Voxel-based finite element modelling with VOX-FE2

Voxel-based finite element modelling with VOX-FE2

Neelofer Banglawala[1], Iain Bethune[1], Michael Fagan[2] and Richard Holbrey[2]

[1]EPCC, The University of Edinburgh, [2]The University of Hull

Version 1.0, May 20, 2015

Acknowledgements

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk)

1 Introduction

1.1 Background

VOX-FE is a voxel-based FE software package for bone modelling, developed jointly between Hull Medical & Biological Engineering, Hull-York Medical School and Edinburgh Parallel Computing Centre. It has a sophisticated graphical user interface that allows the complex loading regimes that are inevitably present in biomechanical analyses to be readily applied to the model geometry, and the resultant 3D stress and strain patterns to be visualized easily.

The project has grown out of a number of research efforts over the past decade. In 2009, in conjunction with Hull York Medical School, various strands of code were brought together within a GUI facilitated by Borland/Embarcadero C++ Builder suite.

In contrast to the GUI, the solver was developed on Ubuntu and later parallelised on HECToR in 2012. A parallel input/output interface [1] (via NetCDF) was also added at this time.

1.2 Limitations of the GUI

In retrospect, the choice of C++ Builder was unfortunate for three reasons. Firstly, the GUI was tied to the Windows platform. Secondly, Embarcadero did not provide a 64-bit version of their compiler until 2013 (so that the GUI could only handle models of up to around ten million elements). Third, support for OpenGL graphics calls was only made possible by importing an external component developed by the user community.

The situation was not improved by the eventual release of the 64-bit version of C++ Builder, which did not support the existing OpenGL component. Further development

of the GUI, in particular to allow the analysis of larger models, would therefore have required a very significant re-write.

1.3 Limitations of the solver

Although written separately (and previously known as PARA-BMU), development of the solver was also rather organic, in that features were added over time in without restructuring. This made further development, debugging and refactoring incredibly difficult. Two aspects of the evolution were of particular concern.

Firstly, a combinatorial lookup scheme was used which meant that the number of materials was limited to three and; second, that partitioning for MPI was arbitrarily limited to division into similar sized blocks along the Z-axis of the model. These factors severely limited the potential scalability of models which can be handled by the solver.

1.4 Aims of the project

In 2014, ARCHER eCSE funding (project ID 'ecse01-015') was awarded to redevelop VOX-FE to improve its capabilities, performance and usability. In order to take advantage of libraries and technologies which have become *de facto*, open source standards over the intervening years, it was decided to move to different platforms for both the GUI and solver. Details of this this development project and discussion of the results are detailed herein: the new GUI is discussed in section 2, and development of the solver is described in section 3. Section 4 introduces the new remodelling tools which have been developed on ARCHER.

2 The VOX-FE2 ParaView plugin GUI

2.1 ParaView

ParaView [2] was chosen as the basis for the new GUI for several reasons:

- VOX-FE2 is essentially a visual tool to import data, generate and inspect meshes,

add boundary conditions and overlay strain data. Many of the methods needed to support this kind of functionality are already available in the Visualization Toolkit (VTK) which underpins ParaView.

- The incorporation of many existing visualization tools (clipping, contouring etc) into ParaView means that clear policies for passing data between filters, support for undoing/redoing, etc. already exist.

- The parallelization of VTK algorithms within ParaView affords the possibility of working with very large models (at least 10^{10} elements), given adequate computing resources.

The most flexible method of extending Paraview is to exploit the plugin interface. One drawback, however, is that the interface places particular demands upon the programmer and is currently only briefly documented. It is assumed, for example, that the data reside on a remote server to which the ParaView interface is really only able to communicate one way (read only). Fortunately, there are now quite a large number of plugin examples and resources which have been released into the public domain.

2.2 Plugin architecture

To support the requirements of VOX-FE2, six new functions – known as 'filters' in VTK – have been added to the ParaView interface via a plugin. The filters are available via buttons (where the Qt library is enabled) or from the ParaView Filters menu and are described briefly below:

- *voxfeITKReader:* The 'Reader' component uses the Insight Toolkit (ITK), if available, to import image data and convert to a voxel model. The Connected Component filter of ITK is employed to remove noise (disconnected voxels) and to check valid labels. A script header file is then generated, which is assumed to be fixed (although parameters can be adjusted later). A new type of group header file can also be read which allows similar materials to be considered together for the pur-

pose of adding boundary conditions. If ITK is not available, the VOX-FE solver script/model file can be read directly.

- *ExtractBlock:* This filter uses vtkExtractBlockFilter, with the important difference that the 'Prune Output' parameter is switched off to prevent further errors downstream. The main purpose is to be able to select generic surfaces such as 'bone', which might comprise different user-defined types (trabecular, cortical etc).

- *voxfeAddConstraintFilter:* ParaView already provides the ability to select and extract points from the model. This requires extra input from the user, but permits editing of point selections. The voxfeAddConstraintFilter filter allows additional data to be added to each selected point which is used to indicate nodal constraints or loading conditions (forces).

- *voxfeGlyphAnnotationFilter (Highlight Boundary Condition (BC)):* Points defined as BCs are passed through the annotation filter automatically to generate a 'glyph': 2D triangles and arrows are shown in the display to indicate fixing of nodal axes or loads. A highlight button allows emphasis on designated BC points and glyphs by pseudo-random colouring and increased point size (ParaView display only).

- *voxfeOutputScriptFilter:* This filter generates a file of BC data for input to the solver.

- *voxfeStrainFilter:* Solving the FE problem generates a displacement vector for each node of the model. Executing this filter reads the displacement file, computes strain parameters for each voxel/element and attaches these data to the underlying VTK model for colour-map display.

2.3 Test Results

To illustrate the capability of the current system, we devised several simple test models for which analytical solutions are available. An example – the cantilever beam – is shown in Figure 1. The beam is assumed to be a bone plug, 10mm long and 1mm in diameter with properties: Young's Modulus = 17 GPa, Poisson's ratio = 0.3.

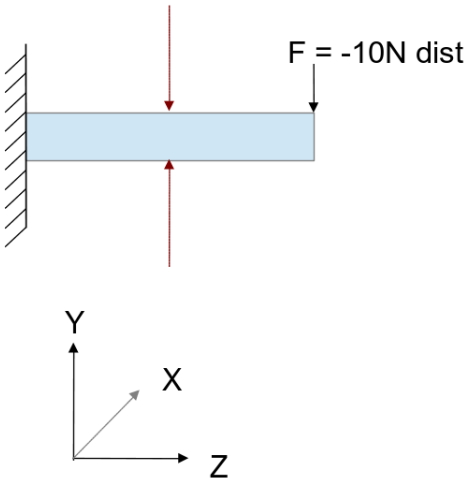

Figure 1: A cantilever beam model

One advantage of such a model is that it is reasonably easy to see if a poor solution is being generated, since we also have a theoretical result to compare against (for example, the maximum displacement on the Y-axis should be $-4.0 \times 10^{-6} m$). The solution would also be expected to improve as the resolution of the model increases. A screenshot showing the model and overlaid displacement/strain data is shown in Figure 2. A table of maximum displacements for some different sizes of model is given in Table 1.

Table 1: Maximum displacements for the beam model

Model size (elements)	**Max Y-Displacement (m)**
33,280	-3.71×10^{-6}
2,081,280	-3.91×10^{-6}
16,583,680	-3.94×10^{-6}

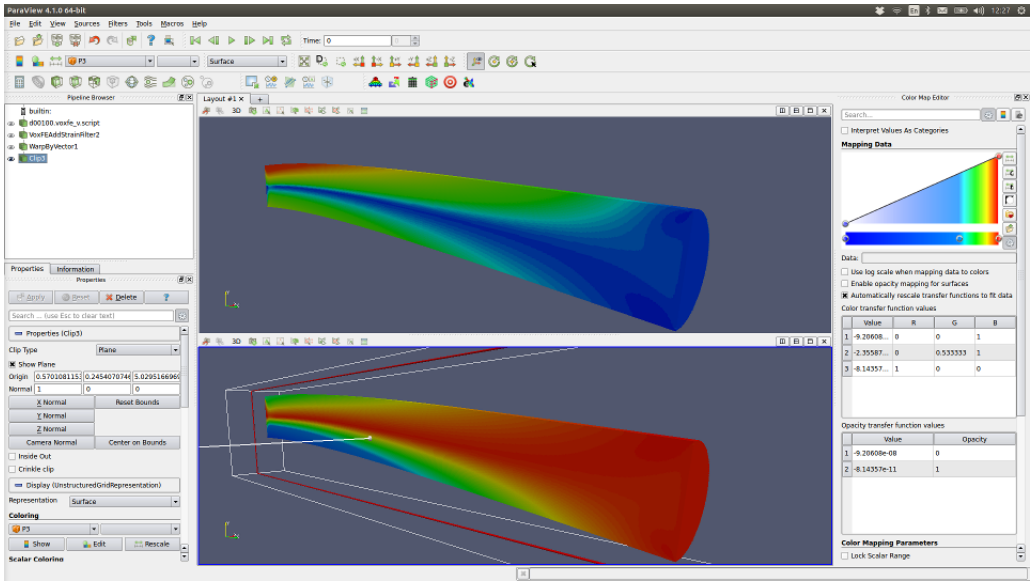

Figure 2: An 8-million element beam model with displacement and strain data overlaid

2.4 Discussion

The new VOX-FE2 plugin has been built and tested on Ubuntu 12/14 and Windows 7 with ParaView v4.3.1 and ARCHER with ParaView v4.1.0. ParaView has matured in recent years and now has an extensive user manual and wiki alongside user-contributed data and video tutorials. Many of the aims of the earlier VOX-FE project have been re-implemented within a single plugin library and a scheme that is clearly documented and extensible. Besides being cross-platform, the 64-bit code can reasonably accommodate models of c. 10^8 elements on current desktop machines.

Although the plugin interface creates a something of a barrier between the user and the data, the immediate benefit is that a client-server approach is automatically adopted, so that potentially models of $10^9 - 10^{10}$ elements may be handled, where HPC resources are available.

The plugin has been built on ARCHER, but following the Phase 2 upgrade in November 2014, we have not yet been able to test the plugin using the new RSIP protocol [3]. On a desktop machine (with 32GB RAM), however, the VOX-FE2 plugin has been shown to

handle models of over 100 million nodes.

3 The VOX-FE2 PETSc-based solver

3.1 PARA-BMU

The role of the solver is to determine the final displacements of a constrained bone model subject to specified of forces. It does this by iteratively solving a system of coupled linear equations that encode the relationships between the bone elements (the geometry), their material properties, and any forces and constraints which exist.

Written in C++ and MPI, the old solver (PARA-BMU) could successfully handle at most a 20 million element model, with a maximum of 3 different material types (for example bone, marrow and metal). It was designed to parallelise the bone model along the z-dimension (x- and y-dimensions were treated serially). A model with its greatest length in the z-dimension would therefore show the best scaling. For a 20 million element model, the old solver had good scaling up to 256 cores, but struggled to run on more than 512 cores (see [1], Figure 1). The C++ classes that collectively formed the old solver code also handled pre- and post-processing, and all MPI communication, including decomposition of the linear system across processors.

For VOX-FE to be a viable tool for modelling realistic large and complex bone models the solver needed to be able to:

- Solve models with at least 100 million elements with complex, sparse geometries.
- Parallelise the system along all dimensions, not just the z-dimension, to make the solution of large models feasible.
- Run efficiently on a large number of cores to take advantage of massively parallel HPC resources such as ARCHER.
- Capture varying bone densities and soft tissue detail by modelling an arbitrary number of different material types (at least 255 different types)

To implement the above functionality, and to ensure future extensibility of the solver, we decided to replace the old solver with an entirely new design.

3.2 PETSc-based solver

The key idea behind the new solver's design was to make use of the PETSc library [4]. "PETSc, the Portable, Extensible Toolkit for Scientific Computation developed by Argonne National Laboratory, is a powerful suite of data structures and routines for the scalable (parallel) solution of scientific applications"[1]. Specifically, PETSc's optimised parallel vector, matrix and Krylov SubsPace (KSP) routines efficiently set up and solve linear systems like those generated by VOX-FE. It was therefore a natural choice for providing the new solver's core functionality. PETSc also offers:

- A large choice of solution algorithms, allowing users to mix and match different combinations at runtime.
- The future possibility of increasing parallelism through use of heterogenous architectures (e.g. GPUs).

The new solver design is kept as simple as possible, so that users could treat it as a "black box". Figure 3 shows the architecture of the new solver. Users supply the solver with a set of input text files (script, model, materials and constraints), which may be generated using the GUI or written by hand. These are parsed by the solver I/O class (`VoxFE.cpp`) and used to drive the main solver class (`pFESolver.cpp`). This sets up the model, constructs the relevant system matrices, decomposes the system amongst all processors and calls PETSc to solve the linear system. The final displacements are output to a text file, ready for use by the GUI for visualisation, or for remodelling (see Section 4). This separates post-processing tasks from the main function of the solver (unlike in the old solver).

The current MPI decomposition is a simple contiguous splitting of the equations (or rows of the PETSc matrix) across processors, with each processor also keeping a local

[1] http://www.mcs.anl.gov/petsc/

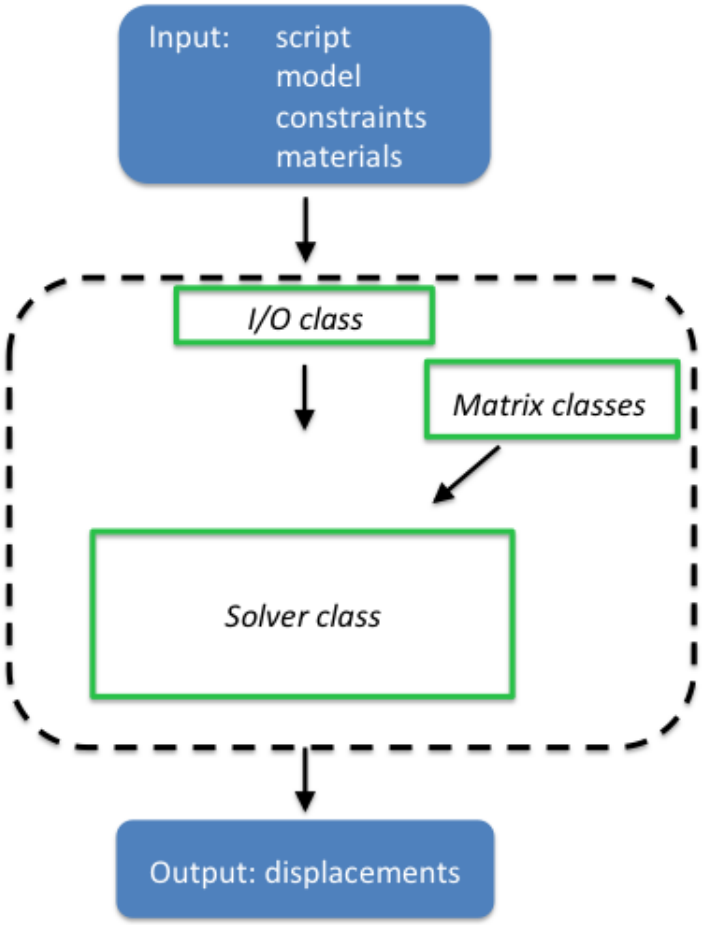

Figure 3: New solver architecture

copy of the entire model. This works reasonably well for small models with simple, dense geometries. However, to ensure good load balance for large, sparse, complex models, a better decomposition scheme will be needed for the system matrices, and efficient memory utilisation requires that large models be shared amongst processors instead of copied in entirety to each processor (see Section 3.4 for further discussion).

3.3 Performance results

We tested both solvers with the 8 million element cantilever model shown in Figure 1. Two different model alignments were used to test how well the new solver performed

against the old solver when tested under optimal conditions for the old solver (cantilever with length along z-axis, as shown in the figure) and when otherwise (cantilever with length along y-axis).

Figures 4 and 5 show the strong scaling performance of the new solver relative to the old solver for the z- and y-aligned models respectively. In both cases, the new solver has at least as good scaling as the old solver, despite the old solver's near-perfect scaling for an optimally aligned model (Fig. 4). The new solver also has better absolute performance: it is 2.5x - 4x faster than the old solver for a fixed number of cores and can run on at least 512 cores. With a non-optimally aligned model (Fig. 5), the new solver clearly outperforms the old solver: the old solver's performance degrades from around 32 cores and it fails to run on 128 cores (or more). In this case, the new solver (using 512 cores) is around 30x faster than the old solver running at its limit of 64 cores.

The new solver has also been tested on similar cantilever models made up of more than 4 different material types, showing agreement with theoretical results. We also tested the use of different solution algorithms (e.g. Conjugate Gradient with a Jacobi preconditioner, or GMRES with ILU[2]). In all cases, the new solver has consistently performed well. We have been able to run larger models (20 million elements) on the new solver, although even larger models will require investigation of the memory consumption of the new solver, as it currently appears to be in excess of what we expect.

3.4 Discussion

We set out to improve VOX-FE's ability to solve realistic bone models and this has been achieved with the new PETSc-based solver. The new solver can handle an arbitrary number of different materials, parallelise the problem in all dimensions, scale superlinearly to more than a thousand cores and solve models larger than the old solver could comfortably cope with. The next step in improving the solver's performance will be to parallelise the model loading (avoiding the replicated data bottleneck in the current implementation) and selecting an optimal partitioning of the system using ParMETIS [5].

[2]Generalized Minimal Residual method with an Incomplete Lower Upper preconditioner.

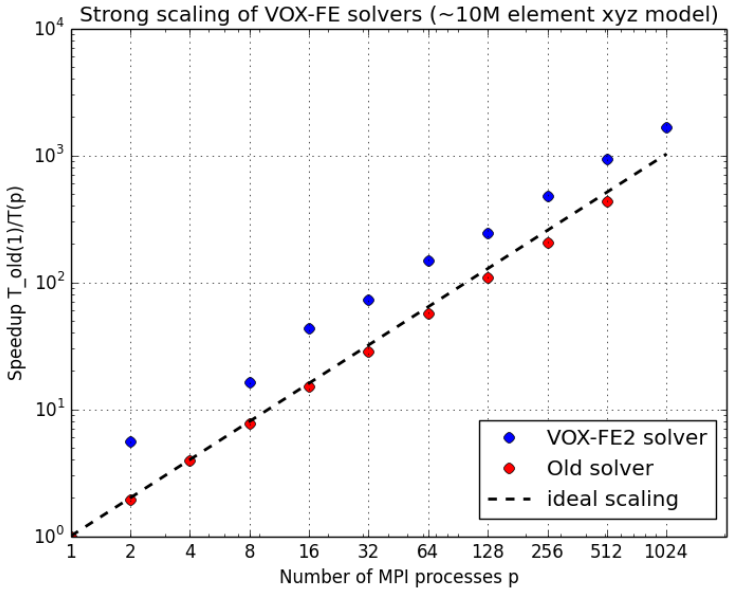

Figure 4: Strong scaling of new solver relative to old solver with optimal model alignment for old solver.

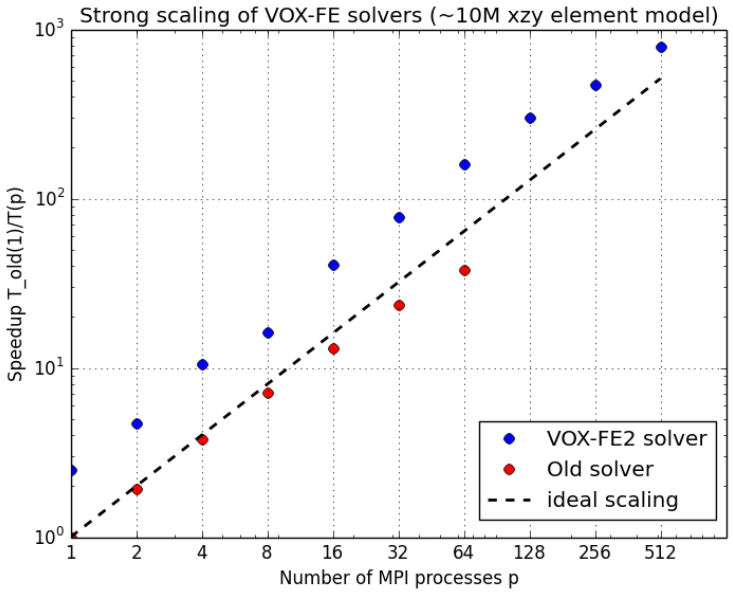

Figure 5: Strong scaling of new solver relative to old solver with non-optimal model aligned for old solver.

4 Remodelling

Remodelling describes the natural process of resorption and formation which occurs as bone responds to the load it experiences, where the parameters controlling the remodelling may be affected by age or disease etc. The precise mechanisms are still a matter of debate, and the quantitative effects on the parameters are currently unknown.

Our approach here is essentially prototypical: given a specimen under load and a minimal set of parameters, we would like to assess whether the process can be automated such that the remodelling history and convergence can be monitored. A key point, however, is that models should be sufficiently detailed such that the representation of the trabecular architecture is not undermined by evident weaknesses in the structure. A working hypothesis is that the trabeculae should be at least 5 voxels thick and preferably at least 10 [6].

4.1 Workflow

A flow diagram for remodelling tasks is included in Fig. 6. The solver is the most critical component in terms of demanding computer resources. Hence the other components fit around the solver since it is parallelized. It was anticipated that with N processors, $N-1$ would be allocated to the solver and the other algorithms would run in serial on the remaining processor.

To support the remodelling process, the solver makes two concessions:

- A small text file is generated to flag when the model data have been read.
- A second file is generated to flag when the solution displacement file has been written.

For the remaining steps (undertaken on the Nth process), a control structure maintains a record of voxel connectivity through the remodelling cycle and identifies surface voxels. This component – the control graph – is described below. Through the design of the plugin (see Section 2), the remaining elements of the cycle were already available to build as standalone executables, notably:

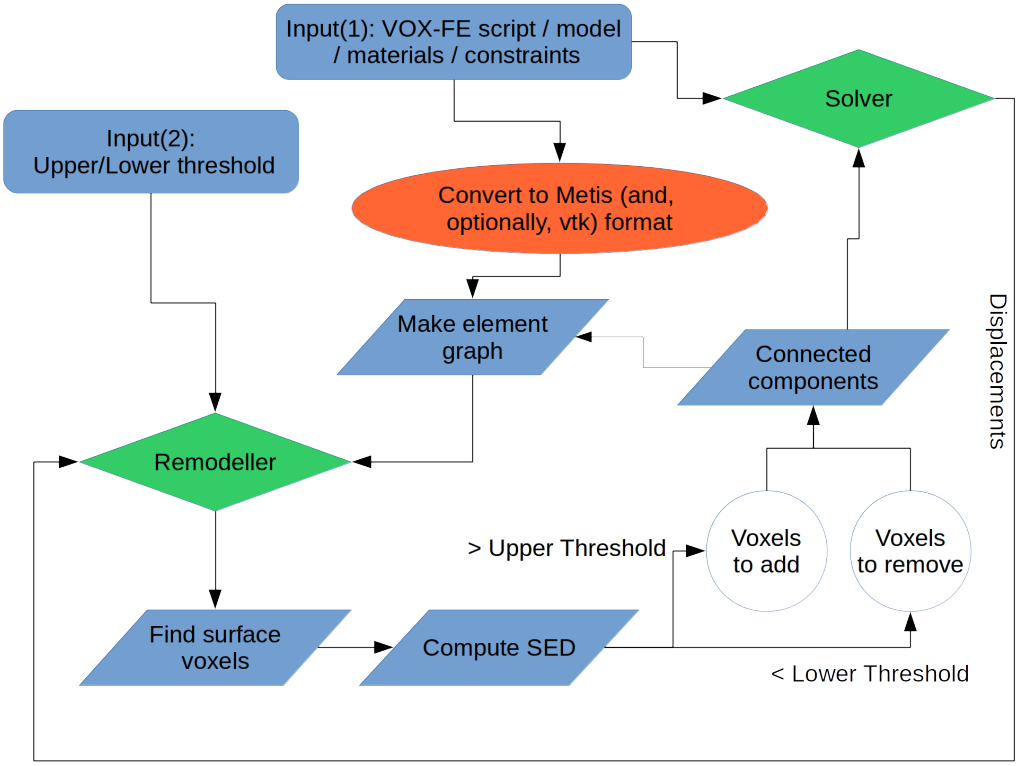

Figure 6: Workflow diagram of remodelling

- A utility to read the VOX-FE script/model files and convert to either a METIS graph format or to a legacy VTK file format.
- The capability to compute strain data for each voxel given dimensions and the displacements at each node.

The METIS [7] graph partitioning library provides a number of routines to manipulate graphs and is used to create the initial element adjacency file. METIS is available from http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.

4.2 Implementation

On ARCHER, the solver and various utility programs described are controlled through the use of bash scripting within the job submission file and a separate remodelling script,

executed first in the background.

In brief, execution is as follows:

1. The remodelling script starts (if not already running) but then pauses, waiting for the file 'InputRead.txt' to be created.

2. The solver reads the input model and generates a file 'InputRead.txt'.

3. As the solver continues towards a solution, the remodelling script reads the input model to create a graph of node connectivity for each voxel.

4. The utility `m2gmetis` (distributed with Metis) is used to create the 'dual' graph, in this case, a graph/adjacency table of the elements in the model

5. Optionally, the model is re-read to create a legacy VTK file. This allows the remodelling process to be easily replayed in ParaView as sequence.

6. The remodelling script pauses until the solver completes, creating a file 'FECompleted.txt'.

7. The element graph and displacement data are then read into the control graph routine to compute the remodelling parameter, currently strain energy density (SED), for each surface voxel.

8. Lower and upper remodelling thresholds are specified in the remodelling script (at step 1). Surface elements with an SED below the lower threshold are marked for removal, while those with an SED above the upper threshold are surrounded by a layer of new elements so that they are no longer at the surface i.e. they are completely surrounded by 26 neighbours.

9. The new model, incorporating resorption and formation determined at step 8, is written to the next directory in the iteration sequence.

10. Execution switches to the new directory and repeats from step 1.

4.3 Control Graph

The control graph reads element adjacency data generated by the METIS utility `m2gmetis`. It should be noted that the METIS library cannot handle the graph directly, as the underlying graph structure, once created, cannot be updated. Instead, to keep the memory requirement to a minimum, the code makes use of the property that a given voxel can only have up to 26 neighbours. Each graph entry ('vertex') is identified by a 64-bit integer ID into a C++ STL map and represents a voxel element. Each element records data for the following:-

Table 2: Control graph vertex data

Item	**Size (bytes)**	**Comment**
connectivity	4	Each neighbour represented by one bit on 3x3x3 grid
material	1	255 materials is an assumed maximum (0=background)
remodel	1	Specifies whether the given element can be remodelled (e.g. teeth would not be allowed to remodel)
SED	8	Computed strain energy density
status	2	If the voxel has been added (> 0) or removed (< 0)
component	4	For checking disconnected components

4.4 Bone Remodelling

The graph structure allows surface voxels to be identified easily by checking the connectivity bit pattern. The status member records if voxels have been added or removed. Where the computed SED:

- is less than the set lower threshold, status is decremented.
- is greater than set upper threshold, all neighbours are filled and new elements are added to the graph using the current material type.

One difficulty that arises from removing voxels is that some regions may become disconnected, potentially leading a singular system of equations during the next solution step. A two-pass connected components algorithm is employed, first to define component labels (searching immediate neighbours) and then, to flatten the equivalence table as component regions coalesce. Both 6 and 26 neighbour versions of the algorithm are implemented. Finally, the largest component is retrieved and all other component labelled regions are removed (by setting negative status).

At the end of the remodelling step, a new model file is written to the directory of the next iteration cycle.

4.5 Discussion

The process outlined above could be streamlined, particularly with regard to (re-)reading input files. Ideally, the control graph would remain resident throughout, checking periodically for the generation of appropriate files, perhaps, via notifications from the solver through a socket interface. The current scheme, however, has the advantage of simplicity and ease of debugging as all intermediate files are available.

Although use of the connected components algorithm is advised to remove separated bone 'islands', the solver has so far proved to be robust to models with small numbers of disconnected elements. Possibly, this is because such regions are generally well away from regions of high strain, so that the impact is slight. The less severe 26-face check is therefore favoured currently, but it should be noted that only the 6-face connectivity option removes all weakly linked (ie. node or edge only) element connections.

Due to the aforementioned memory usage of the current solver, only models of around 2 million elements have been tested within the remodelling scheme. The wall-clock time for solving the model shown in Fig. 7, lies between 280-370 seconds (on 16 cores), with the remodelling itself requiring less than 10 seconds.

It is considered that the remodelling process presented above serves as a basis for developing a much more powerful scheme, possibly employing METIS to partition the FE mesh much more optimally (in the sense of reducing MPI communications).

To illustrate the remodelling process, several links to YouTube videos showing re-

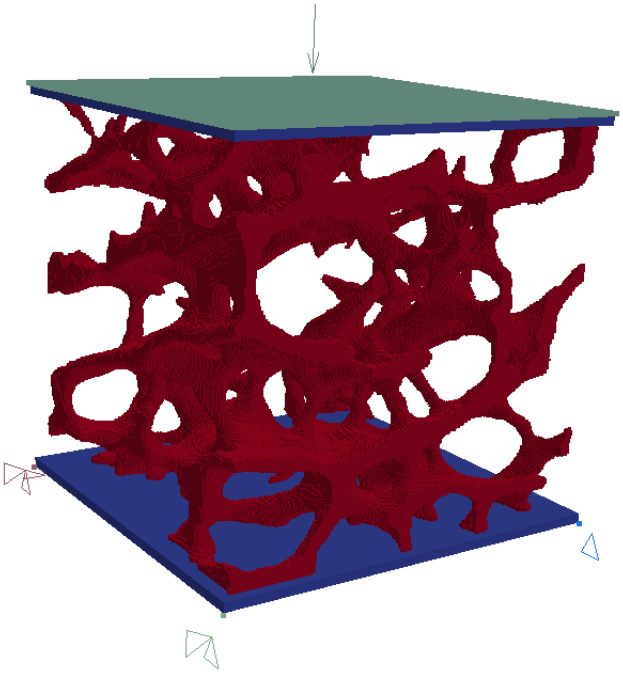

Figure 7: A loaded cube of trabecular bone (data from Biomedtown.org)

modelling of various geometries are given in Appendix A.

5 Summary

We have presented the design and implementation of version 2.0 of VOX-FE, comprising a ParaView plugin GUI, a new solver based on PETSc, and a set of scripts to carry out iterative remodelling. We have verified the correctness and performance of the new version on both desktop hardware and on ARCHER. As a result, VOX-FE2 is now a much more portable, high performance, and extensible tool than the legacy version. The new release is made freely available under a BSD license from http://www.sourceforge.com/p/vox-fe. We have recently obtained further funding from the ARCHER eCSE scheme (project ID 'ecse04-11') to further improve VOX-FE, with the objectives of implementating a new parallelisation scheme in the solver, adding new functionality to the GUI to make VOX-

FE more usable by a wider community including paleobiologists, and further developing the remodelling tools to add adaptive determination of the resorption and formation limits. We expect to release version 2.1 containing these improvements at the end of 2015.

A Appendix: VOX-FE videos

- *'VOXFE' model:* http://youtu.be/fjtjrM1Z1JQ
- *Biomedtown cube:* http://youtu.be/WVp1u1jlD3g
- *'O' model:* http://youtu.be/nT4xWlcm7aQ
- *Tutorial for GUI plugin:* http://youtu.be/DDeAxaZnE8U

References

[1] N. Johnson and I. Bethune. Adding Parallel I/O to PARA-BMU. http://www.hector.ac.uk/cse/distributedcse/reports/voxfe/voxfe.pdf, 2012.

[2] Utkarsh Ayachit. *The ParaView Guide: A Parallel Visualization Application*. Kitware, 2015. ISBN 978-1930934306.

[3] I. Bethune. Using RSIP Networking with Parallel Applications on ARCHER Phase 2, 2015.

[4] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, and Hong Zhang. PETSc Web page. http://www.mcs.anl.gov/petsc, 2014.

[5] ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview.

[6] M. L. Bouxsein, S. K. Boyd, B. A. Christiansen, R. E. Guldberg, K. J. Jepsen, and R. Müller. Guidelines for assessment of bone microstructure in rodents using micro-computed tomography. *Journal of Bone and Mineral Research*, 25(7):1468–1486, Jun 7 2010.

[7] George Karypis and Vipin Kumar. A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20:359–392, 1999.