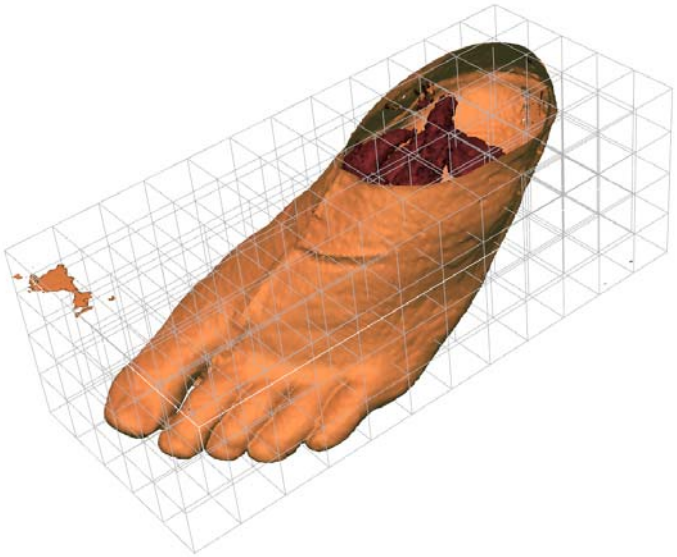
Auckland Bioengineering Institute
An Institute of The University of Auckland

Auckland Bioengineering Institute
CMISS CMGUI tutorial



Part of the IUPS 2009 Physiome/VPH projects tools tutorial

1 Table of Contents

Auckland Bioengineering Institute CMISS CMGUI tutorial

2 Getting started with CMGUI

This section will introduce you to the capabilities of CMGUI, as well as how to install and begin using the software.

2.1 Where to go for help

All of the content in this handout is available on our website, as well as additional documentation on the user interface, information about CMGUI's inner workings, and further examples.

Documentation on using CMGUI is available at:

```
http://www.cmiss.org/cmgui/wiki/UsingCmgui
```

There is a large collection of examples at:

```
http://cmiss.bioeng.auckland.ac.nz/development/examples/a/index_thumbs.html
```

Issues with the software can be reported and discussed on the tracker, which is found at:

```
https://tracker.physiomeproject.org
```

2.2 What is CMGUI and what can it do?

CMGUI is an advanced 3D visualization open source software package with modelling capabilities. It originated as part of CMISS, a collection of software for "Continuum Mechanics, Image processing, Signal processing and System identification". CMISS includes **cm**, a closed-source application for finite element method computation and related computation. Historically CMGUI has been mostly used in conjunction with cm. OpenCMISS-cm (www.opencmiss.org) is an open source project, currently under development, aiming to replace the closed source CMISS-cm.

A gallery demonstrating some of the uses of CMGUI is available here:
http://cmiss.bioeng.auckland.ac.nz/development/examples/a/index_thumbs.html

Some of the main areas of functionality of CMGUI are as follows:

- 3D visualization of finite element and boundary element meshes
- Mesh creation
- Mathematical field visualization and manipulation

CMGUI is mainly controlled from its built in "command line" interpreter. Usually script files are read or commands are typed in the CMGUI command window to read in your mesh, create a 3D visualisation and manipulate it. Many tasks can also be accomplished by using the mouse to select options from various menus and dialogs. CMGUI has a large amount of functionality but it can be difficult for a new user to figure out what command to use to accomplish exactly what they want to do. Fortunately there are a large number of examples demonstrating the use of various different commands.

CMGUI is also used as part of the Zinc extension to Mozilla Firefox. The Zinc extension allows users to view Zinc applications in Firefox. A Zinc application uses CMGUI for visualization while providing a nice

customized user interface for a specific task. For example Zinc applications have been written to do the following:

- Interactively explore Colon endoscopy
- Explore the Physiome eye model
- Create data points to give a digital description of a geometry based on medical images.

Note that Zinc is compiled as a separate application from CMGUI. You do NOT have to install CMGUI to use Zinc. For more information on Zinc see: http://www.cmiss.org/CMGUI/zinc

2.3 Installation

CMGUI is currently available for download for a wide variety of platforms. You can download the appropriate executable for your operating system from the Release centre: http://www.cmiss.org/ReleaseCenter/CMGUI.

CMGUI is quite a large application so the executable has been archived (tarred) and compressed (zipped) to make it faster to download. Once you have downloaded the executable all you need to do is untar and unzip it and it is ready to run. On Linux you can untar and unzip the tar.gz file with the command: `tar -xf filename.tar.gz.`

On Windows you will need WinZip or something similar to unzip the file. You should now be able to run CMGUI by either clicking on the executable or by changing into the directory it is located in and then typing `./` followed by the name of the executable at the command prompt, eg `./CMGUI-wx.` This will bring up the CMGUI command window interface.

For convenience it is a good idea to place the directory containing CMGUI into your PATH environmental variable. That way you will be able to run CMGUI without having to be in the directory that contains the executable. To do this on a Linux system you can specify your PATH variable in your .profile file. On Windows you can edit your PATH variable by right clicking on my computer and then selecting: properties, advanced, environment variables. Ask your local system administrator if you do not know how to edit the PATH variable for your operating system.

3 The CMGUI windows

The CMGUI interface consists of a number of different windows:

- Command window
- Scene editor
- Graphics window
- Material editor
- Spectrum editor

3.1 Command window

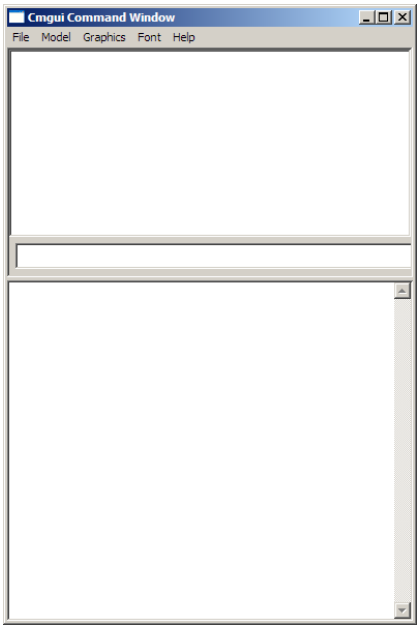


Figure 1: The command window

This is the first window that appears when CMGUI is loaded. It consists of standard menus at the top, and three panels below. From top to bottom these are the history, command line, and output panels. The history panel is where all commands that have been executed are displayed. The second, single-line panel is the command line where commands may be entered directly. The lowermost panel is the output panel; this displays the text output of commands, error messages, as well as help information.

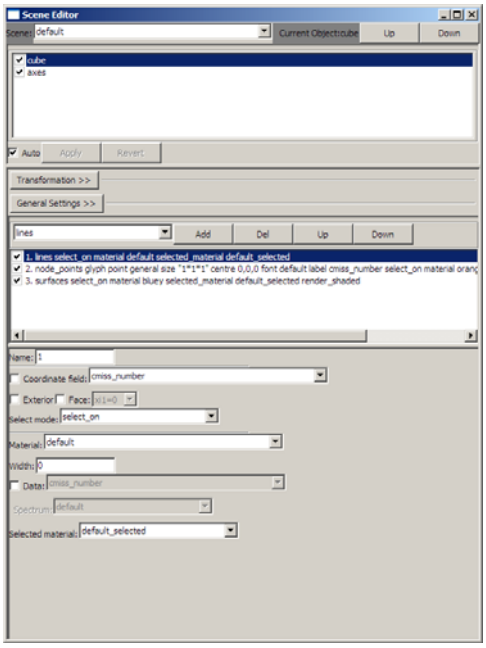
The history and output areas have scrollbars that allow you to view all of the contents of these panels. In the case of the history panel, this enables you to scroll back and click on commands - clicking on a command in the history panel will enter that command into the command line. This command can then be edited and executed. In the case of lengthy commands, this can save a lot of typing. Double-clicking on a command in the history window will execute it immediately.

The output panel is where any text output from commands or error messages appears. If something is not working in CMGUI, this panel is the first place to look for a reason. This panel also displays the help information when you a execute command with the `?` or `??` argument.

3.2 Scene editor

The scene editor is where you control how visualizations appear in the graphics window. From this window you can select which graphical representations and settings are displayed. You can add, remove, and edit visual elements such as lines, glyphs or surfaces; apply materials or spectra to graphical settings; and alter the order in which graphical representations are drawn. The window itself is broken into three main panels: (Figure 2)

- Scene objects list
- Graphical settings list
- Settings editor



Scene object list

Graphical settings list

Settings editor

Figure 2: The scene editor window. This window allows you to control the visibility and visual appearance of the objects and *graphical settings* in a *scene*.

Scene object list

At the top of the scene objects list is a drop-down menu which usually contains the word *default*. This is the scene selection menu, which you can use to swap between scenes. The *default* scene is present when you load CMGUI, and is the default scene in which scene objects are created. There are *up* and *down* buttons to the right of this menu, which allow you to change the order of the objects in the *scene object list*. This is important when using transparent objects to ensure they are correctly rendered.

Below the *scene object list* is the *transformation* button. This button will open up a set of controls that allow you to spatially manipulate the selected scene object. From these controls you can scale, translate or rotate the object within the scene.

If a scene object made up of elements is selected, the *general settings* button will become available. This button opens a series of settings that allow you to edit the coordinate field and discretization options for the selected object. The discretization options allow you to change the level of detail objects are represented within the graphics window (Figure 3).

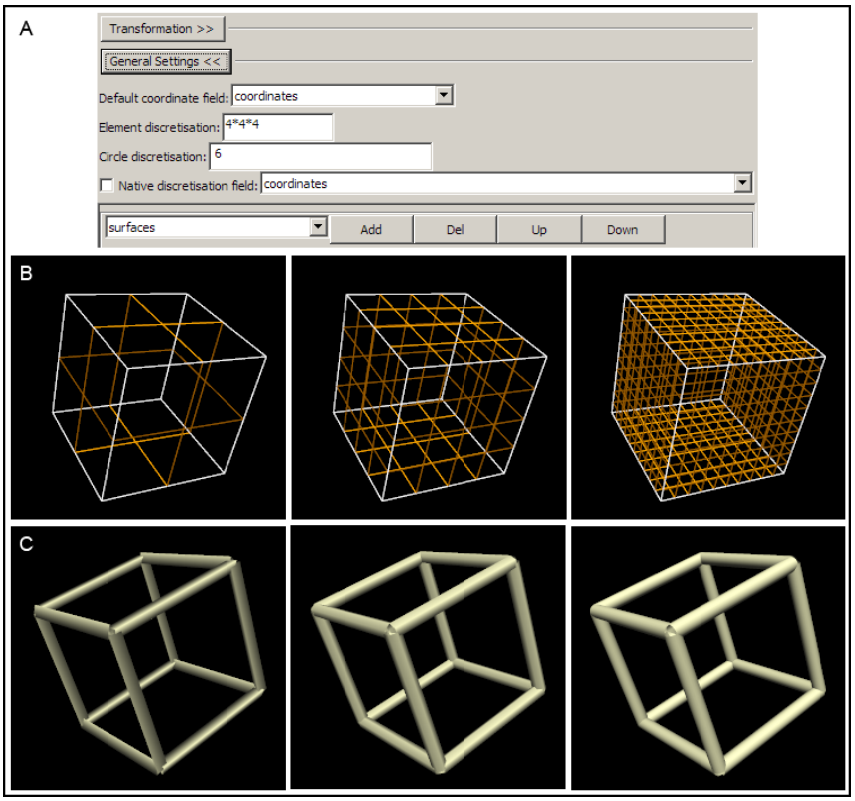


Figure 3: The general settings. A) This is the general settings panel, showing the settings available. B) A simple cube rendered in wireframe to show the effect of the element discretization. The three cubes have 2*2*2, 4*4*4 (default), and 10*10*10 discretizations from left to right. C) The effect of circle discretization on the display of cylinders. These three cubes are rendered with cylinder edges, with the circle discretization set at 3, 6 (default), and 20 from left to right.

Graphical settings list

Below the transformation and general settings buttons is the *graphical settings list*. This is where the visual representations (*graphical settings*) of the currently selected scene object are listed. This panel allows creation, deletion, visibility switching (on or off) and re-ordering of these visual representations.

Settings editor

Below the *graphical settings list* is the *settings editor* where each graphical setting can be edited. When a graphical setting is selected from the list, all of its editable properties appear in this area. The range of editable properties will vary depending on what sort of graphical setting is currently selected. Graphical settings will be covered in more detail later in this document.

3.3 Graphics window

The graphics window is where all visualizations are displayed - it also has tools which allow some interactive manipulation of the data being visualized. The window consists of a control panel on the left hand side, and the display area on the left. The display area can contain one or more *scene viewers*, or renderings of the selected scene. At the top of the control panel area are a selection of general controls under the "Options" label: view all, save as, and perspective. The *view all* button will zoom out the viewing area *(scene viewer)* so that everything in the currently viewed scene is visible. The *save as* option provides the ability to save the viewing area as a raster graphic, such as a `png` or `jpg` file. The *perspective* check box switches convergence on or off in the 3D display.

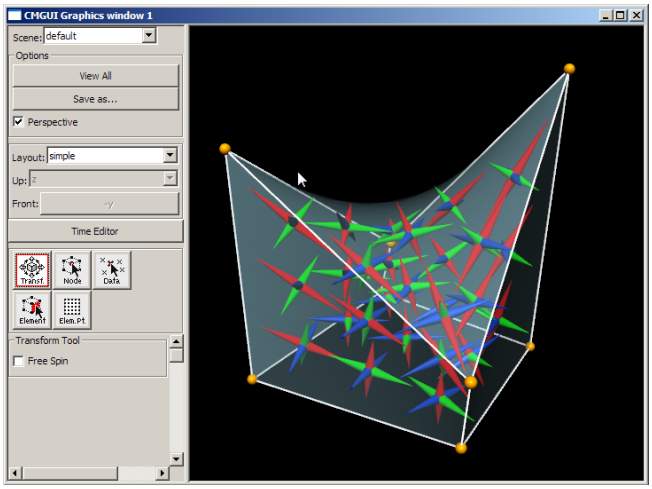


Figure 4: The graphics window, showing a single scene viewer in the display area on the right. The controls and tools in the left hand panel are used to manipulate the display area.

Immediately underneath the options section are controls for selecting how the 3D display is configured. The *Layout* drop down list contains a list of display layouts, such as orthogonal or pseudo-3D views. If an applicable layout is selected, the *up* and *front* controls will become available in order to change the viewpoint. Many of these layouts contain multiple *scene viewers*, which may be useful in specific situations. The default layout *simple* consists of a single 3D scene viewer.

Below the layout section is a button labelled *Time Editor*. This opens the timeline controls (much like the controls in a media player) that allow you to play animations if they have been set up.

Below this are the 5 "Tools" buttons. These offer different ways of interacting with the 3D display.

Transformation mode

This is the default mode, and is used for simple viewing of the graphical representations you are working on. In this mode the objects in the 3D window can be rotated, translated and zoomed using the mouse. Holding the left mouse button within the 3D view and moving it around will rotate or tumble the view. Holding down the right mouse button and moving the mouse up and down will zoom the view in and out, and holding the middle button will allow you to translate the view around along two axes. It is useful to spend some time getting used to the way these manipulations work.

In CMGUI, the rotate function works slightly differently from how similar view manipulations work in most software. This may not be immediately obvious, as the function does not "feel" particularly different in use; nevertheless there are some useful features of CMGUI's particular technique for rotating the view. Essentially, where you initially click in the view is the "handle" which you then move around by moving the mouse. In CMGUI this handle is different to most applications, in that it is like "grabbing" a point on a sphere that bounds the object in the 3D window. This allows manipulations using the rotate function that

are not possible in most 3D views, such as rotating the object around an arbitrary axis, or rotating it in a circular fashion around the centre of the view. These abilities can be useful when looking at data that has aligned features.

The four other tools available are used for the selection and limited editing of the type of item they refer to. Selected items are able to be targeted by commands input to the command line, or edited from within the *graphics window*.

When any of the following four tools is selected, holding down the `Ctrl` key will temporarily switch you back into transformation mode in order to manipulate the view.

Node Tool

The node tool allows the selection and editing of individual nodes from within the graphics window. Selected nodes will turn red by default - the selected colour of a node is editable via the *scene editor* window. There are a range of other options to allow the editing, (moving nodes within the scene viewer) deletion, or creation of nodes. It is also possible to create 1D, 2D or even 3D (lines, surfaces and volumes) elements using the node tool; this functionality is somewhat experimental and is of limited use in most cases.
NOTE: It is generally easier to edit nodes (or indeed any of the other editable items) when they are represented by an easily clickable glyph such as a sphere or cube, rather than a point. Changing node glyphs is achieved from the *scene editor* window.

Data Tool

The data tool allows you to select data points and edit them in the same ways that nodes are editable, with the exception of element creation/destruction which is only available in the node tool.

Element Tool

The element tool allows you to select and destroy elements. You can also set up filters that allow only the selection of line, face or volume elements. It is worth noting that volume elements have no indication of their selection unless they contain element points, which will turn red when the volume element they exist in is selected.

Element Point Tool

The element point tool allows the selection of element points within the scene viewer/s.

Note that when using any of the four non-transformation tools, it is possible to perform transformations by holding down the *Ctrl* key.

3.4 Material editor

The material editor window is where you define materials to be applied to graphical elements or objects in the graphics window. Along the top of the material editor window are three buttons; create, delete and rename. You can create a new material, delete or rename an existing material using these buttons. Below these buttons is the list of currently defined materials. These will contain the default materials, as well as any defined in any comfile that has been run.

Below the material list is a panel containing four colour editors. These control the ambient, diffuse, emitted and specular colours.

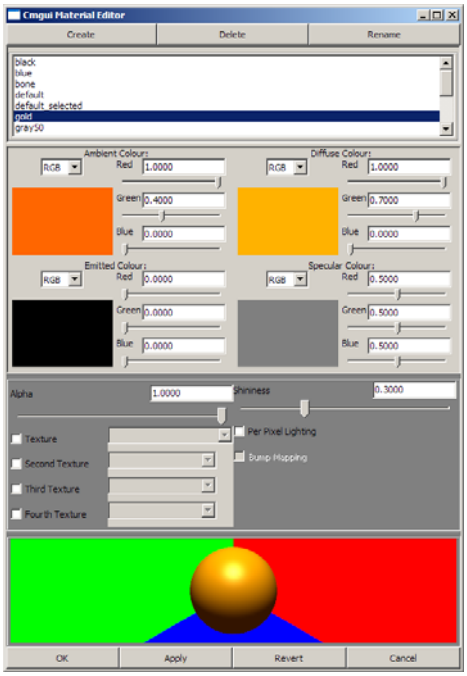
- Ambient - The ambient colour is the “unlit” colour, or the colour of parts of the object that are in shadow.
- Diffuse - This is the overall colour of the material, the colour that the lit parts of the object will appear.
- Emitted - The emitted colour is the “glow” of a material; this colour will appear in both the lit and unlit parts of the material.
- Specular - This is the colour of the shine that appears on the material. This shine appears as a glossy highlight.

Each colour can be edited using three sliders or textboxes, and you can choose from three colour-spaces for editing the colour. The default is RGB, but the HSV and CMY colour-spaces are also available from a pull-down menu at the top of each colour editor. Each colour editor has a preview panel showing a flat sample of the colour that is being edited.

Below the colour editors is the surface editor. This panel allows you to set the alpha, shininess, and texture properties of the surface of the material being edited. The alpha value sets the transparency of the material. The shininess sets the "tightness" or size of the specular highlights of a material; generally the higher the shininess, the smaller and harder-edged the highlights. Higher shininess makes a material look glossier. The surface editor also allows you to assign a texture to the material surface - this option is unavailable unless you have created at least one texture by using the "`gfx create texture`" command to read in some bitmapped graphics. There are two other check box options available in the surface editor: *per pixel shading* which greatly increases the quality of the lighting of the material, and *bump mapping* which allows you to use a texture file to create surface details. These options are only available on hardware that supports advanced forms of shading.

Below the surface editor is a panel that shows a preview of the currently edited material applied to a sphere. You may need to resize the material editor window in order to see a usefully large preview panel. Clicking in the preview panel will cycle the background from the triad of RGB colours through black, white and back to RGB.

At the bottom of the window are four buttons: *OK*, *apply*, *revert* and *cancel*. The *OK* button leaves the editor and applies the changes made. The *apply* button immediately applies current changes to the materials, allowing you to see how they look if they are used in any objects shown in the graphics window. The *revert* button undoes any changes made to the currently edited material, and the *cancel* button exits the material editor window without applying changes.



Material list

Colours editor

Surface editor

Preview

Figure 5: The material editor window

3.5 Spectrum editor

The spectrum editor window is where you define spectra to be applied to graphical elements or objects in the graphics window. Spectra are used to visualize continuous data ranges within models using colour ranges, and can be applied to the graphical settings that have been used to create your visualization. The window is divided into three basic areas; the spectrum list, the preview panel, and the settings editor.

It is useful to step through example a7 to get a feel for the use of spectra in CMGUI. This example uses a mesh containing a number of fields that can be usefully visualized using spectra.

Spectrum List

The spectrum list shows all the currently defined spectra; it will always contain the *default* spectrum if no others have been defined. Three buttons at the top of the window allow you to *create*, *delete*, or *rename* spectra. Just below the list of spectra are some controls that allow you to set some of the general properties of the selected spectrum. The *autorange* button automatically sets the minimum and maximum values of the selected spectrum, according to the smallest and largest values it has been applied to in the scene editor. For example, if the default spectrum has been used to colour a temperature field in the default scene, and that field has values ranging from 0 to 100 degrees Celsius, pressing the *autorange* button will set the minimum and maximum values (in the *Spectrum range* settings - see below) of the selected spectrum to 0 and 100 respectively. If the selected spectrum has been used to colour objects according to more than one field, the *autorange* function will choose the smallest and largest values across all of these fields. The *from scene* drop-down menu allows you to select the scene from which the spectrum will be auto-ranged.

The *overlay* or *overwrite* options allow you to choose whether a spectrum will completely over-ride any other material settings (*overwrite*), therefore completely colouring the object as the spectrum appears in the preview panel - or whether the spectrum combines with the other material settings of the object so that the final colour is a combination of the spectrum and other material settings (*overlay*).

Preview Panel

This panel shows a horizontal bar, coloured using the selected spectrum. The bar also shows the range that the spectrum is currently set to, using a series of numbered labels. Clicking in the preview panel changes the number of these labels from 4 to 10 to 2, then cycles through these values. Currently, the preview panel cannot display multi-component spectra.

Settings Editor

The settings editor is where each spectrum is set up. It contains a number of controls.

- **Spectrum component list:** The top of this list has four buttons; *Add*, *Delete*, *Up*, and *Down*. Below these buttons is a list of the components that make up the selected spectrum. Spectra in CMGUI can be made up of multiple components; these can be added, deleted or re-ordered using this list. Using these "sub-spectra" you are able to create spectra that have different colour ranges for different parts of the data range they cover, or spectra that have different colour ranges for different dimensions.
- **Data component:** This text box allows you to enter which data component of a multi-component field the spectrum will colour according to.
- **Spectrum range:** This set of controls is used to set up the range of values the spectrum covers. The *Min.* and *Max.* text boxes allow you to enter values for the minimum and maximum values of the spectrum. There are also four check boxes that allow you to change the behaviour of a spectrum at its start and end points.
- **Colour:** This drop-down menu provides a selection of pre-set colour settings that can be applied to the currently selected spectrum component. Although many of the colour ranges are fairly self-explanatory, a number of them have special features which are useful for creating specialized spectra. The *contour bands* and *step* colour settings have further settings associated with them.

    - Rainbow: This is a standard rainbow colour range.
    - Red: This is a red to black colour range. It is applied as a single "channel" of red to the spectrum; this can be used to add to other spectrum components to make multi-component spectra.

- Green: This is a single channel green to black colour range.
- Blue: This is a single channel blue to black colour range.
- White to blue: This is a white to blue colour range.
- White to red: This is a white to red colour range.
- Monochrome: This is a black to white colour range.
- Alpha: This is a single channel transparent to opaque spectrum.
- Contour bands: This colour option creates a series of evenly spaced black bands that can be further edited to get the correct number, width and spacing desired.
- Step: This is a colour range which has a sharp transition from saturated red to saturated green.

- **Type:** This drop down menu allows you to choose between linear and log scale spectra. The *Reverse* check box allows you to swap the direction of the spectrum. If a log scale is selected as the type, the *Exaggeration* settings become available.

  - Exaggeration: This text box allows you to specify how strongly the spectrum is exaggerated.
  - Left/Right: These radio buttons allow you to choose the direction of the exaggeration.

- **Normalized colour range:** These two text boxes allow you to specify the portion of the colour range that is displayed across the chosen range of values for the spectrum component. The default values of 0 and 1 display the entire colour range, and other values allow you to choose where in the colour range the spectrum component begins and ends. For example, 0.5 and 1 would display only the upper half of the colour range; 0 and 0.5 would display only the lower half; 0.25 and 0.75 would display the middle section of the colour range.
- **Number of bands:** This text box is enabled when the *contour bands* colour type is chosen. Enter a value to specify the number of contour bands.
- **Step value:** This text box is enabled when the *step* colour type is chosen. Enter a value somewhere between the values specified in *Spectrum range* to specify where the step occurs in the spectrum.

4 Graphical settings

General description of graphical settings

Graphical settings are the building blocks used to create any visualization displayed in the CMGUI graphics window. They are created, edited, re-ordered and deleted from within the scene editor, or via the command line. All graphical settings have the following settings in common:

- Name: This allows you to set the name of the graphical setting.
- Coordinate field: This setting has a check box, which activates a drop-down menu showing a list of fields that can be selected as the coordinate field for the selected graphical setting.
- Select mode: This drop-down menu allows you to select different selection behaviors for the graphical setting:
- select_on - The default setting; graphical settings are able to be selected and selected items are highlighted by rendering in the *default_selected* material.
- no_select - No selection or highlighting of the graphical setting.
- draw_selected - only selected items are drawn.
- draw_unselected - only unselected items are drawn.
- Material: This drop-down menu allows you to select which material should be used to render the graphical setting. Materials are defined and edited in the material editor window.
- Data: This setting has a check box which activates a drop-down menu. This menu allows you to select which field will be mapped on to the graphical setting, allowing you to colour the graphical setting according to the values of some field for example. The check box also activates the *spectrum* drop-down menu.
- Spectrum: This drop-down menu is used to select which spectrum is to be used to colour the graphical element according to the field selected in the *data* setting. Spectra are edited in the spectrum editor window.
- Selected material: This drop-down menu allows you to set which material will be used to render parts of the graphical setting which are selected.

4.1 The eight types of graphical settings

node_points:

Node points are used to visualize nodes. You can use glyphs to represent node points. There are a range of built-in glyphs in CMGUI, and it is possible to create custom glyphs as well. *Node points* graphical settings have the following settings in addition to the common ones listed above:

- Glyph: this drop-down menu allows you to choose the glyph that will be rendered at the node points.
- Centre: this box allows you to offset the origin of the glyph in order to alter its placement at the node point.
- Base glyph size: This box allows you to enter a size (x,y,z) for the glyph in the same scale as the coordinate system for the region.
- Orientation/Scale: This check box enables a drop-down menu that allows selection of the field that the glyphs will be oriented and scaled according to.
- Scale factors: This box allows you to enter how each dimension scales according to the orientation/scale field value.
- Variable scale: This check box enables a drop-down menu that allows selection of the field that acts as the variable scale field. For more information on this and other of the *node points* options, see the document on working with glyphs.

data_points:

Data points are used to visualize data points. Like node points, they can be represented using glyphs. They have the same settings as *node points*.

lines:

Lines are used to visualize 1D elements, or the edges of 2D or 3D elements. They are simple, unshaded lines that have a fixed, specified width. They have the following specific settings:

- Exterior: Checking this box will automatically only render lines on exterior surfaces of a mesh.
- Face: This check box enables a drop-down menu that allows you to choose which faces are drawn according to xi values. You can select xi=0 or 1 for each of the three xi-directions.
- Width: this allows you to specify the width of the lines in pixels. This is a constant width that does not scale according to the zoom level.

cylinders:

Cylinders are used to visualize the same things as lines. They are shaded cylinders of a specified radius, with their number of sides specified by the *Circle discretization* setting in the scene editor *General settings*. They have the following specific settings:

- Exterior: Checking this box will automatically only render cylinders on exterior surfaces of a mesh.
- Face: This check box enables a drop-down menu that allows you to choose which faces are drawn according to xi values. You can select xi=0 or 1 for each of the three xi-axes.
- Constant radius: This allows you to set the radius of the cylinders, in the units of the coordinate system.
- Scalar radius: This check box will activate a drop-down menu allowing you to select which field will be used to scale the radius of the cylinders. It will also activate a text box in which you can enter the scale factor, or how the scale field will scale the radius.
- Texture coordinates: This check box activates a drop-down menu that allows you to select which field will be used to position any textures applied by the material setting.

surfaces:

Surfaces are used to visualize 2D elements or the faces of 3D elements. They are shaded surfaces of zero thickness that are automatically shaped according to the nodes defining the element they represent. Their level of detail is specified by the *Element discretization* setting in the scene editor *General settings*. They have the following specific settings:

- Exterior: This check box will automatically only render surfaces on exterior surfaces of a mesh.
- Face: This check box enables a drop-down menu that allows you to choose which faces are drawn according to xi values. You can select xi=0 or 1 for each of the three xi-axes.

- Render type: This drop down menu allows you to select shaded (default) or wireframe rendering of surfaces. Wireframe rendering renders the surfaces as grids of shaded lines, with the grid detail determined by the *element discretization* setting in the *General settings*.
- Texture coordinates: This check box activates a drop-down menu that allows you to select which field will be used to position any textures applied by the material setting.

iso_surfaces:

Iso-surfaces are used to represent a surface that connects all points that share some common value. For example, in example a7 an iso-surface is used to represent a surface at which every point has a temperature of 100 degrees C. They have the following specific settings:

- Use element type: This drop down menu allows you to select which type of element will have surfaces rendered on it. Type *use_elements* is the default. The types *use_faces* and *use_lines* will render element points only on those components of elements. If faces or lines are chosen, the following options are activated:
- Exterior: This check box will automatically only render iso-surfaces on exterior surfaces of a mesh.
- Face: This check box enables a drop-down menu that allows you to choose on which faces iso-surfaces are drawn, according to xi values. You can select xi=0 or 1 for each of the three xi-axes.

It is worth noting that if you select *use_surfaces* then the equivalent of iso-surfaces becomes iso-lines. If you select *use_lines* then you will not get any visual representation.

- Iso-scalar: This drop down menu allows you to select the field that the iso-surface will be rendered according to the values of.
- Iso-values: This settings box contains the following settings:
- List: This radio button activates a text box that allows you to enter a value at which to draw the iso-surface.
- Sequence: This radio button activates three text boxes that allow you to enter a sequence of evenly spaced values to draw iso-surfaces at. The *Number* box allows you to enter the number of iso-surfaces you want. The *First* and *Last* boxes allow you to enter the starting and ending values of the iso-surfaces. The sequence will automatically space the number of surfaces between these two values.
- Render type: This drop down menu allows you to select shaded (default) or wireframe rendering of surfaces. Wireframe rendering renders the surfaces as grids of shaded lines, with the grid detail determined by the *element discretization* setting in the *General settings*.
- Texture coordinates: This check box activates a drop-down menu that allows you to select which field will be used to position any textures applied by the material setting.

element_points:

Element points are used to visualize the discretized points within an element. Elements may be 1, 2 or 3 dimensional, in which case the element points are spaced along the line, across the surface, or throughout the volume according to the *Element discretization* setting in the scene editor *General settings*. They have the following specific settings:

- Use element type: This drop down menu allows you to select which type of element will have element points rendered on/in it. Type *use_elements* is the default, and renders element points throughout 3D elements. The types *use_faces* and *use_lines* will render element points only on those components of elements. If faces or lines are chosen, the following options are activated:
- Exterior: This check box will automatically only render element points on exterior surfaces of a mesh.
- Face: This check box enables a drop-down menu that allows you to choose on which faces element points are drawn according to xi values. You can select xi=0 or 1 for each of the three xi-axes.
- Xi discretization mode: this drop down menu allows you to select the method by which element points are distributed across the element.

streamlines:

Streamlines are a special graphical setting for visualizing *vector* fields - for example, a fluid flow solution. They can be used to visualize 3, 6 or 9 component vector fields within a 3 dimensional element. In example *ao*, streamlines are used to show the fibre and sheet directions in the heart. Streamlines will align along their length according to the first vector of a vector field, and across their "width" (eg the width of the *ribbon* or *rectangle* streamline types) to the second vector. For single vector (3-component) vector fields, the width of the streamlines will align to the curl of the vector.

Note that streamlines can be quite expensive to compute; changes to streamline settings in the scene editor can take several seconds to appear in the 3D window, especially for complex scenes.
Streamlines have the following specific settings:

- Streamline type: This drop-down box allows you to select the shape of the streamlines; that is, the shape outline that is extruded along the length of the streamline. *Line* and *Cylinder* can be used to visualize streamlines without showing orientation (curl). *Ellipse*, *rectangle* and *ribbon* types will enable visualization of the direction of the vector orthogonal to the streamline direction.
- Length: Enter a value into this box to set the length of the streamline/s.
- Width: Enter a value into this box to set the width of the streamline/s.
- Stream vector: This drop-down box allows you to select the vector that is being visualized by the streamlines.
- Reverse: Checking this box reverses the streamline.
- Seed element: Checking this box allows you to select the single element number from which the streamline will be seeded.
- Xi: Entering three comma-separated values (between 0 and 1) allows you to set the xi location within elements from which streamlines will be seeded.

5 CMGUI fields

Fields in CMGUI: finite element models

CMGUI models are organized into regions, which may contain child regions, so that models can be organized into hierarchies. The representation of the model within each region is given by *fields*. Much of the power of CMGUI is in its generalized representation and manipulation of fields. Mathematically, a field is a function returning values at locations within its domain. Most commonly the values are real numbers, though they may be scalars or multi-component (vectors, tensors etc.).

Consider an object such as a solid cube over which a value like the temperature varies. We would represent this in CMGUI as a field "temperature" which is a function of location in the cube. A cube is a very simple geometry and it is easy to come up with a simple method to specify any location within it; for example the global x, y, z. But real geometries are seldom simple so we follow the finite element method and divide them up into simple shapes such as cubes, wedges, tetrahedra, etc. for 3D geometries; squares, triangles etc. for 2D geometries; lines for 1D geometries, and other shapes in other dimensions.

These finite elements have a local coordinate system of limited span, which is called an element chart. In CMISS we commonly use the Greek character xi (ξ) with subscript to denote each independent coordinate within an element, hence we will commonly refer to the chart of a given element as its *xi space*. A 3-D cube element in CMISS/CMGUI has a range of 0 to 1 in each xi direction, as shown in figure 6A.

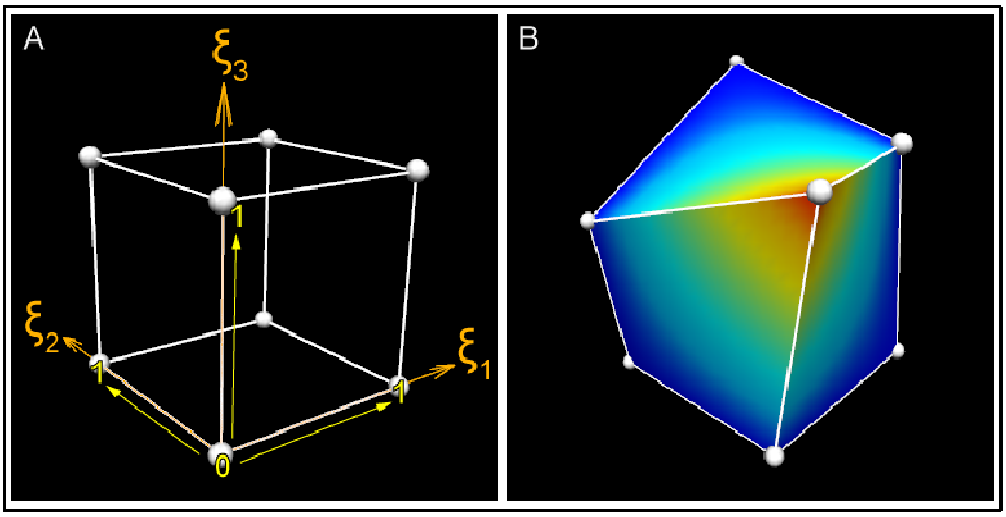


Figure 6: Coordinates and xi space of a 3D element. A) This shows the unit cube xi space, where each dimension of the element ranges from 0-1. It is easy to imagine that the coordinates of this cube could also be 0-1 in the x, y, and z axes. B) The cube element in this picture has been distorted, such that its coordinates are no longer 0-1 in the x, y, and z axes. Despite this, the element's xi values are still 0-1 in each xi direction. This cube has a "temperature" field that is rendered as a rainbow spectrum.

We must define a coordinate field over the domain in order to give it its true position in 3-dimensional space. This applies even to the simple cube model: it can be treated as a unit cube in xi space, but the coordinate field allows its real position to be transformed such that it is not aligned with the global coordinate system, and in fact can be generally distorted as shown in the figure 6B, above. About the only thing special about a coordinate field is that - provided it is of the same dimension as the element xi space - it is usually bijective with it - this means if you have one you can find the other, eventually. (I say usually because the relation cannot generally be enforced: it is possible for coordinates of parts of the mesh to penetrate other parts of the mesh, including within a single element where the Jacobian is zero or negative.) A set of finite elements which make up a domain is called a mesh.

In CMGUI data structures we usually also define the faces of elements as separate elements of lower dimension, and share faces between adjacent elements. The faces of 3D "top-level" elements are called face elements, the faces of 2D elements are called line elements. The top-level elements, whether 3D or 2D (or *n*D) have unique integer element identifiers in their region. Face and line elements also have unique identifiers within their own type; in other words you can have element 1, face 1 and line 1 and they are not the same element.

The zero dimensional counterpart to elements and faces are called nodes. There is one set of nodes per region, again with their own unique integer identifiers. Nodes can be considered as a set of points at which fields are defined. When we define field functions over elements (finite element fields) we most commonly store explicit values and derivatives of the field at nodes and interpolate them across the element. To define such fields each element maintains a list of the nodes which contribute to fields in that element, called the local node list. The number and arrangement of nodes depends very much on the basis function used to compute the field value. Linear Lagrange and simplex basis functions are a simple, common case where nodes contain values of the field at the corners of elements, and these are linearly interpolated in the xi space between. Figure 7 shows the arrangements of faces, lines and nodes (for linear basis) for 3D and 2D elements.

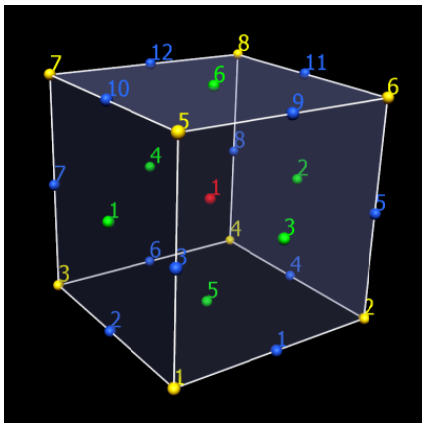


Figure 7: How nodes, lines and faces make up a mesh The simple cube mesh again; nodes (corners, 1-8), elements (whole cube, 1), faces (flat square faces,1-6), and lines (edges, 1-12) are numbered. A single cube element requires 8 nodes, 12 lines, and 6 faces.

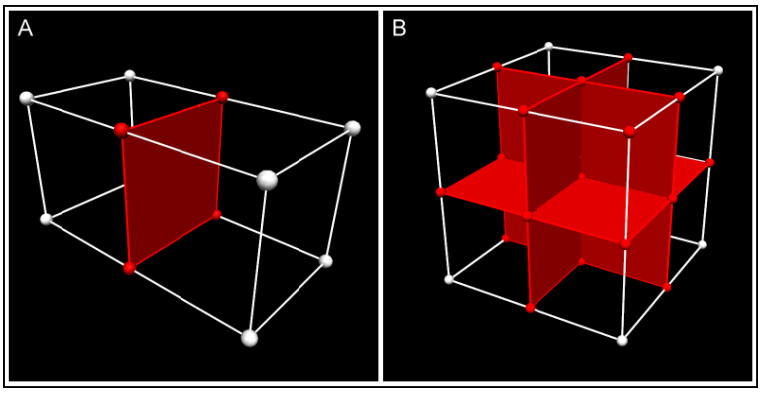


Figure 8: Node, face and line sharing between connected elements In more complex meshes, connected elements share nodes, lines and faces (shared nodes lines and faces are shown in red). In panel A, this two-cube mesh has only 12 nodes; 4 of the nodes are shared by both elements. In panel B, an eight-cube mesh is made up of only 27 nodes - many of the nodes are shared by more than one of the elements. The central node in this example is shared by all 8 elements. Field values are continuous across these shared parts.

Getting back to the original statement of what a field is, we can generally state that a domain is a set of 0..N-dimensional manifolds, i.e. there can be any number of manifolds of any dimension. In CMGUI finite element meshes, nodes supply the point manifolds and elements supply all the higher dimensional manifolds. There is no requirement for the domain to be connected. CMGUI fields are not limited to returning real numbers; there are fields that can return integers, strings, element-xi locations and other values.

Computed fields

We have now introduced the main building blocks of finite element fields which are just one type of field representation in CMGUI. In CMGUI we offer a large number of other field types which do not work directly with finite element meshes, but may be based on finite element fields. Most of these are mathematical operators which act on one or more source fields to produce a new field result.

A simple example is the *add* field which adds two other fields together to return a new field. The add field has two source fields as arguments. If you made field C which added finite element field A to finite element field B, the resulting field is defined over the intersection of the domains of field A and field B.

Other types of field

- **Arithmetic and (non-trigonometric) transcendental functions:** Add, subtract, multiply, divide, sum_components, log, power, square root, exp.
- **Trigonometric functions:** sin, cos, tan, asin, acos, atan, atan2.
- **Derivative functions:** Derivative, divergence, gradient, curl.
- **Vector functions:** Dot_product.
- **Matrix functions:** Matrix_multiply, matrix_invert, transpose, projection, eigenvalues, eigenvectors, quaternion_to_matrix, matrix_to_quaternion.
- **Logical functions:** Less_than, greater_than.
- **Conditional functions:** If.
- **Constant fields:** Constant fields have a value which is independent of location within the domain. You may also have a constant field that is constant (non-varying) across chosen dimension/s but varies across other dimension/s.
- **Composite field:** Makes a new field built from other fields and field components in any order.
- **Image-based fields:** These can be used for texture-mapping and image processing. Image processing fields using ITK filters.
- **Many more.**

Comment [RB1]: Trig functions are actually transcendental too.

These types of fields can be created via `gfx define field` commands or through the API. Fields are a modular part of the CMGUI application. If a new function is required, it can be added as a field. To get a list of the computed fields available in CMGUI, enter `gfx define field ??` in the command line.

Fields and visualization

When creating visualizations, you need to choose which field controls which part of a graphics object. Coordinates in one, two or three dimensions can be used to create spatial representations. Texture coordinate fields can be used to position textures. Orientation or data fields can be used to position glyphs or colour objects such as surfaces. CMGUI allows an enormous amount of flexibility in how fields can be visualized.

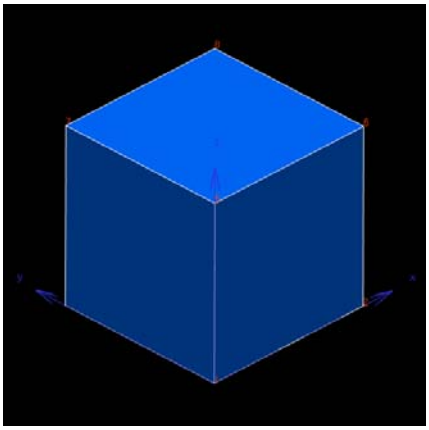
6 Examples

The CMGUI examples exist as *comfiles* (CMGUI script files) and accompanying data files (.exnode, .exelem, graphics files, etc) which can be downloaded as compressed archives or individual files from the CMGUI website. In this handout we provide a small selection of these examples in order to show some of the basic capabilities of CMGUI.

All of the examples below, as well as many others, can be found at:
http://cmiss.bioeng.auckland.ac.nz/development/examples/a/index_thumbs.html

6.1 Example a1: graphical element groups: viewing a cube

This example shows the first steps to displaying a model with graphical element groups. Also demonstrates automatic updating of the graphics with node changes.



Example_a1: Graphical element groups: viewing a cube

In CMGUI, every element group has a graphical rendition - a list of attributes such as lines, surfaces etc. that describes how it is to look. Combined, this 'graphical element group' monitors changes in its nodes and elements and automatically updates the graphics to reflect the changes. By default, every group is drawn using lines, ie. as a wireframe mesh, and later examples will show how other graphical representations can easily be created. In this example, a one-element cube is read in and displayed. It also shows how the graphics are updated in response to node changes, and shows some other useful graphics commands on the way.

This example .com file will be loaded in a comfile window with *'All'*, *'Selected'* and *'Close'* buttons at the bottom. Lines beginning with a hash (#) are comments which are ignored (in this handout, these commented lines have been reformatted into normal text). All other lines are commands, and all graphics commands begin with `gfx`. Note that command tokens can be abbreviated, eg. `cre` instead of `create` - as long as the short form is not also short for any other token. Do not abbreviate commands too much otherwise they are not readable to other users or by scripts used to update comfiles when new commands with similar names are added.

You should not run all commands at once since it is important to understand each step. Instead, perform the commands one at a time (in the order they appear) by double-clicking them. Alternatively, select a group of commands in the comfile window with the mouse and press the *'Selected'* button. Between the commands are comments describing them, or instructions.

Read in the nodes and elements representing the cube:

```
gfx read region $example/cube.fml;
```

Open a 3-D graphics window (named 1). You can also do this by selecting '3-D window' from the Tools menu on the Command Window.

```
gfx create window 1
```

The cube is automatically shown at the centre of the window.
Create blue axes of 1.2 units length and draw them in the scene:

```
gfx create material blue ambient 0.2 0.2 0.9 diffuse 0.2 0.2 0.9
gfx create axes material blue length 1.2
gfx draw axes
```

Now spend some time getting used to the mouse actions that change your view of the object. Press the left mouse button in the graphics window and drag it around to 'tumble' the object in 3-D. The middle and right mouse buttons can be used in the same way to translate and zoom the object. Try these, with and without the *'Perspective'* button checked. You can always press the *'View_all'* button to return to a view of the whole object, or alternatively type:

```
gfx modify window 1 image view_all
```

To demonstrate moving nodes, first show node numbers. This involves adding node_points labelled with their cmiss_number to graphical element *'cube'*. We will also give them their own colour:

```
gfx create material orange ambient 1 0.25 0 diffuse 1 0.25 0
gfx modify g_element cube node_points material orange label cmiss_number
```

Now bring up the node viewer so we can change view and change nodal coordinates:

```
gfx create node_viewer
```

To change the position of a node, type the node number in at the top of the dialog, press ENTER, choose the '*coordinates'* field, change the x, y, or z values, then click *'Apply'*. Try changing the x coordinate of node 2 to 0.5. The cube will now be distorted in the graphics window.
Now add blue surfaces to the distorted cube:

```
gfx create material bluey ambient 0 0.25 0.5 diffuse 0 0.4 1 specular 0.5 0.5 0.5 shininess
   0.3;
gfx modify g_element cube surfaces mat bluey
```

You can also change the position of nodes by reading in nodes from a file. Since we have distorted the cube from its original position, just read in the original node file to demonstrate this effect:

```
gfx read region $example/cube.fml;
```

If you want to continue entering commands, the following sets the command prompt to `gfx` , saving you from typing it all the time. Most commands allow this capability:

```
gfx
```

TIPS

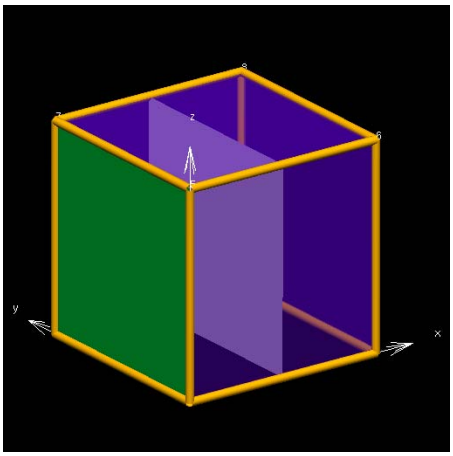
By studying the commands used in this example you will hopefully start to see some pattern to how they work. While many interactive editors are available to control parts of the program, text commands are also available to control most features so that command files such as this may be written. Many features are only available through text commands at this time.

There is a very simple way to find out what commands you can use and what parameters they can take: using the `?` and `??` tokens. If you enter `gfx ?` on the command line of the command window, CMGUI will list all the keywords you can possibly enter after `gfx`. Enter `gfx create ?` for a further list of keywords. Now you can see where the 'gfx create axes' command used above fits in. Now enter `gfx create axes ?`. The possible options are now placed in angled brackets indicating that you can supply any of the parameters. All of the options have default values, given in square brackets. You can put 'length 1.2' after the command stem, which is equivalent to `lengths 1.2*1.2*1.2`. The `material` option allows the colour of the axes to be controlled; you can list available materials using the command `gfx list material`.

The `??` help mode works much like `?` except that it is recursive - it will list all the keywords that may follow the current one, and all that may follow them, and so on. Type `gfx ??` to see all the graphics commands and parameters available - this will take a few seconds.

6.2 Example_a2: Scene editor, lighting: decorating a cube

In this example you will use materials, lighting and different graphic types to decorate a cube.



Provided you have placed your elements (and nodes) into appropriate groups, giving them a graphical rendition is as simple as a few clicks in the 'scene editor'. This example demonstrates how to use this editor to make a cube look really stylish. It shows off several features such as materials, lighting and transparency, and shows how to move nodes visually with the mouse.

Be sure to run the commands in this example in the order they are presented, and not all at once. Comments and instructions for things you can do are given between these commands. The following commands create a few materials in addition to the default for use later. Material *'kermit'* is green and shiny, *'trans_purple'* is a semi-transparent purple colour, while *'gold'* looks somewhat like the precious metal.

```
gfx create material kermit ambient 0 0.7 0.2 diffuse 0 0.7 0.2 specular 1 1 1 shininess 0.5;
gfx create material trans_purple ambient 0.4 0 0.9 diffuse 0.4 0 0.9 alpha 0.5;
gfx create material gold ambient 1 0.7 0 diffuse 1 0.7 0 specular 1 1 0.8 shininess 0.8;
```

Read in the cube and draw axes.

```
gfx create axes length 1.2
gfx draw axes
gfx read nodes example cube.exnode
gfx read elements example cube.exelem
```

Note that you do not need a graphics window open to set axes or create graphics - they exist in the default 'scene'. A graphics window is one of the tools available to view the scene. Other ways include exporting to VRML or Alias|Wavefront formats. Now open the graphics window and set the view you want by dragging in it with the mouse.

```
gfx create window 1
```

Now open the scene editor, either by selecting it from the 'Graphics' menu, or by typing:

```
gfx edit scene
```

Spend some time looking at the parts that make up the editor. At the very top you can set which object in the scene you are looking editing. Every element group may have a different rendition in every scene. Usually you will only use the default scene. To the left of each item, for example 'cube', is a toggle button for changing its visibility. Click it off, then on again to see the whole cube disappear and reappear in the graphics window.

Visibility can also be controlled by commands:

```
gfx set visibility cube off
gfx set visibility cube on
```

Below the list of objects in the scene is some information and controls for manipulating them. There are up and down arrows for changing the order in which objects are drawn. The 'Auto' button controls whether changes you make in the areas below are applied immediately, and whether changes made externally are automatically reflected in the editor. Normally you should leave this on, but if your graphics are slow to create and you wish to make several changes before they are rebuilt, disable this function. You will then need to click on the *'Apply'* and *'Revert'* to bring the rest of the program in line with your changes, or vice-versa.

If you select an object such as *'cube'* that represents part of your model, the remainder of the dialog will be filled with the graphical element editor. This editor contains a list of graphics making up the rendition of group cube. Initially, it shows just lines, and if selected, the parameters that make up the lines are available for editing below the list of graphics. To the left of each listed graphic is a visibility control. Click on this control to hide the lines so we can show more interesting graphics instead. The change in visibility takes place instantly because *"Auto Apply"* is on.
The lines can also be hidden with the following text command:

```
gfx modify g_element cube lines invisible
```

Along one line of the editor you should see four buttons to add, delete, move up and move down an item. To the left of the *'Add'* button is a menu for selecting which type of graphic you wish to add. Choose *'cylinders'* then click *'Add'*. Change the Material to gold, and enter a value of 0.02 in the *'constant radius'* box. If you look really closely at the cylinders you will see that they are in fact 6-sided polygons. You can control the number of segments around the cylinders by clicking on the *"+"* button to expand the *'General settings'* and changing *'Circle discretization'* to, say, 12.
The following text commands do the same as the above interactions (note that setting the discretization first means the graphics are not rebuilt twice):

```
gfx modify g_element cube general circle_discretization 12
gfx modify g_element cube cylinders constant_radius 0.02 material gold
```

You may have to enlarge the window, or hide the general settings by clicking on the '-' button, to see the entire editor.

Now add semi-transparent surfaces by selecting *'surfaces'*, *'Add'*, changing the material to *trans_purple*. This produces surface out of all the 2-D elements in the group. Element groups may contain 3-D, 2-D and 1-D elements, and by reference, [0-D] nodes. While most graphic types work on just one dimensionality, some allow you to choose the dimension of elements to be used.

The following text command adds the semitransparent surfaces:

```
gfx modify g_element cube surfaces material trans_purple render_shaded
```

Select *'surfaces'* then *'Add'* again, change the material to *kermit*, select *'Face'*, then apply. Look at the model. This should have made the surface xi1=0 green, but it had no effect! The reason is that the purple surfaces were drawn first, and despite being translucent, they hide the other surface. The solution is to draw the face before the rest of the surfaces by moving it up the list with the 'Up' button. To get the best transparency effects draw objects from the back (or inside) to the front (or outside). (Note: Another [imperfect] solution that does not require reordering of the graphics is to use the *'slow_transparency'* option with the `gfx modify window 1 set` command).

The following line will add the green face for rendering in the right order:

```
gfx modify g_element cube surfaces face xi1_0 material kermit render_shaded position 3
```

The geometry of the element is described by the field "coordinates", which has three components: x, y and z. Now we want to draw the surface inside the cube at which coordinate "x" is a constant value, say 0.5. First we have to make the x component of the "coordinates" field look like a field itself:

```
gfx define field coordinates.x component coordinates.x
```

In the graphical element editor select *'iso_surfaces'*, *'Add'*, and complete the line `coordinates.x = 0.5`. You will have to move it up before the purple surfaces in the list before it is visible. In later examples you will see much more interesting fields plotted with iso-surfaces. Again the text command equivalent is:

```
gfx modify g_element cube iso_surfaces iso_scalar coordinates.x iso_value 0.5 use_elements
   material default render_shaded position 4
```

Now add *'node_points'*, with "cmiss_number" as the label field. You can now see the node numbers, although they will be partly obscured by the cylinders. There is also a small dot beside the number marking the exact nodal position:

```
gfx modify g_element cube node_points glyph point label cmiss_number select_on material
   default selected_material default_selected
```

Choose the node tool icon in the graphics window. Click on the nodes to select them - they will then be drawn in the "selected material" chosen in the graphical element editor. Dragging across several nodes displays a rubber band for group selection. Click a second time on the node tool icon to bring up a dialog for controlling its behaviour. Activate the *'edit'* control. Now the node tool is able to move nodes in the plane of the window. Click on a node in the graphics window and drag it to a new position. Shift-click on other nodes to add or remove them from the current selection. Group selection with and without the shift-key also enables several nodes to be selected for editing.

Activate edit capability in the node tool with the command:

```
gfx node_tool edit
```

Remember you can transform the view by returning to *'Transform'* mode, or hold down the *'Ctrl'* key to temporarily transform the view with the mouse without leaving the control of the node tool.

Move the nodes on the green surface to make it quite curved. Transform it to see the light glistening off its shiny surface. You will notice that this "specular highlight" is rather cheesy, giving away the polygonal nature of the surface. Display the general settings on the graphical element editor. The *'element discretization'* displays the number of segments used to draw curves in the xi1*xi2*xi3 directions. The default of 4 is adequate in many cases, although 1 is fine for the cube - until it is distorted. Enter '16' in this box to discretizes lines into 16 segments, surfaces into 16*16 facets and volumes - used to calculate iso_surfaces - into 16*16*16 cells. Consequently, it takes a bit longer to produce the graphics. You may briefly see a busy cursor while this takes place. (Remember this is only a one element model... be very careful with big discretizations on large models; it may take several minutes to do what you ask for!).

Adjust the element discretization up with:

```
gfx modify g_element cube general element_discretization "16*16*16"
```

Now rotate the model to see nice, round specular highlights. Use higher discretization values for curved surfaces, when accurate lighting is required and when producing output for presentation.

If you have produced a nice rendition with the graphical element edito, you will be please to know that there is an easy way to produce the text commands for reproducing it - for putting in your own .com file. To get the commands for rebuilding the cube as you see it, enter:

```
gfx list g_element cube commands
```

Select and paste the commands into your .com file. (Had you omitted the keyword *'commands'* in the above, a different list is produced, more useful for editing the group with text commands.) Note that the listed commands are more exhaustive than those shown in this command file. Many of the settings have reasonable defaults that you do not have to enter every time, thus shortening the commands.

You have now used lines, surfaces, cylinders, iso-surfaces and node_points. You can't create fibres without having a fibre field in your elements, while volumes require a volume texture. These and other graphics are left for later examples.

As a final exercise, we will take a look at lighting your object. Initially there is one light called *'default'* which is placed in each window. It works like a head lamp in that it stays in the same position relative to the viewer. Have a look at the parameters of this light by entering:

```
gfx list light default
```

The basic coordinate system of the graphics window is x to the right, y up, and z out of the screen. From the parameters you will see that the default light is mostly in to the screen and partly down. Now change it to point straight into the screen.

```
gfx mod light default dir 0 0 -1
```

Change it to a spot light. Note that you'll have to set a sensible position for the spot light if it is pointing at some other angle to the object - otherwise it may not be shining on it!

```
gfx mod light default spot
```

Change the light back to an infinite source (also called a directional light) and make it point straight down on the object.

```
gfx mod light default infinite dir 0 -1 0
```

Now remove the light from the window so that the object is only lit by ambient light (which is set by the light model - see gfx modify lmodel ??).

```
gfx mod win 1 image remove_light default
```

Add the default light to the default scene.

```
gfx mod scene default add_light default
```

The light now maintains its position and orientation relative to the scene, not the viewer. Now restore lighting to how it was originally.

```
gfx mod scene default remove_light default
gfx mod win 1 image add_light default
gfx mod light default dir 0 -0.5 -1
```

Set the command prompt to gfx.

```
gfx
```

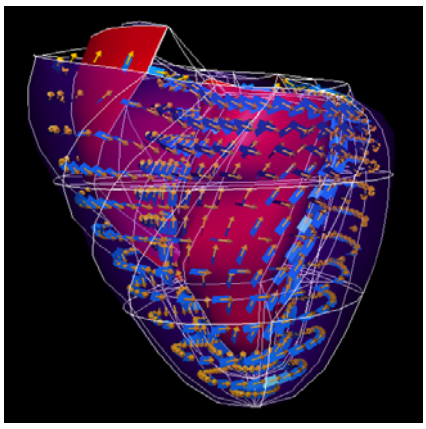
TIPS:

Feel free to experiment with the features of CMGUI, esp. at the end of this and other examples. Switch materials, draw different surfaces, show and hide graphics. You have already seen several graphical materials with interesting properties. Start up the 'graphical material editor' from the 'Tools' menu, create or modify existing materials and apply the changes to see how they affect the result.

You can create extra lights varying in position, direction, colour and type using the `gfx create light` command, and add them to windows and scenes. Current OpenGL implementations limit the total number of lights in an image to 8, but this is ample for most users. Experimenting with different materials, lights and light models will help you make a more impressive image - or make a garish mess.

6.3 Example_a3: Prolate spheroidal coordinates, fibres: viewing the heart

This example demonstrates the structure of the heart model, including fibre architecture. A possible outcome of this example is shown below with gold cylinders aligned in the fibre direction and blue "sheets" indicating the sheet orientation.



The heart model uses prolate spheroidal coordinates to describe its complex, rounded shape with minimal degrees of freedom. The mesh still requires many elements, and the heart wall is two elements thick in places. In addition, fibre fields describe the orientation of muscle fibres and sheets over the heart. This example creates graphics to help visualize the structure of the heart, demonstrates prolate spheroidal coordinates, and shows how to view a fibre field.

Be sure to run the commands in this example in the order they are presented, and not all at once. Comments and instructions for things you can do are given between these commands.

The exnode and exelem files used in this example were generated from example 141

Create a few materials in addition to the default for use later:

```
gfx create material heart ambient 0.3 0 0.3 diffuse 1 0 0 specular 0.5 0.5 0.5 shininess 0.5;
gfx cre mat trans_purple ambient 0.4 0 0.9 diffuse 0.4 0 0.9 alpha 0.3
gfx cre mat bluey ambient 0 0.2 0.4 diffuse 0 0.5 1 specular 0.5 0.5 0.5 shininess 0.8
gfx cre mat gold ambient 1 0.7 0 diffuse 1 0.7 0 specular 1 1 0.8 shininess 0.8
gfx cre mat axes ambient 0.5 0.5 0.5 diffuse 0.5 0.5 0.5
```

Read in the heart and draw axes:

```
gfx read nodes example heart.exnode
gfx read elements example heart.exelem
gfx create axes material axes length 1.5
gfx draw axes
```

Open the graphics window and reorient the heart up the right way (since -x is "up" in the heart model). Also turn on perspective:

```
gfx cre win 1
gfx mod win 1 image rotate 0 1 0 -90
gfx mod win 1 view perspective
```

Draw the endo-cardial surfaces in the heart colour. More exactly, all exterior surfaces of the model for which coordinate xi3 = 0:

```
gfx mod g_e heart surfaces exterior face xi3_0 mat heart
```

Now we'll spend some time looking at iso-surfaces of the prolate spheroidal coordinates. Firstly, define a field for each of the components of the coordinates; lambda, mu, theta:

```
gfx define field coordinates.lambda component coordinates.lambda
gfx define field coordinates.mu component coordinates.mu
gfx define field coordinates.theta component coordinates.theta
```

Next open the scene editor.

```
gfx edit scene
```

Set *'Auto'* to off, so that *'Apply'* will be needed to see the iso-surfaces. The group *'heart'* on scene default should be selected, and it will have the default lines and the endo-cardial surfaces added earlier. Select *'iso_surfaces'*, *'Add'* and make sure the iso-scalar *'coordinates.lambda'* is to equal to 0.5 before clicking *'Apply'*. Coordinate lambda increases in curves away from the central axis of the heart, (-)x. Change the iso value to 0.75 and apply the changes. Next change the iso-component to mu, which increase up the heart. Set mu to 1.5708 (=PI/2) to see a nice slice through the heart. Finally, try the theta component which varies from 0 to 2*PI around the axis of the heart.

Try having several different iso-surfaces visible simultaneously, and with different materials. Change the element discretization (in the general settings) to 8*8*4 and apply this. It will take a few seconds to compute the graphics at this higher quality rendition, but the surfaces and iso-surfaces should look somewhat nicer. Note that due to the shape of the mesh and the elements used, it is fine to use less detail to draw graphics in the xi3 (radial/lambda) direction.

Now add semi-transparent purple surfaces to the exterior of the heart to see its outer shape and interior simultaneously:

```
gfx mod g_e heart surfaces exterior face xi3_1 mat trans_purple
```

(If you haven't enabled the "Auto Apply/Revert" feature of the scene editor, you'll have to click on *'Revert'* to reload the rendition from the scene into the editor.)

For the next part of this example, select all the iso-surfaces one-by-one and click on the *'Del'* key to delete them. Also make the outside surfaces invisible, then click *'Apply'*.

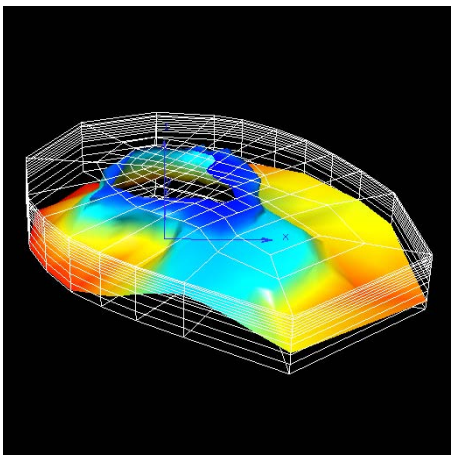
Add *element_points* to the rendition with the discretization set at 4*4*1 and the material gold. Choose the *'cylinder'* glyph, make the base glyph size 2*0.5 (becomes 2*0.5*0.5 when you press enter), choose field *'fibres'* as the orientation/scale field and set the *scale_factors* (multiplying the magnitude of the orientation/scale field vectors) to 0, since the fibre axes are unit vectors specifying direction only. The heart is now covered in small cylinders indicating the direction of the fibres. Following the discretization values, there are 4*4*1 cylinders across each element in its xi1*xi2*xi3 directions, respectively. Note that the *'Glyph_size'* paramaters display the size as length*width*height - the width is in the direction of the fibre sheet.

Cylinders are just one way of viewing the fibres; they don't show the plane of the sheet, nor do they indicate which direction they are pointing in along their axis - which may be useful for checking your fibre field. You can choose several other "Glyphs" besides cylinders to display over the fibre field. Apply the *'arrow_solid'* glyph and take a close look at it. (The centre of the cylinder is half-way along its length, whereas the arrow is centred at its tail. The *'Centre'* can be changed on the editor. All glyphs are designed to be of unit dimension - 1 unit long, 1 unit diameter at the widest point etc. - so sensible centre values should be of a similar magnitude.)

Go back to using cylinders, add another set of fibres using *'sheets'* and material *'bluey'*. Enter `1.5` in the *'Glyph size'* field. You can now see both the the fibre direction and sheet plane. Experiment with different glyphs, sizes, discretizations and materials to see how you can interpret the fibre field. In general, keep the discretization the same for all fibre graphics.

6.4 Example_a7: Fields, spectrums and iso_surfaces: geothermal field

This example shows the practical use of iso-surfaces and scalar plots for visualizing field information in a geothermal model. It also introduces spectrums and the spectrum editor.



Geothermal models have to keep track of numerous state variables such as temperature, pressure, density, vapour saturation, fluxes and more. CMGUI provides the ability to plot such fields (or field components), coloured by spectrums, over any lines, surfaces or other graphics it can produce. Furthermore, when fields are defined over a volume, iso-surfaces - surfaces of constant data field value - can also be computed and displayed.

This example shows the Wairakei Broadlands geothermal field, modelled by the MULKOM package using finite volumes. The data set has many fields defined over it, only a few of which will be looked at here. Note that this is a large example and may be tedious to view on a slow computer or network.

Be sure to run the commands in this example in the order they are presented, and not all at once. Comments and instructions for things you can do are given between these commands.

Create some materials for use later:

```
gfx create material red ambient 1 0.2 0.2 diffuse 1 0.2 0.2
gfx create material green ambient 0.2 1 0.2 diffuse 0.2 1 0.2
gfx create material blue ambient 0.2 0.2 1 diffuse 0.2 0.2 1
gfx create material trans_brown ambient 0.6 0.3 0.2 diffuse 0.6 0.3 0.2 alpha 0.8
gfx cre mat bluey ambient 0 0.2 0.4 diffuse 0 0.5 1 specular 0.5 0.5 0.5 shininess 0.8
gfx cre mat gold ambient 1 0.7 0 diffuse 1 0.7 0 specular 1 1 0.8 shininess 0.8
```

Read in the nodes and elements for the Wairakei Broadlands field:

```
gfx read node example wai1225.exnode
gfx read elem example wai1225.exelem
```

Open the graphics window:

```
gfx create window 1
```

Draw axes close to the mesh and of appropriate size and colour to be seen. Note that the town of Taupo is at approximately at coordinate (265400,596800,367), so we choose this as the origin for the axes:

```
gfx create axes material blue origin 265400 596800 367 length 5000
gfx draw axes
```

Now adjust the view to see all the model in perspective:

```
gfx mod win 1 image view_all
gfx mod win 1 view perspective
```

Change the lines to be on the exterior only, and reduce discretization:

```
gfx modify g_element wai1225 lines delete
gfx mod g_e wai1225 general element_discretization 1
gfx mod g_e wai1225 lines exterior
```

Now add the surface at which temperature = 100. Colour it gold:

```
gfx mod g_e wai1225 iso_surfaces iso_scalar Temperature.value iso_value 100 material gold
```

Rotate and zoom in on the model to have a good look at it. Note especially the proximity of Taupo to this high temperature zone.

For the remainder of this example we will use the scene editor to vary the fields we look at. We need to first define the fields that we will be visualising, otherwise they are not available as options in the scene editor:

```
gfx define field Temperature.value component Temperature.value
gfx define field Pressure.value component Pressure.value
```

Next we open the scene editor.

```
gfx edit scene
```

First select the *iso_surfaces* that were created earlier and experiment with different *iso_values* from 0 to the maximum temperature of just over 250 degrees. Set it to a value of 150 and change its material to 'bluey'. Now add a second iso_surface, with material *'trans_brown'* and *'Temperature.value=100'*. This new surface must be drawn after the blue surface to achieve the desired effect of seeing the 150 surface through the 100 surface. You may wish to make the lines invisible when looking at this.

Now delete one of the iso-surfaces and ensure the remaining one uses material *'gold'* and is showing the surface of temperature=100. For this iso_surface, click *'Data'* and choose field component *'Pressure.value'*. Now click OK. The surfaces are probably black. What you have done is told the program to ask the "spectrum" called *'default'* to supply the colour of the surface based on the value of the Pressure field (which has only one component). The spectrum currently does not know the range of that field over the graphics object you have drawn; we now have to get it to find the range.
Open the spectrum editor:

```
gfx edit spectrum
```

A spectrum is a mapping from the value of a field to a colour. CMGUI allows a huge degree of flexibility in specifying this mapping and the colours that are applied. You should now be looking at spectrum *'default'* which currently draws a rainbow for field values ranging from value 0 to 1. You can tell the spectrum to automatically update its range to match the data range of all graphics objects using it simply by pressing the *'Autorange'* button on the this editor (the same option is available from the `gfx modify spectrum` commands). Since the editor, like all others works on a local copy of the object concerned, you must click on *'Apply'* to copy the changes to the real object. Do this and look at the result.

You should create a separate spectrum for each field/component you wish to plot. Temperature and pressure values, for example, are vastly different in magnitude and therefore do not belong on the same scale. Select the *'default'* spectrum on the editor, click *'Rename'* and change its name to *'Pressure'*. Its name will be updated in the scene editor as soon as you press ENTER. You should use this spectrum whenever you are plotting this Pressure field (you may have a completely different pressure in a different region of the model, in which case you may choose to give it its own spectrum). If you wish to exaggerate the scale or emphasis certain parts of the range, you may do so by adding extra settings to the spectrum. The following exercise shows how to do this.

On the spectrum editor, click *'Create'* to make a new spectrum and rename it *'Temperature'*. Hide the coloured iso_surfaces using their visibility toggle on the graphical element editor. Add surfaces with the *'exterior'* flag set and plot scalar *'Temperature.value'* with the Temperature spectrum. Autorange the spectrum to see the result. The outside surface will be rainbow coloured, and red where it is hottest.

Suppose we are not so interested in areas where the temperature is below 100 degrees. We can colour this part of the spectrum differently. Click *'Add'* on the Spectrum editor to add more settings to the Temperature spectrum. It will be a copy of the original rainbow, with the same range of approx. 20 to 263. Change its lower limit to `100` and toggle *'Reverse'* to change the direction of the rainbow. Now select the first settings item and change its colour to Green. Make sure it is not reversed either and that its upper limit is 100. You also need to switch off *'Extend below'* and *'Extend above'* on both parts of the spectrum.

Click apply to see the changes on the graphics window. The hottest region is now blue, with "unimportant" regions coloured green.

With this kind of smooth colouring it can be hard to tell the exact temperature at a specific point on the surface. By overlaying the colour with contour bands, this situation is greatly improved. Add another component to the Temperature spectrum of type *'Contour Bands'* and apply it. You can adjust the number and thickness of the bands on the editor. Note that there is a visibility beside each component making up the spectrum. Experiment with turning them on and off and applying the changes.

You may notice that abrupt changes in colour on the spectrum, such as where green changes to red in this example cause some rather bizarre shading effects. Increasing the element discretization of the graphical element narrows the band where this problem occurs, but only ridiculously high discretization values will fix it completely. *'Contour Bands'* and *'Step'* types do not suffer from these problems since they use texture mapping to apply the spectrum. However, since only one texture can be used at a time, you will not get the correct behaviour with these types unless you use just one set of bands OR just one step function.

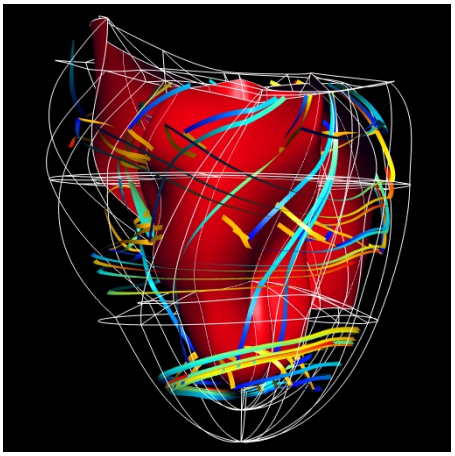
Spectrums combine their colours with those of the material you gave to the graphics object. If that material was semi-transparent, it should remain so. The components making up the spectrum may override the ambient, diffuse, emissive and specular colour components of the material. You can even have the spectrum vary the transparency (alpha) of the object according to a field value. The editor option 'Clear colour before settings' is important here. When plotting a green spectrum, the red and blue components of the original material will be cleared if this button is set, and unchanged if it is not. Red and blue colours apply similarly to their component.

Covering all the capabilities of spectrums is beyond the scope of this one example. Feel free to view other fields defined for this example and experiment with other spectrum options.

Note that you can get CMGUI to list the commands needed to reproduce a particular spectrum for putting in your .com files using, eg: `gfx list spectrum Temperature commands`

6.5 Example_ao: Streamlines : Showing heart fibres

This example streamlines to visualise the fibre and sheet directions in the heart. The image below shows the results of this example, with the coloured ribbons aligned along the fibre axis with the width of the ribbons aligned with the sheets. The colour spectrum indicates the distance along the ribbon: blue (0) at the start of the streamline and red (100) at the end. The gold ribbons use the reordered fibre tensor to be aligned with the sheet axis with the width oriented to the fibre direction.



Create the graphics window:

```
gfx create window 1
gfx modify window 1 view perspective eye_point 9.70453 -289.811 -5.00082 interest_point
   9.70453 6.38585 -5.00082 up_vector -1 0 6.12323e-17 view_angle 18.2876 near_clipping_plane
   2.96197 far_clipping_plane 1058.51 relative_viewport ndc_placement -1 1 2 2
   viewport_coordinates 0 0 1 1
```

Load the exnode and exelem files:

```
set dir example a3
gfx read nodes example heart.exnode
gfx read elements example heart.exelem
```

Define a few materials:

```
gfx create material red ambient 1 0 0 diffuse 1 0 0 emission 0 0 0 specular 0.8 0.8 0.8 alpha
   1 shininess 0.8
gfx create material green ambient 0 0.6 0 diffuse 0 0.6 0 emission 0 0 0 specular 0.8 0.8 0.8
   alpha 1 shininess 0.8
gfx create material bluey ambient 0 0.2 0.4 diffuse 0 0.5 1 emission 0 0 0 specular 0.5 0.5
   0.5 alpha 1 shininess 0.8
gfx create material gold ambient 1 0.7 0 diffuse 1 0.7 0 emission 0 0 0 specular 1 1 0.8
   alpha 1 shininess 0.8
gfx create material purple ambient 1 0 1 diffuse 1 0 1 emission 0 0 0 specular 1 1 1 alpha 1
   shininess 0.74
gfx create material heart ambient 0.3 0 0.3 diffuse 1 0 0 specular 0.5 0.5 0.5 shininess 0.5
```

By default when the three fibre angles are used as a vector they are generalized into a 9 component tensor. The first three components are a vector aligned with the fibre direction, the second three components are the vector which is orthogonal to the fibre direction but which also lies in the sheet, and the third group of three components are a vector orthogonal to the first two. Therefore by drawing stream ribbons aligned with this coordinate system you get ribbons which track the fibre direction with the ribbons lying in the sheet plane. As this happens automatically if you list the values of the fibres field you see the three angles.

```
gfx modify g_element heart element_points use_elements glyph point label fibres
   discretization "1*1*1" cell_centres material default
```

Delete these labels:

```
gfx modify g_element heart element_points use_elements glyph point label fibres delete
```

You can manually expand the three angles into the 9 component tensor:

```
gfx define field fibre_vector fibre_axes fibre fibres coordinate coordinates
gfx modify g_element heart element_points use_elements glyph point label fibre_vector
   discretization "1*1*1" cell_centres material default
```

Delete these again:

```
gfx modify g_element heart element_points use_elements glyph point label fibre_vector delete
```

Display basic streamlines representing the fibres:

```
gfx mod g_e heart streamlines ribbon material green vector fibres length 100
```

Display surfaces of heart chambers:

```
gfx mod g_e heart surfaces exterior face xi3_0 mat heart
```

By reordering the components of the fibre_vector field we get a 3-vector field which is orthogonal to the fibre axes but still lies in the sheet. Note this makes a left-handed coordinate system unless we reverse the fibre direction, but this makes no difference for streamlines:

```
gfx define field sheet_vector composite fibre_vector.4 fibre_vector.5 fibre_vector.6
   fibre_vector.1 fibre_vector.2 fibre_vector.3 fibre_vector.7 fibre_vector.8 fibre_vector.9
gfx mod g_e heart streamlines ribbon material gold vector sheet_vector length 100
```

Streamlines can be rendered with data and spectrums in the normal way but can also show a value which is proportional to their length. This is particularly useful in fields, unlike the fibre system, where the velocity (magnitude) of the vector field is changing.

```
gfx mod g_e heart streamlines ribbon material green vector fibres length 100 travel_scalar
gfx mod spectrum default auto
```

There are additional options, all accessible through the graphical element editor for changing the profile of the streamline (ribbon is the current option, extruded rectangles or extruded ellipses are other options), a shortcut for showing the magnitude of the tracked field as data and for restricting the seed to only a singular specific element.

Additionally using the old commands you can seed streamlines at the points described by a data group which has an element_xi field. See the seed_data_group and seed_data_field under gfx create streamlines.

6.6 Example a/segmentation: Segmentation of image based fields

This example demonstrates how to segment image based computed fields. It also shows how to use the results of the segmentation to generate data points to describe the boundary of an anatomical feature. Segmentation refers to the process of partitioning an image into various regions. These regions are identified by sets of pixels with the same intensity values. Segmentation of medical images is typically used to identify the boundaries between different types of tissue and pinpoint the position of objects such as bones and organs.

The human brain is very good at segmenting images but it is quite a difficult task to automate. Many different segmentation algorithms have been developed and for a given image one approach may produce better results than another.

CMGUI implements segmentation via computed fields, using the Insight Tookit (ITK) library code. For background information on segmentation see chapter 9 of the ITK software guide. The latest version of the guide can be found at www.itk.org/ItkSoftwareGuide.pdf. Currently CMGUI supports the following segmentation algorithms:

- Connected threshold, a region growing algorithm (see section 9.1.1 of the ITK software guide)
- Fast marching, a level set segmentation algorithm (see section 9.3.1 of the ITK software guide)

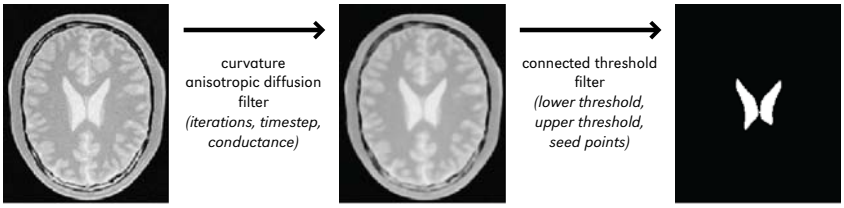
This example demonstrates the use of both methods to segment out different areas from a sample brain image

Connected Threshold segmentation

Connected threshold segmentation is a very simple region growing method. One or more seed points are specified for the image and the region grows from those points, adding neighbouring pixels to the region if the pixel intensity falls within a specific interval. The regions grow until no more pixels can be added.

Noise present in an input image can reduce the effectiveness of this filter, so it is often useful to remove noise by first passing the image through an edge preserving smoothing filter.

The filter pipeline is as follows (inputs to the filters shown in brackets):



We will use the connected threshold filter to segment out the left and right ventricle.

Read in a mesh to hold image which is currently required for cmiss:

```
gfx read nodes $example/square.exnode
gfx read elements $example/square.exelem
```

Read in the image to a texture and define a field based on it:

```
gfx create texture input_image image png:$example/BrainProtonDensitySlice.png nearest width 1
   height 1;
gfx define field input_image sample_texture coordinate xi texture input_image;
gfx create material input_image texture input_image
```

Set up a dummy image of the correct size for the output field This will be used to write out filtered images:

```
gfx create texture output_image image png:$example/BrainProtonDensitySlice.png specify_format
   i nearest width 1 height 1;
gfx create material output_image texture output_image
```

Create a spectrum:

```
gfx create spectrum gray_spectrum clear overwrite_colour
gfx modify spectrum gray_spectrum linear range 0 1 extend_above extend_below monochrome
   colour_range 0 1 ambient diffuse component 1
```

Set up the pipeline for the connected threshold filter:
Smooth the image using anisotropic diffusion with 5 iterations Note that if trying to duplicate the itk segmentation tests that the conductance and time step do not require dividing by 255:

```
gfx define field smoothed curvature_anisotropic_diffusion_filter field input_image
   num_iterations 5 time_step 0.125 conductance 9.0;
```

Segment the image using the connected threshold filter:
The seed points are defined in terms of their position within the texture described by normalised coordinates. In terms of pixel value the seed points are (85,107) and (96,107) The image pixel dimensions are 181x217 which gives the normalized seed points as (0.47,0.5) and (0.53,0.5) Pixels will be added to the region if the fall within the intensity range 0.85 to 1.0.

```
gfx define field connected_threshold_segmentation connected_threshold_filter field smoothed
   num_seed_points 2 dimension 2 seed_points 0.47 0.5 0.53 0.5 lower_threshold 0.85
   upper_Threshold 1.0
```

Write out the segmented image to a file:

```
gfx modify texture output_image nearest width 1 height 1 evaluate_image field
   connected_threshold_segmentation spectrum gray_spectrum texture_coord xi element_group
   square;
gfx write texture output_image file connected_threshold_segmentation.png;
```

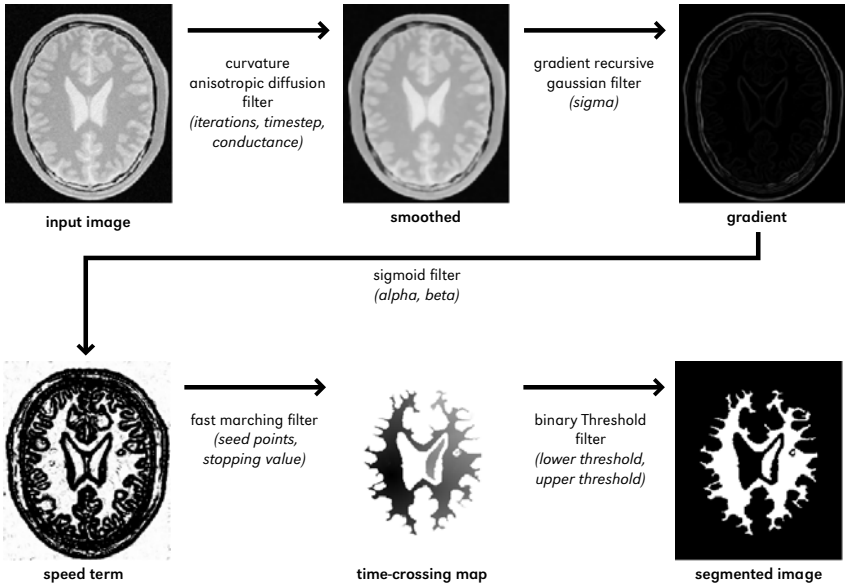
Fast marching segmentation

Fast marching segmentation is a level set segmentation method which can be used if the equation governing the level set evolution is very simple. Level set segmentation is a very powerful technique that uses a numerical method for tracking the evolution of contours and surfaces, based on terms derived from an input image.

In the case of the fast marching segmentation filter, the "speed term" required to calculate the segmentation is typically some function of the image gradient. As with the connected threshold example, the input image is usually smoothed first, before computing the gradient and then some function of the gradient to give the speed image. The fast marching filter then produces a time map, indicating how long it takes to get from a seed point to any given point on the image. By using a threshold filter on the time map a segemented image can be obtained.

The fast marching filter segmentation method demonstrated here uses the same filters outlined in the collaboration diagram in the ITK software guide.

The filter pipeline is as follows (inputs to the filters shown in brackets):



We will use the fast marching image filter to segment out the white matter and the right ventricle.

Assemble the pipeline for the fast marching image filter segmentation:

Smooth the image using anisotropic diffusion with 5 iterations. Note that if trying to duplicate the ITK segmentation tests, that the conductance and time step do not require dividing by 255.

```
gfx define field smoothed curvature_anisotropic_diffusion_filter field input_image
   num_iterations 5 time_step 0.125 conductance 9.0;
```

Calculate the gradient of the image using a sigma value of 1.0/255:

```
gfx define field gradient gradient_magnitude_recursive_gaussian_filter field smoothed sigma
   0.0039216;
Calclulate the speed term using a sigmoid with alpha -0.3/255 and 2.0/255:
gfx define field sigma sigmoid_filter field gradient alpha -0.0011765 beta  0.0078431;
```

Calculate a time-crossing map indicating how long it takes to reach pixels from the starting seed points, using the fast marching image filter The seed points are defined in terms of their position within the texture described by normalised coordinates. In terms of pixel value the seed points are (56,92) and (96,107) The

image pixel dimensions are 181x217 which gives the normallised seed points as (0.30939,0.423396) and (0.53,0.5).

```
gfx define field time_cross_map fast_marching_filter field sigma num_seed_points  2
   seed_points 0.31 0.42 0.53 0.5 stopping_value 1000000
```

Calculate the segmented image by threshold the image:
Pixels with intensities between 0 (black) and 0.95 (almost white) will be included in the regions.

```
gfx define field fast_marching_segmentation binary_threshold_filter field time_cross_map
   lower_threshold 0 upper_threshold 0.95
```

Write out the output images for each stage of filtering to check that the filter pipeline has worked:

```
gfx modify texture output_image nearest width 1 height 1 evaluate_image field smoothed
   spectrum gray_spectrum texture_coord xi element_group square;
gfx write texture output_image file smoothed.png;
gfx modify texture output_image nearest width 1 height 1 evaluate_image field gradient
   spectrum gray_spectrum texture_coord xi element_group square;
gfx write texture output_image file gradient.png;
gfx modify texture output_image nearest width 1 height 1 evaluate_image field sigma spectrum
   gray_spectrum texture_coord xi element_group square;
gfx write texture output_image file sigma.png;
gfx modify texture output_image nearest width 1 height 1 evaluate_image field time_cross_map
   spectrum gray_spectrum texture_coord xi element_group square;
gfx write texture output_image file time_cross_map.png;
gfx modify texture output_image nearest width 1 height 1 evaluate_image field
   fast_marching_segmentation spectrum gray_spectrum texture_coord xi element_group square;
gfx write texture output_image file fast_marching_segmentation.png;
```

Generate node points around the segmented image:
Use a fine discretization with for generating the isolines.

```
gfx modify g_element square general element_discretization "512*512*512"
```

Define the density field used to control the number of points generated Here we want a constant number of points per segment length:

```
gfx define field point_density constant 50;
```

Create the isolines:

```
gfx modify g_element square iso_surfaces as isolines select_on iso_scalar
   fast_marching_segmentation iso_value 0.5 use_elements select_on material red
   selected_material default_selected render_shaded data point_density;
```