

Software from the

DArT_Toolbox

(D' Arcy Thompson Laboratory,

Computing Sciences, University East Anglia, Norwich, NR4 7TJ)

<http://cmpdartsvr1.cmp.uea.ac.uk/wiki/BanghamLab/index.php/Software>

<http://www.uea.ac.uk/cmp/People/Honorary/I.+Andrew+Bangham>

Funded by BBSRC SABR, BBSRC ANR

April 2012

Publically available software is boldface

Toolboxes

AreaFinder

CellFinder

CellSegment

ClusterControl

FilterTubulesExperiments

GrowthToolbox <http://cmpdartsvr1.cmp.uea.ac.uk/wiki/BanghamLab/index.php/Software>

HapticToolbox

MediawikiToolbox

MicrotubuleAnalysis

MicrotubuleModeller

OpenMicroscopyToolbox

OpticalFlowTool

OpticalFlowToolPlus

PyMicrotubules

SectorAnalysisToolbox <http://cmpdartsvr1.cmp.uea.ac.uk/wiki/BanghamLab/index.php/Software>

SegmentationToolbox

ShapeModelToolbox <http://cmpdartsvr1.cmp.uea.ac.uk/wiki/BanghamLab/index.php/Software>

TICA

TrackingToolbox

VolumeAnalysisToolbox

VolViewer <http://cmpdartsvr1.cmp.uea.ac.uk/wiki/BanghamLab/index.php/Software>

ToolBag

AlgPicker

Demo of JRK GUI

FEMGrowthTool

FEMViewer

ImageCompositer

ImageReScaler

LeicaScript

MakeTestShapes

ReactionDiffusionTool

RecursivelyFindDependencies

SelectOrderListDlg

ShapeEdgeEditor

SieveStack

SpatialAnnotation

StackViewer

Usefull

VolViewerAPI <http://cmpdartsvr1.cmp.uea.ac.uk/wiki/BanghamLab/index.php/Software>

dipum_toolbox

The following pages are captured from the BanghamLab public Wiki and document our publically available software.

Other software can be shared on request. A full download of the DArT_Toolshed (i.e. checkout all the software) is 2.4 GB.

Software

Contents [\[hide\]](#)

- 1 [Computational biology toolboxes](#)
 - 1.1 [Modelling the growth of shapes: Gftbox](#)
 - 1.2 [Viewing and measuring volume images: VolViewer](#)
 - 1.3 [Analysing shapes in 2D and 3D: AAMToolbox](#)
 - 1.4 [Analysing the shapes of clones: SectorAnalysisToolbox](#)
- 2 [Open source systems to which we contribute](#)
 - 2.1 [OMERO](#)
- 3 [Tools and Utilities](#)
 - 3.1 [BioformatsConverter](#)
- 4 [Software not yet used in publications](#)
 - 4.1 [MTtbox](#)

Current activity: a **collaboration** with the [CoenLab](#) with the aim of understanding how patterns of gene activity in biological organs influence the developing shape. The BanghamLab is focussed on the conceptual underpinning: concepts captured in computational growth models, experimental data visualisation and analysis.

Computational biology toolboxes

Modelling the growth of shapes: **Gftbox**



For modelling the growth of shapes.

What? How? Where?

Tutorials: [from the beginning](#)

Examples: [from publications](#)

Download Gftbox from [SourceForge](#)

Download Gftbox project files:

[Leaves](#) [Kuchen et al 2012](#)

[Principles and concepts](#) [Kennaway et al 2011](#)

[Snapdragon](#) [Green et al 2011](#), [Cui et al 2010](#)

Ready Reference Manual

(PC, Mac, Linux, uses Matlab
no Mathworks toolboxes needed
[Matlab 30 day free trial](#) and
[student edition](#))

Gftbox is an implementation of the Growing Polarised Tissue Framework for understanding and modelling the relationship between gene activity and the growth of shapes such leaves, flowers and animal embryos ([Kennaway et al 2011](#)).

The GPT-framework was used to capture an understanding of (to model) the growing leaf ([Kuchen et al 2012](#)) and Snapdragon flower ([Green et al 2011](#)). The Snapdragon model was validated by comparing the results with other mutant and transgenic flowers [Cui et al 2010](#).

The icon shows an asymmetrical outgrowth. Conceptually, it is specified by two independent patterns under genetic control: a pattern of growth and a pattern of organisers. The outgrowth arises from a region of extra overall growth. Growth is aligned along axes set by two interacting systems. Organisers at the ends of the mesh create a lengthwise gradient. This gradient interacts with the second due to an organiser that generates polariser in a region that becomes the tip of the outgrowth. ([Kennaway et al 2011](#))

Viewing and measuring volume images: **VolViewer**

For viewing and measuring **volume images** on both normal and **stereo** screens. Typical images from: confocal microscope and

VolViewer is used as a **stand-alone** app. or as a **viewport for other systems**, e.g. Matlab programs. VolViewer uses [OpenGL](#) and [Qt](#) to provide a user



Optical Projection Tomography (OPT) images

What? How? Where?

Tutorials: [from the beginning](#)

Download

(Windows, Mac, Linux)

Output from VolViewer has appeared in:
[Front cover: Handbook of Plant Science](#) |
[Front cover: The Plant Cell](#) | [Royal Microscopical Society: Infocus Magazine](#) |
[Bundled with the Biotonic 3001 scanner: Biotonics Viewer](#) | [The Guardian newspaper: 3D Fruit fly](#) | [Qt Ambassador program](#) | [Triffid Nurseries website](#)

friendly application to interactively explore and quantify multi-dimensional biological images. It has been successfully used in our lab to explore and quantify confocal microscopy and optical projection tomography images. It is open-source and is also compatible with the Open Microscopy Environment ([OME](#)).

Analysing shapes in 2D and 3D: **AAMToolbox**



For analysing populations of shapes and colours within the shapes using principal component analysis.

What? How? Where?

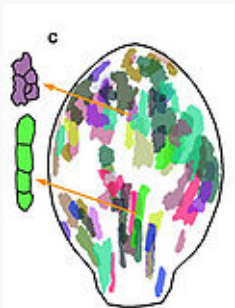
Tutorials: [from the beginning](#)

Download

(PC, Mac, Linux, uses Matlab
no Mathworks toolboxes needed
[Matlab 30 day free trial](#) and
[student edition](#))

The AAMToolbox enables the user analyse the shape and colour of collections of similar objects. Originally developed to analyse face shapes for lipreading ([Matthews et al. 2002](#) [version of pdf](#)), we have used it extensively for analysing the shapes of leaves ([Langlade et al 2005](#), [Bensmihen et al. 2010](#)) and petals ([Whibley et al 2006](#), [Feng et al. 2010](#)). The analysis can be applied to art, for example, finding systematic differences between portraits by, for example, Rembrandt and Modigliani.

Analysing the shapes of clones: **SectorAnalysisToolbox**



For analysing the shapes of marked cell clones.

What? How? Where?

Tutorials: [from the beginning](#)

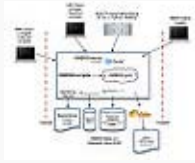
Download

(PC, Mac, Linux, uses Matlab
no Mathworks toolboxes needed
[Matlab 30 day free trial](#) and
[student edition](#))

The SectorAnalysisToolbox enables the user analyse the shapes of marked clones in a sheet of tissue.

Open source systems to which we contribute

OMERO



For working with the OME image database.

See [Details](#), [Download](#)
[OMERO Workshop](#)

(Windows, Mac, Linux)

[Open Microscopy Environment Remote Objects \(OMERO\)](#) for visualising, managing, and annotating scientific image data. See also our [OMERO Workshop](#) training course we ran in April 2011.

Tools and Utilities

BioformatsConverter



For converting microscope manufacturer proprietary file formats.

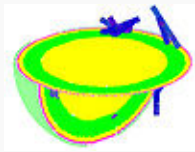
See [Details](#)

(Windows, Mac, Linux)

This tool allows for the batch conversion of microscope manufacturer proprietary file formats, to the open source OME-TIFF standard. Uses the [Bioformats](#) library.

Software not yet used in publications

MTtbox



For modelling the behaviour of microtubules within a cell.

See [Details](#)

(Windows, Mac, Linux)

In development. The idea is to be able to model the behaviour of growing microtubules and factors as they react chemically and diffuse within the different cell compartments.


The icon shows a spherical cell sliced open to show concentric components: cell wall (magenta), plasma-membrane (yellow), cytoplasm (green) and vacuole (yellow). Microtubules (blue) grow in 3D within the cytoplasm.

modified on 17 April 2012 at 16:28 *** 2,910 views

University of East Anglia / School of Computing Science / Powered by MediaWiki

Gftbox

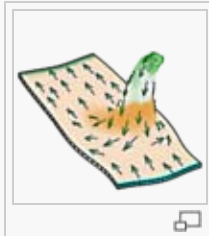
[Back to BanghamLab software](#)

[Tutorials](#) and [Download](#) 

Contents [\[hide\]](#)

- [1 What is Gftbox, how does it work and where is it?](#)
- [2 What does Gftbox require?](#)
- [3 How to start using Gftbox](#)
- [4 Limitations of Gftbox](#)

What is *Gftbox*, how does it work and where is it?



We wish to understand how patterns of gene activity in biological organs influence the developing shape. Our suggestion is that genes may regulate growth direction independently of growth rate. We formalised our ideas in the Growing Polarised Tissue Framework (reference 1, below). The icon is based on Fig. 4 of reference 2, and shows an asymmetrical spur that has grown under the influence of a polariser gradient, indicated by the arrows.

Intuition is not a good guide to the behaviour of tissue growing under the influence of patterns of growth. Patterns interact with each other and the continuous sheet of tissue to produce complex shapes. It is therefore necessary to translate intuition into a mathematical model whose behaviour can be calculated. *Gftbox* is a

package for creating and simulating such models.





A *Gftbox* project comprises two parts, the Mesh and the Interaction function. The Mesh is a finite element model of the tissue, together with patterns of gene activity, tissue identity, and biochemical properties. The Interaction function specifies the local interactions of these properties.

Does *Gftbox* work correctly? Validation is an important question to be asked of all modelling software. There are various simple models for which analytical solutions can be obtained and compared with the computations performed by *Gftbox*. An extensive range of such validation tests can be found in Supplemental Text 1 of reference 2.




We have used the **GPT-framework** to understand the development of the Snapdragon flower (reference 3 below). **Using *Gftbox*** we started with a crown shaped sheet of tissue (the canvas), laid out observed or hypothesised patterns of regulatory activity and then grew the canvas in 3D. Patterning can continue during growth and the final shape was compared quantitatively with its biological counterpart -- so **testing the hypotheses** made in the model (reference 4 below).

To work with *Gftbox* is to practice thinking quantitatively about the relationship between genes, growth and form.

References

1. Erika E. Kuchen, Samantha Fox, Pierre Barbier de Reuille, Richard Kennaway, Sandra Bensmihen, Jerome Avondo, Grant M. Calder, Paul Southam, Sarah Robinson, Andrew Bangham, Enrico Coen, 2012 "Generation of Leaf Shape through Early Patterns of Growth and Tissue Polarity", *Science*: 2 March 2012: 1092-1096. 
2. Kennaway R, Coen E, Green A, Bangham A, 2011 Generation of Diverse Biological Forms through Combinatorial Interactions between Tissue Polarity and Growth. *PLoS Comput Biol* 7(6): e1002071. 
3. Amelia Green, Richard Kennaway, Andrew Hanna, Andrew Bangham, Enrico Coen, "Genetic Control of Organ Shape and Tissue Polarity", *PLoS Biol.* 2010, vol.8, no.11. 
4. Min-Long Cui, Lucy Copsey, Amelia Green, Andrew Bangham, Enrico Coen, "Quantitative Control of Organ Shape by Combinatorial Gene Activity", *PLoS Biol.* 2010, vol.8, no.11. 

What does *Gftbox* require?

***Gftbox* is written in Matlab** . It does not require any extra Mathworks toolboxes, nor any separately compiled modules. Matlab is available as a [30 day free trial](#)  and as a [student edition](#) . *Gftbox* comprises around 1000 subroutines and 90,000 lines of code. It allows individual models, the part that a user writes, to be very compact -- 10 to 100 lines of Matlab code. For example, the model shown in Fig. 1 has a mesh that was created using the graphical user interface, and this code in its interaction function:

```
if (Steps(m)==0) && m.globalDynamicProps.doinit % Initialisation code.
    id_a_p(m.nodes(:,1) < -0.03) = 1; % set up region for A where identity factor A is represented by id_a_p
```

```

id_b_p(m.nodes(:,2) < -0.01) = 1; % set up region for B
else
% @@KRN Growth Regulatory Network
kapar_p(:) = id_a_l .* inh(1,id_b_l); % growth rate
kaper_p(:) = kapar_p; % isotropic growth
kbpap_p(:) = kapar_p; % same on both sides of the sheet
kbper_p(:) = kapar_p; % same
knor_p(:) = 0; % thickness not growing
end

```

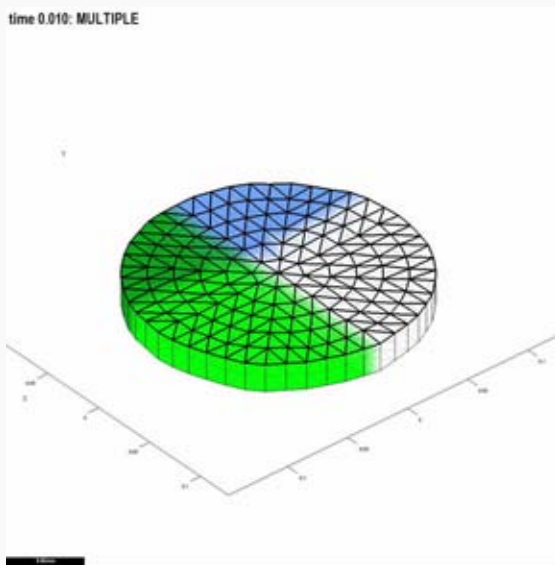


Fig. 1. Mesh with two intersecting patterns of growth factors. Blue is factor A, Green factor B

A movie of the mesh growing and subdividing. The intensity of red represents the growth rate

Why Matlab? The language suits our problem - the reasoning goes like this. Tissue is represented by a thin 3D mesh. Growth factors levels vary spatially forming patterns, e.g. Fig. 1. Here there are two, A and B. We might make the hypothesis that the growth rate is specified by A but partially inhibited by B (inhibited by an amount K). This is a simple idea that can be expressed in Matlab equally simply by writing $Growth = A .* inh(K, B)$. This is because variables A and B can represent vectors - in this case a level for each node in the mesh. We also define a general inhibition function (*inh*). It means that it is straightforward to convert our thoughts on the biology into a programmatic description of a computational model. Moreover, the language is well documented with lots of convenient tools. In particular, Matlab has an extensive library of portable graphical user interface (GUI) functions - and this is convenient for producing tools to visualise the mesh and patterns of growth factors.

The computational bottleneck in *Gftbox* is solving the finite element equations. We find that the Matlab [cgs](#) function is quicker and more accurate than alternatives from other sources (e.g. [lsqr](#), [culaSgesvd](#)). The speed of *Gftbox* appears to be limited by memory-cpu bandwidth rather than cpu speed. It runs best interactively on a 64 bit machine with a large memory.

Growing *Gftbox* models does not require the GUI. Everything done by the GUI can also be done at the Matlab command prompt or within a Matlab script file, Multiple models can be run as Matlab scripts on a computing cluster, and the results examined visually in the GUI.

Matlab is sufficiently flexible that inexperienced programmers (and half our team are biologists, not programmers) can pick it up on-the-fly. It is an interpreted language, which makes it easy to step through code to follow what is happening in case of difficulty.

Matlab runs on Windows, Mac and Linux. The key Matlab numerical algorithms are documented and generally visible and there is a large user-base providing lots of web based help.

How to start using *Gftbox*

[Gftbox can be freely downloaded from sourceforge](#). After downloading, start Matlab, cd to the *Gftbox* directory, and give the command `Gftbox`. *Gftbox* will start and present you with its graphic interface.

Gftbox automatically adds all of its subdirectories to the Matlab command path. If you give a `savepath` command at the Matlab console, then Matlab will remember where *Gftbox* is, and on subsequent runs of Matlab you will not have to cd to the *Gftbox* directory before running it.

In addition to the tutorials given here, *Gftbox* comes with several example projects and the projects behind our published Figures. These projects appear on the Projects menu of the *Gftbox* window, in the Models submenu. There is a Help menu and when a new project is formed - the new associated interaction function is filled with comments that illustrate how the model can be written. In the GUI, mouseover text is provided for each of the GUI functions and the last item in every menu is a local help submenu. Documentation files can be found in the `doc` subdirectory of *Gftbox*.

Ready Reference Manual

Gftbox For modelling the growth of shapes.

Details: [what? How? Where?](#)

Tutorials: [from the beginning](#)

Examples: [from publications](#)

Download from [SourceForge](#) 

Contents [\[hide\]](#)

[1 Types of morphogens](#)

[2 Summary of controls](#)

[2.1 Models](#)

[2.2 Running models](#)

[2.3 Outputs](#)

[2.3.1 Plotting panel](#)

[2.3.2 Snapshots](#)

[2.3.3 Exporting meshes for external use](#)

[2.3.4 Movies](#)

[3 Useful interaction function programming constructs](#)

Types of morphogens

[Types of morphogens and factors click here](#)

Summary of controls

Roll the mouse over any control and information about the control will pop-up.

Models

- ▶ The larger the time step, the fewer iterations will be needed to simulate a given amount of time, but the less accurate the simulation will be. If the time step is too large, diffusion calculations may even become unstable. As a rule of thumb, the time step should be chosen small enough that:
 - ▶ The product of growth rate and time step is everywhere less than 0.1 -- that is, no part of the mesh grows by more than 10% in a single iteration.
 - ▶ In a single iteration, no part of the mesh rotates by more than 10 degrees.
 - ▶ The product of time step and the relative rate of change of any morphogen should be no more than 0.1. The rate of change may derive from the decay rate you have set, or from interactions programmed in the interaction function.
 - ▶ If k is the diffusion constant of a morphogen, D is the diameter of the mesh, and dt is the time step, $k dt^2/D$ should be no more than 0.1.
- ▶ Errors in the interaction function will cause the system to stop, and the interaction function will be disabled. The controls in the Interaction Function panel will all turn red to indicate this.
 - ▶ To re-enable the interaction function, click the Reset button at the top of the GUI.
 - ▶ To find the error
 - ▶ There will be a message on the Matlab console saying in which line of the interaction function it detected an error. Open the interaction function in the Matlab editor, find that line, and set a breakpoint there or shortly before (by clicking in the left margin -- a red dot will appear).
 - ▶ The next time the interaction function runs, Matlab will suspend execution at the breakpoint and go into debug mode. You can then examine the values of variables and step through the code one line at a time in order to discover why the error is happening.
 - ▶ Matlab provides extensive tutorials on how to program and there is much information on the web.

Running models

The normally **green Run panel** turns **red** whilst the model is running.

A **yellow** Run panel means that the model is running on a version of the Gftbox that is older than the model itself (look at the interaction header to see our source code version control number for the copy of Gftbox that created the model).

An interaction function error (or any other error) will stop the model from running. It is then necessary to Reset using the **Reset key - which will have turned red**.

To run a model from the GUI

- ▶ Step, execute one step of dt . A whole lot of information pours out at the command console. Sometimes it takes a while to solve the equations. The process is monitored by a series of dots Further information is provided below the Plotting panel, e.g. Step number, time per iteration, time per finite element, growth, etc.
- ▶ Run for, and enter the number of steps into the box.
- ▶ Run until, and enter the model time to which you want to run.
- ▶ Run to, and enter the multiple of the initial area you want to run to.
- ▶ Stop, stops execution at the end of the current step.
- ▶ Call, makes a call to the interaction function but does NOT grow. This is useful for checking initial conditions - set patterns of morphogens. Having made a Call to the interaction function it is good practice to then Restart the model before running a model.
- ▶ Menu:Stages. **We usually run models** by setting up a set of, say 5, stages that we want stored. Then, having run the model we can revert back to any of the stages - make a change - and re-run starting from that point. The stages are stored in the project directory with the suffix *000# where # is the time. Stage times are stored with the project - and copied into the daughter project with Save As.
 - ▶ Compute more stages, pops up a box asking for a set of times that you want stored then starts computing
 - ▶ Request more stages, does the same but does not start computing.
 - ▶ Save experiment stages, saves all the current stage files in a subdirectory.
 - ▶ Import experiment stages, recovers a saved experiment. This pair of commands help document results and make figures. In addition, when the system runs on a cluster, the results are returned as experimental results to be accessed in this way.
- ▶ **What to do when the running system gets stuck**. Usually this is because the shape is too regular and cannot initiate a bulge.
 - ▶ Interrupt processing using standard Matlab convention - Control-C. Then reset the *Gftbox*. Panel:Select Tool:Mesh Editor and slightly modify the Z shape a random amount: click Random.

Outputs

Plotting panel

To view

- ▶ Specified values, i.e. kapar or a particular morphogen (factor) select from the drop down menu at the top right, and tick the Plot Current Factor checkbox.
 - ▶ To autoscale the axes, set the checkbox Panel:Plot Options:Auto axis range. To autozoom as well, click Menus:Plot:Auto zoom and centre.
 - ▶ To autoscale the morphogen colours. Panel:Plot Options:Auto color range. To fix the colour range the easiest is to click From Picture and unclick the Auto color range. Alternatively, unclick the Auto color range box and key in values you want - click elsewhere on the GUI to force a replot.
 - ▶ To find out the level of a morphogen at a particular point on the canvas. Select Tool:Simulation and select the morphogen of interest (top right pull down) and click on the point of interest. The value at that point will appear in bold just below the Panel:Timescale box.
- ▶ Output variables, e.g. growth rate, tick Plot output value and make selection from the two drop down menus.
- ▶ thickness, tick/untick Menu:Plot:Thickness.

To show/hide

- ▶ thickness, tick/untick Menu:Plot:Thickness.
- ▶ mesh, tick/untick FE edges.
- ▶ axes, tick/untick Menu:Plot:Show/Hide axes.
- ▶ legend, tick/untick Menu:Plot:Show/Hide legend - also Set legend to choose heading.

To set/modify values of morphogens (best done in the interaction function)

- ▶ Choose morphogen (list box), Click on Select Tool:Factors, '---' list box to select action and click on appropriate button (Add constant or Add radial, etc).
- ▶ at individual nodes (vertices). Choose morphogen (list box), Click on Select Tool:Factors, '---' list box to select action and click on node.

To clip the mesh (hide part of the mesh)

- ▶ Tick Clipping plane and choose the plane with Az (azimuth), El (elevation), D (distance from origin).
- ▶

Tick Clip and select the region to hide by using morphogen values (click on Mgens button).

Snapshots

Saved in png format in the project snapshots subdirectory

- ▶ Clicking Take snapshot makes a png file copy of the plotting screen. At the same time a copy of the interaction function (relabelled to have .txt as a suffix) is also made - to help document the snapshot.
- ▶ Publication quality screenshot is made by changing the action of the Snapshot button using Menu:Misc:Hi-res snapshots and choosing a number of dots per inch.

Exporting meshes for external use

The current mesh can be saved in various formats for use outside GFTbox, using the various Save... commands on the Mesh menu.

- ▶ VRML is a standard format for 3D graphics, for which there are free viewers (for example, [BS Contact](#) or [ParallelGraphics' Cortona 3D Viewer](#)). It is also a format supported by [ZCorp](#)'s 3D printers. When you export to VRML, after providing a file name you will be presented with a dialog that allows you to rescale the object and also to change its thickness. Changing the thickness is only important for 3D printing, as a physical model will be too fragile if the mesh is too thin.
- ▶ OBJ is another standard format for 3D models, and can be imported into most 3D modelling software.
- ▶ FIG is the Matlab format for figure files. These files can be reopened in Matlab.
- ▶ MAT is Matlab's format for storing arbitrary Matlab objects.

MAT is the only format that saves the entire mesh structure. The other formats save only its current graphical representation.

Movies

By default, movies are uncompressed, since this is the only format that is sure to play on all machines. However, uncompressed movies can easily be gigabytes in size. Various compressed formats can be selected in the Movie:Codec menu. Several formats are available, although not all of them are supported on all Matlab installations, and where supported, the resulting movies may not play in all movie-playing software. You only need to select a codec once, and it will apply to all subsequently created movies.

- ▶ For recent versions of Matlab (2010 and later) we recommend the Motion JPEG AVI format. (Older versions of Matlab do not provide it.)
- ▶ Flash movie format cannot be created directly, but there is an option to convert completed movies to Flash automatically, with the menu Movie:Convert to Flash. This option requires the ffmpeg program to be present on the Matlab command path. Conversion from .avi to .flv is achieved with the help of

```
MakeVideosAndFLV.m
```

- ▶ Click Record Movie to start creating a movie. The current picture of the mesh will be the first frame of the movie. The button will change to Stop Movie. Every time a simulation step is performed, the picture of the mesh will be added to the movie file. Click Stop Movie to stop recording and close the movie file. The movie file will also be automatically closed if you close the project, load a new project, or close GFTbox. When the movie file is closed, the final frame will also be stored as a PNG image file, as well as a copy of the interaction function (with extension .txt instead of .m).
- ▶ By default, the name of the movie file is chosen automatically, and consists of the project name followed by a numerical suffix. If you turn off the Auto-name checkbox below the Record Movie button, you will be prompted to give the movie a name. Movies (and the associated snapshots and interaction functions) are stored in a "movies" subdirectory of the project directory.

Useful interaction function programming constructs

[Types of morphogens and factors click here](#)

| | | |
|------------------------------|--------------------------|--|
| Pre-defined variables | dt | computational step size |
| | realtime | virtual time of the model |
| Pre-defined functions | Steps(m) | current step number in Mesh m |
| | pro(k,id) | promote by k in regions designated by factor, id ($pro(k, \mathbf{x}) = 1 + k\mathbf{x}$, i.e. promote by $k=0$ and the value of $pro()$ is 1). |
| | inh(k,id) | inhibit by k in regions designated by factor, id ($inh(k, \mathbf{x}) = 1/(1+k\mathbf{x})$, i.e. inhibit by $k=0$ and the value of $inh()$ is 1). |
| | local_setproperties(m) | initialise Mesh properties |
| | leaf_*(m, ...) | A large library of functions for manipulating the Mesh data structure, m. All of them are listed in the main Help menu of the GFTbox GUI, both in alphabetical order and categorised by topic. |

The patterns of morphogens A and B are often set up by using logical indexing, e.g.

```
id_a_p(m.nodes(:,1) < -0.03) = 1;  
id_b_p(m.nodes(:,2) < -0.01) = 1;
```

where id_a_p is the vector of values of the A morphogen. $m.nodes(:,1)$ refers to the x coordinates of all nodes (vertices) in the mesh.

The expression $(m.nodes(:,1) < -0.03)$ means find all vertices with x coordinates that are less than -0.03.

Similarly, $(m.nodes(:,2) < -0.01)$ means find all vertices with y coordinates that are less than -0.01.

And the pattern of polariser (P) is set up by

```
P((m.nodes(:,1) < -0.05) & (m.nodes(:,2) > 0.03)) = 1;
```

Where $(m.nodes(:,1) < -0.05) \& (m.nodes(:,2) > 0.03)$ means find all vertices with x coordinate less than -0.05 and y coordinate greater than 0.03.

Thus the full code describing the model is:

```
if (Steps(m)==0) && m.globalDynamicProps.doinit % Initialisation code.
    id_a_p(m.nodes(:,1) < -0.03) = 1; % setup region for A where identity factor A is represented by id_a_p
    id_b_p(m.nodes(:,2) < -0.01) = 1; % setup region for B
else
    % @@KRN Growth Regulatory Network
    kapar_p(:) = id_a_l .* inh(1,id_b_l); % growth rate
    kaper_p(:) = kapar_p; % isotropic growth
    kbpar_p(:) = kapar_p; % same on both sides of the sheet
    kbper_p(:) = kapar_p; % same
    knor_p(:) = 0; % thickness not growing
end
```

modified on 1 February 2012 at 08:56 *** 535 views

University of East Anglia / School of Computing Science / Powered by MediaWiki

Types of morphogens and factors

Contents [\[hide\]](#)

- [1 Notes on Matlab syntax](#)
- [2 GfTbox Growth factor naming conventions](#)
 - [2.1 Prefix](#)
 - [2.2 Suffix](#)
- [3 Standard morphogens](#)
 - [3.1 Specifying growth](#)
 - [3.2 Controlling growth](#)
- [4 User \(your\) morphogens](#)
 - [4.1 Identity factors, signalling factors, viewable variables](#)

[Back to Tutorial pages](#)

[Ready Reference Manual](#)

Notes on Matlab syntax

- ▶ Scalar: a single value
- ▶ Vector: a column of values - one value of each node (vertex) in the mesh.

In an interaction function a variable, lets use a variable called *id_a_p*. This is a vector with one element for each node (vertex).

- ▶ `id_a_p(:)=0;`

sets the value of all the nodes to 0 (the colon : means all).

- ▶ `id_a_p(1)=3;`

sets the value of node 1 to 3. Actually, we rarely know where the nodes are - and we don't care. If we want to set up a pattern of nodes we usually refer to them by their spatial position. Thus, we could find all nodes that have an *x* value that is equal to the minimum *x* value in the mesh.

- ▶ `minx=min(m.nodes(:,1));`

sets *minx* to the minimum *x* and the expression `(m.nodes(:,1)==minx)` finds all nodes with an *x* value that equals the minimum *x* value. Thus,

- ▶ `id_a_p(m.nodes(:,1)==minx)=3;`

sets *id_a_p* at the selected nodes to the value 3.

GfTbox Growth factor naming conventions

Prefix

- `id_` **identity** factors (non-diffusible). Identity factors can be labelled as on the A or B side (`ida_`, `idb_`)
- `s_` **signalling** factors (diffuse). Polariser level is unique and has a particular name *P*.
- `v_` visualising labels (e.g. theta, bend, kpar, other labels)
- `f_` finite element manipulation (e.g. seams, making holes)

Suffix

- `_p` vector representing **promotor**: how much the expression of the factor is being promoted in the model at every node (vertex) in the mesh.
- `_l` vector representing the **level** of morphogen, i.e. `*_p` times `*_a` at every node in the mesh. This gets updated at the end of the interaction function and might change according to growth or mutation, see `*_a`.
- `_i` scalar index into the main data structure morphogen field, `m.morphogens`.
- `_a` scalar specifying activity: by default it is 1 and if the factor is mutated or silenced then it is 0. However, it can be in the

Standard morphogens

Specifying growth

| | |
|----------------------------------|--|
| <i>kapar_p, kapar_a, kapar_l</i> | vector that specifies growth on the 'A' side parallel to the polariser and in the plane of the tissue at every node. |
| <i>kbpar_p, kbpar_a, kbpar_l</i> | vector that specifies growth on the 'B' side parallel to the polariser and in the plane of the tissue at every node. |
| <i>kaper_p, kaper_a, kaper_l</i> | vector that specifies growth on the 'A' side perpendicular to the polariser and in the plane of the tissue at every node. |
| <i>kbper_p, kbper_a, kbper_l</i> | vector that specifies growth on the 'B' side perpendicular to the polariser and in the plane of the tissue at every node. |
| <i>knor_p, knor_a, knor_l</i> | vector that specifies growth normal to the plane of the tissue (i.e. thickness) at every node. |

Controlling growth

| | |
|--|---|
| <i>P</i> | vector that specifies Polariser level at every node. |
| <i>strainret_p, strainret_a, strainret_l</i> | vector that specifies strain retention at every node. It is a number between 0 (no strain retention, default) and 1 (full strain retention). |
| <i>arrest_p, arrest_a, arrest_l</i> | vector that specifies an overall inhibition of growth at every node of the mesh. It is a number between 0 (no overall inhibition of growth, default) and 1 (fully inhibited). |

User (your) morphogens

Within the interaction function all the usual Matlab variables and types are available. Variables that are to be visible from the GUI must follow certain conventions and be declared within the GUI, see below.

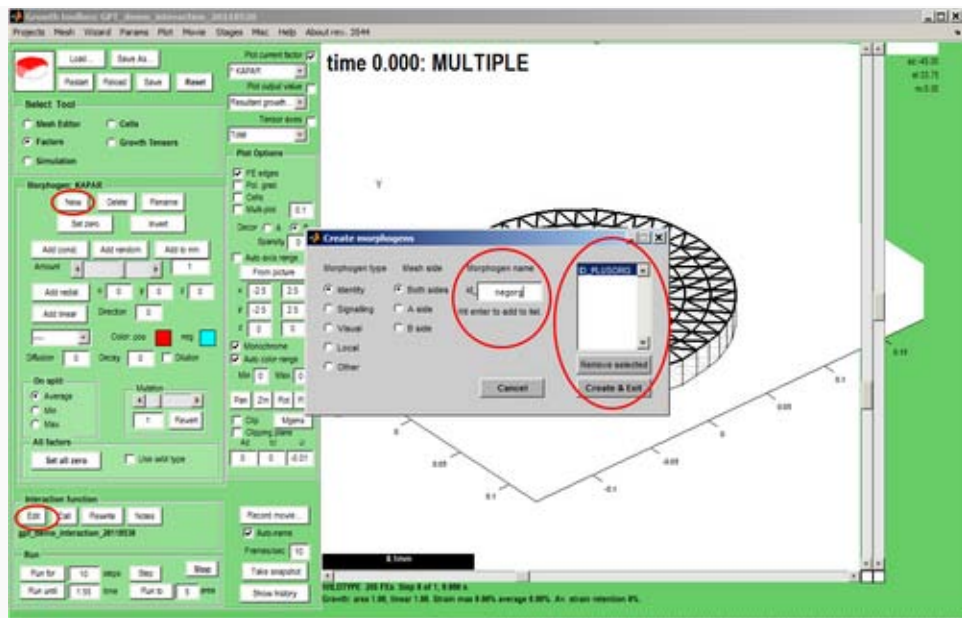
(Note to programmers: we could have produced a class library to handle the different types of variable - identity factors, signals, etc. However, in our view it is too early in the development of software for modelling the growth of biological tissues to be confident of what will be required. Too build a class library now would be to build a conceptual straightjacket that would inhibit further research.)

Identity factors, signalling factors, viewable variables

Growth is all about patterns of growth factors interacting in space (spatially) and over time (temporally). It is, therefore, essential to be able to view them on the mesh. This is the main purpose of the graphical user interface (GUI). Such variables are created (declared) from the GUI as follows. They have a name e.g. *id_plusorg*. This is how the variable will be known as a case insensitive string (according to Matlab conventions, 'id_plusorg' or 'ID_PLUSORG'). Within the interaction function this name will be lowercase and have a suffix e.g. *id_plusorg_p* - see above.

Adding (declaring) variables that are visible in the GUI. *Select Tool: Factors* makes the *Morphogen Panel* visible. We add a two new morphogens, *id_plusorg* and *id_negorg*. Select a type (in this case *id_*), key in the name, press Enter and the name will be added to a list. Finally, create the morphogens and exit the dialogue. Select a colour for each, *Panel: Morphogen: Color. pos.* and *Color. neg* (for negative values).

Save to make sure they are saved in the Mesh. Pressing *Panel: Interaction Function: Edit* **will reload the editor** with a revised copy of the interaction function. The new copy will retain any edits that you have made already together with the new variables.



modified on 24 June 2011 at 17:08 *** 391 views

University of East Anglia / School of Computing Science / Powered by MediaWiki

GFtbox Tutorial pages

GFtbox Details

Notes from a new user

Running example models and using a cluster

Examples from our papers

Types of morphogens and factors are given here

(A newly created interaction function is shown here - the green comments are another ready reference.)

Ready Reference Manual

The models shown in these tutorials illustrate features of the GFtbox software. They are not designed to understand the Growing Polarised Tissue Framework which is better done with Examples from our papers.

Viewing these pages. Some versions of *Firefox* and *Explorer* do not create satisfactory prints even though you can view the pages with no problems. *Chrome* does appear to produce good printouts.

Three ways to use *GFtbox*

1) **Do everything from the Graphical User Interface (GUI).**

2) **Do only some things from the GUI.** Use the GUI to generate the mesh (canvas) and create growth factors, but capturing your ideas on how the regulatory processes work by writing Matlab code in what we call the interaction function.

3) **Without the GUI.** For example, run many examples (instances) of a pre-existing project on a cluster. This is the best way to explore the parameter space of a model for comparison with biological observations. We use this extensively once we have roughed out the basic ideas of a model interactively.

1 Modelling using the Graphical User Interface

Isotropic growth

How to use the tutorial. Open GFtbox and attempt to repeat the results shown.

1 A

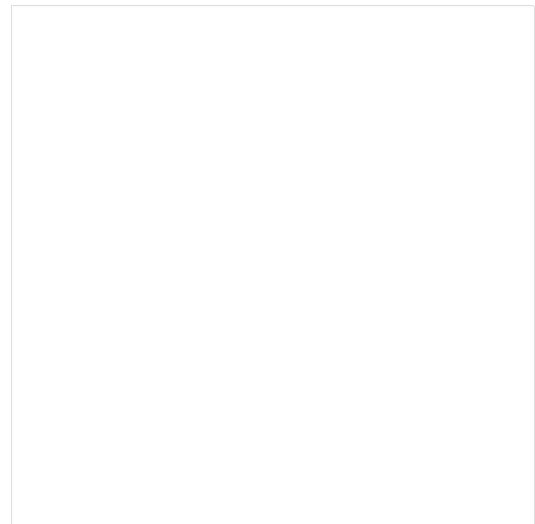
Tutorial on uniform growth.

Consider a disc shaped canvas (tissue) in which the **specified growth is uniform**, isotropic and on both sides.

Into what shape will the disc grow?

This model is as simple as it gets. Notice that, during growth, the mesh is automatically subdivided. Notice also that the final surface is not quite flat. This is because, to allow it to deform in 3D, it is not flat initially. There are options to initialise a flat mesh and others to force it to remain flat - see options on the GUI (hover over controls to get prompts).

In the absence of a polariser (there is no polariser in this example) growth will be isotropic, in other words growth in the plane of the canvas will be the average of what is specified for *Kapar* and *Kaper* (*A* side) and *Kbpar* and *Kbper* (*B* side).



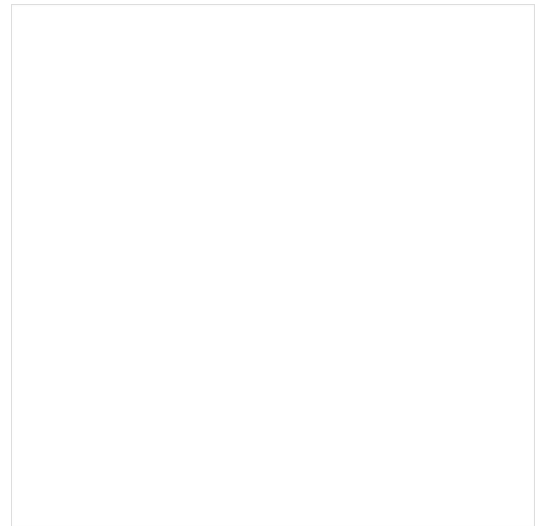
1 B

Tutorial on non-uniform growth.

Consider a disc shaped canvas (tissue) in which the **non-uniform specified growth** increases in proportion to the distance from the centre.

Into what shape will the disc grow?

Already we are into the realms of modelling biological systems. Compare this result with the discussion of Lily petals and Gaussian curvature (Lianga and Mahadevana (<http://www.pnas.org/content/early/2011/03/14/1007808108.abstract>), Sharon, Marder and Swinney (<http://www.americanscientist.org/issues/feature/leaves-flowers-and-garbage-bags-making-waves>), Nath, Crawford, Carpenter and Coen (http://rico-coen.jic.ac.uk/uploads/0/0f/Nath_Science.pdf)).



Adding polariser

1 C

Tutorial on uniform growth with non-uniform polariser.

In the presence of polariser, *GFtbox* growth will be anisotropic, in other words growth in the plane of the canvas can be different parallel and perpendicular to the axis of the polariser: *Kpar* and *Kper* (A side) and *Kbpar* and *Kbper* (B side).

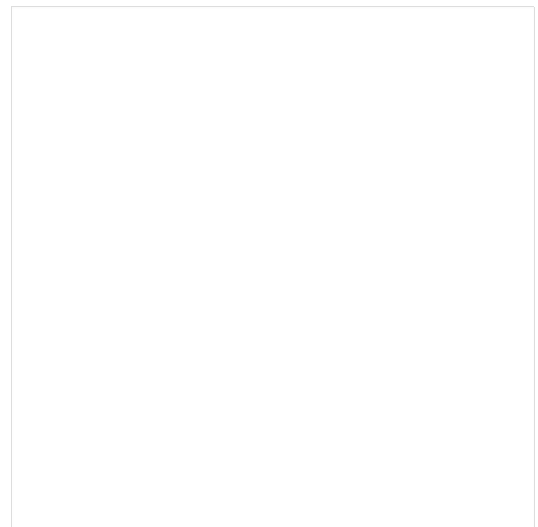
We now **add polariser**. Start with example A - uniform growth - and add a radial polarising gradient.

Arrows show the direction of the gradient. They are turned on using the Panel:Plot option:Pol. grad. tick box. The gradient defines local directions and local growth can be specified either parallel to (*Kpar*) or perpendicular to (*Kper*) that direction.

Given a **uniform pattern of specified growth parallel** to the polariser and **zero specified growth perpendicular** to the polariser.

What will be the final shape?

Note: the gradient of the polariser, green to cyan, is shown using the arrows. Specified growth rate parallel to the arrows, red, is uniform.



2 Modelling using a combination of GUI and interaction function

How to use these tutorials. Use a combination of the GUI to set up the Mesh structure and editing the associated interaction function to repeat the results shown. Full listings of the interaction functions are given from which you can copy the key, editable, elements.

The **full specification** of a *GFtbox* model is stored in a combination of the mesh data structure (**Mesh**) and the **interaction function**. The Mesh stores all the physical properties of the system: spatial structure, mechanical properties, etc. It is usually set up using the GUI. The Mesh is stored on disc as a Matlab data file (.mat) and in memory as a data structure (m). The **interaction function** (a Matlab program file .m) contains all the details of the growth regulation system: morphogen concentrations, signal interactions etc.

When a new project is first edited **the interaction function is generated automatically**. Thereafter, it is automatically kept in synchrony with the GUI. It is divided into several sections. Some are generated automatically and should not be edited. Others are set aside for the user to specify the model. To ensure that the automatic and manual edits are synchronised **always invoke the Editor from the GUI** (Panel: Interaction Function: Edit).

2 A

Basic interaction function The patterns of morphogens A and B are set up by

```
id_a_p(m.nodes(:,1) < -0.03) = 1;
id_b_p(m.nodes(:,2) < -0.01) = 1;
```

where id_a_p is the A morphogen. $m.nodes(:,1)$ refers to the x coordinates of all nodes (vertices) in the mesh. The expression $(m.nodes(:,1) < -0.03)$ means find all vertices with x coordinates that are less than -0.03.

Similarly, $(m.nodes(:,2) < -0.01)$ means find all vertices with y coordinates that are less than -0.01.

And the pattern of polariser (P) is set up by

```
P((m.nodes(:,1) < -0.05) & (m.nodes(:,2) > 0.03)) = 1;
```

Where $(m.nodes(:,1) < -0.05) \& (m.nodes(:,2) > 0.03)$ means find all vertices with x,y coordinates that are less than -0.05 and greater than 0.03 respectively

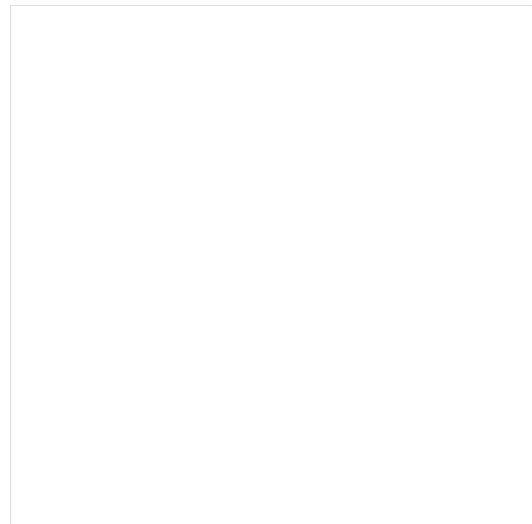
Thus the full code describing the model is:

```
if (Steps(m)==0) && m.globalDynamicProps.doinit % Initialisation code.
    id_a_p(m.nodes(:,1) < -0.03) = 1; % setup region for A where identity factor A is represented by id_a_p
    id_b_p(m.nodes(:,2) < -0.01) = 1; % setup region for B
else
    % @@KRN Growth Regulatory Network
    kapar_p(:) = id_a_l .* inh(1,id_b_l); % growth rate
    kaper_p(:) = kapar_p; % isotropic growth
    kbpar_p(:) = kapar_p; % same on both sides of the sheet
    kbper_p(:) = kapar_p; % same
    knor_p(:) = 0; % thickness not growing
end
```

Tutorial on a basic interaction function.

The tutorial explains how the code shown above appears in the interaction function.

To create the interaction function, first create and save a project (see above) then click Panel:Interaction function>Edit. The interaction function will contain all the default variables and any patterns of variables (*Kapar*, etc.) that have already been set through the GUI. Types of morphogens and factors are given here.



Play and the mesh will grow and subdivide. Normally we do not want to see the mesh. Red is growth rate

2 B

Interaction function in detail

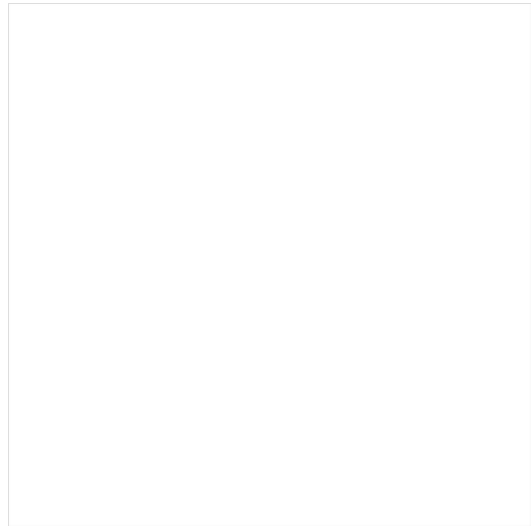
(A newly created interaction function is shown here - this one has two, user specified morphogens: id_a and id_b .)

There are two submodels. An identical growth pattern and polariser pattern is specified in both, but only the specified growth in the second is anisotropic and so uses the axially specified by the polariser gradient. The gradient is zero in the region where the polariser is held at a constant value of one - and therefore growth in the region without arrows is isotropic.

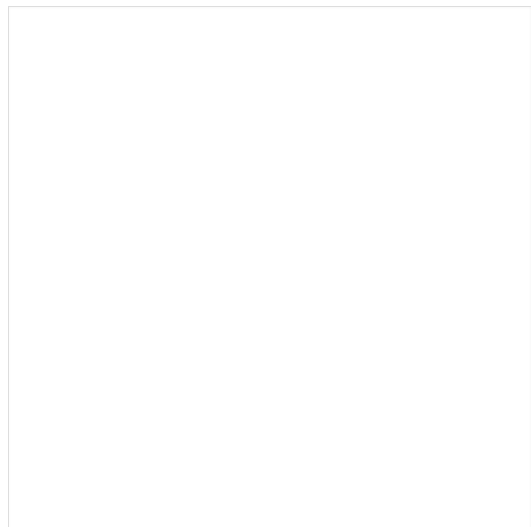
Tutorial on the interaction function details.

Types of morphogens and factors are given here.

Submodel 1



Submodel 2. Growth in the region that has no arrows is isotropic and, as result, the outgrowth is blunt.



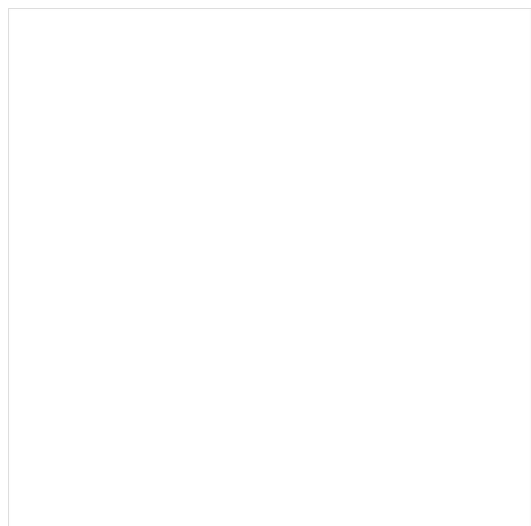
2 C Illustrating independent ways to form shapes and the use of submodels.

Conclusion: interesting shapes can be generated either by **patterns of differential growth** or **patterns of local growth axes**.

Tutorial on different ways of specifying the growth of shapes.

Submodel 1. Uniform specified polariser (**no polariser gradient**).
Creating a shape using a specified pattern of isotropic growth.

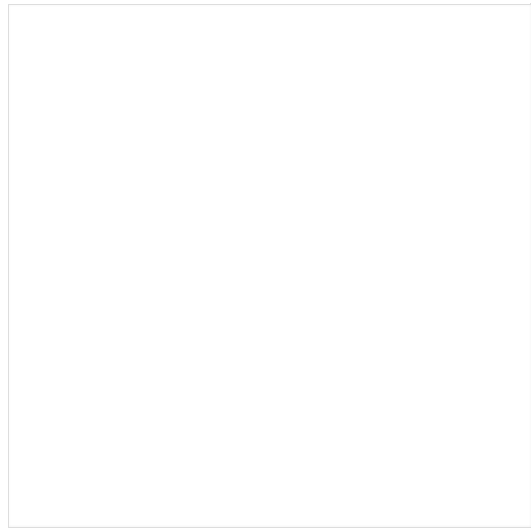
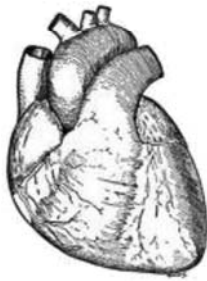
Result: simple patterns of differential growth can readily produce blobby shapes.



Play submodel 1. Pattern of isotropic specified growth (no polariser)

Submodel 2. **Uniform specified growth.** Creating a shape using a specified pattern of diffusible polariser.

Result: simple patterns organising a diffusible polariser can readily produce sharp shapes.



Play submodel 2. Pattern of specified polariser levels (green-cyan). Polariser can diffuse and the gradient is arrowed. Uniform specified growth (red).

Outputs from these models bear only a romantic relation to hearts, flowers and lepidoptra.

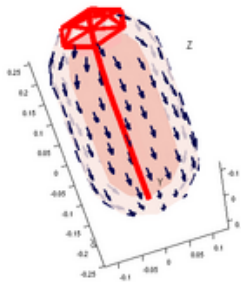
2 D

Retaining strain and cutting the mesh

There are two submodels. They are the same except that in the first strain is not retained from step to step whereas in the second strain is fully retained. The shape changes during growth look similar, the difference is only revealed when the mesh is cut. The behaviour of plant tissue can be similar.

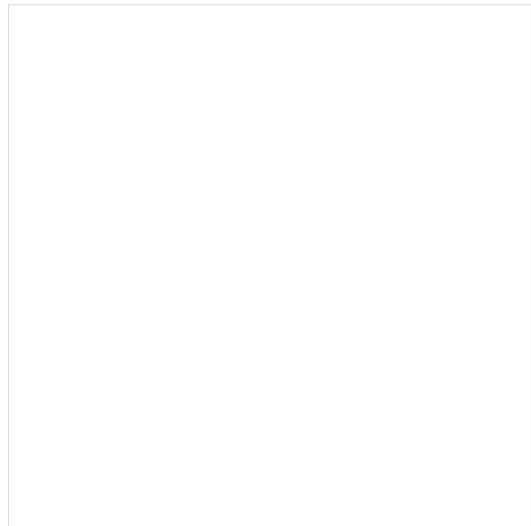
Tutorial on retaining residual strain and cutting

time 50.000:



8.2mm

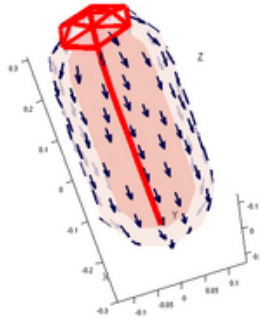
Strain not retained after each step. In plant tissue terms this would appear to be equivalent to there being a homeostatic mechanism that



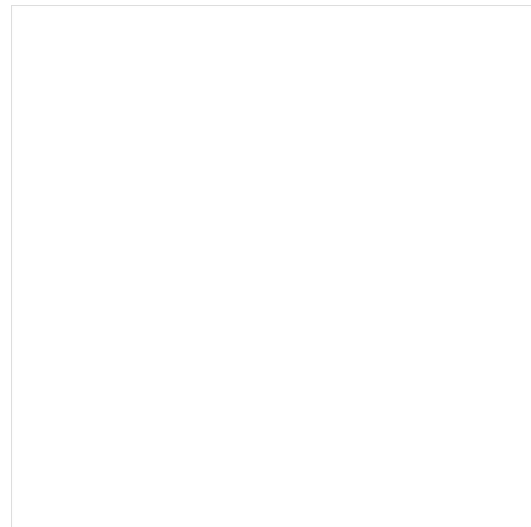
operates such as to re-strengthen the cell walls after they have been weakened to the point that they yield under turgor pressure during growth.

Play submodel 1

time 50.000:



Retaining the strain from step to step allows the growth on the inside to force the mesh as a whole to grow more than when strain is not retained.



Whilst the frame rate relates directly to growth rate, the relaxation rate after cutting does not. After cutting the computation has to be done in smaller steps and the rate at which the sides bend back shown here is too slow.

3 Running models without the GUI

Having developed the concepts underpinning a pattern of growth in an interaction function it is often desirable to explore a range of model parameters. Given that each run of the model can take between 5 minutes and hour it is appropriate to run the models in parallel on a computing cluster. (Each node of the cluster to be used by *GFtbox* needs to be licensed to run Matlab - however for the purpose of running on a cluster without the GUI we are exploring the possibility of making *GFtbox* compatible with Octave (<http://www.gnu.org/software/octave/>) .)

How to use these tutorials. First ensure that your model is working as you would expect using the GUI. Then run the model from the Matlab command line. Finally, submit the project to the linux computing cluster with one or more ranges of parameters. For further details Running example models and using a cluster

modified on 2 August 2011 at 07:52 ••• 1,139 views

In the beginning Uniform

[Back to Tutorial pages](#)

Contents [\[hide\]](#)

- [1 Modelling uniform growth using the GUI alone](#)
 - [1.1 First view](#)
 - [1.2 Add a mesh \(canvas that represents tissue\)](#)
 - [1.3 Add a pattern of growth factor](#)
 - [1.4 And grow](#)

Modelling uniform growth using the GUI alone

First view

The GUI controls are organised into a number of panels. The largest is the plotting window on the right. Just to the left is a column of buttons, tick boxes, drop-down lists and buttons that control what appears in the plotting window.

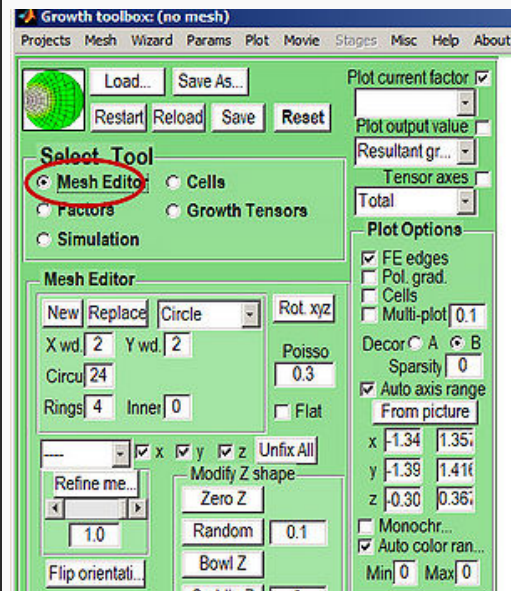
At the top left is a panel for loading and saving projects.

Below this is a panel for selecting tools. These buttons select what panel will be visible below, e.g. Mesh Editor, Factor, etc.

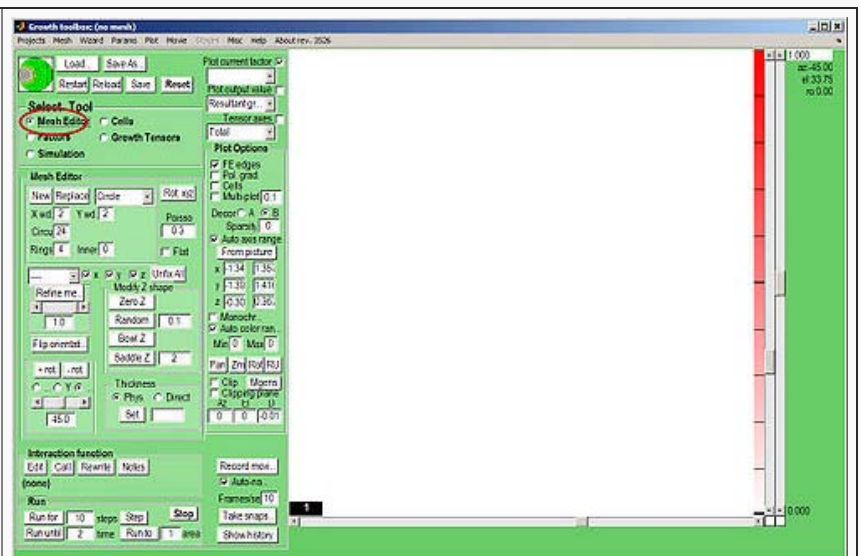
At the bottom left the Run panel contains controls for running the model: Step, run for a specified number of steps, for a specified (model) time, run to a certain area (e.g. 2 times starting area). Above the Run panel is a panel for creating and editing the interaction function.

There are also controls on the Menu bar.

The *GFTbox* user interface. Focus on the *Select Tool* Panel. Here the *Mesh Editor* has been selected and the *Mesh Editor* panel is visible.



Idea: change all *GFTbox* font sizes using *Menu Misc: Gui Format*

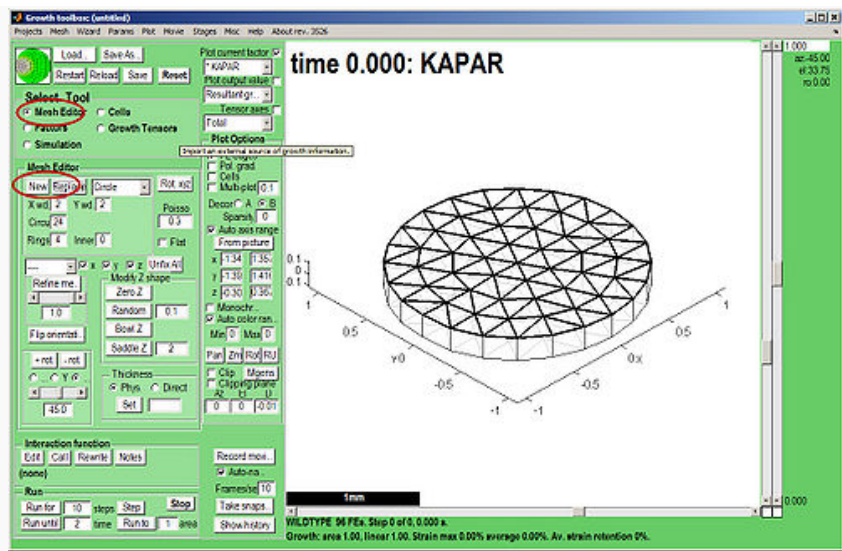


Add a mesh (canvas that represents tissue)

Create a canvas by clicking on *Panel Mesh Editor: new*. A variety of preset shapes is available. Here we choose a Circle with 4 concentric rings of nodes that is 2 mm in diameter.

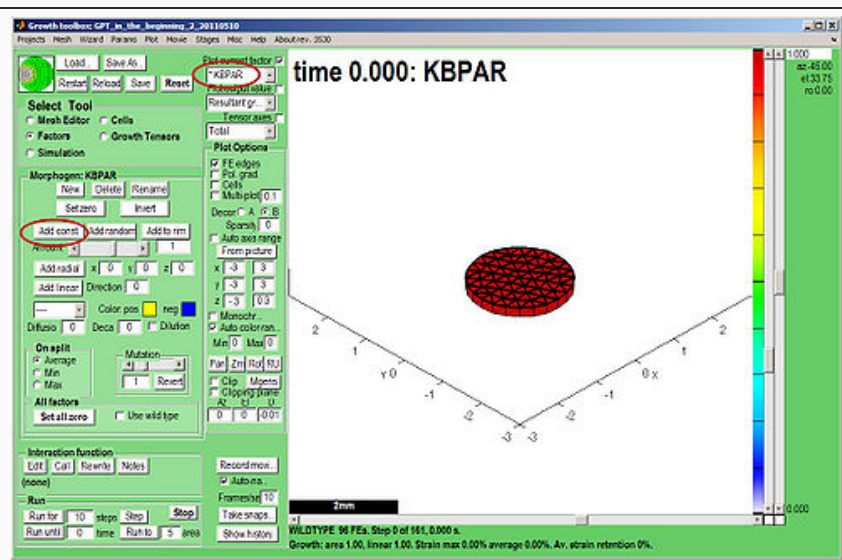
By default the mesh is autoscaled and autocentred - it makes it easier to see shape changes but to see growth these are best turned off. Set the Panel:Plot options:Auto axis range to unticked and insert plot ranges -2.5 to 2.5 on x and y axes and 0 0 on z. Also turn off Menu:Plot:Autozoom and centre.

Idea: Special shapes can be imported as OBJ files.



Add a pattern of growth factor

Select Tool: Factors makes the Morphogen Panel visible. Currently we are controlling the KAPAR factor. It has been selected from a drop down list top right (Plot current factor). We add a constant level of KAPAR. Make sure that the Panel Plot Options: Monochrome is ticked and the gradient of KAPAR will be as shown. Do the same for the top face, KBPAR.



And grow

Now save the project using the Save As button. A dialogue box will open - here we keyed in in_the_beginning, and the prefix GPT_ and the suffix _date were added automatically.

A directory is created with the project name which contains all the files needed for the project and any results (snapshots, movies) we create.

Select Tool: Simulation makes the Simulation Panel visible.

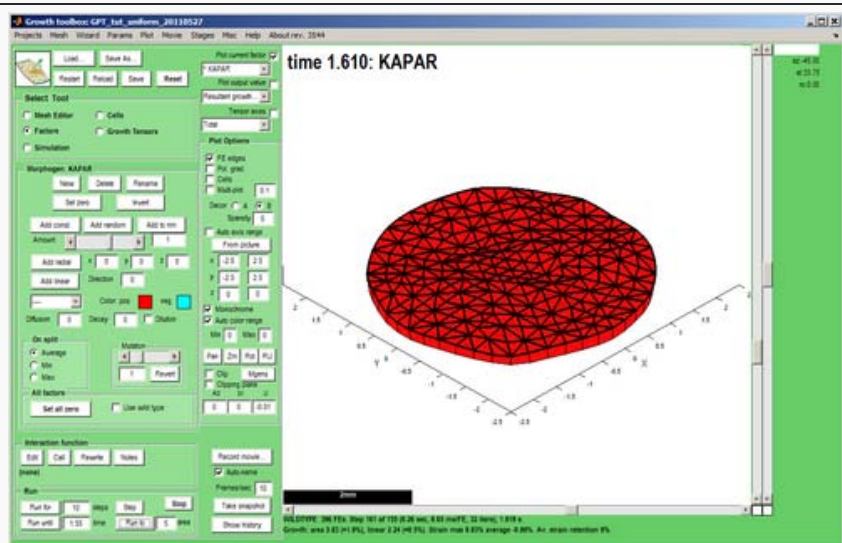
Change the Panel Run: Run until box to 2 and click the Panel Run: Run until button. The canvas will grow in steps set by the Panel Simulation: Timescale box.

Warning: saving is essential if you want to Restart the simulation or if you want to save a movie.

Idea: set the axes manually (here to +-3) and turn off

Panel:Plot Options: Auto axis range and Menu:Plot:Auto zoom and centre. This enables growth to be seen as a change of size rather than change of axis range. Idea: take a snapshot of the canvas by pressing Take snap... (c.f. image on right, button is bottom right of GUI controls)

Idea: to create a movie of your simulation, before running click the Record movie and when it is finished click Stop movie. The movie will be in the directory containing the current project, which you can open with the menu command Projects>Show Current Project Folder.



The canvas is not flat because it started not flat. By default, a new mesh is given a small random perturbation to allow deformations out of the plane.

In the beginning non-uniform

[Back to Tutorial pages](#)

We start with a number of very large screenshots (don't be daunted they are just here to orient you). They show the *GFTbox* graphical interface.

Contents [\[hide\]](#)

[1 Modelling non-uniform growth using the GUI alone](#)

[1.1 First view](#)

[1.2 Add a mesh \(canvas that represents tissue\)](#)

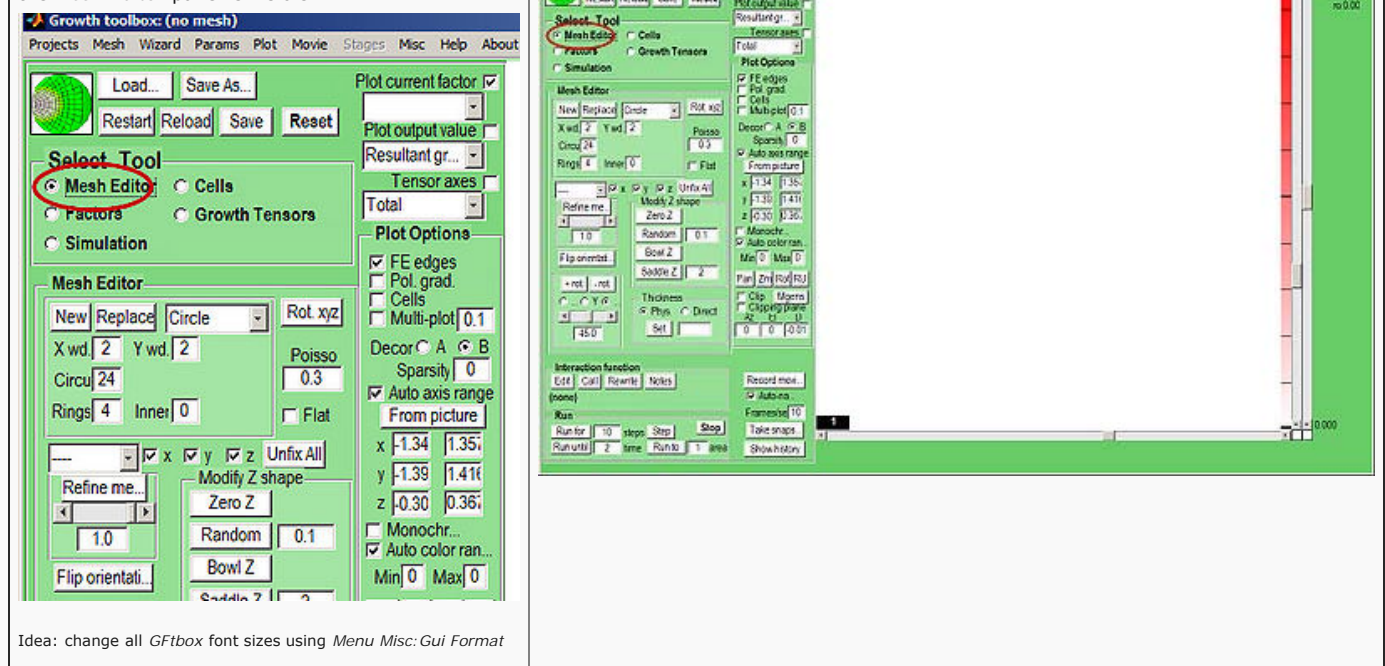
[1.3 Add a pattern of growth factor](#)

[1.4 And grow](#)

Modelling non-uniform growth using the GUI alone

First view

The *GFTbox* user interface. Focus on the *Select Tool* Panel. Here the *Mesh Editor* has been selected and the *Mesh Editor* panel is visible.



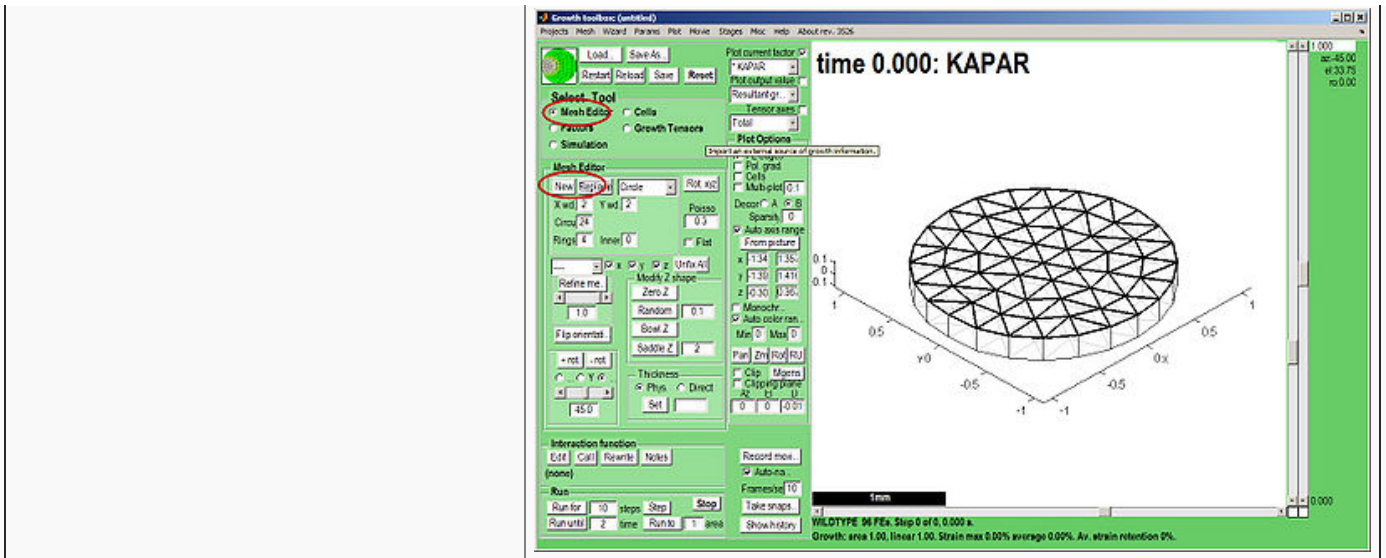
The screenshot displays the *GFTbox* software interface. The 'Select Tool' panel is active, with 'Mesh Editor' selected. The 'Mesh Editor' panel shows various settings for creating a mesh, including 'New', 'Replace', and 'Circle' options, along with dimensions like 'X wd', 'Y wd', 'Circu', 'Rings', and 'Inner'. The 'Plot Options' section includes checkboxes for 'FE edges', 'Pol. grad', 'Cells', and 'Multi-plot'. The 'Interaction function' panel at the bottom shows 'Run' and 'Stop' buttons.

Idea: change all *GFTbox* font sizes using *Menu Misc: Gui Format*

Add a mesh (canvas that represents tissue)

Create a canvas by clicking on *Panel Mesh Editor: new*. A variety of preset shapes are available, here we choose a Circle with 4 concentric rings of nodes that is 2 mm in diameter.

Idea: Special shapes can be imported as *OBJ* files.



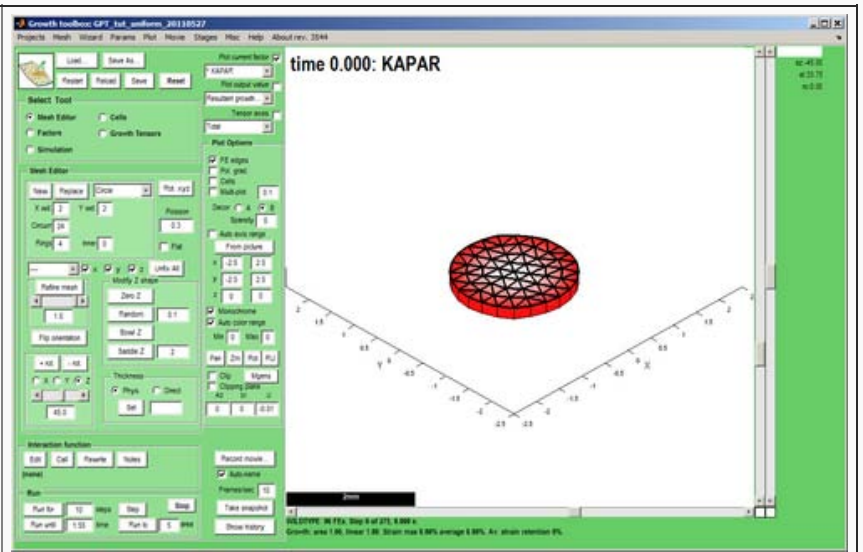
Add a pattern of growth factor

Select Tool: Factors makes the Morphogen Panel visible. Currently we are controlling the KAPAR factor. It has been selected from a drop down list top right (Plot current factor). We add a radially increasing gradient of KAPAR. Make sure that the Panel Plot Options: Monochrome is ticked and the gradient of KAPAR will be as shown. Do the same for the top face, KBPAR.

What will be the result of growth on just the bottom side of the canvas, KAPAR?

Note: the different ways to fix the levels of morphogens - 'Add constant', 'Add random', 'Add to rim', 'Add radial' centred on x,y,z position, 'Add linear'. In addition, if Panel: Plot Options: FE edges is ticked to reveal the mesh, clicking on a node will add the specified amount (slider) to that node alone. To set values (as opposed to adding) use 'Set zero' or select 'Set' in the list box.

Colours are set by clicking on the colour boxes.



And grow

Now save the project using the Save As button. A dialogue box will open - here we keyed in *in_the_beginning* and the prefix *GPT_* and the suffix *_date* was added automatically.

A directory is created with the project name which contains all the files needed for the project and any results (snapshots, movies) we create.

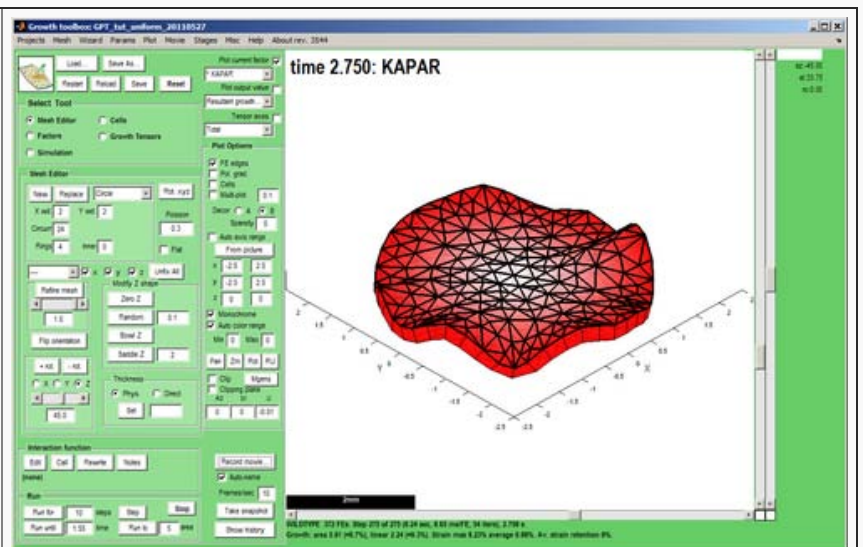
Select Tool: Simulation makes the Simulation Panel visible.

Change the Panel Run: Run until box to 2 and click the Panel Run: Run until button. The canvas will grow in steps set by the Panel Simulation: Timescale box.

Warning: saving is essential if you want to Restart the simulation.

Idea: set the axes manually (here to +-3) and turn off Panel: Plot Options: Auto axis range and Menu: Plot: Auto zoom and centre. This enables growth to be seen as a change of size rather than change of axis range. Idea: take a snapshot of the canvas by pressing Take snap... (c.f. image on right, button is bottom right of GUI controls)

Idea: to create an uncompressed movie of your simulation, before running click the Record movie and when it is finished click Stop movie. The movie will be in the Project directory. Compressors can be selected using the Menu: Movie: Codec option and the result can be converted to a flash movie (for the web) by Menu: Movie: Convert to flash this, however, requires ffmpeg to be installed on the computer.



Extra growth around the perimeter introduces Gaussian curvature. The canvas is readily able to distort out the plane because it started out with a small amount of random noise in the Z direction. (If the canvas started perfectly flat, residual strain (stress) would build up at each step rather than the surface distorting to accommodate the extra growth.)

In the beginning Growing a cone

[Back to Tutorial pages](#)

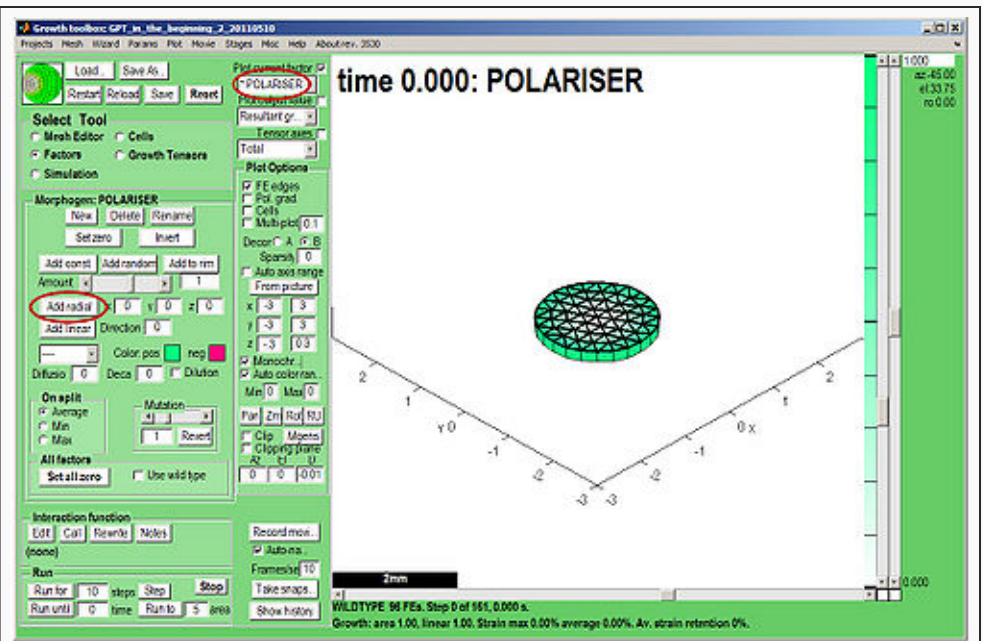
Having already produced a disc with uniform growth on A and B sides (see First tutorial)

Add a pattern of polariser

Select *Tool: Factors* makes the *Morphogen Panel* visible. Currently we are controlling the *Polariser* factor. It has been selected from a drop down list top right (*Plot current factor*). We add a radially increasing gradient of *Polariser*. Make sure that the *Panel Plot Options: Monochrome* is ticked and the gradient of *Polariser* will be as shown. Turn on *Panel: Plot Options: Pol. grad.* and turn off *Panel: Plot Options: FE edges*.

Then, when growth starts, arrows will point down the polariser gradient.

What will be the result of growth?



And grow

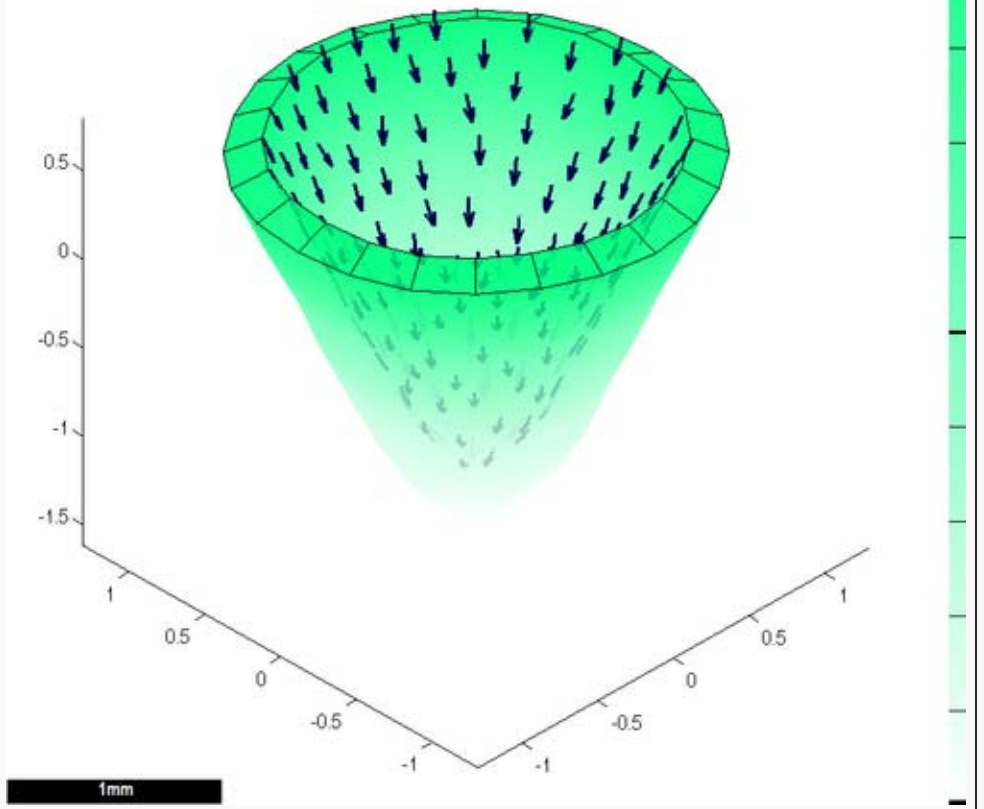
Now save the project using the *Save As* button.

Change the *Panel Run: Run until* box to 3 and click the *Panel Run: Run until* button. The canvas will grow in steps set by the *Panel Simulation: Timescale* box.

Idea: To see the arrows more clearly reduce the number by setting *Panel: Plot Options: Sparsity* to 0.1. Larger values reduce the number of arrows and makes them larger

Note: there are options to programmatically change the size, colour and density of arrows.

time 1.000: POLARISER



modified on 3 June 2011 at 11:04 *** 194 views

University of East Anglia / School of Computing Science / Powered by MediaWiki

Tutorial on the basic interaction function

[Back to tutorial pages](#)

Contents [\[hide\]](#)

[1 Creating an interaction function](#)

[1.1 Creating the interaction function from the GUI](#)

[2 Variables and utility functions](#)

[3 The interaction function in more detail](#)

Creating an interaction function

The goal is to re-create the model used to illustrate [why we use matlab](#). We will create two new variables A and B - in the interaction function they are labelled *id_a_p* and *id_b_p* (why? [see below](#)). The *x,y,z* coordinates of mesh nodes (vertices) are stored in the Mesh data structure in a field called *nodes*. Thus the *x* coordinate of node 1 is stored in *m.nodes(1,1)*, the *y* coordinate in *m.nodes(1,2)* and *z* coordinate in *m.nodes(1,3)*. The patterns of morphogens A and B are set up by

```
id_a_p(m.nodes(:,1)<-0.03)=1;
id_b_p(m.nodes(:,2)<-0.01)=1;
```

where *id_a_p* is the A morphogen. *m.node(:,1)* refers to the *x* coordinates of all nodes (vertices) in the mesh. The expression *(m.nodes(:,1)<-0.03)* means find all vertices with *x* coordinates that are less than -0.03.

Similarly, *(m.nodes(:,2)<-0.01)* means find all vertices with *y* coordinates that are less than -0.01.

The model is articulated in the following code.

```
if (Steps(m)==0) && m.globalDynamicProps.doinit % Initialisation code.
    id_a_p(m.nodes(:,1)<-0.03)=1; % setup region for A where identity factor A is represented by id_a_p
    id_b_p(m.nodes(:,2)<-0.01)=1; % setup region for B
else
    % @@KRN Growth Regulatory Network
    kapar_p(:) = id_a_l .* inh(1,id_b_l); % growth rate
    kaper_p(:) = kapar_p; % isotropic growth
    kbpar_p(:) = kapar_p; % same on both sides of the sheet
    kbper_p(:) = kapar_p; % same
    knor_p(:) = 0; % thickness not growing
end
```

Creating the interaction function from the GUI

The *GFtbox* user interface. Start a new project, with a disc shaped mesh and add two new morphogens *id_a* and *id_b* (Panel: Morphogens: New).

Save As

Now create and edit the interaction function by clicking on (Panel: Interaction function: Edit) (bottom left)

There is lots of 'green stuff' - lines that are comments. Just for the moment we will more than ignore them - delete them. Look at the listing below to get an idea of what to delete. Now in

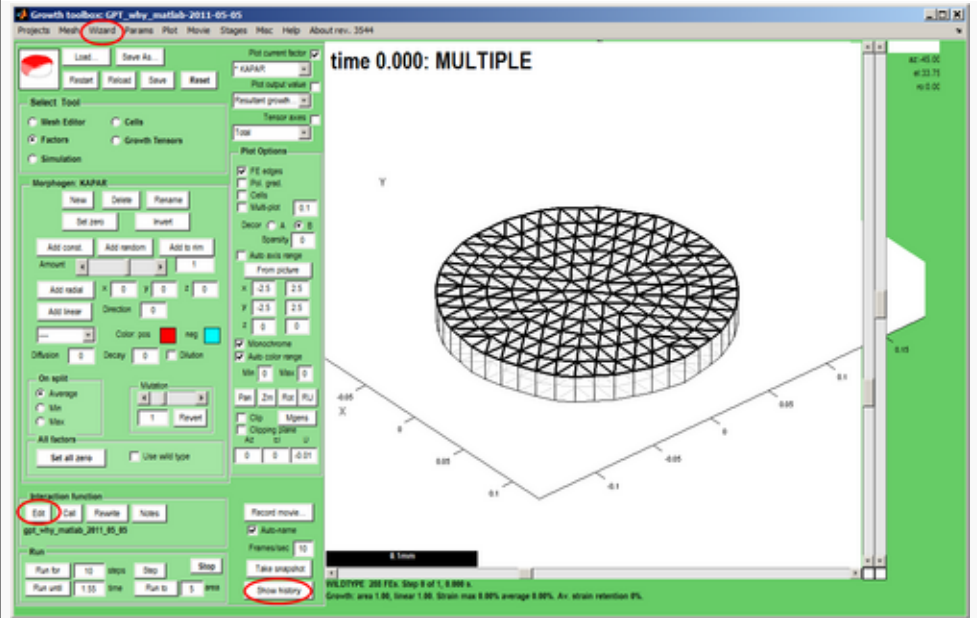
Section 4 add the code shown above - it is shown in the listing below.

A free-text file with the same name as the interaction function is also created automatically on pressing Panel:Interaction function: notes. This makes it easier to keep notes on what you are doing.

Note: you can see the internal data structure by clicking Menu: Wizard: Export mesh and follow the instructions shown at the command line.

Another way to see Mesh m is to set a breakpoint in the interaction function and running (e.g. Panel: Run: Step, or Panel: Interaction function: Call, etc.)

Note: a history of commands that have been run recently can be displayed (can be used to help write custom modelling programs that exploit the Gftbox library)



Variables and utility functions

Variables that need to be seen in the GUI (morphogens, growth factors, signals, etc.) are always set up (declared) from the GUI using Morphogen Panel. We have evolved a naming convention

| | | |
|----------|----------------|------------------------|
| prefixes | id_ | identity factor |
| | s_ | signalling factor |
| | v_ | visual monitor |
| | f_ | finite element monitor |
| suffixes | _p, _l, _a, _i | |

Full details of types of morphogens and factors

| | | |
|------------------------------|--------------------------|---|
| Pre-defined variables | dt | computational step size |
| | realtime | virtual time of the model |
| Pre-defined functions | Steps(m) | current step number in Mesh m |
| | pro(k,id) | promote by k in regions designated by factor, id ($pro(k, \mathbf{x}) = 1 + k\mathbf{x}$) |
| | inh(k,id) | inhibit by k in regions designated by factor, id ($inh(k, \mathbf{x}) = 1 / (1 + k\mathbf{x})$) |
| | local_setproperties(m) | initialise Mesh properties |
| | leaf_*(m, *) | A large library of functions for manipulating the Mesh data structure, m. Many are listed in the GUI help system, Menu:Help |

The interaction function in more detail

A large number of comments have been deleted to allow us to focus on the essential components. (What are these deleted comments? They provide hints and clues on how to program a model - many people use the system and we found the easiest way

for people to remember how to program models is to provide comments in the appropriate place - which can be deleted when not needed. Ok so you want see an uncensored version, [here is a full freshly minted interaction function](#). [It is discussed in more detail here.](#))

This part is generated automatically from the GUI

The name matches the project name (the project name is forced into lowercase), GFTbox revision number refers to our source code control, lines starting with % are comments

```
function m = gpt_why_matlab_2011_05_05( m )
% m = gpt_why_matlab_2011_05_05( m )
% Morphogen interaction function.
% Written at 2011-05-28 08:08:11.
% GFTbox revision 3544, 2011-05-25 13:37:23.907904.
```

```
% The user may edit any part of this function between delimiters
% of the form "USER CODE..." and "END OF USER CODE...". The
% delimiters themselves must not be moved, edited, deleted, or added.
```

```
if isempty(m), return; end
```

```
fprintf( 1, '%s found in %s\n', mfilename(), which(mfilename()) );
```

```
try
```

```
    m = local_setproperties( m );
```

```
catch
```

```
end
```

```
    realtime = m.globalDynamicProps.currenttime;
```

```
%%% USER CODE: INITIALISATION
```

Section 2 This part could contain user code

but in this case there is nothing we need to do before the system extracts information from the Mesh data structure

```
%%% END OF USER CODE: INITIALISATION
```

Section 3 This part is generated automatically from the GUI

Variables are extracted from Mesh and the mesh itself is summarised in a set of comments

```
%%% SECTION 1: ACCESSING MORPHOGENS AND TIME.
```

```
%%% AUTOMATICALLY GENERATED CODE: DO NOT EDIT.
```

```
if isempty(m), return; end
```

```
setGlobals();
```

```
global gNEW_KA_PAR gNEW_KA_PER gNEW_KB_PAR gNEW_KB_PER
```

```
global gNEW_K_NOR gNEW_POLARISER gNEW_STRAINRET gNEW_ARREST
```

```
dt = m.globalProps.timestep;
```

```
polariser_i = gNEW_POLARISER;
```

```
P = m.morphogens(:,polariser_i);
```

```
[kpar_i,kpar_p,kpar_a,kpar_l] = getMgenLevels( m, 'KAPAR' );
```

```
[kaper_i,kaper_p,kaper_a,kaper_l] = getMgenLevels( m, 'KAPER' );
```

```
[kpar_i,kpar_p,kpar_a,kpar_l] = getMgenLevels( m, 'KAPAR' );
```

```
[kbpar_i,kbpar_p,kbpar_a,kbpar_l] = getMgenLevels( m, 'KBPAR' );
```

```
[kbper_i,kbper_p,kbper_a,kbper_l] = getMgenLevels( m, 'KBPER' );
```

```
[knor_i,knor_p,knor_a,knor_l] = getMgenLevels( m, 'KNOR' );
```

```
[strainret_i,strainret_p,strainret_a,strainret_l] = getMgenLevels( m, 'STRAINRET' );
```

```
[arrest_i,arrest_p,arrest_a,arrest_l] = getMgenLevels( m, 'ARREST' );
```

```
[id_a_i,id_a_p,id_a_a,id_a_l] = getMgenLevels( m, 'ID_A' );
```

```
[id_b_i,id_b_p,id_b_a,id_b_l] = getMgenLevels( m, 'ID_B' );
```

```
% Mesh type: circle
% centre: 0
% circumpts: 48
% coneangle: 0
% dealign: 0
% height: 0
% innerpts: 0
% randomness: 0.1
% rings: 6
% version: 1
% xwidth: 0.2
% ywidth: 0.2
```

```
% Morphogen Diffusion Decay Dilution Mutant
% -----
% KAPAR -----
% KAPER -----
% KBPAR -----
% KBPER -----
% KNOR -----
% POLARISER -----
% STRAINRET -----
% ARREST -----
% ID_A -----
% ID_B -----
```

Section 4 This part contains user code that articulates the model

```
%%% USER CODE: MORPHOGEN INTERACTIONS
```

```
if (Steps(m)==0) && m.globalDynamicProps.doinit % Initialisation code.
```

```
    id_a_p(m.nodes(:,1)<-0.03)=1; % setup region for A
```

```
    id_b_p(m.nodes(:,2)<-0.01)=1; % setup region for B
```

```
else
```

```
    % @@KRN Growth Regulatory Network
```

```
    kpar_p(:) = id_a_l .* inh(1,id_b_l); % growth rate
```

```
    kaper_p(:) = kpar_p; % isotropic growth
```

```
    kbpar_p(:) = kpar_p; % same on both sides of the sheet
```

```
    kbper_p(:) = kpar_p; % same
```

```
    knor_p(:) = 0; % thickness not growing
```

```
end
```

Section 5 This part is generated automatically from the GUI

Variables are put back into the data structure (Mesh).

Note: only variables with the _p suffix (promoters) are stored (levels _l can be computed on the fly and activities _a are generally set in the GUI - they can only be changed in the interaction function by altering the data structure itself.

```
%%% END OF USER CODE: MORPHOGEN INTERACTIONS
```

```
%%% SECTION 6: INSTALLING MODIFIED VALUES BACK INTO MESH STRUCTURE
```

```
%%% AUTOMATICALLY GENERATED CODE: DO NOT EDIT.
```

```
m.morphogens(:,polariser_i) = P;
```

```
m.morphogens(:,kapar_i) = kapar_p;
m.morphogens(:,kaper_i) = kaper_p;
m.morphogens(:,kbpar_i) = kbpar_p;
m.morphogens(:,kbper_i) = kbper_p;
m.morphogens(:,knor_i) = knor_p;
m.morphogens(:,strainret_i) = strainret_p;
m.morphogens(:,arrest_i) = arrest_p;
m.morphogens(:,id_a_i) = id_a_p;
m.morphogens(:,id_b_i) = id_b_p;
%%% USER CODE: FINALISATION

%%% END OF USER CODE: FINALISATION

end

%%% USER CODE: SUBFUNCTIONS

Section 5
function m = local_setproperties( m )
end
```

modified on 6 June 2011 at 17:36 *** 545 views

University of East Anglia / School of Computing Science / Powered by MediaWiki

Full freshly minted interaction function

[Back to tutorial pages](#)

[Tutorial on the basic interaction function](#)

The large number of comments do two things. Firstly, they provide **guidance on how to structure a growth model** - in other words how one might organise thoughts on how things grow - the polarity regulation network, the gene regulation network and the specified growth regulation network. Secondly, they **help us remember how** to do certain things without having to look them up in the manual (that is not always complete and up to date). It is easier to delete what is not wanted than to look up and add what is wanted.

Remember: a GFTbox model comprises two parts, the Mesh and the Interaction function. At the point that the interaction function is created the Mesh exists (shape and general physical properties) but the regulatory systems do not - it is these that you add to the interaction function. A loose analogy is to say that the Mesh corresponds to the cell membranes and cytoplasm (the structure) and the Interaction function corresponds to the nucleus and DNA. You need both to grow - and they must be compatible with each other.

[This file is output automatically on clicking Panel: Edit in a newly saved project.](#)

Once the file has been edited save it in the usual way from the Matlab editor. The project as whole can be saved under a different name using Panel: Save as. This will automatically create a new copy of the current interaction function, but with the new name and in the new project directory.

The code has been divided into Sections to help with the tutorial.

```

% Section 1
function m = gpt_tut_interaction_example_20110601( m )
m = gpt_tut_interaction_example_20110601( m )
% Morphogen interaction function.
% Written at 2011-06-02 14:55:46.
% GFTbox revision 3554, 2011-06-02 12:30:52.789869.

% The user may edit any part of this function between delimiters
% of the form "USER CODE..." and "END OF USER CODE...". The
% delimiters themselves must not be moved, edited, deleted, or added.

if isempty(m), return; end

fprintf( 1, '%s found in %s\n', mfilename(), which(mfilename()) );

try
    m = local_setproperties( m );
catch
end

realtime = m.globalDynamicProps.currenttime;

% Section 2
%%% USER CODE: INITIALISATION

% In this section you may modify the mesh in any way whatsoever.
if (Steps(m)==0) && m.globalDynamicProps.doinit % First iteration
% Zero out a lot of stuff to create a blank slate.
% If no morphogens are set in the GUI it may be useful to
% zero some arrays by uncommenting the following.
% m.morphogens(:) = 0;
% m.morphogenclamp(:) = 0;
% m.mgen_production(:) = 0;
% m.mgen_absorption(:) = 0;
% m.seams(:) = false;
% m.mgen_dilution(:) = false;

% Set up names for variant models. Useful for running multiple models on a cluster.
m.userdata.ranges.modelname.range = { 'MODEL1', 'MODEL2' }; % CLUSTER
m.userdata.ranges.modelname.index = 1; % CLUSTER
end
modelname = m.userdata.ranges.modelname.range{m.userdata.ranges.modelname.index}; % CLUSTER
disp(sprintf('\nRunning %s model %s\n',mfilename, modelname));
switch modelname
    case 'MODEL1'
        % Set up the parameters (e.g. mutations) for this model here.
    case 'MODEL2'
        % Set up the parameters (e.g. mutations) for this model here.
    otherwise
        % If you reach here, you probably forgot a case.
end

% More examples of code for all iterations.

```

```

% Set priorities for simultaneous plotting of multiple morphogens, if desired.
% m = leaf_mgen_plotpriority( m, {'MGEN1', 'MGEN2'}, [1,2], [0.5,0.75] );

% Set colour of polariser gradient arrows.
% m = leaf_plotoptions(m, 'highgradcolor', [0,0,0], 'lowgradcolor', [1,0,0]);

% setup a multiplot of the following morphogens
% m = leaf_plotoptions( m, 'morphogen', {'V_PROFILE1', 'V_PROFILE2', 'KAPAR', 'S_LEFTRIGHT'});

% to plot polariser on the A side and resultant areal growth rate on the B side:
% m = leaf_plotoptions( m, 'morphogenA', 'POLARISER', ...
% 'outputquantityB', 'resultantgrowthrate', ...
% 'outputaxesB', 'areal' );

% monitor properties of vertices must be done here - so that it reports newly equilibrated levels
% m=leaf_profile_monitor(m,... % essential
% 'REGIONLABELS',{'V_PROFILE1','V_PROFILE2'},... % essential
% 'MORPHOGENS',{'S_LEFTRIGHT','S_CENTRE'},... % optional (one element per REGIONLABEL)
% 'VERTLABELS',false,'FigNum',1,'EXCEL',true,'MODELNAME',modelname); % optional (file in snapshots directory)

%%% END OF USER CODE: INITIALISATION

% Section 3
%%% SECTION 1: ACCESSING MORPHOGENS AND TIME.
%%% AUTOMATICALLY GENERATED CODE: DO NOT EDIT.

if isempty(m), return; end

setGlobals();
global gNEW_KA_PAR gNEW_KA_PER gNEW_KB_PAR gNEW_KB_PER
global gNEW_K_NOR gNEW_POLARISER gNEW_STRAINRET gNEW_ARREST
dt = m.globalProps.timestep;
polariser_i = gNEW_POLARISER;
P = m.morphogens(:,polariser_i);
[kapar_i,kapar_p,kapar_a,kapar_l] = getMgenLevels( m, 'KAPAR' );
[kaper_i,kaper_p,kaper_a,kaper_l] = getMgenLevels( m, 'KAPER' );
[kbpar_i,kbpar_p,kbpar_a,kbpar_l] = getMgenLevels( m, 'KBPAR' );
[kbper_i,kbper_p,kbper_a,kbper_l] = getMgenLevels( m, 'KBPER' );
[knor_i,knor_p,knor_a,knor_l] = getMgenLevels( m, 'KNOR' );
[strainret_i,strainret_p,strainret_a,strainret_l] = getMgenLevels( m, 'STRAINRET' );
[arrest_i,arrest_p,arrest_a,arrest_l] = getMgenLevels( m, 'ARREST' );
[id_a_i,id_a_p,id_a_a,id_a_l] = getMgenLevels( m, 'ID_A' );
[id_b_i,id_b_p,id_b_a,id_b_l] = getMgenLevels( m, 'ID_B' );

% Mesh type: circle
% centre: 0
% circumpts: 48
% coneangle: 0
% dealign: 0
% height: 0
% innerpts: 0
% randomness: 0.1
% rings: 6
% version: 1
% xwidth: 0.2
% ywidth: 0.2

%
% Morphogen Diffusion Decay Dilution Mutant
% -----
% KAPAR ----
% KAPER ----
% KBPAR ----
% KBPER ----
% KNOR ----
% POLARISER ----
% STRAINRET ----
% ARREST ----
% ID_A ----
% ID_B ----

%%% USER CODE: MORPHOGEN INTERACTIONS

% In this section you may modify the mesh in any way that does not
% Section 4
% alter the set of nodes.

if (Steps(m)==0) && m.globalDynamicProps.doinit % Initialisation code.
% Put any code here that should only be performed at the start of
% the simulation, for example, to set up initial morphogen values.

% m.nodes is the set of vertex positions, an N by 3 array if there
% are N vertices. Row number K contains the X, Y, and Z
% coordinates of the Kth vertex. To obtain a list of the X
% coordinates of every vertex, write m.nodes(:,1). The Y
% coordinates are given by m.nodes(:,2) and the Z coordinates by
% m.nodes(:,3).

% Set up a morphogen promoter (_p suffix) region where x values are minimum
% id_prox_p(m.nodes(:,1))==min(m.nodes(:,1));
% if the morphogen level (_l suffix) is to be used in this iteration
% set the level using the morphogen activity (_a suffix).
% id_prox_l=id_prox_p * id_prox_a; % when a mutation is specified in the GUI
% the activity (_a) is set to zero

% One way to set up a morphogen gradient is by ...
% Setting up a gradient by clamping the ends (execute only once)
% P=id_prox_p;
% m.morphogenclamp( ((id_prox_p==1)|(id_dist_p==1)), polariser_i ) = 1;
% m = leaf_mgen_conductivity( m, 'POLARISER', 0.01 ); %specifies the diffusion rate of polariser
% m = leaf_mgen_absorption( m, 'POLARISER', 0.1 ); % specifies degradation rate of polariser

% Fixing vertices, i.e. fix z for the base to prevent base from moving up or down
% m=leaf_fix_vertex(m,'vertex',find(id_prox_p==1),'dfs','z');

% To cut the mesh, set a temporary morphogen to 1 in places to cut

```

```

% seams=zeros(size(P));
% seams(indexes to places to cut)=1;
% m=leaf_set_seams(m,seams);

end

% Section 5
% Second way to generate a gradient
% generating (+) and sinking (-) a diffusing signal (in this case polariser)
% m.mgen_production( :, polariser_i ) = + 5*s_spur_p - P .* id_dist_p;

% Monitor growth by scattering discs that deform over time (c.f. inducing biological clones)
% (CARE - if the canvas is flat ensure that Plot:Hide Thickness is true,
% because a quirk of the Matlab z-buffer means that they can get hidden by mistake)
% if (340>realtime-dt) && (340<realtime+dt) % discs to be added at realtime==340
%     m = leaf_makesecondlayer( m, ... % This function adds discs that represent transformed cells.
%     'mode', 'each', ... % Make discs randomly scattered over the canvas.
%     'relarea', 1/16000, ... % Each disc has area was 1/16000 of the initial area of the canvas.
%     'probpervx', 'V_FLOWER', ... % induce discs over whole canvas (V_FLOWER is 1 over whole canvas)
%     'numcells',4500,...%number of discs (that will become ellipses)
%     'sides', 6, ... % Each disc is approximated as a 6-sided regular polygon.
%     'colors', [0.5 0.5 0.5], ... % Default colour is gray but
%     'colorvariation',1,... % Each disc is a random colour
%     'add', true ); % These discs are added to any discs existing already
% end

% Section 6
% Directives for creating latex representation directly from Matlab code
% not fully implemented yet but will use @@ directives
% @@at t
% @@before t
% @@after t
% @@between t1 t2

% Section 7
% If you want to define different phases according to the absolute
% time, create a morphogen for each phase and modulate
% expressions using the morphogen
% like. For example:
% if (realtime < 10) % first growth phase
%     f_firstgrowth_p = 1;
% else
%     f_firstgrowth_p = 0;
% end
% if (realtime >= 10) % second growth phase
%     f_secondgrowth_p = 1;
% else
%     f_secondgrowth_p = 0;
% end
%
% If you want one morphogen to affect others only during a certain
% phase, write something like:
%
% mgen_a_p = f_firstgrowth_p .* (various terms); % will zero except in firstgrowth

% Section 8
% Code common to all models.
% @@PRN Polariser Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% @@GRN Gene Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% @@KRN Growth Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.

% Section 9
% Code for specific models.
switch modelname
% Section 10
case 'MODEL1' % @@model MODEL1
% @@PRN Polariser Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% P(:) = ... % @@ Eqn xx
% @@GRN Gene Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% @@KRN Growth Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% kapar_p(:) = 0; % @@ Eqn xx
% kaper_p(:) = 0; % @@ Eqn xx
% kbpar_p(:) = 0; % @@ Eqn xx
% kbper_p(:) = 0; % @@ Eqn xx
% knor_p(:) = 0; % @@ Eqn xx
% Section 11
case 'MODEL2' % @@model MODEL2
% @@PRN Polariser Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% P(:) = ... % @@ Eqn xx
% @@GRN Gene Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% @@KRN Growth Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% kapar_p(:) = 0; % @@ Eqn xx
% kaper_p(:) = 0; % @@ Eqn xx
% kbpar_p(:) = 0; % @@ Eqn xx
% kbper_p(:) = 0; % @@ Eqn xx
% knor_p(:) = 0; % @@ Eqn xx
otherwise
% If this happens, maybe you forgot a model.
end

% Section 12
%%% END OF USER CODE: MORPHOGEN INTERACTIONS

%%% SECTION 3: INSTALLING MODIFIED VALUES BACK INTO MESH STRUCTURE
%%% AUTOMATICALLY GENERATED CODE: DO NOT EDIT.
m.morphogens(:,polariser_i) = P;
m.morphogens(:,kapar_i) = kapar_p;
m.morphogens(:,kaper_i) = kaper_p;
m.morphogens(:,kbpar_i) = kbpar_p;

```

```

m.morphogens(:,kbper_i) = kbper_p;
m.morphogens(:,knor_i) = knor_p;
m.morphogens(:,strainret_i) = strainret_p;
m.morphogens(:,arrest_i) = arrest_p;
m.morphogens(:,id_a_i) = id_a_p;
m.morphogens(:,id_b_i) = id_b_p;

%%% USER CODE: FINALISATION

% In this section you may modify the mesh in any way whatsoever.

% Section 13
% If needed force FE to subdivide (increase number FE's) here
% if realtime==280*dt
%     m = leaf_subdivide( m, 'morphogen','id_vent',...
%         'min',0.5,'max',1,...
%         'mode','mid','levels','all');
% end
% Cut the mesh along the seams (see above)
% if m.userdata.CutOpen==1
%     m=leaf_dissect(m);
%     m.userdata.CutOpen=2;
%     Relax accumulated stresses slowly i.e. 0.95 to 0.999
%     m = leaf_setproperty( m, 'freezing', 0.999 );
% end

%%% END OF USER CODE: FINALISATION

end

%%% USER CODE: SUBFUNCTIONS

% Section 14
function m = local_setproperties( m )
% This function is called at time zero in the INITIALISATION section of the
% interaction function. It provides commands to set each of the properties
% that are contained in m.globalProps. Uncomment whichever ones you would
% like to set yourself, and put in whatever value you want.
%
% Some of these properties are for internal use only and should never be
% set by the user. At some point these will be moved into a different
% component of m, but for the present, just don't change anything unless
% you know what it is you're changing.

% m = leaf_setproperty( m, 'trinodesvalid', true );
% m = leaf_setproperty( m, 'prismnodesvalid', true );
% m = leaf_setproperty( m, 'thicknessRelative', 0.020000 );
% m = leaf_setproperty( m, 'thicknessArea', 0.000000 );
% m = leaf_setproperty( m, 'thicknessMode', 'physical' );
% m = leaf_setproperty( m, 'activeGrowth', 1.000000 );
% m = leaf_setproperty( m, 'displayedGrowth', 1.000000 );
% m = leaf_setproperty( m, 'displayedMulti', [] );
% m = leaf_setproperty( m, 'allowNegativeGrowth', true );
% m = leaf_setproperty( m, 'usePrevDispAsEstimate', true );
% m = leaf_setproperty( m, 'perturbInitGrowthEstimate', 0.000010 );
% m = leaf_setproperty( m, 'perturbRelGrowthEstimate', 0.010000 );
% m = leaf_setproperty( m, 'perturbDiffusionEstimate', 0.000100 );
% m = leaf_setproperty( m, 'resetRand', false );
% m = leaf_setproperty( m, 'mingradient', 0.000000 );
% m = leaf_setproperty( m, 'relativepolgrad', false );
% m = leaf_setproperty( m, 'usefrozengradient', true );
% m = leaf_setproperty( m, 'userpolarisation', false );
% m = leaf_setproperty( m, 'thresholdsq', 0.000841 );
% m = leaf_setproperty( m, 'splitmargin', 1.400000 );
% m = leaf_setproperty( m, 'splitmorphogen', );
% m = leaf_setproperty( m, 'thresholdmgen', 0.500000 );
% m = leaf_setproperty( m, 'bulkmodulus', 1.000000 );
% m = leaf_setproperty( m, 'unitbulkmodulus', true );
% m = leaf_setproperty( m, 'poissonsRatio', 0.300000 );
% m = leaf_setproperty( m, 'starttime', 0.000000 );
% m = leaf_setproperty( m, 'timestep', 0.010000 );
% m = leaf_setproperty( m, 'timeunitname', );
% m = leaf_setproperty( m, 'distunitname', 'mm' );
% m = leaf_setproperty( m, 'scalebarvalue', 0.000000 );
% m = leaf_setproperty( m, 'validateMesh', true );
% m = leaf_setproperty( m, 'rectifyverticals', false );
% m = leaf_setproperty( m, 'allowSplitLongFEM', true );
% m = leaf_setproperty( m, 'longSplitThresholdPower', 0.000000 );
% m = leaf_setproperty( m, 'allowSplitBentFEM', false );
% m = leaf_setproperty( m, 'allowSplitBio', true );
% m = leaf_setproperty( m, 'allowFlipEdges', false );
% m = leaf_setproperty( m, 'allowElideEdges', true );
% m = leaf_setproperty( m, 'mincellangle', 0.200000 );
% m = leaf_setproperty( m, 'alwaysFlat', 0.000000 );
% m = leaf_setproperty( m, 'flattenforceconvex', true );
% m = leaf_setproperty( m, 'flatten', false );
% m = leaf_setproperty( m, 'flattenratio', 1.000000 );
% m = leaf_setproperty( m, 'useGrowthTensors', false );
% m = leaf_setproperty( m, 'plasticGrowth', false );
% m = leaf_setproperty( m, 'totalinternalrotation', 0.000000 );
% m = leaf_setproperty( m, 'stepinternalrotation', 2.000000 );
% m = leaf_setproperty( m, 'showinternalrotation', false );
% m = leaf_setproperty( m, 'performinternalrotation', false );
% m = leaf_setproperty( m, 'internallyrotated', false );
% m = leaf_setproperty( m, 'maxFEcells', 0 );
% m = leaf_setproperty( m, 'inittotalcells', 0 );
% m = leaf_setproperty( m, 'bioApostsplitproc', );
% m = leaf_setproperty( m, 'bioApostsplitproc', );
% m = leaf_setproperty( m, 'maxBioAcells', 0 );
% m = leaf_setproperty( m, 'maxBioBcells', 0 );
% m = leaf_setproperty( m, 'colors', (6 values) );
% m = leaf_setproperty( m, 'colorvariation', 0.050000 );
% m = leaf_setproperty( m, 'colorparams', (12 values) );
% m = leaf_setproperty( m, 'freezing', 0.000000 );
% m = leaf_setproperty( m, 'canceledrift', false );
% m = leaf_setproperty( m, 'mgen_interaction', );
% m = leaf_setproperty( m, 'mgen_interactionName', 'gpt_temp_fresh_20110602' );

```



```

% m = leaf_setproperty( m, 'allowInteraction', true );
% m = leaf_setproperty( m, 'interactionValid', true );
% m = leaf_setproperty( m, 'gaussInfo', (unknown type struct) );
% m = leaf_setproperty( m, 'stitchDFs', [] );
% m = leaf_setproperty( m, 'D', (36 values) );
% m = leaf_setproperty( m, 'C', (36 values) );
% m = leaf_setproperty( m, 'G', (6 values) );
% m = leaf_setproperty( m, 'solver', 'cgs' );
% m = leaf_setproperty( m, 'solverprecision', 'double' );
% m = leaf_setproperty( m, 'solvertolerance', 0.001000 );
% m = leaf_setproperty( m, 'solvertolerancemethod', 'norm' );
% m = leaf_setproperty( m, 'diffusiointolerance', 0.000010 );
% m = leaf_setproperty( m, 'allowsparse', true );
% m = leaf_setproperty( m, 'maxIters', 0 );
% m = leaf_setproperty( m, 'maxsolvetime', 1000.000000 );
% m = leaf_setproperty( m, 'cgiters', 0 );
% m = leaf_setproperty( m, 'simsteps', 0 );
% m = leaf_setproperty( m, 'stepsperrorrender', 0 );
% m = leaf_setproperty( m, 'growthEnabled', true );
% m = leaf_setproperty( m, 'diffusionEnabled', true );
% m = leaf_setproperty( m, 'flashmovie', false );
% m = leaf_setproperty( m, 'makemovie', false );
% m = leaf_setproperty( m, 'moviefile', );
% m = leaf_setproperty( m, 'codec', 'None' );
% m = leaf_setproperty( m, 'autonamemovie', true );
% m = leaf_setproperty( m, 'overwritemovie', false );
% m = leaf_setproperty( m, 'framesize', [] );
% m = leaf_setproperty( m, 'mov', [] );
% m = leaf_setproperty( m, 'jiggleProportion', 1.000000 );
% m = leaf_setproperty( m, 'cvtperiter', 0.200000 );
% m = leaf_setproperty( m, 'boingNeeded', false );
% m = leaf_setproperty( m, 'initialArea', 0.031326 );
% m = leaf_setproperty( m, 'bendunitlength', 0.176992 );
% m = leaf_setproperty( m, 'targetRelArea', 1.000000 );
% m = leaf_setproperty( m, 'defaultinterp', 'min' );
% m = leaf_setproperty( m, 'readonly', false );
% m = leaf_setproperty( m, 'projectdir', 'D:\ab\Matlab stuff' );
% m = leaf_setproperty( m, 'modelname', 'GPT_temp_fresh_20110602' );
% m = leaf_setproperty( m, 'allowsave', true );
% m = leaf_setproperty( m, 'addedToPath', false );
% m = leaf_setproperty( m, 'bendsplit', 0.300000 );
% m = leaf_setproperty( m, 'usepolifreezebc', false );
% m = leaf_setproperty( m, 'dorsaltop', true );
% m = leaf_setproperty( m, 'defaultazimuth', -45.000000 );
% m = leaf_setproperty( m, 'defaulttelevation', 33.750000 );
% m = leaf_setproperty( m, 'defaulttroll', 0.000000 );
% m = leaf_setproperty( m, 'defaultViewParams', (unknown type struct) );
% m = leaf_setproperty( m, 'comment', );
% m = leaf_setproperty( m, 'legendTemplate', '%T: %q\n%m' );
% m = leaf_setproperty( m, 'bioAsplitcells', true );
% m = leaf_setproperty( m, 'bioApullin', 0.142857 );
% m = leaf_setproperty( m, 'bioAfakepull', 0.202073 );
% m = leaf_setproperty( m, 'interactive', false );
% m = leaf_setproperty( m, 'coderevision', 0 );
% m = leaf_setproperty( m, 'coderevisiondate', );
% m = leaf_setproperty( m, 'modelrevision', 0 );
% m = leaf_setproperty( m, 'modelrevisiondate', );
% m = leaf_setproperty( m, 'savedrunname', );
% m = leaf_setproperty( m, 'savedrundesc', );
% m = leaf_setproperty( m, 'vxgrad', (108 values) );
% m = leaf_setproperty( m, 'lengthscale', 0.200000 );
end

```

```

% Section 15
% Here you may write any functions of your own, that you want to call from
% the interaction function, but never need to call from outside it.
% Remember that they do not have access to any variables except those
% that you pass as parameters, and cannot change anything except by
% returning new values as results.
% Whichever section they are called from, they must respect the same
% restrictions on what modifications they are allowed to make to the mesh.

```

```

% For example:

```

```

% Section 16
% function m = do_something( m )
% % Change m in some way.
% end

```

```

% Call it from the main body of the interaction function like this:
% m = do_something( m );

```

modified on 2 June 2011 at 19:50 *** 404 views

Tutorial on the interaction function details

[Back to tutorial pages](#)

The aim is to illustrate how to program the interaction function to have two submodels. In this example, the first grows isotropically and the second anisotropically where the growth axis is specified by a gradient of polariser.

1) Create and save a new mesh exactly as for [the basic interaction function](#). Next, add two new morphogens, *id_a* and *id_b*. Having saved the project (here we call it *GPT_tut_interaction_20110530*), click Panel: Interaction function:Edit to automatically create the new interaction function. [It is shown here.](#)

There are lots of **comments shown in green**. These are designed to both help us structure our thoughts on the growth model and provide reminders of useful commands and syntax without having to dive into a manual.

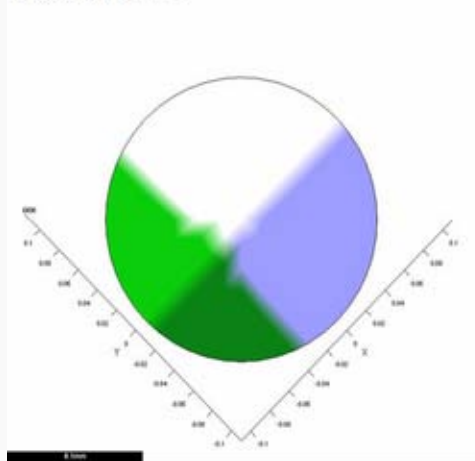
The comments are easily removed - but for the moment we will leave them in.

2) Add the [code illustrating why we chose Matlab](#) into the first model. This is shown in **Magenta** in [GPT_tut_interaction_20110530 second edit](#). Note that it is in the space left for MODEL 1 and that MODEL 1 has been renamed 'NoPolariser'.

3) Add the second model - new code is shown in **MediumBlue**. Notice that the polariser is set up in the same way for both models - but in the NoPolariser (first) model the growth rates parallel and perpendicular to the polariser are identical so the polariser gradient can have no effect. The final interaction function will look like [GPT_tut_interaction_20110530 second edit](#).

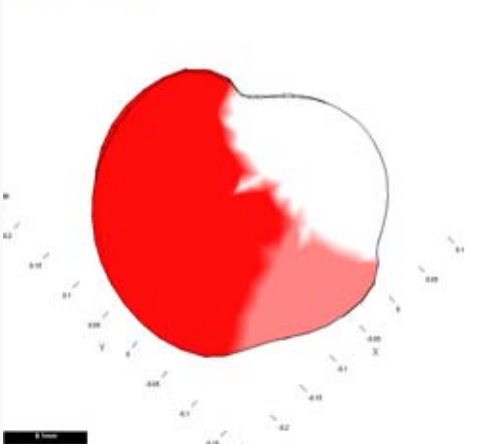
Pattern of A (blue) and B (green) morphogens at the start. Growth will be promoted by A but partially inhibited by B.

time 0.010: MULTIPLE



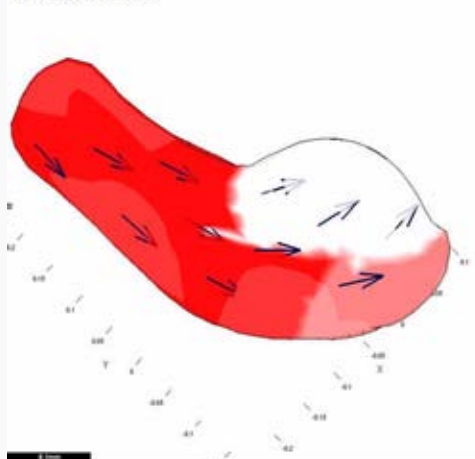
Initial shape.

time 1.280: KAPAR



Isotropic growth, i.e. specified growth parallel to and perpendicular to any polariser is identical ($K_{par}=K_{bpar}=K_{per}=K_{bper}$).

time 2.290: KAPAR



Anisotropic growth where specified growth perpendicular to the polariser (arrows) is 0.1 that parallel to the polariser.

GPT tut interaction 20110530 second edit

[Back to tutorial on the interaction function details.](#)

The green comments have not been removed in this copy - to allow you to see the context in which the new code has been placed. If you copy this function we recommend removing all the comments that you no longer need to see.

There are two submodels. The two are the same except that the first has isotropic growth (see lines shown in [magenta](#)) and the second has anisotropic growth (see lines shown in [MediumBlue](#)). The choice of submodel (1 or 2) is made on the line highlighted in RED.

```

% Section 1
function m = gpt_tut_interaction_example_20110601( m )
% m = gpt_tut_interaction_example_20110601( m )
% Morphogen interaction function.
% Written at 2011-06-02 14:55:46.
% GFTbox revision 3554, 2011-06-02 12:30:52.789869.

% The user may edit any part of this function between delimiters
% of the form "USER CODE..." and "END OF USER CODE...". The
% delimiters themselves must not be moved, edited, deleted, or added.

    if isempty(m), return; end

    fprintf( 1, '%s found in %s\n', mfilename(), which(mfilename()) );

    try
        m = local_setproperties( m );
    catch
    end

    realtime = m.globalDynamicProps.currenttime;

% Section 2
%%% USER CODE: INITIALISATION

% In this section you may modify the mesh in any way whatsoever.
if (Steps(m)==0) && m.globalDynamicProps.doinit % First iteration
    % Zero out a lot of stuff to create a blank slate.
    % If no morphogens are set in the GUI it may be useful to
    % zero some arrays by uncommenting the following.
    % m.morphogens(:) = 0;
    % m.morphogenclamp(:) = 0;
    % m.mgen_production(:) = 0;
    % m.mgen_absorption(:) = 0;
    % m.seams(:) = false;
    % m.mgen_dilution(:) = false;

    % Set up names for variant models. Useful for running multiple models on a cluster.
    m.userdata.ranges.modelname.range = { 'NoPolariser', 'WithPolariser' }; % CLUSTER
    m.userdata.ranges.modelname.index = 2; % CLUSTER
end
modelname = m.userdata.ranges.modelname.range{m.userdata.ranges.modelname.index}; % CLUSTER
disp(sprintf('\nRunning %s model %s\n',mfilename, modelname));

% More examples of code for all iterations.

% Set priorities for simultaneous plotting of multiple morphogens, if desired.
% m = leaf_mgen_plotpriority( m, {'MGEN1', 'MGEN2'}, [1,2], [0.5,0.75] );

% Set colour of polariser gradient arrows.
% m = leaf_plotoptions(m, 'highgradcolor', [0,0,0], 'lowgradcolor', [1,0,0]);

% setup a multiplot of the following morphogens
% m = leaf_plotoptions( m, 'morphogen', {'V_PROFILE1', 'V_PROFILE2', 'KAPAR', 'S_LEFTRIGHT'});

% to plot polariser on the A side and resultant areal growth rate on the B side:
% m = leaf_plotoptions( m, 'morphogenA', 'POLARISER', ...
% 'outputquantityB', 'resultantgrowthrate', ...
% 'outputaxesB', 'areal' );

% monitor properties of vertices must be done here - so that it reports newly equilibrated levels
% m=leaf_profile_monitor(m,... % essential
% 'REGIONLABELS',{'V_PROFILE1','V_PROFILE2'},... % essential
% 'MORPHOGENS',{'S_LEFTRIGHT','S_CENTRE'},... % optional (one element per REGIONLABEL)
% 'VERTLABELS',false,'FigNum',1,'EXCEL',true,'MODELNAME',modelname); % optional (file in snapshots directory')
%%% END OF USER CODE: INITIALISATION

% Section 3
%%% SECTION 1: ACCESSING MORPHOGENS AND TIME.
%%% AUTOMATICALLY GENERATED CODE: DO NOT EDIT.

    if isempty(m), return; end

    setGlobals();

```

```

global gNEW_KA_PAR gNEW_KA_PER gNEW_KB_PAR gNEW_KB_PER
global gNEW_K_NOR gNEW_POLARISER gNEW_STRAINRET gNEW_ARREST
dt = m.globalProps.timestep;
polariser_i = gNEW_POLARISER;
P = m.morphogens(:,polariser_i);
[kapar_i,kapar_p,kapar_a,kapar_l] = getMgenLevels( m, 'KAPAR' );
[kaper_i,kaper_p,kaper_a,kaper_l] = getMgenLevels( m, 'KAPER' );
[kbpar_i,kbpar_p,kbpar_a,kbpar_l] = getMgenLevels( m, 'KBPAR' );
[kbper_i,kbper_p,kbper_a,kbper_l] = getMgenLevels( m, 'KBPER' );
[knor_i,knor_p,knor_a,knor_l] = getMgenLevels( m, 'KNOR' );
[strainret_i,strainret_p,strainret_a,strainret_l] = getMgenLevels( m, 'STRAINRET' );
[arrest_i,arrest_p,arrest_a,arrest_l] = getMgenLevels( m, 'ARREST' );
[id_a_i,id_a_p,id_a_a,id_a_l] = getMgenLevels( m, 'ID_A' );
[id_b_i,id_b_p,id_b_a,id_b_l] = getMgenLevels( m, 'ID_B' );

% Mesh type: circle
%
%   centre: 0
%   circumpnts: 48
%   coneangle: 0
%   dealign: 0
%   height: 0
%   innerpts: 0
%   randomness: 0.1
%   rings: 6
%   version: 1
%   xwidth: 0.2
%   ywidth: 0.2

%
%   Morphogen   Diffusion   Decay   Dilution   Mutant
%   -----
%   KAPAR       ----       ----       ----       ----
%   KAPER       ----       ----       ----       ----
%   KBPAR       ----       ----       ----       ----
%   KBPER       ----       ----       ----       ----
%   KNOR        ----       ----       ----       ----
%   POLARISER   0.01      0.1       ----       ----
%   STRAINRET   ----       ----       ----       ----
%   ARREST      ----       ----       ----       ----
%   ID_A        ----       ----       ----       ----
%   ID_B        ----       ----       ----       ----

%%% USER CODE: MORPHOGEN INTERACTIONS

% In this section you may modify the mesh in any way that does not
% Section 4
% alter the set of nodes.

if (Steps(m)==0) && m.globalDynamicProps.doinit % Initialisation code.
% Put any code here that should only be performed at the start of
% the simulation, for example, to set up initial morphogen values.

% m.nodes is the set of vertex positions, an N by 3 array if there
% are N vertices. Row number K contains the X, Y, and Z
% coordinates of the Kth vertex. To obtain a list of the X
% coordinates of every vertex, write m.nodes(:,1). The Y
% coordinates are given by m.nodes(:,2) and the Z coordinates by
% m.nodes(:,3).

% Set up a morphogen promoter (_p suffix) region where x values are minimum
% id_prox_p(m.nodes(:,1))==min(m.nodes(:,1))=1;
% if the morphogen level (_l suffix) is to be used in this iteration
% set the level using the morphogen activity (_a suffix).
% id_prox_l=id_prox_p * id_prox_a; % when a mutation is specified in the GUI
% the activity (_a) is set to zero

id_a_p(m.nodes(:,1)<-0.03)=1; % setup region for A where identity factor A is represented by id_a_p
id_b_p(m.nodes(:,2)<-0.01)=1; % setup region for B

% One way to set up a morphogen gradient is by ...
% Setting up a gradient by clamping the ends (execute only once)
P((m.nodes(:,1)<-0.05)&(m.nodes(:,2)>0.03))=1;
m.morphogenclamp( P=1, polariser_i ) = 1;
m = leaf_mgen_conductivity( m, 'POLARISER', 0.001 ); %specifies the diffusion rate of polariser
m = leaf_mgen_absorption( m, 'POLARISER', 0.1 ); % specifies degradation rate of polariser

% Fixing vertices, i.e. fix z for the base to prevent base from moving up or down
% m=leaf_fix_vertex(m,'vertex',find(id_prox_p=1),'dfs','z');

% To cut the mesh, set a temporary morphogen to 1 in places to cut
% seams=zeros(size(P));
% seams(indexes to places to cut)=1;
% m=leaf_set_seams(m,seams);

end

% Section 5
% Second way to generate a gradient
% generating (+) and sinking (-) a diffusing signal (in this case polariser)
% m.mgen_production( :, polariser_i ) = + 5*s_spur_p - P .* id_dist_p;

% Monitor growth by scattering discs that deform over time (c.f. inducing biological clones)
% (CARE - if the canvas is flat ensure that Plot:Hide Thickness is true,
% because a quirk of the Matlab z-buffer means that they can get hidden by mistake)
% if (340>realtime-dt) && (340<realtime+dt) % discs to be added at realtime==340
%   m = leaf_makesecondlayer( m, ... % This function adds discs that represent transformed cells.
%   'mode', 'each', ... % Make discs randomly scattered over the canvas.
%   'relarea', 1/16000, ... % Each discs has area was 1/16000 of the initial area of the canvas.
%   'probpervx', 'V_FLOWER', ... % induce discs over whole canvas (V_FLOWER is 1 over whole canvas)
%   'numcells',4500,...%number of discs (that will become ellipses)
%   'sides', 6, ... % Each discs is approximated as a 6-sided regular polygon.
%   'colors', [0.5 0.5 0.5], ... % Default colour is gray but
%   'colorvariation',1,... % Each disc is a random colour
%   'add', true ); % These discs are added to any discs existing already
% end

% Section 6
% Directives for creating latex representation directly from Matlab code

```

```

% not fully implemented yet but will use @@ directives
% @@at t
% @@before t
% @@after t
% @between t1 t2

% Section 7
% % If you want to define different phases according to the absolute
% % time, create a morphogen for each phase and modulate
% % expressions using the morphogen
% % like. For example:
% % if (realtime < 10) % first growth phase
% %     f_firstgrowth_p = 1;
% % else
% %     f_firstgrowth_p = 0;
% % end
% % if (realtime >= 10) % second growth phase
% %     f_secondgrowth_p = 1;
% % else
% %     f_secondgrowth_p = 0;
% % end
% %
% % If you want one morphogen to affect others only during a certain
% % phase, write something like:
% %
% % mgen_a_p = f_firstgrowth_p .* (various terms); % will zero except in firstgrowth

% Section 8
% Code common to all models.
% @@PRN Polariser Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% @@GRN Gene Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% @@KRN Growth Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.

% Section 9
% Code for specific models.
switch modelname
% Section 10
case 'NoPolariser' % @@model MODEL1
% @@PRN Polariser Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% % P(:) = ... % @@ Eqn xx
% @@GRN Gene Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% @@KRN Growth Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% % kapar_p(:) = 0; % @@ Eqn xx
% % kaper_p(:) = 0; % @@ Eqn xx
% % kbpar_p(:) = 0; % @@ Eqn xx
% % kbper_p(:) = 0; % @@ Eqn xx
% % knor_p(:) = 0; % @@ Eqn xx
% kapar_p(:) = id_a_1 .* inh(1,id_b_1); % growth rate
% kaper_p(:) = kapar_p; % isotropic growth
% kbpar_p(:) = kapar_p; % same on both sides of the sheet
% kbper_p(:) = kapar_p; % same
% knor_p(:) = 0; % thickness not growing
% Section 11
case 'WithPolariser' % @@model MODEL2
% @@PRN Polariser Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% % P(:) = ... % @@ Eqn xx
% @@GRN Gene Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% @@KRN Growth Regulatory Network
% % Every equation to be formatted should end with an at-at Eqn N comment.
% % kapar_p(:) = 0; % @@ Eqn xx
% % kaper_p(:) = 0; % @@ Eqn xx
% % kbpar_p(:) = 0; % @@ Eqn xx
% % kbper_p(:) = 0; % @@ Eqn xx
% % knor_p(:) = 0; % @@ Eqn xx
% kapar_p(:) = id_a_1 .* inh(1,id_b_1); % growth rate
% kaper_p(:) = 0.1*kapar_p; % anisotropic growth
% kbpar_p(:) = kapar_p; % same on both sides of the sheet
% kbper_p(:) = 0.1*kapar_p; % also anisotropic
% knor_p(:) = 0; % thickness not growing
otherwise
% % If this happens, maybe you forgot a model.
end
% Section 12
%%% END OF USER CODE: MORPHOGEN INTERACTIONS

%%% SECTION 3: INSTALLING MODIFIED VALUES BACK INTO MESH STRUCTURE
%%% AUTOMATICALLY GENERATED CODE: DO NOT EDIT.
m.morphogens(:,polariser_i) = P;
m.morphogens(:,kapar_i) = kapar_p;
m.morphogens(:,kaper_i) = kaper_p;
m.morphogens(:,kbpar_i) = kbpar_p;
m.morphogens(:,kbper_i) = kbper_p;
m.morphogens(:,knor_i) = knor_p;
m.morphogens(:,strainret_i) = strainret_p;
m.morphogens(:,arrest_i) = arrest_p;
m.morphogens(:,id_a_i) = id_a_p;
m.morphogens(:,id_b_i) = id_b_p;

%%% USER CODE: FINALISATION

% In this section you may modify the mesh in any way whatsoever.

% Section 13
% % If needed force FE to subdivide (increase number FE's) here
% % if realtime==280*dt
% %     m = leaf_subdivide( m, 'morphogen','id_vent',...
% %         'min',0.5,'max',1,...
% %         'mode','mid','levels','all');
% % end
% Cut the mesh along the seams (see above)

```

```

% if m.userdata.CutOpen==1
%   m=leaf_dissect(m);
%   m.userdata.CutOpen=2;
%   Relax accumulated stresses slowly i.e. 0.95 to 0.999
%   m = leaf_setproperty( m, 'freezing', 0.999 );
% end
%%% END OF USER CODE: FINALISATION

end

%%% USER CODE: SUBFUNCTIONS

% Section 14
function m = local_setproperties( m )
% This function is called at time zero in the INITIALISATION section of the
% interaction function. It provides commands to set each of the properties
% that are contained in m.globalProps. Uncomment whichever ones you would
% like to set yourself, and put in whatever value you want.
%
% Some of these properties are for internal use only and should never be
% set by the user. At some point these will be moved into a different
% component of m, but for the present, just don't change anything unless
% you know what it is you're changing.

% m = leaf_setproperty( m, 'trinodesvalid', true );
% m = leaf_setproperty( m, 'prismnodesvalid', true );
% m = leaf_setproperty( m, 'thicknessRelative', 0.020000 );
% m = leaf_setproperty( m, 'thicknessArea', 0.000000 );
% m = leaf_setproperty( m, 'thicknessMode', 'physical' );
% m = leaf_setproperty( m, 'activeGrowth', 1.000000 );
% m = leaf_setproperty( m, 'displayedGrowth', 1.000000 );
% m = leaf_setproperty( m, 'displayedMulti', [] );
% m = leaf_setproperty( m, 'allowNegativeGrowth', true );
% m = leaf_setproperty( m, 'usePrevDispAsEstimate', true );
% m = leaf_setproperty( m, 'perturbInitGrowthEstimate', 0.000010 );
% m = leaf_setproperty( m, 'perturbRelGrowthEstimate', 0.010000 );
% m = leaf_setproperty( m, 'perturbDiffusionEstimate', 0.000100 );
% m = leaf_setproperty( m, 'resetRand', false );
% m = leaf_setproperty( m, 'mingradient', 0.000000 );
% m = leaf_setproperty( m, 'relativepolgrad', false );
% m = leaf_setproperty( m, 'usefrozengradient', true );
% m = leaf_setproperty( m, 'userpolarisation', false );
% m = leaf_setproperty( m, 'thresholdsq', 0.000841 );
% m = leaf_setproperty( m, 'splitmargin', 1.400000 );
% m = leaf_setproperty( m, 'splitmorphogen', );
% m = leaf_setproperty( m, 'thresholdmgen', 0.500000 );
% m = leaf_setproperty( m, 'bulkmodulus', 1.000000 );
% m = leaf_setproperty( m, 'unitbulkmodulus', true );
% m = leaf_setproperty( m, 'poissonsRatio', 0.300000 );
% m = leaf_setproperty( m, 'starttime', 0.000000 );
% m = leaf_setproperty( m, 'timestep', 0.010000 );
% m = leaf_setproperty( m, 'timeunitname', );
% m = leaf_setproperty( m, 'distunitname', 'mm' );
% m = leaf_setproperty( m, 'scalebarvalue', 0.000000 );
% m = leaf_setproperty( m, 'validateMesh', true );
% m = leaf_setproperty( m, 'rectifyverticals', false );
% m = leaf_setproperty( m, 'allowSplitLongFEM', true );
% m = leaf_setproperty( m, 'longSplitThresholdPower', 0.000000 );
% m = leaf_setproperty( m, 'allowSplitBentFEM', false );
% m = leaf_setproperty( m, 'allowSplitBio', true );
% m = leaf_setproperty( m, 'allowFlipEdges', false );
% m = leaf_setproperty( m, 'allowElideEdges', true );
% m = leaf_setproperty( m, 'mincellangle', 0.200000 );
% m = leaf_setproperty( m, 'alwaysFlat', 0.000000 );
% m = leaf_setproperty( m, 'flattenforceconvex', true );
% m = leaf_setproperty( m, 'flatten', false );
% m = leaf_setproperty( m, 'flattenratio', 1.000000 );
% m = leaf_setproperty( m, 'useGrowthTensors', false );
% m = leaf_setproperty( m, 'plasticGrowth', false );
% m = leaf_setproperty( m, 'totalinternalrotation', 0.000000 );
% m = leaf_setproperty( m, 'stepinternalrotation', 2.000000 );
% m = leaf_setproperty( m, 'showinternalrotation', false );
% m = leaf_setproperty( m, 'performinternalrotation', false );
% m = leaf_setproperty( m, 'internallyrotated', false );
% m = leaf_setproperty( m, 'maxFEcells', 0 );
% m = leaf_setproperty( m, 'inittotalcells', 0 );
% m = leaf_setproperty( m, 'bioAprepsplitproc', );
% m = leaf_setproperty( m, 'bioApostsplitproc', );
% m = leaf_setproperty( m, 'maxBioAcells', 0 );
% m = leaf_setproperty( m, 'maxBioBcells', 0 );
% m = leaf_setproperty( m, 'colors', (6 values) );
% m = leaf_setproperty( m, 'colorvariation', 0.050000 );
% m = leaf_setproperty( m, 'colorparams', (12 values) );
% m = leaf_setproperty( m, 'freezing', 0.000000 );
% m = leaf_setproperty( m, 'canceledrift', false );
% m = leaf_setproperty( m, 'mgen_interaction', );
% m = leaf_setproperty( m, 'mgen_interactionName', 'gpt_tut_interaction_example_20110601' );
% m = leaf_setproperty( m, 'allowInteraction', true );
% m = leaf_setproperty( m, 'interactionValid', true );
% m = leaf_setproperty( m, 'gaussInfo', (unknown type struct) );
% m = leaf_setproperty( m, 'stitchDFs', [] );
% m = leaf_setproperty( m, 'D', (36 values) );
% m = leaf_setproperty( m, 'C', (36 values) );
% m = leaf_setproperty( m, 'G', (6 values) );
% m = leaf_setproperty( m, 'solver', 'cgs' );
% m = leaf_setproperty( m, 'solverprecision', 'double' );
% m = leaf_setproperty( m, 'solvertolerance', 0.001000 );
% m = leaf_setproperty( m, 'solvertolerancemethod', 'norm' );
% m = leaf_setproperty( m, 'diffusiointolerance', 0.000010 );
% m = leaf_setproperty( m, 'allowsparse', true );
% m = leaf_setproperty( m, 'maxIters', 0 );
% m = leaf_setproperty( m, 'maxsolvetime', 1000.000000 );
% m = leaf_setproperty( m, 'cgiters', 0 );
% m = leaf_setproperty( m, 'simsteps', 0 );
% m = leaf_setproperty( m, 'stepsperrender', 0 );
% m = leaf_setproperty( m, 'growthEnabled', true );
% m = leaf_setproperty( m, 'diffusionEnabled', true );

```

```

% m = leaf_setproperty( m, 'flashmovie', false );
% m = leaf_setproperty( m, 'makemovie', false );
% m = leaf_setproperty( m, 'moviefile', );
% m = leaf_setproperty( m, 'codec', 'None' );
% m = leaf_setproperty( m, 'autonamemovie', true );
% m = leaf_setproperty( m, 'overwritemovie', false );
% m = leaf_setproperty( m, 'framesize', [] );
% m = leaf_setproperty( m, 'mov', [] );
% m = leaf_setproperty( m, 'jiggleProportion', 1.000000 );
% m = leaf_setproperty( m, 'cvtperiter', 0.200000 );
% m = leaf_setproperty( m, 'boingNeeded', false );
% m = leaf_setproperty( m, 'initialArea', 0.031326 );
% m = leaf_setproperty( m, 'bendunitlength', 0.176992 );
% m = leaf_setproperty( m, 'targetRelArea', 1.000000 );
% m = leaf_setproperty( m, 'defaultinterp', 'min' );
% m = leaf_setproperty( m, 'readonly', false );
% m = leaf_setproperty( m, 'projectdir', 'D:\ab\Matlab stuff' );
% m = leaf_setproperty( m, 'modelname', 'GPT_tut_interaction_example_20110601' );
% m = leaf_setproperty( m, 'allowsave', true );
% m = leaf_setproperty( m, 'addedToPath', false );
% m = leaf_setproperty( m, 'bendsplit', 0.300000 );
% m = leaf_setproperty( m, 'usepolifreezebc', false );
% m = leaf_setproperty( m, 'dorsaltop', true );
% m = leaf_setproperty( m, 'defaultazimuth', -45.000000 );
% m = leaf_setproperty( m, 'defaulttelevation', 33.750000 );
% m = leaf_setproperty( m, 'defaultroll', 0.000000 );
% m = leaf_setproperty( m, 'defaultViewParams', (unknown type struct) );
% m = leaf_setproperty( m, 'comment', );
% m = leaf_setproperty( m, 'legendTemplate', '%T: %q\n%m' );
% m = leaf_setproperty( m, 'bioAsplitcells', true );
% m = leaf_setproperty( m, 'bioApullin', 0.142857 );
% m = leaf_setproperty( m, 'bioAfakepull', 0.202073 );
% m = leaf_setproperty( m, 'interactive', false );
% m = leaf_setproperty( m, 'coderevision', 0 );
% m = leaf_setproperty( m, 'coderevisiondate', );
% m = leaf_setproperty( m, 'modelrevision', 0 );
% m = leaf_setproperty( m, 'modelrevisiondate', );
% m = leaf_setproperty( m, 'savedrunname', );
% m = leaf_setproperty( m, 'savedrundesc', );
% m = leaf_setproperty( m, 'vxgrad', (108 values) );
% m = leaf_setproperty( m, 'lengthscale', 0.200000 );
end

```

% Section 15

```

% Here you may write any functions of your own, that you want to call from
% the interaction function, but never need to call from outside it.
% Remember that they do not have access to any variables except those
% that you pass as parameters, and cannot change anything except by
% returning new values as results.
% Whichever section they are called from, they must respect the same
% restrictions on what modifications they are allowed to make to the mesh.

```

```

% For example:

```

% Section 16

```

% function m = do_something( m )
% % Change m in some way.
% end

```

```

% Call it from the main body of the interaction function like this:
% m = do_something( m );

```

modified on 2 June 2011 at 20:11 *** 262 views

Tutorial on different ways of specifying the growth of shapes

[Back to GFTbox Tutorial pages](#)

We illustrate the practical advantage of having submodels within a project and an important consequence of understanding biological growth within the GPT-framework.

Conclusion: using a **combination** of polarity patterns to set local axes for anisotropic growth and patterns of differential specified growth to regulate the growth of shape would be powerful.

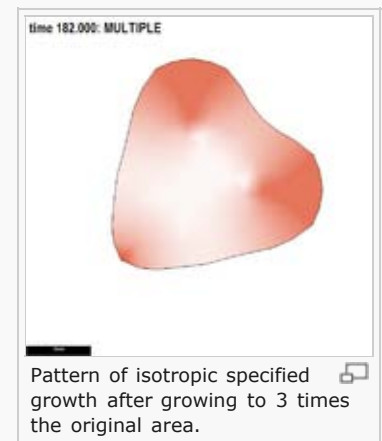
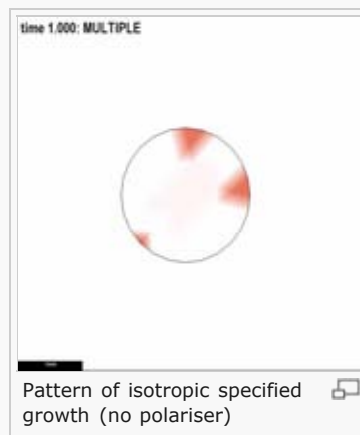
Illustrating independent ways to form shapes and the use of submodels.

The full interaction function is shown at the bottom. The line of code that selects the submodel and the start of each submodel is shown in red.

Uniform specified polariser (no polariser gradient). Creating a shape using a specified pattern of isotropic growth.

Result: simple patterns tend to produce blobby shapes.

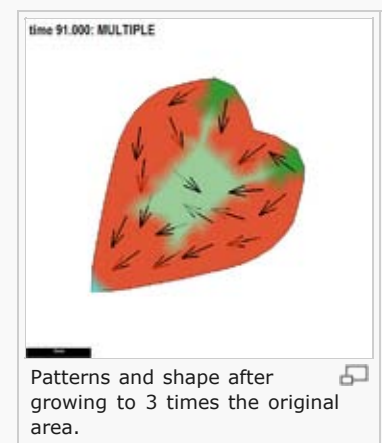
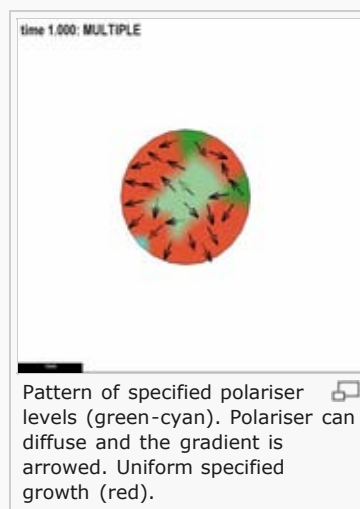
Conclusion: Whilst it may be possible to produce complex shapes such as the outgrowth shown in the GFTbox icon, we don't know how.



Uniform specified growth. Creating a shape using a specified pattern of diffusible polariser.

Result: simple patterns can readily produce sharp shapes.

Conclusion: It is not so easy to produce blobby shapes using patterns of polariser alone.



```

% Section 1
function m = gpt_twowayheart_20110531( m )
% m = gpt_twowayheart_20110531( m )
% Morphogen interaction function.
% Written at 2011-05-31 19:51:32.
% GFTbox revision 3548, 2011-05-31 14:37:10.747930.

% The user may edit any part of this function between delimiters
% of the form "USER CODE..." and "END OF USER CODE...". The

```

```

% delimiters themselves must not be moved, edited, deleted, or added.

if isempty(m), return; end

fprintf( 1, '%s found in %s\n', mfilename(), which(mfilename()) );

try
    m = local_setproperties( m );
catch
end

realtime = m.globalDynamicProps.currenttime;

% Section 2
%%% USER CODE: INITIALISATION

% In this section you may modify the mesh in any way whatsoever.
if (Steps(m)==0) && m.globalDynamicProps.doinit % First iteration
    % Set up names for variant models. Useful for running multiple models on a cluster.
    m.userdata.ranges.modelname.range = { 'PolariserBased', 'DifferentialGrowthBased' }; % CLUSTER
    m.userdata.ranges.modelname.index = 1; % CLUSTER
end
modelname = m.userdata.ranges.modelname.range{m.userdata.ranges.modelname.index}; % CLUSTER
disp(sprintf('\nRunning %s model %s\n',mfilename, modelname));

% Set priorities for simultaneous plotting of multiple morphogens, if desired.
m = leaf_mgen_plotpriority( m, {'ID_PLUSORG', 'ID_MINUSORG'}, [1,2], [0.4,0.4] );

% Set colour of polariser gradient arrows.
m = leaf_plotoptions(m,'highgradcolor',[0,0,0],'lowgradcolor',[1,0,0]);
m = leaf_plotoptions(m,'decorscale',1.5);

% setup a multiplot of the following morphogens
m = leaf_plotoptions( m, 'morphogen', {'V_KAREAL','ID_PLUSORG','ID_MINUSORG'});
%%% END OF USER CODE: INITIALISATION

% Section 3
%%% SECTION 1: ACCESSING MORPHOGENS AND TIME.
%%% AUTOMATICALLY GENERATED CODE: DO NOT EDIT.

if isempty(m), return; end

setGlobals();
global gNEW_KA_PAR gNEW_KA_PER gNEW_KB_PAR gNEW_KB_PER
global gNEW_K_NOR gNEW_POLARISER gNEW_STRAINRET gNEW_ARREST
dt = m.globalProps.timestep;
polariser_i = gNEW_POLARISER;
P = m.morphogens(:,polariser_i);
[kapar_i,kapar_p,kapar_a,kapar_l] = getMgenLevels( m, 'KAPAR' );
[kaper_i,kaper_p,kaper_a,kaper_l] = getMgenLevels( m, 'KAPER' );
[kbpar_i,kbpar_p,kbpar_a,kbpar_l] = getMgenLevels( m, 'KBPAR' );
[kbper_i,kbper_p,kbper_a,kbper_l] = getMgenLevels( m, 'KBPER' );
[knor_i,knor_p,knor_a,knor_l] = getMgenLevels( m, 'KNOR' );
[strainret_i,strainret_p,strainret_a,strainret_l] = getMgenLevels( m, 'STRAINRET' );
[arrest_i,arrest_p,arrest_a,arrest_l] = getMgenLevels( m, 'ARREST' );
[id_plusorg_i,id_plusorg_p,id_plusorg_a,id_plusorg_l] = getMgenLevels( m, 'ID_PLUSORG' );
[id_minusorg_i,id_minusorg_p,id_minusorg_a,id_minusorg_l] = getMgenLevels( m, 'ID_MINUSORG' );
[v_kareal_i,v_kareal_p,v_kareal_a,v_kareal_l] = getMgenLevels( m, 'V_KAREAL' );
[id_tip_i,id_tip_p,id_tip_a,id_tip_l] = getMgenLevels( m, 'ID_TIP' );
[id_top_i,id_top_p,id_top_a,id_top_l] = getMgenLevels( m, 'ID_TOP' );
[s_growth_i,s_growth_p,s_growth_a,s_growth_l] = getMgenLevels( m, 'S_GROWTH' );
[id_mid_i,id_mid_p,id_mid_a,id_mid_l] = getMgenLevels( m, 'ID_MID' );

% Mesh type: circle
% centre: 0
% circumpts: 24
% coneangle: 0
% dealign: 0
% height: 0
% innerpts: 0
% randomness: 0.1
% rings: 4
% version: 1
% xwidth: 2
% ywidth: 2

%
% Morphogen Diffusion Decay Dilution Mutant
%-----
% KAPAR ----
% KAPER ----
% KBPAR ----
% KBPER ----
% KNOR ----
% POLARISER 0.1 ----
% STRAINRET ----
% ARREST ----
% ID_PLUSORG ----
% ID_MINUSORG ----
% V_KAREAL ----
% ID_TIP ----
% ID_TOP ----
% S_GROWTH 0.01 ----
% ID_MID ----

%%% USER CODE: MORPHOGEN INTERACTIONS

% In this section you may modify the mesh in any way that does not
% Section 4
% alter the set of nodes.

% Use the same pattern for both submodels
RangeTip=(m.nodes(:,1)<-0.8)&...
(abs(m.nodes(:,2))<0.2);
RangeMid=(m.nodes(:,1)<=0.5)&...
(m.nodes(:,1)>-0.5)&...
(abs(m.nodes(:,2))<0.3);
RangeTops=(m.nodes(:,1)<=max(m.nodes(:,1))&...

```

```

(m.nodes(:,1)>0.5)&...
(abs(m.nodes(:,2))>0.3));
if (Steps(m)==0) && m.globalDynamicProps.doinit % Initialisation code.
switch modelname
case 'PolariserBased' %
% One way to set up a morphogen gradient is by ...
% Setting up a gradient by clamping the ends (execute only once)
P(RangeTip)=0;
P(RangeMid)=0.5;
P(RangeTops)=1;
id_plusorg_p=P;
id_minusorg_p(RangeTip)=1;
m.morphogenclamp( RangeTops|RangeTip|RangeMid, polariser_i ) = 1;
m = leaf_mgen_conductivity( m, 'POLARISER', 0.1 ); %specifies the diffusion rate of polariser
m = leaf_mgen_absorption( m, 'POLARISER', 0.0 ); % specifies degradation rate of polariser
case 'DifferentialGrowthBased' %
P(:)=0;
% One way to set up a morphogen gradient is by ...
% Setting up a gradient by clamping the ends (execute only once)
s_growth_p(RangeTip)=1;
s_growth_p(RangeMid)=0.05;
s_growth_p(RangeTops)=0.8;
m.morphogenclamp( RangeTops|RangeTip|RangeMid, s_growth_i ) = 1;
m = leaf_mgen_conductivity( m, 's_growth', 0.001 ); %specifies the diffusion rate of polariser
m = leaf_mgen_absorption( m, 's_growth', 0.0 ); % specifies degradation rate of polariser
end
end
BasicGrowth=0.01;
switch modelname
case 'PolariserBased' %
% Every equation to be formatted should end with an at-at Eqn N comment.
kapar_p(:) = BasicGrowth; % when isotropic this will be 0.005
kaper_p(:) = 0.0; % when isotropic this will be 0.005
kbpap_p(:) = BasicGrowth; % when isotropic this will be 0.005
kbper_p(:) = 0.0; % when isotropic this will be 0.005
knor_p(:) = 0; % thickness
case 'DifferentialGrowthBased' %
% Every equation to be formatted should end with an at-at Eqn N comment.
kapar_p(:) = BasicGrowth*s_growth_p; %0.01; % when isotropic this will be 0.005
kaper_p(:) = BasicGrowth*s_growth_p; %0.0; % when isotropic this will be 0.005
kbpap_p(:) = BasicGrowth*s_growth_p; %0.01; % when isotropic this will be 0.005
kbper_p(:) = BasicGrowth*s_growth_p; %0.0; % when isotropic this will be 0.005
knor_p(:) = 0; % thickness
end
v_kareal_p=kapar_p+kaper_p; % total specified areal growth
% Section 5
%%% END OF USER CODE: MORPHOGEN INTERACTIONS

%%% SECTION 3: INSTALLING MODIFIED VALUES BACK INTO MESH STRUCTURE
%%% AUTOMATICALLY GENERATED CODE: DO NOT EDIT.
m.morphogens(:,polariser_i) = P;
m.morphogens(:,kapar_i) = kapap_p;
m.morphogens(:,kaper_i) = kaper_p;
m.morphogens(:,kbpap_i) = kbpap_p;
m.morphogens(:,kbper_i) = kbper_p;
m.morphogens(:,knor_i) = knor_p;
m.morphogens(:,strainret_i) = strainret_p;
m.morphogens(:,arrest_i) = arrest_p;
m.morphogens(:,id_plusorg_i) = id_plusorg_p;
m.morphogens(:,id_minusorg_i) = id_minusorg_p;
m.morphogens(:,v_kareal_i) = v_kareal_p;
m.morphogens(:,id_tip_i) = id_tip_p;
m.morphogens(:,id_top_i) = id_top_p;
m.morphogens(:,s_growth_i) = s_growth_p;
m.morphogens(:,id_mid_i) = id_mid_p;

%%% USER CODE: FINALISATION
%%% END OF USER CODE: FINALISATION

end

%%% USER CODE: SUBFUNCTIONS

% Section 6
function m = local_setproperties( m )
end

```

modified on 1 July 2011 at 09:08 *** 114 views

Tutorial on retaining residual strain and cutting

[Back to tutorial pages](#)

Note

Crimson: code relating to submodel selection

MediumBlue: code to tell plotter to put morphogens on their respective A and B sides

RosyBrown: code for setting up the seam and cutting the mesh

When the system is arrested (ARRESTTIME=50) there is no further growth and the seams are cut. (The dummy morphogen `f_seams` is used to specify the nodes that will be divided into two, unconnected nodes - by declaring the morphogen in the GUI it is easy to check that the cuts will be in the right place. There is a plot option to allow the seams to be seen.)

```

% Section 1
function m = gpt_retainstrainandcut_20110603( m )
% m = gpt_retainstrainandcut_20110603( m )
% Morphogen interaction function.
% Written at 2011-06-03 14:38:33.
% GFTbox revision 3554, 2011-06-02 12:30:52.789869.

% The user may edit any part of this function between delimiters
% of the form "USER CODE..." and "END OF USER CODE...". The
% delimiters themselves must not be moved, edited, deleted, or added.

if isempty(m), return; end

fprintf( 1, '%s found in %s\n', mfilename(), which(mfilename()) );

try
    m = local_setproperties( m );
catch
end

realtime = m.globalDynamicProps.currenttime;

% Section 2
%%% USER CODE: INITIALISATION

% In this section you may modify the mesh in any way whatsoever.
if Steps(m)==0 % First iteration
    m.userdata.ranges.modelname.range = { 'NORESIDUALS', 'FULLRESIDUALS' };
    m.userdata.ranges.modelname.index = 1;
end
modelname = m.userdata.ranges.modelname.range{m.userdata.ranges.modelname.index};
switch modelname
    case 'NORESIDUALS'
        % Set up the parameters (e.g. mutations) for this model here.
        StrainRetention=0;
    case 'FULLRESIDUALS'
        % Set up the parameters (e.g. mutations) for this model here.
        StrainRetention=1;
    otherwise
        % If you reach here, you probably forgot a case.
end
% override the GUI and plot the two growth rates on the different sides
% this line must be commented out if you want to watch the residual
% strain build up and decay (Plot output value, Residual growth, Areal)
m = leaf_plotoptions( m, 'morphogenA', 'KAPAR','morphogenB', 'KBPAR' );
% There is a bug in the code that colours in the mesh and colourbar
% The work around is to click the Monochrome tick box off and on again
% likewise the Auto color range
% For this example it is best to turn off the auto color range

%%% END OF USER CODE: INITIALISATION

% Section 3
%%% SECTION 1: ACCESSING MORPHOGENS AND TIME.
%%% AUTOMATICALLY GENERATED CODE: DO NOT EDIT.

if isempty(m), return; end

setGlobals();
global gNEW_KA_PAR gNEW_KA_PER gNEW_KB_PAR gNEW_KB_PER
global gNEW_K_NOR gNEW_POLARISER gNEW_STRAINRET gNEW_ARREST
dt = m.globalProps.timestep;
polariser_i = gNEW_POLARISER;
P = m.morphogens(:,polariser_i);
[kapar_i,kapar_p,kapar_a,kapar_l] = getMgenLevels( m, 'KAPAR' );
[kaper_i,kaper_p,kaper_a,kaper_l] = getMgenLevels( m, 'KAPER' );
[kbpar_i,kbpar_p,kbpar_a,kbpar_l] = getMgenLevels( m, 'KBPAR' );
[kbper_i,kbper_p,kbper_a,kbper_l] = getMgenLevels( m, 'KBPER' );
[knor_i,knor_p,knor_a,knor_l] = getMgenLevels( m, 'KNOR' );
[strainret_i,strainret_p,strainret_a,strainret_l] = getMgenLevels( m, 'STRAINRET' );
[arrest_i,arrest_p,arrest_a,arrest_l] = getMgenLevels( m, 'ARREST' );
[id_sidelines_i,id_sidelines_p,id_sidelines_a,id_sidelines_l] = getMgenLevels( m, 'ID_SIDELINES' );

```

```

[id_topcap_i,id_topcap_p,id_topcap_a,id_topcap_l] = getMgenLevels( m, 'ID_TOPCAP' );
[id_botcap_l,id_botcap_p,id_botcap_a,id_botcap_l] = getMgenLevels( m, 'ID_BOTCAP' );
[f_seam_i,f_seam_p,f_seam_a,f_seam_l] = getMgenLevels( m, 'F_SEAM' );

% Mesh type: capsule
% basecap: 1
% baseheight: 1
% baserings: 5
% circumdivs: 12
% height: 0.2
% heightdivs: 16
% randomness: 0.1
% topcap: 1
% topheight: 1
% toprings: 5
% version: 1
% xwidth: 0.2
% ywidth: 0.2

%
% Morphogen Diffusion Decay Dilution Mutant
% -----
% KAPAR ----
% KAPER ----
% KBPAR ----
% KBPER ----
% KNOR ----
% POLARISER 0.002 ----
% STRAINRET ----
% ARREST ----
% ID_SIDE LINES ----
% ID_TOPCAP ----
% ID_BOTCAP ----
% F_SEAM ----

%%% USER CODE: MORPHOGEN INTERACTIONS

% In this section you may modify the mesh in any way that does not
% Section 4
% alter the set of nodes.
if Steps(m)==0 % Initialisation code.
% Put any code here that should only be performed at the start of
% the simulation, for example, to set up initial morphogen values.
% Force global variables to zeros to make debugging more reliable
m.morphogens=zeros(size(m.morphogens));
m.seams=logical(zeros(size(m.seams)));
m.polfreeze=zeros(size(m.polfreeze));
m.mgen_production=zeros(size(m.mgen_production));

% Caps
id_topcap_p(m.nodes(:,3)>0.17)=1;
id_botcap_p(m.nodes(:,3)<-0.17)=1;
% Sidelines
indx=find(abs(m.nodes(:,1))>0.08);
indy=find(abs(m.nodes(:,2))<0.02);
id_sidelines_p(intersect(indx,indy))=1;
% Seams along which the capsule will be cut (not the bottom cap)
inds=[find(id_sidelines_p>0.5);find(id_topcap_p>0.5)];
f_seam_p(inds)=1;
m=leaf_set_seams(m,f_seam_p);
% Seams will be cut when this flag is set to 1 - see see FINALISATION code
m.userdata.CutOpen=0;
end

% Section 5
% Code common to all models. (the different strain retention is setup
% in the initial routines above)
strainret_p=StrainRetention.*ones(size(strainret_p));
ARRESTTIME=50;
if m.userdata.CutOpen==0 % Grow the silique
% Section 6
% Code for specific models.
% @@PRN Polariser Regulatory Network
% Every equation to be formatted should end with an at-at Eqn N comment.
m = leaf_mgen_conductivity( m, 'Polariser', 0.002);% diffusion constant
m = leaf_mgen_dilution( m, 'Polariser', false );% it will not dilute with growth
m = leaf_mgen_absorption( m, 'Polariser', 0); % it will not decay everywhere
P(id_topcap_l>0.5)=1;
m.morphogenclamp(id_topcap_l>0.5,polariser_i) = 1;
P(id_botcap_l>0.5)=0;
m.morphogenclamp(id_botcap_l>0.5,polariser_i) = 1;
% having clamped the values at each end and allowed POL to diffuse
% diffusion will occur automatically and produce a linear gradient
kpar_p=0.01; % Specified growth on the inside
kbp_p=0.001; % Much less specified growth on the outside
kaper_p=0.2*kpar_p; % anisotropic growth (grows lengthwise)
kbper_p=0.2*kbp_p;
if realtime>ARRESTTIME
m.userdata.CutOpen=1;
end
elseif m.userdata.CutOpen==1
else % relax the cut silique
kpar_p=0;
kbp_p=0;
kaper_p=0;
kbper_p=0;
knor_p =0;
m = leaf_setproperty( m, 'freezing', 0.9 );
end
% Section 7
%%% END OF USER CODE: MORPHOGEN INTERACTIONS

%%% SECTION 3: INSTALLING MODIFIED VALUES BACK INTO MESH STRUCTURE
%%% AUTOMATICALLY GENERATED CODE: DO NOT EDIT.
m.morphogens(:,polariser_i) = P;
m.morphogens(:,kpar_i) = kpar_p;
m.morphogens(:,kaper_i) = kaper_p;
m.morphogens(:,kbp_i) = kbper_p;
m.morphogens(:,kbp_i) = kbper_p;

```

```
m.morphogens(:,kbper_i) = kbper_p;
m.morphogens(:,knor_i) = knor_p;
m.morphogens(:,strainret_i) = strainret_p;
m.morphogens(:,arrest_i) = arrest_p;
m.morphogens(:,id_sidelines_i) = id_sidelines_p;
m.morphogens(:,id_topcap_i) = id_topcap_p;
m.morphogens(:,id_botcap_i) = id_botcap_p;
m.morphogens(:,f_seam_i) = f_seam_p;
```

```
%%% USER CODE: FINALISATION
```

```
% In this section you may modify the mesh in any way whatsoever.
```

```
if m.userdata.CutOpen==1
    m=leaf_dissect(m);
    m.userdata.CutOpen=2;
    m = leaf_setproperty( m, 'freezing', 0.999 );
end
```

```
%%% END OF USER CODE: FINALISATION
```

```
end
```

```
%%% USER CODE: SUBFUNCTIONS
```

```
% Here you may add any functions of your own, that you want to call from
% the interaction function, but never need to call from outside it.
% Whichever section they are called from, they must respect the same
% restrictions on what modifications they are allowed to make to the mesh.
% This comment can be deleted.
```

modified on 3 June 2011 at 16:08 *** 144 views

University of East Anglia / School of Computing Science / Powered by MediaWiki

Running example models and using a cluster

[Back to GFtbox Tutorial pages](#)

The purpose of these tutorials is to learn how to run the example growth simulations included in GFtbox. We will describe methods for running simulations locally (on your own computer) and remotely (on a computing cluster). It is assumed that you have already downloaded the GFtbox software and have Matlab installed.

Contents [\[hide\]](#)

- [1 Getting Started](#)
- [2 1 Explaining the tools](#)
 - [2.1 GFtboxCommand](#)
 - [2.2 ClusterMonitor](#)
- [3 2 Computer or cluster?](#)
- [4 3 Running a growth simulation non-interactively on your local computer](#)
 - [4.1 3A - GFtboxCommand: Running a simulation on your computer](#)
 - [4.2 3B - VMSreport: getting results from a completed simulation](#)
- [5 4 Altering the simulation parameters](#)
- [6 5 GFtboxCommand: Altering the model parameters on the cluster](#)
- [7 Notes on using Unix \(Linux\) computing clusters](#)

Getting Started

The remainder of this page is split into five sub-tutorials, each building on the preceding parts.

- 1) **Explaining the tools.** In these tutorials, we will be using *GFtboxCommand* and *ClusterMonitor*. This section explains the purpose of these tools.
- 2) **Your local computer or cluster?** Here, we illustrate when and why you should use a computing cluster for your growth simulations, and conversely, when you should use your desktop computer.
- 3) **Running a growth simulation non-interactively on your local computer.** This section demonstrates how to use *GFtboxCommand* to run a growth simulation on your computer. The model used for the simulation is an example model included with the GFtbox.
- 4) **Altering the simulation parameters.** Following on from part 3, here we show how to adjust a simulation parameter within *GFtboxCommand*. Specifically, we alter the value of *dt*, the temporal resolution of the simulation, and show how it can be used to verify that the value specified in the published literature is reasonable.
- 5) **Running on a cluster.** Altering the model parameters on the cluster. Finally, we demonstrate how a number of model parameters can be varied by specifying a range of values for each model argument. We show how the computationally expensive task of simulating all combinations of specified ranges can be processed more efficiently if a computer cluster is used via the *ClusterMonitor* tool.

1 Explaining the tools

GFtboxCommand

This is a command line version of the GFtbox. By command line, we mean that all program **functions are operated via typed commands, without the GUI**. Like GFtbox, GFtboxCommand is capable of running growth simulations of an interaction function, and allows the user to specify model and simulation parameters. Unlike GFtbox, this also allows the user to select **ranges of values for a number of input parameters**, and will automatically spawn multiple simulations which explore the various combinations of those parameters. This can be used to evaluate the effect of various parameters on the growth of a given model.

ClusterMonitor

Provides a graphical user interface for managing simulations **running remotely on a computer cluster**. Specifically, it allows you to see which projects are present and running on the cluster, to retrieve the completed projects, to generate images of the simulations at specified stages of growth, and to remove projects from the cluster. If you do not intend to use a computer cluster, then you will not need to use ClusterMonitor.

2 Computer or cluster?

In basic terms, a **computing cluster is effectively a network of many computer processors** (often hundreds), centrally managed by a

queuing system. When a job is submitted to a cluster, the job is sent to a processor that is not being used, or queued until one becomes available. In contrast, a typical desktop computer will contain one processor, limiting the number of tasks that can be performed at any one time. Jobs which must be run independently and sequentially on a desktop computer can be executed in parallel on a cluster, **greatly reducing the total time required to complete all of the jobs**. Although the exact details of your cluster might vary from those described in the rest of these tutorials, we aim to illustrate the generic processes involved in using GFTbox on a cluster. We are happy to offer assistance, where possible, to setup GFTbox for your cluster, so please contact us if you have any queries.

Whilst the time savings offered by using a cluster can be significant, there is an overhead associated with returning the results to your personal computer. The total time to run 2 or more simulations on a cluster and to return the results will be less than running those simulations sequentially on one computer. Therefore, using a **cluster is ideal for situations where you would like to run several simulations**, such as to evaluate the effect of a range of parameters on a growth model. You are not advised to use a cluster for running single simulations, or where you would like to step through a simulation and change parameters in a more interactive fashion. In such circumstances, you are advised to run GFTbox or GFTboxCommand on your own computer.

The **GFTbox GUI provides quick feedback** about how the changes you have made to an interaction function have affected the growth simulation, as **you can see the result of each simulation iteration as it completes**. It is quick and easy therefore to see if you have dramatically changed the course of the growth simulation, and then to adjust the parameters according to your observations. This approach is well suited to the early stages of design, where you might wish to **tweak some parameters to gauge whether or not they have had the desired effect**. Or in the final stages, where you think your model is almost finalised, but where small adjustments are required. **An alternative approach is to start many simulations**, based upon your initial model and by making intelligent choices about which parameters to explore. This allows you to **harvest many results, to quickly and easily overview them** in their finalised form, or to select interesting-looking simulations and examine them more closely on your desktop computer, using GFTbox.

3 Running a growth simulation non-interactively on your local computer

This tutorial is aimed at running a **growth simulation for one of the example interaction functions** included with the GFTbox. The purpose of this exercise is to firstly demonstrate how simulations can be invoked using GFTboxCommand, and secondly to show how to reproduce experimental results (specifically, those published in Kennaway et al (2011)) given an interaction function.

Assuming Matlab is installed on your computer, and the latest GFTbox has been downloaded, you can add the GFTbox directory to Matlab's search path, which makes the toolbox accessible from any other path that you choose to work from. For a short tutorial on how to do this, please click [here](#).

Once the GFTbox is added to Matlab, you are ready to run a growth simulation using GFTboxCommand. In this example, **the model that we will simulate is called GPT_CASE_RST**. Results generated using this model are published in Kennaway et al (2011). By running this simulation, we can confirm the results in the published literature and investigate the suitability of the various parameters.

3A - GFTboxCommand: Running a simulation on your computer

The following command can be typed into Matlab to run a simulation of the GPT_CASE_RST interaction function, which contains three growth models: R, S and T. **Three separate simulations are run sequentially on your computer**, one for each model, each producing results corresponding to five intervals in the growth simulation.

```
GFTboxCommand('Path','/GrowthToolbox/Models/Published/Kennaway-et-al-2011/','Name','GPT_CASE_RST',...
'Stages',[20 100 140 180 200],'modelname',[1:3]);
```

GFTboxCommand accepts input arguments as name and value pairs, e.g. 'modelname', [1:3] or 'Use','Cluster'. The argument names entered in the example above are: *Path*, *Name*, *Stages* and *modelname*.

The optional **Path argument name refers to the location of the folder** (or directory) on your computer where the model interaction function you wish to simulate is stored. **Name, is the name of the folder itself**. In this case, we are using the GPT_CASE_RST folder which is included in the GrowthToolbox. You may wish to copy this folder elsewhere, if you intend to make changes to the interaction function.

During a growth simulation, a mesh can be generated at each time step of the simulation, which provides a visual representation of the growth of the biological tissue, given the various parameters of growth specified by the interaction function and how they have changed over time. Put another way, the mesh shows exactly what the growing tissue actually looks like. *Stages* refers to the points in the simulation (measured in hours) at which meshes should be generated and saved. In this example, **five stages of growth will be written to disk**. These values are chosen to best capture the appearance of the tissue at important stages of the tissue growth.

The final argument name listed here is *modelname*. This is a model-specific argument, and in this case the GPT_CASE_RST interaction function contains **three separate models for plant growth**, allowing the desired model to be selected. Here, the value [1:3] is specified, which is evaluated in the same way as entering [1 2 3]. This instructs GFTboxCommand to run three separate simulations, one for each of the growth models contained in the interaction function.

The function of every permissible argument is given by keying the following command into Matlab:

```
help GFTboxCommand
```

3B - VMSreport: getting results from a completed simulation

1) - Generating images from a simulation you ran on your computer

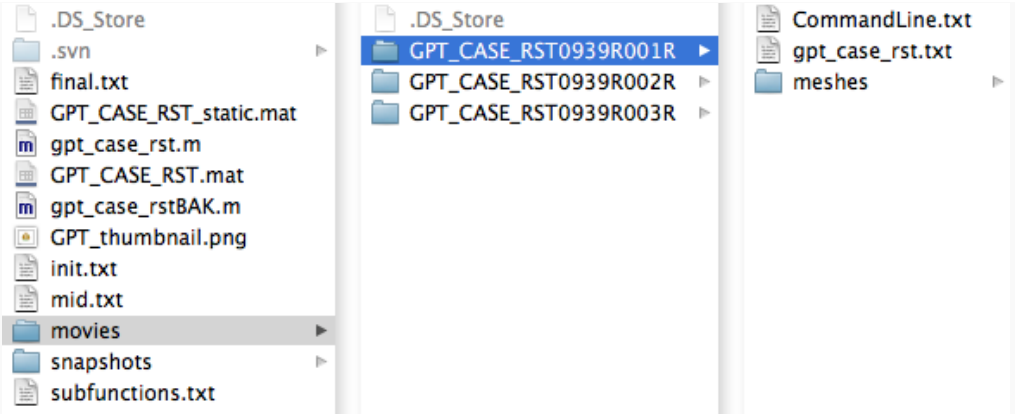
Once a growth simulation is completed, the project folder (GPT_CASE_RST, in our example) will contain another folder named "movies". Within movies, there are folders which contain results for the executed simulations. As was instructed in 3A, here we can see three folders corresponding to simulations for the three separate models. Within the first folder there are three items:

```
CommandLine.txt - This file
contains the Matlab command
which was used to generate the
```

results that this sub-directory contains.

gpt_case_rst.txt - This file, named according to the project name, contains a copy of the interaction function which was used to produce these results.

meshes - This folder contains the mesh files corresponding to the stages of growth specified by the value of the Stages argument, described in 3A.

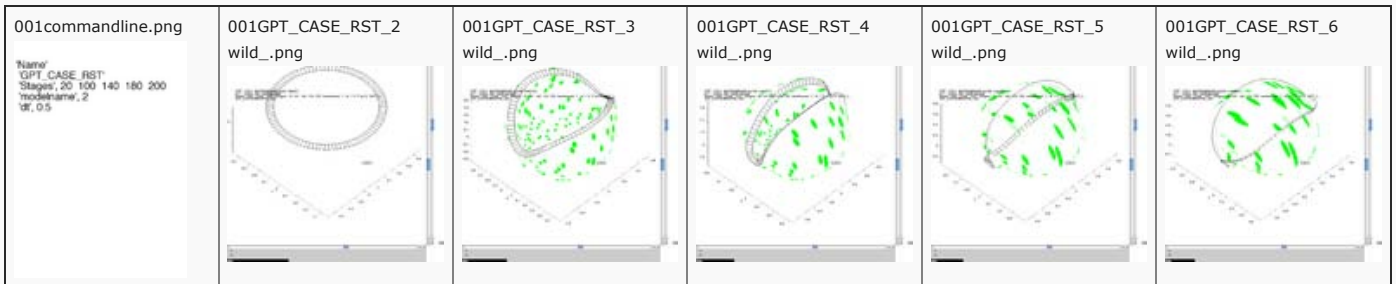


The mesh files contain vertices information regarding the shape of the growth model at a particular stage of growth, but are not visualisable in that form. In order to **convert these mesh files into viewable images**, we can execute the following command in Matlab:

```
VMSreport('Path','/GrowthToolbox/Models/Published/Kennaway-etal-2011/', 'Project','GPT_CASE_RST',...
'Experiment','All','flattentime',572.5,'morphogen','KPAR','SNAPFIG',true);
```

Where the *Path* and *Project* arguments have the same function as *Path* and *Name* in 3A, i.e. *Path* refers to the location of the project folder and *Project* is the name of the project folder itself. The images that are created will be stored in the *meshes* directory within the project folder.

Here we can see the generated images, including the command line arguments, for one of the simulations performed in 3A. Click on a thumbnail to view a larger image.



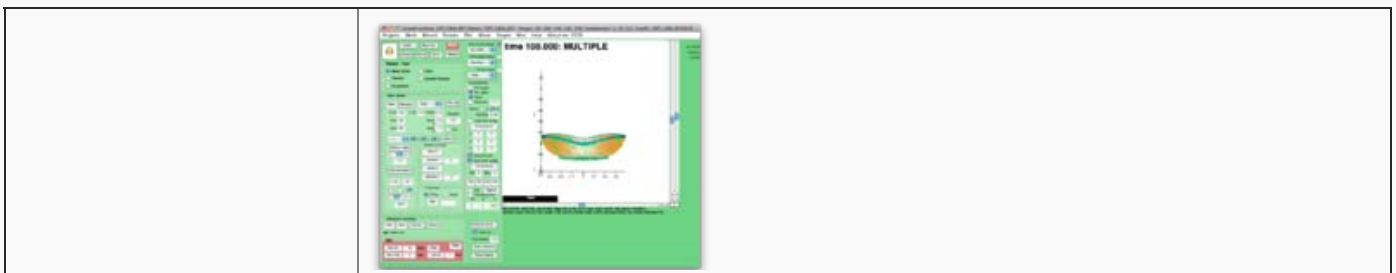
NB. The results produced may not be visibly identical to those in the published literature. This is because of small, random perturbations which are applied to the initial model meshes to stop them from containing surfaces which are perfectly flat, and therefore biologically unrealistic. The results produced should be **qualitatively but not quantitatively the same**.

2) - Generating images from a simulation you ran on the cluster

Another way to generate images from results is by using the ClusterMonitor tool. When a cluster is used to run simulations, the result files are stored remotely, on the cluster. These files must be retrieved, and then the 'Make project pngs' button can be pressed to generate images, which performs the same function as *VMSReport* in part 3A1. This process is described fully in part 5.

3) - Interacting with your results using the GFTbox GUI

Once result mesh files have been created, either on your local computer (using *GFTboxCommand*) or from retrieved files that were generated on a cluster, it is also possible to visualise the results in the GFTbox GUI.



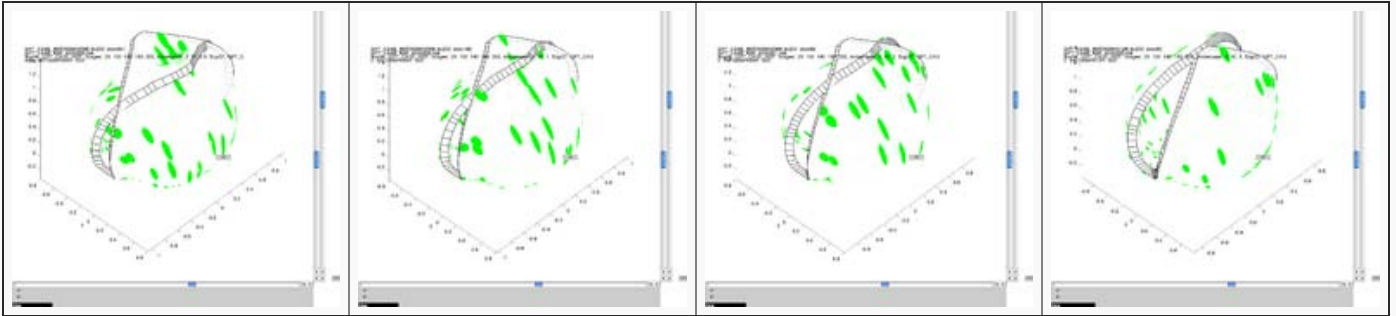
4 Altering the simulation parameters

One such simulation parameter is *dt*, which is the time in seconds between iterations in the growth simulation. Large values of *dt* mean that fewer steps and therefore fewer calculations are required to complete a simulation. Whereas smaller values of *dt* mean that more steps are required, and therefore more processing time too. Therefore, a value of *dt* must be selected which is not so small that it is computationally unmanageable, but not so large that the observed growth is an artifact of the value of *dt*, rather than the underlying growth model. It is necessary therefore to test a range of values for *dt* to ensure that the patterns of growth observed in a simulation are consistent across the range, and to find a value which is sufficient to demonstrate the model of growth and computationally efficient.

Testing a range of *dt*s can be achieved in several ways. One way is to use the *dt* argument when calling *GFTboxCommand*. This allows a single value to be tested. Another way is to make a batch of jobs, each using a different value for *dt*, by using the 'State' argument. Lastly, a range of *dt*s can be specified in *GFTboxCommand*, by using the *dt* argument where the values within the square brackets are the *dt*s to simulate:

```
GfTboxCommand('Path','/GrowthToolbox/Models/Published/Kennaway-et-al-2011/', 'Name','GPT_CASE_RST', ...
'Stages',[20 100 140 180 200], 'modelname',3, 'dt',[0.1 0.5 1 5]);
```

Here, we have specified **four separate values for dt** , and this means that **four separate simulations will be run** sequentially on your computer. As in 3B, images can then be generated for the mesh files produced, and compared to ensure consistency across the range of dt s. In this figure, we can see the same model, at the same stage of growth, generated using the four values of dt . Though quantitatively different, they are qualitatively the same, illustrating the suitability of the default dt value of 5.



5 GfTboxCommand: Altering the model parameters on the cluster

It is easy to see that when even a small number of range variables, mutations or dt s are specified, the **total number of simulations increases quickly**. If many ranges are specified, then the amount of processing time becomes unmanageable on a single computer. Via *GfTboxCommand* and the *ClusterMonitor* tool, we can remotely **run a number of simulations, in parallel, on a computing cluster**. It is assumed that the GrowthToolbox and Matlab are installed on your cluster, and PuTTY is required on your local computer for the tools *pscp* (for transferring files from your computer to a cluster) and *plink* (for remotely executing commands on a cluster). Using the dt range argument from 4 as an example, here we add the 'Use' name argument with the value 'Cluster':

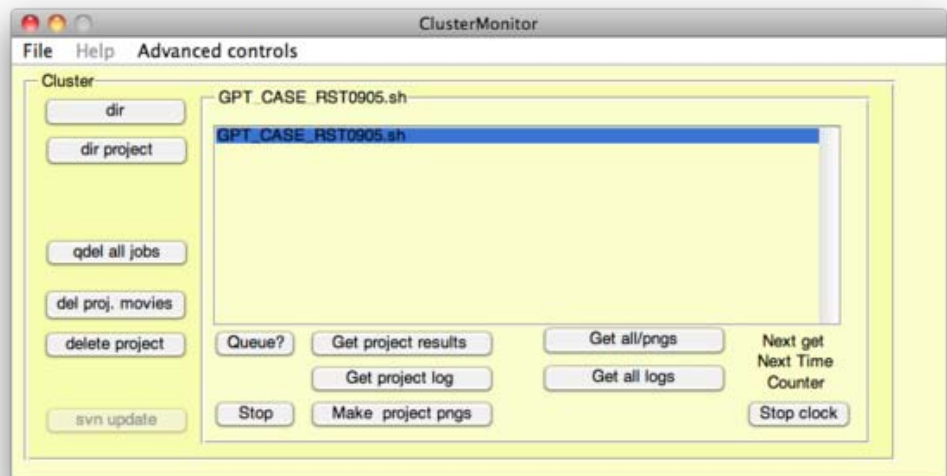
```
GfTboxCommand('State','Start','Path','/GrowthToolbox/Models/Published/Kennaway-et-al-2011/', 'Name','GPT_CASE_RST', ...
'Stages',[20 100 140 180 200], 'modelname',3, 'dt',[0.1 0.5 1 5], 'Use','Cluster');
```

The 'Cluster' option for the 'Use' name argument instructs *GfTboxCommand* to upload the required project directory to a remote Linux server. Instead of running the simulation on your own computer, *GfTboxCommand* then works out how many individual simulations are specified by the command that invoked it. In this example, the only argument which will generate multiple simulations is dt , of which there will be 4. Separate commands for each of these jobs are then automatically generated, each with an accompanying unique ID (The value of the *ExpID* name argument), and **these are submitted as individual jobs to the cluster**.

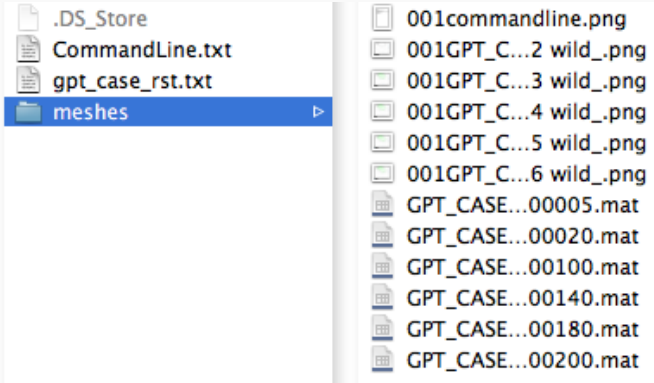
```
GfTboxCommand('Name','GPT_CASE_RST','Stages',[20 100 140 180 200], 'modelname',3, 'dt',0.1, 'ExpID','GPT_CASE_RST_1');
GfTboxCommand('Name','GPT_CASE_RST','Stages',[20 100 140 180 200], 'modelname',3, 'dt',0.5, 'ExpID','GPT_CASE_RST_2');
GfTboxCommand('Name','GPT_CASE_RST','Stages',[20 100 140 180 200], 'modelname',3, 'dt',1, 'ExpID','GPT_CASE_RST_3');
GfTboxCommand('Name','GPT_CASE_RST','Stages',[20 100 140 180 200], 'modelname',3, 'dt',5, 'ExpID','GPT_CASE_RST_4');
```

Once these jobs have been submitted, the *ClusterMonitor* tool opens and the new job batch ID is added to the list of jobs. Here, we will concentrate on three of the functions of *ClusterMonitor*: *Queue?*, *Get project results* and *Make project pngs*. As in part 4 of this page, these functions will enable us to visualise the growth simulation results.

This is an example of what you will see when the *ClusterMonitor* tool launches after submitting a job using the 'Use','Cluster' argument pair in *GfTboxCommand*.



Clicking the 'Queue?' button tells *ClusterMonitor* to check the status of your jobs on the cluster. It then prints the status of the current jobs in Matlab's

| | |
|--|--|
| <p>command window. In the case of our cluster, this is the equivalent of the <i>qstatme</i> command. This screenshot shows an example job list, where two jobs are currently running.</p> | <pre> Your Cluster job queue job-ID prior name user state submit/start at queue slots ja-task-ID ----- 2507197 0.50500 GPT_CASE_R w0454974 r 06/24/2011 10:17:17 short.q@node041.escluster.uea. 1 2507196 0.50500 GPT_CASE_R w0454974 r 06/24/2011 10:17:17 short.q@node096.escluster.uea. 1 OK >> </pre> |
| <p>Once your jobs have completed, the 'Queue?' button will show that there are no jobs running. Now you are ready to retrieve the project files from the cluster. This is achieved using the 'Get project results' button. Once pressed, the Matlab command window will show trace information to indicate which files are currently being downloaded. This is the equivalent of downloading the files manually, using an FTP program.</p> | <pre> Copying the files in GPT_CASE_RST0905R001R GPT_CASE_RST_s000005.mat GPT_CASE_RST_s000020.mat GPT_CASE_RST_s000100.mat GPT_CASE_RST_s000140.mat GPT_CASE_RST_s000180.mat GPT_CASE_RST_s000200.mat OK CommandLine.txt 0 kB 0.1 kB/s ETA: 00:00:00 100% gpt_case_rst.txt 16 kB 16.6 kB/s ETA: 00:00:00 100% has been copied from cluster OK copying file GPT_CASE_RST0905R001R GPT_CASE_RST_s000005.mat >> </pre> |
| <p>Once you have successfully downloaded your project results, you are ready to generate result images, as in part 3B1. Instead of entering a <i>VMSReport</i> command, you can press the 'Make project pngs' button, which will perform the same command. The images that are created will be in the <i>meshes</i> directory within the project folder.</p> |  |

Notes on using Unix (Linux) computing clusters

Whilst *ClusterMonitor* is a useful tool for managing your cluster jobs, it is advisable to **have at least a rudimentary understanding of and ability to use a Unix-based computer system**. In particular, the abilities to list the contents of a folder, view the contents of a file, change the current working folder, delete, and to copy or move files or folders, are essential Unix skills for making sure everything is working as intended. It is beyond the scope of these tutorials to provide an in depth Unix tutorial (many good and simple tutorials exist on the web), but here is a short description of the Unix commands (which may be specific to our Unix-based system) that we believe to be important.

- ls* - list the files and folders in a directory. Use the *-al* option (e.g. *ls -al*) to see modification dates, file permissions, sizes, etc.
- cd X* - Change the current directory to directory X.
- rm X* - Delete the file X.
- cp X NewPath/X2* - Copy file X to the NewPath directory and name the copy X2.
- mv X NewPath/X2* - Move file X to the NewPath directory and rename it to X2.
- cat test.txt* - Print the contents of the text file test.txt onto the screen.
- pwd* - Print the present working directory onto the screen.
- more test.txt* - Print the file test.txt to the screen, a screenful at a time, scrolling each time you hit the enter key.

modified on 26 October 2011 at 19:37 *** 2,861 views

VolViewer

[Back to BanghamLab software](#)

Contents [\[hide\]](#)

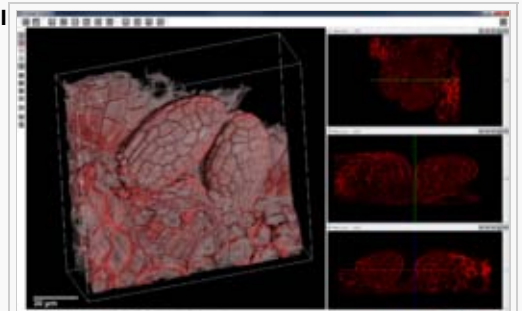
- [1 What? How? Where?](#)
- [2 User Documentation](#)
- [3 Sample Data](#)
- [4 Download](#)
 - [4.1 Windows Specific Notes](#)
- [5 Source Code](#)
 - [5.1 Building from source](#)
- [6 Image Gallery](#)
- [7 Media/Press](#)
- [8 Author](#)

What? How? Where?

What? VolViewer is used for **viewing volume images** from, for example, **confocal microscopy** or **optical projection tomography (OPT)**.

Features:

- ▶ Real-time volume rendering using an optimized 3D texture slicing algorithm.
- ▶ Interactive transfer functions to independently adjust opacity and intensity for up to three data channels.
- ▶ Real-time per channel thresholding, brightness and contrast operators.
- ▶ On-the-fly gradient computation for local illumination.
- ▶ Iso-surface computation with surface smoothing.
- ▶ Section viewing in any orientation / position.
- ▶ Real-time volume clipping.
- ▶ 3D measurements, filters & segmentation.
- ▶ Key frame interpolation for movie export.
- ▶ Stereo rendering using either quad buffer or anaglyph mode.
- ▶ Scripting interface to other systems, e.g. Matlab, OMERO, etc.



The VolViewer main application window.

How? It is open source and written in C++ using OpenGL, OpenCL and Qt.

Where? Binaries are available for the Windows, Mac OS X and Linux, see below.

Requirements: An OpenGL 2.1 / GLSL 1.20 compatible GPU with a recommended 512MB of memory.

User Documentation

[Quick Guide](#)
[TUTORIALS](#)
[Video Demos](#)
[SCRIPTING](#)

Sample Data



Antirrhinum



Arabidopsis



Arabidopsis Leaf (GL2:GUS expression in



Arabidopsis Leaf (Ath8::GUS expression in

Meristem

[Download](#)

Seedling

[Download](#)

red)

[Download](#)

red)

[Download](#)

* all data courtesy of Karen Lee [1]

Download

Although we try to keep up to date builds these sometimes lag behind the SVN trunk. So if you want the latest version / features, it is best to build the application from the trunk of the SVN. The build system is based on [qmake](#) for easy cross platform compilation.

[Windows \(32bit\)](#) [Windows \(64bit\)](#) [Linux](#) [MacOS X \(i386/x86 64/10.5+\)](#)

Windows Specific Notes

- ▶ You may need to install the corresponding Microsoft Visual C++ 2008 Redistributable Package which can be found here: [32bit](#) and [64bit](#).
- ▶ WindowsXP users will need to change the `view_gldrawbuffer = "GL_FRONT_AND_BACK"` to `view_gldrawbuffer = "GL_BACK"` in the settings.ini file.
- ▶ The binaries are built with OMERO 4.3.4 support.

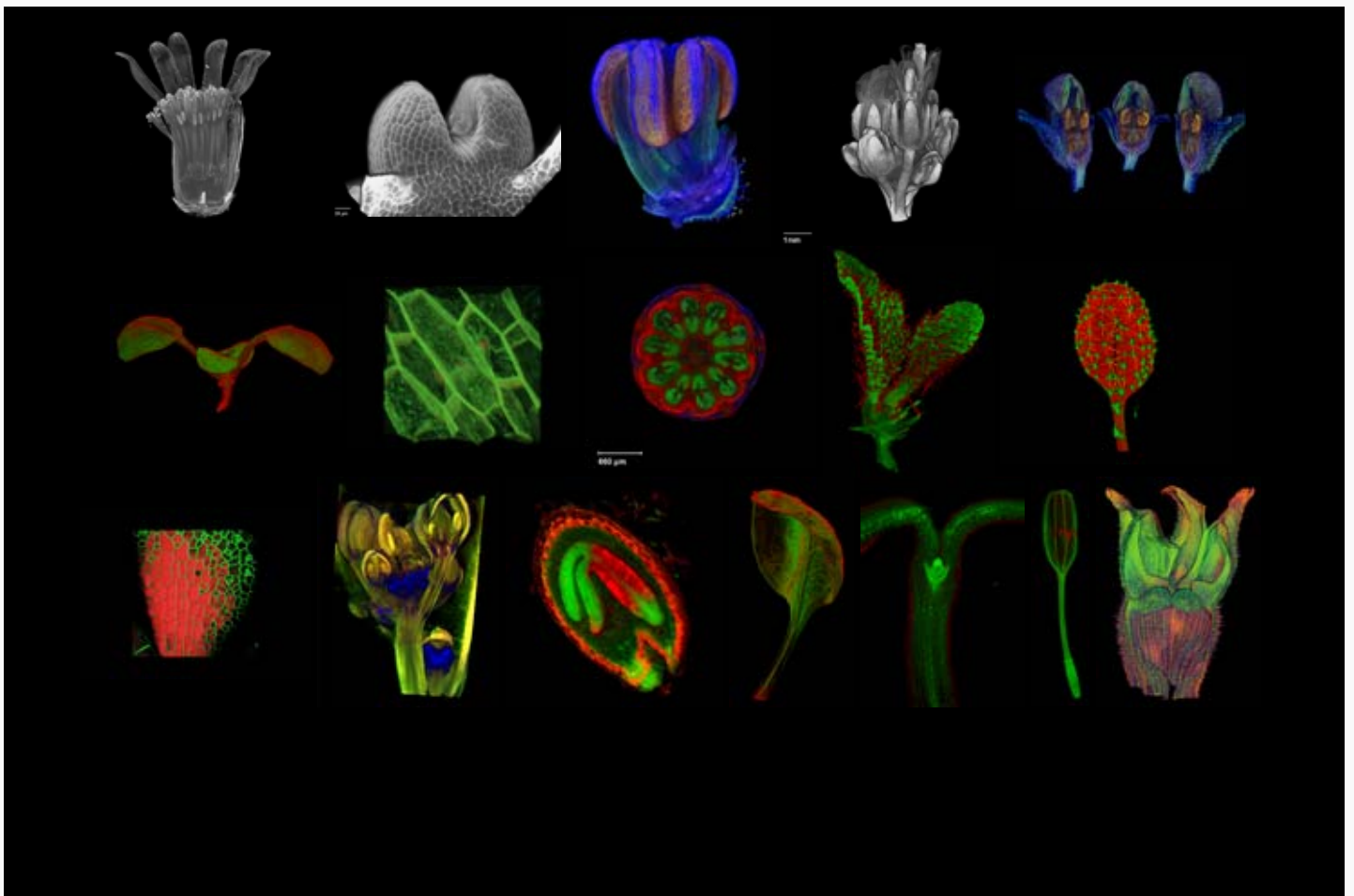
Source Code

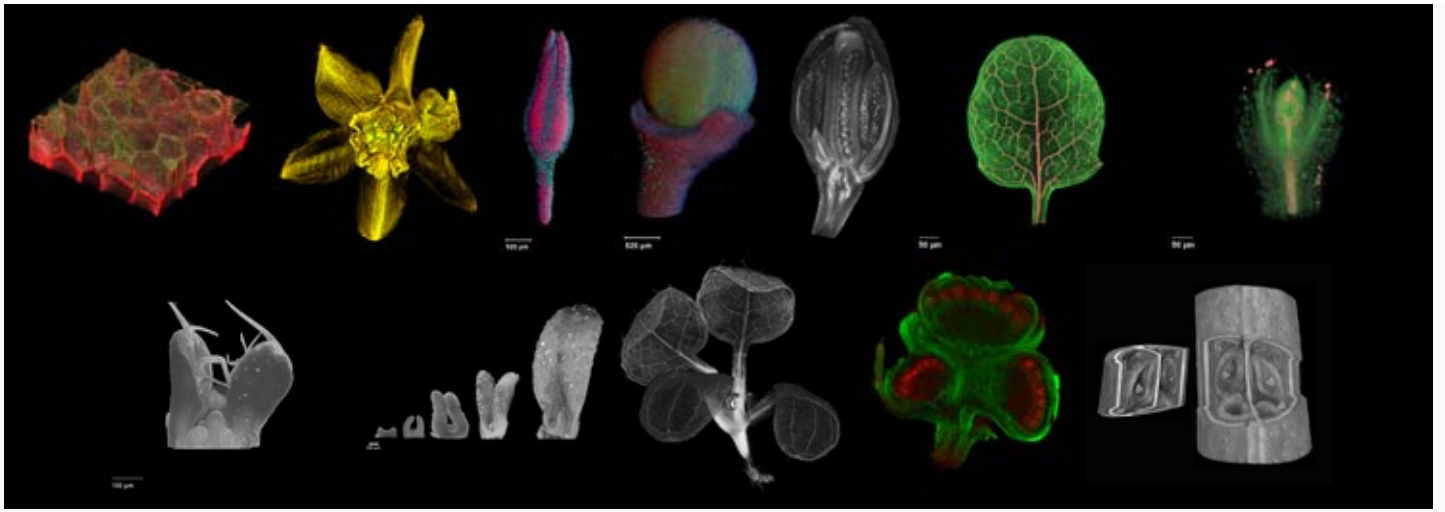
Public SVN: <https://cmpdartsvr1.cmp.uea.ac.uk/banghamlabSVN/VolViewer/>

Building from source

- ▶ [Building VolViewer from source](#)

Image Gallery





Media/Press

VolViewer has appeared in the following:

Front cover: [Handbook of Plant Science](#) | Front cover: [The Plant Cell](#) | [Royal Microscopical Society: Infocus Magazine](#) | Bundled with the [Bioptic 3001 scanner](#): [Bioptics Viewer](#) | [The Guardian newspaper: 3D Fruit fly](#) | [Qt Ambassador program](#) | [Triffid Nurseries website](#)

Author

► [Dr Jerome Avondo](#) Supported by the BBSRC through UEA Computing School and JIC.

modified on 27 March 2012 at 11:53 *** 3,908 views

University of East Anglia / School of Computing Science / Powered by MediaWiki

VoViewerUserManual

Contents [\[hide\]](#)

- [1 Quick Guide](#)
- [2 Loading Volume Data](#)
 - [2.1 Supported File Formats](#)
 - [2.1.1 Image Stacks](#)
 - [2.1.2 Raw Volume](#)
 - [2.2 Loading Data](#)
 - [2.2.1 The File Dropdown Menu](#)
 - [2.2.2 The Drag&Drop Event](#)
 - [2.2.3 Scripting Interface](#)
- [3 Saving Volume Data](#)
- [4 Interacting with the Volume](#)
 - [4.1 Arcball Rotation](#)
 - [4.2 Zooming In/Out](#)
 - [4.3 Translation](#)
- [5 Toolbars](#)
 - [5.1 Viewing](#)
 - [5.2 Tools](#)
 - [5.3 Settings](#)

Quick Guide

Loading Volume Data

The viewer supports two types of file formats for loading volume data. It also supports four different ways to actually load data into the viewer. See the subsections for more information about the file formats and data loading mechanisms.

Supported File Formats

Image Stacks

The viewer supports image stacks in the following file formats: PNG, BMP, TIFF, JPG. For data that does not have equal spacing in the X,Y,Z dimensions, ie: confocal data, you may also create a text file called voxelspacing.txt to specify the voxel spacing independently for each axis. This file needs to be in the same folder as where your image stack files are. This text file takes the following format:

```
x 0.488281
y 0.488281
z 1.506024
```

Note that an image stack is a series of 2D images for each z-series of your stack. The images can be 8bit grey scale or 8bit RGB.

Raw Volume

Raw volumes are supported, these volumes are made up of two files, a .raw and a .dat file. The .raw file is the volume values. These values are stored in binary format, and the voxel values are ordered as R/G/B triplets and in X,Y,Z traversal order. The .dat file with the same filename as the .raw file contains the metadata of the volume. This metadata is the extra information you need to be able to load the volume. This data is X,Y,Z dimensions and the data type.

Loading Data

To load data into the viewer, this can be achieved by one of three mechanisms:

- ▶ Using the **FILE** dropdown menu.
- ▶ Using the **Drag&Drop** event.
- ▶ Using the **scripting interface**

The File Dropdown Menu

The easiest method to load data is to do this via the **FILE** dropdown menu.

Once you select either a stack or a raw volume you will be presented with a file dialogue which you can point to the file you want to load. Note that when loading an image stack you can point the viewer to any file in your stack, it does not have to be the first image of the stack.

Note there are keyboard short-cuts to facilitate the loading of volume data:


```
* CTRL + R: load a raw volume
* CTRL + S: load an image stack
```


The Drag&Drop Event

With the Drag&Drop event you can load data into the viewer by simply dragging one of the supported files to the main rendering window. See the figure below.

The Drag&Drop accepts drops of either; a single image file in you stack, a .dat file or a .raw file.

Scripting Interface

See [this](#)  site for more information.

```
*Exercise 1:
Try loading a volume by downloading the sample volume here  and using the drag and drop method.
```

Saving Volume Data

The viewer allows you to save your volume data either as a .RAW binary file or a stack of PNG images. This is achieved by using the **FILE** dropdown menu and choosing the **SAVE** option. Note that when saving a PNG image stack you will be asked to point the viewer to a directory that it store the volume stack.

Interacting with the Volume

Once a volume is loaded you can interact with the volume by rotating it, moving it (translating), or zooming in and out. This is achieved via the mouse.

Arcball Rotation

To rotate a volume you use the left mouse button. Click on the centre (more or less) of the view screen and without releasing the left mouse button drag the volume in your desired direction you wish to rotate it. See the figure below for more details.

Zooming In/Out

Zooming in and out is achieved using the right mouse button. You can zoom in by clicking and not releasing the right mouse button and moving the mouse down. Or to zoom out you click the right mouse button and without releasing move the mouse up.

Translation

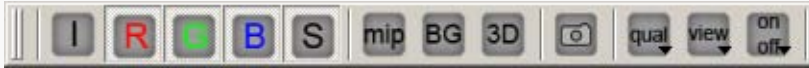
To move or translate a volume you use the middle mouse button. Translation is achieved by clicking and keeping pressed the middle mouse button, then by either moving left, right, up or down you are able to move the volume. To stop translation simply release the middle mouse button.

(NB) Currently the centre of rotation is not affected by the translation, and the volume will still rotate about it's centre even after translation. The ability to change this behaviour will be introduced in a future release.

***Exercise 2:**
Try rotating and zooming the volume.

Toolbars

Viewing



This toolbar allows you to control various viewing options.

- ▶ **I**: Toggles the volume greyscale channel
- ▶ **R**: Toggles the volume red channel
- ▶ **G**: Toggles the volume green channel
- ▶ **B**: Toggles the volume blue channel
- ▶ **S**: Toggles the iso-surface (if computed)
- ▶ **mip**: Toggles the Maximum Intensity Projection mode
- ▶ **BG**: Allows you to specify the background colour
- ▶ **3D**: Allows you to enable stereo rendering (requires a 3D screen)
- ▶ **camera icon**: Takes a screen grab of the current view and prompts you with a save file dialogue.
- ▶ **qual**: Various viewing quality presets.
- ▶ **view**: Various view direction presets.
- ▶ **on/off**: Various viewing toggles.

***Exercise 3:**
Enable/Disable the red channel
Toggle the MIP rendering mode
Use the view menu to view your data from the LEFT orientation.
Use the on/off menu to toggle the axis

Tools



- ▶ **graph icon**: Brings up the transfer function widget.
- ▶ **crop icon**: Brings up the crop widget.
- ▶ **clipping plane icon**: Brings up the clipping widget.
- ▶ **measure icon**: Brings up the measure widget.
- ▶ **surface icon**: Brings up the iso-surface widget.
- ▶ **filter icon**: Brings up the filter widget.

***Exercise 4:**
Open the transfer function widget and adjust the opacity and threshold the volume.
Open the clipping widget to clip the volume
Open the filter widget to apply some Gaussian smoothing to your volume.

Settings



- ▶ **visual bookmarks icon**: Allows you to save an orientation, clip state and transfer function state of your volume as a bookmark
- ▶ **movie icon**: Allows you to save movie frames for your volume.
- ▶ **lighting icon**: Allows you to apply lighting to your volume data.
- ▶ **settings icon**: Allows you to control various performance and quality setting

***Exercise 5:**
Use the visual bookmarks to save two different views of you volume, and toggle between them by double clicking the thumbnails.
Use the movie icon to save the frames for a movie



VolViewerNotes1

Overview

In this session we will learn how to convert proprietary file formats to a VolViewer friendly version.

BioformatsConverter

Dealing with file formats

There exist a wide variety of different file formats used to represent biological images. These file formats can vary due to different manufacturers, imaging software and the type of imaging modality. How we deal with this large variety of different file formats is a challenge. Thankfully there exist an open source community solution called the [LOCI Bio-Formats library](#) developed by the University of Wisconsin.

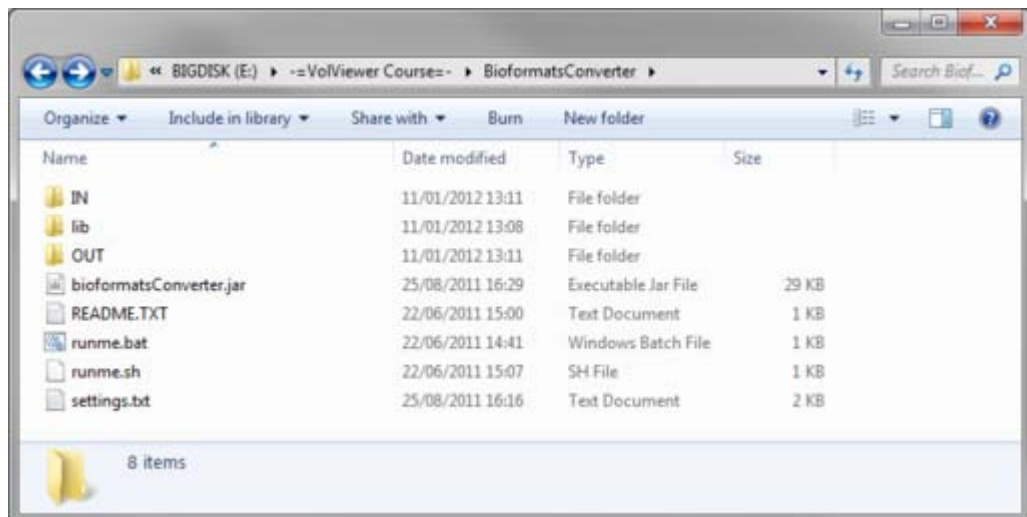
The [LOCI Bio-Formats library](#) supports the following [file formats](#).

A tool using this library has been created called the [Bioformats Converter](#) to help us convert our files to a VolViewer friendly format.

*Exercise 1:

Download the Bioformats Converter tool to your desktop from [here](#).
Extract the file contents to your desktop.
Copy the sample data folder to your desktop.

You should now have the Bioformats Converter tool folder extracted on your desktop. Inside the folder you should see the following layout:



You can convert proprietary file formats to a VolViewer friendly version using this tool. In order to do so you will require to copy a file from the Proprietary File Formats directory found in the sample data directory into the IN folder of the Bioformats converter tool.

*Exercise 2:

Copy the E346_100h_PI_01.lsm from the sample data folder into the IN folder.

The Bioformats Converter tool can be configured by editing the settings.txt file. This file will give you various options for how you wish to split the original data. For the purpose of this workshop we will focus on PNG stacks.

*Exercise 3:

Open the settings.txt file and check that the parameters are set to PNG and SINGLE_PNG.

navigation

- [Main Page](#)
- [Recent changes](#)

help

- [Help](#)

search

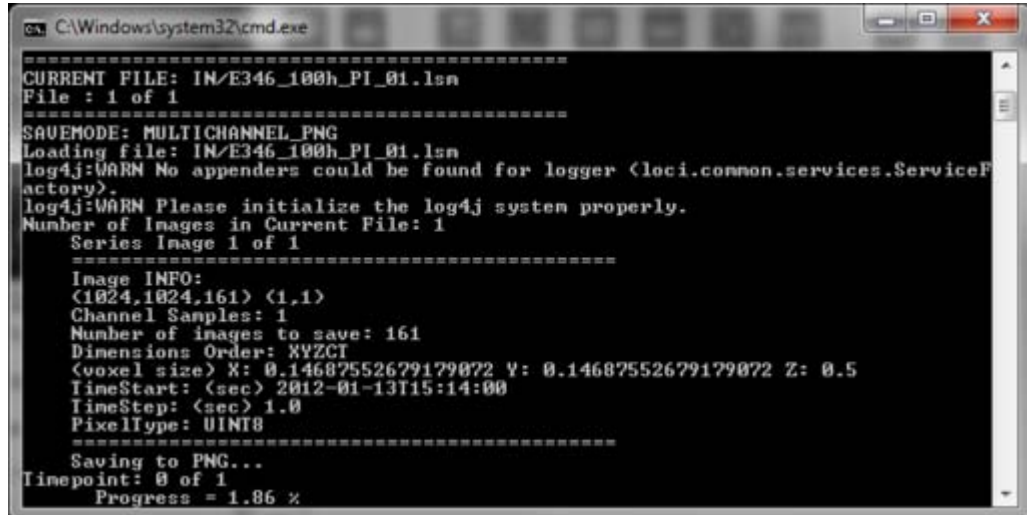
toolbox

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Printable version](#)
- [Permanent link](#)

To run the tool you need to double click on the runme.bat file. This will launch the conversion process.

***Exercise 4:**
Launch the conversion process.

During the conversion process the tool will display some information about your file as well as it's current process as shown below.



```
C:\Windows\system32\cmd.exe
=====
CURRENT FILE: IN/E346_100h_PI_01.lsm
File : 1 of 1
=====
SAVEMODE: MULTICHANNEL_PNG
Loading file: IN/E346_100h_PI_01.lsm
log4j:WARN No appenders could be found for logger (loci.common.services.ServiceFactory).
log4j:WARN Please initialize the log4j system properly.
Number of Images in Current File: 1
Series Image 1 of 1
=====
Image INFO:
(1024,1024,161) (1,1)
Channel Samples: 1
Number of images to save: 161
Dimensions Order: XYZCT
(voxel size) X: 0.14687552679179072 Y: 0.14687552679179072 Z: 0.5
TimeStart: (sec) 2012-01-13T15:14:00
TimeStep: (sec) 1.0
PixelType: UINT8
=====
Saving to PNG...
Timepoint: 0 of 1
Progress = 1.86 %
```

Once the conversion process has finished you should see a *Finished...* message in the information window. The converted data will appear in the *OUT* folder of the tool.

***Exercise 5:**
Locate the converted data found in the *OUT* folder.

What you will find in this folder is a Z series of PNG files representing your image as well as two text files called *voxelscale.txt* and *voxelspacing.txt*. These two additional files will be created automatically by the tool and will contain your image pixel/voxel physical dimensions.

Note that the tool can convert more than one file at the time by simple adding files to the *IN* folder.

***Exercise 6:**
Edit the *settings.txt* file to change the conversion mode to *MULTICHANNEL_PNG*.
Add some additional files from the *Proprietary File Formats* folder
And relaunch the conversion process

We have now covered the basic task of converting data to a common file format that you will be able to load into VoIViewer. We also saw the basic data format supported by VoIViewer which was a Z series of PNG files along with two metadata text files; *voxelscale.txt* and *voxelspacing.txt*



navigation

- Main Page
- Recent changes

help

- Help

search

toolbox

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link

VolViewerNotes5

Contents [\[hide\]](#)

- 1 Overview
- 2 VolViewer
 - 2.1 Loading data
 - 2.1.1 Menu Icon method
 - 2.1.2 Drag & Drop method
 - 2.2 Interacting with the Volume
 - 2.2.1 Rotation
 - 2.2.2 Zooming In/Out
 - 2.2.3 Translation
 - 2.3 Loading Multi-Channel data
 - 2.4 Loading Time-lapse data
 - 2.5 Saving Data

Overview

In this session we will go over the different methods to import single channel, multi-channel and timelapse data into VolViewer. We will get a quick overview to the different ways we can interact with the Volume view. And we will also learn how to export data.

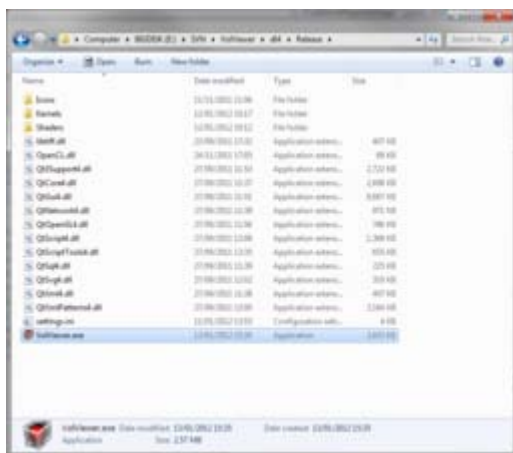
VolViewer

Loading data

We will now load some of our converted data into VolViewer. First you need to download the VolViewer application.

***Exercise 1:**
 Download VolViewer to your desktop from [here](#).
 Extract the file contents to your desktop.

You should now have a folder called VolViewer as shown below:




There are two important files in this folder. The *VolViewer.exe* file which is the program executable and the *Settings.ini* file which contains many options to configure VolViewer's default states.

***Exercise 2:**
 Launch VolViewer by double clicking on the VolViewer.exe file.

Once VolViewer has launched you should see the following main window:



To load data into the viewer there are four main options:

- The  menu icon
- Drag and Drop
- Keyboard shortcuts
- Scripts

To start simple we will focus on the two first methods.

Menu Icon method

First we will start by using the menu icon method.

*Exercise 3:

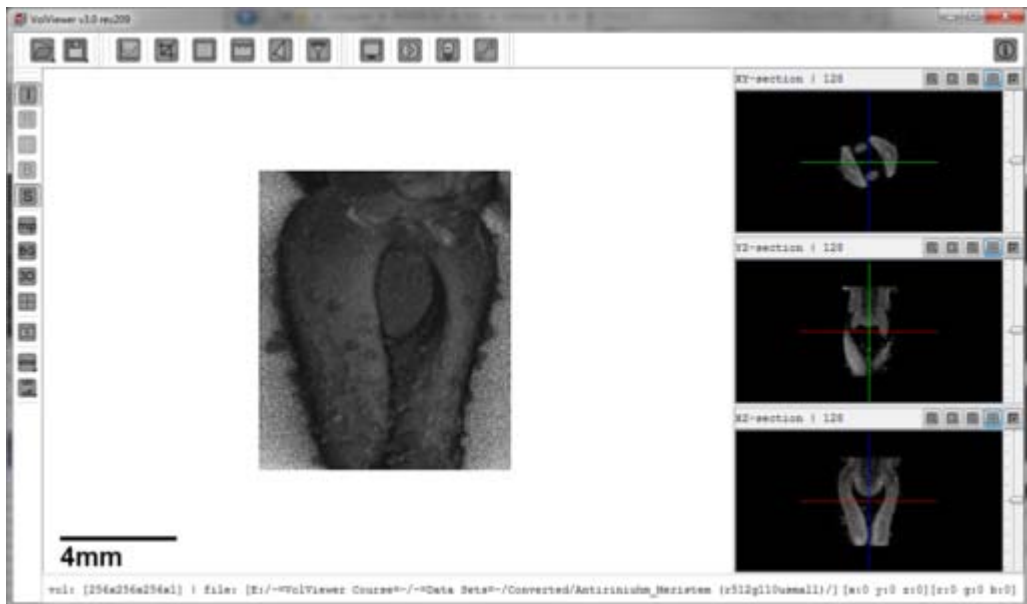
Click on the  icon.

Select the *Image Stack BMP/PNG/JPG/TIFF* entry.
Point the program to the *Sample Data\Converted\Antiriniuhm_Meristem (r512g110usmall)*' folder that contains a Z stacks of PNGs.

The viewer will then prompt you with the channel window as shown below:

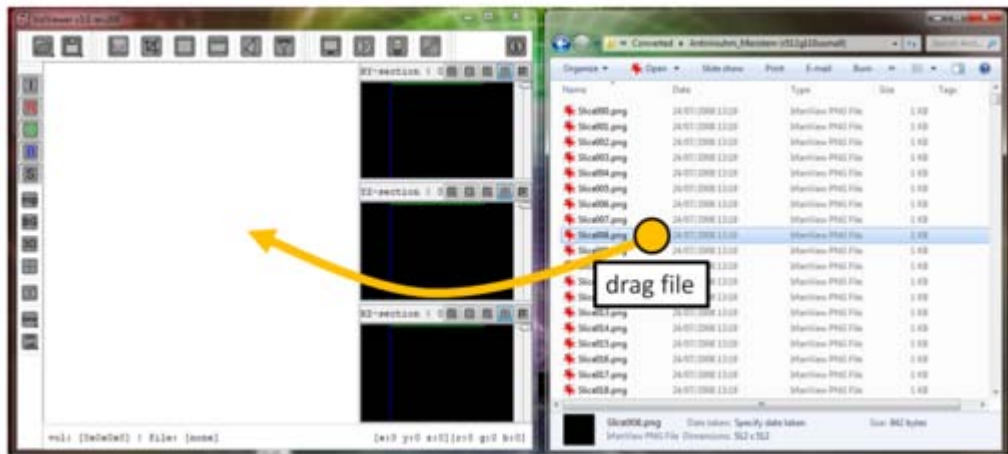


Click on the RGBA button and you should then see the following:



Drag & Drop method

Another way to load data into VolViewer is to use drag and drop.

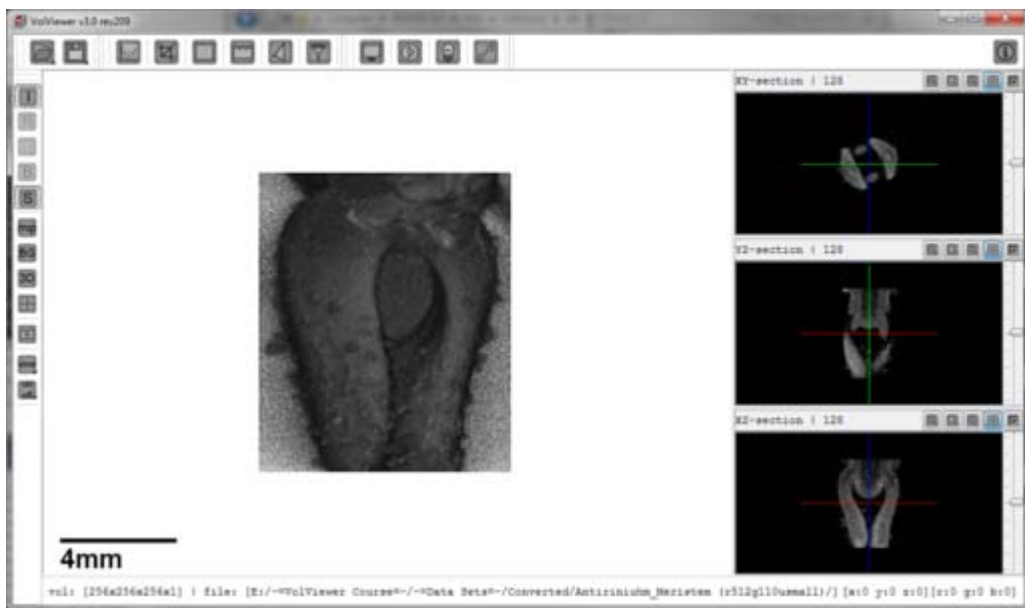


***Exercise 4:**
 Locate a folder in *Sample Data\Converted* which contains a Z stack of PNGs.
 Drag one of the PNG files onto the central window.

Once you have dragged the file on the central window (white) the channel window will appear again.



Click on the RGBA button and you should then see the following:



Interacting with the Volume

Once a volume is loaded you can interact with the volume by rotating it, moving it (translating), or zooming in and out. This is achieved via the mouse.

Rotation

To rotate a volume you use the left mouse button. Click on the centre (more or less) of the view screen and without releasing the left mouse button drag the volume in your desired direction you wish to rotate it.

Zooming In/Out

Zooming in and out is achieved using the right mouse button. You can zoom in by clicking and not releasing the right mouse button and moving the mouse down. Or to zoom out you click the right mouse button and without releasing move the mouse up.

Translation

To move or translate a volume you use the middle mouse button. Translation is achieved by clicking and keeping pressed the middle mouse button, then by either moving left, right, up or down you are able to move the volume. To stop translation simply release the middle mouse button.

*Exercise 5:
Try rotating and zooming the volume.

Loading Multi-Channel data

It is also possible to load multi-channel data into the viewer. VolViewer supports up to 3 different data channels. Loading multi-channel data is achieved in a similar manner as to how we previously loaded data. For simplicity we will use the *drag & drop* method.

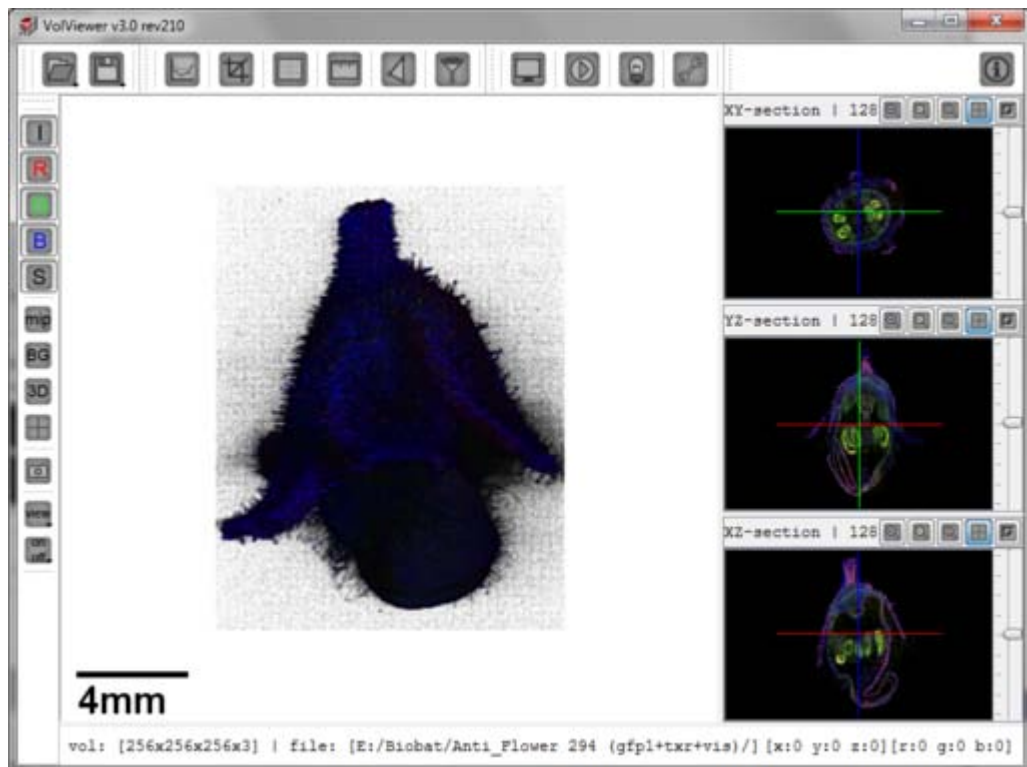
*Exercise 6:
Locate the `Converted\Anti_Flower 294 (gfpl+txr+vis)` folder
Drag and drop a PNG file from this stacks into the main window.

The channel window will now appear again:




With multi-channel data, you can either import all three channels at once (RGB), or individual channels at a time (R or G or B).


*Exercise 7:
Click on the R button to import only the red channel.
Repeat the drag and drop procedure but this time import the green channel.



Loading Time-lapse data

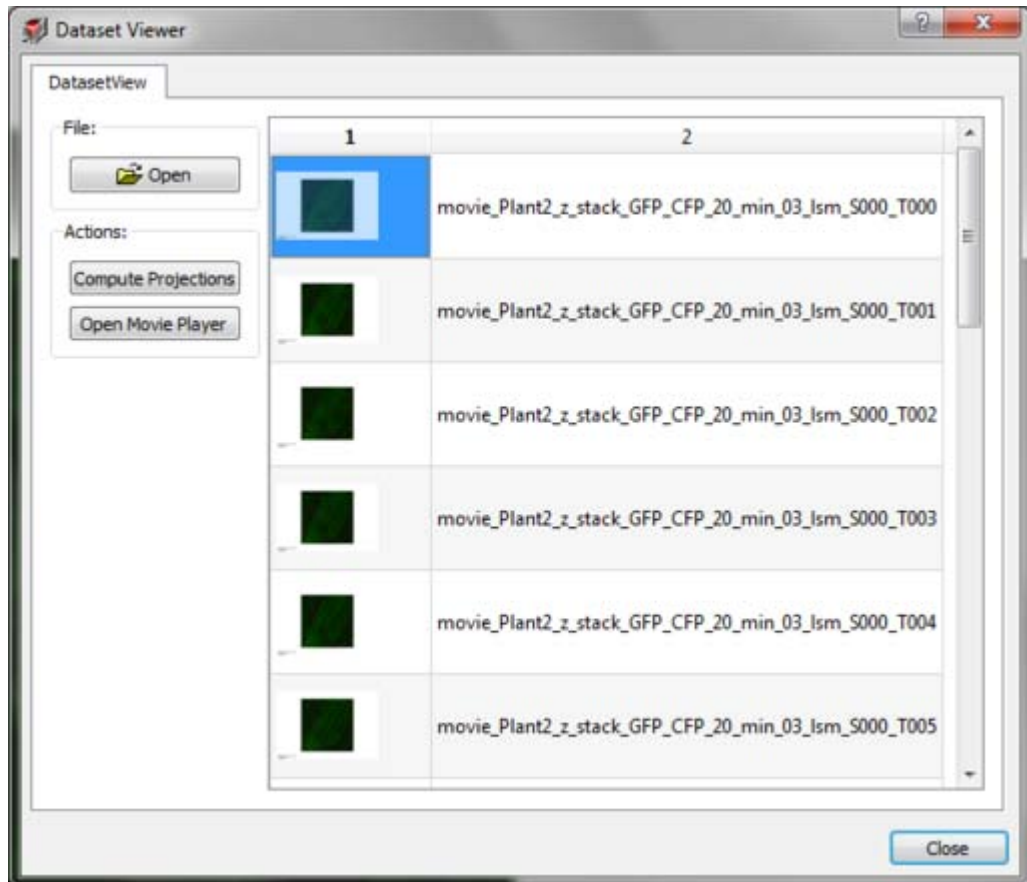
It is also possible to load time-lapse data into the viewer. Time-lapse data is supported by loading a directory containing many Z stacks. The data is loading using the  icon.

*Exercise 8:

Click on the  icon.

Select the *Directory of Stacks BMP/PNG/JPG/TIFF* entry.
Point the program to the *Sample Data\Converted\movie_Plant2_z_stack_GFP_CFP_20_min_03_lsm'* folder that contains the *timepoint Z stacks*.


The following window will then appear.



This window contains a list of all the time points for the folder you have loaded. Loading individual timepoints is achieved by double clicking on one of the thumbnail images of the list.

***Exercise 9:**
Double click on an individual timepoint to load it.

Saving Data

A stack can be exported to VolViewer by pressing the  icon. You can choose to save your stack either as a .raw binary file or as a PNG z stack.

***Exercise 10:**
Save a stack as a PNG z stack to a folder on your desktop.



VolViewerNotes5

Contents [\[hide\]](#)

- 1 Overview
- 2 VolViewer
 - 2.1 Loading data
 - 2.1.1 Menu Icon method
 - 2.1.2 Drag & Drop method
 - 2.2 Interacting with the Volume
 - 2.2.1 Rotation
 - 2.2.2 Zooming In/Out
 - 2.2.3 Translation
 - 2.3 Loading Multi-Channel data
 - 2.4 Loading Time-lapse data
 - 2.5 Saving Data

Overview

In this session we will go over the different methods to import single channel, multi-channel and timelapse data into VolViewer. We will get a quick overview to the different ways we can interact with the Volume view. And we will also learn how to export data.

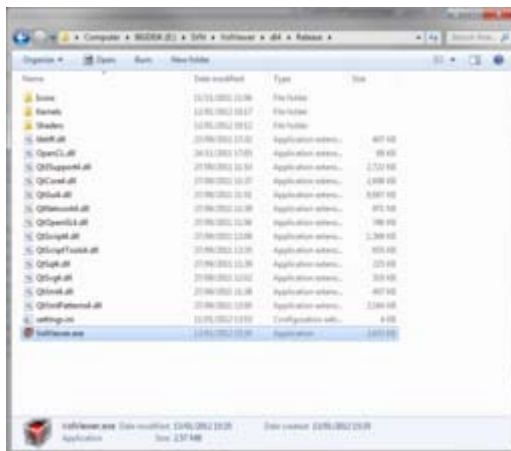
VolViewer

Loading data

We will now load some of our converted data into VolViewer. First you need to download the VolViewer application.

***Exercise 1:**
 Download VolViewer to your desktop from [here](#).
 Extract the file contents to your desktop.

You should now have a folder called VolViewer as shown below:



There are two important files in this folder. The *VolViewer.exe* file which is the program executable and the *Settings.ini* file which contains many options to configure VolViewer's default states.

***Exercise 2:**
 Launch VolViewer by double clicking on the VolViewer.exe file.

navigation

- [Main Page](#)
- [Recent changes](#)

help

- [Help](#)

search


toolbox

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Printable version](#)
- [Permanent link](#)

Once VolViewer has launched you should see the following main window:



To load data into the viewer there are four main options:

- The  menu icon
- Drag and Drop
- Keyboard shortcuts
- Scripts

To start simple we will focus on the two first methods.

Menu Icon method

First we will start by using the menu icon method.

*Exercise 3:

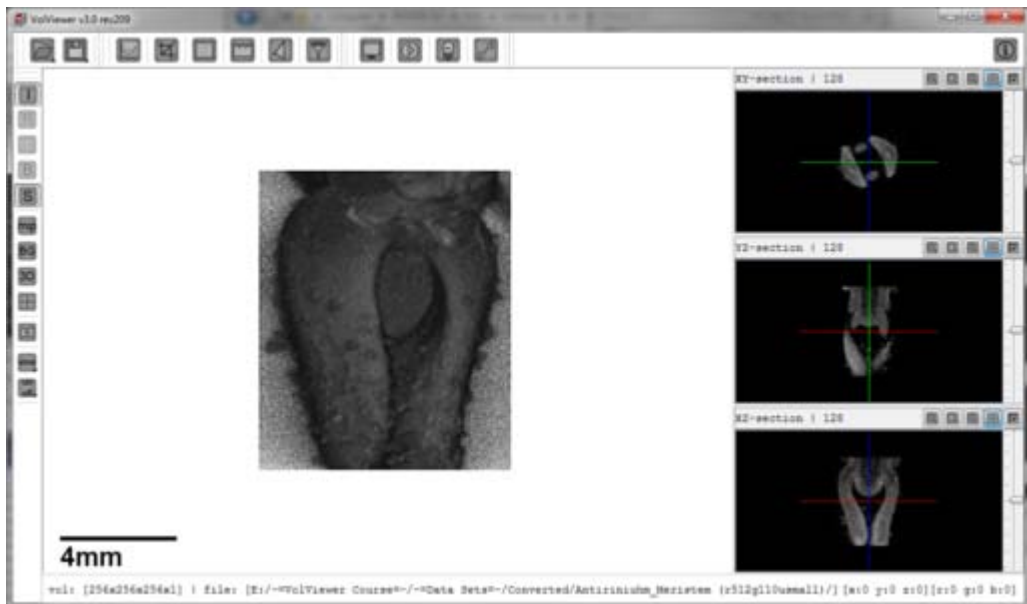
Click on the  icon.

Select the *Image Stack BMP/PNG/JPG/TIFF* entry.
Point the program to the *Sample Data\Converted\Antiriniuhm_Meristem (r512g110usmall)*' folder that contains a Z stacks of PNGs.

The viewer will then prompt you with the channel window as shown below:

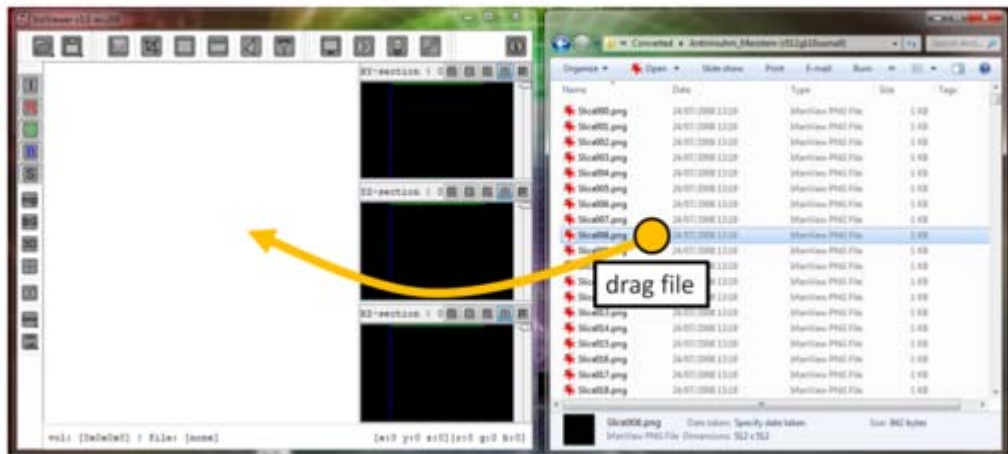


Click on the RGBA button and you should then see the following:



Drag & Drop method

Another way to load data into VolViewer is to use drag and drop.

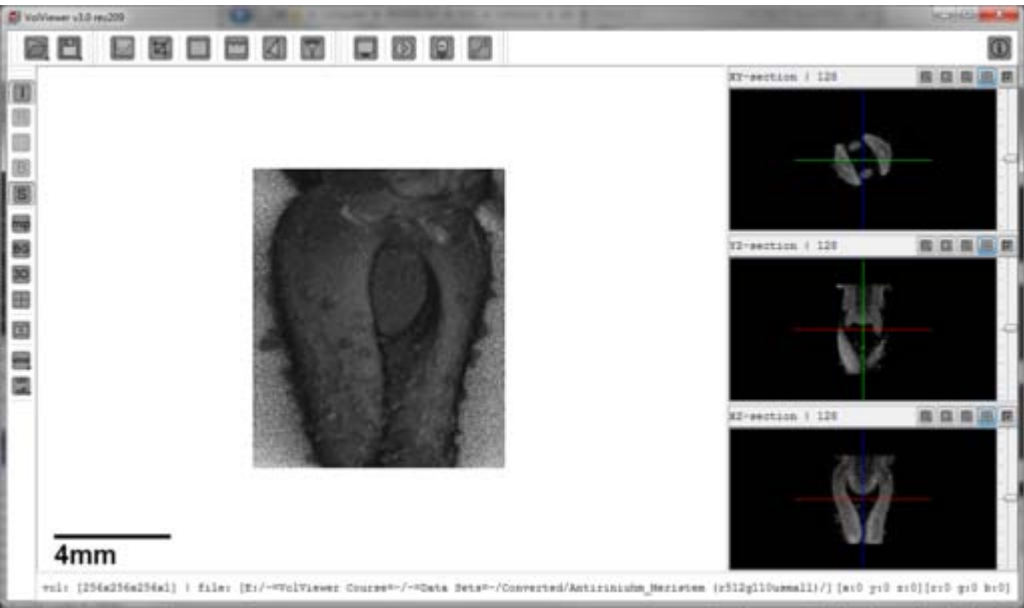


***Exercise 4:**
 Locate a folder in *Sample Data\Converted* which contains a Z stack of PNGs.
 Drag one of the PNG files onto the central window.

Once you have dragged the file on the central window (white) the channel window will appear again.



Click on the RGBA button and you should then see the following:



Interacting with the Volume

Once a volume is loaded you can interact with the volume by rotating it, moving it (translating), or zooming in and out. This is achieved via the mouse.

Rotation

To rotate a volume you use the left mouse button. Click on the centre (more or less) of the view screen and without releasing the left mouse button drag the volume in your desired direction you wish to rotate it.

Zooming In/Out

Zooming in and out is achieved using the right mouse button. You can zoom in by clicking and not releasing the right mouse button and moving the mouse down. Or to zoom out you click the right mouse button and without releasing move the mouse up.

Translation

To move or translate a volume you use the middle mouse button. Translation is achieved by clicking and keeping pressed the middle mouse button, then by either moving left, right, up or down you are able to move the volume. To stop translation simply release the middle mouse button.

*Exercise 5:
Try rotating and zooming the volume.

Loading Multi-Channel data

It is also possible to load multi-channel data into the viewer. VolViewer supports up to 3 different data channels. Loading multi-channel data is achieved in a similar manner as to how we previously loaded data. For simplicity we will use the *drag & drop* method.

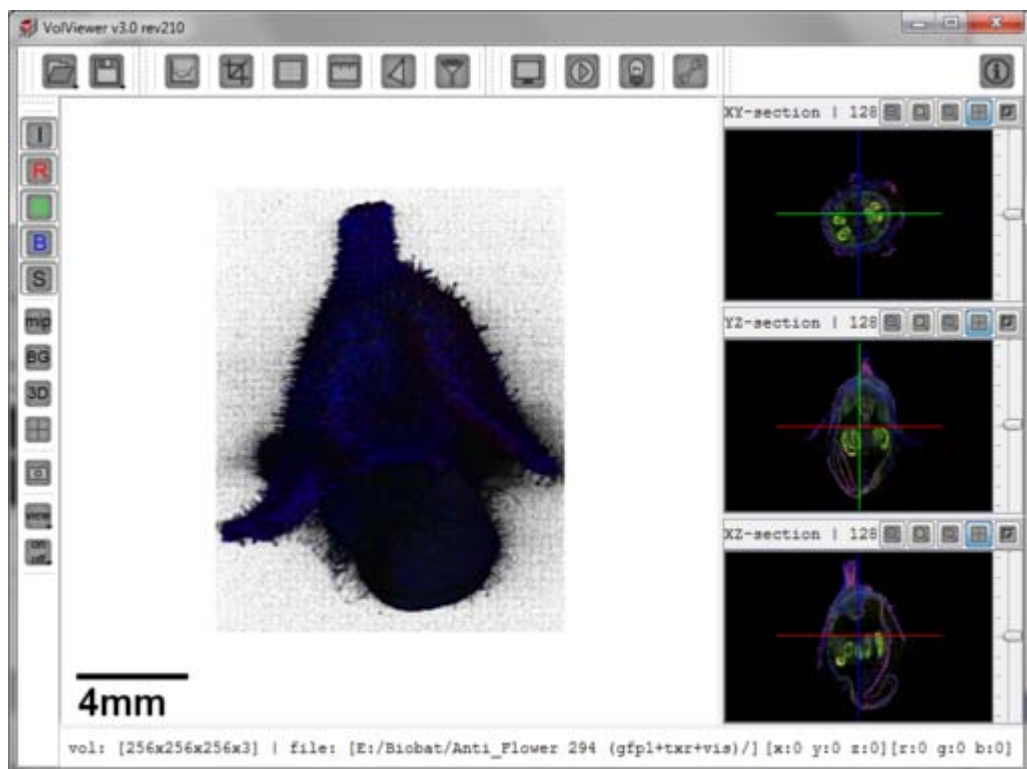
*Exercise 6:
Locate the `Converted\Anti_Flower 294 (gfpl+txr+vis)` folder
Drag and drop a PNG file from this stacks into the main window.

The channel window will now appear again:




With multi-channel data, you can either import all three channels at once (RGB), or individual channels at a time (R or G or B).


*Exercise 7:
Click on the R button to import only the red channel.
Repeat the drag and drop procedure but this time import the green channel.



Loading Time-lapse data

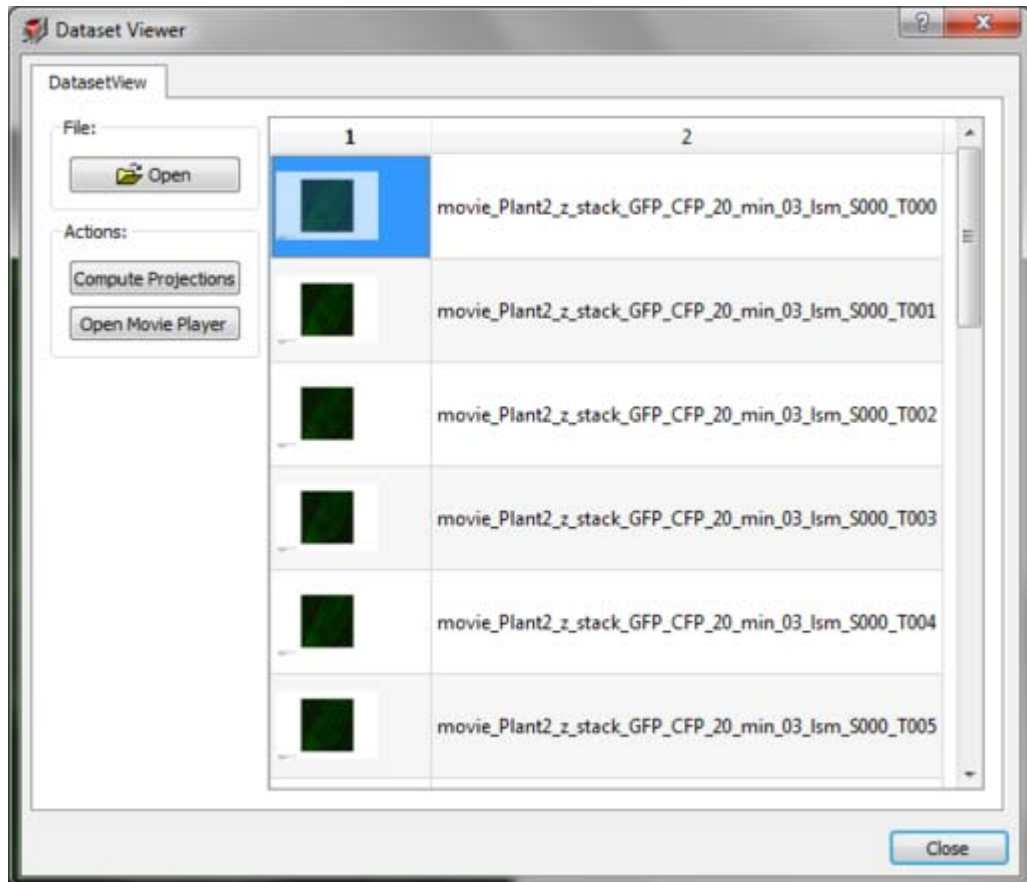
It is also possible to load time-lapse data into the viewer. Time-lapse data is supported by loading a directory containing many Z stacks. The data is loading using the  icon.

*Exercise 8:

Click on the  icon.

Select the *Directory of Stacks BMP/PNG/JPG/TIFF* entry.
Point the program to the *Sample Data\Converted\movie_Plant2_z_stack_GFP_CFP_20_min_03_lsm'* folder that contains the *timepoint Z stacks*.


The following window will then appear.



This window contains a list of all the time points for the folder you have loaded. Loading individual timepoints is achieved by double clicking on one of the thumbnail images of the list.

***Exercise 9:**
Double click on an individual timepoint to load it.

Saving Data

A stack can be exported to VolViewer by pressing the  icon. You can choose to save your stack either as a .raw binary file or as a PNG z stack.

***Exercise 10:**
Save a stack as a PNG z stack to a folder on your desktop.



- navigation
- [Main Page](#)
 - [Recent changes](#)
- help
- [Help](#)
- search
-
- toolbox
- [What links here](#)
 - [Related changes](#)
 - [Special pages](#)
 - [Printable version](#)
 - [Permanent link](#)

VolViewerNotes2

Contents [hide]

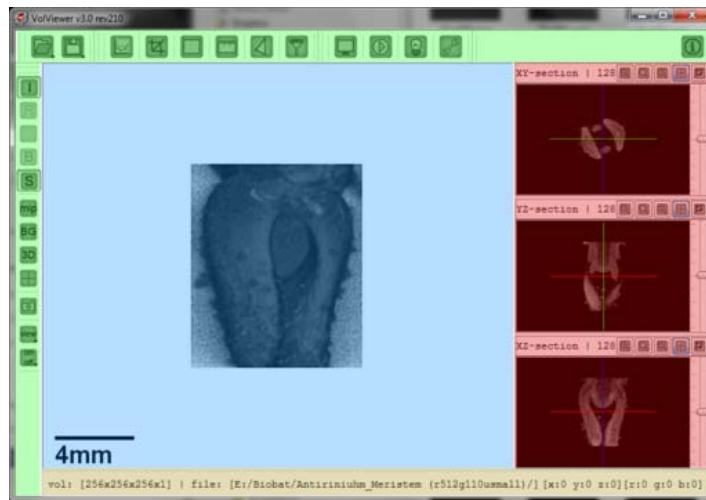
- 1 Overview
 - 1.1 The Application
 - 1.2 Section windows
 - 1.2.1 Layout Manager
 - 1.3 Toolbars
 - 1.4 Render Window
 - 1.4.1 Rotation
 - 1.4.2 Zooming In/Out
 - 1.4.3 Translation
 - 1.4.4 Render Settings
 - 1.5 The Status Bar

Overview

In this session we will go over the main features of the user interface.

The Application

The application is divided into 4 main sections:



- The section windows (red)
- The toolbars (green)
- The rendering window (blue)
- The status bar (orange)

***Exercise 1:**
Load the *Sample Data\Converted\Antiriniuhm_Meristem (r512g110usmall)* dataset as an RGBA volume.

Section windows

The section windows contain 3 orthogonal views of the data. These views are complementary to the 3D view. Each view has an associated slider which allows you to navigate through the volume.

***Exercise 2:**
Try navigating up and down a specific section view using the slider.

Each section view also contains a toolbar found on the top of the view. This toolbar has the following icons which allow you to:

- : Zoom in
- : Zoom out
- : Reset the zoom

***Exercise 3:**
Try zooming in and out on a specific section view.

You can also click on a particular region of a section, this will also update the other 2 section views to reflect the new region of interest. The sections are labelled, using the section cursor, with the red/green/blue lines. This cursor is used as a navigation guide. The cursor can be toggled on and off using the icon.

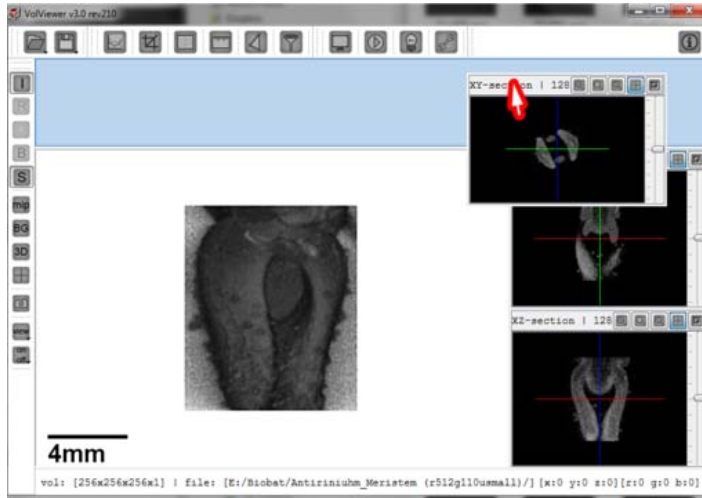
***Exercise 4:**
Try clicking on a region of a section view, and notice how the other views get updated. Try toggling on and off the section cursor for a particular view.

Layout Manager

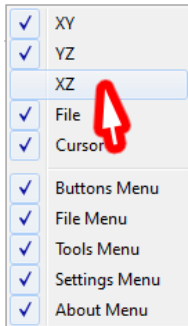
The application allows for a flexible user interface layout. You are free to specify your own custom view and save this as the default layout for future sessions. The toolbars and sections can be *pulled off* from their current location and be placed in new *hot spot* areas in the application.

***Exercise 5:**
 Try moving one of the section views by click and dragging the left mouse button on the toolbar section of the section view. Without releasing the mouse button move the section around the application. Blue areas (hot spots) will appear, showing you where you can drop and release the section.

You should be able to drag off a section view as shown below:

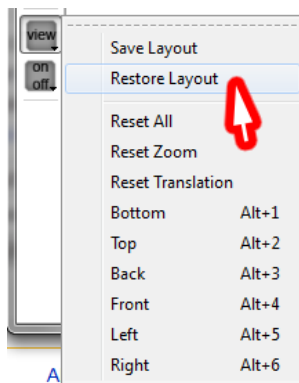


You can also hide and show certain areas of the user interface using the layout menu. The menu can be accessed by right clicking on any of the toolbars. The following menu should appear, allowing you to toggle various layout elements.



***Exercise 6:**
 Try toggling on and off the section views.

The default layout can be restored or saved for future sessions. This is achieved by using the layout options found in the **view** menu on the left toolbar.



***Exercise 7:**
 Restore the default view by selecting the *Restore Layout* option in the **view** menu.

Using the layout manager it is now possible to specify various layouts for the application as shown below:



***Exercise 8:**
 Try to reproduce the 4 layouts shown above.

Toolbars

The first toolbar we will focus on is the *Viewing Toolbar* found on the left of the application.

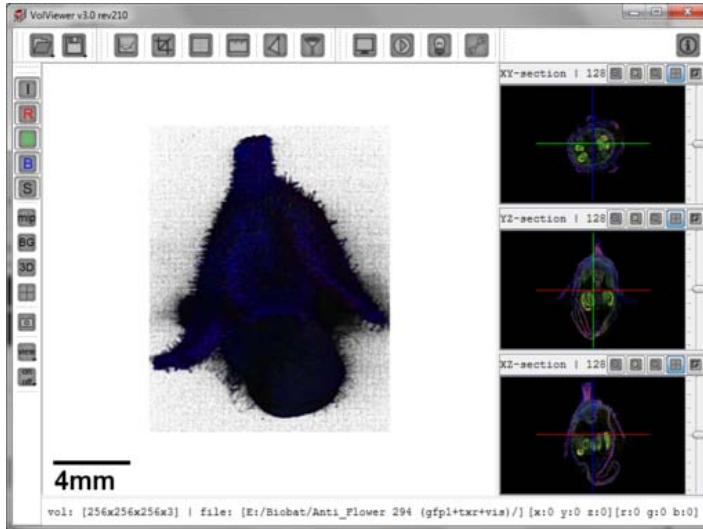


This toolbar contains many of the viewing options. We will first start by loading a multi-channel volume.

***Exercise 9:**

Load the `Sample Data\Converted\Anti_Flower 294 (gfp1+txr+vis)` volume as an RGB volume.

What you should end up with is a similar view as shown below:



The first part of the toolbar contains various viewing toggles.

- **I** : This will toggle all the channels on/off
- **R** : This will toggle the red channel on/off
- **G** : This will toggle the green channel on/off
- **B** : This will toggle the blue channel on/off
- **S** : This will toggle the surfaces on/off

***Exercise 10:**

Try toggling some of the data channels on and off.

Next we can also toggle the different projection modes. You can toggle between a *SUM* projection or a *Maximum Intensity Projection* mode.

- **mip** : This will toggle between a *SUM* and *MIP* projection.

***Exercise 11:**

Toggle between the two projection modes.

You can also choose the default background colour for the rendering window.

- **BG** : This will allow you to pick a background colour for the rendering window.

The viewer also supports 3D/stereo rendering.

- **3D** : This will bring up the stereo rendering menu.

3D/Stereo rendering supports two modes. The first is a quad buffered mode to use on high end graphic cards and 3D monitors. Whilst the second mode is an anaglyph mode which supports colour filtered glasses such as red/green filters.

The viewer also has a 3D cursor, which when enabled allows you to click on the rendering window and will update the section views to reflect the cursor position.

- **+** : This will toggle the 3D cursor on and off.

For easy access a save projection button allows you to quickly save the current projection of your data as show in the rendering window.

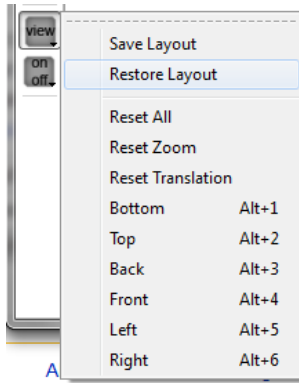
- **📷** : This will allow you to quickly save the current projection.

***Exercise 12:**

Try changing the background colour, using the 3D cursor and saving a projection.

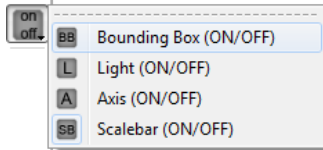
The toolbar also contains two sub-menus called the view menu and the on/off menu.

The view menu allows you to save/restore the user interface layout as we previously saw. But also allows you to restore the default zoom and translation values. And also allows you to specify a desired viewing direction.



***Exercise 12:**
Try changing the current viewing direction.

Lastly the on/off menu allows you to toggle various display objects in the rendering window. You can turn on/off a bounding box, the light (which we will cover later), an axis and the scale bar.



***Exercise 12:**
Try enabling the bounding box and then rotating the volume.
Try disabling the scale bar.

All sub-menus have a dotted line on the top of the menu entries. This allows you to *tear-off* the menu for easier access.

***Exercise 13:**
Bring up the view menu and tear it off by clicking on the dotted lines entry.

Render Window

Central to the application is the render window. This window will display a projection of your data.



As we saw previously you can interact with the render window by rotating, moving (translating), and zooming in and out. This is achieved via the mouse:

Rotation

To rotate a volume you use the left mouse button. Click on the centre (more or less) of the view screen and without releasing the left mouse button drag the volume in your desired direction you wish to rotate it.

Zooming In/Out

Zooming in and out is achieved using the right mouse button. You can zoom in by clicking and not releasing the right mouse button and moving the mouse down. Or to zoom out you click the right mouse button and without releasing move the mouse up.

Translation

To move or translate a volume you use the middle mouse button. Translation is achieved by clicking and keeping pressed the middle mouse button, then by either moving left, right, up or down you are able to move the volume. To stop translation simply release the middle mouse button.

***Exercise 14:**
Load the Sample Data\Converted\Antiriniuhm_Meristem (x512g110usmall) dataset as an *RGBA* volume.
Try rotating and zooming the volume.

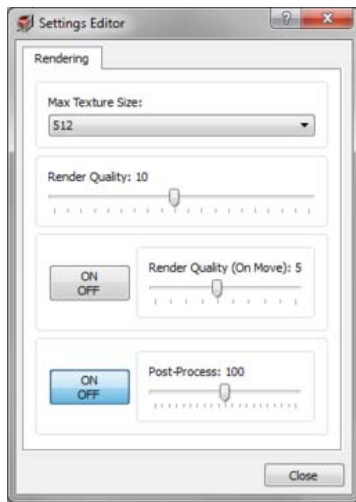
Render Settings

You can control the quality of the render window by using the *Settings Editor*  icon found on the top toolbar.



: This will bring up the *Settings Editor* menu.

Clicking on this icon will bring up the following menu:



This menu allows you to control the maximum volume size you can load, the quality of the rendering and the amount of post-processing applied to the rendering.

***Exercise 15:**

Try moving the *Render Quality* slider and see the effect it has on the render window.

The *Render Quality* slider allows you to specify the trade-off between speed and quality for the render window. It is sometimes useful to reduce the quality whilst we interact with the volume, but then increase it when we need to save a projection.

***Exercise 16:**

Try enabling the *Render Quality (On Move)* button. Rotate the volume to see what happens.

The *Render Quality (On Move)* button and slider allow you to enable a separate lower quality render setting for when you are interacting with the volume. When you stop interacting with the volume the render window will switch back to the main render quality setting.

***Exercise 17:**

Try enabling/disabling and moving the slider of the *Post-Process* options.

The *Post-Process* option allows you to enable/disable a post-processing stage in the rendering. This allows you to enhance the low resolution of the render window.

Lastly and one of the most important options is the *Max Texture Size* drop down menu. This menu will setup a maximum image dimension size that the viewer can load. Any Z stack that is larger than this user specified limit will get automatically resampled upon loading.

***Exercise 18:**

Set the *Max Texture Size* to 512
And reload the `Sample Data\Converted\Antiriniuhm_Meristem (r512g110usmall)` dataset as an RGBA volume.
Note the improved image quality due to less resampling.

The Status Bar

The status bar contains 3 useful pieces of information.

```
vol: [256x256x256x1] | file: [E:/Biobat/Antiriniuhm_Meristem (r512g110usmall)/] [x:128 y:90 z:163] [r:132 g:132 b:132]
```

- The image dimensions (X, Y, Z, C)
- The path of the currently loaded data
- The X,Y,Z and R,G,B values of the cursor



VolViewerNotes3

Contents [[hide](#)]

- 1 Overview
 - 1.1 Cleaning up your data
 - 1.1.1 Transfer Functions
 - 1.1.1.1 The Sliders
 - 1.1.1.2 The Function Editor
 - 1.1.2 Cropping
 - 1.2 Exploring your data
 - 1.2.1 Clipping Planes
 - 1.2.2 Visual Bookmarks

Overview

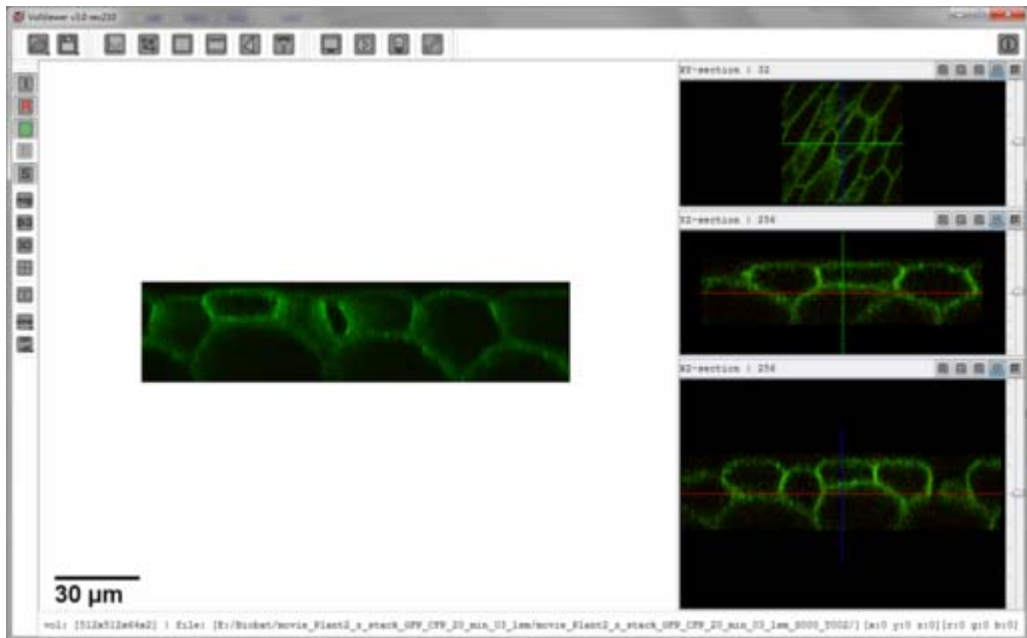
In this session we will go over some of the tools available to clean up and explore your data.

Cleaning up your data

We will start by loading a multi-channel data set.

*Exercise 1:
Load a single stack from the Sample
Data\Converted\movie_Plant2_z_stack_GFP_CFP_20_min_03_lsm\ dataset as an RGBA volume.

You should end up with the following:



Notice how this dataset contains two data channels, the red and green channels.

*Exercise 2:
Try toggling on/off the red and green channels to view them independently.

Transfer Functions

One of the simplest ways we have to adjust our data is to use transfer functions.

A transfer function is responsible for taking an input value and remapping this input to a desired output value. We can use this in VolViewer to control how our data values are mapped to both

intensity and opacity values which are used for the rendering window.

The transfer function window is brought up by clicking on the *Transfer Functions*

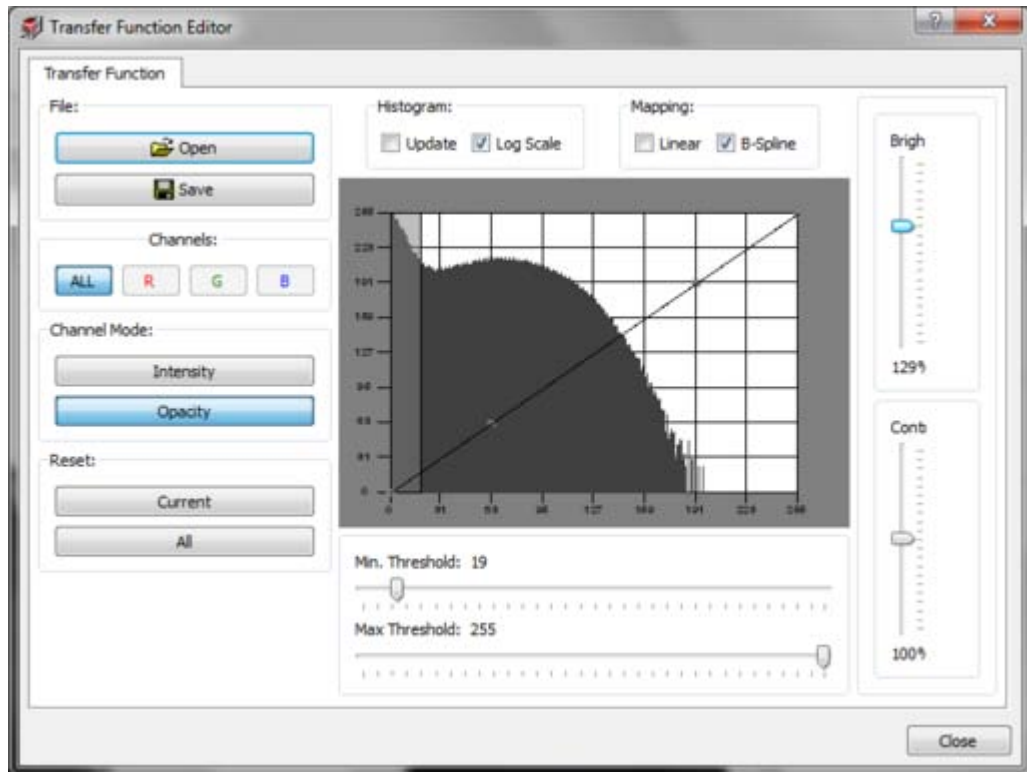


button

found on the top toolbar.

*Exercise 3:
Bring up the *Transfer Function* window.

You should then see the following window:



Central to this tool is a histogram view/panel of our data, along with a mapping function (straight line) which we use see later. Using this tool we can adjust either all the channels at once or individual channels using the *Channels*: toggle buttons.



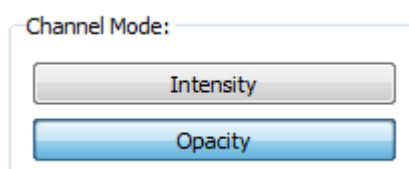
Selecting *ALL* will apply the transfer function settings to all the channels whilst selecting the *R*, *G* or *B* will allow you to adjust each channel independently.

For the purpose of this exercise we will start by adjusting the Green channel of our sample data.

*Exercise 4:
Start by turning off the Red channel in the render window.
Select the TOP view from the view sub-menu.
Select the Green channel in the *Channels* of the transfer functions.

Notice how the histogram becomes green when you select the green channel.

It is possible to independently adjust the channels intensity and opacity. This is specified by selecting the appropriate option in the *Channel mode*: button group.



We will start by leaving this at it's default value and start by adjusting the opacity. There are two main ways to adjust the channel property;

- The sliders
- The mapping function editor

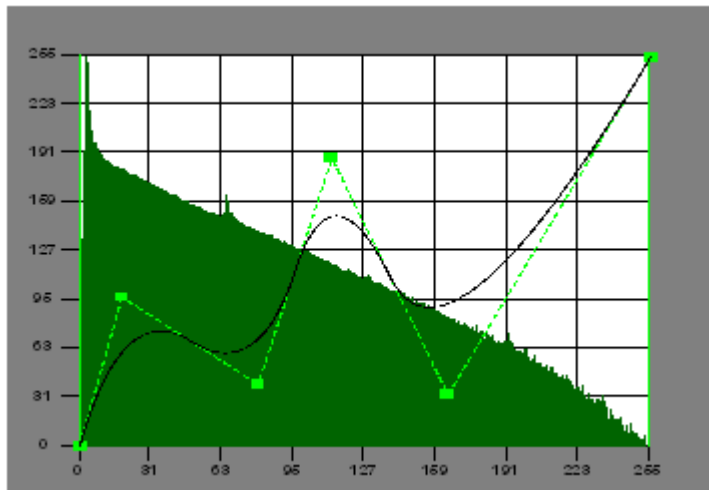
The Sliders

There are 4 sliders you can use to set the minimum, maximum threshold and brightness, contrast values.

```
*Exercise 5:  
Remove some of the noise by setting the minimum threshold to 15  
Note how some of the noise is removed  
Increase the channels transparency by reducing the brightness to 50%
```

The Function Editor

The function editor gives you greater control over how you wish to remap the data. This is achieved by moving the *control points* that are found in the histogram panel. The control points are little squares you can click and drag with your mouse. You can also add control points by right clicking anywhere in the histogram panel. Control points can be removed by dragging them off the histogram panel. Using this it is possible to specify more complex mapping functions than what is possible with the sliders.



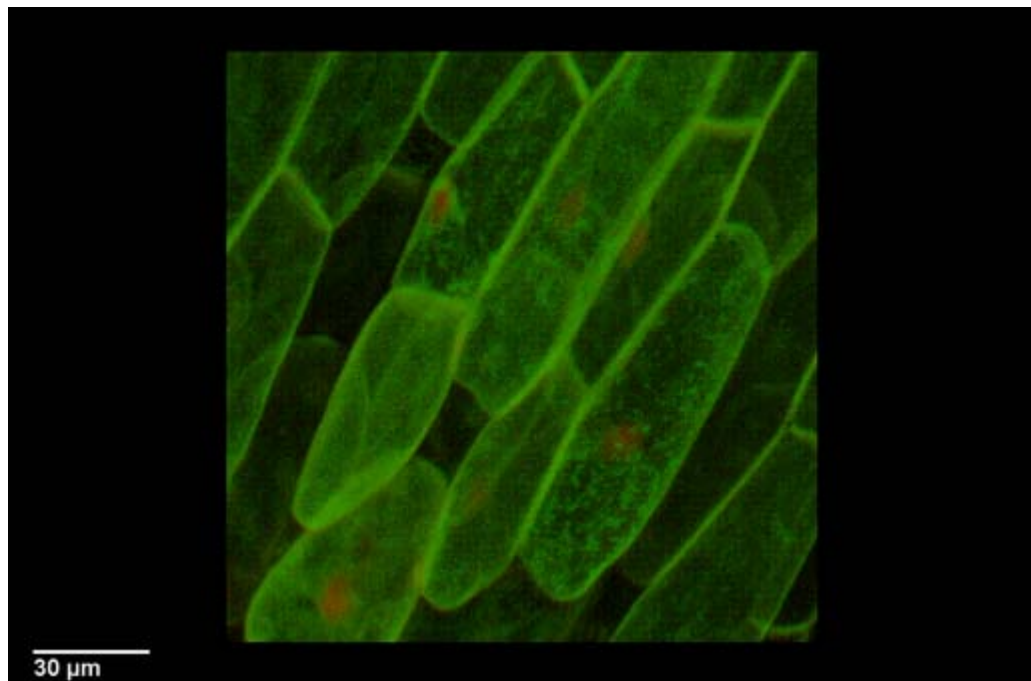
```
*Exercise 6:  
Try adjusting the mapping function to a shape as shown above.
```

You can also toggle the type of mapping you wish to apply. This is selected using the *Mapping:* checkbox group. Where you can specify either to do a linear map or use B-splines to do a curve map.

Lastly you may also choose how the histogram is to be displayed using the *Histogram:* checkbox group. You can toggle a log scale on/off and also specify whether the histogram data should update itself due to the remapping.

```
*Exercise 7:  
Change the mapping to linear.  
Disable the histogram log scale.
```

In the same way we adjusted our data's opacity you can adjust the data intensity. As an exercise by individually adjusting the opacity/intensity of each channel try and reproduce the following projection:



*Exercise 8:
Try and reproduce the above rendering.
HINT: Green low opacity, Red lowish opacity and high intensity

The transfer function settings can be saved and loaded. This is particularly useful if you find a good mapping and wish to apply this to a different dataset. Note that transfer function are saved as files with a .tfn extension. You can also drag and drop these to the render window for easy loading.

*Exercise 9:
Save your transfer function to your desktop.
Load another stack from the time series.
Load your saved transfer function by drag and drop.

Cropping

Another method to remove unwanted regions in our data is to use the cropping tool.

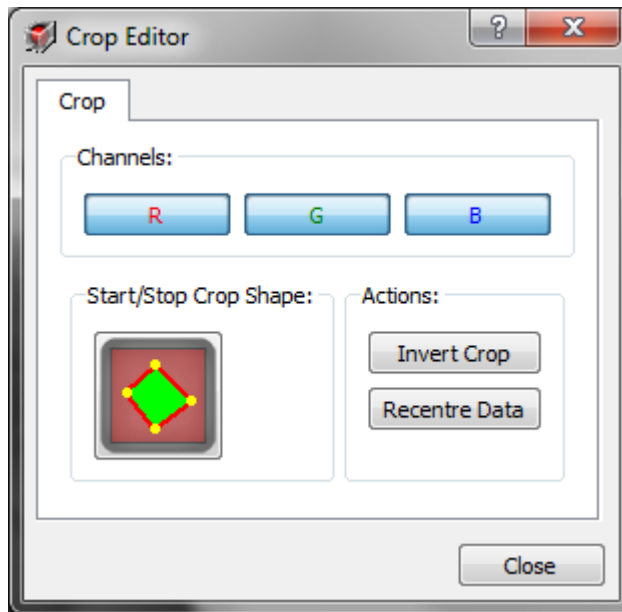
The cropping tool is accessed by pressing the




found on the top toolbar.

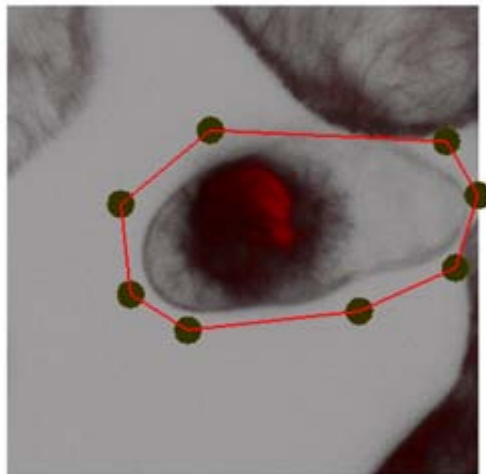
*Exercise 10:
Click on the cropping tool icon


This will bring up the crop tool window as shown below.



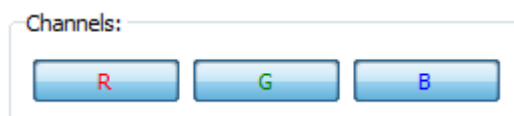
The cropping tool allows you to draw a 2D crop shape which VolViewer will extrude in 3D and crop the data with. To start drawing a 2D crop shape you click on the  icon.

When this is enabled you can click around your region of interest in the render window to produce a 2D shape as shown below.



You can edit/move points by placing your mouse over a point until it becomes highlighted and clicking and dragging it to a new position. To accept the crop you click on the  icon again. If you wish to cancel a crop you click on the cancel button.

Note you can rotate your data to different orientations and repeat the crop process in order to incrementally improve your cropping. Cropping can be applied on a per channel basis by using the *Channels:* button group shown below:



You can also specify an inverted crop, where the region kept/discarded is reversed by enabling the *invert crop* button. And once your data is cropped you may wish to recentre it in the volume view using the *recentre data* button. Both these actions are found in the *Actions:* button group shown below:

Actions:

Invert Crop

Recentre Data

*Exercise 11:

Load a single stack from the Sample Data\Converted\TL01-x63zoomx2-TL5sec-1s5%_lsm\ dataset as an RGBA volume.
Open the cropping tool
Crop out the cell

Exploring your data

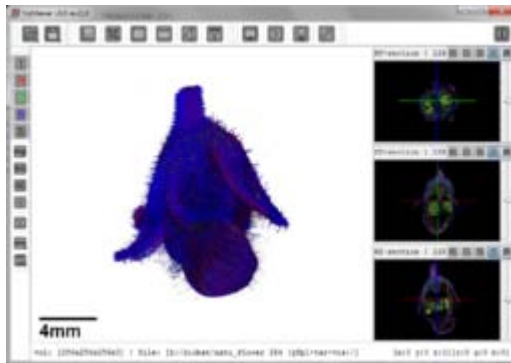
Clipping Planes

Clipping planes allow for interactive cutting of the data. VolViewer allows for up to 6 clipping planes to be moved and rotated inside the volume.

*Exercise 12:

Load the Converted\Anti_Flower 294 (gfpl+txr+vis)\ dataset as an RGBA volume.
Drag and drop the *TransferFunction.tfn* file from the same folder onto the render window

You should then see the following:



To open the clipping plane tool you need to click on the



icon found in the top toolbar.

This will bring up the window shown below.

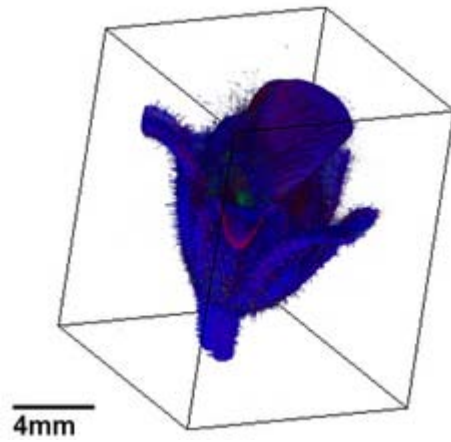


*Exercise 13:

Open the clipping plane tool.

A clipping plane can be selected using one of two methods; using the dropdown menu or by clicking on the render window. Both these options are found in the *Actions:* menu. Clicking on the

select button will display the clipping cube in the render window as shown below:



This cube shows the 6 clipping planes in their default state. You can select any of the planes by simply clicking on one of the cube faces. Once a plane is selected it will be hi-lighted in yellow.

*Exercise 14:
Press the *Select* button in the *Actions:* button group.
Click on one of the cube faces in the render window to select a plane.

You will notice that once a plane is active some additional options in the *Selected Clipping Plane:* group become active. These options allow you to specify how the plane is viewed and how you wish to interact with it.

To start we will leave everything as default and focus on interacting with the plane by using the render window.

Once you have selected a plane, you can move/translate it by clicking anywhere in the centre of the yellow shape and dragging up/down with the left mouse button clicked. You can also rotate the plane around a fixed axis by selecting either the *Rotation AXIS 1* or *Rotation AXIS 2* in the *Interaction Mode:* group.

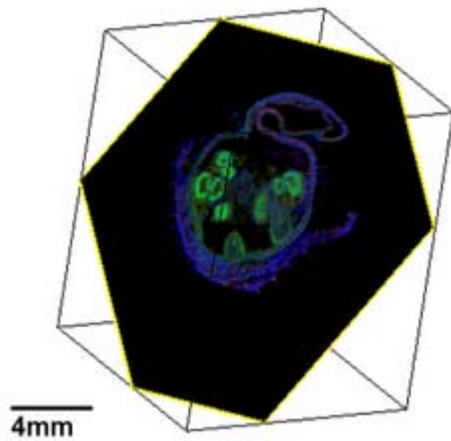
Note that clicking outside of the yellow shape will allow you rotate, zoom and translate your data.

*Exercise 15:
Try translating your plane into the volume.
Try rotating the plane around an axis.

You will notice that the volume is clipped away as you move/rotate your plane giving you an easy way to see the internal parts of your volume. You can also specify how to view the clipping plane. Four viewing modes exist:

- None
- Wireframe
- Solid Texture
- Alpha mapped Texture

By using either the *Solid Texture* or *Alpha mapped Texture* viewing modes you can produce section views in any orientation through the volume.

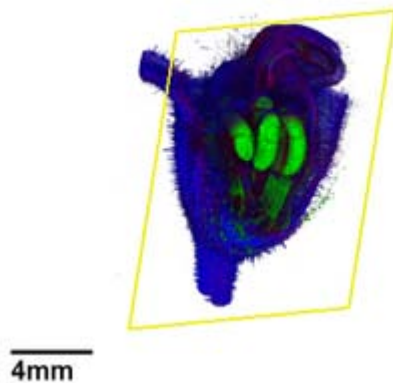


*Exercise 16:
 Try to reproduce the above image.
 Turn off the volume view by pressing the *I* icon in the viewing toolbar.
 Turn on the bounding box by pressing the *BB* icon in the on/off toolbar.
 Select *Solid Texture* as the view mode.
 Move and rotate your clipping plane to any desired orientation.

The clipping can also be applied on a per-channel basis such that one or more channels do not get clipped away.



This is specified by using the toggle buttons in the *Channels:* group.



*Exercise 17:
 Try and reproduce the above rendering.
 HINT: Do not clip the green channel.

The clipping plane settings can be saved and loaded. This is particularly useful if you find a clipping and wish to apply this to a different dataset. Note that clipping planes are saved as files with a *.slc* extension. You can also drag and drop these to the render window for easy loading.

*Exercise 18:
 Save your clipping planes to your desktop.
 Load another stack.
 Load your saved clipping planes by drag and drop.

Visual Bookmarks

It will sometimes be the case that different views of your volume will reveal different areas of interest in the data. To this end VolViewer introduces the notion of Visual Bookmarks.

A visual bookmark will save the following settings of your view:

- View orientation (rotations)

- Zoom level
- Translation
- The transfer function
- The clipping planes

You can save multiple visual bookmarks for a given dataset and also apply a set of visual bookmarks from a different dataset to your current dataset to get corresponding views.

To open the visual bookmarks you need to click on the



icon found on the top toolbar.

This will bring up the following window:



*Exercise 19:
Open the Visual Bookmarks tool

You can then modify any of the previously mentioned parameters and save these as a bookmark by clicking the *Add* button. A bookmark can be updated by selecting it in the list and clicking on the *Update* button. You can also delete a bookmark by pressing the *Delete* button.

*Exercise 20:
Try modifying the transfer function and adding a bookmark.
Now modify the transfer function and use the clipping planes to reveal an internal structure and adding another bookmark.

You can quickly switch between these saved states by double clicking on the bookmark icons.

*Exercise 21:
Try switching between the two different views by double clicking on the bookmark icons.

Visual bookmarks can be loaded and saved. It is sometimes useful to export the visual bookmarks from one dataset and apply these to a different dataset.

This page was last modified on 17 January 2012, at 21:06.

This page has been accessed 225 times.

[Privacy policy](#)

[About DMBI - Data Management for Bio-Imaging](#)

[Disclaimers](#)





VolViewerNotes6

navigation

- [Main Page](#)
- [Recent changes](#)

help

- [Help](#)

search

toolbox

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Printable version](#)
- [Permanent link](#)

Contents [\[hide\]](#)

- 1 [Overview](#)
- 2 [High Quality Projections](#)
 - 2.1 [Lighting](#)
 - 2.1.1 [Photo-Realistic Lighting](#)
 - 2.1.2 [Non-Photo-Realistic Lighting](#)
- 3 [Movies](#)
 - 3.1 [MovieMake](#)

Overview

In this session we will learn how to produce high quality projections and movies.

High Quality Projections


We previously saw how to use the Render Settings user interface to increase the quality of the render window. As previously mentioned when saving a projection it is useful to try and maximise our render quality to produce high quality projections.

In this section we will now focus on an additional tool we have to increase our projection quality.


Lighting

Applying lighting to a volume can help increase the details we are able to perceive. In VolViewer we have 3 lighting modes:


- **No Lighting:** This is the default state.
- **Photo-Realistic Lighting:** This mode tries to create a realistic illumination of the volume
- **Non-Photo-Realistic Lighting:** This mode trades photorealism for a more expressive style.

The lighting window is accessed by clicking on the  icon found in the top toolbar.

*Exercise 1:

Set the *Max Texture Size* in the *Settings Editor*  to 512.

Load the *Datasets\Converted\Ler_PhaseA_63xLens_20xDecon0000* data as an R volume.

Open the *Lighting Editor* .

What you should see is the following window:



The *Shader Programs*: drop down menu will allow you to change between the three different rendering modes.

Photo-Realistic Lighting

*Exercise 2:
Set the *Shader Programs*: to *Photo-Realistic Lighting*

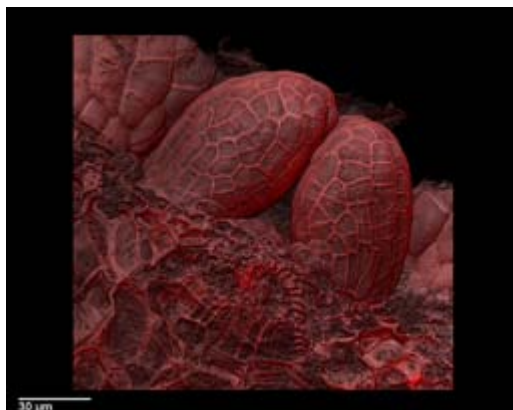
With Photo-Realistic lighting enabled you will notice your data is now illuminated. You can change the position of the light to adjust where the data is illuminated from.

*Exercise 3:
Turn on the light visualisation from the on/off menu on the left toolbar.
Zoom out until you see the light cone.
Adjust the Altitude and Azimuth slider in order to move the light around the scene.

There are 4 colour parameters to control:

- Ambient Colour: A constant colour added to the scene
- Diffuse Colour: The *soft* light that gets reflected in many angles
- Specular Colour: The *sharp* light that gets reflected in one angle.
- Depth Cue Colour: A constant colour that is added to parts of the data which are far away from the viewpoint.

Lastly the *Scale* slider allows you to control when the *Depth Cue* is applied.



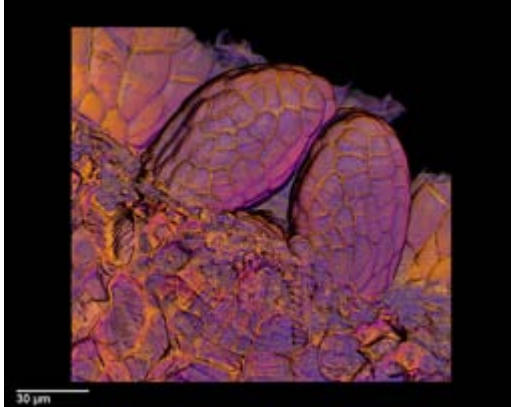
*Exercise 4:
Try to reproduce the above rendering.

Non-Photo-Realistic Lighting

*Exercise 5:
Set the *Shader Programs*: to *Non-Photo-Realistic Lighting*

With Non-Photo-Realistic lighting, the scene is illuminated using a warm and cool colour. These two colours are used to shade the scene according to the light position. Using a warm and cool colour increase surface curvature perception.

Additionally a silhouette shading term is also computed in order to better perceive sharp distinct surface changes. The silhouette term is controlled using the three sliders in the silhouette group.



*Exercise 6:
Try to reproduce the above rendering.

Movies

VolViewer can be used to create movies from your data. The *Movie Editor* can be accessed by

clicking on the



icon.

*Exercise 7:
Load the Sample Data\Converted\Antiriniuhm_Meristem (r512g110usmall) dataset as an RGBA volume.
Open the *Movie Editor*

The following window should appear.



There are four movie modes in VolViewer:

- **Rotation:** This will create a rotation movie and allows you to specify an axis and an angle of rotation.
- **Rock:** This will create a rock movie, where you can specify an axis and rock angle. The data will be rocked by +/- the angle
- **Orthogonal Sections:** This will export the 3 orthogonal sets of sections for your dataset.
- **Visual Bookmarks:** This will allow you to use visual bookmarks as key movie frames. The program will interpolate between these to create a smooth transition between the bookmarks.

The *Choose Output Directory* allows you to specify where the movie frames will be saved to.


The *Frame Step Size* allows you to specify how big a jump you want between movie frames. The smaller the jump the smoother, but this will of course take more computation time.

The *Preview Mode* button allows you toggle whether the frames will be saved or not allowing you to preview your movie.

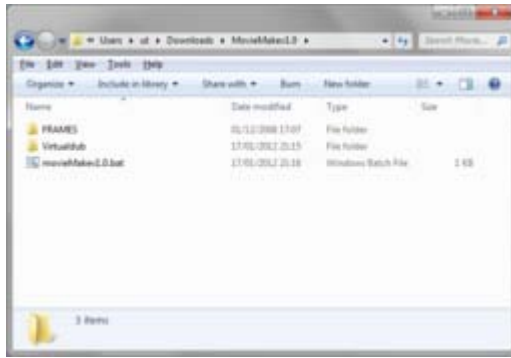
```
*Exercise 8:  
Export a Rotation movie to a folder on your desktop  
Create a couple of bookmarks for you data and export a Visual Bookmarks movie
```

MovieMake

We will now use a simple tool to create movie files out of our movie frames. This tool is called MovieMake.

```
*Exercise 8:  
Download MovieMake from here   
Extract the program to your desktop
```

What you should end up with is a folder as shown below:



To create a movie we copy our movie frames into the folder called *FRAMES*. Then to create the movie we double click on the *movieMakev1.0.bat* file. This will prompt us for the filename of the first frame. Once the program is finished a new file will appear in the folder called *movie.avi*.

```
*Exercise 9:  
Copy your movie frames to the FRAMES folder  
Run movieMakev1.0.bat by double clicking on it  
Enter the first frame filename: 100000.png  
And wait for the process to finish
```

This page was last modified on 17 January 2012, at 21:35.

This page has been accessed 84 times.

[Privacy policy](#)

[About DMBI - Data Management for Bio-Imaging](#)

[Disclaimers](#)





navigation

- [Main Page](#)
- [Recent changes](#)

help

- [Help](#)

search

toolbox

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Printable version](#)
- [Permanent link](#)

VolViewerNotes4

Contents [[hide](#)]

1 Overview

1.1 Filters

- 1.1.1 [Voxel Operators](#)
- 1.1.2 [Channel Operators](#)
- 1.1.3 [Morphological Operators](#)
- 1.1.4 [Filter Operators](#)
- 1.1.5 [Example Workflow](#)

1.2 Quantification

- 1.2.1 [Measuring Editor](#)
 - 1.2.1.1 [The Line Tool](#)
 - 1.2.1.2 [The Mesh Tool](#)

1.3 Segmentation

- 1.3.1 [Smoothed Dilate](#)
- 1.3.2 [Max flow](#)

1.4 Scripts

- 1.4.1 [Scripts API](#)
- 1.4.2 [Executing Scripts](#)
- 1.4.3 [Examples](#)

Overview

In this session we will cover some of the more advanced features of VolViewer. We will cover how to use the, filters, quantification, segmentation and scripting tools.

Filters

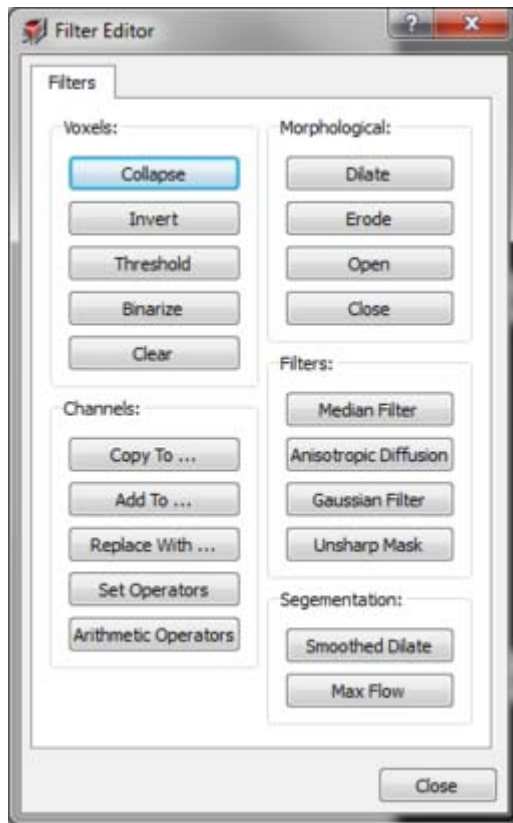
VolViewer has many 3D filters that are useful to process your data with. This can be to remove noise or to extract regions of interest. Below is a quick overview of the different methods available. We will then take an example workflow to demonstrate the combination of using some of these methods.

The filter window is accessed by clicking on the



icon found on the top toolbar.

This will bring up the following window:



A quick overview of the various tools is given below:

Voxel Operators

- Collapse: This will convert a multichannel dataset to greyscale.
- Invert: This will invert the intensity values of a dataset.
- Threshold: This will threshold the data values using the opacity threshold slider value from the transfer functions.
- Binarize: This will convert the data into binary, 0 or 255. 255 will be set for all non-zero voxels. This is useful for creating masks.
- Clear: This allows you to clear a specific data channel.

Channel Operators

- Copy To: This will copy a SOURCE channel to a DESTINATION channel
- Add To: This will append a SOURCE channel to a DESTINATION channel
- Replace with: This allows you to replace a DESTINATION channel with a REPLACE channel only if a CONDITION is met in a SOURCE channel.
- Set Operators: This allows you to compute for two channel A and B the intersection, the difference, the A not B and B not A of two channels.
- Arithmetic Operators: This allows you to add or subtract two channels together

Morphological Operators

- Dilate: This allows you to morphologically expand a channel.
- Erode: This allows you to morphologically shrink a channel.
- Open: This is a dilate followed by an erode, allowing you to close holes in your data.
- Close: This is an erode followed by a dilate, allowing you to open holes in your data.

Filter Operators

- Median Filter: Computes the median value of a voxel given a neighbourhood size. With a small neighbourhood size this is useful to de-speckle your data.
- Anisotropic Diffusion Filter: This filter allows you to remove noise in your images, whilst keeping strong edges.
- Gaussian Filter: This is a blur filter.

- Unsharp Mask: This filter will enhance the edges in your data.

Example Workflow

In this example we will try and remove noise from the green channel of a multi-channel dataset, giving you a feel for what is possible using the filter tools.

We will start by loading our data.

```
*Exercise 1:  
Load a single stack from the Sample  
Data\Converted\movie_Plant2_z_stack_GFP_CFP_20_min_03_lsm\ dataset as an RGBA volume.
```

Next we will make a copy of the green channel into the red channel. We will also turn off the red channel for now.

```
*Exercise 2:  
Use the Copy To tool to copy the green channel to the red channel.  
Turn off visibility of the red channel using the toolbar on the left.
```

Next we will apply some filtering to our green channel to remove noise.

```
*Exercise 3:  
Apply a Median filter of kernel width 1 to the GREEN channel.  
Apply an Anisotropic Diffusion filter of lambda 1.0 dt 0.5 iterations 10 to the GREEN  
channel.  
Set the opacity threshold of the green channel in the transfer functions to 15.  
Apply the Threshold filter tool to the green channel.
```

Now when looking at the green channel you should see that although we have removed noise, our signal appears blurred and we have lost detail. However the cell shapes/boundaries are much clearer.

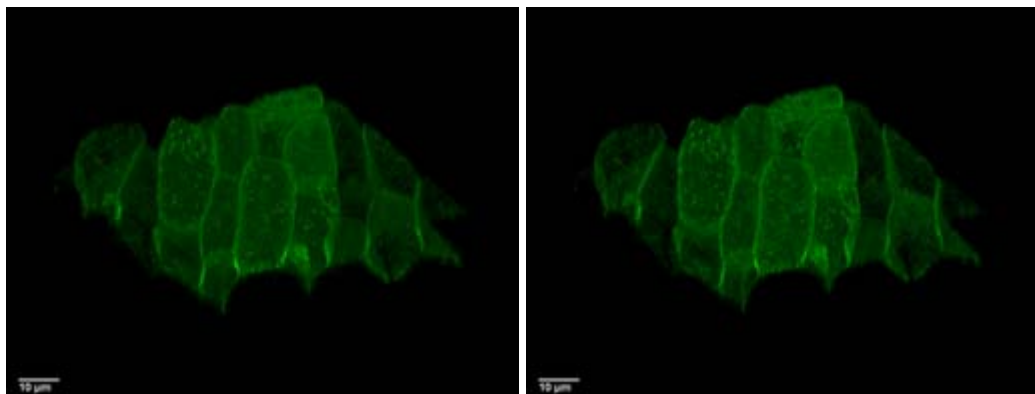
We can now use this green channel as a mask for our original data which we kept in the red channel allowing us to keep the original detail but improve the cell shape/boundaries.

```
*Exercise 4:  
Apply the Set Operators using the intersection mode to our red and green data.
```

The resulting intersection will be copied to the blue channel. We can now copy back the blue channel to the green channel and clear the red and blue channels to obtain a filtered version of the original green channel data.

```
*Exercise 5:  
Use the Copy To tool to copy the blue channel to the green channel.  
Use the Clear tool to clear the blue and green channels.
```

For comparison, below is the unfiltered (left) vs filtered image.



```
*Exercise 6:  
Use some what you learnt above to try and improve the red channel.
```

Quantification

VolViewer supports quantification by using the measuring tool.

Measuring Editor

Before we start measuring we will load some data and it will be useful to clean it up slightly.


```
*Exercise 7:  
Load the Sample Data\Converted\PD3_3_15_48\ dataset as an RGBA volume.  
Apply an Anisotropic Diffusion filter of lambda 1.0 dt 0.75 iterations 10 to the volume.  
Set the opacity threshold to 15 in the transfer function.  
Set the opacity contrast to 120% in the transfer function.
```

The measuring tool is accessed by clicking on the *Measuring Editor*

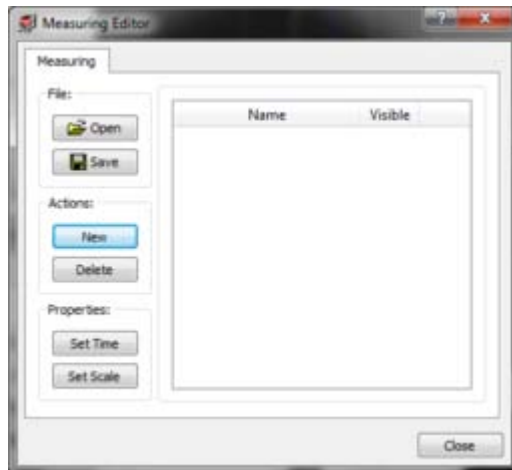


icon found on the

top toolbar.

```
*Exercise 8:  
Click on the  icon to bring up the Measuring Editor
```

Clicking on the icon will bring up the *Measuring Editor* window as shown below:






You can create a new measure object by clicking on the *New* button.

```
*Exercise 9:  
Click on the New button.
```

This will bring up the *Measure Object* window as shown below.




The *Measure Object* supports 3 different measuring modes, you can create measure objects using the:

- Points Tool: 
- Lines Tool: 
- Mesh Tool: 


To facilitate editing and creation each measure object is further divided into 3 main components;

- Vertices: This represents a 3d point/position in your data.
- Edges: This is a set of 2 connected vertices.
- Faces: This is a collection of vertices and edges which form a closed polygon.

The Line Tool

To start simple we will focus on the Lines Tool .

*Exercise 10:


Select the Lines Tool  measuring mode by clicking on it.


You will notice that when you clicked the line tool and as a line is defined by vertices and edges, the edges tab become available.

You can add or remove items in the measure components using the following icons:


- Add 
- Remove 

*Exercise 10:

Enable the adding of vertices by clicking on the add  icon.
Now click on two points of your scan to create a line.

You will notice that when you are adding points the application cursor will change to a cross-hair. When you are happy with what you have captured you can stop adding points by clicking again on the Add  icon. Once this is done you can not add any more points, until you press the Add icon again, but you are able to move/edit any existing points.

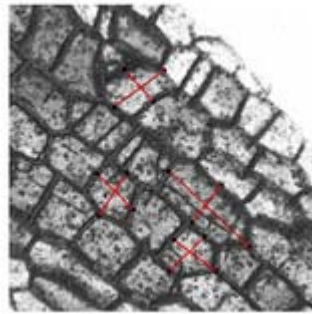
*Exercise 11:
Disable the Adding of points.
Try moving/editing an existing point.

Points can be deleted by selecting them in the list and pressing the Remove  icon. You can also highlight a point in the render window by moving your mouse over it, it will automatically get selected in the list.


*Exercise 12:
Try deleting some points.

When you are happy with the object you have created, pressing the *OK* button will add it to the *Measuring Editor* list. Pressing *Cancel* will discard the current object and return you to the *Measuring Editor* tool.

*Exercise 13:
Use the Line Tool to measure the width and length of a couple cells as shown below.





Basic information about the vertices, edges and faces is shown in the *Selected* and *Info* panels of the corresponding tabs. Notice how in the vertices in the *Selected* panel you can see the vertex position in X, Y, Z coordinates as well as the vertex colour information as R, G, B values. Selecting a different vertex in the list will update this information. In the *Info* panel you will see a summary of the whole collection, in the case of vertices we can see the vertex count.

You can export this information for vertices, edges and faces directly to Excel using the Excel spreadsheet  icon.

*Exercise 14:
Click on the edges tab.
And try exporting the data to your desktop.

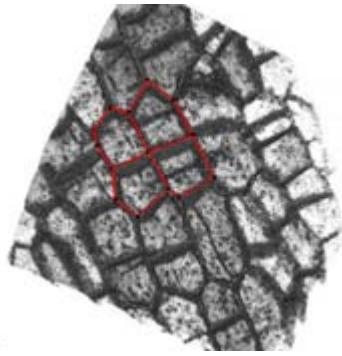
The Mesh Tool

The next measuring tool we will focus on is the *Mesh Tool* .

*Exercise 15:
Create a new measuring object by pressing *New* in the *Measure Editor*
Select the *Mesh Tool* .

Since a mesh object is made up of vertices, edges and faces you will notice how these tabs now become enabled.

*Exercise 16:
Use the Add button in the vertices tab to create cell outlines as shown below.



When you wish to close a mesh, you can do so by moving the cursor to an existing vertex which will become high-lighted.

In the edge tab we have an additional tool to help us capture 3D shapes. This tool is the *Edge Clip-Plane*. The edge clip plane allows you to set a clipping plane aligned to an existing edge.


*Exercise 18:
Set the current view to TOP.
Select the edges tab in the Measure Object window.
Double click on an edge in the Render window.
Now rotate the volume to examine the clipped data.

Double clicking on an edge will place a clipping plane defined by the edge you clicked on and the current viewing direction. This can be useful to reveal the internal parts you wish to measure.

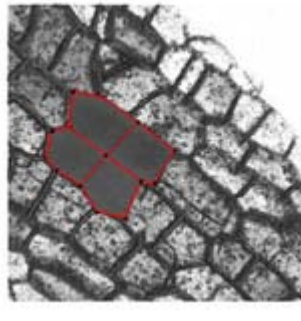
*Exercise 19:
Try capturing some of the lateral wall shapes by using the *Edge Clip-Plane*.

You can reset the clipping plane by pressing the *Reset Planes* button. You also have the option to display the clipping plane as a solid object. And you can also flip the clipping direction.

We will now focus how to add faces to the mesh.

Faces are added by pressing the Add  icon. Vertices can then be selected by hovering over them in the render window. As you hover over the vertices you wish to add to the face the edges will also be highlighted. Note you need to add your vertices in an order (clockwise/anticlockwise) manner to produce a valid face. To remove a vertex from the current face you hover over it again and it will be removed. Once you are happy with the current face, pressing the Add icon will add the face to the face list.


*Exercise 20:
Select the face tab.
Create the faces for your captured geometry.



Segmentation

We will now look at two different segmentation methods that are available with VolViewer. Both of these tools are available from the filter window we previously looked at.

*Exercise 21:
Load the *Sample Data\Converted\PD3_3_15_48* dataset as a Red volume.

Open the filter window by clicking on the  icon found on the top toolbar.

The two tools we will now go over the *Smoothed Dilate* and the *Max Flow* segmentation methods.

Smoothed Dilate

The *Smoothed Dilate* method is a region growing algorithm that incorporates a smoothing step. It requires a seed point or start position to grow from. This segmentation method is very good at finding a region that has roughly a uniform intensity value and is surrounded by signal of a higher intensity. We will use this to segment a single cell from our stack by filing the inside of the cell.

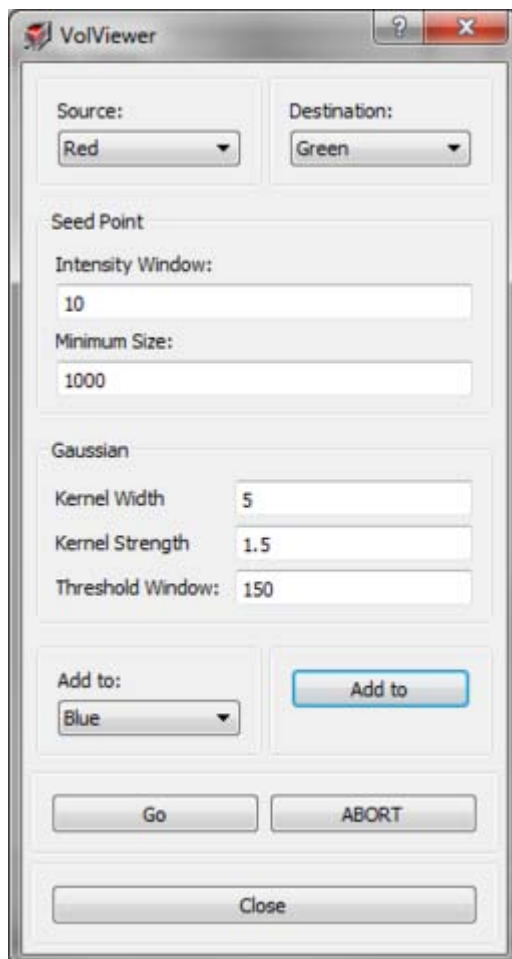
The first step is to setup our seed point as the centre of a cell of interest.

*Exercise 22:
Select a seed point by clicking in the centre of a cell.
Make sure you are in the centre by adjusting the position whilst looking at the other two orthogonal views.

Now that we have a seed point we can open the *Smoothed Dilate* tool by pressing the corresponding button on the *Filter* window.

*Exercise 23:
Open the *Smoothed Dilate* tool.

Once the tool is open you will see the following window:



The tool has 5 parameters that you can control.

- Intensity Window: This is the intensity range of voxels you allow to be in the region of interest. Increase this value to consider more voxels.
- Minimum Size: This is the size we allow our region to grow before we start smoothing.
- Kernel Width & Strength: These parameters control the smoothing step of our algorithm. Increase these values to increase the smoothing.

- **Threshold Window:** This value is the threshold we apply to the smoothed region. Increase this value to increase effect of the smoothing.

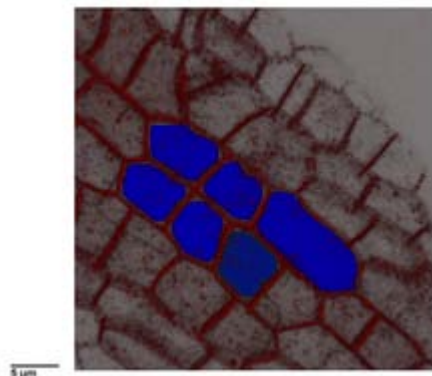
For simplicity we will leave the parameters at their default values. To start the segmentation press the *GO* button. The section views will get update in real time giving you feedback on whether the segmentation is producing the desired result. The segmentation can be stopped at any point by clicking on the *ABORT* button.

Once the segmentation is finished you will see your region appear in the Destination channel.

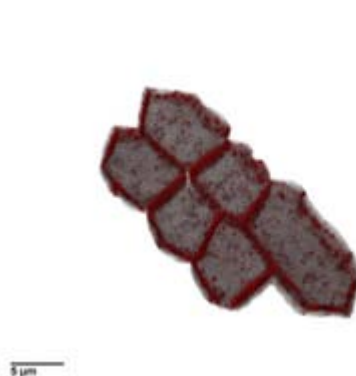
*Exercise 23:
Run the *Smoothed Dilate* tool by pressing the *GO* button.

The result can be added to another channel such as we that we can accumulate the results of many segmentations using the *Add to* button.

*Exercise 24:
Try running the *Smoothed Dilate* on a few cells and appending them to a channel to produce a similar figure as shown below.



*Exercise 25:
From the computed mask above, try producing the figure below.
HINT: Dilate the computed mask, compute the intersection between the mask and original data.




Max flow

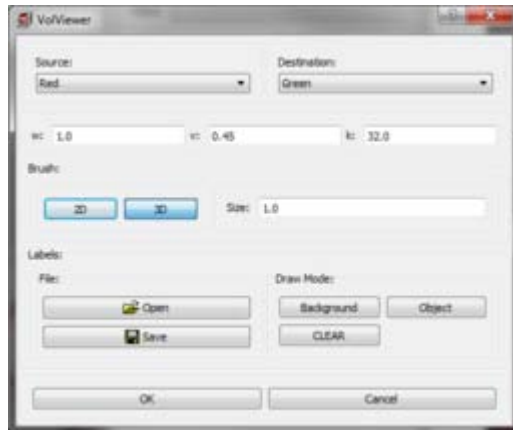
The next segmentation tool we will focus on is the *Max Flow* segmentation method. The *Max Flow* segmentation method is a graph based method based on the minimum cut algorithm. The algorithm requires the user to label the data into two categories, object and background. These are then used to classify all remaining voxels in the data into one of these two categories. It is a general purpose algorithm at the cost of computation complexity.

For the purpose of this course we will focus again on trying to segment a cell from our data.

*Exercise 26:
Load the *Sample Data\Converted\PD3_3_15_48* dataset as a Red volume.
Set the section cursor in the centre of a cell.
Make sure you are in the centre by adjusting the position whilst looking at the other two orthogonal views.

Open the filter window by clicking on the  icon found on the top toolbar.
Open the *Max Flow* tool.

Clicking on the *Max Flow* tool button will bring up the following window.



The first step is to create the set of labels. This is achieved by painting on the section views. We can select whether we paint object or background voxels by pressing on the *Background* or *Object* buttons in the *Draw Mode* panel. You can clear what you have painted using the *CLEAR* button. A brush size can also be specified in the *Brush* panel using the *Size* box.

Note that the bigger the brush you specify the better the regions will be labelled, but note that by default the brush is set to paint in 3D so be careful not to paint outside the regions of interest.

■ Exercise 27:

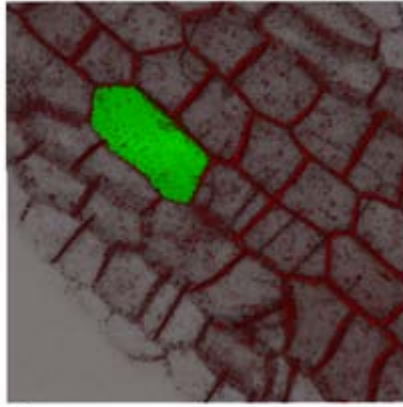
Set the brush size to 8
Select the *Object* button and start painting the parts of your object in the section views.
Select the *Background* button and start painting the parts which are your background.
Try and paint something similar to what is shown below.



Once you have labelled your regions, you can press the *OK* button which will start the algorithm. Once the algorithm is finished the segmented region will appear in the *Destination* channel.

Labels can be imported and exported using the *Open* and *Save* buttons. This is useful to setup your labels but then compute the segmentation at a later time, for example on a more powerful computer.

The main parameter of the algorithm is the *v* parameter which allows you to specify a tradeoff between splitting your data along strong edges or similar intensity values. The parameter ranges from 0.0 to 1.0 with 1.0 favouring strong edges.



Scripts

VolViewer scripts allow you to have greater freedom on how to work with VolViewer. Many tasks can be automated. To see the full details of all the simplified access functions that are available see the Scripts API below.

Scripts API

- [VolViewer Scripts API Documentation](#)

Executing Scripts

Scripts can be called in two ways. Using the *open_script* function or setting up a *watchfile*

- *open_script*

The *open_script* function is parsed to the VolViewer executable as a parameter as shown below:

```
VolViewer.exe open_script('myscript.txt')
```

When you run this command, VolViewer will open and execute the commands you supply in the *myscript.txt* file

- *open_watchfile*

The second method allows you to setup a watch file. A watch file is a file that VolViewer will watch and will execute everytime the contents of this file changes. This is useful to debug/write scripts and also provides a useful way for external programs to interface with VolViewer. Setting up a watch file is shown below:

```
VolViewer open_watchfile('myscript.txt')
```

Now whenever the contents of the file *myscript.txt* changes, VolViewer will execute it's contents.

Examples

Below is a list of examples you should try to learn how to use the scripting features.

- Open an Image, Filter it, Save a projection, Save the processed stack

```
open_image_stack(0, 'E:\VOLUME DATA\Anti_Flower 294 (gfp1+txr+vis)\')
compute_GPUfilter_AnisotropicDiffusion(0, 5.0, 0.1, 10)
save_projection('E:\VOLUME DATA\Projections\Anti_Flower 294 (gfp1+txr+vis).png')
save_image_stack('E:\VOLUME DATA\Anti_Flower 294 (gfp1+txr+vis)\Filtered\')
```

- Open all Images in a folder, Filter them, Save projections, Save the processed stacks

```
open_images_folder(0, 'E:\VOLUME DATA\Confocal Timepoints\')
compute_GPUfilter_AnisotropicDiffusion(0, 5.0, 0.1, 10)
save_projection('E:\VOLUME DATA\Confocal Timepoints\Projections\##ORIGINALNAME##.png')
save_image_stack('E:\VOLUME DATA\Confocal Timepoints\Processed\##ORIGINALNAME##\')
```

- Open an image, load an MSR shape template, wait for user to do something, save MSR at quit

```
open_image_stack(0, 'E:\VOLUME DATA\ScanID1934_Leaf1\')
open_msr('E:\ShapeModel\Shapes\Template.msr')
wait_for_quit()
save_msr('E:\ShapeModel\Shapes\ScanID1934_Leaf1.msr')
```

- Open first time point, wait for user to create visual bookmarks, project all time points using visual bookmarks

```
open_image_stack(0, 'E:\VOLUME DATA\Timelapse\T0\')
wait_for_quit()
save_vvr('E:\VOLUME DATA\Timelapse\Bookmarks\Bookmarks.vvr')
set_hide()
open_images_folder(0, 'E:\VOLUME DATA\Timelapse\')
open_vvr('E:\VOLUME DATA\Timelapse\Bookmarks\Bookmarks.vvr')
set_projection_mode('SUM')
set_current_visual_bookmark(0)
save_projection('E:\VOLUME DATA\Timelapse\Projections\view0_##ORIGINALNAME##.png')
set_current_visual_bookmark(1)
save_projection('E:\VOLUME DATA\Timelapse\Projections\view1_##ORIGINALNAME##.png')
set_current_visual_bookmark(2)
save_projection('E:\VOLUME DATA\Timelapse\Projections\view2_##ORIGINALNAME##.png')
```

This page was last modified on 19 January 2012, at 12:37.

This page has been accessed 246 times.

[Privacy policy](#)



[About DMBI - Data Management for Bio-Imaging](#)

[Disclaimers](#)

VolViewer Scripting in more detail

[Back to VolViewer overview](#)

Contents [\[hide\]](#)

- [1 See also](#)
- [2 VolViewer scripting and accessing VolViewer from Matlab](#)
 - [2.1 VolViewer scripting](#)
 - [2.2 Matlab to Volviewer](#)
 - [2.2.1 Accessing VolViewer from Matlab](#)

See also

[MSR file specification](#) [↗](#) MSR file specification (MSR stands for *Measure*.) MSR files describe 3D objects and are used to pass data between the 3D BanghamLab toolboxes: VolViewer, GFtbox and AAMToolbox.

[VolViewerScriptsAPI](#) [↗](#) The VolViewerScriptsAPI enables external applications, such as the AAMToolbox, to exploit VolViewer.

- ▶ [List of VolViewer commands](#) [↗](#)
- ▶ [Table of VolViewer commands by function](#)
- ▶ [VolViewer saves spatial data in msr files](#)

VolViewer scripting and accessing VolViewer from Matlab

VolViewer scripting

A script is a limited form of program - a list of commands. In this case the commands (<http://dmbi.nbi.bbsrc.ac.uk/index.php/VolViewerScriptsAPI> [↗](#) see VolViewerScriptsAPI) cover most of the operations that you would usually do interactively. On Windows, VolViewer is an exe file that can *either* be **interactive**, i.e. launched by clicking its icon or from the command prompt using the command

```
VolViewer.exe
```

or **scripted**. In which case the operations (commands) are stored in a script file. In which case VolViewer is launched with

```
VolViewer set_watchfile('file.txt')
```

where *file.txt* contains the commands. For example, if VolViewer is to be a viewing tool for an external program, e.g. a Matlab program, then the file.txt could contain

```
open_image_stack(0, 'E:\VOLUME DATA\ScanID1934_Leaf1\')
```

where *ScanID1934_Leaf1* is a directory containing a stack of images. VolViewer will automatically load and display these images as soon as it finds that the watchfile has been updated.

[Other examples](#) [↗](#)

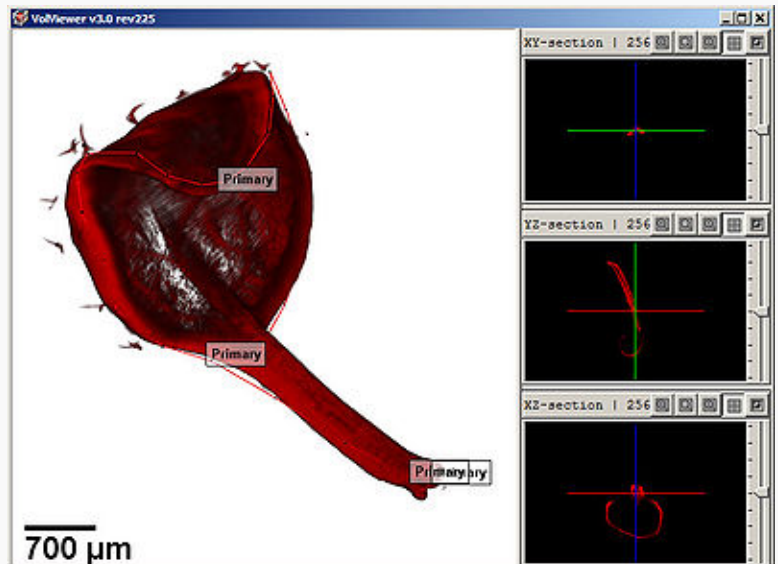
It is **not practical to script** VolViewer using these commands manually. It is better to **use VolViewer as a tool** that is controlled through a library of commands in, for example, Matlab. Or, when running VolViewer on the server side of a server/client session where the user is interacting with VolViewer through a client web page.

Matlab to Volviewer

The DART_Toolbox contains a [library of functions](#) for using and controlling VolViewer from Matlab.

Accessing VolViewer from Matlab

There is a VolViewer Matlab class library: launching and controlling Volviewer for the display of 3D volume images (the display window has zoom, rotate, etc.). On computers with an appropriate graphics card and stereo display VolViewer can be used to display volume images and graphics objects from msr files.



VolViewer viewport without its usual menu-bars and under control of a Matlab program. It shows endogenous fluorescence signal from a young leaf that has been annotated with a series of points set on the mid point of trichomes and joined by lines. These points characterise the 3D shape of the leaf and are stored in an MSR text file. To manipulate the view: left-click-drag **rotate**, middle-click-drag **translate**, right-click-drag **zoom**.

This image was created using steps 2, 3, and 4 below.

1) Install VolViewer

- ▶ For example, in 'C:\VolViewer_x64'. This is the **PathToVolViewer**.

2) Launch VolViewer from Matlab

- ▶ Temporary data files on paths are saved in '...\DARt_Toolshed\ToolBag\VolViewerAPI\CLASSFILES'

This should mean that once VolViewer is installed on a computer - the system will remember where to find everything without having to respecify paths.

```
VO=VolViewer(1,'D:\VolViewer_x64','D:\VolViewer_x64')
```

creates a VolViewer object - which, because there is no semicolon at the end, is displayed as:

```
PathToVolViewer: D:\VolViewer_x64
PathToWatchFile: D:\VolViewer_x64
WatchFileNumber: 1.00000
  PathTo:
  imageFileNames:
  msrFilename:
  tfnFilename:
  slcFilename:
  vvrFilename:
  objFilename:
  Commands:
```

Notice that the paths have been filled in and that this is the first copy of VolViewer (WatchFileNumber is 1).

3) Display an image stack in VolViewer

- ▶ The key step is to notify VolViewer where to find the images by putting a command into WatchFile1.txt. This is done automatically with the following

```
VO=imshowVOL(VO, 'PathTo','D:\Shape_T1_only\Image_data\11to14DFS\1554')
```

In this case the image stack is in the directory 'D:\Shape_T1_only\Image_data\11to14DFS\1554'. Notice, that VOL has been updated

```
>> VOL
PathToVolViewer: D:\VolViewer_x64
PathToWatchFile: D:\VolViewer_x64
WatchFileNumber: 1.00000
PathTo: D:\Shape_T1_only\Image_data\11to14DFS\1554
imageFileNames:
```

```
msrFilename:  
tfnFilename:  
slcFilename:  
vvrFilename:  
objFilename:  
Commands:
```

4) Display an MSR (results: points/lines/planes) file

- ▶ Put the command into WatchFile1.txt using

```
VO=imshowVOL(VO, 'PathTo', 'D:\Shape_T1_only\Image_data\11to14DFS\1554', 'msrFilename', '1554_T1_Shape.msr')
```

To hide the image

```
VO=commandsVOL(VO, 'set_channels(0,0,0)');
```

To see the image again

```
VO=commandsVOL(VO, 'set_channels(1,1,1)');
```

To clear the MSR points/lines/planes

```
VO=commandsVOL(VO, 'set_clear_all_MSR()')
```

modified on 11 April 2012 at 13:19 *** 223 views

University of East Anglia / School of Computing Science / Powered by MediaWiki

VolViewer commands by function

[Return to VolViewer Scripting in more detail](#)

[Full list of VolViewerScriptsAPI](#)

Selection of VolViewer commands

The Matlab 'VolViewer' class library provides *commandsVOL* which will pass commands to VolViewer, e.g. in Matlab

```
VO=commandsVOL(VO,'set_clear_all_MSR()')
```

VO is a VolViewer object created by Matlab when it launches VolViewer with

```
VO=VolViewer(1,'D:\VolViewer_x64','D:\VolViewer_x64')
```

where 'D:\VolViewer_x64' is the path to both VolViewer itself and the location of a 'WatchFile.txt' into which commands are written.

| | | |
|------------------|---|---|
| Image | <pre>open_image_stack(int channel, 'somepath\slice000.png') save_image_stack('somepath') open_image_raw(int channel, 'somepath\myfile.raw') save_image_raw('somepath\file.dat') open_omero_image('imageid') (in preparation) save_omero_image('name', 'description') (in preparation) -----</pre> | 0 all, 1 red, 2 green, 3 blue, directory of image slices |
| Objects | <pre>open_msr('somepath\file.msr') save_msr('somepath\file.msr') set_clear_all_MSR() open_slc('somepath\file.slc') save_tfn('somepath\file.slc') open_tfn('somepath\file.tfn') save_tfn('somepath\file.tfn') -----</pre> | <p>points, lines and facets placed around images</p> <p>clears the points, lines and facets</p> <p>clipping planes</p> <p>transfer function, i.e. brightness/contrast/thresholds</p> |
| Viewing | <pre>set_orientation(float angleX, float angleY, float angleZ) set_projection_mode('mode') open_slc('somepath\file.slc') set_channels(int red, int green, int blue) set_zoom(float zoomamount) set_shader(int shader_mode) -----</pre> | <p>set the current view based on the three Euler floating point angles specified in degrees.</p> <p>projection rendering mode based on the mode text. The function supports, MIP and SUM projection modes</p> <p>clipping planes</p> <p>0 sets colour channel off, 1 turns it on. 0,0,0 turns off all channels, i.e. hides the image</p> <p>percentage</p> <p>0: No Lighting, 1: Photorealistic Lighting, 2: Non-Photorealistic Lighting, 3: Depth Peel (also see, depthpeal, depthcue, gooch_lighting, light_colour,</p> |
| Interface | <pre>set_watchfile('file.txt') set_logfile('log.txt') set_close() wait_for_quit() set_hide() set_show()</pre> | <p>sets the VolViewer to watch a specific ascii file: automatically installed by Matlab functions</p> <p>set the debug output to a text file. Note under windows this will also hide the application console if it exists</p> <p>close VolViewer</p> <p>queues subsequent commands which are then executed when the user closes the application.</p> <p>hide VolViewer</p> <p>show VolViewer</p> |

set_hide_menubars()
set_show_menubars()
set_hide_orthosections()
set_show_orthosections()

hide menubars
show menubars
hide orthosections
show orthosections

modified on 1 March 2012 at 18:37 ••• 26 views

University of East Anglia / School of Computing Science / Powered by MediaWiki



VolViewerScriptsAPI

Contents [\[hide\]](#)

- 1 Simplified script access functions for VolViewer
- 2 Variables
- 3 Keywords
- 4 Functions
- 5 Open Functions
 - 5.1 open_image_stack
 - 5.2 open_image_raw
 - 5.3 open_images_folder
 - 5.4 open_msr
 - 5.5 open_slc
 - 5.6 open_tfn
 - 5.7 open_vrv
 - 5.8 open_obj
 - 5.9 open_ini
 - 5.10 open_omero_connection
 - 5.11 open_omero_image
 - 5.12 open_omero_project
 - 5.13 open_omero_dataset
- 6 Close
 - 6.1 close_omero_connection
- 7 Save Functions
 - 7.1 save_image_stack
 - 7.2 save_image_raw
 - 7.3 save_msr
 - 7.4 save_tfn
 - 7.5 save_vrv
 - 7.6 save_obj
 - 7.7 save_slc
 - 7.8 save_projection
 - 7.9 save_text
 - 7.10 save_omero_image
- 8 Set Functions
 - 8.1 set_watchfile
 - 8.2 set_logfile
 - 8.3 set_orientation
 - 8.4 set_projection_mode
 - 8.5 set_close
 - 8.6 set_hide
 - 8.7 set_hide_menubars
 - 8.8 set_hide_orthosections
 - 8.9 set_show
 - 8.10 set_show_menubars
 - 8.11 set_show_orthosections
 - 8.12 set_current_visual_bookmark
 - 8.13 set_channels
 - 8.14 set_zoom
 - 8.15 set_shader
 - 8.16 set_depthpeal

navigation

- [Main Page](#)
- [Recent changes](#)

help

- [Help](#)

search

toolbox

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Printable version](#)
- [Permanent link](#)

| | |
|---------|--|
| 8.17 | set_depthcue |
| 8.18 | set_gooch_lighting |
| 8.19 | set_light_colour |
| 8.20 | set_background_colour |
| 8.21 | set_view |
| 8.22 | set_clear_all_MSR |
| 9 | Movie Functions |
| 9.1 | movie_rock |
| 9.2 | movie_rotate |
| 9.3 | movie_orthosections |
| 9.4 | movie_visualbookmarks |
| 10 | Compute Functions |
| 10.1 | compute_filter |
| 10.1.1 | compute_filter_Collapse |
| 10.1.2 | compute_filter_Invert |
| 10.1.3 | compute_filter_Threshold |
| 10.1.4 | compute_filter_Binarize |
| 10.1.5 | compute_filter_Clear |
| 10.1.6 | compute_filter_CopyTo |
| 10.1.7 | compute_filter_AddTo |
| 10.1.8 | compute_filter_ReplaceWith |
| 10.1.9 | compute_filter_SetOperator |
| 10.1.10 | compute_filter_ArithmeticOperator |
| 10.1.11 | compute_filter_Dilate |
| 10.1.12 | compute_filter_Erode |
| 10.1.13 | compute_filter_Open |
| 10.1.14 | compute_filter_Close |
| 10.1.15 | compute_filter_Median |
| 10.1.16 | compute_filter_AnisotropicDiffusion |
| 10.1.17 | compute_filter_Bilateral |
| 10.1.18 | compute_filter_Gaussian |
| 10.1.19 | compute_filter_UnsharpMask |
| 10.1.20 | compute_filter_Sobel |
| 10.1.21 | compute_filter_Sieve |
| 10.1.22 | compute_filter_FloodFill |
| 10.1.23 | compute_filter_SmoothedDilate |
| 10.1.24 | compute_filter_MaxFlow |
| 10.2 | compute_GPUfilter |
| 10.2.1 | compute_GPUfilter_AnisotropicDiffusion |
| 10.2.2 | compute_GPUfilter_Bilateral |
| 10.2.3 | compute_GPUfilter_Gaussian |
| 10.2.4 | compute_GPUfilter_Median |
| 10.2.5 | compute_GPUfilter_Dilate |
| 10.2.6 | compute_GPUfilter_Erode |
| 10.2.7 | compute_GPUfilter_SmoothedDilate |
| 10.3 | compute_isosurface |
| 11 | Wait Functions |
| 11.1 | wait_for_opengl |
| 11.2 | wait_for_quit |
| 12 | Examples |
| 12.1 | Example 1 |
| 12.2 | Example 2 |
| 12.3 | Example 3 |
| 12.4 | Example 4 |
| 12.5 | Example 5 |

- [12.6 Example 6](#)
- [12.7 Example 7](#)
- [12.8 Example 8](#)
- [12.9 Example 9](#)
- [12.10 Example 10](#)
- [12.11 Example 11](#)

Simplified script access functions for VolViewer

VolViewer can be controlled externally through a 'WatchFile.txt'. On launch VolViewer can be pointed to the WatchFile.txt. VolViewer will then monitor the file and execute any commands it contains. There is a VolViewer Matlab class library.

- [Selection of commands by function](#)
- [Scripting from Matlab](#)

Variables

The script interface supports 4 types of variables that are parsed as function parameters.

- int integers (whole numbers ie: 1,2,3,4)
- float floating point (numbers with decimal places ie: 2.4, 1.222, 0.55)
- bool booleans (TRUE or FALSE)
- strings an alpha-numeric sequence of characters (ie: 'my string', 'afile.txt', 'c:\data\file.txt')

Keywords

The scripting interface supports key words which are available when using the open_image_folder command.

- ##ORIGINALNAME##

This is a string containing the loaded file/folder name

- ##ORIGINALPATH##

This is a string containing the full loaded file/folder path

See [example 2](#), [example 4](#), [example 5](#) and [example 6](#) for usage.

Functions

Open Functions

open_image_stack

```
open_image_stack(int channel, 'somepath\slice000.png')
```

Opens an image stack, takes any of the stack files as argument. Supports PNG, BMP, TIFF, JPG. The channel parameter specifies how to load the data. The supported modes are:

- 0 (red)
- 1 (green)
- 2 (blue)
- 3 (rgb/greyscale)

open_image_raw

```
open_image_raw(int channel, 'somepath\myfile.raw')
```


Opens a raw binary volume, takes the file path to a .dat or .raw as argument. The channel parameter specifies how to load the data. The supported modes are:

- 0 (red)
- 1 (green)
- 2 (blue)
- 3 (rgb/greyscale)

open_images_folder

```
open_images_folder(int channel, 'somepath\', bool recursive)
```

Opens a folder containing multiple image stacks. The supported channel modes are:

- 0 (red)
- 1 (green)
- 2 (blue)
- 3 (rgb/greyscale)

The *recursive* parameter can be set to TRUE or FALSE to allow for recursive traversal of folder and sub-folders.

open_msr

```
open_msr('somepath\file.msr')
```

Opens an MSR file (Measure Objects).

open_slc

```
open_slc('somepath\file.slc')
```

Opens an SLC file. (Clipping Plane Objects).

open_tfn

```
open_tfn('somepath\file.tfn')
```

Opens a TFN file. (Transfer Function Objects).

open_vrv

```
open_vrv('somepath\file.vrv')
```

Opens a VRV file. (Visual Bookmark Objects).

open_obj

```
open_obj('somepath\file.obj')
```

Opens an OBJ file. (Mesh Object).

open_ini

```
open_ini('somepath\settings.ini')
```

Loads an INI file, Settings.ini object.

open_omero_connection

```
open_omero_connection('server.ac.uk', int port, 'username', 'password')
```

Establishes an OMERO connection using a new session.

```
open_omero_connection('server.ac.uk', int port, 'sessionid')
```

Establishes an OMERO connection using an existing session.

open_omero_image

```
open_omero_image(int imageid)
```

Load an OMERO image.

open_omero_project

```
open_omero_project(int projectid)
```

Load all OMERO images in a given Project.

open_omero_dataset

```
open_omero_dataset(int datasetid)
```

Load all OMERO images in a given Dataset.

Close

close_omero_connection

```
close_omero_connection()
```

This will close an OMERO session as well as any currently open services.

Save Functions

save_image_stack

```
save_image_stack('somepath\')
```

Saves the volume data as an image stack to disk. The stack is saved as PNGs.

save_image_raw

```
save_image_raw('somepath\file.dat')
```

Saves the volume data as a RAW binary file. Note you must specify the filename and .dat file extension.

save_msr

```
save_msr('somepath\file.msr')
```

Saves the measure objects to an MSR file.

save_tfn

```
save_tfn('somepath\file.tfn')
```

Saves the transfer function objects to a TFN file.

save_vrv

```
save_vrv('somepath\file.vrv')
```

Saves the visual bookmark objects to a VRV file.

save_obj

```
save_obj('somepath\file.obj')
```

Saves the mesh object to an OBJ file.

save_slc

```
save_slc('somepath\file.slc')
```

Saves the clipping plane objects to an SLC file.

save_projection

```
save_projection('somepath\file.png')
```

Saves the current projection to a PNG file.

save_text

```
save_text('some text', 'somepath\file.txt')
```

Saves the text string to a TXT file.

save_omero_image

```
save_omero_image('description')
```

Saves the currently loaded image to OMERO, note the image is an orphan ie: does not belong to a parent dataset or project.

```
save_omero_image(int datasetID, 'description')
```

Saves the currently loaded image to OMERO and attaches it to the dataset at datasetID.

Set Functions

set_watchfile

```
set_watchfile('file.txt')
```

This sets the VolViewer to watch a specific ascii file. The watch file can then contain script logic and the application will execute the contents anytime a changes made to the file. This is a simple way to script or interact with the viewer from an external application. See Example X for a use case.

set_logfile

```
set_logfile('log.txt')
```

This will set the debug output to a text file. Note under windows this will also hide the application console if it exists.

set_orientation

```
set_orientation(float angleX, float angleY, float angleZ)
```

This function will set the current view based on the three Euler floating point angles specified in degrees.

set_projection_mode

```
set_projection_mode('mode')
```

This will set the projection rendering mode based on the mode text. The function supports, MIP and SUM projection modes.

set_close

```
set_close()
```

This will close the application. This function is useful to call at the end of a script.

set_hide

```
set_hide()
```

This will hide the application user interface. This function is useful to call a script headless without showing any user interface.

set_hide_menubars

```
set_hide_menubars()
```

This will hide the menu bars of the user interface. This function is useful when running VolViewer as a window for other applications.

set_hide_orthosections

```
set_hide_orthosections()
```

This will hide the orthogonal section views of the user interface. This function is useful when running VolViewer as a window for other applications

set_show

```
set_show()
```

This will hide the application user interface. This is the default state.

set_show_menubars

```
set_show_menubars()
```

This will show the menu bars of the user interface. This is the default state.

set_show_orthosections

```
set_show_orthosections()
```

This will show the orthogonal section views of the user interface. This is the default state.

set_current_visual_bookmark

```
set_current_visual_bookmark(int bookmark_index)
```

Sets the current view to that of the visual bookmark at index.

set_channels

```
set_channels(int red, int green, int blue)
```

This function will toggle the channels on and off by specifying 0 or 1 to the channel integer. Note this function has no effect if the volume is greyscale.

set_zoom

```
set_zoom(float zoomamount)
```

This function will set the zoom for the current view.

set_shader

```
set_shader(int shader_mode)
```

This function will set the current shader program. The supported `shader_mode` are:

- 0: No Lighting
- 1: Photorealistic Lighting
- 2: Non-Photorealistic Lighting
- 3: Depth Peel

No Lighting is the default state.

set_depthpeal

```
set_depthpeal(int offset, int thickness)
```

This function will set the depth peel offset and thickness variables. Offset and Thickness are specified as integers from 0 to 100. This will only work if the Depth Peel shader is enabled. See also [set_shader\(\)](#)

set_depthcue

```
set_depthcue(int red, int green, int blue, int scale)
```

This will allow you to control the depth cue shading for the photo-realistic lighting and non-photo-realistic lighting shader programs. Colour is specified using integers in the 0 to 255 range. See also [set_shader\(\)](#)

set_gooch_lighting

```
set_gooch_lighting(int warm_r, int warm_g, int warm_b, int cool_r, int cool_g, int cool_b,  
int contribution, int exponent, int threshold)
```

This will set the warm and cool colour and silhouette rendering parameters for the non-photo-realistic lighting shader Good default values are:

```
warm_r:255  
warm_g:0  
warm_b:0  
cool_r:0  
cool_g:0  
cool_b:255  
contribution: 8  
exponent: 16  
threshold: 8
```

See also [set_shader\(\)](#)

set_light_colour

```
set_light_colour(int ambient_red, int ambient_green, int ambient_blue, int diffuse_red, int  
diffuse_green, int diffuse_blue, int specular_red, int specular_green, int specular_blue)
```

This will set the ambient, diffuse and specular light colours. Colour is specified as integers from 0 to 255.

See also [set_shader\(\)](#)

set_background_colour

```
set_background_colour(int red, int green, int blue)
```

This function sets the current background colour. Values are integers in the 0 to 255 range.

set_view

```
set_view('TOP')
```

This function sets the current view orientation, can be:

- 'TOP'
- 'BOTTOM'
- 'FRONT'
- 'BACK'
- 'LEFT'
- 'RIGHT'

set_clear_all_MSR

```
set_clear_all_MSR()
```

This function will clear all the loaded MSR objects.

Movie Functions

movie_rock

```
movie_rock(int stepsize, 'path\', float angle)
```

This will create the PNG movie frames for a rock movie. The angular step size, path and rock angle have to be specified. Note the angle is specified in degrees (0.0 to 360.0).

movie_rotate

```
movie_rotate(int stepsize, 'path\', int x, int y, int z, float angle)
```

This will create the PNG movie frames for a rotation movie. The angular step size, path, rotation axis, and rotation angle have to be specified. Note the angle is specified in degrees (0.0 to 360.0). Note the rotation axis is specified with 0 or 1.

movie_orthosections

```
movie_orthosections(int stepsize, 'path\')
```

This will create the PNG movie frames for the three sets of orthogonal sections. The sampling rate and path have to be specified.

movie_visualbookmarks

```
movie_visualbookmarks(int stepsize, 'path\')
```

This will create the PNG movie frames by interpolating the visual bookmarks. Note visual bookmarks have to be loaded, see the [open_vvr function](#). The sampling rate and path have to be specified.

Compute Functions

compute_filter

compute_filter_Collapse

```
compute_filter_Collapse()
```

This function will convert a multi-channel volume to single channel (greyscale), but computing the average using each data channel that contain data.

compute_filter_Invert

```
compute_filter_Invert(int channel)
```

This function will invert a specific channel. Note only values > 0 are inverted.

compute_filter_Threshold

```
compute_filter_Threshold(int channel, int threshold_value)
```

This function computes a threshold using the threshold_value on a specific channel. Note

thresholded values are set to 0.

compute_filter_Binarize

```
compute_filter_Binarize(int channel)
```

This function will binarize a specific channels. All values above 0 will be set to 255.

compute_filter_Clear

```
compute_filter_Clear(int channel)
```

This function will clear a specific channel by setting all values to 0.

compute_filter_CopyTo

```
compute_filter_CopyTo(int source_channel, int destination_channel)
```

This function will copy a source channel to a destination channel.

compute_filter_AddTo

```
compute_filter_AddTo(int source_channel, int destination_channel)
```

This function will append/add a source channel to a destination channel.

compute_filter_ReplaceWith

```
compute_filter_ReplaceWith(int source_channel, int destination_channel, int test_channel, int test_operator)
```

This function will replace data in the *destination_channel* using data from the *source_channel* if the condition *test_operator* is met in the *test_channel*.

Two *test_operator* are currently supported:

- 0 (voxel is not equal to zero)
- 1 (voxel is equal to zero)

for example:

```
compute_filter_ReplaceWith(1, 2, 0, 0)
```

Will replace data in channel 1 with data in channel 2 if data in channel 0 is not equal to zero.

compute_filter_SetOperator

```
compute_filter_SetOperator(int channelA, int channelB, int set_operator)
```

This function allows you to apply set operations on two channels. The supported *set_operators* are:

- A not B
- B not A
- B and A not A intersect B
- A intersect B

The result will be automatically written to the third unspecified channel.

compute_filter_ArithmeticOperator

```
compute_filter_ArithmeticOperator(int channelA, int channelB, int operator)
```

This function allows you to perform arithmetic operations on two channels. The supported operators are:

```
0 (addition, A+B)
1 (subtraction, A-B)
```

The result will be automatically written to the third unspecified channel.

compute_filter_Dilate

```
compute_filter_Dilate(int channel)
```

This function will perform a morphological dilate to the channel.

compute_filter_Erode

```
compute_filter_Erode(int channel)
```

This function will perform a morphological erode to the channel.

compute_filter_Open

```
compute_filter_Open(int channel)
```

This function will perform a morphological open to the channel.

compute_filter_Close

```
compute_filter_Close(int channel)
```

This function will perform a morphological close to the channel.

compute_filter_Median

```
compute_filter_Median(int channel, int filter_size)
```

This function will apply a median filter of size, filter_size to the channel.

compute_filter_AnisotropicDiffusion

```
compute_filter_AnisotropicDiffusion(int channel, float sigma, float dT, int iterations)
```

This function will apply an anisotropic diffusion filter of strength, sigma and size, dT for N iterations. Good starting values for the filter are:

```
sigma = 5.0
dT = 0.1
iterations = 5.0
```

compute_filter_Bilateral

```
compute_filter_Bilateral(int channel, int spatial, float range)
```

This function will apply a bilateral filter to a channel.

Good starting values for the filter are:

```
spatial = 1
range = 0.1
```

compute_filter_Gaussian

```
compute_filter_Gaussian(int channel, float sigma, int size)
```

This function will apply a gaussian filter of sigma and size to a channel.

Good starting values for the filter are:

```
sigma = 0.5  
size = 3
```

compute_filter_UnsharpMask

```
compute_filter_UnsharpMask(int channel, float sigma, int size)
```

This function will apply an unsharp mask to sharpen edges of a channel.

compute_filter_Sobel

```
compute_filter_Sobel(int channel)
```

This function will apply a sobel edge detector to a channel.

compute_filter_Sieve

```
compute_filter_Sieve(int source_channel, int size)
```

This function will apply a sieve on the channel for a specific size. A good starting value is:

```
size = 1000
```

compute_filter_FloodFill

```
compute_filter_FloodFill(int source_channel, int destination_channel, int window, int x, int  
y, int z)
```

This function will apply a region growing algorithm seeded at image space point x,y,z on the source_channel and the copy results to the destination channel. The seed point is allowed to grow if the channel data is with +/- window.

compute_filter_SmoothedDilate

```
compute_filter_SmoothedDilate(int source_channel, int destination_channel, int seed_window,  
int min_size, int gaussian_width, float gaussian_strength, int gaussian_window)
```

This function computes a smoothed dilate operator. The function is split into two parts:

1. The source channel is allowed to dilated around a seed point if the neighbouring voxel fall within the seed_window and until it reaches the min_size (in voxels).
2. Then it will start smoothing the resulting dilation using a gaussian of gaussian_width and gaussian_strength. The region is then thresholded using the gaussian_window. This operation is repeated until the region does not grow anymore.

compute_filter_MaxFlow

```
compute_filter_MaxFlow(int source_channel, int destination_channel, float w, float v, float  
k)
```

compute_GPUfilter

compute_GPUfilter_AnisotropicDiffusion

```
compute_GPUfilter_AnisotropicDiffusion(int channel, float sigma, float dT, int iterations)
```

This function will apply an anisotropic diffusion filter of strength, sigma and size, dT for N iterations. Good starting values for the filter are:

```
sigma = 5.0  
dT = 0.1  
iterations = 5.0
```

compute_GPUfilter_Bilateral

```
compute_filter_Bilateral(int channel, int spatial, float range)
```

This function will apply a bilateral filter to a channel.

Good starting values for the filter are:

```
spatial = 1  
range = 0.1
```

compute_GPUfilter_Gaussian

```
compute_GPUfilter_Gaussian(int channel, float sigma, int size)
```

This function will apply a gaussian filter of sigma and size to a channel.

Good starting values for the filter are:

```
sigma = 0.5  
size = 3
```

compute_GPUfilter_Median

```
compute_GPUfilter_Median(int channel, int filter_size)
```

This function will apply a median filter of size, filter_size to the channel.

compute_GPUfilter_Dilate

```
compute_GPUfilter_Dilate(int channel)
```

This function will perform a morphological dilate to the channel.

compute_GPUfilter_Erode

```
compute_GPUfilter_Erode(int channel)
```

This function will perform a morphological erode to the channel.

compute_GPUfilter_SmoothedDilate

```
compute_GPUfilter_SmoothedDilate(int source_channel, int destination_channel, int  
seed_window, int min_size, int gaussian_width, float gaussian_strength, int gaussian_window)
```

compute_isosurface

```
compute_isosurface(int source_channel, int quality, int threshold)
```

Computes a iso-surface on a given source channel using the threshold value (0-255). The quality/downsampling of the volume used to compute the surface can be specified using the quality value (0-100) from low to high quality.

Wait Functions

wait_for_opengl

```
wait_for_opengl()
```

This will force a call to glFinish() in order to force a block until all GL execution is complete. Unless this is called OpenGL commands will be called asynchronously from the main thread.

wait_for_quit

```
wait_for_quit()
```

Calling this function will make any successive functions calls be only executed when the user closes the application. This is useful for saving files to a specific location without requiring the user to explicitly do so. See example X for a simple use case.

Examples

Example 1

Open an Image, Filter it, Save a rendering, Save the processed stack

```
open_image_stack(0, 'E:\VOLUME DATA\Anti_Flower 294 (gfpl+txr+vis)\')
compute_filter_AnisotropicDiffusion(0, 5.0, 0.1, 10)
save_projection('E:\VOLUME DATA\Projections\Anti_Flower 294 (gfpl+txr+vis).png')
save_image_stack('E:\VOLUME DATA\Anti_Flower 294 (gfpl+txr+vis)\Filtered\')
```

Example 2

Open all Images in a folder, Filter them, Save projections, Save the processed stacks

```
open_images_folder(0, 'E:\VOLUME DATA\Confocal Timepoints\', FALSE)
compute_filter_AnisotropicDiffusion(0, 5.0, 0.1, 10)
save_projection('E:\VOLUME DATA\Confocal Timepoints\Projections\##ORIGINALNAME##.png')
save_image_stack('E:\VOLUME DATA\Confocal Timepoints\Processed\##ORIGINALNAME##\')
```

Example 3

Open an image, load an MSR shape template, wait for user to modify template, save MSR at quit

```
open_image_stack(0, 'E:\VOLUME DATA\ScanID1934_Leaf1\')
open_msr('E:\ShapeModel\Shapes\Template.msr')
wait_for_quit()
save_msr('E:\ShapeModel\Shapes\ScanID1934_Leaf1.msr')
```

Example 4

Open first time point, wait for user to create visual bookmarks, project all time points using the

visual bookmarks

```
open_image_stack(0, 'E:\VOLUME DATA\Timelapse\T0\')
wait_for_quit()
save_vrv('E:\VOLUME DATA\Timelapse\Bookmarks\Bookmarks.vrv')
set_hide()
open_images_folder(0, 'E:\VOLUME DATA\Timelapse\')
open_vrv('E:\VOLUME DATA\Timelapse\Bookmarks\Bookmarks.vrv')
set_projection_mode('SUM')
set_current_visual_bookmark(0)
save_projection('E:\VOLUME DATA\Timelapse\Projections\view0_##ORIGINALNAME##.png')
set_current_visual_bookmark(1)
save_projection('E:\VOLUME DATA\Timelapse\Projections\view1_##ORIGINALNAME##.png')
set_current_visual_bookmark(2)
save_projection('E:\VOLUME DATA\Timelapse\Projections\view2_##ORIGINALNAME##.png')
```

Example 5

Compute a depth projection for all images in the source folder.

```
set_hide()
open_images_folder(0, 'E:\-=VOLUME DATA=-\-=SCRIPT TEST=-\Batch Tile Test\', FALSE)
set_projection_mode('MIP')
compute_filter_Threshold(0, 20)
set_zoom(-300)
set_shader(3)
set_depthpeal(0,33)
save_projection('E:\-=VOLUME DATA=-\-=SCRIPT TEST=-\Projected\##ORIGINALNAME##.png')
set_close()
```

Example 6

Traverse all folders/subfolders and replicate the folder structure in the save path and save 3 projections for each dataset

```
set_hide()
open_images_folder(0, 'E:\-=VOLUME DATA=-\-=OPT GALLERY=-\', TRUE)
set_view('BOTTOM')
save_projection('E:\-=Generated=-\Renderings\##ORIGINALPATH##\##ORIGINALNAME##_BOTTOM.png')
set_view('FRONT')
save_projection('E:\-=Generated=-\Renderings\##ORIGINALPATH##\##ORIGINALNAME##_FRONT.png')
set_view('LEFT')
save_projection('E:\-=Generated=-\Renderings\##ORIGINALPATH##\##ORIGINALNAME##_LEFT.png')
```

Example 7

Use VolViewer as a standalone viewer. This will display an MSR file in a standalone window with no user interface.

```
set_hide_menubars()
set_hide_orthosections()
open_msr('C:\Data\Leaf.msr')
```

Example 8

This will load a custom settings.ini file, load a stack and apply some non-photo-realistic rendering to it.

```
open_ini('E:\-=VOLUME DATA=-\-=SCRIPT TEST=-\Tests\settings.ini')
open_image_stack(3, 'E:\-=VOLUME DATA=-\-=OPT GALLERY=-\Antiriniuhm_Meristem
(r512g110usmall)')
set_shader(2)
set_gooch_lighting(255,0,0, 0,0,255, 8,16,8)
set_depthcue(0,0,0, 20)
```

Example 9

This will connect to an OMERO server and load a specific image.

```
open_omero_connection('my.omero.server.com', 4064, 'username', 'password')
open_omero_image(2933)
close_omero_connection()
```

Example 10

This will connect to an OMERO server and process all images in the dataset (id=143) and save them back on the server to the dataset (id=1223).

```
open_omero_connection('my.omero.server.com', 4064, 'username', 'password')
open_omero_dataset(143)
compute_filter_AnisotropicDiffusion(0, 5.0, 0.1, 10)
save_omero_image(1223, 'Anisotropic Diffusion 5.0 0.1 10')
close_omero_connection()
```

Example 11

This will batch upload a directory of images to OMERO and link them to the dataset (id=1012).

```
open_images_folder(3, 'E:\--VOLUME DATA--\--SCRIPT TEST--\Batch Test', false )
open_omero_connection('my.omero.server.com', 4064, 'username', 'password')
save_omero_image(1012, 'description')
close_omero_connection()
```

This page was last modified on 27 March 2012, at 10:38.

This page has been accessed 358 times.

[Privacy policy](#)

[About DMBI - Data Management for Bio-Imaging](#)

[Disclaimers](#)



VolViewer msr file specification

[Back to VolViewer Scripting in more detail](#)

[Back to VolViewer](#)

Contents [\[hide\]](#)

- [1 Concepts represented in the MSR file](#)
 - [2 General Syntax Overview](#)
 - [2.1 Number and String values](#)
 - [3 Field names and specifications](#)
 - [3.1 FILE HEADER: Global Directives](#)
 - [3.1.1 File version](#)
 - [3.1.2 Location of the original data](#)
 - [3.1.2.1 Supported reference types](#)
 - [3.1.3 Scale](#)
 - [3.1.4 Time of the data](#)
 - [3.1.5 Algorithm identification](#)
 - [3.1.6 Number of objects](#)
 - [3.2 FILE BODY: Structured Directives](#)
 - [3.2.1 Objects](#)
 - [3.2.2 Vertices](#)
 - [3.2.3 Edges](#)
 - [3.2.4 Faces](#)
 - [3.2.5 Volumes](#)
 - [3.2.6 Structured Directive Properties](#)
 - [3.2.6.1 Render Type](#)
 - [3.2.6.2 Normals](#)
 - [3.2.6.3 Colours](#)
 - [3.2.6.4 Morphogens](#)
 - [3.2.6.5 Growth](#)
 - [3.2.6.5.1 Face Growth](#)
 - [3.2.6.5.2 Edge Growth](#)
 - [3.2.6.6 Labels](#)
 - [3.2.6.7 Lists](#)
- [4 Examples](#)
 - [4.1 A Triangle](#)
 - [4.2 Two Triangles as different objects](#)
 - [4.3 Triangle with labels](#)
 - [4.4 A Leaf with Labels and Lists](#)

Concepts represented in the MSR file

An MSR file represents objects (geometry and growth) associated to a unique data file (for example a 3D stack).

The file is a succession of directives. Directives are either global (in the header) or per object (in the body). All directives are mandatory, unless stated otherwise.

General Syntax Overview

MSR files are simple text files (ASCII), whose general structure is :

```
FIELDNAME = FIELDVALUE # Comment
```

All field names are case insensitive. Comments start with a '#' and end at the end of the line. The file should be parsed after removing the comments from the file.

When parsing, empty lines (i.e. truly empty or with only spaces and comments) must be skipped.

Except for version 0, that shouldn't be used anymore, the first line should be:

```
MSR_VERSION = major.minor
```

The file should be readable in one pass. To help, each list of fields is preceded with a list count with the syntax:

```
fieldnameCOUNT = count  
fieldname = value1  
fieldname = value2  
...
```

Number and String values

Numbers can be either in decimal or scientific notation format:

```
0.001  
1e-3  
1E-3
```

are all accepted numbers.

String are represented using a character delimiter to allow for spaces, numbers etc.. in the string and easy reading. This special character is the single quote (').

```
'hello this is my 1st string'
```

is a valid string. This however means we require special handling of the single quote character in a string. This is represented by using two single quote characters as follows:

```
'hello it''s my first string'
```

is a valid string to represent: **hello it's my first string**.

(NB) Note that newlines are not accepted in strings.

Field names and specifications

FILE HEADER: Global Directives

These directives define the header of the file.

File version

```
MSR_VERSION = major.minor
```

This field has to be the first one of the file. If not present, it means the file is a legacy version 0, which is not covered by this document.

Version: ≥0.1 | Mandatory

Location of the original data

```
ORIGINALDATA = TYPE SPEC
```

Reference to the original data.

Version: ≥0.1 | Mandatory

Supported reference types

▶ NONE

```
ORIGINALDATA = NONE
```

This is when no original image is specified.

Version: ≥ 1.1 | Mandatory

▶ PATH

```
ORIGINALDATA = PATH '/path/to/someimage'
```

The path to some data ie: an image is stored using '/' as the directory separator. The path may be absolute or relative. If relative, it is expressed relative to the position of the MSR file.

Version: ≥ 0.1 | Mandatory

▶ OMEROID

```
ORIGINALDATA = OMEROID 'serveraddress:port/imageid'
```

Example:

```
ORIGINALDATA = OMEROID 'cmpdartsvr1.cmp.uea.ac.uk:4063/456215'
```

Where SERVERADDRESS can be either the IP address or the host name of the server, PORT is the port used by the OMER server and IMAGEID is the unique id within the database.

Version: ≥ 0.1 | Mandatory

Scale

```
SCALE = sx sy sz
```

Size of a unit in meters. The three component correspond to the size of a unit square along the X, Y and Z axes. If not present, it is assumed be equivalent to:

```
SCALE = 1 1 1
```

Version: ≥ 0.1

Time of the data

Can be either TIME or DATE.

```
TIME = time
```

The time is expressed in hours from an unspecified reference.

```
DATE = 'YYYY/MM/DD hh:mm:sec'
```

The time and day the image has been taken. The seconds may be a floating point value.

Version: ≥ 0.1 | Mandatory

Algorithm identification

String that uniquely identifies the algorithm used to generate that file, and its version.

```
ALGORITHM = STRING
```

Example:

```
ALGORITHM = 'measure_growth.py 1539'
```

Version: ≥0.1 | **Optional**

Number of objects

NB This field must be the last one of the header!

```
OBJECTCOUNT = number_of_objects
```

Version: ≥0.1

FILE BODY: Structured Directives

These are directives defining objects that can be labelled, used in lists, ...

Objects

```
OBJECTCOUNT = count  
OBJECT = 'name'
```

Defines objects by their name. This element must be right after the header. Actually, OBJECTCOUNT defines when the header ends.

Version: ≥1.1 | **Mandatory**

Vertices

```
VERTCOUNT = count  
VERT = Vx Vy Vz
```

Defines the list of vertices of an object, and their positions. Vx, Vy and Vz should be floating point values.

Version: ≥1.1 | **Contained in OBJECT** | **Mandatory**

Edges

```
EDGECOUNT = count  
EDGE = source target
```

Defines the list of edges of an object. Source and target are 0-based indices in the list of vertices.

Version: ≥1.1 | **Contained in OBJECT** | **Optional** | **Requires VERT**

Faces

```
FACECOUNT = count  
FACE = v1 v2 v3 ... vn
```

Defines the list of faces of an object. The vertices are specified by 0-based index in the list of vertices.

Version: ≥1.1 | **Contained in OBJECT** | **Optional** | **Requires VERT**

Volumes

```
VOLCOUNT = count  
VOL = facel face2 face3 ... facen
```

Defines the list of volumes of an object. The faces are specified by 0-based index in the list of faces.

Version: ≥1.1 | **Contained in OBJECT** | **Optional** | **Requires FACE**

Structured Directive Properties

This contains the elements that can be contained within a structured directive.

Render Type

```
RENDERTYPE = 'type'
```

This defines how an object should be rendered. This is only a hint that is application-specific.

Version: ≥0.1 | **Contained in** OBJECT | **Optional**

Normals

```
xxxxNORMAL = Nx Ny Nz
```

Normals are 3D normalized vectors and xxxx is to be replaced by the parent directive. Note that normals can only be specified after the directive count has been specified.

Example:

```
FACENORMAL = 0.0 1.0 0.0
```

Version: ≥1.1 | **Contained in** VERT, EDGE, FACE | **Optional**

Colours

```
xxxxCOLOUR = R G B
```

Colours are specified as an R,G,B vector with values from 0 to 255, xxxx is to be replaced by the parent directive. Note that colours can only be specified after the directive count has been specified.

Example:

```
FACECOLOUR = 0 255 0
```

Version: ≥1.2 | **Contained in** VERT, EDGE, FACE, VOLUME | **Optional**

Morphogens

Morphogens are scalar values associated with vertices. Each morphogen has a name and each morphogen is defined for every vertex. Morphogen names are specified by a directive of the form:

```
VERTMGENNAMES = name name name...
```

listing the names of all of the morphogens as quoted strings. The values for the morphogens are specified by directives of the form:

```
VERTMGEN = amount amount amount...
```

The amounts are listed in the same order as their names were. There is one VERTMGEN line for each vertex.

Version: ≥1.2 | **Contained in** VERT | **Optional** | **Requires:** VERTCOUNT

Growth

Face Growth

This is a list of growths specifications for all faces. As the number has to be the same as the number of faces, it is not re-specified. The start of the list is given by:

```
FACEGROWTHDT = dt
```

Where dt is the difference in time between this object and the object used as reference for this growth. Note that, if dt is negative, it means the growth represent the growth from the previous time to the current one, but expressed in the current reference frame.

Next, the growth parameters are given by:

```
FACEGROWTH = kmax kmin Vx Vy Vz
```

Where k_{max} is the relative elemental growth rate of growth along the major axis, k_{min} the relative elemental growth rate of growth along the minor axis, and (V_x, V_y, V_z) is a unit vector defining the major axis. The minor axis can be found with the cross-product between the major axis and the normal to the face. Note that k_{max} is taken as the largest absolute value of the two growth rates.

Note, there might be many growth specified for a same object, but only one per dt.

Version: ≥ 0.1 | **Optional** | **Requires:** FACECOUNT

Edge Growth

This is a list of growths specifications for all edges. As the number has to be the same as the number of edges, it is not re-specified. The start of the list is given by:

```
EDGEGROWTHDT = dt
```

Where dt is the difference in time between this object and the object used as reference for this growth. Note that, if dt is negative, it means the growth represent the growth from the previous time to the current one, but expressed in the current reference frame.

Next, the growth parameters are given by:

```
EDGEGROWTH = k
```

Where k is the relative growth rate along the edge.

Note, there might be many growth specified for a same object, but only one per dt.

Version: ≥ 0.1 | **Optional** | **Requires:** EDGECOUNT

Labels

```
xxxxLABEL = n 'label1' 'label2' ... 'labeln'
```

This is a list of labels attached to some directive. If a directive is labelled, every element of it must be.

Labels start with the count for the number of labels followed by the label strings. The xxxx is to be replaced by the labelled directive. Note that a directive can be labelled only after the directive count has been specified.

Example:

```
VERTLABEL = 2 'petiole' 'cell corner'
```

Empty labels are represented by:

```
VERTLABEL = 0
```

Version: ≥ 1.2 | **Contained in** VERT, EDGE, FACE, VOLUME | **Optional**

Lists

Lists are a named ordered series of indices attached to some directive. Note that a list can be only after the directive count has been specified. Lists start with a xxxxLISTCOUNT field to facilitate reading followed by the xxxxLIST directive that starts with a sting name label followed by the indices making up the list. The number of indices in the list must match the xxxxLISTCOUNT.

```
xxxxLISTCOUNT = N  
xxxxLIST = 'name' idx1 idx2 idx3 ... idxN
```

example:

```
EDGELISTCOUNT 5  
EDGELIST 'lamina outer edge' 5 6 4 8 2
```

Version: ≥ 1.2 | **Contained in** VERT, EDGE, FACE, VOLUME | **Optional**

Examples

A Triangle

```
MSR_VERSION = 1.2
ORIGINALDATA = PATH '/path/to/image'
SCALE = 1 1 1
TIME = 0
OBJECTCOUNT = 1
OBJECT = 'Triangle'
VERTCOUNT = 3
VERT = -10 0 0
VERT = 10 0 0
VERT = 0 10 0
EDGECOUNT = 3
EDGE = 0 1
EDGE = 1 2
EDGE = 2 0
FACECOUNT = 1
FACE = 0 1 2
```

[File:Simple_triangle.svg](#)

Two Triangles as different objects

```
MSR_VERSION = 1.2
ORIGINALDATA = PATH '/path/to/image'
SCALE = 1 1 1
TIME = 0
OBJECTCOUNT = 2
OBJECT = 'Triangle1'
VERTCOUNT = 3
VERT = -10 0 0
VERT = 10 0 0
VERT = 0 10 0
EDGECOUNT = 3
EDGE = 0 1
EDGE = 1 2
EDGE = 2 0
FACECOUNT = 1
FACE = 0 1 2
OBJECT = 'Triangle2'
VERTCOUNT = 3
VERT = -10 0 10
VERT = 10 0 10
VERT = 0 10 10
EDGECOUNT = 3
EDGE = 0 1
EDGE = 1 2
EDGE = 2 0
FACECOUNT = 1
FACE = 0 1 2
```

[File:Twotriangles.svg](#)

Triangle with labels

```
MSR_VERSION = 1.2
ORIGINALDATA = PATH '/path/to/image'
SCALE = 1 1 1
TIME = 0
OBJECTCOUNT = 1
OBJECT = 'Triangle'
VERTCOUNT = 3
VERT = -10 0 0
VERT = 10 0 0
VERT = 0 10 0
VERTLABEL = 0
VERTLABEL = 0
VERTLABEL = 1 'Tip'
EDGECOUNT = 3
EDGE = 0 1
EDGE = 1 2
EDGE = 2 0
EDGELABEL = 1 'Base'
EDGELABEL = 0
EDGELABEL = 0
FACECOUNT = 1
FACE = 0 1 2
```

[File:Triangle_labelled.svg](#)

A Leaf with Labels and Lists

```
MSR_VERSION = 1.2
ORIGINALDATA = PATH '/path/to/image'
SCALE = 1 1 1
TIME = 0
OBJECTCOUNT = 1
OBJECT = 'ShapeModelLeaf'
VERTCOUNT = 15
VERT = 54 144 0
VERT = 120 143 0
```

[File:LeafLabelsandList.svg](#)

```

: VERT = 157 164 0
: VERT = 201 204 0
: VERT = 254 221 0
: VERT = 325 208 0
: VERT = 385 171 0
: VERT = 422 127 0
: VERT = 374 82 0
: VERT = 318 62 0
: VERT = 255 36 0
: VERT = 200 50 0
: VERT = 154 72 0
: VERT = 120 100 0
: VERT = 47 97 0
: VERTLABEL = 1 'Primary'
: VERTLABEL = 1 'Primary'
: VERTLABEL = 1 'Secondary'
: VERTLABEL = 1 'Secondary'
: VERTLABEL = 1 'Primary'
: VERTLABEL = 1 'Secondary'
: VERTLABEL = 1 'Secondary'
: VERTLABEL = 2 'Primary' 'Tip'
: VERTLABEL = 1 'Secondary'
: VERTLABEL = 1 'Secondary'
: VERTLABEL = 1 'Primary'
: VERTLABEL = 1 'Secondary'
: VERTLABEL = 1 'Secondary'
: VERTLABEL = 1 'Primary'
: VERTLABEL = 1 'Secondary'
: VERTLABEL = 1 'Secondary'
: VERTLABEL = 1 'Primary'
: VERTLABEL = 1 'Primary'
: VERTLISTCOUNT = 13
: VERTLIST = 'Lamina' 1 2 3 4 5 6 7 8 9 10 11 12 13
: VERTLISTCOUNT = 4
: VERTLIST = 'Petiole' 1 0 14 13
: EDGECOUNT = 15
: EDGE = 0 1
: EDGE = 1 2
: EDGE = 2 3
: EDGE = 3 4
: EDGE = 4 5
: EDGE = 5 6
: EDGE = 6 7
: EDGE = 7 8
: EDGE = 8 9
: EDGE = 9 10
: EDGE = 10 11
: EDGE = 11 12
: EDGE = 12 13
: EDGE = 13 14
: EDGE = 14 0

```

modified on 1 March 2012 at 21:32 ••• 28 views

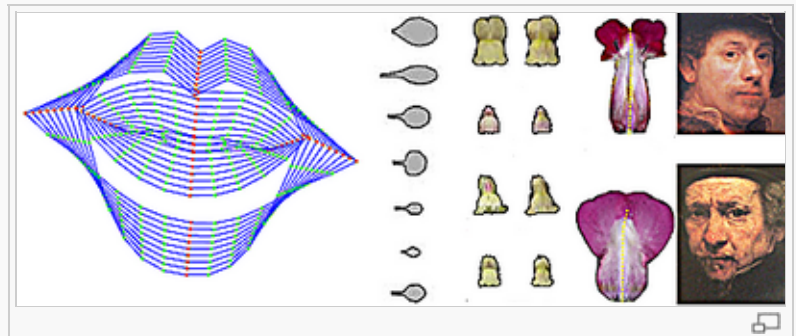
AAMToolbox Details

[Back to Software](#)

What is the *AAMToolbox* and why?

We wish to understand how the biological organs grow to particular shapes. For this we need a tool to help us think through what we expect to see, i.e. model growth (*Gfibox*), and we need to **make measurements** of real biological organs to test our expectations (hypotheses).

However, the shapes of biological organs rarely make measurement simple - how do you measure the two or three dimensional (2 or 3D) shape of an ear, leaf or Snapdragon flower? The *AAMToolbox* is **designed to measure the shapes of organs** relative to each other.



Some shapes of mouths, leaves, petals and portraits

How does is measure shapes?

Limitations?

modified on 13 February 2012 at 18:19 ••• 75 views

Tutorials on the Shape modelling toolbox

[Back to AAMToolbox Documentation](#)

The models shown in these tutorials illustrate features of the AAMToolbox software. They are not designed to understand the shape and appearance modelling which is better done from the published literature [for example](#). Viewing these pages. Some versions of *Firefox* and *Explorer* do not create satisfactory prints even though you can view the pages with no problems. *Chrome* does appear to produce good printouts.

Fives ways to use AAMToolbox

- 1) **Analysing shapes.** i.e. the arrangement of points around a shape
- 2) **Viewing the data in *shape space*.** i.e. approximating the data with two principle components
- 3) **Comparing shapes from samples of different groups** for example, comparing faces from different cartoon characters
- 4) **Analysing shape and appearance.** In addition to the points around a shape, analyse the appearance (grey scale or colour) within the shape.
- 5) **Analysing 3D shapes**

How to use these tutorials. First [download and install the AAMToolbox](#). A zip file containing the project (*PRJ_CartoonFaces*) is available [here](#). Download and unzip into a directory. Then, from Matlab, change directory into the project and launch the AAMToolbox

```
cd PRJ_CartoonFaces
AAMToolbox
```

This project contains as set of faces that have been analysed using 2D shape models

The set of faces



1 How to analyse 2D shapes using the Graphical User Interface

The process of analysing a set of images is:-

1. Create a **new project**. AAMToolbox project names are automatically prefaced with PRJ_. They have a particular directory structure and the images to be analysed need to be copied into the subdirectory called Cropped. It is best if they are all the same size.
 - ▶ [Tutorial on making a new project.](#)
2. **Create a point model** template. Points are placed around the object of interest, i.e. around a face or leaf. The set of points constitute the **point model**. Every image will be marked up in the same way.
 - ▶ [Tutorial on point model template.](#)
3. To **digitise each image**, move the points to the corresponding positions in each image in turn. The positions must **correspond to the same material points** in each image, i.e. the tip of the leaf, the corner of an eye, or halfway along a line between the two ends of the mouth.
 - ▶ [Tutorial on point model editor.](#)
4. Generate the shape model using **principal component analysis (PCA)**
 - ▶ [Tutorial on statistical model generator.](#)
5. **View the result** by varying each important component in turn. We call this *walking* the shape model. This movie shows a walk.
 - ▶ [Tutorial on viewing statistical model.](#)

Mean shape (points) joined by lines. The movie

6. **Best fit point model** using only the principle components.

▶ [Tutorial on fitting a model to a particular image.](#)

shows deviations from the mean by varying the principle component.

2 Viewing the results in *Shape-Space*. What is Shape-Space?

1. PCA allows us to view the data from a new angle: *Shape Space* rather than normal viewing space. This movie shows a trajectory through shape space that is projected back into normal viewing space.

▶ [Tutorial on shape space.](#)

Interpolating a walk from one point model shape to another in shape space.

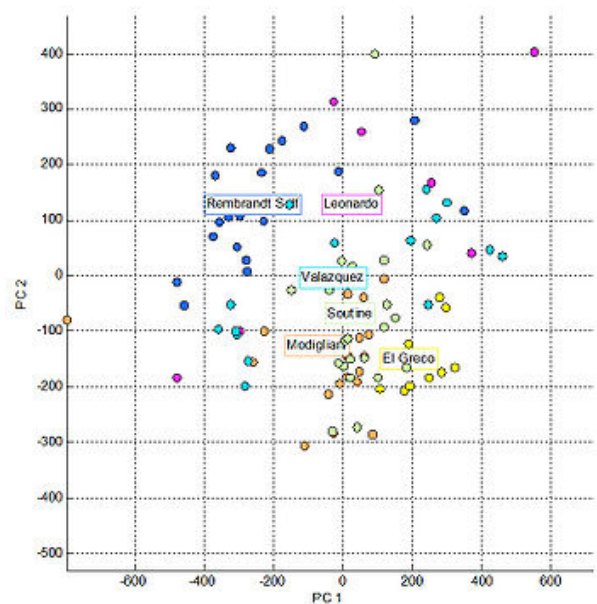
Seen these elsewhere?



3 Comparing shapes in *Shape-Space*.

None of the above portraits exist. They were created by placing point models on samples of each painter's work and computing the PCA of both the shape and the appearance. Each image is the mean shape coloured with the mean appearance. They are arranged in date order: Giotto (~1300), Leonardo da Vinci (~1600), Velazquez (~1630), Rembrandt (~1650), Modigliani (~1900), Soutine (~1930).

1. Compare shapes of different populations in shape space.
 - ▶ Groups of samples can be projected into a common shape space and compared. In this case portraits.
 - ▶ By focussing on the main contributions to shape it might be possible to relate positions in shape space to, for example, genotype in leaves ([Langlade et al 2005](#), [Bensmihen et al. 2010](#)) and petals ([Whibley et al 2006](#), [Feng et al. 2010](#))

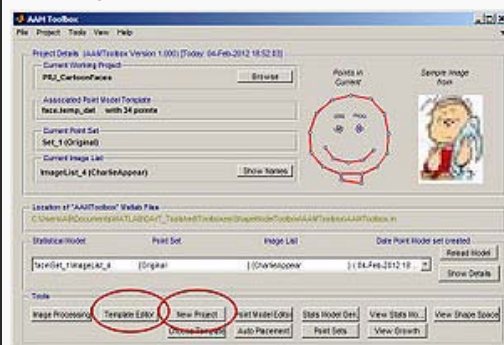


AAMToolbox new project

[Back to Tutorial pages](#)

Making a new *AAMToolbox* project

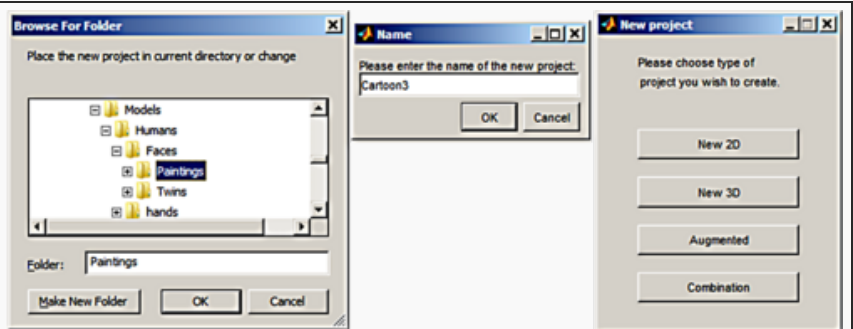
The *AAMToolbox* user interface. Click on the **New Project**.



This will lead you through three steps

1. Select the directory in which to create the project
2. Name the project: the prefix PRJ_ will be added automatically
3. Select the type of project, in this case a 2D shape model

Next **copy the images to be analysed into the directory 'Cropped'**. *Template Editor* button circled in red.

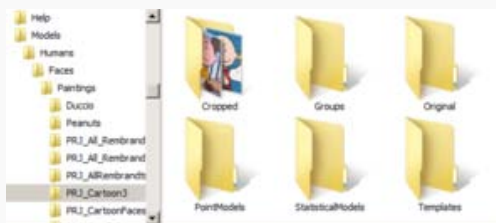


Step 1

Step 2

Step 3

Creating a new project with the name: PRJ_Cartoon3



The subdirectories in a project are created automatically. Whilst the original images should be stored in the 'Original' subdirectory, images that have been cropped to the same size ready for shape analysis are stored in the directory called 'Cropped'.

modified on 5 February 2012 at 12:15 *** 28 views

AAMToolbox template

[Back to Tutorial pages](#)

Contents [[hide](#)]

[1 Template Editor](#)

[1.1 1\) Start a new project from the AAMToolbox control panel](#)

[1.2 2\) Select an image on which to base the template](#)

[1.3 3\) Create a new Template using the Template Editor](#)

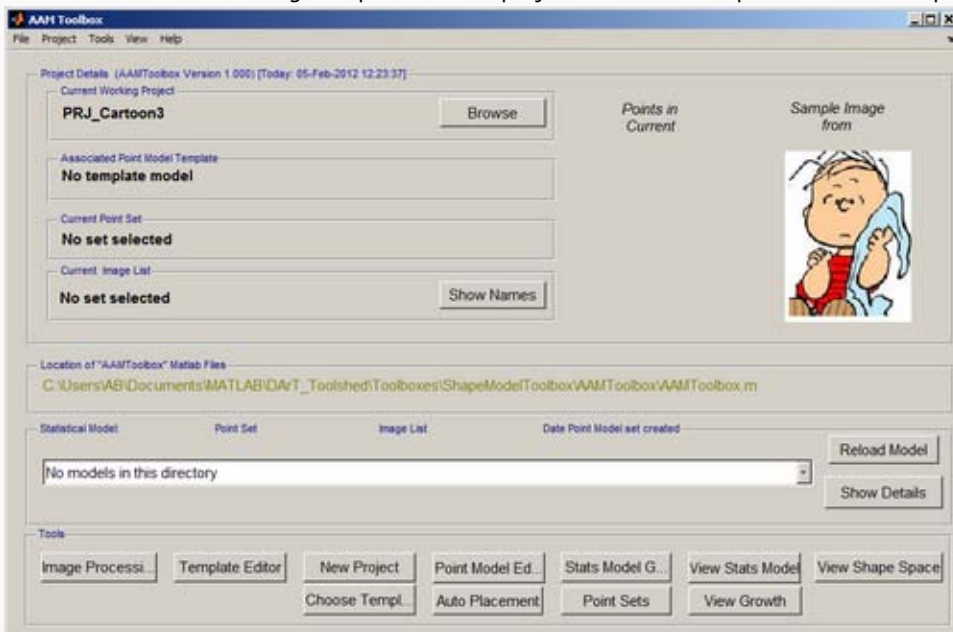
[1.4 4\) Choose a template using the AAMToolbox control panel](#)

Template Editor

1) Start a new project from the AAMToolbox control panel

[Tutorial on starting a new project](#)

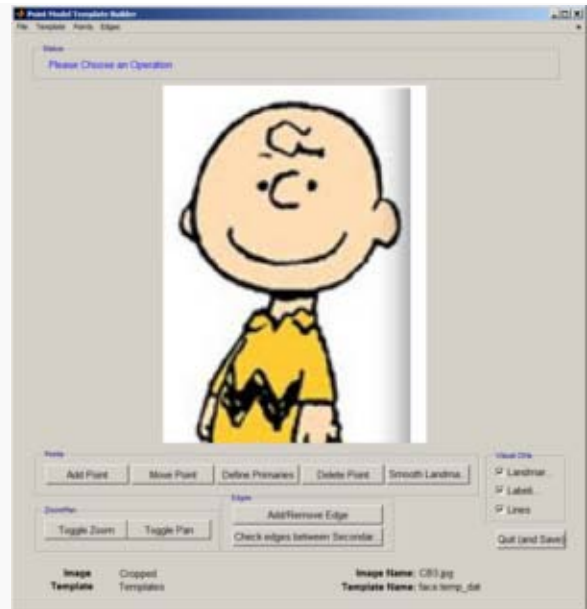
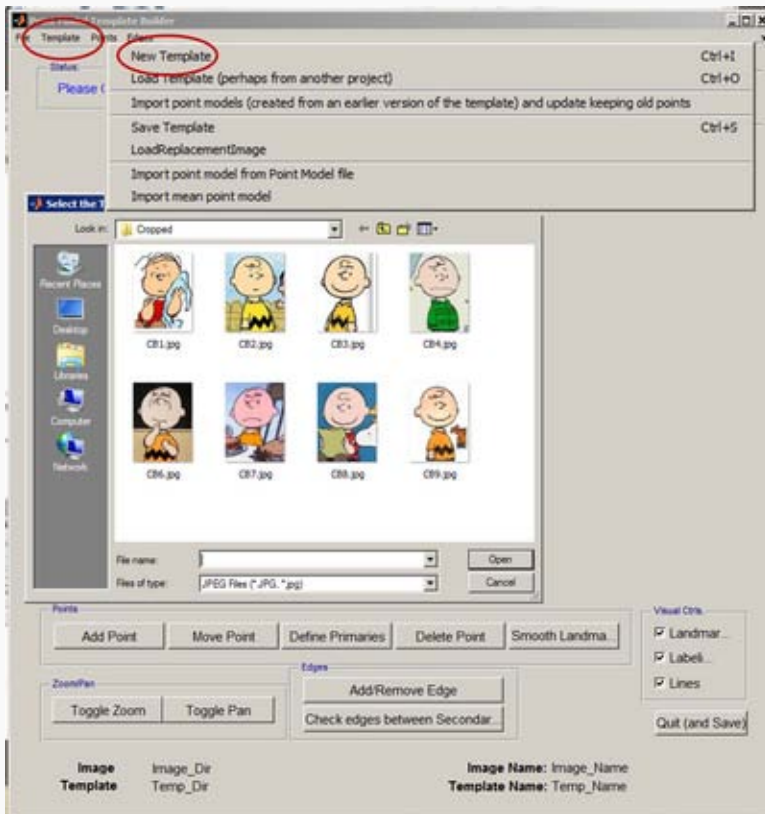
Notice that one of the images copied into the project has been co-opted as an icon representing the project.



Once the template has been added, a second icon will be automatically created that represents the template itself.

2) Select an image on which to base the template

Select the **Template Editor** from the **AAMToolbox control panel**. Then, from the Template Editor menu select **New Template**. This will open the directory of Cropped images and allow you to select one on which to base the template.



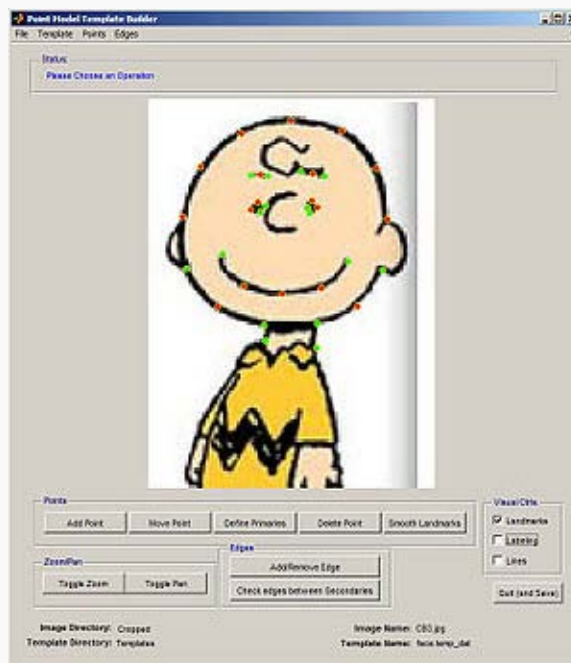
The new template will be based on this image.

Idea: From the Templates menu, Templates (and associated images) can be copied from other, existing, projects.

3) Create a new Template using the Template Editor

1. Add points, **select 'Add Point'** and click on a feature that can be recognised in every image. For example, the junction between face and neck, or face and ear lobe, the ends.
2. Having added the points, define those that you are most reliable as 'primary'. They will turn green. The remaining points are called secondary points and should be spread evenly between primary points.
3. Secondary points can be **spread evenly** along a cubic spline that joins the primary landmarks by selecting 'Smooth Landmarks'.
4. Lines joining points can aid understanding, but they do not contribute to the final shape model.

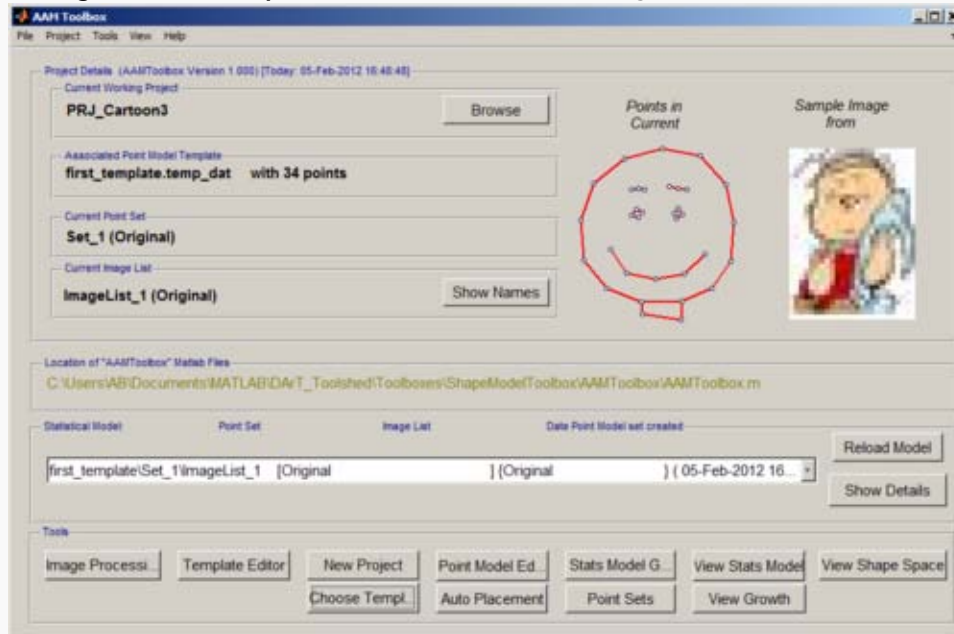
When the template is correct 'Quit (and Save)' using the button, bottom right.



1. The **point model is a set of points** (red and green dots) that are used to build the shape model
2. These are subdivided into primary (red) and secondary (green) landmarks
3. Edges (red lines) are added to make the point model more understandable to humans

4) Choose a template using the AAMToolbox control panel

Having created a template it should be selected using the AAMToolbox main interface, 'Choose Template'.



Once the template has been added, a second icon will be automatically created that represents the template itself. Here it is the red smiling face.

Notice:

1. The name of the current working project, top left
2. The template name together with the number of points
3. The current set of points (that will be selected automatically)
4. The current list of images (that will be selected automatically)
5. Most importantly, these details are summarised in the Listbox. Each model that is created will be associated with an entry in this listbox.
6. The next step is to move the point model into position over each of the images using the **Point Model Editor**.

modified on 5 February 2012 at 17:00 ••• 64 views

AAMToolbox point model editor

[Back to Tutorial pages](#)

Point Model Editor

1) Select the Point Model Builder from the AAMToolbox control panel

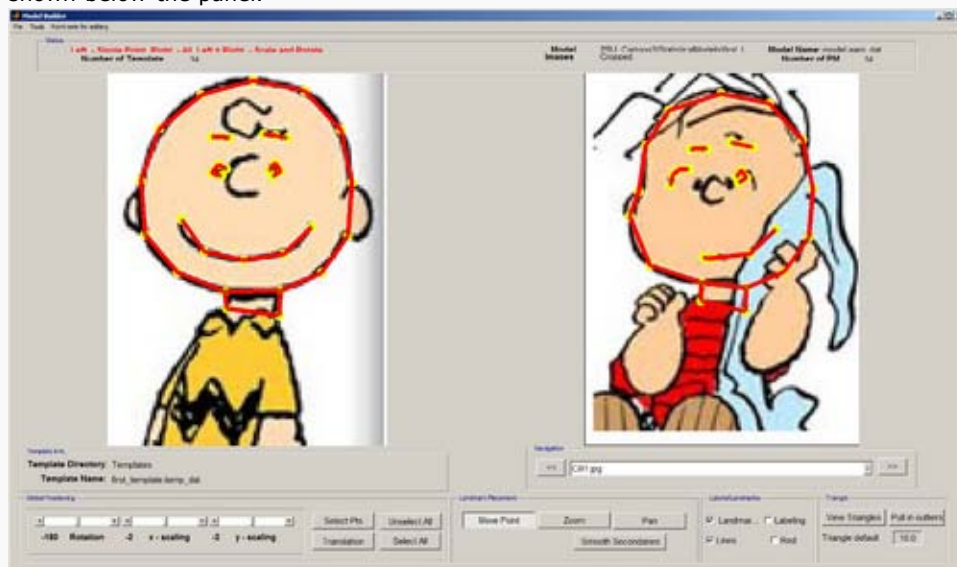
The point model editor looks like.



The original template is shown on the left to provide a reference. The current image point model is shown in the right hand panel. The point model and lines shown here are enlarged.

Notice that the point model does not fit the image on the right. This editor allows you to **move the points to the corresponding positions in the new image**.

The following shows the result of moving into place the points on the left side of the righthand panel. The associated image name is shown below the panel.



Once the points have all been moved into place, **smooth the distribution of secondary points**. Finally, click the '>>>' button to save the point model and **move to the next image**.

Once a point model has been built for all the images, close the control panel and **generate a shape model**.

modified on 5 February 2012 at 17:30 ••• 37 views

University of East Anglia / School of Computing Science / Powered by MediaWiki

AAMToolbox statistical model generator

[Back to Tutorial pages](#)

Contents [\[hide\]](#)

- [1 Statistical shape generator \(PCA\)](#)
 - [1.1 Principle component analysis of point-models and image pixel values \(appearance models\)](#)
- [2 Inputs and outputs](#)
- [3 Statistical Model Generator control panel.](#)
 - [3.1 Steps to build a shape model using principle component analysis \(PCA\)](#)
 - [3.1.1 Step 1\) Options](#)
 - [3.1.2 Step 2\) Type of model](#)
 - [3.1.3 Step 3\) Point models and images contributing to the model](#)
 - [3.1.4 Step 4\) Generate Model](#)
 - [3.1.5 Step 5\) Exit](#)

Statistical shape generator (PCA)

Principle component analysis of point-models and image pixel values (appearance models)

To run the PCA select **Stats Model Generator** from the AAMToolbox control panel:

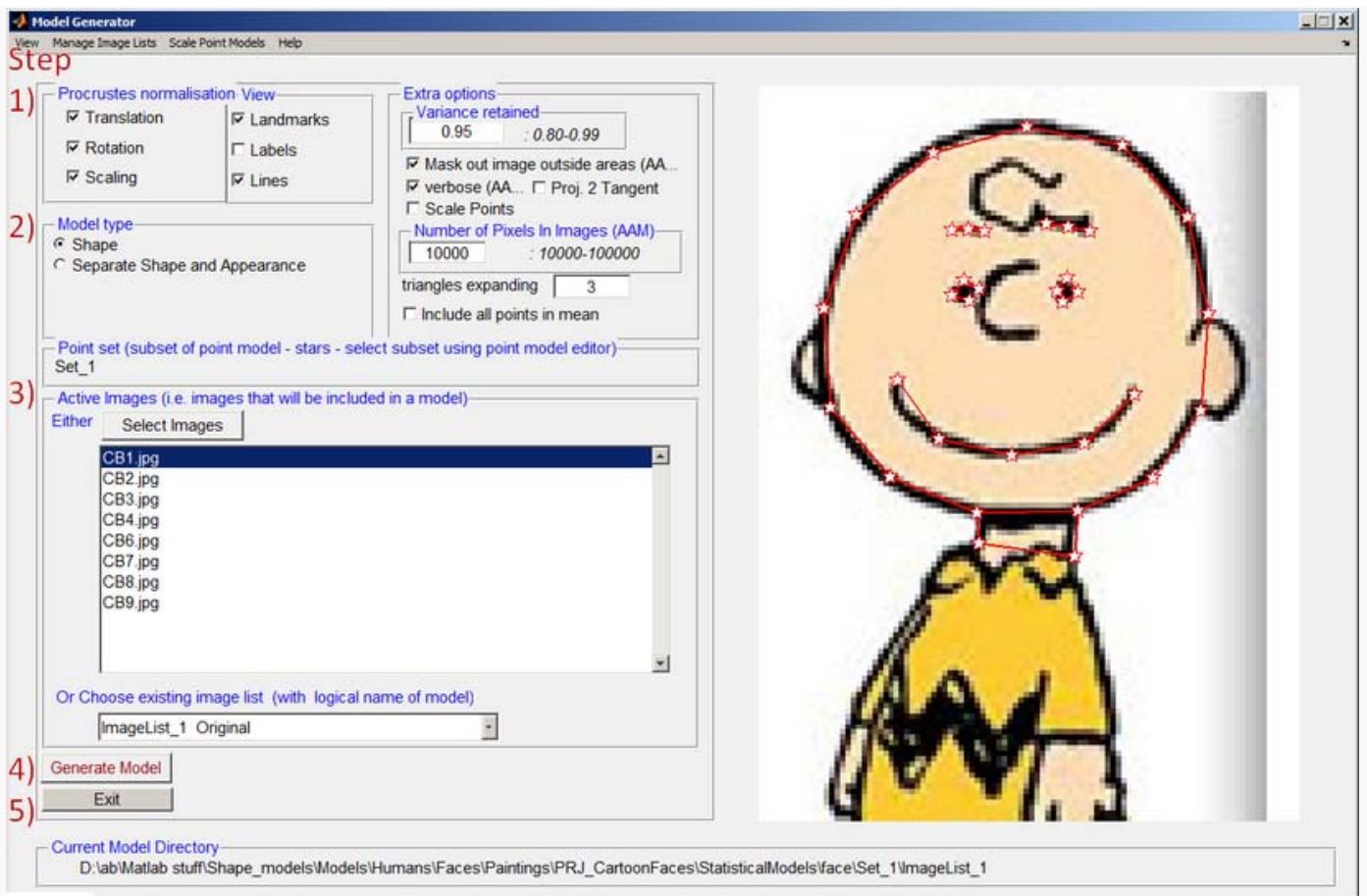
Inputs and outputs

- ▶ Shape model
 - ▶ Input: landmark points, i.e. the point-models for all images of interest.
 - ▶ Output: mean shape and associated shape model that captures deviations from the mean. Enough principle components are included to account for 95% of the variance from the mean (set using Options).
- ▶ Appearance model
 - ▶ Input: landmark points, i.e. the point-models together with the image intensity within triangles determined by a Delaunay triangulation of the points.
 - ▶ Output: mean appearance and associated appearance model that captures deviations from the mean.

Statistical Model Generator control panel.

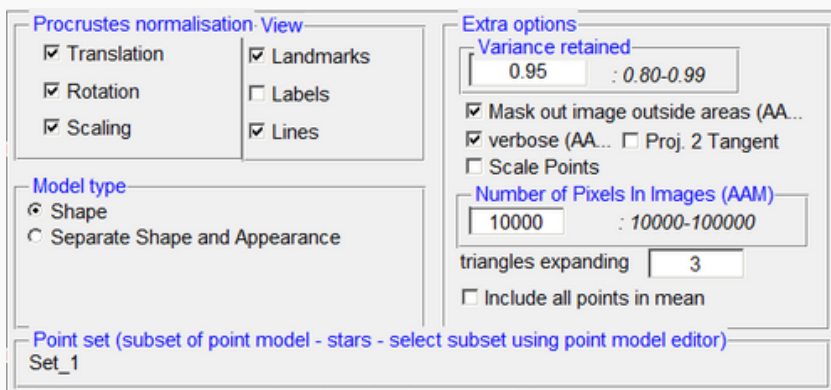
Showing the steps (in red) to build a statistical model.

Notice: the set of landmarks represented as stars will be included in the shape model, in this case all of them. For this model you can accept the default settings and jump directly to step 3 and select all the images. Then compute the principle component analysis (PCA) using step 4 and return to the AAMToolbox on the way to viewing the results.



Steps to build a shape model using principle component analysis (PCA)

Step 1) Options



- ▶ The first step in PCA is to align all the point models. In this case we assume that the images should be normalised to the **same scale, same rotation and same translational** position. This might not always be the case. If we want to model leaf growth then the point model should **not** be scaled. Alternatively, if we want to capture the pose of a face portrait the point models should not be rotationally aligned.

For the **cartoon faces we normalise for all three.**

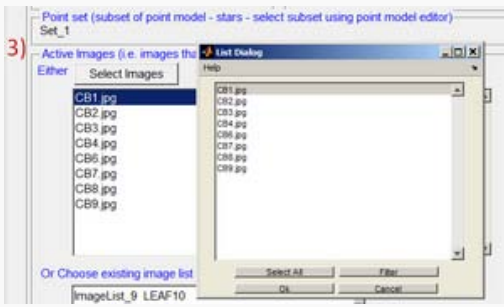
Step 2) Type of model

In this case the statistical **Shape** model will be generated from the point models alone. In other cases, we might want to include the image colours, pixel by pixel, in the PCA in which case we create two models **Shape and Appearance (AAM)** and accept the defaults for all the other options. However, ...

- ▶ ▶ If AAM then
 - ▶ Select whether to mask out the image not within the a boundary set by the outermost landmark points
 - ▶ Select the number of pixels to be used when forming the mean image intensity (appearance).
- ▶ **Point set.** The AAMToolbox supports the concept of point model 'sets'. In other words, subsets of landmarks can be grouped together into a set. Set_1 is always all the points. The points that are currently selected are shown as stars. Here, all the points are selected. The options are discussed in the Tutorial on 'sets'.
- ▶ If modelling a subset of landmark points it can be useful to place the model at the mean position of all the landmarks. This can be selected by ticking 'include all points in mean'. (A subset of landmark points is selected in the AAMToolbox:Point Model Editor.)

Step 3) Point models and images contributing to the model

- ▶ Select the image point models you want to include. Usually all of them. Here, select all.



- ▶ ▶ You will be invited to give a name to the model you are about to compute. You choose a name.

Step 4) Generate Model

Step 5) Exit

- ▶ The next step is to view results from the model, AAMToolbox: **View Stats Mode**

modified on 13 February 2012 at 16:03 *** 138 views

University of East Anglia / School of Computing Science / Powered by MediaWiki

AAMToolbox viewing statistical model

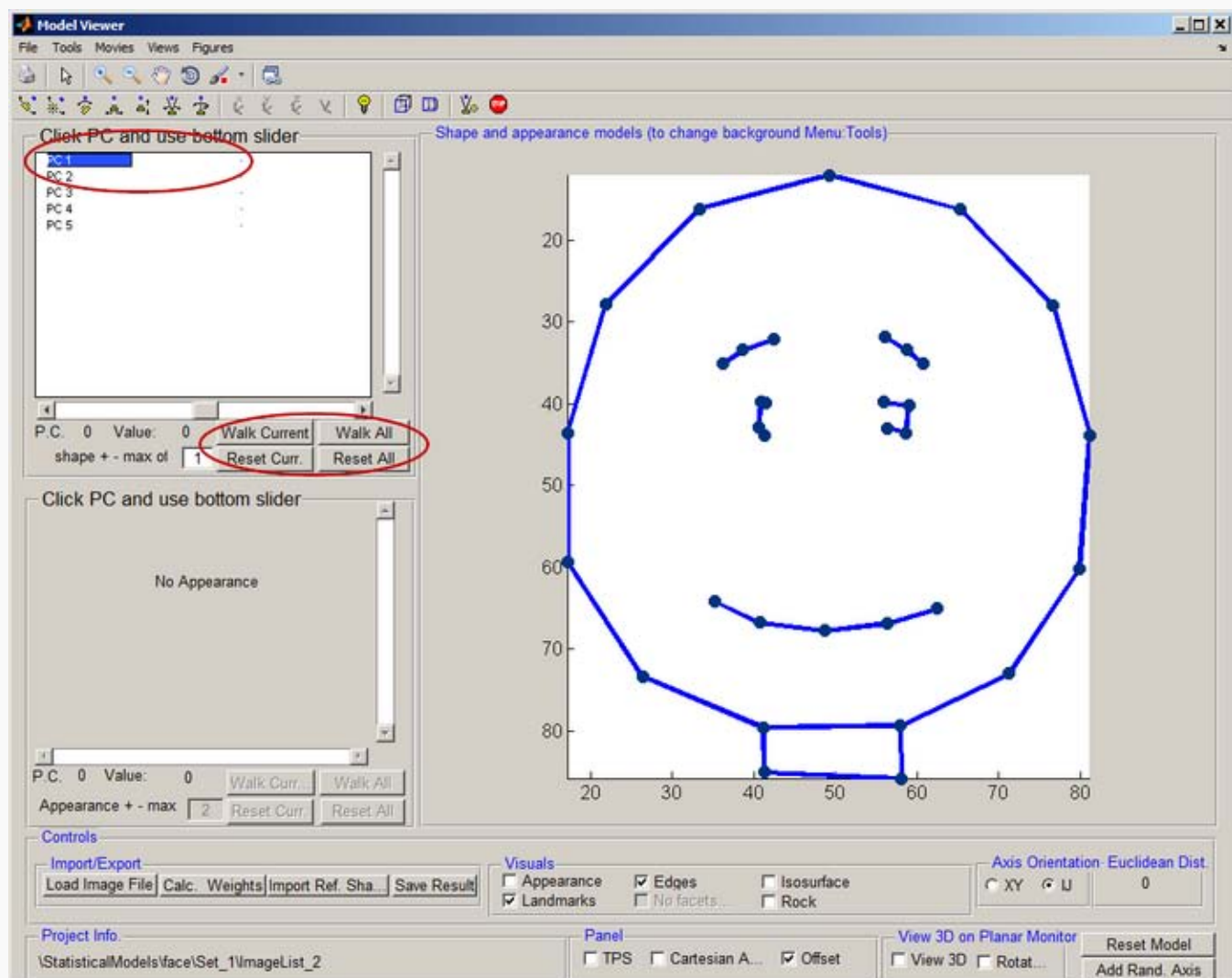
[Back to Tutorial pages](#)

Contents [hide]

- [1 Viewing shape and appearance models in 2D](#)
- [2 Walking through contributions of principle components.](#)
 - [2.1 Mean shape](#)
 - [2.2 Variations along the principle component axis](#)
 - [2.3 Variations along axis with the next largest variation](#)

Viewing shape and appearance models in 2D

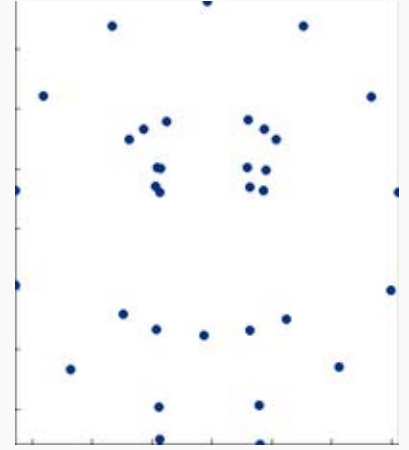
Walking through contributions of principle components.



Mean shape

The dots on the right represent the mean positions of the landmarks averaged over all the images in the model. These points mean more to us once they are joined by lines (see above screenshot). Capturing that higher meaning requires more analysis or, as in this case, human interaction (when we created the template).

Note that the point models had been normalised to the **same scale, same rotation and same translational** position.



Variations along the principle component axis

To visualise sliding (walking) along an axis imagine a straight line fitted to a set of data points. Then read-out the x,y coordinates as we move back and forth along the line. Correlation between x and y (the straight line) means that the outputs are correlated. In this model we are doing a similar thing as we slide along the axis (fitted line) we read-out values in the model coordinate system that are then transformed (by the model) back into the viewing coordinates.



Left, minus one standard deviation along PC1.

Middle, mean shape.

Right, plus one standard deviation along PC1.

Sliding (walking) along the principle component axis.

Variations along axis with the next largest variation



Left, minus one standard deviation along PC2.

Middle, mean shape.

Right, plus one standard deviation along PC2.

Sliding (walking) along the axis accounting for the second most amount of variation.

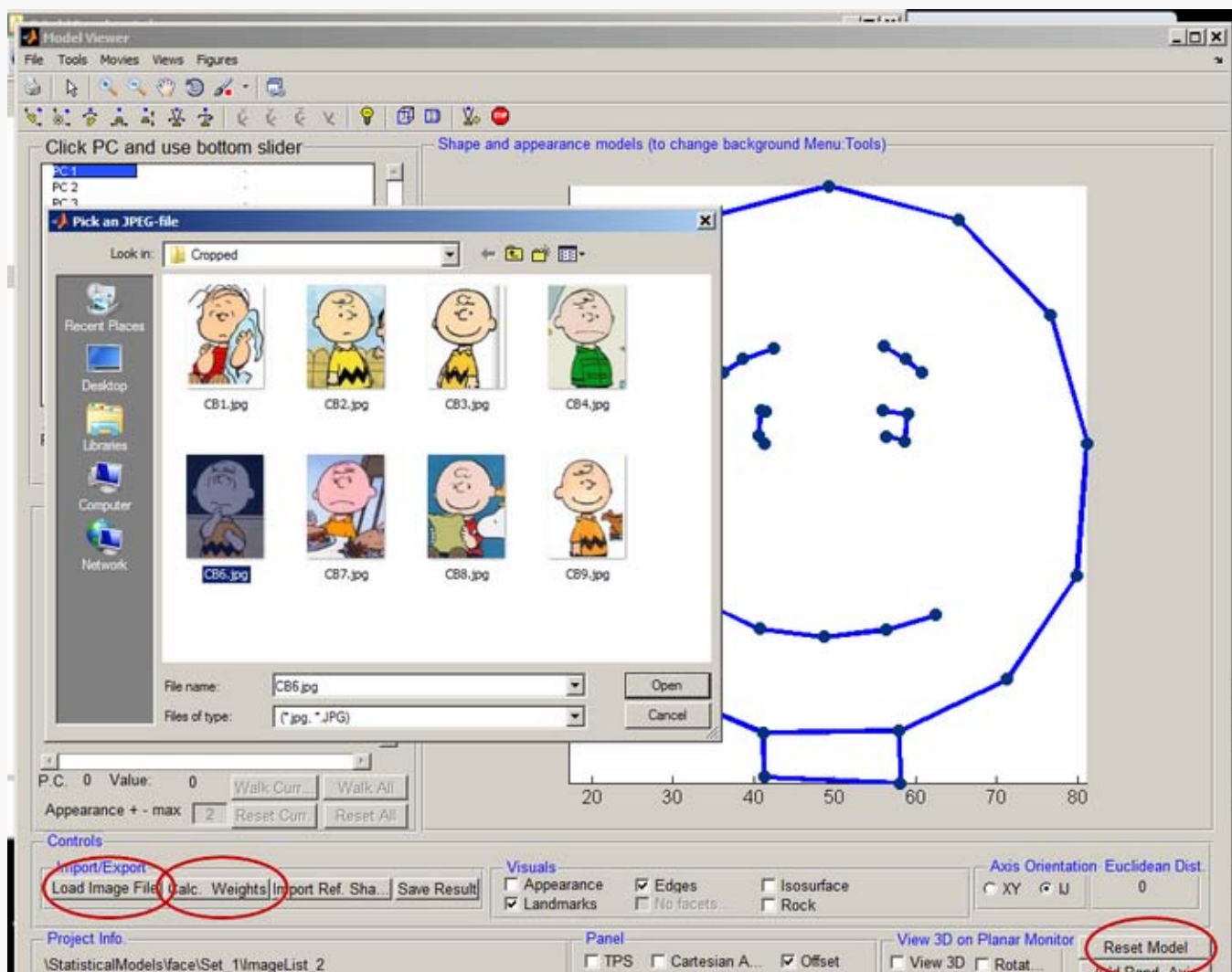
modified on 9 March 2012 at 10:24 ••• 75 views

AAMToolbox simplify point model

[Back to Tutorial pages](#)

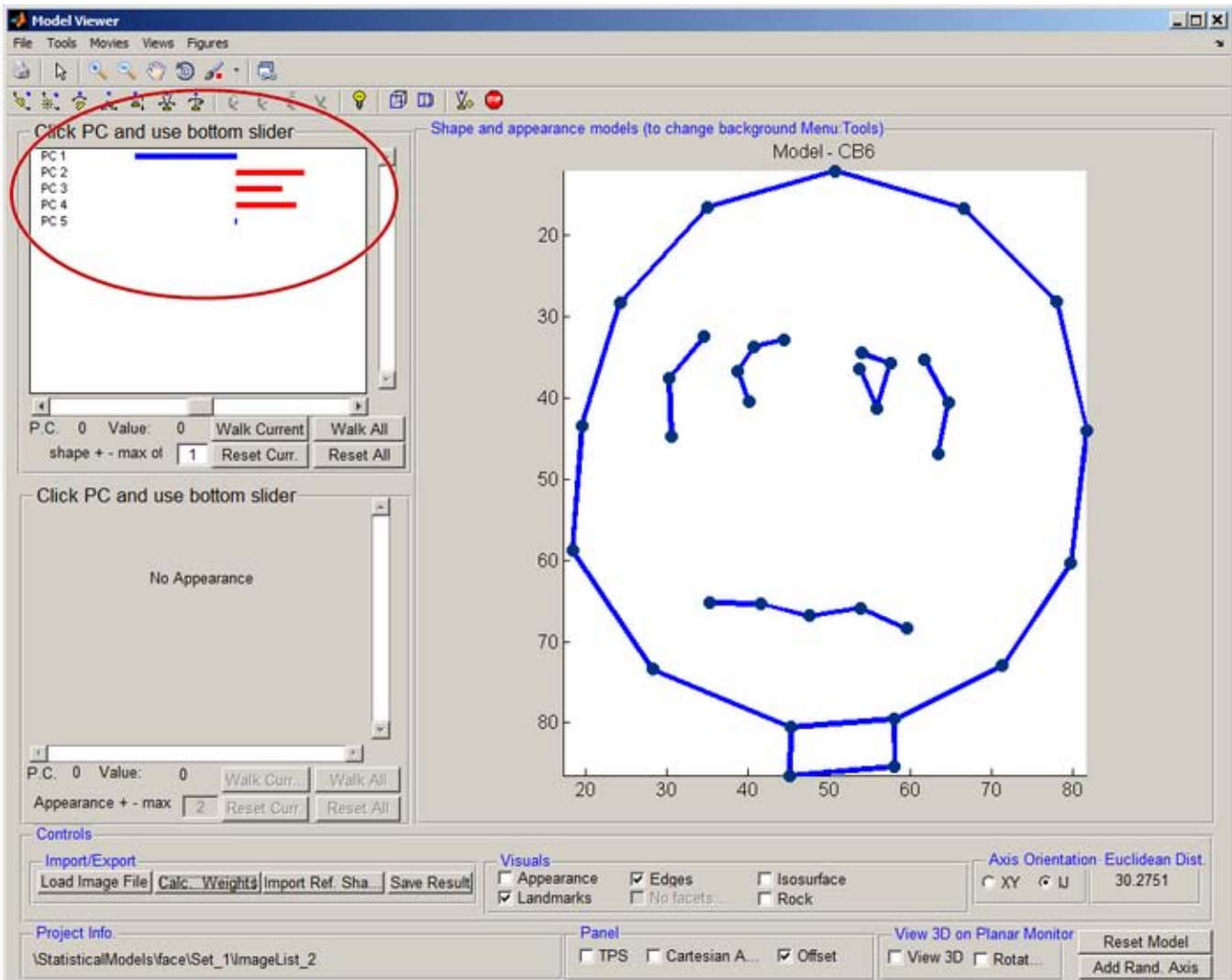
What does it mean to best fit the shape model to a particular image point model

All the point models were used to find the shape model. If there were just two dimensions, this would be similar to fitting a straight line to $y=ax$ by finding a . In the Cartoons shape model, there are $34 \times 2 = 68$ dimensions yielding 68 components. However, most of these represent noise and the Stats Model Generator found that 95% of the variance could be accounted for by just 5 of the components. So we should (actually, might is a better word) be able to represent any of the point models that we used to build the shape model. Here, we **project a particular point model into shape space, take the 5 principle components (setting the rest to zero), and project the result back into our normal viewing space**. The result should look similar to the point model itself.



In the act of **loading an image file** (highlighted in sub-window). Red ellipses highlight the buttons needed to load an image - actually the point model associated with an image - calculate the associated weights and reset the model back to the mean again. It is important to reset back to the mean before each import.

The reset is required because each time a point model is imported, its deviation from the mean is added to the current model. Thus, to add the deviations for one place in shape space to another, first load the other (reference shape) by using the *Import Ref. shape* button.



Result of Calculating the Weights (see button) . Notice that the bars representing the principle components have been adjusted to show how far this approximation to the point model deviates from the mean.

Here, we have referred to **shape space but what is it?** Project into shape space? See next tutorial.

modified on 20 February 2012 at 20:39 *** 47 views

University of East Anglia / School of Computing Science / Powered by MediaWiki

AAMToolbox viewing one population in shape space

[Back to Tutorial pages](#)

Contents [\[hide\]](#)

[1 What is Shape Space?](#)

[1.1 Mean shape](#)

[2 How do we view shape space?](#)

[3 Exploring further.](#)

[3.1 Exploring the data in shape space, where are the samples?](#)

[3.2 What is the distance between samples?](#)

[3.3 Can we interpolate between samples?](#)

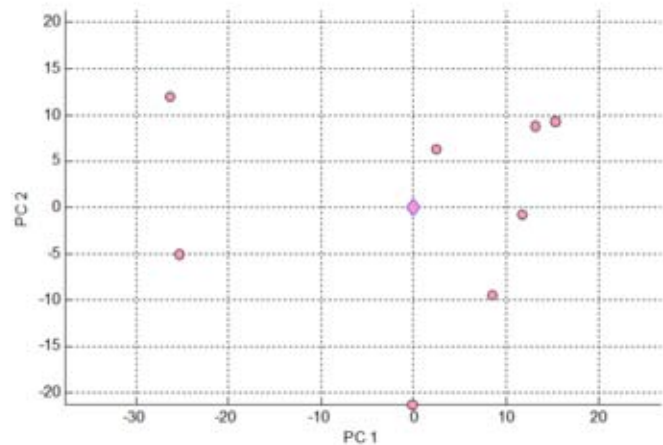
What is Shape Space?

Mean shape

The **pink diamond** shows the position of the **mean** cartoon shape plotted in shape space.

Each **spot** represents one of the **input point models** (images).

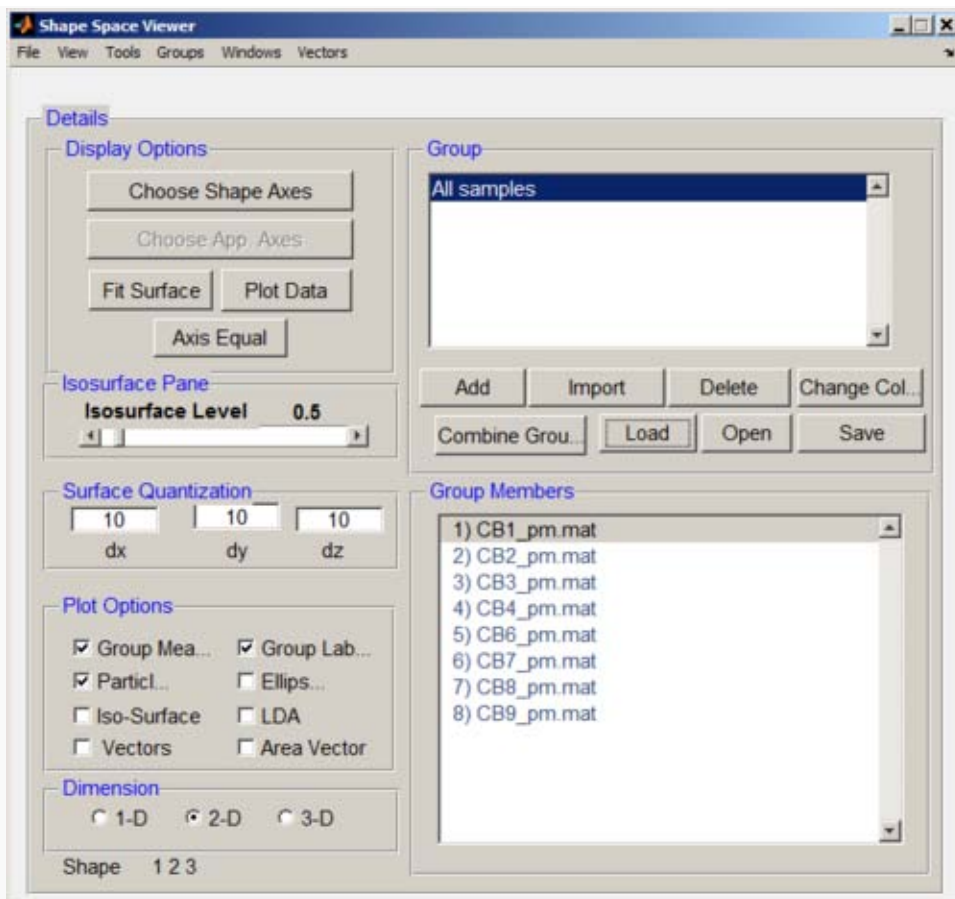
Each point model represents 38 points i.e. $38 \times 2 = 76$ values in the normal two dimensional viewing space OR a single point in 72 dimensional space. However, we cannot visualise 72 dimensional space so we have used PCA to choose a view point (a rotation of the space) such that we can see the two principle components. This allows us to visualise and **focus on the significant (in the sense of accounting for the most variance) parameters**. Of course, it might be another component that is most significant in biological terms. It is also quite possible that the biologically important parameters are correlated non-linearly in which case they could be invisible in this project because they are distributed into the new components as awkwardly as they were in the original data.



Shape Space: the principle component is plotted on the x-axis (abscissa) and the component that accounts for the second largest amount of variation is plotted on the y-axis (ordinate). Pink diamond shows the **position of the mean shape**. Each of the other points is a point model that was manually fitted to a sample image. In other words, each sample image (with several vertices) is a point in shape space.

How do we view shape space?

From the AAMToolbox interface click the **View Shape Space** button.



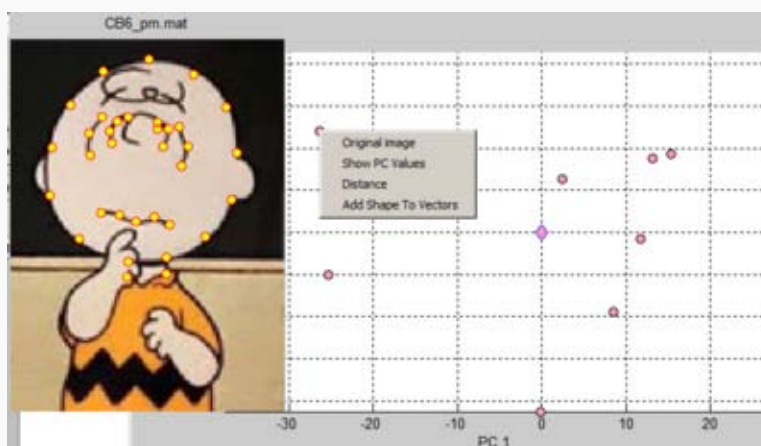
1. First **Add** samples (Add button).
2. **Load** the sample data (point models).
3. **Plot** Data, optionally make the **axis equal** (to keep a balanced view)
 - ▶ Change Colour (far right button) to select a colour
 - ▶ Ellipse to draw an ellipse around most of the data

Exploring further.

Exploring the data in shape space, where are the samples?

Each *pink circle marker represents a data sample*. Right clicking on a marker opens a context menu. Here we have selected to view the associated original image and its point model (shown to the left).

Showing the PC values causes them to be echoed to the command window.



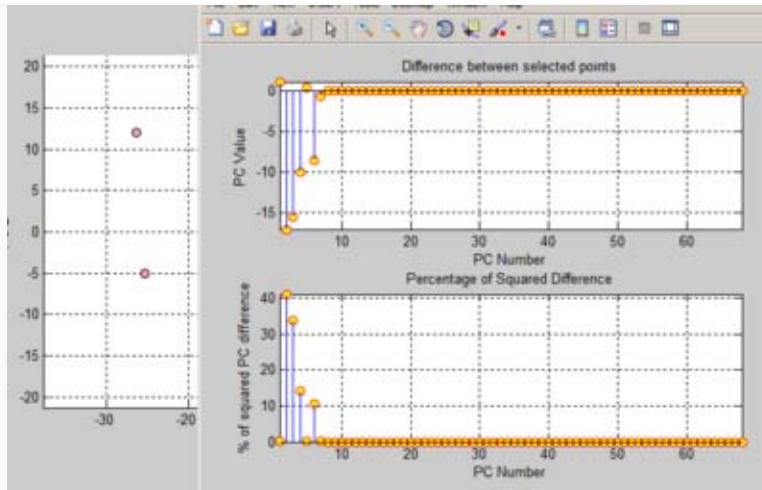
Shape Space context menu for the markers. The top-left sample has been selected and the associated image/point model is shown on the left.

What is the distance between samples?

Right clicking on a marker opens a context menu. Here we have selected to view the **distance between the two** visible markers.

The distances (and squared distances) for each of the components is shown in the right.

Showing the PC values causes them to be echoed to the command window.



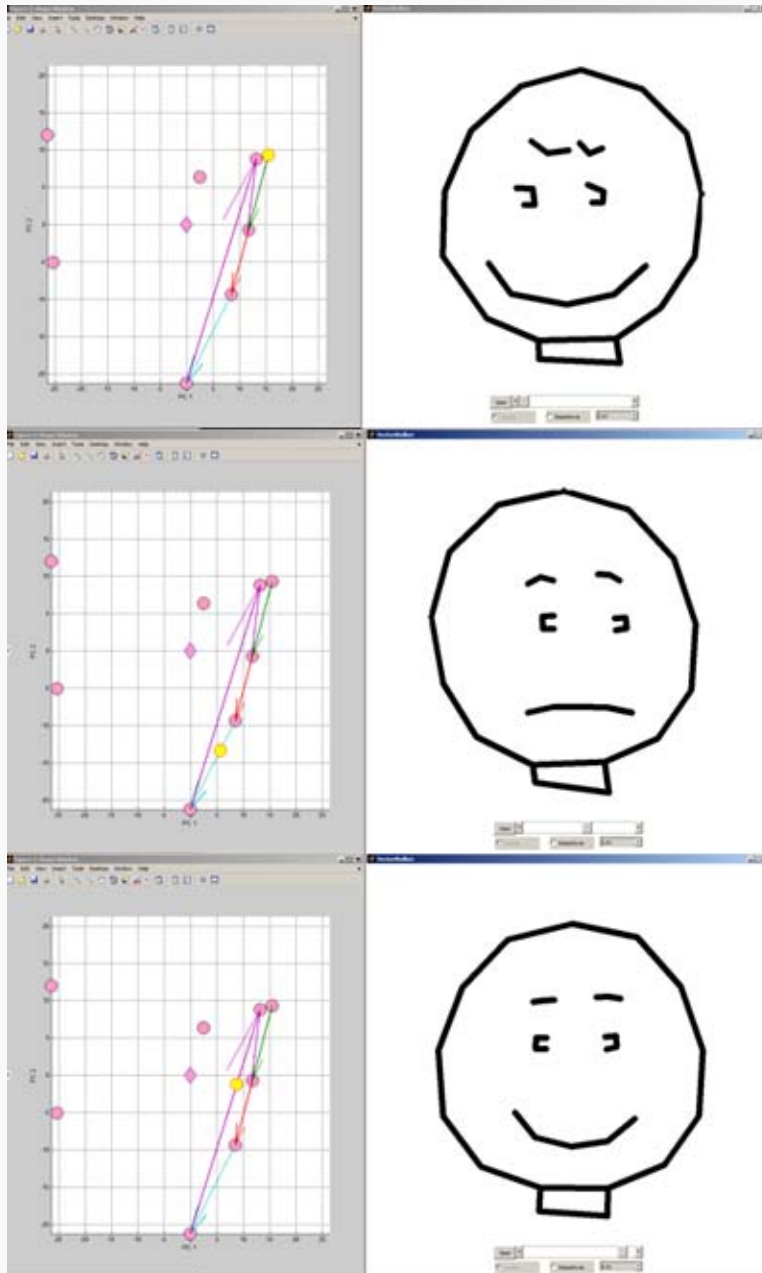
Shape Space: distance between two points.

Can we interpolate between samples?

Here we add a 'vector' (arrow) from one point and another.

Click on the Vectors button in the Shape Space Viewer interface and the vector will be displayed as an arrow. Click on a sequence of samples to create a path through shape space.

Select Menu: Vectors: Walk along vectors 'walk' along the vectors and see the sequence of results obtained by interpolation. In the sequence of images shown on the right the position in shape space is shown by the yellow disc together with the interpolated image.



Shape Space: **Yellow disc** shows a position in shape space and on the right is the interpolated shape.

modified on 17 April 2012 at 08:23 ••• 52 views

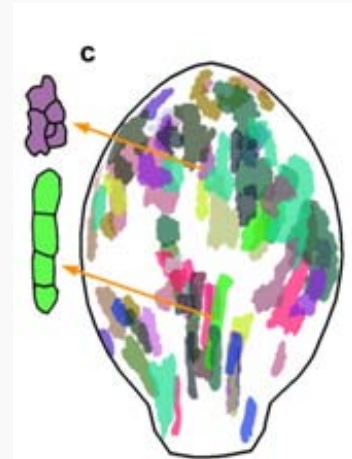
SectorAnalysisToolbox Details

[Back to BanghamLab software](#)

What? How? Where?

What? We wish to understand how **patterns of gene activity** in biological organs influence the developing **shape**. The shapes arise because different regions of, for example a sheet of cells, grow at different rates. The problem is that a particular shape outline could arise through a number of different patterns of growth inside the outline. To understand growth it is, therefore, necessary to **observe and measure growth rates** throughout the growing organ. Small organs growth can be tracked at the cell scale using confocal microscopy. As the organ gets larger, this becomes impractical and we use an alternative: clonal analysis.

Clones of cells arise from divisions of a single cell. Clones become useful if the cell (and its daughters) are marked in some way. The mark could be the shape of hairs in a fly wing, the **colour** of cells in a flower petal, or the presence of induced GFP. As result of multiple cell divisions, growth, can be followed by measuring the shape of the marked clone (sometimes called sector). The Sector Analysis Toolbox (**SAT**) is used **to quantify patterns of growth** by analysing the shapes of these marked sectors or clones.



Left: individual clones observed using confocal microscopy in two regions of an Arabidopsis leaf together with (right) the shapes of clones (from a number of leaves) that have been identified, labelled (by colour) and warped into the average leaf shape.

How? Procrustes analysis is used to find the average shape of the organ and each outline together with internal features, the marked sectors, is warped to the average. Then ellipses are fitted to each sector to provide an estimate of growth along the major and minor axes in the region of the sector. The **Sector Analysis Toolbox is written in Matlab** [\[1\]](#). It does not require any extra Mathworks toolboxes, nor any separately compiled modules. Matlab is available as a [30 day free trial](#) [\[2\]](#) and as a [student edition](#) [\[3\]](#). *Sector Analysis Toolbox* comprises around 20,000 lines of code.

Where? [Download](#) [\[4\]](#)

modified on 19 March 2012 at 07:52 ••• 81 views

SectorAnalysisToolbox Documentation

[Go back to software](#)

The models shown in these tutorials illustrate features of the SectorAnalysisToolbox software. Viewing these pages. Some versions of *Firefox* and *Explorer* do not create satisfactory prints even though you can view the pages with no problems. *Chrome* does appear to produce good printouts.

Analysing the shapes of cell clones in *Antirrhinum* petals

How to use the tutorial. First [download and install the SectorAnalysisToolbox](#). Then, from Matlab, change directory into the project and launch the SectorAnalysisToolbox.

antirrhinum petals sample data

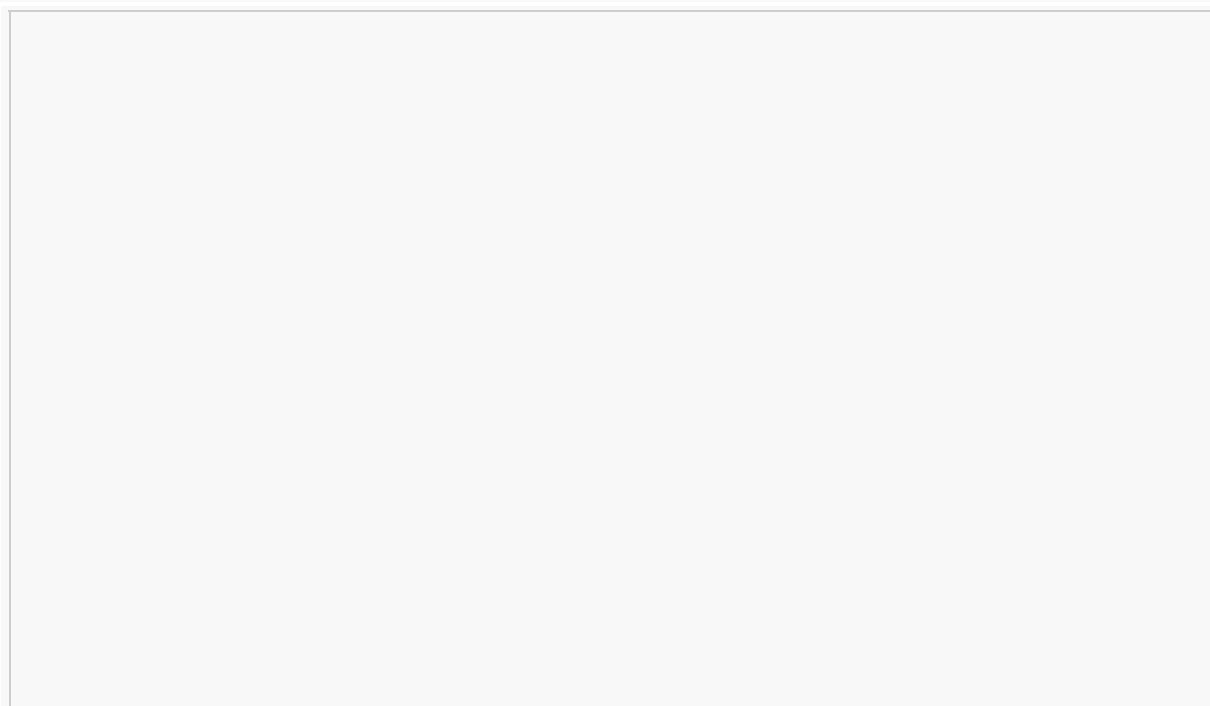


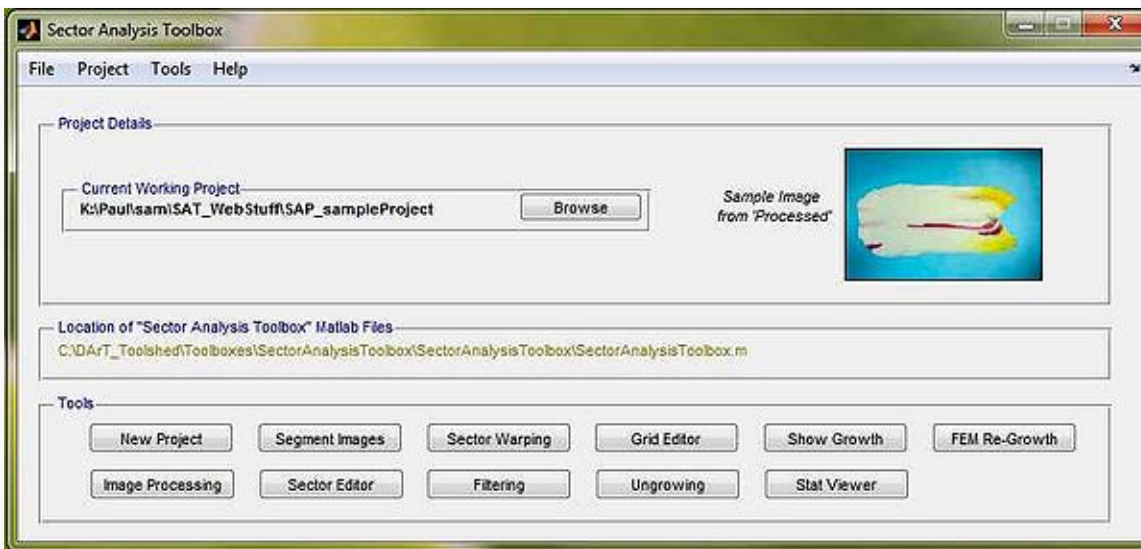
1 Creating a new project

To create a new project, click the **New Project** button and name your project when prompted. The toolbox automatically gives each project the prefix 'SAP_' (Sector Analysis Project). The project directory structure will be saved to disk and contains the folders **Original**, **Processed** and **Staged**. Copy the sample antirrhinum images (P9F17P1I.JPG, P9F17P2I.JPG, P10F17P1I.JPG) into the **Staged** dir.

Once a project has been created, you can return to that project either by selecting that project in the **Current Directory** line at the top of the Matlab window, or by opening the toolbox and using the **Browse** button to select your project.

SAT main window





2 Segmenting the petal and clones

So far, we have created a new project and populated it with some sample antirrhinum petal images. We would now like to automatically segment the antirrhinum petal and the purple sectors.

Click the button **Segment Images**. A dialog box will ask the user to select a segmentation algorithm. Select **amelia_petal_sectors.m** from the drop down box as this segmentation algorithm is specifically designed to work with our data. The SAT will look in the **Staged** directory for *.jpg images to segment. A dialog box will ask the user which images to segment - in this case we will use the **Select All** button. A task bar indicates segmentation progress. When an image has been segmented, the image file in the **Staged** folder is replaced by a folder containing a matlab file called **edge_xy** (the organ outline coordinates), a .jpg snapshot of the original image, a .tif black and white (binary) image with the suffix **_organ** (corresponding to the petal/leaf outline), and a .tif binary image with the suffix **_sectors** (corresponding to the sectors).

The original image (left) segmented petal (centre) and segmented clones (right)



3 Editing the Segmented clones and organ edges

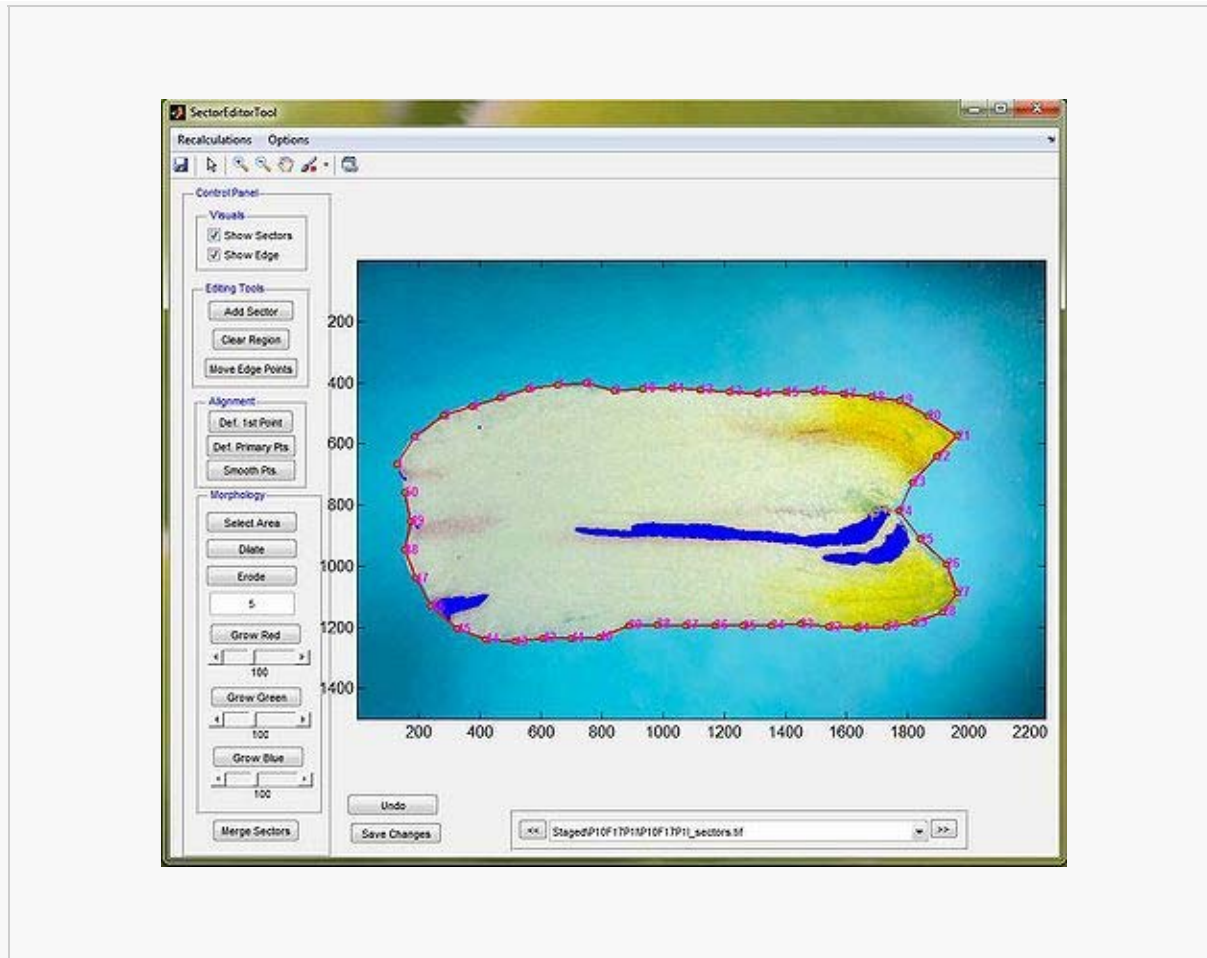
So far, we have used automated segmentation algorithms to segment petals and sectors within the petals. We would now like to view the results and if necessary perform any manual editing.

Click **Sector Editor** to open the Sector Editor Tool. The first image from the **Staged** folder should be displayed in this window. The outline of the petal section is marked by 50 edge points joined by a red line. Edge information are provided by the **edge_xy.mat** file and **filename_organ.tif**. The latter shows the organ shape as a binary (black and white) image. The file **filename_sectors.tif** is opened (the segmented out sectors) to superimpose the sectors on the image in blue. The boxes **Show Sectors** and **Show Edge** can be toggled to show or hide the edge outline and the sector pattern found by the computer. The tools at the top of the window can be used to zoom into, pan around and rotate the image (standard symbols are used for each function). Edges can be moved using the **Move Edge Points** button. To smooth the edges, click on **Def. primary points** and select at least 2 primaries (for example tip and base: these must correspond to the same number and the same morphological feature for each image). Then click **smooth pts**, all the edge points will be smoothed around the 2 fixed primaries (these will not move). It is important that the edges are consistent from one image to another as they will be used to calculate the mean shape and the warping of the sectors. When the edge points are judged to capture the petal outline satisfactorily, the same button (now named **Save New Edge**) should be clicked to save the edited outline under **edge_xy.mat**. The primary points are saved under **primary_landmarks.mat**.

The rest of the tools can be used to edit the sectors. The superimposed blue sectors should be found in the same places as the red sectors on the petal image. If there is no blue sector where there is a red sector, the sector can be drawn in by clicking **Add Sector**, then clicking around a polygon (double click to finish) of the correct shape and size where you want the sector to be. If there is a blue sector where there is no red sector, the erroneous sector can be removed by clicking **Clear Region**, then drawing a box around sectors to be removed (double click to finish). If two or more blue sectors are found across the area of one red sector, the sectors can be joined by clicking **Merge Sectors**. This makes the image window into a black and white image of the sectors (white) against a black background. Click once in each sector and then press **Enter** to fill in the gaps between the sectors so they appear as one sector. Any changes that have been made to the sectors should be saved by clicking **Save Changes**. Clicking **Undo** will revert to the previous saved sector pattern.

To move onto another image, either click the >> or << arrows below the image window to go one forward in the list or one back in the list, or click the downward arrow beside the list to select any image from the **Staged** folder.

SectorEditor

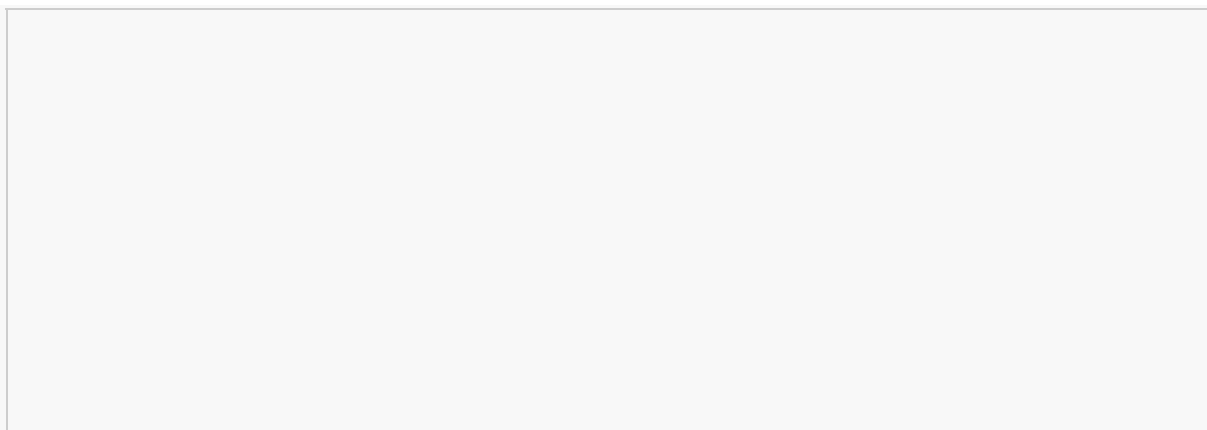


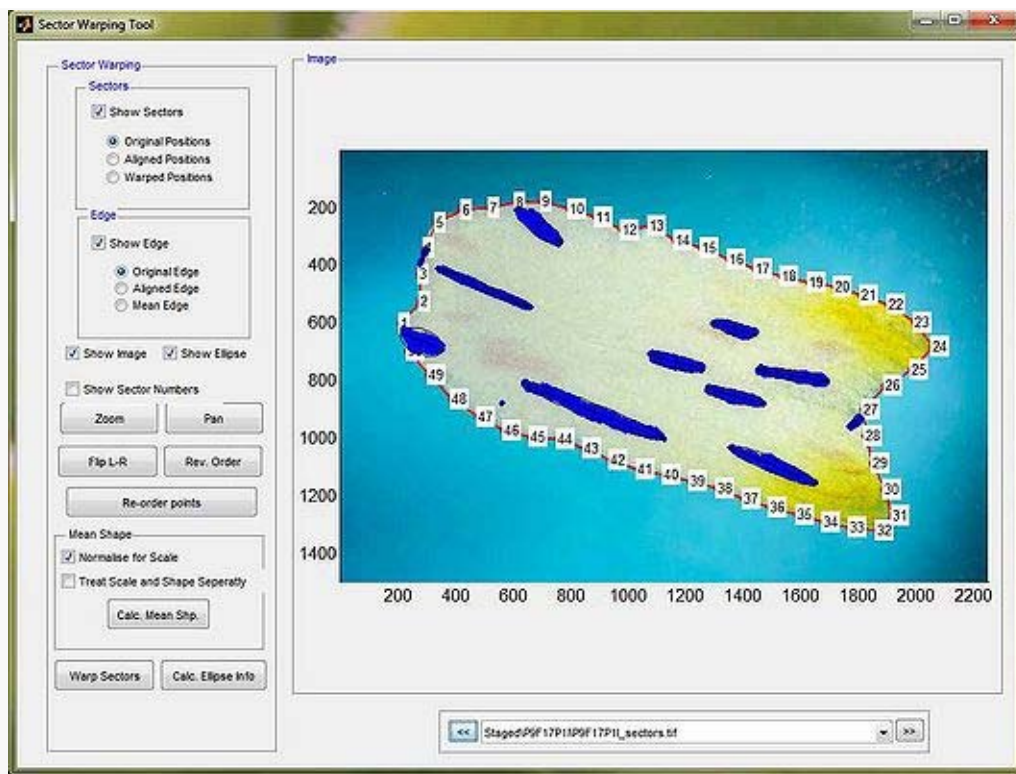
4 Sector and Organ warping

In this section we are going to use our three segmented petal outlines to calculate a **mean** petal shape. Then we are going to warp the segmented sectors onto the mean shape.

Select the **Sector Warping** button. This opens a new window entitled Sector Warping Tool.

Sector Warping



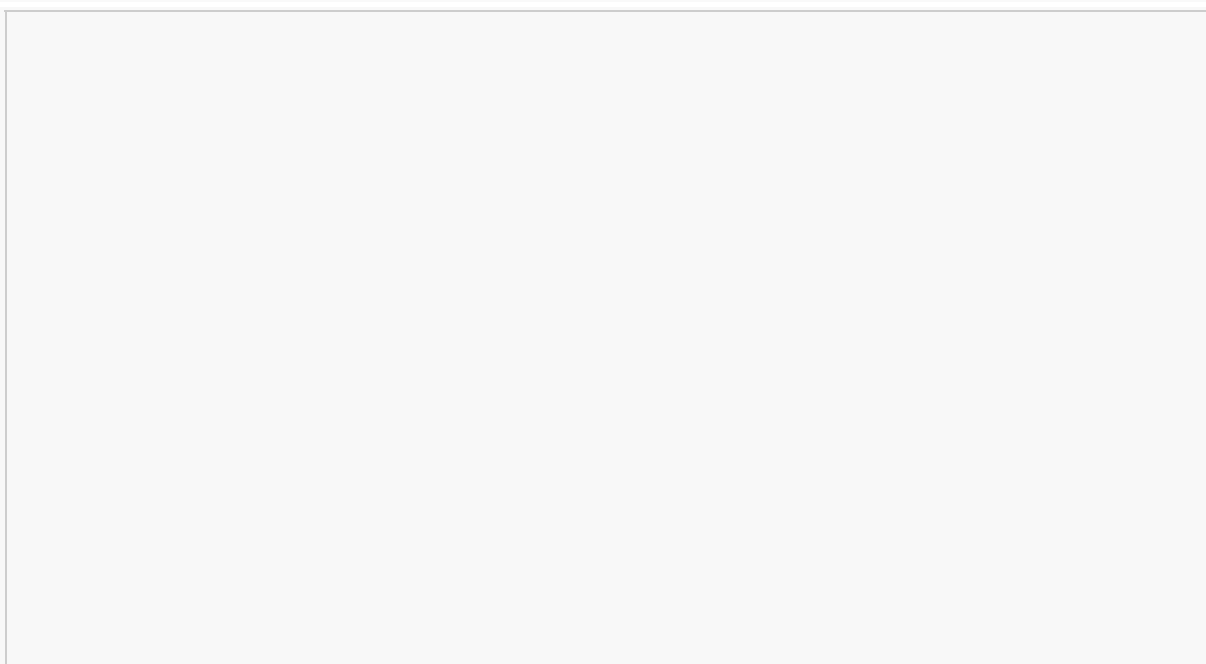


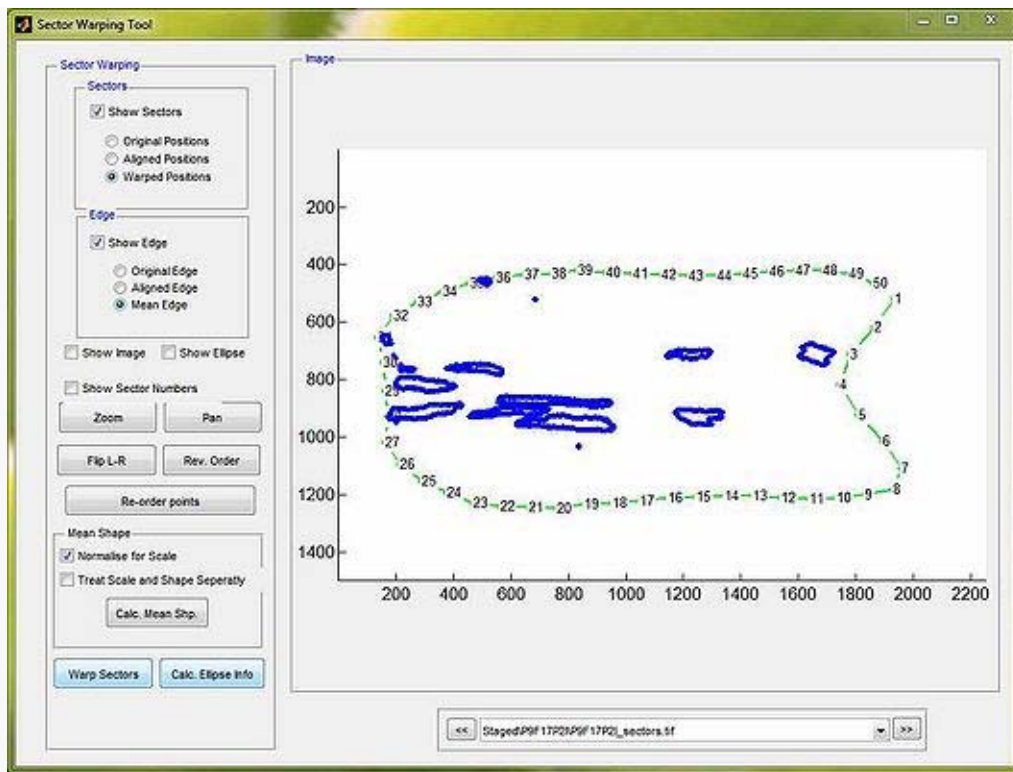
The first image from the **Staged** folder should be displayed in this window. The outline of the petal section and the sectors are marked on the image in the same way as they were in the Sector Editor Tool – now, the edited edge points and sectors should be displayed. As in the Sector Editor Tool, the boxes **Show Sectors** and **Show Edge** can be toggled to show or hide the edge outline and the pattern of blue sectors. The image can be shown or hidden by ticking or making blank the **Show Image** box. Once the sector ellipses have been calculated, each sector is also displayed as a red ellipse. The sector ellipses can be shown or hidden by ticking or making blank the **Show Ellipse** box. Note that the **view image** button can be unticked to speed up the process of going through the different files.

When all images have been checked so they have the points running in the same order around their outlines, the button **Calc. Mean Shape** should be clicked, followed by **Warp Sectors** (user selects the images to be warped and chooses the appropriate warping algorithm), then **Calculate Ellipse Info** (user selects images to be processed, and also the algorithm to be used (choose AIH for the closest fit of ellipses)). These buttons calculate the mean organ shape of the dataset (information saved under mean_shape.mat in the **Data** folder), then warp the sectors of every image to fit this mean shape. Then, the sector ellipses are calculated for every image in the **Staged** folder. Warped information is saved in the structure file warped_sectors_info.mat.

By selecting the **Mean Edge** selection bullet the mean petal shape is displayed (as a green outline) and selecting the **Warped Positions** selection bullet, the warped sector patterns are shown on the mean shape.

Mean petal shape and warped sector positions





modified on 12 March 2012 at 16:03 ••• 305 views

University of East Anglia / School of Computing Science / Powered by MediaWiki