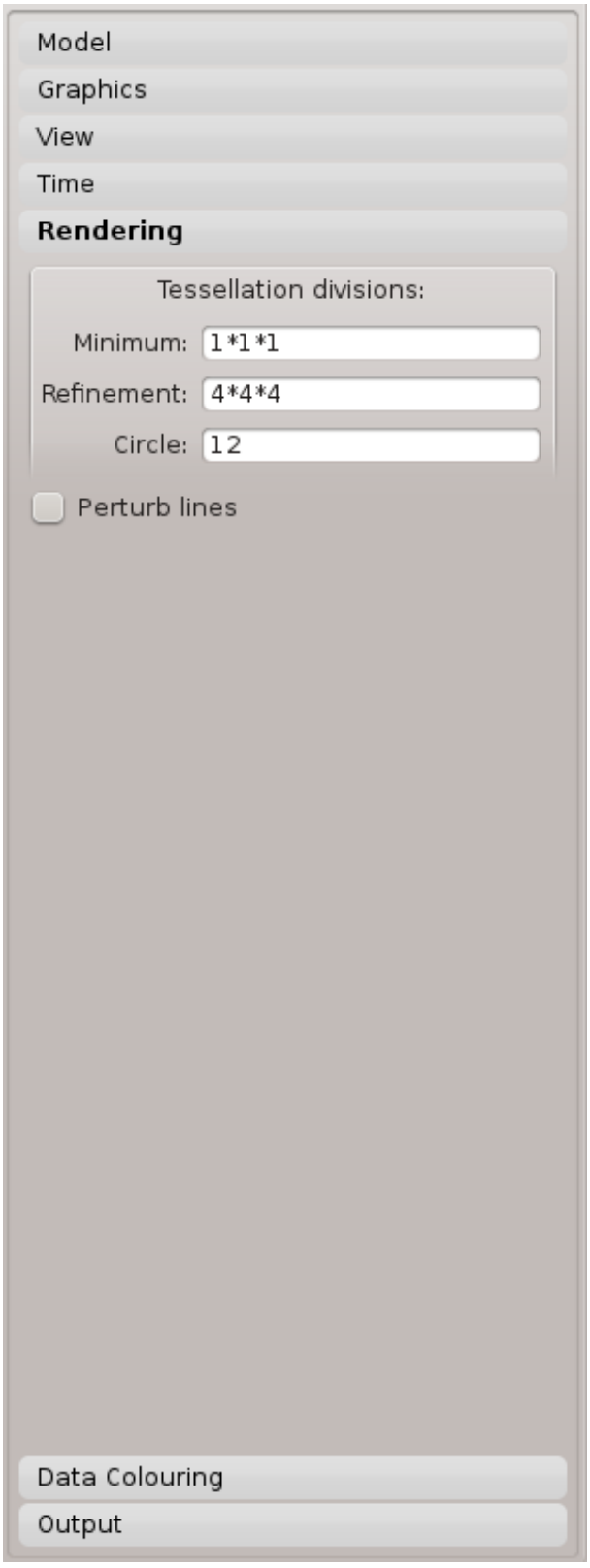

Fig. 2.25: Rendering pane

Task 4

Ideally I would have liked to have got images into the rendering; I'm sure Alan could get the volume texture example going pretty quickly; can use that to segment part of the foot. Images combined with a model requires us to support multiple regions or groups, which I haven't had time to do; adding a region chooser would be simplest I think, but probably no time. Could always draw image in another region (from the loading script) but wouldn't be able to hide it.

2.6 Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Introduction to (ODE) Modelling Best Practices

In this series of tutorials we demonstrate some modelling practices that we recommend you follow when developing any mathematical model. The key principles are modularity and reuse - we demonstrate those here using the tool OpenCOR and the CellML exchange format, but the principles are valid across the spectrum of computational physiology.

For the actual DTP the tutorial will be part of this documentation, but for the May 2015 trial run the tutorial is available in this `PDF document`. If you do not already have OpenCOR installed, the tutorial will lead you through the process of downloading and installing OpenCOR.

CHAPTER 4

MSK Workflow Demonstration

In the actual DTP Computational Physiology module, this section of the module would lead the students through the assembly of a comprehensive MSK workflow. Leading from clinical images through to some kind of kinematic model, visualisation, and output to some clinically relevant observations/interface.

For the May 2015 trial run, we will not be presenting this part of the Computational Physiology module.

CHAPTER 5

DTP Computational Physiology to do list

5.1 General

Todo

- any general todo's go here.
- make sure the todo's are turned off in the config?

5.2 Within sections

Todo

- any general todo's go here.
- make sure the todo's are turned off in the config?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/dtp-compphys/checkouts/v2015.07/source/todolist.rst, line 9.)

Todo

- add in more useful links in the virtual machine
- make sure the password is what is specified above.
- check consistent use of HOME

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/dtp-compphys/checkouts/v2015.07/source/virtualmachine.rst, line 14.)

CHAPTER 6

Indices and tables

- genindex
- modindex
- search