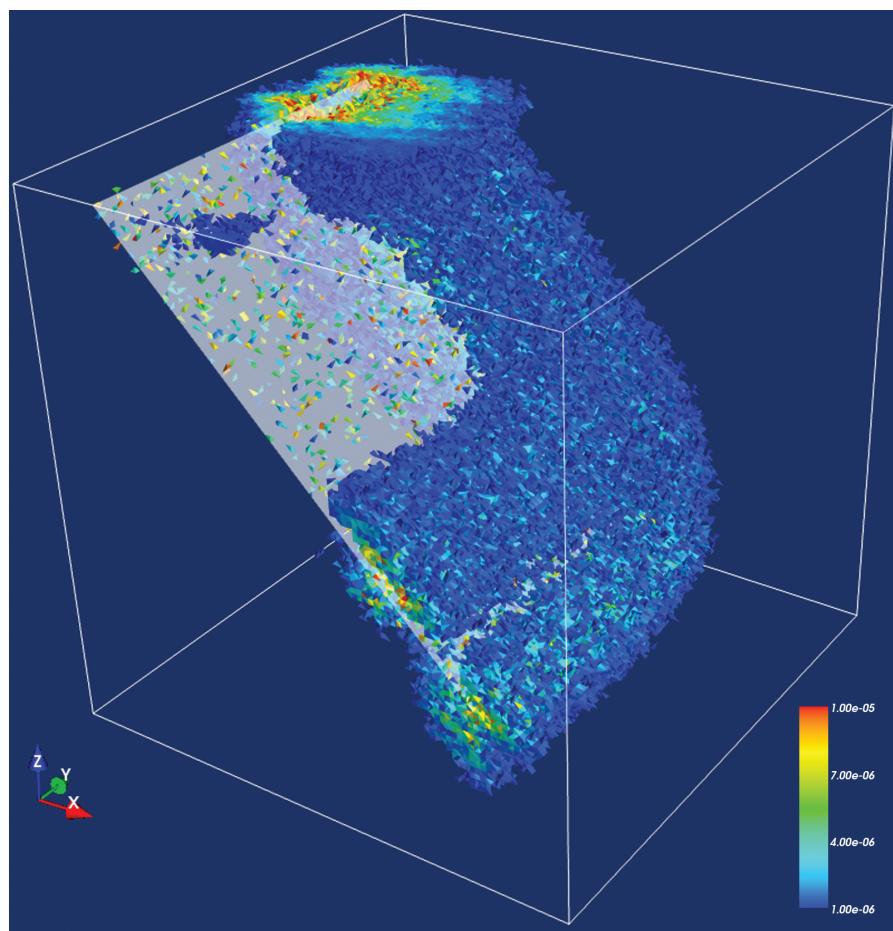


Cigma

User Manual
Version 0.9.1



www.geodynamics.org

Luis Armendariz
Susan Kientz

Cigma

© California Institute of Technology
Version 0.9.1

April 9, 2008

Contents

1	Introduction	7
1.1	About Cigma	7
1.2	Citation	7
1.3	Support	8
2	Installation and Getting Help	9
2.1	Getting Help	9
2.2	Installating from Source	9
2.2.1	HDF5	9
2.2.2	VTK	10
2.2.3	NumPy	10
2.2.4	PyTables	10
2.2.5	FIAT	10
2.2.6	HDFView (optional)	11
3	Error Analysis	13
3.1	Introduction	13
3.2	Distance Measures	13
3.3	Global Error Approximation	13
3.4	Comparing Residuals	15
4	Running Cigma	17
4.1	Command Line Interface	17
4.2	Input and Output Formats	17
4.3	Cigma Datasets	18
4.3.1	Comparing Two Finite Element Fields	19
4.3.2	Comparing against Known Values	19
4.3.3	Comparing against a Known Function	20
4.3.4	Specifying an Integration Mesh	20
4.3.5	Specifying a Quadrature Rule	21
5	Examples	23
5.1	Convergence	23
5.2	Visualization	24
A	File Formats	25
A.1	Input File Format	25
A.1.1	HDF5	25
A.1.2	VTK	26
A.1.3	Text	26
A.2	Output File Format	27
B	License	29

List of Figures

2.1	With HDFView, you can view the internal file hierarchy in a tree structure, add new datasets, and modify or delete existing datasets	11
4.1	Reference Tetrahedron	19
4.2	Reference Hexahedron	19
5.1	Geometry of Strike-Slip Benchmark	23

Chapter 1

Introduction

1.1 About Cigma

The CIG Model Analyzer (Cigma) consists of a suite of tools for comparing numerical models. CIG has developed Cigma in response to demand from the short-term tectonics community for a simple tool that can perform rigorous error analysis on their finite element codes. The long-term goal for Cigma, however, is for it to be used for nearly all geodynamics modeling codes.

The current version of Cigma is intended for the calculation of L_2 residuals for finite element models. This software was designed for performing the following three main tasks: (1) error analysis, (2) benchmarking, and (3) code verification. In error analysis, Cigma can calculate a global measure of the error between two solutions by performing an integration over a discretized version of the common domain. This comparison can take place even when the underlying discretizations do not overlap. In benchmarking, Cigma can help the geodynamics community agree on a standard solution to specific problems by facilitating the process of comparing different numerical codes against each other. Lastly, as an automated tool, Cigma can help application developers create regression tests to ensure that software changes do not affect the consistency of the results.

At its core, Cigma draws from a variety of libraries. The FInite element Automatic Tabulator (FIAT) (www.fenics.org/wiki/FIAT) Python Library supports generation of arbitrary order instances of the Lagrange elements on lines, triangles, and tetrahedra. It can also generate higher order instances of Jacobi-type quadrature rules on the same element shapes. The Approximate Nearest Neighbor (ANN) (www.cs.umd.edu/~mount/ANN/) Library, written in C++, supports data structures and algorithms for both exact and nearest-neighbor searching in arbitrarily high dimensions.

Both of these libraries extend and generalize Cigma's functionality so it can handle other types of elements, and provide the ability to compare vector fields.

1.2 Citation

Computational Infrastructure for Geodynamics (CIG) is making this source code available to you in the hope that the software will enhance your research in geophysics. This is a brand-new code and at present no papers are published or press. Please cite this manual as follows:

- Armendariz, L., and S. Kientz. *Cigma User Manual*. Pasadena, CA: Computational Infrastructure of Geodynamics, 2008. URL: geodynamics.org/cig/software/cs/cigma/cigma.pdf

CIG requests that in your oral presentations and in your papers that you indicate your use of this code and acknowledge the author of the code and CIG (geodynamics.org).

1.3 Support

Cigma development is based upon work supported by the National Science Foundation under Grant No. EAR-0406751. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The code is being released under the GNU General Public License.

Chapter 2

Installation and Getting Help

2.1 Getting Help

For help, send an e-mail to the CIG Computational Science Mailing List (cig-cs@geodynamics.org). You can subscribe to the `cig-cs` mailing list and view archived discussions at the CIG Mail Lists web page (geodynamics.org/cig/lists). If you encounter any bugs or have problems installing Cigma, please submit a report to the CIG Bug Tracker (geodynamics.org/bugs).

2.2 Installating from Source

To install Cigma, download the source package from the CIG Cigma web page (geodynamics.org/cig/software/packages/cs/cigma). You will need the GNU C++ compiler for this step. The HDF5 and VTK libraries, described later in this section, are also required.

After installing the necessary dependencies, simply unpack the source tar file and issue the following commands

```
$ tar xvfz cigma-0.9.1.tar.gz
$ cd cigma-0.9.1
$ ./configure --with-hdf5=$HDF5_PREFIX --with-vtk=$VTK_PREFIX
$ make
$ make install
```

Both HDF5 and VTK prefixes are required if you have installed these libraries in a custom location.

2.2.1 HDF5

The Hierarchical Data Format (HDF5) library is available for download from The HDF Group (hdfgroup.org/HDF5). Binaries can be obtained at hdfgroup.org/HDF5/release/obtain5.html (hdfgroup.org/HDF5/release/obtain5.html). To install from source, download the latest stable 1.6 version of this library (currently 1.6.7) and issue the following commands

```
$ tar xvfz hdf5-1.6.7
$ cd hdf5-1.6.7
$ ./configure --prefix=$HOME/opt/hdf5/1.6.7
$ make
$ make install
```

Without the custom prefix, this installation procedure will target your `/usr` directory.

2.2.2 VTK

The Visualization Toolkit (VTK) library is a popular open source graphics library for scientific visualizations. We recommend that you obtain the binaries from a package manager, making sure to install the associated development headers along with the library. The source for the VTK library is available from Kitware, Inc. (www.vtk.org/get-software.php). If you decide to compile VTK from source, you will also need the CMake build environment, available from CMake (cmake.org).

After downloading the source package, you can issue the following commands

```
$ tar xvfz vtk-5.0.4.tar.gz
$ cd VTK
$ mkdir build
$ cd build
$ ccmake ..
$ make
$ make install
```

2.2.3 NumPy

NumPy is a Python module which adds support for large multi-dimensional arrays and matrices, and is available for download from the NumPy Home Page (numpy.scipy.org). To install this extension module from source, download it and issue the following command in the NumPy source directory

```
$ python setup.py install
```

To verify the installation, the command ‘`import numpy`’ should run successfully from within your standard Python interpreter shell.

2.2.4 PyTables

PyTables is a Python extension module that builds on top of the HDF5 library. It provides a convenient scripting interface to manipulate HDF5 files, which can be used to manipulate the input/output files created by Cigma. PyTables is available for download from PyTables (www.pytables.org).

To install this extension from source, download the latest stable version (currently 2.0.2) and issue the following commands

```
$ tar xvfz pytables-2.0.2
$ cd pytables-2.0.2
$ python setup.py install
```

To verify the installation, the command ‘`import tables`’ should run successfully from within your standard Python interpreter shell.

2.2.5 FIAT

The Finite Element Automatic Tabulator (FIAT) Python module supports generation of arbitrary order instances of the Lagrange elements, as well as arbitrary order instances of the Jacobi-type quadrature rules on the same element shapes. It is available from The FEniCS Project (www.fenics.org/wiki/FIAT).

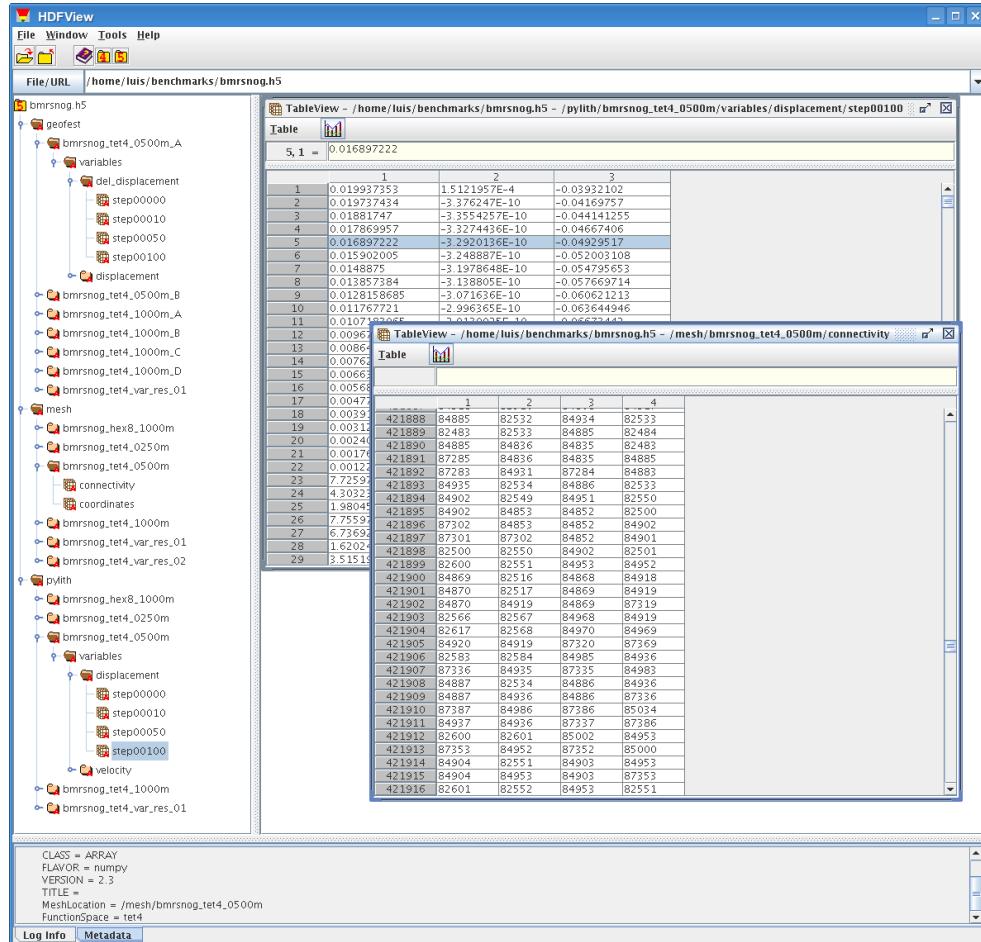
To install this module, download it and issue the following command inside the FIAT source directory:

```
$ sudo python setup.py install
```

You should now be able to run ‘`import FIAT`’ from within your standard Python interpreter shell.

2.2.6 HDFView (optional)

NCSA HDFView is a graphical user interface tool for accessing data in your HDF5 files. You can use it for viewing the internal group hierarchy (see Figure 2.1), adding new datasets, and modifying or deleting existing datasets, or altering metadata on groups and datasets. You can download it from the HDFView home page (hdf.ncsa.uiuc.edu/hdf-java-html/hdfview).



TableView - /home/luis/benchmarks/bmrsnog.h5 - /pylith/bmrsnog_tet4_0500m/variables/displacement/step00100						
5.1	0.016897222	1	2	3		
1	0.019937253	1.5121957E-4	-0.03932102			
2	0.019737434	-3.376247E-10	-0.04169757			
3	0.01881747	-3.3554257E-10	-0.044141255			
4	0.017869957	-3.3274436E-10	-0.04667406			
5	0.016897222	-3.2920136E-10	-0.04929517			
6	0.016897205	-3.2576484E-10	-0.051824408			
7	0.0168972	-3.1978648E-10	-0.054795653			
8	0.0168577384	-3.158805E-10	-0.057666714			
9	0.0128158685	-3.071636E-10	-0.060621213			
10	0.011767721	-2.993656E-10	-0.063644946			
11	0.01071623065	-2.932035E-10	-0.066323442			
12	0.00961					
13	0.00864					
14	0.00762					
15	0.00663					
16	0.00564					
17	0.00473					
18	0.0039	1	2	3		
19	0.0031	421888	84885	82532	82533	
20	0.0024	421889	82483	82533	84885	
21	0.00172	421890	84885	84836	82483	
22	0.0012	421891	87285	84836	84885	
23	7.7392	421892	67283	84931	87284	84883
24	4.3022	421893	84935	82534	84886	82533
25	1.9804	421894	84902	82549	84951	82550
26	7.7559	421895	84902	84853	84852	82500
27	6.7369	421896	87302	84853	84852	84902
28	1.6202	421897	87302	87302	84852	84911
29	3.5151	421898	82500	82551	84902	82501

TableView - /home/luis/benchmarks/bmrsnog.h5 - /mesh/bmrsnog_tet4_0500m/connectivity				
1	2	3	4	5
421898	84885	82532	84934	82533
421899	82483	82533	84885	82484
421900	84885	84835	84852	82500
421901	87285	84836	84885	84883
421902	67283	84931	84886	84918
421903	84902	82534	84851	84919
421904	87302	84853	84852	84902
421905	84919	87320	87320	87369
421906	82581	82584	84985	84936
421907	87336	84935	87335	84983
421908	84887	82534	84886	84936
421909	84887	84936	84886	87336
421910	87387	84986	87386	85034
421911	84937	84936	87337	87386
421912	82601	82601	85002	84953
421913	87352	84952	87352	85000
421914	84934	82551	84903	84953
421915	84904	84953	84903	87353
421916	82601	82552	84953	82551

```

CLASS = ARRAY
FLAVOR = numpy
VERSION = 2.3
TITLE =
MeshLocation = /mesh/bmrsnog_tet4_0500m
FunctionSpace = tet4

```

Figure 2.1: With HDFView, you can view the internal file hierarchy in a tree structure, add new datasets, and modify or delete existing datasets

Chapter 3

Error Analysis

3.1 Introduction

When studying differential equations that represent physical systems we often obtain solutions by using a variety of techniques. Solving these equations using classical analytical methods is almost impossible, so we often resort to numerical algorithms such as the finite element method.

For assessing the quality of the solution to our equations, it is important to quantify the error in our numerical solutions. To calculate this error, we will have to compare these solutions against each other through the use of a distance measure between two functions.

3.2 Distance Measures

The simplest possible quantitative measure of the difference between two distinct functions consists of taking the pointwise difference at a common set of points. While no finite sample of points can perfectly represent a continuum of values, valuable information can be inferred from the statistics of the resulting set of differences.

XXX: motivate this measure..validation

A very useful distance measure we can use is the L_2 norm, defined by the following integral

$$\varepsilon = \|u - v\|_{L_2} = \sqrt{\int_{\Omega} \|u(\vec{x}) - v(\vec{x})\|^2 d\vec{x}}$$

where u and v are the two functions on a global coordinate system. This gives us a single global estimate ε representing the distance between the two fields $u(\vec{x})$ and $v(\vec{x})$. Alternatively, you may think of this as the size, or norm, of the residual field $\rho(\vec{x}) = u(\vec{x}) - v(\vec{x})$.

3.3 Global Error Approximation

If we discretize the domain Ω into an appropriate set of cells $\Omega_1, \Omega_2, \dots, \Omega_{n_{el}}$, we can compute the global error ε^2 as a sum over localized cell contributions, $\varepsilon^2 = \sum_e \varepsilon_e^2$, where each cell residual ε_e^2 is given by

$$\varepsilon_e^2 = \int_{\Omega_e} \|u(\vec{x}) - v(\vec{x})\|^2 d\vec{x}$$

In general, we won't be able to integrate each cell residual ε_e^2 exactly since either of the functions u and v may have an incompatible representation relative to the finite element space on each domain Ω_e . However, we can still calculate an approximation of each cell residual by applying an appropriate quadrature rule with a tolerable truncation error [2].

To obtain an approximation to the integral of a function $F(\vec{x})$ over a cell Ω_e , we simply apply the quadrature rule with weights $w_{e,1}, w_{e,2}, \dots, w_{e,n_Q}$ and integration points $\vec{x}_{e,1}, \vec{x}_{e,2}, \dots, \vec{x}_{e,n_Q}$ appropriate for the physical element Ω_e :

$$\int_{\Omega_e} F(\vec{x}) d\vec{x} = \sum_{q=1}^{n_Q} w_{e,q} F(\vec{x}_{e,q})$$

Applying this quadrature rule directly over the entire physical domain $\Omega = \cup \Omega_e$ gives us

$$\int_{\Omega} F(\vec{x}) d\vec{x} = \sum_{e=1}^{n_{el}} \sum_{q=1}^{n_Q} w_{e,q} F(\vec{x}_{e,q})$$

For efficiency reasons, it is undesirable in finite element applications to perform calculations in a global coordinate system. To avoid duplication of work, shape function evaluations may be performed once on a reference cell $\hat{\Omega}$ and then transformed back into the corresponding physical cell Ω_e as needed.

To compute integrals of F in a reference coordinate system, we need to apply a change of variables:

$$\int_{\Omega_e} F(\vec{x}) d\vec{x} = \int_{\hat{\Omega}} F(\vec{x}_e(\vec{\xi})) J_e(\vec{\xi}) d\vec{\xi}$$

where the additional factor $J_e(\vec{\xi}) = \det \left[\frac{d\vec{x}_e}{d\vec{\xi}} \right]$ is the Jacobian determinant of the reference map $\vec{x}_e(\vec{\xi}) : \hat{\Omega} \rightarrow \Omega_e$. This map describes how the physical points $\vec{x} \in \Omega_e$ are transformed from the reference points $\vec{\xi} \in \hat{\Omega}$. Put another way, the inverse reference map $\vec{\xi} = \vec{x}_e^{-1}(\vec{x})$ tells us how the physical domain Ω_e maps into the reference cell $\hat{\Omega}$.

At this point, we can assume without loss of generality that every physical cell Ω_e can be derived from a single reference cell $\hat{\Omega}$, so that our quadrature rule becomes simply a set of weights w_1, \dots, w_{n_Q} and points $\vec{\xi}_1, \dots, \vec{\xi}_{n_Q}$ over $\hat{\Omega}$. After changing variables, we end up with the final form of the quadrature rule that we can use to integrate the global residual field:

$$\int_{\hat{\Omega}} F(\vec{x}_e(\vec{\xi})) J_e(\vec{\xi}) d\vec{\xi} = \sum_{q=1}^{n_Q} w_q F(\vec{x}_e(\vec{\xi}_q)) J_e(\vec{\xi}_q)$$

If we let $F(\vec{x}) = \|u(\vec{x}) - v(\vec{x})\|^2$ in the above expression, we can find the cell residual contribution over Ω_e from

$$\begin{aligned} \varepsilon_e^2 &= \int_{\Omega_e} \|u(\vec{x}) - v(\vec{x})\|^2 d\vec{x} \\ &= \int_{\hat{\Omega}} \|u(\vec{x}_e(\vec{\xi})) - v(\vec{x}_e(\vec{\xi}))\|^2 J_e(\vec{\xi}) d\vec{\xi} \\ &= \sum_{q=1}^{n_Q} w_q \|u(\vec{x}_e(\vec{\xi}_q)) - v(\vec{x}_e(\vec{\xi}_q))\|^2 J_e(\vec{\xi}_q) \end{aligned}$$

The global error $\varepsilon = \sqrt{\sum_e \varepsilon_e^2}$ is then approximated by the expression

$$\varepsilon = \sqrt{\sum_{e=1}^{n_{el}} \sum_{q=1}^{n_Q} w_q \|u(\vec{x}_e(\vec{\xi}_q)) - v(\vec{x}_e(\vec{\xi}_q))\|^2 J_e(\vec{\xi}_q)}$$

We can make a number of observations based on our final expression for the global error, (XXX: add observations here).

3.4 Comparing Residuals

For comparing errors of different solutions, you can normalize the global error by the norm of the exact solution. For a family of solutions $u^h(\vec{x})$, parametrized by the maximum element size h of the underlying discretization, and an exact solution $u(\vec{x})$, this normalized error is given by

$$\begin{aligned}\varepsilon_{rel} &= \frac{\|u - u^h\|_{L_2}}{\|u\|_{L_2}} \\ &= \frac{\int_{\Omega} \|u(\vec{x}) - u^h(\vec{x})\|^2 d\vec{x}}{\int_{\Omega} \|u(\vec{x})\|^2 d\vec{x}}\end{aligned}$$

This normalized error can be interpreted as the average error in the physical quantity being evaluated, so that a value of 0.01 corresponds to a 1% averaged error.

Even if the exact solution is not currently known, this normalized error may be used to test the accuracy between two or more numerical solutions, defined on successively refined meshes.

Chapter 4

Running Cigma

Cigma is primarily designed for calculating error estimates between arbitrary fields, so its primary usage is centered on the following operation

```
$ cigma compare -a field1 -b field2 -o residuals.vtk
```

You will need to provide two datasets describing each of the two fields, along with an integration rule and a discretization over which to integrate, although these last two will have reasonable defaults if they are not specified. This chapter aims to summarize the information you'll need to run Cigma.

4.1 Command Line Interface

Cigma is designed to be scriptable. Thus, all operations can be specified as command-line arguments given to a single executable called `cigma`. A set of available commands can be obtained by typing

```
$ cigma help
Usage: cigma <subcommand> [options]
Type 'cigma help <subcommand>' for help on a specific subcommand.
Available subcommands
    help
    list
    extract
    eval
    compare
```

XXX Quick description of each command

XXX: Also add the same description to the help page of the respective command

4.2 Input and Output Formats

XXX Reformat from newfile1.lyx

The default data storage format for Cigma is the Hierarchical Data Format (HDF5), a portable file format developed at the National Center for Supercomputing Applications (NCSA) (hdf.ncsa.uiuc.edu/HDF5). The HDF5 is designed for storing multi-dimensional arrays together with meta-data in a portable self-describing format.

The underlying data storage format for Cigma is the HDF5 format, due to its flexibility for storing and organizing large amounts of data. The Hierarchical Data Format (HDF) is designed for storing, retrieving, analyzing, visualizing, and converting scientific data. It uses a hierarchical structure that provides application programmers with a host of options for organizing how data is stored in HDF5 files. Using HDF5 datasets

in Cigma allows us to avoid having to convert between too many distinct formats. Moreover, due to the amount of disk I/O, large finite element meshes can be handled more efficiently in binary format.

Another VTK format for providing mesh and field inputs. [← XXX something wrong with that fragment. Did you mean "Another format for providing mesh and field inputs is the VTK format"?] You can easily examine the structure of an input file by using the `cigma list` command, which will simply reveal the names and dimensions of all datasets inside the specified file.

Inputs can be either HDF5 or VTK. Specifying the complete path to a dataset consists of the special form `filepath:dataset`, a colon-delimited pair of file path and dataset path.

Mesh inputs are currently only unstructured grids.

Only exception are the residuals, which are written in legacy VTK format as scalar cell data over an unstructured grid.

XXX New section?

Because Cigma relies on the ability of the user to specify dataset paths, we have provided a command called `list` for viewing the structure of an input file. Its usage is very simple.

To view the structure of an HDF5 file:

```
$ cigma list file.h5
/mesh/coordinates      Dataset {119827, 3}
/mesh/connectivity      Dataset {661929, 4}
/vars/displacement/step0 Dataset {119827, 3}
```

You can also view the structure of a VTK file with this command:

```
$ cigma list file.vtk
Reading file.vtk
Points = 119827
Cells = 661929
PointDataArray[0] = displacements_t0 (119827 x 3)
```

4.3 Cigma Datasets

Here we describe in a bit more detail the kind of datasets that Cigma expects.

The first kind of dataset required are meshes. In this version of Cigma, meshes are expected to be specified as an unstructured grid.

A field is a function which assigns a physical quantity to every point in a domain Ω . This quantity may correspond to a scalar, a vector, or even a tensor. As in Chapter 3, we shall use the same discretization Ω_e .

A finite element approximation to an solution field $\phi(\vec{x})$ on an element Ω_e is essentially specified by the weighed sum over a fixed set of local shape functions $\phi_{e,1}(\vec{\xi}), \phi_{e,2}(\vec{\xi}), \dots, \phi_{e,N}(\vec{\xi})$ defined on the appropriate reference cell,

$$\phi(\vec{x}) = \sum_{n=1}^N d_{e,n} \phi_{e,n}(\vec{\xi})$$

where the global point $\vec{x} \in \Omega_e$ corresponds to the local reference point $\vec{\xi} = \vec{x}_e^{-1}(\vec{x}) \in \hat{\Omega}$, and the weights $d_{e,n}$, also known as degrees of freedom, are given on. [← XXX does this sentence end correctly?] These shape functions define a basis for the function space on the finite element Ω_e , [← XXX more to this?]

For a mesh consisting of a single element type, we need to specify the (x_n, y_n, z_n) of its degrees of freedom, and the connectivity relations $\Omega_e = \{n_1, n_2, \dots\}$ among them which define each individual element in the corresponding discretization.

The most common element types used in the finite element method are given by the following shape functions, which define the interior of a tetrahedron with vertices at $a = (0, 0, 0)$, $b = (1, 0, 0)$, $c = (0, 1, 0)$, $d = (0, 0, 1)$,

$$\begin{aligned}
 N_a &= \frac{1}{2}(-1 - x - y - z) \\
 N_b &= \frac{1}{2}(1 + x) \\
 N_c &= \frac{1}{2}(1 + y) \\
 N_d &= \frac{1}{2}(1 + z)
 \end{aligned}$$

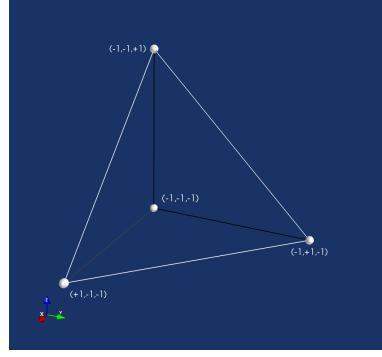


Figure 4.1: Reference Tetrahedron

Likewise, a hexahedral element with vertices located at $a = (-1, -1, -1)$, $b = (1, -1, -1)$, $c = (1, 1, -1)$, $d = (-1, 1, -1)$, $e = (-1, -1, 1)$, $f = (1, -1, 1)$, $g = (1, 1, 1)$, $h = (-1, 1, 1)$.

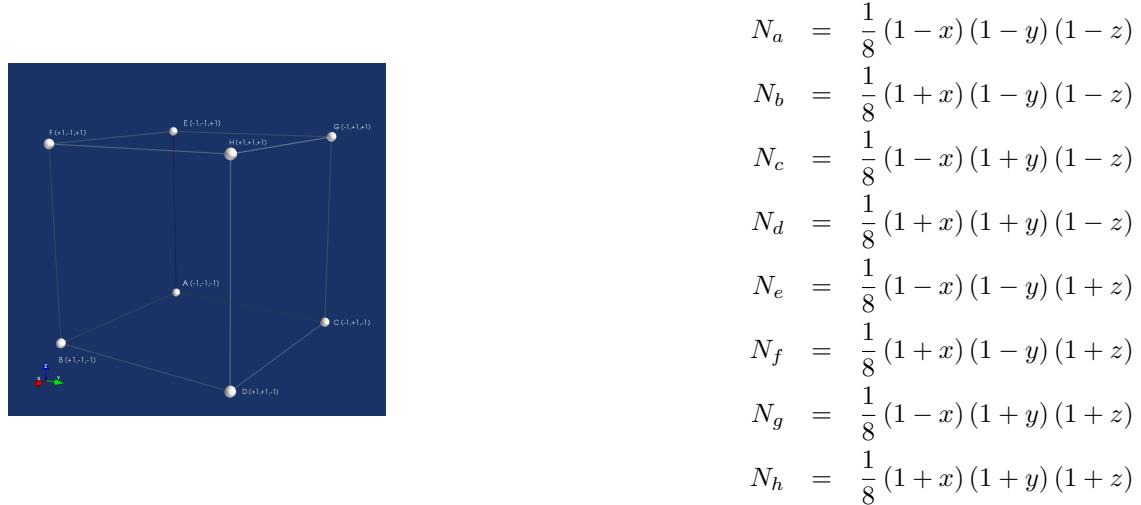


Figure 4.2: Reference Hexahedron

XXX: Take description of higher order elements from Karniadakis book

4.3.1 Comparing Two Finite Element Fields

Comparing two arbitrary finite element fields can be accomplished with the `cigma compare` command-line utility. By default, the comparison will take place over the elements in the mesh of the first field. Finally, the square of each of the local residual values are written to the specified VTK output file as cell-based scalars.

A basic comparison can be as simple as specifying the following arguments:

```
cigma compare --output=residuals.vtk
--first=field1.h5:/field1/stepN
--second=field2.h5:/field2/stepN
```

4.3.2 Comparing against Known Values

A finite element description might not always be available for one of the fields. However, you can break the comparison into several steps if you have a means to compute that field on any of the required points.

First, extract the global coordinates of the integration points. This will result in an explicit list of points over which to evaluate your field.

```
cigma extract --mesh=field1.h5:/model/mesh/
--output=points.h5:/projected_points
```

Now you can evaluate your function at the designated points. You can provide the path to the explicit set of values with

```
cigma compare --output=residuals.vtk
--first=field1.h5:/field1/stepN
--second-points=points.h5:/projected_points
--second-values=values.h5:/projected_values
```

4.3.3 Comparing against a Known Function

If one of your fields is easily described by an analytic expression, then you also have the option to compile your analytic function into Cigma, which will enable the `compare` command to reference your function by name. For example,

```
cigma compare --output=residuals.vtk
--first=field1.h5:/vars/displacement/step0
--second=disloc3d
```

You may also interact with your analytic function by using the `cigma eval` command, and obtain a set of values which may then be passed back to the `compare` command.

```
cigma eval --function=disloc3d
--points=points.h5:/projected_points
--values=values.h5:/disloc3d_values
```

Once these values have been obtained, we can use it to resume the calculation of

```
cigma compare --output=residuals.vtk
--first=field1.h5:/vars/displacement/step0
--second-points=points.h5:/projected_points
--second-values=values.h5:/disloc3d_values
```

Another built-in function you might find useful is the `zero` function, which is defined to return 0 everywhere. You can use this function for estimating the norm of your field.

4.3.4 Specifying an Integration Mesh

To override the mesh used in the integration, you can specify an extra argument providing the location of the mesh,

```
cigma compare [...]
--quadrature-mesh=mesh.h5:/model/mesh/
```

Alternatively, you can also specify the coordinates and connectivity arrays separately, in case they reside in separate files.

```
cigma compare [...]
--quadrature-mesh-coordinates=file1.h5:/model/mesh/coordinates
--quadrature-mesh-connectivity=file2.h5:/model/mesh/connectivity
```

4.3.5 Specifying a Quadrature Rule

To specify a quadrature rule, you will have to provide the quadrature weights and points in the appropriate reference element. This can be done with the following additional argument:

```
cigma compare [...]
--quadrature-rule=quadrature-rules.h5:/path/to/rule
```

You may also specify the location of the points and weights separately:

```
cigma compare [...]
--quadrature-rule-points=file.h5:/path/to/rule/points
--quadrature-rule-weights=file.h5:/path/to/rule/weights
```

In the `src/` directory of the Cigma distribution, you can find a Python script called `rules.py` that uses FIAT for calculating an appropriate set of quadrature rules for use in Cigma.

Chapter 5

Examples

In this chapter we will use sample datasets from the strike-slip benchmark case defined by the CIG Short-Term Tectonics working group. In this benchmark problem, we solve for the viscoelastic relaxation of stresses from a single finite earthquake while ignoring gravity. The problem is defined on a cube domain with sides of 24 km consisting of two layers of different material types. The top layer of the cube is nearly elastic, while the bottom layer is viscoelastic. (XXX: Anything else from Pylith manual or source?)

5.1 Convergence

When the exact solution of your equations is known, the norm of the error for a finite element solution is easily computed through the procedure given in Chapter 3.

Another way of checking the accuracy of a solution is by rerunning the same problem with a finer mesh. Recall that in the finite element method, the weight functions and trial solutions are chosen so that the finite element method is convergent. In other words, as the element size h decreases, the solution tends to the correct solution. Thus, if the difference between the coarse and fine mesh solutions is small, we can conclude that the coarse mesh solution is quite accurate. However, if a solution changes noticeably with refinement of the mesh, the coarse mesh solution is inaccurate, and even the finer mesh may be inadequate as well.

As can be seen from these results, the log of the error varies linearly with element size and the slope depends on the order of the element. Denoting the slope by α , then the error in the function can be expressed as

$$\log \varepsilon = \log C + \alpha \log h$$

where $\log C$ is an arbitrary constant, the y-intercept of the curve. The slope α is the rate of convergence of the element. We can rewrite this as

$$\varepsilon = Ch^\alpha$$

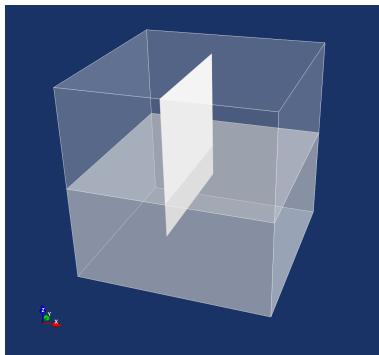


Figure 5.1: Geometry of Strike-Slip Benchmark

5.2 Visualization

Using MayaVi2, we may obtain a visualization of the Cigma datasets

Appendix A

File Formats

To differentiate between these formats, you will need to provide an appropriate extension for your input and output files. The supported extensions are currently `.h5` for HDF5, `.vtk` for VTK files, and `.txt` for simple text files.

A.1 Input File Format

There are five different objects you will want to use as input to Cigma, all of which can be represented as two-dimensional datasets. These are the

- mesh coordinates
- mesh connectivity
- field coefficients (degrees of freedom)
- quadrature rule points
- quadrature rule weights

A.1.1 HDF5

HDF5 files are organized in a hierarchical structure, similar to a UNIX file system. Two types of primary objects, groups and datasets, are stored in this structure. A group contains instances of zero or more groups or datasets, while a dataset stores a multi-dimensional array of data elements. Both kinds of objects are accompanied by supporting metadata known as attributes.

A dataset is physically stored in two parts: a header and a data array. The header contains miscellaneous metadata describing the dataset as well as information that is needed to interpret the array portion of the dataset. Essentially, it includes the name, datatype, dataspace, and storage layout of the dataset. The name is a text string identifying the dataset. The datatype describes the type of the data array elements. The dataspace defines the dimensionality of the dataset, i.e., the size and shape of the multi-dimensional array.

In Cigma you always provide an explicit path to every dataset, so you have a fair amount of flexibility for how you organize your datasets inside HDF5 files. For example, a typical Cigma HDF5 file could have the following structure.

```
model.h5
\__ model
    \__ mesh
        |__ coordinates      [nno x nsd]
        |__ connectivity     [nel x ndof]
    \__ variables
        \__ temperature
```

```

|   |__ step00000 [nno x 1]
|   |__ step00010 [nno x 1]
|   \...
|   __ displacement
|       |__ step00000 [nno x 3]
|       |__ step00010 [nno x 3]
|       \...
|   __ velocity
|       |__ step00000 [nno x 3]
|       |__ step00010 [nno x 3]
|       \...

```

Generally, Cigma will only require you to specify the path to a specific field dataset. If a small amount of metadata is present in your field dataset, the rest of the required information, such as which mesh and finite elements to use, will be deduced from that metadata.

MeshID an identifier assigned for use in linking child datasets to their parent mesh.

MeshLocation points to the HDF5 group which contains the appropriate coordinates and connectivity datasets.

FunctionSpace string identifier to determine which shape functions to use for interpolating values inside the element (e.g., tet4, hex8, quad4, tri3, ...).

DatasetType string identifier for classifying the type of data contained in the dataset (e.g., points, connectivity, degrees of freedom, quadrature rules, global quadrature points, global field values).

A.1.2 VTK

The current version of Cigma only supports VTK unstructured grid datasets of a single element type. You can still compare various unstructured grids with different element types to each other.

Note that while you typically provide a path (or name) for every dataset, this is not necessary when specifying a VTK mesh, since this data is taken from the special Points and Cells arrays, which you cannot rename. However, you will still need to provide a name when referring to the field coefficients, which are assumed to be stored as Point Data in the input VTK file.

For more information, you may want to refer to Visualization ToolKit's File Formats (www.vtk.org/pdf/file-formats.pdf) document.

A.1.3 Text

The text format is always in table form, with the dimensions of the table specified in the first line. For example, mesh coordinates can be specified in the following format

```

nno nsd
x1 y1 z1
x2 y2 z2
x3 y3 z3
...

```

Mesh connectivity with

```

nel ndofs
node_1 node_2 node_3 node_4 ...
node_1 node_2 node_3 node_4 ...
node_1 node_2 node_3 node_4 ...
...

```

A generic field with `ndim` components (i.e., scalar, vector, or tensor) is specified by

```
num ndim
f1 f2 f3 ..
f1 f2 f3 ..
...
```

In this last case, the number of rows could refer to either `nno` or `nel`, depending on whether the field is node-based or cell-based.

A.2 Output File Format

The output format for residuals consists simply of a list of scalars for each element in the discretization. If you specify a `.vtk` extension for your output file, this will result in an scalar dataset named `epsilon` stored using the legacy VTK file format in the Cell Data section of the output file. Note that this output consists of the squared values of the local residuals, so further post-processing will be necessary.

Appendix B

License

GNU GENERAL PUBLIC LICENSE Version 2, June 1991. Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software – to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. Copyright the software, and
2. Offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program" below refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification.") Each licensee is addressed as "you."

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
 - (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version," you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found. For example:

One line to give the program's name and a brief idea of what it does. Copyright © (year) (name of author)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright © year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items – whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

(signature of Ty Coon)

1 April 1989

Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Bibliography

- [1] Akin, J.E. (2005), *Finite Element Analysis with Error Estimators*, Butterworth-Heinemann, Oxford, 447 pp.
- [2] Encyclopaedia of Cubature Formulas (1998-2005), Katholieke Universiteit Leuven, Dept. Computerwetenschappen, www.cs.kuleuven.be/~nines/research/ecf/
- [3] Hughes, Thomas J.R. (2000), *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*, Dover, 672 pp.
- [4] Karniadakis, George E. and Spencer J. Sherwin (2005), *Spectral/hp Element Methods for Computational Fluid Dynamics, Second Edition*, Numerical Mathematics and Scientific Computation, Oxford, 657 pp.
- [5] Uesu, D., L. Bavoil, S. Fleishman, J. Shepherd, and C. Silva (2005), Simplification of Unstructured Tetrahedral Meshes by Point-Sampling, *Proceedings of the 2005 International Workshop on Volume Graphics*, 157-165, doi: 10.2312/VG/VG05/157-165.