

GfTbox: a tool for modelling and visualising leaf and petal development

Richard Kennaway
School of Computing Sciences, University of East Anglia
jrk@cmp.uea.ac.uk

3 March 2009

1 User guide

1.1 What GfTbox is

GfTbox is a Matlab tool for modelling the growth and three-dimensional deformation of biological surfaces such as leaves and petals (hereafter called “leaves”). The name is short for “Growth factor toolbox”.

Growth is modelled by specifying a distribution of substances called *growth factors* or *morphogens* over the surface of the leaf. The concentrations or concentration gradients of these substances at a point determine the rate at which the surface grows or bends in the neighbourhood of that point.

1.2 Installation and startup

Locally at UEA and JIC, GfTbox is best installed over SVN. Instructions are on the JIC/UEA wiki.

Other users will receive it as a zip file called *Growth_Toolbox.zip*. This unzips to give a folder called *Growth_Toolbox*. Place this anywhere convenient.

However you obtain GfTbox, to use it for the first time, run Matlab and go to the *Growth_Toolbox\growth* subdirectory. Give the command `GfTbox`. The GUI for GfTbox should appear, looking like Figure 1.

GfTbox automatically puts all subdirectories of the *Growth_Toolbox* directory onto the Matlab command path. You may wish to give the `savepath` command at the Matlab command prompt in order to keep GfTbox on the command path, so that you can use its command-line facilities without running the GUI.

This document is *Growth_Toolbox\docs\gfTbox.pdf*. It can also be opened by the **Open manual** command on the **Help** menu. The manual is updated less frequently than the program itself.

The GfTbox window contains a control panel on the left and a graphic area on the right, where the leaf will be drawn. Because there are too many controls to display at once, the **Select Tool** subpanel contains a set of radio buttons which select different subpanels to be displayed immediately below.

1.3 Workflow

To create a new mesh from scratch, select **Mesh editor** in the **Select Tool** panel, to make the mesh editor panel visible. You can either make a new leaf from scratch, or load an existing leaf.

To make a new leaf, click the **New** button. This will create a leaf of one of a predefined set of shapes, selected from the menu to the right of the buttons.

To load an existing leaf, you can use the **Load. . .** button near the top of the window. In the resulting dialog, select a directory containing a previously saved model. However, it is usually more convenient to load projects from the **Projects** menu. The **Set User Projects Folder...** command lets you nominate a folder within which to store all

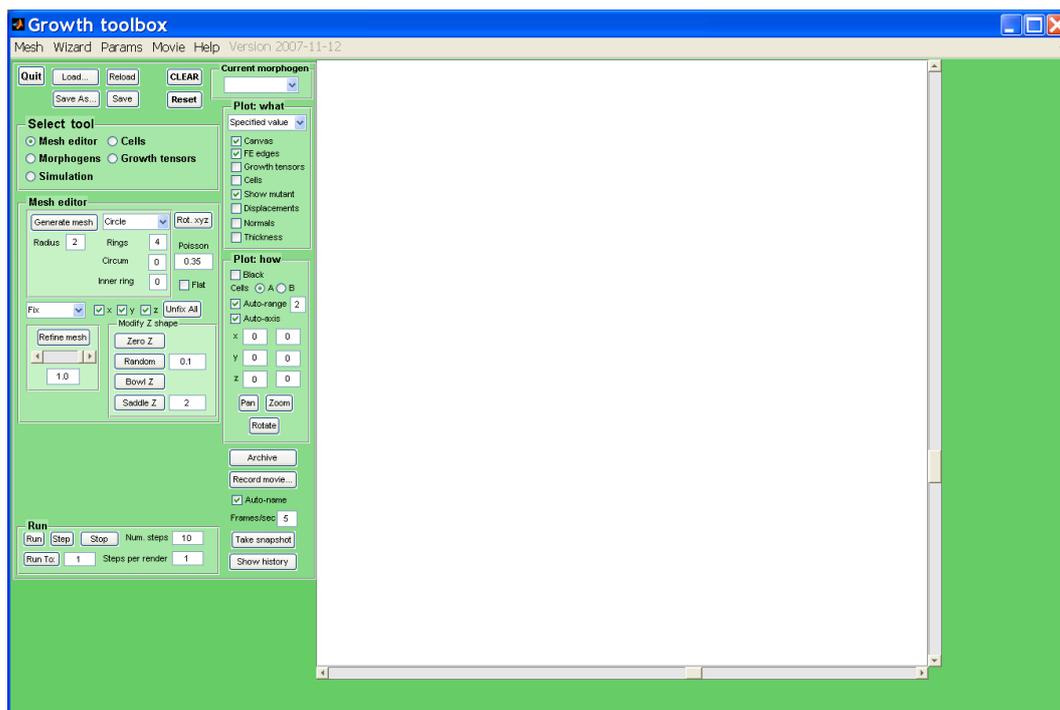


Figure 1: GFTbox on startup

of your projects. All GFTbox projects found within that folder will be made available through the Projects menu, allowing to to

To save a leaf, use the **Save As...** button. A leaf is always saved as a directory, containing all the files relevant to that leaf. Such a directory is called a *model*. To save a leaf again, after having modified it, use the **Save** button. This will overwrite the previously saved version. **Restart** reloads the previously loaded version, discarding any changes made.

To apply morphogens to a leaf, use the **Growth factors** panel.

Biological cells can be added to the mesh by the **Cells** panel.

In some specialised situations, you may have growth tensor data available for a leaf. Such data can be loaded via the **Growth tensors** panel. (Warning: this is a little-used feature that nobody has used for a long time. It may not work.)

The **Simulation** panel allows some parameters of the simulation to be set.

Having created or loaded a leaf and set its morphogens, biological layer, and simulation options as desired, its development can be simulated by the controls in the **Run** panel at the foot. (This panel is always visible.) The simulation can be run for a specified number of time steps, a single step, or until the area of the leaf reaches a specified multiple of its initial area. The **Stop** button will terminate the run (but it may take an iteration or two before the program notices).

What is plotted in the picture area depends on the settings in the **Plot options** panel.

The group of buttons at the top left are responsible for loading and saving projects.

1.4 Model structure

A *model* or *project* is a folder containing files that all relate to a single leaf model. If the folder is called *amodel*, then it will contain a MAT-file called *amodel.mat*. This is a binary Matlab file which contains the Matlab representation of a leaf. This is the only file that must exist; the existence of this file, with the same base name as the containing folder, is what tells GFTbox that this is a project directory.

The following other files and folders may also be present.

1. Files with names of the form *amodel_sNNNdNNN.mat*, where the N's are digits. These are files saved at stages of the simulation later than the initial state. The suffix *NNNdNNN* represents a simulation time of NNN.NNN.
2. A MAT-file called *amodel_static.mat*. This contains those parts of the leaf structure which apply to every saved file within the project. This is almost everything except the geometry of the mesh, the current distribution of growth factors, and the current time.
3. An M-file called *amodel.m*. This is a text file containing a user-written Matlab function which specifies how morphogens interact. (See section 4.)
4. The subfolder *movies* will contain any movies recorded from the program.
5. The subfolder *snapshots* will contain still pictures capturing moments in development.

The Load... menu command invites the user to select a model directory, within which subsequent files will be saved.

All dialogs for loading or saving files should automatically open in the appropriate model subfolder, if there is a current model.

2 Creating, loading and saving projects

GFtbox is distributed with a set of ready-made demonstration projects. These can be accessed through the Projects/Motifs menu. This menu displays all of the projects within the directory *Growth_Toolbox/Motifs*. Whenever one of these projects is opened, it is automatically saved as a copy into the default directory for your own projects, to avoid corrupting the originals.

To create a new project, select the Mesh editor button in the Select Tool panel. This makes the Mesh editor panel visible. From the pull-down menu near the top of the panel, select a mesh type and click the New button. A new mesh of the required type will be created. Having made a mesh, clicking the Save As... button will let you save it as a project. In the resulting dialog, create a new directory whose name is the name you want to give the project. With that directory selected in the dialog, click the OK button. You can open the resulting folder in Windows Explorer with the menu command Projects/Open Current Project Folder. (This is not implemented on Linux. I will implement it when I discover (a) how to tell from within Matlab that I'm running on Linux, and (b) how to open a Linux folder from Matlab.)

If you have saved the new project somewhere within your user projects folder, you should find that it then appears on the Projects menu.

3 Mesh creation and editing

In the Select Tool panel at the top, select the Mesh editor button. This will make the Mesh editor panel visible. (Figure 2.) This panel contains all the tools for creating a mesh and modifying its geometry.

3.1 Mesh creation

Mesh creation is performed by the Mesh editor panel, As mentioned in the previous section, a mesh is created by the New button. The mesh will be of the type selected in the pull-down menu to the right of that button, and various parameters can be set by the text boxes just below it. The new mesh will be drawn in the picture area. Below the picture is a text area containing summary details about the mesh, relating to its size and the progress of the simulation.

These are the available shapes and their parameters:

Circle Radius sets the radius (in arbitrary units). Rings sets the number of rings of triangles that the circle is divided into. Circum sets the number of vertexes around the circumference of the circle. A value of 0 will

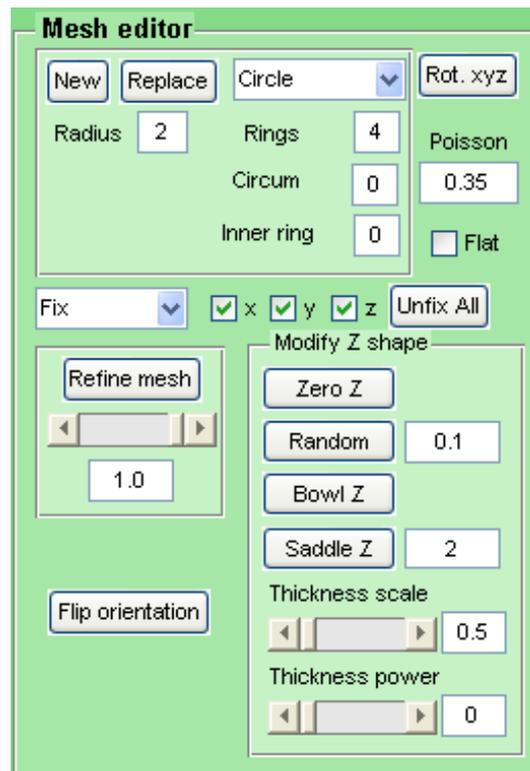


Figure 2: The mesh editor panel

give the default, which is 6 times the number of rings. The value must be at least 4, and for best results should be at least 4 times the number of rings. Inner ring specifies the number of triangles in the innermost ring. A value of zero will give the default, which is the maximum of 4 or the circumference divided by the number of rings. If a value is specified, it must be at least 4.

Semicircle Radius sets the radius (in arbitrary units). Rings sets the number of rings of triangles that the semicircle is divided into. Unlike Circle, there are no Circum or Inner ring parameters.

Hemisphere The parameters are the same as for a circle.

Lobes A lobe is a semicircle on top of a rectangle. Radius and Refinement specify the semicircle. (Refinement is the equivalent of Rings.) Height specifies the height of the rectangular part, as a proportion of the diameter of the semicircle. Lobes specifies the number of lobes. They are joined together in a strip, edge to edge.

Lune (Not implemented yet. A lune is a pointed ellipse.)

Rectangle Width and height are specified by X width and Y width. The number of divisions of the mesh in each direction are specified by X cells and Y cells.

One element This makes a mesh consisting of a single triangular finite element. This exists primarily for testing the Finite Element code.

Cylinder This makes a cylinder with a specified radius, height, and number of subdivisions around and vertically.

Cup This makes a cylinder with a hemispherical cap on the lower end. The parameters are as for a cylinder, plus two extra ones. Base ht specifies the height of the cap as a proportion of the cylinder radius (so 1 gives a hemisphere and 0 a flat end). Base rg gives the number of rings of triangular finite elements that the cap is divided into.

Cap This is like Cup, but the end cap is on the top end. The parameters are the same, replacing Base by Top.

Capsule This combines Cup and Cap to give a cylinder with a cap on both ends. It has all the parameters of Cup and Cap.

Snapdragon This makes a flower consisting of a number of lobe-shaped petals joined together into a tube closed at the foot by a hemispherical cap. The parameters are the same as for **Lobes**, except that **Radius** now refers to the radius of the cylinder, not the radius of the individual lobes. The lobe radius will be half the circumference of the cylinder, divided by the number of lobes. In addition, it has the same **Base ht** and **Base rg** parameters of **Cup**.

Instead of creating a mesh of one of these predefined shapes, you can import a mesh defined in an external file. There are several file formats that may be used to describe a mesh. These formats are created when saving a mesh to a file, described in section 3.4.

MAT files. These are Matlab binary files containing the current state of the mesh (including morphogens, command history, everything). Such files are created by saving the mesh to a MAT file.

M files. These are Matlab text files containing Matlab commands that will create or modify a mesh. For more information on the Matlab commands that create and modify meshes, see sections 13 and 15.

OBJ files. These are text files specifying the coordinates of all the mesh vertexes and the sets of vertexes that are the triangles of the mesh. Each line of an OBJ file begins with either `v` or `f`. All of the former should precede all of the latter. `v` is followed by three floating point numbers, the x , y , and z coordinates of a vertex of the mesh. `f` is followed by three positive integers, which are the indexes of the three vertexes of a triangle. The number n references the vertex defined in the n th line beginning with `v`. The triangles do not have to be consistently oriented, but the surface as a whole must be orientable. No Klein bottles or Moebius bands!

3.2 Modifying meshes

There are several ways to modify the geometry of a mesh.

The **Zero Z** button will flatten the mesh by setting the Z coordinate of every node to zero.

The **Random** button will displace every vertex by a random amount perpendicular to the surface. The amplitude of the random perturbation is given by the value in the box to the right of the button.

The **Bowl Z** button will add to the z coordinate an amount proportional to the square of the distance from the centre, giving a bowl shape. The amplitude is specified in the same way as the the **Random Z** button.

The **Saddle Z** button adds a saddle-shaped deformation to the z coordinates. The amplitude is specified as before, and the number of waves is given by the value in the lower text box.

The **Thickness** subpanel determines how the thickness of the mesh is modelled. If the **Physical** checkbox is unchecked, then the thickness of the mesh will be determined by the **Initial** and **Scaling** controls. The thickness of the mesh will be uniform, and equal to $K\sqrt{(A_0)}(A/A_0)^{P/2}$, where K is the value of the **Initial** slider, P is the value of the **Scaling** slider, A_0 is the initial area of the leaf, and A is its current area. So a scaling value of zero will cause the leaf to have a constant absolute thickness throughout the simulation, while a scaling of 1 will make the thickness grow proportionally to the linear size of the leaf.

If **Physical** is checked, then the initial thickness of the mesh is set by the **Initial** slider to $K\sqrt{(A_0)}$, but its subsequent growth is determined by the **KNOR** growth factor.

If one wants a mesh to remain flat and deform only in the xy plane, click the **Always flat** checkbox. The z coordinate of every mesh point will be set to zero, and will be forced to remain zero throughout every computation. If a deformation is applied to a mesh marked as always flat, you will be asked for confirmation that you want to allow the operation, whereupon it will be allowed to deform in three dimensions..

The **Refine mesh** button causes a proportion of the edges of the mesh to be split, giving a finer mesh. The slider and text box below the button specify the proportion of edges to be split.

The **Rotate xyz** button will rotate the mesh so that what was its x axis now points along the y axis, y along z , and z along x . Clicking again will rotate x to z , z to y , and y to x . Clicking a third time will rotate the mesh back to its original orientation. This can be useful if you want to establish a vertical gradient of a morphogen.

The **Poisson's ratio** text box specifies the degree to which the mesh can withstand shearing forces. The value must be greater than 0 and less than 0.5. As it approaches 0.5, the resistance to shearing drops towards zero. The default value is 0.35, which is typical of many materials. In practice, the behaviour of the simulation is not much

affected by the value. Values very close to 0 or 0.5 may result in poor performance, as the equations of elasticity become ill-conditioned and approach singularities.

It is possible to constrain individual vertexes so that they will not move in either the x , y , or z directions. Select **Fix** on the menu just below the panel containing the mesh generation controls, and three checkboxes will appear alongside, labelled x , y , and z . Use the checkboxes to select the directions in which you want to constrain some vertexes, and then click on the mesh to select the vertexes you want to constrain. With all the checkboxes checked, the selected vertexes will be completely immobilised. The vertexes whose constraints are exactly those specified by the checkboxes are highlighted. Clicking a highlighted vertex will remove all of its constraints; clicking an unhighlighted vertex will apply the selected constraints.

Clicking the **Unfix All** button will remove all constraints (but if **Always flat** is checked, the mesh will still be constrained to be flat).

To delete parts of the mesh, select **Delete element** on the same menu, and then click on the mesh. Finite elements clicked on will be deleted. Be careful with this: there is no undo for this operation, and no way to add elements to the mesh.

The **Flip orientation** button does almost nothing visible and can be ignored. It actually inverts the orientation of every triangle in the mesh, and swaps the A and B sides.

3.3 Saving models

To save a mesh as a new project, click the **Save As...** buttons. The first time you save a mesh, you will be asked to say where to create a new model folder. The mesh will be saved there as a *.mat* file.

A model folder holds all of the files associated with a mesh. Initially, this is just the *.mat* file, but it may also contain a *movies* subfolder where recorded movies will be stored, and a *snapshots* subfolder for image snapshots. These subfolders will be automatically created when necessary. There may also be an interaction function (see section 4) defined in a *.m* file.

When a mesh has been saved as a model folder, the **Save** button will save the current stage of the mesh. If no simulation step have been performed, this overwrites the initial mesh, but otherwise, a new file is saved in the project folder whose name has a suffix of the form “*_sNNNdNNN*” added, where the current simulation time is *NNN.NNN*. These files, called *stage files*, are listed in the **Stages** menu, from where they can be reloaded.

If you have a current project open, the **Save As...** button will save its initial state as a new project.

3.4 Saving meshes to files

A mesh can also be saved to a file instead of a model directory, using the various **Save...** commands on the **Mesh** menu. They can be saved in a number of different formats.

M files. These are Matlab script commands. For every action in the GUI, there is a corresponding Matlab command. The history of all the actions the user takes in the GUI is recorded as a series of these Matlab commands. When the mesh is saved as an M-file, the file will contain these commands. For further information about the Matlab command equivalents of the GUI actions, see sections 13 and 15.

MAT files These are Matlab binary files containing the current state of the mesh.

OBJ files. These are text files which contain a description of the current state of the mesh. Although they are text files, the format is for practical purposes not human-readable.

FIG files. A FIG-file contains, not the mesh, but a graphical plot of the mesh. A FIG-file can be opened in Matlab, and the figure zoomed and rotated, but it does not contain the mesh itself, only a 3D graphical representation of it. Unlike the other formats, a FIG-file cannot be loaded into GFtbox with the **Load** button.

4 Morphogen distribution and interaction

Morphogens are edited by the **Growth factors** panel shown in figure 3

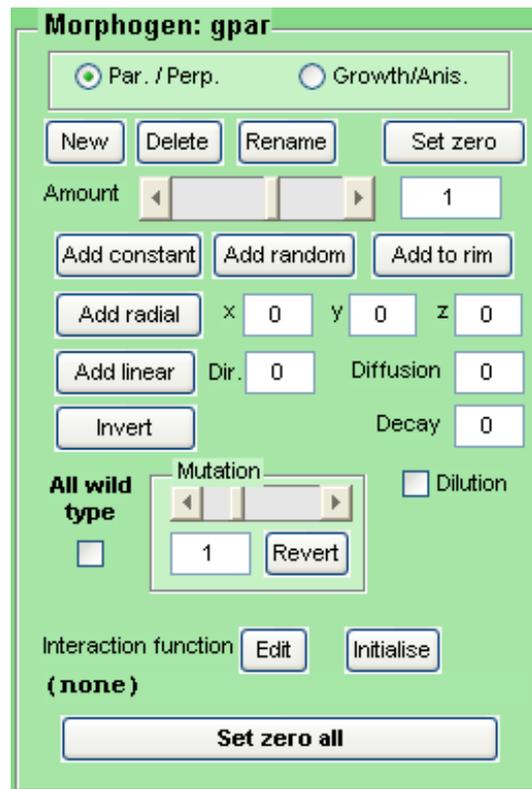


Figure 3: The Growth factors panel

A morphogen is a substance whose concentration may vary over the mesh. In the finite approximation we make to a continuous surface, morphogen values are stored at each vertex, and should be imagined to be linearly interpolated over each triangular element.

A mesh may have any number of morphogens. Seven are predefined, and have specific effects on growth. The user may add any number of others, and define ways in which morphogens locally interact.

These are the seven standard morphogens and their effects on growth:

KAPAR, KBPAR: These determine the local rate of linear growth on the A and B sides of the mesh in the direction of the gradient of *polariser*.

KAPER, KBPER: These determine the local rate of linear growth on the A and B sides of the mesh perpendicular to the direction of the gradient of *polariser*.

KNOR: These determine the local rate of linear growth perpendicular to the surface of the mesh.

POLARISER: The gradient of this morphogen determines the directions of maximum and minimum linear growth as just described.

When the gradient of *polariser* is zero, growth in all directions parallel to the surface is the average of *gpar* and *gperp*.

STRAINRET: This determines how much of the residual strain is retained from one time step to the next. For most purposes this should be left at zero.

ARREST: This determines whether biological cells are allowed to split. Where the *ARREST* morphogen has a value of 1 or more, no splitting of biological cells is allowed.

The KAPAR, KBPAR, KAPER, and KBPER factors can be specified in an alternative way, accessed by the Mesh/Change Morphogen Version menu. The A/B menu item corresponds to the growth factors just described. The K/BEND menu item replaces those factors by these:

KPAR: The average growth of the A and B sides of the mesh parallel to the polariser gradient.

KPER: The average growth of the A and B sides of the mesh perpendicular to the polariser gradient.

BENDPAR: Half the difference between the growth on the B side and A side, parallel to the polariser gradient.
Positive means the B side is growing faster.

BENDPER: Similarly, perpendicular to the polariser gradient.

For historical reasons, another standard morphogen appears in K/BEND mode, called NOTUSED. This growth factor is not used. KNOR is also called THICKNESS instead but has the same effect, and some of the factor appears in a different order.

Any mesh can be converted back and forth between these two versions. The formulas relating them are:

$$\begin{aligned}KPAR &= (KAPAR + KBPAR)/2 \\KPER &= (KAPER + KBPER)/2 \\BENDPAR &= (KBPAR - KAPAR)/2 \\BENDPER &= (KBPER - KAPER)/2 \\KAPAR &= KPAR - BENDPAR \\KAPER &= KPER - BENDPER \\KBPAR &= KPAR + BENDPAR \\KBPER &= KPER + BENDPER\end{aligned}$$

When a mesh is created, the standard morphogens are set to have a concentration of zero everywhere. The distribution of each morphogen can be modified through the **Growth factors** panel.

At the top right of the main control panel there is a pull-down menu of morphogen names. (The menu will be empty until a mesh has been created.) This selects which of the morphogens the other controls operate on. The standard morphogens are prefixed by a *. In the picture of the mesh, the concentration of the current morphogen is colour-coded. Zero is white, increasing positive values are blue, green, yellow, orange, and red, and increasing negative values are violet. If the **Auto-range** checkbox in the **Plot options** panel is ticked, the colour scale is fitted to the range of the current morphogen. Otherwise, the user can specify the maximum and minimum values of the scale. The maximum absolute value on the colour bar is represented by deep red (if positive) or deep violet (if negative), and the rest of the colour bar is scaled to fit.

In K/BEND mode, the colour-coding of BENDPAR and BENDPER is different, since these, unlike the other morphogens can legitimately take either sign. Green, cyan, and blue represent negative values, and yellow, orange, and red positive values.

There are two ways of modifying the level of a morphogen. One is to click on the mesh to add an amount of the morphogen at a single vertex. The amount added is specified by the **Amount** slider and text box. Holding down the shift key while clicking will subtract instead of adding. Note that morphogens can be negative. The other method is to add a predefined distribution of the morphogen over the whole mesh, by means of the buttons. The amplitude of the distribution is the **Amount** value.

Add constant: Add the specified amount to every vertex of the mesh.

Add radial: Add an amount to each vertex, proportional to the square of the distance of that vertex from the position indicated by the **x**, **y**, and **z** boxes. The maximum amount is the specified amplitude.

Add linear: Add a linear gradient of morphogen. The range will be from zero to the specified amplitude. The direction of the gradient is given by the **Direction** text box in degrees. (0 degrees is the positive *x* axis, 90 degrees is the positive *y* axis.) The gradient is always in the *xy* plane. If you want a gradient in the *xz* plane, click the **Rotate xyz** button in the **Mesh editor** panel to rotate the mesh so that what was the *xz* plane is now the *xy* plane. Then add the gradient you want, then click the **Rotate xyz** button twice more to restore the original orientation.

Add to rim: Add the specified amount to every vertex on the rim of the mesh (including vertexes bordering internal holes).

Add random: Add a random amount to every vertex of the mesh. The random amount is uniformly distributed between zero and the specified amplitude, and is chosen independently for every node of the mesh.

Set zero: Set the morphogen to zero everywhere.

Set zero all: Set *all* morphogens to zero everywhere, as well as their diffusion and decay constants (see below).

Each morphogen may have the capability of diffusing through the mesh. The diffusion coefficient is specified in the Diffusion text box. A value of zero means that it does not diffuse. This is the initial default.

Each morphogen may also decay with time, at a rate set by the value in the Decay text box. The result is an exponential decay to zero. The higher the value, the faster it decays.

The Dilution checkbox causes the current growth factor to behave as a conserved substance: expansion of the mesh will reduce its concentration in proportion. The default is off: expansion has no effect on concentration.

The On split radio buttons specify how to determine the new value of a morphogen at the midpoint of an edge that has been split. Either the average value of the two ends, the minimum, or the maximum can be chosen. The average value is generally the most physically meaningful, but where a growth factor is being used to define the identity of vertices, the minimum may be more suitable.

The Set all zero button sets all growth factors everywhere to zero.

The Use wild type button causes all mutant levels to be ignored.

To cause the value of a morphogen at a vertex to remain fixed at its current value, regardless of diffusion and decay, control-click the vertex. Control-click the same vertex again to let the value vary. A vertex fixed at a high value will thus act as a source, and a vertex fixed at zero will act as a sink. A combination of a source, diffusion, and decay can be used to create a morphogen localised to a region around its source, which remains of fixed size even while the leaf grows.

To create a new morphogen, click the New button in the morphogens panel. A dialog will appear asking for the name of the new morphogen. The name of the new morphogen, e.g. *foo*, will be added to the Displayed m'gen menu. Its initial value is zero everywhere, with zero diffusion and decay rate.

To delete the currently selected morphogen, click the Delete button. The user will be asked for confirmation. The built-in morphogens cannot be deleted.

To rename the currently selected morphogen, click the Rename button. The user will be asked for confirmation. The built-in morphogens cannot be renamed, nor can a morphogen be renamed to have the same name as another morphogen.

5 The Interaction Function

To specify interactions between morphogens, click the Edit button on the Interaction function panel. An M-file will be created in the model directory. Its name will be the name of the model. A standard outline of a morphogen interaction function will be written to the file, which will then be opened in the editor.

Clicking the Edit button later will reopen the file in the editor.

When you add, delete, or rename morphogens, the interaction function will automatically be rewritten, so that the boilerplate code at the top and bottom will refer to the new set of morphogen names. Any code of your own which accesses renamed or deleted morphogens will have to be manually edited as necessary. You must save your changes to disk before doing anything that causes the file to be regenerated, otherwise your changes will be lost. Besides adding, deleting, and renaming morphogens, the file is also regenerated every time you load or reload the project, or use the Edit or Rewrite button.

Here is a typical interaction function. We assume there are two user-defined morphogens, called *foo* and *rad*.

```
function m = snapdragon( m )
%m = snapdragon( m )
% Morphogen interaction function.

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

```

if isempty(m), return; end

growth_i = FindMorphogenIndex( m, 'growth' );
G = m.morphogens(:,growth_i);
polariser_i = FindMorphogenIndex( m, 'polariser' );
P = m.morphogens(:,polariser_i);
anisotropy_i = FindMorphogenIndex( m, 'anisotropy' );
A = m.morphogens(:,anisotropy_i);

curl_i = FindMorphogenIndex( m, 'curl' );
curl = m.morphogens(:,curl_i);
curlpolariser_i = FindMorphogenIndex( m, 'curlpolariser' );
curlpolariser = m.morphogens(:,curlpolariser_i);
curlanisotropy_i = FindMorphogenIndex( m, 'curlanisotropy' );
curlanisotropy = m.morphogens(:,curlanisotropy_i);
arrest_i = FindMorphogenIndex( m, 'arrest' );
arrest = m.morphogens(:,arrest_i);
foo_i = FindMorphogenIndex( m, 'foo' );
foo = m.morphogens(:,foo_i);
rad_i = FindMorphogenIndex( m, 'rad' );
rad = m.morphogens(:,rad_i);
%%% END OF AUTOMATICALLY GENERATED CODE.

%%% YOUR CODE BEGINS HERE.

...

%%% END OF YOUR CODE.

%%% AUTOMATICALLY GENERATED CODE: DO NOT EDIT.
m.morphogens(:,growth_i) = G;
m.morphogens(:,polariser_i) = P;
m.morphogens(:,anisotropy_i) = A;
m.morphogens(:,curl_i) = curl;
m.morphogens(:,curlpolariser_i) = curlpolariser;
m.morphogens(:,curlanisotropy_i) = curlanisotropy;
m.morphogens(:,arrest_i) = arrest;
m.morphogens(:,foo_i) = foo;
m.morphogens(:,rad_i) = rad;
end

```

All of the above code is automatically generated. To make this function do something useful, you need to add code in the section marked YOUR CODE BEGINS HERE.

The first line declares a Matlab function called *snaptdragon*. It takes one argument and returns one result. The argument is a structure containing the current state of the mesh. The result of the function is the new state of the mesh.

The subsequent lines down to the beginning of your code define one variable for each morphogen, holding the values of that morphogen at every node of the mesh.

Your code can then modify the values of these variables.

The final boilerplate section copies the variables back into the appropriate place in the mesh structure.

Here are some examples of what might appear in your code.

1. `G(:) = 0;`

This will set the growth morphogen to zero everywhere, on every iteration.

2. `G = foo_p .* 2;`

This will set the concentration of the growth morphogen to twice that of the *foo* morphogen, on every iteration. This overrides whatever distribution of growth morphogen may have been set through the GUI.

```
3.      G = foo_p .* rad_p;
```

This will set the concentration of the growth morphogen to the product of those of the *foo* and *bar* morphogens.

```
4.      G = G .* 1.1;
```

This increases the concentration of growth morphogen by 10% on every iteration.

```
5.      if Steps(mesh) < 40
          G = G .* 1.1;
        end
```

This gives the same exponential increase of growth morphogen, but stops after 40 steps.

```
6.      if Steps(mesh) > 40
          arrest = 1;
        end
```

This will prevent all biological cell division after 40 steps. (ARREST IS NOT YET IMPLEMENTED.)

The interaction function can in principle change the mesh in any way whatsoever: it is not limited to changing the morphogens. However, one must understand what one is doing before changing any other part of the mesh data structure.

If the interaction function sets `m.stop` to 1, then this will cause the simulation to stop after the current iteration. (The toolbox will automatically then set `m.stop` back to 0.)

The **Enable** checkbox turns the interaction function on and off.

If the interaction function causes an error, it will be automatically disabled and the controls on the **Interaction** function panel will turn red. Normal operation can be resumed by clicking the **Edit** button (GFtbox assumes that you will fix the error) or the **Reset** button.

6 The notes file

The **Interaction** function panel also contains a button called **Notes**. This creates a text file in the project directory whose name is the model name, and opens it in the editor. In this file you can keep a record for your self of the work that you do on a model. You can type anything you like into it; GFtbox never does anything with it except open it when you click the button.

When you save a project as a new project, the notes file is copied across and automatically opened in the editor.

7 Growth tensors

Instead of driving growth by morphogens, it is possible to specify growth tensors directly. This is only supported for flat meshes.

WARNING: Nobody has used this for a long time, and it may not work.

Growth tensors are imported through the **Growth tensors** panel shown in figure 4.

A growth tensor describes the way a small region of the mesh (a single Finite Element) is growing at a single point in time. There are several different ways of representing this numerically, but they all require three numbers. Two representations are supported.

1. The three numbers are:

- The number of time steps that it would take the region to double its area.



Figure 4: The Growth tensors panel

- The ratio of the rate of growth in the fastest growing direction to the rate of growth in the slowest growing direction.
 - The angle of the fastest growing direction from the positive X axis. (A positive angle turns towards the positive Y axis.)
2. The three numbers are:
- The rate of growth in the fastest growing direction. (E.g. a value of 0.1 would mean a growth of 10% over some standard time interval.)
 - The rate of growth in the slowest growing direction.
 - The angle of the fastest growing direction from the positive X axis.

GFtbox attempts to guess from the data which interpretation of the three numbers is more likely.

A mesh with growth tensors can be loaded from an OBJ file. Such a file specifies the locations of all the vertexes, the triples of vertexes that make the finite element triangles, and for each finite element, a growth tensor.

A new set of growth tensors for the current mesh can be loaded by the **Load growth...** button on the **Growth tensors** panel. This loads an OBJ file just like the one loaded by the **Load...** button, but ignores all information except the growth tensors. For this to work, the topology of the mesh must be identical to that contained in the OBJ file.

When a new set of growth tensors is loaded, although the mesh geometry in the file is discarded, the total area of the mesh is calculated. The ratio of that area to the original area of the currently loaded mesh is calculated, and that value is inserted into the text box beside the **Run To...** button on the **Simulation** panel.

This allows GFtbox to perform regrowth of ungrowth data, by means of the following steps. We assume that we are given a series of files which represent the mesh and its growth tensors at successive stages of growth.

1. For convenience, put all of the OBJ files into the *meshes* directory of a new model.
2. Load the file that represents the first stage (i.e. the earliest stage).
3. Load the growth tensors from the file that represents the second stage.
4. In the **Simulation** panel, make sure that the checkbox for **Use tensors** is on, and that for **Split edges** is off. Check that the value in the box beside the **Run To...** button is greater than 1. (If it is less than 1, that means that you are handling the files in the wrong order. Typically, the files are named *stage_1.obj*, *stage_2.obj*, etc., but sometimes *stage_1.obj* is the earliest stage and sometimes it is the final stage.)
5. Click the **Run To...** button. The mesh should grow until its area has increased by the specified amount.
6. Repeat stages 3 to 5 for each of the remaining OBJ files.

When you are done, you can save the mesh as a script file (use the **Save mesh...** button and select “M files” in the dialog). Running this script file will rerun the entire process automatically. If you look at the script file with a text editor, it should be clear how to modify it for other data sets, instead of having to go through the manual process every time.

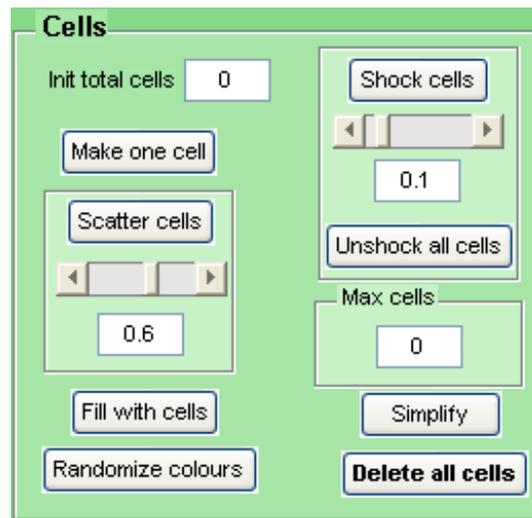


Figure 5: The Cells panel

8 Biological cells

Biological cells are simulated by adding a layer of cells painted onto the underlying finite element mesh and growing and moving with it. The biological layer can be created or modified through the **Cells** panel shown in figure 5.

The **Fill with cells** button will completely cover the surface with cells, rendered in shades of dark green. When this has been done, the **Shock cells** button will randomly turn a specified proportion of them light green, simulating the effect of a randomly distributed genetic change, such as the initiation of expression of GFP (green fluorescent protein). **Unshock all cells** reverts them all to dark green.

Alternatively, the **Scatter cells** button will randomly scatter a number of cells over the surface. One cell will be created in the middle of each randomly selected element of the finite element mesh. These cells are all coloured red, and the **Shock cell** button has no effect on them: they are intended to model some randomly chosen set of real cells, while avoiding the computational load of simulating an entire layer of cells.

The **One cell** button adds a randomly chosen cell to the current set of cells.

Delete all cells deletes the biological layer.

If splitting of cells during growth is enabled (by a checkbox on the **Simulation** panel), then as the biological cells grow with the mesh, they will divide according to the following rules. A cell splits when it has grown to more than $\sqrt{2}$ times the average area of the cells when they were initially created. The direction of the new cell wall depends on both the shape of the cell and the presence of a polarising morphogen.

If there is no polarising gradient, the shape of the cell is estimated by mathematically approximating the cell by an ellipse (using PCA). The new cell wall is then chosen to be perpendicular to the major axis of the ellipse, and pass through its centre. A small random perturbation is made to the direction (...WHICH THE USER SHOULD BE ABLE TO SPECIFY).

If there is a polarising gradient, the new cell wall will be either perpendicular to or parallel to that gradient, depending on the length of the projection of the cell onto each of those directions. If the cell is longer along the gradient, the new wall will be perpendicular to the gradient, and vice versa. There is again a small random perturbation to the direction.

Cell division can be disabled by turning off the **Split bio cells** checkbox in the **Simulation** panel.

Cell division can also be enabled or disabled by the **arrest morphogen**. Where this is 1 or more, cell splitting is disabled. (THIS IS NOT IMPLEMENTED YET.)

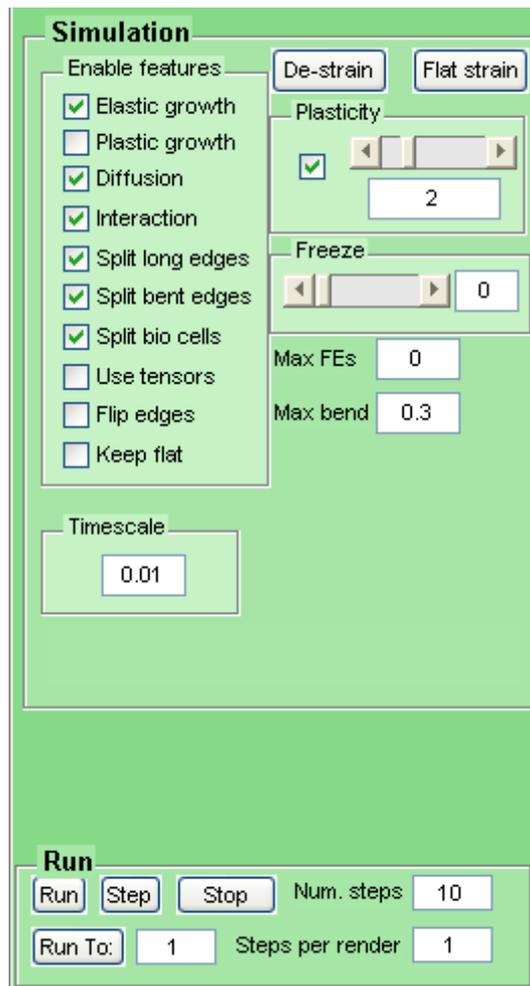


Figure 6: The Simulation and Run panels

9 Running a simulation

Once a mesh has been created and morphogens added to it, its development can be simulated, using the controls in the Run panel. These controls are always visible. Additional parameters of the simulation are accessible through the Simulation panel. These are shown in figure 6.

The Run for, Run until, Run to, and Step buttons, respectively do a specified number of steps, run to a certain time, run until the area has expanded by a certain multiple, and do a single step.

The Run panel turns red when the simulation is running.

As a rough guide to the meaning of a single step, if the concentration of growth morphogen is equal to 1 everywhere, then in one time step, the mesh will grow linearly by about 1%.

Run for, Run until and Run to are interruptible with the Stop button.

Most of the GUI controls can still be used during the simulation, although it may take one or two steps before they have a visible effect. E.g. enabling or disabling cell splitting, changing the plot options, etc. Operations which are not allowed during simulation will sound a beep or write a message to the Matlab command window.

Currently, the simulation runs tolerably fast for meshes of a few thousand finite elements and biological cells, gradually degrading with increasing numbers.

10 Simulation options

Several different computations take place during each simulation step. The controls in the Simulation panel allow some of these to be turned on and off.

Elastic growth, Plastic growth: At most one of these is active at once. These determine whether the effect of growth on the shape of the leaf is calculated using the equations for a compressible solid or for a compressible two-dimensional fluid. Elastic growth is generally more realistic. The main difference is that fluids do not buckle. If both checkboxes are turned off, no growth computations are performed. The default is elastic growth.

Diffusion: When turned on, this enables the calculation of diffusion and decay, for morphogens having positive values for either of these.

The default is on.

Retriangulate: This allow certain transformations of the mesh to be made in order to maintain numerical quality by avoiding long thin triangles.

Split long edges: This determines whether edges of the finite element mesh should be split when they become too long. The default is on.

Split bent edges: If the finite elements on either side of an edge of the mesh make too great an angle with each other, enabling this checkbox allows them to be split, so as to better approximate the shape of the underlying smooth surface in regions where its curvature is increasing. The threshold angle for splitting is set in the Max bend text box in radians. The default is off, as under some circumstances its behaviour is unstable.

Split bio cells: This determines whether biological cells are allowed to be split when they become too large. The default is on.

Use tensors: If on, use externally specified growth tensors instead of morphogens to determine the growth. The default is off.

Flip edges: When on, this can improve the subdivision of the mesh by detecting when the edge between two finite elements should be replaced by the edge joining those elements' opposite corners.. The default is off.

Internal rotation: Rotate the mesh by N radians before computing the Nth iteration, and rotate it back before displaying it. This helps to smooth out some numerical artefacts.

Negative growth: Allow the rate of growth specified by the growth factors to be negative. Negative growth might be considered biologically unrealistic.

The other controls in the Simulation panel are:

De-strain: Clicking this button will remove all residual elastic strain from the mesh.

Flat strain: Clicking this button will set the residual elastic strain to what it would be if the unstrained state of the mesh was flat.

Dissect: Clicking this button will cut the mesh along all its seams.

Explode: Clicking this button will move all the connected components of the mesh away from each other.

Flatten: Clicking this button will deform every connected component of the mesh until it is flat, while trying to minimise the distortion.

Freeze: This slider and text box set the proportion of computed growth that is not actually applied to the mesh. The normal value is zero; a value of 1 would prevent the mesh moving at all.

Max FEs: If positive, splitting of finite elements will be disabled when the number of elements reaches this figure. If zero, there is no limit on the number of FEs.

Max bend: This is the threshold angle for splitting the cells on either side of an edge, if Split bent edges is turned on.

Edge scaling: This specifies how the length threshold for splitting edges increases as a power of the size of the mesh. Zero means that the threshold does not change, while 1 means that it scales as the square root of the area.

Split margin: Mesh quality is better maintained if edges are not split one by one as they reach the threshold length, but are split in batches. If the value of split margin is m and the current edge length threshold is d , then no edge will be split unless some edge is above length $d \times \sqrt{m}$, and when that happens, every edge will be split at once, whose length is above d/\sqrt{m} . A value of 1.5 is suitable. In a uniformly growing mesh, any value above 2 will tend to split nothing until all of the edges have doubled in length, then split all of them at once.

Min. pol. grad.: Where the magnitude of the polariser gradient is below this value, growth is instead directed by the gradient which existed the last time it was above this value. If the gradient has never been above the threshold, growth is unpolarised. The areal growth rate is the same as it would be if there was an effective gradient, but the growth is isotropic.

When polariser gradient arrows are plotted, where the magnitude is above the threshold the arrows are drawn in dark blue. Where it is below, the “frozen-in” gradient is drawn in red.

Tol. diff.: Sets the tolerance for solving the diffusion equations. Default is 0.00001.

Tol. elast.: Sets the tolerance for solving the elasticity equations. Default is 0.001.

Time limit: Sets the maximum number of iterations allowed in the `cgs` solver.

11 Display of the mesh

The scrollbars on the picture vary the direction of the viewpoint.

Just to the left of the picture is the Plot options panel. The settings determine what information is plotted.

The three pull-down menus above the Plot options panel and the checkboxes next to them choose what quantity is to be represented by the colour of the mesh.

Plot current factor: The value of the current morphogen.

Plot output value: The quantity specified by the two menus just below the checkbox is plotted. The upper menu specifies a tensor quantity, and the lower menu specifies which attribute of that tensor to plot.

If both checkboxes are off, no quantity is plotted and the mesh is drawn in white.

The tensor quantities are:

Actual growth The amount of growth occurring in the last time step.

Actual bend The amount of bending occurring in the last time step.

Residual growth The difference between the amount of growth specified by the growth factors and the actual growth in the last time step.

Residual bend The difference between the amount of bend specified by the growth factors and the actual bend in the last time step.

Rotation The rotational velocity of each element.

The tensor properties are:

Total The total of the three principal components of the tensor.

Areal The total of the two principal components of the tensor corresponding to the two principal axes most nearly parallel to the surface.

Major The larger of the two principal components most nearly parallel to the surface.

Minor The smaller of the two principal components most nearly parallel to the surface.

Parallel The principal component most nearly parallel to the polariser gradient.

Perpendicular The principal component most nearly perpendicular to the polariser gradient and parallel to the surface.

Normal The principal component most nearly perpendicular to the surface.

Tensor values are defined per finite element, while growth factor values are defined per vertex and interpolated across each finite element. Thus tensor value appear as a flat colour on each FE, while growth factors appear as more continuous colour fields.

The checkboxes in the Plot options panel select various other things to plot.

Canvas Plot the finite elements, colour-coded to represent the value of the property selected in the pull-down menu described above.

FE edges Draw the edges that separate the finite elements.

Polariser gradient Draw an arrow in each FE to show the direction of the polarisation gradient. FEs in which there is no gradient have no arrow drawn.

Tensor axes If a tensor quantity is being plotted, draw its axes in each FE.

Cells Draw the biological cells.

Mutant level If a growth factor is being plotted, and it has a mutation level, draw the mutated level (the default). If this is off, the unmutated level is drawn.

Some less frequently used options are on the Plot menu.

Black/White Background Set the background colour of the picture area.

Show/Hide Legend Show or hide the large text at the top of the picture.

Show/Hide Thickness Draw the mesh as a surface of finite or zero thickness.

Show/Hide Seams Highlight the seam edges in red.

Show/Hide Axes Show or hide the axes.

Show/Hide Displacements Draw an arrow at each vertex to indicate its displacement during the previous time step. The relative lengths of the arrows are accurate, but the absolute lengths are scaled to fit the scale of the picture.

Show/Hide Normals Draw the surface normal to each finite element.

Turn Light On/Off Turn a light on or off, to enhance the appearance of the surface.

Set Legend... Specify additional text to draw at the top of the picture.

Decor The A/B radio buttons determine which side of the mesh some of the “decorations” are drawn: the polariser gradient arrows and the FE normals. Biological cells are always drawn on both sides.

When there are very many FEs, the decorations can be difficult to see if they are drawn in every one. The Sparsity value can be used to draw these decorations larger, and in fewer FEs. No FE will be decorated if any other FE has a decoration within a distance equal to the sparsity value times the maximum diameter of the mesh along any of the axes.

Auto axis range If this checkbox is on, then the axis range is automatically scaled to fill the picture. If off, the specified values are used for the axis bounds.

Monochrome If checked, then a monochrome colour scale of a user-specified colour will be used to represent plotte values, instead of the default rainbow style.

Auto color range If on, then plotting a morphogen, scale the colours so as to go from blue=0 to red=the maximum value. If off, the color bar spans the user-specified range.

Pan, Zoom, Rot, RU: When clicked, these allow the mouse to be used to change the picture viewpoint accordingly: panning, zooming, rotating arbitrarily, and rotating while holding the camera up vector aligned with the vertical.

Clip This allows clipping by morphogen value. Clicking the **Mgen** button brings up a dialog in which you can select any set of growth factors, together with a criterion for clipping. The part of the mesh that will be visible will be the set of vertexes where either all or any of the selected factors is either below or above a user-specified threshold.

Clipping plane The Az, El, and D boxes specify the direction and distance of a clipping plane. The clipping plane and clipping by morphogen value can be used together.

12 Movies and snapshots

To start recording a movie, click the **Record movie...** button. If the **Auto-name** checkbox is on, a name for the movie will be generated automatically, otherwise th user is asked to choose a name. The movie will begin with an image of the current state of the mesh, as displayed on screen, and an image will be added after every simulation step. To close the movie, click the same button again (its name will have changed to **Stop movie**).

To capture a single image, click the **Take snapshot** button. The **Auto-name** setting applies in the same way to snapshot.

Movies and snapshots are just two-dimensional images. They contain none of the structure of the mesh and show whatever the picture area showed at the time, including the legend.

13 Scripting and non-interactive use

Most of the GUI controls are equivalent to Matlab commands. Each of these Matlab commands can be executed at the command prompt, without running GFtbox. Most of the available commands are equivalent to GUI operations.

GFtbox maintains a history of the GUI operations as a Matlab script, which can be saved to a file or written to the Matlab command window. When a script file is loaded (by the **Load...** button), the commands in it are executed.

A script file can be executed at the Matlab prompt without running GFtbox. If it does not ever draw the mesh, it can be run even on a machine without graphics, such as a remote cluster.

Every command has a name beginning “leaf_”. A command may take a number of required arguments, identified by their position in the argument list, followed by a number of optional arguments identified by keyword. The optional arguments can be given in any order, and usually default values will be assumed for optional arguments not specified.

If a command modifies an existing canvas, the current canvas is always the first required argument.

The result of every command is the new canvas. If a command which creates a new canvas fails (for example, because of invalid arguments) then it returns the empty structure. If a command which modifies an existing canvas fails, it returns the old canvas unchanged.

If you edit a script file by hand, you can put in any other Matlab code you like. However, if the script file is to be read and executed by GFtbox, every line of the script must be a self-contained Matlab command — commands cannot be split over multiple lines. (This is a limitation of the implementation: GFtbox reads and executes script files one line at a time, in order to be able to interrupt execution if required after any command.)

A script file to be processed by GFtbox must also use the variable “m” to store the mesh, because GFtbox expects to find the mesh there when the script has finished.

Script files intended only for execution at the Matlab command prompt can contain completely arbitrary Matlab code.

14 Forthcoming

14.1 New features

1. There should be a command window for typing in Matlab commands that can operate on the current mesh.
2. When deleting elements from the mesh, it should be possible to undo changes.

14.2 Existing problems to be fixed

1. This manual is not up to date.
2. The various view controls do not always maintain their effect the next time the mesh is redrawn.
3. Gravity is not modelled.
4. Running on a stereo display is currently being implemented.

15 Reference to all leaf commands

This section of the manual reproduces the help text from all of the script commands. The help text is also available at the Matlab command prompt by typing `help commandname`.

15.1 leaf_add_mgen

```
m = leaf_add_mgen( m, mgen_name, ... )
```

Add a new morphogen to `m` with the given name. If there is already a morphogen with that name, this command is ignored. Any number of names can be given at once.

Equivalent GUI operation: the "New" button on the "Morphogens" panel. A dialog will appear in which the user chooses a name for the new morphogen.

See also:

`LEAF_DELETE_MGEN`, `LEAF_RENAME_MGEN`

15.2 leaf_add_userdata

```
m = leaf_add_userdata( m, ... )
```

Add fields to the userdata of `m`. The arguments should be alternately a field name and a field value. Existing fields of those names in `m.userdata` will be left unchanged.

See also: `LEAF_SET_USERDATA`, `LEAF_DELETE_USERDATA`.

Equivalent GUI operation: none.

15.3 leaf_addbioregion

```
m = leaf_addbioregion( m, cells )
```

Add the specified finite element patches to the region within which biological cells will be simulated, or remove them if they are already included. The patches are identified by their number, which is not easily ascertained by the user. This command is primarily intended to be generated by the GUI when the user clicks to select a patch.

Arguments:

1: A list of the patches to add to the region.

Equivalent GUI operation: clicking on the mesh when "3rd layer" is selected in the "Mouse mode" pull-down menu.

15.4 leaf_addpicture

```
m = leaf_addpicture( m, ... )
```

Create a new picture window to plot the mesh in. This is primarily intended for plotting the mesh in multiple windows simultaneously.

Options:

'figure' A handle to a window. The previous contents of the window will be erased.

'position' An array [x y w h] specifying the position and size of the

window relative to the bottom left corner of the screen. x is horizontal from the left, y is vertical from the bottom, w is the width and h is the height, all in pixels.

'relpos' A similar array, but this time measured relative to the bottom left corner of the previous picture, if there is one, otherwise the bottom left corner of the screen.

Only one of position or relpos may be supplied.

w and h can be omitted from relpos, in which case the size defaults to the size of the existing window, if any, otherwise the default size for a new window.

x and y can be omitted from position, in which case the new window is centred on the screen.

If both position and relpos are omitted, a window of default size and position is created.

'vergence' A number in degrees, default 0. The azimuth of the figure is offset by this amount, so that the eye seeing the figure sees it as if the eye was turned towards the centre line by this angle.

'eye' 'l' or 'r', to specify which eye this is for.

If no eye is specified, vergence defaults to zero. If vergence is specified, an eye must also be specified.

'properties' A structure containing any other attribute of the figure that should be set.

15.5 leaf_addseam

```
m = leaf_addseam( m, ... )
```

Marks some edges of m as being or not being seams, according to criteria given in the options. Unlike all other toolbox commands, the options to this command are processed sequentially and the same option may occur multiple times in the list.

Options:

'init' Either 'all' or 'none'. Sets either all of the edges, or none of them, to be seam edges.

'edges' An array of edge indexes. All of these edges will become seam edges.

'nonedges' An array of edge indexes. All of these edges will become non-seam edges.

'nodes' An array of node indexes. All edges joining two edges in this set will become seam edges.

'nonnodes' An array of node indexes. All edges touching any node in this set will become non-seam edges.

'morphogen' A cell array of two elements. The first is a morphogen name or index. The second is a string specifying how the value of the morphogen will be used as a criterion for deciding whether an edge should become a seam. It consists of three parts. It begins with one of 'min', 'mid', or 'max'. This is followed by one of '<', '<=', '>=', or '>'. This is followed by a number. Examples: 'min>0.5' means that an edge becomes a seam if the minimum value of the morphogen at either end is greater than 0.5. 'max' would take the maximum of the ends, and 'mid' would take the average.

15.6 leaf_allowmutant

```
m = leaf_allowmutant( m, enable )
```

Enable or disable the whole mutation feature.

Arguments:

enable: 1 to allow mutation, 0 to disable mutation.

15.7 leaf_alwaysflat

```
m = leaf_alwaysflat( m, flat )
```

If FLAT is 1, force the mesh to never bend out of the XY plane. The mesh will be flattened if it is not already flat, by setting the Z coordinate of every node to zero.

If FLAT is 0, allow the mesh to bend out of the XY plane. If the mesh happens to be flat, it will not actually bend unless it is perturbed out of the XY plane, e.g. by adding a random Z displacement with the leaf_perturbz command.

Example:

```
m = leaf_alwaysflat( m, 1 );
```

Equivalent GUI operation: clicking the "Always flat" checkbox on the "Mesh editor" panel.

15.8 leaf_archive

```
m = leaf_archive( m )
```

Create an archive of the current state of the project.

Archived states are kept in a subfolder ARCHIVE of the current project. Each archived state is in a project folder whose name is the name of the current project, with '_Ann' appended, where nn is a number 1 greater than the maximum number that has already been used, or 1 for the first created archive.

Arguments: none.

Equivalent GUI operation: clicking the "Archive" button.

INCOMPLETE -- DO NOT USE.

15.9 leaf_attachpicture

```
m = leaf_attachpicture( varargin )
```

NOT SUPPORTED. INCOMPLETE. NON-OPERATIONAL.

Load a picture from a file. Create a rectangular mesh of the same proportions.

If no filename is given for the picture, a dialog will be opened to choose one.

Equivalent GUI operation: none.

15.10 leaf_bowlz

```
m = leaf_bowlz( m, ... )
```

Add a bowl-shaped perturbation to the z coordinate of every point of the finite element mesh. The z displacement will be proportional to the square of the distance from the origin of coordinates.

Arguments:

A number, being the maximum displacement. The displacement will be scaled so that the farthest point from the origin is displaced by this amount. The default is 1.

Examples:

```
m = leaf_bowlz( m, 2 );
```

Equivalent GUI operation: the "Bowl Z" button on the "Mesh editor" panel. The amount of bowl deformation is specified by the value in the upper of the two text boxes to the right of the button.

15.11 leaf_circle

```
m = leaf_circle( m, ... )  
Create a new circular mesh.
```

Arguments:

M is either empty or an existing mesh. If it is empty, then an entirely new mesh is created, with the default set of morphogens. If M is an existing mesh, then its geometry is replaced by the new mesh. It retains the same set of morphogens (all set to zero everywhere on the new mesh), interaction function, and all other properties not depending on the specific geometry of the mesh.

Options:

'radius'	The radius of the circle. Default 1.
'rings'	The number of circular rings of triangles to divide it into. Default 4.
'circumpts'	The number of vertexes around the circumference. The default is rings*6. It must be at least 4, and for best results should be at least rings*4. As a special case, if zero is specified, rings*6 is chosen.
'innerpts'	The number of vertexes around the innermost ring of points. Default is max(floor(circum/nrings), 3).
'dealign'	Dealign the vertexes on adjacent rings. Default false. Only applies when circumpts is nonzero.
'hemisphere'	Create a hemisphere instead of a flat circle. The argument is the height of the hemisphere as a proportion of the radius. Default is 0, i.e. make a flat circle. It can be negative.

Example:

```
m = leaf_circle( [], 'radius', 2, 'rings', 4 );
```

Equivalent GUI operation: selecting "Circle" or "Hemisphere" on the pulldown menu on the "Mesh editor" panel, and clicking the "Generate mesh" button.

15.12 leaf_colourA

```
m = leaf_colourA( m )
```

Assign colours to all the cells in the bio-A layer. If there is no bio-A layer, the command is ignored.

Optional arguments:

`colors`: The colour of the cells, as a pair of RGB values. The first is for the unshocked state and the second for the shocked state. If this is not specified, the command does nothing.

`colorvariation`: The amount of variation in the colour of the new cells. Each component of the colour value will be randomly chosen within this ratio of the value set by the 'color' argument. That is, a value of 0.1 will set each component to between 0.9 and 1.1 times the corresponding component of the specified colour. (The variation is actually done in HSV rather than RGB space, but the difference is slight.) The default is zero.

15.13 leaf_cylinder

```
m = leaf_cylinder( m, ... )
```

Create a new surface, in the form of an open-ended cylinder whose axis is the Z axis, centred on the origin.

Arguments:

M is either empty or an existing mesh. If it is empty, then an entirely new mesh is created, with the default set of morphogens. If M is an existing mesh, then its geometry is replaced by the new mesh. It retains the same set of morphogens (all set to zero everywhere on the new mesh), interaction function, and all other properties not depending on the specific geometry of the mesh.

Options:

'radius' The radius of the cylinder. Default 2.
'height' The height of the cylinder. Default 2.
'circumdivs' The number of divisions around the cylinder.
 Default 12.
'heightdivs' The number of divisions along the axis of the
 cylinder. Default 4.

Example:

```
m = leaf_cylinder( [], 'radius', 2, 'height', 2, 'circumdivs', 12,  
                              'heightdivs', 4 );
```

See also: LEAF_CIRCLE, LEAF_LUNE, LEAF_ONECELL, LEAF_RECTANGLE, LEAF_SEMICIRCLE.

Equivalent GUI operation: selecting "Cylinder" on the pulldown menu on the "Mesh editor" panel, and clicking the "Generate mesh" button. The radius, height, number of divisions around, and number of divisions vertically are given by the values of the text boxes named "radius", "y width", "x divs", and "y divs" respectively.

15.14 leaf_delete_mgen

```
m = leaf_delete_mgen( m, mgen_name, ... )
```

Delete from m the morphogen that has the given name. If there is no such morphogen, or if the name is one of the reserved morphogen names,

this command is ignored. Any number of names can be given at once.

Equivalent GUI operation: clicking the "Delete" button in the "Morphogens" panel, which deletes the currently selected morphogen.

See also:

LEAF_ADD_MGEN, LEAF_RENAME_MGEN

15.15 leaf_delete_userdata

```
m = leaf_delete_userdata( m, ... )
```

Delete specified fields from the userdata of m. The arguments should be strings. If no strings are given, all the user data is deleted.

If a named field does not exist, it is ignored.

See also: LEAF_SET_USERDATA, LEAF_ADD_USERDATA.

Equivalent GUI operation: none.

15.16 leaf_deletecells

```
m = leaf_deletecells( m )
```

Delete all the biological cells in the third layer. The region within which biological cells may be created is left unchanged.

Example:

```
m = leaf_deletecells( m );
```

Equivalent GUI operation: none.

15.17 leaf_deletepatch

```
m = leaf_deletepatch( m, cells )
```

Delete the specified finite element patches from the leaf.

Arguments:

1: A list of the cells to add to the region.

Equivalent GUI operation: clicking on the mesh when the "Delete canvas" item is selected in the "Mouse mode" pulldown menu.

15.18 leaf_deletepatch_from_morphogen_level

```
ind=find(wound>0.5);
listcells=[];
for i=1:size(m.tricellvxs,1)
    if length(intersect(m.tricellvxs(i,:),ind'))==3
        listcells(end+1)=i;
    end
end
```

15.19 leaf_deletessecondlayer

```
m = leaf_deletessecondlayer( m )
```

Delete the first biological layer.

Equivalent GUI operation: clicking the "Delete all cells" button in the "Bio 1" panel.

15.20 leaf_deletestages

```
m = leaf_deletestages( m )
```

Delete all the stage files for *m*, and optionally, the stage times. Deleted files are gone at once, not put in the wastebasket.

Options:

'times' Boolean. If true, also delete the stored stage times from the mesh.

Equivalent GUI operation: the "Delete All Stages..." and "Delete Stages and Times" commands on the Stages menu.

15.21 leaf_deletethirdlayer

```
m = leaf_deletethirdlayer( m )
```

Delete the second biological layer.

Equivalent GUI operation: clicking the "Delete all cells" button in the "Bio 2" panel.

15.22 leaf_destrain

```
m = leaf_destrain( m )
```

Remove all residual strain from the mesh.

Equivalent GUI operation: clicking the "De-strain" button on the "Simulation" panel.

15.23 leaf_dissect

```
m = leaf_dissect( m )
```

Cut *m* along all of its seam edges.

Arguments and options: none.

```
[m,ok] = leaf_dointeraction( m, enable )
```

Execute the interaction function once, without doing any simulation steps. This will happen even if the `do_interaction` property of the mesh is set to false,

If there is no interaction function, this has no effect.

If there is an interaction function and it throws an exception, OK will be returned as FALSE.

Arguments:

enable: 1 if the interaction function should be enabled for all subsequent simulation steps, 0 if its state of enablement should be left unchanged (the default). If the i.f. throws an error then it will be disabled for subsequent steps regardless of the setting of

this argument.

```
m = leaf_edit_interaction( m, ... )
```

Open the interaction function in the Matlab editor.
If there is no interaction function, create one.

Options:

'force' If true, the interaction function will be opened even if the model m is marked read-only. If false (the default) a warning will be given and the function not opened.

Extra results:

ok is true if the function was opened, false if for any reason it was not.

15.24 leaf_enablelegend

```
m = leaf_enablelegend( m, enable )
```

Cause the legend to be drawn or not drawn.
When not drawn, the graphic item that holds the legend text will be made invisible.

Arguments:

1. A boolean specifying whether to draw the legend (default true).

15.25 leaf_enablemutations

```
m = leaf_enablemutations( m, enable )
```

Enable or disable mutations.

Arguments:

enable: True to enable all mutations, false to disable them.

Examples:

```
m = leaf_enablemutant( m, 0 );  
% Disable all mutations, i.e. revert to wild-type.
```

15.26 leaf_explode

```
m = leaf_explode( m, amount )
```

Separate the connected components of m.

Arguments:

amount: Each component of m is moved so as to increase the distance of its centroid from the centroid of m by a relative amount AMOUNT. Thus AMOUNT==0 gives no movement and AMOUNT < 0 will draw the pieces inwards.

15.27 leaf_fix_mgen

```
m = leaf_fix_mgen( m, morphogen, ... )
```

Make the current value of a specified morphogen at a specified vertex or set of vertexes be fixed or changeable.

Arguments:

1: The name or index of a morphogen.

Options:

'vertex' Indexes of the vertexes. The default is the empty list (i.e. do nothing).

'fix' 1 or true if the value is to be made fixed, 0 or false if it is to be made changeable. The default is true.

Equivalent GUI operation: control-clicking or right-clicking on the canvas when the Morphogens panel is selected.

15.28 leaf_fix_vertex

`m = leaf_fix_vertex(m, ...)`

Constrain vertexes of the mesh so that they are only free to move along certain axes.

Options:

'vertex' The vertexes to be constrained. If the empty list is supplied, the constraint is applied to all vertexes.

'dfs' The degrees of freedom to be constrained. This is a string made of the letters 'x', 'y', and 'z'. This defaults to 'xyz', i.e. fix the vertexes completely.

Each degree of freedom not in dfs will be made unconstrained for all of the given vertexes. Vertexes not in the list of vertexes will have their constraints left unchanged.

It is only possible to constrain vertexes in directions parallel to the axes.

Equivalent GUI operation: clicking on the mesh while the Mesh editor panel is selected and 'Fix' is selected in the Fix/Delete menu. The 'x', 'y', and 'z' checkboxes specify which degrees of freedom to constrain or unconstrain.

15.29 leaf_flatstrain

`m = leaf_flatstrain(m)`

Set the residual strains in the mesh to what they would be if the whole mesh were flat.

Arguments: none.

Options: none.

Equivalent GUI operation: clicking the "Flat strain" button in the "Simulation" panel.

`[m,ok] = leaf_flatten(m)`

Flatten each of the connected components of m.

Options:

interactive: If true (default is false), then the flattening will

be carried out interactively. The user can skip the flattening of components that appear not to be well flattenable, or cancel the whole operation.

15.30 leaf_flattenX

```
m = leaf_flatten( m )
```

Flatten each of the connected components of m.

Options:

ratio: This is the proportion of the flattening displacements to apply. The default value is 1, i.e. complete flattening.

15.31 leaf_fliporientation

```
m = leaf_fliporientation( m )
```

Interchange the two surfaces of the mesh.

15.32 leaf_gyrate

```
m = leaf_gyrate( m, ... )
```

Spin and/or tilt the mesh about the Z axis, leaving it at the end in exactly the same orientation as when it started. If a movie is currently being recorded, the animation will be appended to the movie. The current view is assumed to have already been written to the movie.

Options:

'frames': The number of frames to be added. Default 32.
'spin': The number of complete rotations about the Z axis. Default 1.
'tilt': The number of cycles of tilting up, down, and back to the initial elevation. Default 1.
'tiltangle': The angle to tilt up and down to, in degrees from the horizontal. Default 89.99.

15.33 leaf_hemisphere

```
m = leaf_hemisphere( m, ... )
```

Create a new hemispherical mesh. The mesh is oriented so that the cell normals point outwards.

Arguments:

M is either empty or an existing mesh. If it is empty, then an entirely new mesh is created, with the default set of morphogens. If M is an existing mesh, then its geometry is replaced by the new mesh. It retains the same set of morphogens (all set to zero everywhere on the new mesh), interaction function, and all other properties not depending on the specific geometry of the mesh.

Options:

'radius' The radius of the hemisphere. Default 1.
'divisions' The number of divisions around the circumference. Default 20.
'rings' The number of circular rings of triangles to divide

it into. Default is floor(divisions/6).

Example:

```
m = leaf_hemisphere( [], 'radius', 2, 'divisions', 15, 'rings', 3 );
```

Equivalent GUI operation: selecting "Hemisphere" on the pulldown menu on the "Mesh editor" panel, and clicking the "Generate mesh" button. The radius and the number of rings are specified in the text boxes with those labels.

15.34 leaf_icosahedron

```
m = leaf_icosahedron( m, ... )  
Create a new icosahedral mesh.
```

Arguments:

M is either empty or an existing mesh. If it is empty, then an entirely new mesh is created, with the default set of morphogens. If M is an existing mesh, then its geometry is replaced by the new mesh. It retains the same set of morphogens (all set to zero everywhere on the new mesh), interaction function, and all other properties not depending on the specific geometry of the mesh.

Options:

'radius' The radius of the icosahedron. Default 1.

Example:

```
m = leaf_circle( [], 'radius', 2, 'rings', 4 );
```

Equivalent GUI operation: selecting "Circle" or "Hemisphere" on the pulldown menu on the "Mesh editor" panel, and clicking the "Generate mesh" button.

```
[m,ok] = leaf_iterate( m, numsteps, ... )
```

Run the given number of iterations of the growth process.

In each iteration, the following things happen:

- * Strains are set up in the leaf according to the effects of the morphogens at each point causing the material to grow.
- * The elasticity problem is solved to produce a new shape for the leaf.
- * Morphogens which have a nonzero diffusion coefficient are allowed to diffuse through the mesh.
- * If dilution by growth is enabled, the amount of every morphogen is diluted at each point according to how much that part of the leaf grew.
- * A user-supplied routine is invoked to model local interactions between the morphogens.
- * The layer of biological cells is updated.
- * If requested by the options, the mesh is plotted after each iteration.

Results:

ok: True if there was no problem (e.g. invalid arguments, a user interrupt, or an error in the interaction function). False if there was.

Arguments:

1: The number of iterations to perform. If this is zero, the computation will continue indefinitely or until terminated by the 'until' or 'targetarea' options.

Options:

'until' Run the simulation until this time has been reached or exceeded. A value of zero disables this option.

'targetarea' Run the simulation until the area of the canvas is at least this number times the initial area. A value of zero disables this option.

'plot' An integer n. The mesh will be plotted after every n iterations. 0 means do not plot the mesh at all; -1 means plot the mesh only after all the iterations have been completed. The default value is 1.

Example:

```
m = leaf_iterate( m, 20, 'plot', 4 );
```

Equivalent GUI operation: clicking one of the following buttons in the "Simulation" panel: "Run" (do a specified number of steps), "Step" (do one step), or "Run to..." (run until the area has increased by a specified factor).

15.35 leaf_load

```
m = leaf_load( m, filename, ... )
```

Load a leaf from a file. If no filename is given, a dialog will be opened to choose one.

The expected format depends on the extension of the filename:

.MAT The leaf is contained in the file as a Matlab object called m.

.M The file contains Matlab commands to create or modify a leaf. These commands will be executed.

.OBJ Contains only nodes and triangles, in OBJ format. All other properties of the mesh will be set to their default values.

If no filename is given, a dialog will be opened to choose one.

If the filename consists of just an extension (including the initial "."), a dialog will be opened showing only files with that extension.

All of these formats can be generated by leaf_save.

In the case of .MAT and .OBJ files, the existing leaf will be discarded. A .M file will discard the current leaf only if it contains a command to create a new leaf; otherwise, it will apply its commands to the current leaf.

Equivalent GUI operation: the "Load..." button.

15.36 leaf_loadgrowth

```
m = leaf_loadgrowth( m, filename )
```

Load growth data for the leaf from an OBJ or MAT file. If no filename is given, one will be asked for.

This assumes the mesh is in growth/anisotropy mode.

Equivalent GUI operation: the "Load Growth..." button on the "Morphogens" panel.

```
[m,ok] = leaf_loadmodel( m, modelname, projectdir, ... )
```

Load a model. If no model name is given or the model name is empty, a dialog will be opened to choose one. The model will be looked for in projectdir, if given, otherwise the project directory of m, if any, otherwise the current directory. The argument m can be empty; in fact, this will be the usual case.

If the model is successfully loaded, the new model is returned in M and the (optional) return value OK is set to TRUE. Otherwise M is left unchanged and OK is set to FALSE.

Options:

rewrite: Normally, when a model is loaded, its interaction function (if there is one) is read, parsed, and rewritten. This is because it may have been created with an older version of GFTbox. Specifying the rewrite option as false prevents this from being done. This may be necessary when running several simulations concurrently on a parallel machine, all using the same project. Note that when rewrite is true (the default), the interaction function will not actually be rewritten until the first time it is called, or any morphogen is added, deleted, or renamed.

copyname, copydir:

If either of these is given, a new copy of the project will be made and saved with the specified project name and parent folder. The original project folder will be unmodified. If one of these options is given, the other can be omitted or set to the empty string, in which case it defaults to the original project name or project folder respectively. If the value of copyname is '?', then the user will be prompted to select or create a project folder. In this case, copydir will be the folder at which the select-folder dialog starts. If both options are empty, this is equivalent to omitting both of them (the default). If copydir and copyname are the same as modelname and projectdir, a warning will be given, and the copy options ignored.

If for any reason the model cannot be saved, a warning will be output, the loaded model will be discarded, and the empty list returned.

Equivalent GUI operation: the "Load model..." button, or the items on the Projects menu. The items on the Motifs menu use copyname and copydir to force the "motif" projects to be opened as copies in the user's default project directory.

Examples:

```
m = leaf_loadmodel( [], 'flower7', 'C:\MyProjects\flowers', ...  
                  'copyname', 'flower8', ...  
                  'copydir', 'C:\MyProjects\flowers' );
```

This loads a model from the folder 'C:\MyProjects\flowers\flower7', and saves it into a new project folder 'C:\MyProjects\flowers\flower8'. Since the value of copydir is the same as the projectdir argument,

the copydir option could have been omitted.

15.37 leaf_lobes

```
m = leaf_lobes( m, ... )
```

Create a new mesh in the form of one or more lobes joined together in a row. A lobe is a semicircle on top of a rectangle.

Arguments:

M is either empty or an existing mesh. If it is empty, then an entirely new mesh is created, with the default set of morphogens. If M is an existing mesh, then its geometry is replaced by the new mesh. It retains the same set of morphogens (all set to zero everywhere on the new mesh), interaction function, and all other properties not depending on the specific geometry of the mesh.

Options:

'radius'	The radius of the semicircle. Default 1.
'rings'	The number of circular rings of triangles to divide it into. Default 4.
'height'	The height of the rectangle, as a multiple of the semicircle's diameter. Default 0.7.
'strips'	The number of strips of triangles to divide the rectangular part into. If 0 (the default), this will be calculated from the height so as to make the triangles similar in size to those in the lobes.
'lobes'	The number of lobes. The default is 1.
'base'	Half the number of divisions along the base of a lobe. Defaults to rings.
'cylinder'	The series of lobes is to behave as if wrapped round a cylinder and the two ends stitched together. This is implemented by constraining the nodes on the outer edges in such a way that the outer edges remain parallel to the y axis.

Example:

```
m = leaf_lobes( 'radius', 2, 'rings', 4, 'lobes', 3, 'base', 2 );  
See also: LEAF_CIRCLE, LEAF_CYLINDER, LEAF_LUNE, LEAF_ONECELL,  
LEAF_RECTANGLE.
```

Equivalent GUI operation: selecting "Lobes" in the pulldown menu in the "Mesh editor" panel and clicking the "Generate mesh" button.

15.38 leaf_locate_vertex

```
m = leaf_locate_vertex( m, ... )
```

Ensure that certain degrees of freedom of a single node remain constant. This is ensured by translating the mesh so as to restore the values of the specified coordinates, after each iteration.

Options:

'vertex'	The vertex to be held stationary. If the empty list is supplied, no vertex will be fixed and dfs is ignored.
'dfs'	The degrees of freedom to be held stationary. This is a string made of the letters 'x', 'y', and 'z'. This

defaults to 'xyz', i.e. fix the vertex completely. Each degree of freedom not in dfs will be unconstrained.

It is only possible to fix a vertex in directions parallel to the axes.

Equivalent GUI operation: clicking on the mesh while the Mesh editor panel is selected and 'Locate' is selected in the Fix/Delete menu. The 'x', 'y', and 'z' checkboxes specify which degrees of freedom to constrain or unconstrain.

15.39 leaf_lune

```
m = leaf_lune( m, ... )
```

NOT IMPLEMENTED.

Create a new mesh in the shape of a stereotypical leaf, oval with pointed ends.

Arguments:

M is either empty or an existing mesh. If it is empty, then an entirely new mesh is created, with the default set of morphogens. If M is an existing mesh, then its geometry is replaced by the new mesh. It retains the same set of morphogens (all set to zero everywhere on the new mesh), interaction function, and all other properties not depending on the specific geometry of the mesh.

Options:

'xwidth'	The diameter in the X dimension. Default 3.
'ywidth'	The diameter in the Y dimension. Default 2.
'xdivs'	The number of segments to divide it into along the X axis. Default 8.

Example:

```
m = leaf_lune( [], 'xwidth', 3, 'ywidth', 2, 'xdivs', 8 );
```

See also: LEAF_CIRCLE, LEAF_CYLINDER, LEAF_ONECELL, LEAF_RECTANGLE, LEAF_SEMICIRCLE, LEAF_LOBES.

Equivalent GUI operation: selecting "Leaf" in the pulldown menu in the "Mesh editor" panel and clicking the "Generate mesh" button.

15.40 leaf_makecells

```
m = leaf_makecells( m, numcells, numcvt )
```

Create a layer of biological cells. If there is already a biological layer, it is discarded.

Arguments:

- 1: The total number of biological cells to create.
- 2: The number of iterations of the CVT transformation to apply to the cells. The more iterations, the more nearly the cells will be of equal sizes, and with walls tending to meet at 120 degrees. The default is 10.

Examples:

```
m = leaf_makecells( m, 100 );
```

Equivalent GUI operation: clicking the "Make cells" button.

15.41 leaf_makesecondlayer

```
m = leaf_makesecondlayer( m, ... )
```

Make a new Bio-A layer, either adding to or discarding any existing one.

Options:

mode: One of the following strings:

'full': cover the entire surface with a continuous sheet of cells. There will be one cell per FE cell plus one cell per FE vertex.

'grid': cover the entire surface with a continuous sheet of square cells.

'voronoi': cover the entire surface with a continuous sheet of cells. The 'numcells' option specifies how many. This is only valid for flat or nearly flat meshes.

'single': make a single second layer cell in a random position.

'few': make a specified number of second layer cells, each in a different randomly chosen FEM cell.

'each': make one second layer cell within each FEM cell.

abssize: Not valid for mode=full. In all other cases, this is a real number, being the diameter of a single cell.

relsize: Not valid for mode=full. In all other cases, this is a real number, being the diameter of a single cell as a proportion of the average diameter of the current mesh.

relinitarea: Not valid for mode=full. In all other cases, this is a real number, being the area of a single cell as a proportion of the initial area of the mesh.

relFEsize: Not valid for mode=full. In all other cases, this is a real number, being the diameter of a single cell as a proportion of the average diameter of a typical FE cell in the current mesh.

numcells: Valid only for mode='voronoi' or mode='few'. An integer specifying the number of cells to create.

fraccells: Valid only for mode='each'. A real number between 0 and 1, it specifies the proportion of FEs that should have a bio cell placed in them.

add: Boolean. If true, existing cells are retained and new cells are added to them. If false, any existing biological layer is discarded. If 'mode' is 'full' or 'voronoi', the old layer is always discarded, ignoring the 'add' argument.

colors: The colour of the new cells, as an RGB value.

colorvariation: The amount of variation in the colour of the new cells. Each component of the colour value will be randomly chosen within this ratio of the value set by the 'color' argument. That is, a value of 0.1 will set each component to between 0.9 and 1.1 times the corresponding component of the specified colour. (The variation is actually done in HSV rather than RGB space, but the difference is slight.)

probperFE: For 'each' mode, a probability for each FE that a cell will be created there. Any value less than 0 is equivalent to 0, and any value greater than 1 is equivalent to 1.

probpervx: Like probperFE, but the values are given per vertex. Alternatively, the value can be a morphogen name or index, in which case the values of that morphogen at each vertex will be used.

The options 'fraccells', 'probperFE', and 'probpervx' are mutually exclusive. If 'probperFE' or 'probpervx' is given, the probabilities will also be weighted by the areas of the finite

elements. Thus a uniform probability field will give a uniform density of biological cells, regardless of the sizes of the finite elements.

At most one of ABSSIZE, RELSIZE, RELINITAREA, and RELFESIZE should be given.

Equivalent GUI operation: clicking the "Make cells" button on the Bio-A panel. This is equivalent to `m = leaf_makesecondlayer(m, 'mode', 'full')`.

15.42 leaf_mgen_absorption

```
m = leaf_mgen_absorption( m, morphogen, absorption )
```

Set the rate at which a specified morphogen is absorbed.

Arguments:

- 1: The name or index of a morphogen.
- 2: The rate of absorption of the morphogen. A value of 1 means that the morphogens decays by 1% every 0.01 seconds.

Examples:

```
m = leaf_mgen_absorption( m, 'growth', 0.5 );
```

Equivalent GUI operation: setting the value in the "Absorption" text box in the "Morphogens" panel.

15.43 leaf_mgen_conductivity

```
m = leaf_mgen_conductivity( m, morphogen, conductivity )
```

Set the rate at which a specified morphogen diffuses through the leaf.

Arguments:

- 1: The name or index of a morphogen.
- 2: The conductivity of the surface to the morphogen. This is in dimensionless units: try a value of 1.

Examples:

```
m = leaf_mgen_conductivity( m, 'growth', 0.5 );
```

Equivalent GUI operation: setting the value in the "Diffusion" text box in the "Morphogens" panel.

15.44 leaf_mgen_const

```
m = leaf_mgen_const( m, morphogen, amount )
```

Add a constant amount to the value of a specified morphogen everywhere.

Arguments:

- 1: The name or index of a morphogen. Currently, if the name is provided, it must be one of the following:
'growth', 'polariser', 'anisotropy',
'bend', 'bendpolariser', 'bendanisotropy'.
These are respectively equivalent to the indexes 1 to 6 (which are the only valid indexes). There is no default for this option.
- 2: The amount of morphogen to add to every node. A value of 1 will give moderate growth or bend, and a maximum growth or bend anisotropy. A constant field of growth or bend polarizer has no effect: polarising morphogen has an effect only through its gradient.

Examples:

```
m = leaf_mgen_const( m, 'growth', 1 );
```

```
m = leaf_mgen_const( m, 3, 0.8 );  
See also: LEAF_MGEN_RADIAL.
```

Equivalent GUI operation: clicking the "Add const" button in the "Morphogens" panel. The amount is specified by the "Amount slider and test item.

15.45 leaf_mgen_dilution

```
m = leaf_mgen_dilution( m, morphogen, enable )  
Set the rate at which a specified morphogen is absorbed.  
Arguments:  
1: The name or index of a morphogen.  
2: A boolean specifying whether to enable dilution by growth  
Examples:  
m = leaf_mgen_dilution( m, 'growth', 1 );
```

Equivalent GUI operation: setting the "Dilution" checkbox in the "Morphogens" panel.

15.46 leaf_mgen_edge

```
m = leaf_mgen_edge( m, morphogen, amount, ... )  
Set the value of a specified morphogen to a given amount everywhere on  
the edge of the leaf.  
Arguments:  
1: The name or index of a morphogen.  
2: The maximum amount of morphogen to add to every node.  
Examples:  
m = leaf_mgen_edge( m, 'growth', 1 );  
See also: leaf_mgen_const.
```

Equivalent GUI operation: clicking the "Add edge" button in the "Morphogens" panel. The amount is specified by the "Amount slider and test item.

15.47 leaf_mgen_linear

```
m = leaf_mgen_linear( m, morphogen, amount, ... )  
Set the value of a specified morphogen to a linear gradient.  
Arguments:  
1: The name or index of a morphogen.  
2: The maximum amount of morphogen to add to every node.  
Options:  
  'direction'      Either a single number (the angle in degrees  
                   between the gradient vector and the X axis, the  
                   gradient vector lying in the XY plane; or a triple  
                   of numbers, being a vector in the direction of the  
                   gradient. The length of the vector does not  
                   matter. Default is a gradient parallel to the  
                   positive X axis.  
Examples:  
m = leaf_mgen_linear( m, 'growth', 1, 'direction', 0 );  
See also: leaf_mgen_const.
```

Equivalent GUI operation: clicking the "Add linear" button in the "Morphogens" panel. The amount is specified by the "Amount slider and test item. Direction is specified in degrees by the "Direction" text box.

15.48 leaf_mgen_modulate

```
m = leaf_mgen_modulate( m, ... )
```

Set the switch and mutant levels of a morphogen.

Options:

morphogen: The name or index of a morphogen. If omitted, the properties are set for every morphogen.

switch, mutant: Value by which the morphogen is multiplied to give its effective level.

If either switch or mutant is omitted its current value is left unchanged.

The effective value of a morphogen is the product of the actual morphogen amount, the switch value, and the mutant value. So mutant and switch have the same effect; the difference is primarily in how they are intended to be used. Mutant value is settable in the Morphogens panel of the GUI and is intended to have a constant value for each morphogen throughout a run. There is also a checkbox in the GUI to turn all mutations on and off. Switch value has no GUI interface, and is intended to be changed in the interaction function. The switch values are always effective.

The initial values for switch and mutant in a newly created leaf are 1.

Examples:

```
m = leaf_mgen_modulate( m, 'morphogen', 'div', ...  
                        'switch', 0.2, ...  
                        'mutant', 0.5 );
```

Sets the switch level of 'div' morphogen to 0.2 and the mutant level to 0.5. The effective level will then be 0.1 times the actual morphogen.

15.49 leaf_mgen_radial

```
m = leaf_mgen_radial( m, morphogen, amount, ... )
```

Add to the value of a specified morphogen an amount depending on the distance from an origin point.

Arguments:

1: The name or index of a morphogen.

2: The maximum amount of morphogen to add to every node.

Options:

'x', 'y', 'z' The X, Y, and Z coordinates of the centre of the distribution, relative to the centre of the mesh. Default is (0,0,0).

Examples:

```
m = leaf_mgen_radial( m, 'growth', 1, 'x', 0, 'y', 0, 'z', 0 );  
m = leaf_mgen_radial( m, 'g_anisotropy', 0.8 );
```

See also: leaf_mgen_const.

Equivalent GUI operation: clicking the "Add radial" button in the "Morphogens" panel. The amount is specified by the "Amount slider and test item. x, y, and z are specified in the text boxes of those names.

15.50 leaf_mgen_random

```
m = leaf_mgen_random( m, morphogen, amount, ... )
```

Add a random amount of a specified morphogen at each mesh point.

Arguments:

1: The name or index of a morphogen.

2: The maximum amount of morphogen to add to every node.

Options:

'smooth'	An integer specifying the smoothness of the distribution. 0 means no smoothing: the value at each node is independent of each of its neighbours. Greater values imply more smoothness. Default is 2.
----------	--

Examples:

```
m = leaf_mgen_random( m, 'growth', 1 );
```

```
m = leaf_mgen_random( m, 'g_anisotropy', 0.8 );
```

See also: LEAF_MGEN_CONST.

Equivalent GUI operation: clicking the "Add random" button in the "Morphogens" panel. The amount is specified by the "Amount slider and test item.

15.51 leaf_mgen_reset

```
m = leaf_mgen_reset( m )
```

Set the value of all morphogens and all conductivities to zero everywhere.

Example:

```
m = leaf_mgen_reset( m );
```

Equivalent GUI operation: clicking the "Set zero all" button in the "Morphogens" panel.

15.52 leaf_mgen_scale

```
m = leaf_mgen_scale( m, morphogen, scalefactor )
```

Scale the value of a given morphogen by a given amount.

Arguments:

1: The name or index of a morphogen.

2: The scale factor.

Examples:

```
m = leaf_mgen_scale( m, 'bpar', -1 );
```

See also: leaf_mgen_const.

Equivalent GUI operation: clicking the "Invert" button in the "Morphogens" panel will scale the current morphogen by -1. There is not yet a user interface for a general scale factor.

15.53 leaf_mgen_zero

```
m = leaf_mgen_zero( m, morphogen )
```

Set the value of a specified morphogen to zero everywhere.

Arguments:

1: The name or index of a morphogen.

Examples:

```
    m = leaf_mgen_zero( m, 'growth' );
```

See also: LEAF_MGEN_CONST.

Equivalent GUI operation: clicking the "Set zero" button in the "Morphogens" panel.

15.54 leaf_mgeninterpolation

```
m = leaf_mgeninterpolation( m, ... )
```

Set the interpolation mode of morphogens of m. When an edge of the mesh is split, this determines how the morphogen values at the new vertex are determined from the values at either end of the edge.

Options:

'morphogen' This can be a morphogen name or index, a cell array of morphogen names and indexes, or a vector of indexes.

'interpolation' Either 'min', 'max', or 'average'. If 'min', the new values are the minimum of the old values, if 'max' the maximum, and if 'average' the average.

GUI equivalent: the radio buttons in the "On split" subpanel of the "Morphogens" panel. These set the interpolation mode for the current morphogen. As of the version of 2008 Sep 03, new meshes are created with the interpolation mode for all morphogens set to 'min'. Previously the default mode was 'average'.

Example:

```
    m = leaf_mgeninterpolation( m, ...
        'morphogen', 1:size(m.morphogens,2), ...
        'interpolation', 'average' );
```

This sets the interpolation mode for every morphogen to 'average'.

```
    disp(sprintf('%d %f',i,dt(i)))
```

```
    ind=find(pm_l>0.95*max(pm_l(:)));
```

```
    [m,tube_l]=leaf_morphogen_switch(m,...
        'StartTime',OnsetOfTubeGrowth,'EndTime',FinishTubeGrowth,...
        'Morphogen_l','tube','RealTime',realtime);
```

Alternatively, these can be specified

```
    [m,basemid_l]=leaf_morphogen_switch(m,...
        'StartTime',OnsetOfTubeGrowth,'EndTime',FinishTubeGrowth,...
        'Morphogen_l','basemid','RealTime',realtime,...
        'OnValue',1.0,'OffValue',0.0);
```

15.55 leaf_movie

```
m = leaf_movie( m, ... )
```

Start or stop recording a movie.

Any movie currently being recorded will be closed.

If the first optional argument is 0, no new movie is started.

Otherwise, the arguments may contain the following option-value pairs:

FILENAME The name of the movie file to be opened. If this is not given, and `m.globalProps.autonamemovie` is true, then a name will be generated automatically, guaranteed to be different from the name of any existing movie file. Otherwise, a file selection dialog is opened.

MODE (NOT IMPLEMENTED) This is one of 'screen', 'file', or 'fig'.
'screen' will capture the movie frames from the figure as drawn on the screen, using `avifile()`.
'file' will use `print()` to save the figure to a file, then load the file and add it to the movie. This allows arbitrarily high resolution movies to be made, not limited to the size drawn on the screen.
'fig' will save each frame as a MATLAB .fig file and will not generate a movie file. The figures can later be assembled into a movie file by running the command `fig2movie`. The reason for this option is that when running in an environment with no graphics, I have been unable to find a way of creating images from figures.

FPS, COMPRESSION, QUALITY, KEYFRAME, COLORMAP, VIDEONAME: These options are passed directly to the Matlab function `AVIFILE`. `LEAF_MOVIE` provides defaults for some of these:

```
FPS          15
COMPRESSION  'Cinepak'
QUALITY      100
KEYFRAME     5
```

Equivalent GUI operation: clicking the "Record movie..." button.

See also `AVIFILE`.

15.56 leaf_onecell

```
m = leaf_onecell( m, ... )
```

Create a new leaf consisting of a single triangular cell.

Arguments:

`M` is either empty or an existing mesh. If it is empty, then an entirely new mesh is created, with the default set of morphogens. If `M` is an existing mesh, then its geometry is replaced by the new mesh. It retains the same set of morphogens (all set to zero everywhere on the new mesh), interaction function, and all other properties not depending on the specific geometry of the mesh.

Options:

'xwidth' The diameter in the X dimension. Default 1.
'ywidth' The diameter in the Y dimension. Default 1.

Example:

```
m = leaf_onecell( [], 'xwidth', 1, 'ywidth', 1 );
```

See also: `LEAF_CIRCLE`, `LEAF_CYLINDER`, `LEAF_LUNE`, `LEAF_RECTANGLE`, `LEAF_SEMICIRCLE`.

Equivalent GUI operation: selecting "One cell" in the pulldown menu in

the "Mesh editor" panel and clicking the "Generate mesh" button.

15.57 leaf_paintpatch

```
m = leaf_paintpatch( m, cells, morphogen, amount )
```

Apply the given amount of the given morphogen to every vertex of each of the cells. The cells are identified numerically; the numbers are not user-accessible. This command is therefore not easily used manually; it is generated when the user clicks on a cell in the GUI.

Arguments:

- 1: A list of the cells to apply the morphogen to.
- 2: The morphogen name or index.
- 3: The amount of morphogen to apply.

Equivalent GUI operation: clicking on every vertex of the cell to be painted, with the "Current mgen" item selected in the "Mouse mode" menu. The morphogen to apply is specified in the "Displayed m'gen" menu in the "Morphogens" panel.

15.58 leaf_paintvertex

```
m = leaf_paintvertex( m, morphogen, ... )
```

Apply the given amount of the given morphogen to all of the given vertexes. The vertexes are identified numerically; the numbers are not user-accessible. This command is therefore not easily used manually; it is generated when the user clicks on a vertex in the GUI.

Arguments:

- 1: The morphogen name or index.

Options:

- 'vertex' The vertexes to apply morphogen to.
- 'amount' The amount of morphogen to apply. This can be either a single value to be applied to every specified vertex, or a list of values of the same length as the list of vertexes, to be applied to them respectively.
- 'mode' Either 'set' or 'add'. 'set sets the vertex to the given value, 'add' adds the given value to the current value.

Equivalent GUI operation: clicking on the mesh when the "Morphogens" panel is selected. This always operates in 'add' mode. The morphogen to apply is specified in the "Displayed m'gen" menu in the "Morphogens" panel. Shift-clicking or middle-clicking will subtract the morphogen instead of adding it.

15.59 leaf_perturbz

```
m = leaf_perturbz( m, ... )
```

Add a random perturbation to the Z coordinate of every node.

Arguments:

- A number z , the amplitude of the random displacement. The displacements will be randomly chosen from the interval $-z/2 \dots z/2$.

Example:

```
m = leaf_perturbz( m, 0.5 )
```

Equivalent GUI operation: the "Random Z" button on the "Mesh editor" panel. The amount of random deformation is specified by the value in the upper of the two text boxes to the right of the button.

15.60 leaf_plot

```
m = leaf_plot( m, ... )
```

Plot the leaf.

There are many options:

figure	The figure to plot in. By default, the current figure.
plotquantity	String. This determines what sort of quantity is to be plotted. 'morphogen': plot a morphogen. (The default.) 'specifiedgrowth': the quantity of growth specified by the morphogens. 'specifiedbend': the quantity of bending specified by the morphogens. 'actualgrowth', 'actualbend': the actual amount of growth or bend in the last timestep. 'residualgrowth', 'residualbend': the amount of growth or bend in the last timestep that was specified but which did not happen. The difference between specified and actual. 'strain' the magnitude of the residual strain
morphogen	String or number: the name or index of a morphogen to be plotted. The default is 1.
drawedges	Integer, determines which edges of the finite element cells to draw. 0 = draw no edges, 1 = draw only edges on the edge of the leaf, 2 = draw all edges.
drawgradients	Boolean. Draw gradient vectors for the growth polarising morphogen.
drawcmap	Boolean. Specifies whether to draw a colour bar at the foot of the plot, to show the mapping between colours and values.
cmap	Color map. Specifies how to map values to colours. This should be an array suitable for passing to Matlab's COLORMAP function. Default: For morphogens, minimum value = blue, maximum = red, with intermediate values going round the hue space. For labels, yellow for unlabelled, blue for labelled. For stress and strain: white 0% blue 10% green 20% yellow 30% red 40% purple 90% black 100% and above
crange	1x2 element array, specifying the range of values which is mapped to the colour range specified by

	cmap. This value will be supplied to Matlab's CRANGE function. The default is the range of the data, min(data)..0 or 0..max(data), whichever is narrowest and contains all the data.
plottensors	Boolean. Display the growth tensors based on morphogens 1 to 3 (growth, polarisation, and anisotropy). Default 0.
numbering	Boolean. If true, draw the number of each finite element cell. Default 0.
axisRange	Range of axes. This should be a 1x6 array [XLO,XHI,YLO,YHI,ZLO,ZHI] suitable for passing to Matlab's AXIS function. The default is chosen to ensure the entire leaf lies within the range.
axisVisible	Boolean: specifies whether to draw the axes. Default true.
alpha	Real number: transparency of the mesh. (0=invisible, 1=opaque.) Default 0.8 (slightly transparent).
autoScale	Boolean, default 1. If 1, axisRange will be ignored and the current value used.

leaf_plot stores all of the plotting options in the mesh, so that a subsequent call to leaf_plot with only the mesh as an argument will plot the same thing.

Equivalent GUI operation: none. The leaf is plotted automatically. Various options may be set in the "Plot options" panel, and the scroll bars on the picture change the orientation.

15.61 leaf_plotoptions

leaf_plot(m, ...)
Set default plotting options.
See LEAF_PLOT for details.

Equivalent GUI operation: plotting options may be set in the "Plot options" panel, and the scroll bars on the picture change the orientation.

m = leaf_recomputestages(m, ...)
Recompute a set of stages of the project, starting from the current state of m. If this is after any of the stages specified, those stages will not be recomputed.

Options:
'stages' A list of the stages to be recomputed as an array of numerical times. The actual times of the saved stages will be the closest possible to those specified, given the starting time and the time step. If this option is omitted, it will default to the set of stage times currently stored in m, which can be set by leaf_requeststages.

See also: leaf_requeststages

15.62 leaf_record_mesh_frame

15.63 leaf_rectangle

`m = leaf_rectangle(m, varargin)`
Create a new rectangular mesh.

Arguments:

M is either empty or an existing mesh. If it is empty, then an entirely new mesh is created, with the default set of morphogens. If M is an existing mesh, then its geometry is replaced by the new mesh. It retains the same set of morphogens (all set to zero everywhere on the new mesh), interaction function, and all other properties not depending on the specific geometry of the mesh.

Options:

'xwidth' The width of the rectangle in the X dimension. Default 2.
'ywidth' The width of the rectangle in the Y dimension. Default 2.
'xdivs' The number of finite element cells along the X dimension. Default 8.
'ydivs' The number of finite element cells along the Y dimension. Default 8.
'base' The number of divisions along the side with minimum Y value. The default is xdivs.

Example:

```
m = leaf_rectangle( [], 'xwidth', 2, 'ywidth', 2, 'xdivs', 8,  
                  'ydivs', 8, 'base', 5 )
```

See also: LEAF_CIRCLE, LEAF_CYLINDER, LEAF_LEAF, LEAF_ONECELL, LEAF_SEMICIRCLE.

Equivalent GUI operation: selecting "Rectangle" in the pulldown menu in the "Mesh editor" panel and clicking the "Generate mesh" button.

15.64 leaf_refineFEM

`m = leaf_refineFEM(m, amount)`

AMOUNT is between 0 and 1. Split that proportion of the edges of the finite element mesh, in random order.

Example:

```
m = leaf_refineFEM( m, 0.3 );
```

Equivalent GUI operation: clicking the "Refine mesh" button in the "Mesh editor" panel. The scroll bar and text box set the proportion of edges to be split.

15.65 leaf_reload

`m = leaf_reload(m, stage, varargin)`

Reload a leaf from the MAT, OBJ, or PTL file it was last loaded from, discarding all changes made since then. If there was no such previous file, the mesh is left unchanged.

Arguments:

stage: If 0 or more, the indicated stage of the mesh, provided there is a saved state from that stage. If 'reload', the

stage that the mesh was loaded from or was last saved to, whichever is later. If 'restart', the initial stage of the mesh. If the indicated stage does not exist a warning is given and the mesh is left unchanged. The default is 'reload'.

Options:

rewrite: Normally, when a model is loaded, its interaction function (if there is one) is read, parsed, and rewritten. This is because it may have been created with an older version of GFTbox. Specifying the rewrite option as false prevents this from being done. This may be necessary when running several simulations concurrently on a parallel machine, all using the same project.

Equivalent GUI operations: The "Restart" button is equivalent to

```
m = leaf_reload( m, 'restart' );
```

The "Reload" button is equivalent to

```
m = leaf_reload( m, 'reload' );
```

or

```
m = leaf_reload( m );
```

The items on the "Stages" menu are equivalent to

```
m = leaf_reload( m, s );
```

for each valid s. s should be passed as a string. For example, if the Stages menu has a menu item called 'Time 315.25', that stage can be loaded with

```
m = leaf_reload( m, '315.25' );
```

15.66 leaf_rename_mgen

```
m = leaf_rename_mgen( m, oldMgenName, newMgenName, ... )
```

Rename one or more morphogens. Any number of old name/new name pairs can be given. The standard morphogens cannot be renamed, and no morphogen can be renamed to an existing name.

Equivalent GUI operation: clicking the "Rename" button in the "Morphogens" panel.

See also:

```
LEAF_ADD_MGEN, LEAF_DELETE_MGEN
```

```
m = leaf_requeststages( m, ... )
```

Add a set of stage times to the mesh. None of these will be computed, but a subsequent call to leaf_recomputestages with no explicit stages will compute them.

Options:

'stages' A list of numerical stage times. These do not have to be sorted and may contain duplicates. The list will be sorted and have duplicates removed anyway.

'names' A cell array of names in 1-1 correspondence with the list of stage times. These names will appear along with the stage times on the Stages menu. They default to empty strings.

'mode' If 'replace', the list will replace any stage times

stored in `m`. If 'add' (the default), they will be combined with those present. If 'names' is not supplied or is the empty cell array, existing names will be retained.

GUI equivalent: Stages/RequestMore Stages... menu item. This does not support the 'names' or 'mode' options and always operates in 'add' mode.

15.67 leaf_rescale

```
m = leaf_rescale( m, ... )  
Rescale a mesh in space and/or time.
```

Arguments:

`M` is either empty or an existing mesh. If it is empty, then an entirely new mesh is created, with the default set of morphogens. If `M` is an existing mesh, then its geometry is replaced by the new mesh. It retains the same set of morphogens (all set to zero everywhere on the new mesh), interaction function, and all other properties not depending on the specific geometry of the mesh.

Options:

'spaceunitname'	The name of the new unit of distance.
'spaceunitvalue'	The number of old units that the new unit is equal to.
'timeunitname'	The name of the new unit of time.
'timeunitvalue'	The number of old units that the new unit is equal to.

If either `spaceunitname` or `timeunitname` is omitted or empty, no change will be made to that unit.

Example:

Convert a leaf scaled in microns to millimetres, and from days to hours:

```
m = leaf_rescale( m, 'spaceunitname', 'mm', ...  
                  'spaceunitvalue', 1000, ...  
                  'timeunitname', 'hour', ...  
                  'timeunitvalue', 1/24 );
```

Equivalent GUI operation: the 'Params/Rescale...' menu item.

15.68 leaf_rewriteIF

```
m = leaf_rewriteIF( m, ... )  
Rewrite the interaction function of m.
```

Normally the interaction function is rewritten the first time that it is called after loading a mesh. This is to ensure that it is always compatible with the current version of GFTbox. Sometimes it is necessary to prevent this from happening. In this case, if it is later desired to force a rewrite, this function can be called.

`leaf_rewriteIF` will do nothing if the i.f. has already been rewritten, unless the 'force' option has value true.

Note that a rewrite always happens when a morphogen is added, deleted, or renamed, or when the standard morphogen type (K/BEND or A/B) is changed.

Equivalent GUI operation: the Rewrite button on the Interaction function panel.

15.69 leaf_rotate

```
m = leaf_rotate( m, type1, rot1, type2, rot2, ... )
```

Rotate the mesh about the given axes. The rotations are performed in the order they are given: the order matters. Each type argument is one of 'M', 'X', 'Y', or 'Z' (case is ignored). For a type 'M', the following rotation should be a 3*3 rotation matrix. For 'X', 'Y', or 'Z' it should be an angle in degrees. Any sequence of rotations can be given.

See also: leaf_rotatexyz.

15.70 leaf_rotatexyz

```
m = leaf_rotatexyz( m, varargin )
```

Rotate the coordinates of the leaf: x becomes y, y becomes z, and z becomes x. If the argument -1 is given, the opposite rotation is performed.

Equivalent GUI operation: clicking on the "Rotate xyz" button in the "Mesh editor" panel.

See also: leaf_rotate.

15.71 leaf_saddlez

```
m = leaf_saddlez( m, ... )
```

Add a saddle-shaped displacement to the nodes of m.

Options:

'amount'	The maximum amount of the displacement, which is proportional to the distance from the origin. Default 1.
'lobes'	How many complete waves the displacement creates on the edge (an integer, minimum value 2). Default 2.

Example:

```
m = leaf_saddlez( m, 'amount', 1, 'lobes', 2 );
```

Equivalent GUI operation: the "Saddle Z" button on the "Mesh editor" panel. The amount of saddle deformation is specified by the value in the upper of the two text boxes to the right of the button. The number of waves is specified by the lower text box.

15.72 leaf_save

```
m = leaf_save( m, filename, folderpath, ... )
```

Save the leaf to a file.

The way the leaf is saved depends on the extension of the filename:

- .MAT The leaf is saved in a MAT file as a Matlab object called `m`.
- .M Matlab commands to recreate this leaf are saved in a .M file.
- .OBJ Only the nodes and triangles are saved, in OBJ format.
- .FIG The current plot of the leaf is saved as a figure file.

All of these formats except FIG can be read back in by `leaf_load`. Note that OBJ format discards all information except the geometry of the mesh.

If the filename is just an extension (including the initial "."), then a filename will be prompted for, and the specified extension will be the default. If no filename is given or the filename is empty, then one will be prompted for, and any of the above extensions will be accepted.

The folder path specifies what folder to save the file in. If not specified, then the default folder will be chosen, which depends on the file extension. If a filename is then prompted for, the file dialog will open at that folder, but the user can navigate to any other. If the filename is a full path name then the folder name will be ignored and the file will be stored at the exact location specified by filename.

Options:

- `overwrite`: If true, and the output file exists already, it will be overwritten without warning. If false (the default), the user will be asked whether to overwrite it.
- `minimal`: For OBJ files, if this is true, then only the vertexes and triangles of the mesh will be written. OBJ files of this form should be readable by any program that claims to read OBJ files. If false (the default), all of the information in the mesh will be written, in an ad-hoc extension of OBJ format.

Equivalent GUI operations: the "Save model..." button (saves as MAT file) or the "Save script...", "Save OBJ...", or "Save FIG..." menu commands on the "Mesh" menu.

```
[m,ok] = leaf_savemodel( m, modelname, projectdir, ... )  
Save the model to a model directory.
```

MODELNAME is the name of the model folder. This must not be a full path name, just the base name of the folder itself. It will be looked for in the folder PROJECTDIR, if specified, otherwise in the parent directory of `m`, if any, otherwise the current directory.

If MODELNAME is not specified or empty, the user will be prompted for a name using the standard file dialog.

The model directory will be created if it does not exist.

If the model is being saved into its own model directory:

- If it is in the initial state (i.e. no simulation steps have been performed, and the initialisation function has not been called) then it is saved into the file MODELNAME.mat.
- If it is in a later state, it will be saved to MODELNAME_Snnnn.mat, where nnnn is the current simulation time as a floating point

number with the decimal point replaced by a 'd'.

If the model is being saved into a new directory:

The current state will be saved as an initial state or a later stage file as above.

If the current state is not the initial state, then the initial state will be copied across. Furthermore, the initial state of the new project will be loaded.

The interaction function and notes file will be copied, if they exist. If the notes file exists, the new notes file will also be opened in the editor.

Stage files, movies, and snapshots are NOT copied across.

If for any reason the model cannot be saved, OK will be false.

Options:

new: If true, the mesh will be saved as the initial state of a project, even if it is not the initial state of the current simulation. The default is false.

strip: If true, as many fields as possible of the mesh will be deleted before saving. They will be reconstructed as far as possible when the mesh is loaded. The only information that is lost is the residual strains and effective growth tensor from the last iteration. The default is false.

Equivalent GUI operation: the "Save As..." button prompts for a directory to save a new project to; the "Save" button saves the current state to its own model directory. The "strip" option can be toggled with the "Misc/Strip Saved Meshes" menu command.

15.73 leaf_semicircle

```
m = leaf_semicircle( m, ... )
```

Create a new semicircular mesh.

Arguments:

M is either empty or an existing mesh. If it is empty, then an entirely new mesh is created, with the default set of morphogens. If M is an existing mesh, then its geometry is replaced by the new mesh. It retains the same set of morphogens (all set to zero everywhere on the new mesh), interaction function, and all other properties not depending on the specific geometry of the mesh.

Options:

'radius' The radius of the semicircle. Default 1.
'rings' The number of circular rings of triangles to divide it into. Default 4.

Example:

```
m = leaf_semicircle( [], 'radius', 2, 'rings', 4 );
```

See also: LEAF_CIRCLE, LEAF_CYLINDER, LEAF_LEAF, LEAF_ONECELL, LEAF_RECTANGLE.

Equivalent GUI operation: selecting "Semicircle" in the pulldown menu in the "Mesh editor" panel and clicking the "New" button.

15.74 leaf_set_mousemode

```
m = leaf_set_mousemode( m, varargin )
    Set the effect of clicking on the mesh.
```

Arguments

1: the name of the mode. This is a string, the name of the required mode.

Options:

Dependent on the mode. Some modes take extra parameters and some take none.

Whether a click is considered to be on a finite element, a biological cell, an edge, or a node is determined by the click mode. When an edge or a node is required to be clicked, a click on a cell will be interpreted as a click on the nearest edge or node.

The possible arguments and their options are as follows:

- 'morphadd': Clicking adds the current amount to the current morphogen at the clicked vertex.
- 'morphset': Clicking sets the current amount of the current morphogen at the clicked vertex.
- 'morphfix': Clicking fixes the amount of the current morphogen at the clicked vertex to its current value, regardless of the effects of diffusion or dilution. Clicking to add or set the morphogen is still effective, as are modifications made by the interaction function.
- 'morphshow': Clicking displays the value of the quantity currently being plotted. If the value is defined per vertex, the value at the vertex nearest to the click point will be shown; if per element, the value for the element clicked will be shown. If the GUI is running, the value will be displayed there, otherwise printed to the command window.
- 'Fix node': Clicking fixes or unfixes one or more degrees of freedom of movement of the node, as specified by the optional argument, which a string containing any of the letters 'xyz'. If the string is empty, the node becomes unfixed. If the string is nonempty, the node becomes unfixed if its current fixed degrees of freedom coincide with the string; otherwise, the degrees of freedom specified by the string become fixed and the others unfixed.
- 'Locate node': Clicking nominates this node to remain stationary with respect to the specified degrees of freedom. This differs from 'Fix node' in that no more than one node can be specified for each degree of freedom, and the mesh is rigidly translated so as to maintain the constraint.
- 'deletecell': clicking on a cell deletes it from the mesh.
- 'seam': Toggle an edge between being a seam or not.
- 'split': Toggle an edge between being designated for splitting or not.

```
\subsection{leaf\_set\_userdata}\label{section-leaf-set-userdata}
```

```
\begin{verbatim}
```

```
m = leaf_set_userdata( m, ... )
    Set fields of the userdata of m. The arguments should be alternately a
    field name and a field value.
```

You can store anything you like in the userdata field of the canvas. The growth toolbox will never use it, but your own callbacks, such as the morphogen interaction function, may want to make use of it.

See also: LEAF_ADD_USERDATA, LEAF_DELETE_USERDATA.

Equivalent GUI operation: none.

15.75 leaf_setbgcolor

```
m = leaf_setbgcolor( m, color )
```

Set the background colour of the picture, and of any snapshots or movies taken. COLOR is a triple of RGB values in the range 0..1.

15.76 leaf_setgrowthmode

```
m = leaf_setgrowthmode( m, mode )
```

Specify whether growth is described by growth and anisotropy, or by growth parallel and perpendicular to the polarisation gradient. Allowable values for MODE are 'ga' or 'pp' respectively.

THIS FUNCTION HAS BEEN WITHDRAWN, 2008 May 15.

15.77 leaf_setmutant

```
m = leaf_setmutant( m, ... )
```

Set the mutant level of a morphogen.

Options:

morphogen: The name or index of a morphogen. If omitted, the mutation properties are set for every morphogen.
value: The value the morphogen has in the mutant state, as a proportion of the wild-type state.

Examples:

```
m = leaf_setmutant( m, 'morphogen', 'div', 'value', 0 );  
% Set the mutated level of the 'div' morphogen to zero.
```

15.78 leaf_setproperty

```
m = leaf_setproperty( m, ... )
```

Set global properties of the leaf.

The arguments are a series of name/value pairs.

The property names that this applies to are:

'poisson'	Poisson's ratio. The normal value is 0.35 and there is little need to change this.
'bulkmodulus'	The bulk modulus. The normal value is 3000 and there is little need to change this.
'cvtperiter'	The amount of CVT transformation to be carried out per iteration.
'jiggle'	The amount of jiggling of the biological cells to

be carried out per iteration.
 'validate' Whether to validate the mesh after every iteration.
 This is for debugging purposes and should normally
 be off.
 'displayedgrowth' Specifies which morphogen to plot.

...and many others I have omitted to document.

Example:

```
m = leaf_setproperty( m, 'poisson', 0.49 );
```

Equivalent GUI operation: several GUI elements are implemented by this command:

poisson: Text box in "Mesh editor" panel.
 bulkmodulus: No GUI equivalent.
 residstrain: "Strain retention" in "Simulation" panel.
 cvtperiter: "CVT per iter" in "Simulation" panel.
 jiggle: "Jiggle" in "Simulation" panel.
 validate: No GUI equivalent.
 displayedgrowth: "Displayed m'gen" menu in "Morphogens" panel.
 ...etc.

15.79 leaf_setsecondlayerparams

```
m = leaf_setsecondlayerparams( m, varargin )
```

Set various general properties of the second layer. If the second layer does not exist an empty second layer will be created, and the properties set here will be the defaults for any subsequently created nonempty second layer.

If m already has a second layer, this procedure does not affect the colours of existing cells, only the colours that may be chosen by subsequent recolouring operations.

Options:

colors: An N*3 array of RGB values. These are the colours available for colouring cells. The special value 'default' will use the array [[0.1 0.9 0.1]; [0.9 0.1 0.1]]. N should be 2. "Ordinary" cells will be coloured with the first colour, while "shocked" cells will be coloured with the second colour.
 colorvariation: A real number between 0 and 1. When a colour for a cell is selected from the colour table, this amount of random variation will be applied to the value selected from colors. A suitable value is 0.1, to give a subtle variation in colour between neighbouring cells.

15.80 leaf_setthicknessparams

```
m = leaf_setthicknessbyarea( m, value )
```

Set the thickness of the leaf as a function of its current area:

$$\text{thickness} = K \cdot \text{area}^{(P/2)}.$$

K may have any positive value. P must be between 0 and 1.

Options:

'scale' K. Default is 0.5.
'power' P. Default is 0.

15.81 leaf_setzeroz

`m = leaf_setzeroz(m)`

Set the Z displacement of every node to zero.

Equivalent GUI operation: the "Zero Z" button on the "Mesh editor" panel.

15.82 leaf_shockA

`m = leaf_shockB(m, amount)`

AMOUNT is between 0 and 1. Mark that proportion of randomly selected cells of the A layer with random colours. At least one cell will always be marked. If there is no A layer, the command is ignored.

Example:

`m = leaf_shockA(m, 0.3);`

Equivalent GUI operation: "Shock cells" button on the Bio-A panel.

The accompanying slider and text box set the proportion of cells to shock.

15.83 leaf_shockB

`m = leaf_shockB(m, amount)`

AMOUNT is between 0 and 1. Mark that proportion of randomly selected cells of the B layer with random colours. At least one cell will always be marked. If there is no B layer, the command is ignored.

Example:

`m = leaf_shockB(m, 0.3);`

Equivalent GUI operation: "Shock cells" button on the Bio-B panel.

The accompanying slider and text box set the proportion of cells to shock.

15.84 leaf_showaxes

`m = leaf_showaxes(m, axeson)`

Make the axes visible or invisible, according as AXESON is true or false.

15.85 leaf_snapdragon

`m = leaf_snapdragon(m, ...)`

Make an early stage of a snapdragon flower. This consists of a number of petals, each of which consists of a rectangle surmounted by a semicircle. The rectangular parts of the petals are connected to form a tube. The mesh is oriented so that the cell normals point outwards.

Arguments:

M is either empty or an existing mesh. If it is empty, then an entirely new mesh is created, with the default set of morphogens

If M is an existing mesh, then its geometry is replaced by the new mesh. It retains the same set of morphogens (all set to zero everywhere on the new mesh), interaction function, and all other properties not depending on the specific geometry of the mesh.

Options:

'petals'	The number of petals. Default 5.
'radius'	The radius of the tube. Default 1.
'rings'	The number of circular rings of triangles to divide the semicircular parts into. Default 3.
'height'	The height of the rectangle, as a multiple of the semicircle's diameter. Default 0.7.
'base'	The number of divisions along half of the base of each petal. By default this is equal to rings, i.e. the same as the number at the top of the tube.
'strips'	The number of strips of triangles to divide the tubular part into. If 0 (the default), this will be calculated from the height so as to make the triangles similar in size to those in the lobes.

Example:

```
m = leaf_snapdragon( [], 'petals', 5, 'radius', 2, 'rings', 4 );  
See also: LEAF_CIRCLE, LEAF_CYLINDER, LEAF_LEAF, LEAF_ONECELL,  
LEAF_RECTANGLE, LEAF_LOBE.
```

15.86 leaf_snapshot

```
m = leaf_snapshot( m, filename, ... )
```

Take a snapshot of the current view of the leaf into an image file. A name for the image file will be automatically generated if none is given.

Arguments:

1: The name of the file to write. The extension of the filename specifies the image format. This may be any format acceptable to the Matlab function IMWRITE. These include 'png', 'jpg', 'tif', and others.

Options:

'newfile': if true (the default), the file name given will be modified so as to guarantee that it will not overwrite any existing file. If false, the filename will be used as given and any existing file will be overwritten without warning.

'thumbnail': if true (the default is false), the other arguments and options will be ignored (the filename must be given as the empty string), and a snapshot will be saved to the file thumbnail.png in the project directory.

All remaining arguments will be passed as options to IMWRITE. Any arguments taken by IMWRITE may be given. If any such arguments are provided, the filename must be present (otherwise the first argument for IMWRITE would be taken to be the filename). If you do not want to provide a filename, specify it as the empty string. The image will be saved in the 'snapshots' folder of the current project folder, if any, otherwise the current folder. You can override this by specifying an absolute path.

Example:

```
m = leaf_snapshot( m, 'foo.png' );
```

Equivalent GUI operation: clicking the "Take snapshot" button. This saves an image in PNG format into a file with an automatically generated name. A report is written to the Matlab command window. The 'thumbnail' option is equivalent to the "Add Thumbnail" menu command.

See also:

IMWRITE

15.87 leaf_spin

```
m = leaf_spin( m )
```

Spin the mesh by 360 degrees about the Z axis, leaving it in exactly the same orientation as when it started. If a movie is currently being recorded, the animation will be appended to the movie. The current view is assumed to have already been written to the movie.

Options:

'frames': The number of frames to be added. Each frame will rotate the mesh by 360/frames degrees.

15.88 leaf_splitbio

```
m = leaf_splitbio( m )
```

Split all biological cells that are currently too large.

Equivalent GUI operation: "Split cells" button.

15.89 leaf_splitsecondlayer

```
m = leaf_splitsecondlayer( m )
```

Split every cell in the second layer. Reset the splitting threshold to make the new cell sizes the target sizes.

Equivalent GUI operation: "Split L2" button.

15.90 leaf_stitch_vertex

```
m = leaf_stitch_vertex( m, dfs )
```

Constrain sets of vertexes of the mesh so that they move identically.

Arguments:

dfs: a cell array of vectors of degree of freedom indexes. Dfs in the same vector will be constrained to change identically. No index may occur more than once anywhere in dfs.

Equivalent GUI operation: none.

15.91 leaf_subdivide

```
m = leaf_subdivide( m, ... )
```

Subdivide every edge of `m` where a specified morphogen is above and/or below thresholds, and the length of the current edge is at least a certain value.

NB. This function is currently NOT suitable for calling from an interaction function. It will go wrong.

Note that this command will subdivide every eligible edge every time it is called. It does not remember which edges it has subdivided before and refrain from subdividing them again.

Options:

- 'morphogen': The name or index of the morphogen
- 'min': The value that the morphogen must be at least equal to.
- 'max': The value that the morphogen must not exceed.
- 'mode': 'all' [default], 'any', or 'mid'.
- 'minabslength': A real number. No edge shorter than this will be subdivided.
- 'minrellength': A real number. This is a fraction of the current threshold for automatic splitting of edges. No edge shorter than this will be subdivided. The current threshold value is returned by `currentEdgeThreshold(m)`.
- 'levels': The levels of morphogen the new vertices will adopt 'all', 'interp', 'none'

An edge will be subdivided if and only if it satisfies all of the conditions that are specified. Any combination of the arguments can be given. No arguments gives no subdivision.

'mode' is only relevant if 'min' or 'max' has been specified. If mode is 'all', then each edge is split for which both ends satisfy the min/max conditions. If mode is 'any', each edge is split for which either edge satisfies the conditions. If mode is 'mid', each edge is split for which the average of the morphogen values at its ends satisfies the conditions.

This command ignores the setting, that can be set through the GUI or `leaf_setproperty()`, that enables or disables automatic splitting of long edges.

15.92 leaf_unshockA

```
m = leaf_unshockA( m )
```

Restore all cells of the Bio-A layer to their unshocked state.

Example:

```
m = leaf_unshockA( m );
```

Equivalent GUI operation: "Unshock all cells" button on the Bio-A panel.

15.93 leaf_vertex_monitor

```
setaxis(gca, [0, ax(2), ax(3)-0.5, ax(4)+0.5]);
```

15.94 leaf_vertex_set_monitor

```
case 'MARK'  
    marker=arg{2};  
    region=arg{1};
```
