Open-Source Automated Analysis of Pore-Scale Images to Find Interfacial Curvature

Chamitha Gomez[1], Prof. Martin Blunt[1]
[1]Department of Earth Science and Engineering, Imperial College London
cg4118@ic.ac.uk

1) Overview

Within rock samples, the configuration of fluid phases elucidates the pore-scale physics within the rock. Interfacial curvature between two fluid phases is a fundamental property governing immiscible fluid displacement within porous media and is included in the Young-Laplace equation. Accurate automated computation of the properties of the fluid-fluid interface remains an area of intense research. There are existing automated methodologies for measuring curvature, but these commercial software that are designed for alternative industries and use proprietary methodologies which hamper error-analysis and potential improvements. Similarly, there are open-source automated methodologies for analysing contact angle, but these methods are hindered by issues such as contact angle hysteresis and surface heterogeneity. Thus, this study aims to produce an open-source methodology and *C/C++* code for calculation of interfacial curvature in porous media for use on 3-D X-ray datasets in three steps:

- Surface delineation (using the Marching Cubes algorithm) to produce an isosurface mesh from a 3-D micro-CT rasterised dataset
- Surface smoothing to increase the accuracy of the isosurface mesh
- Curvature calculation from surface mesh

The *C/C++* implementations of the marching cubes algorithm and curvature algorithm were very successful, with both algorithms proving to be robust across simple synthetic datasets and complex datasets from real pore-scale images. The smoothing algorithm contained a successful implementation of Laplacian smoothing, whereby vertices within the surface mesh are migrated to the average of their neighboring vertices. This approach produced errors of <2% for fine resolutions (where the voxel size to radius of curvature ratio is less than 0.15) and <1.5% for coarse resolutions (where the voxel size to radius of curvature ratio is greater than 0.15). However, the Gaussian smoothing algorithm, which attempts to minimise some common errors with the Laplacian smoothing algorithm by migrating vertices back towards their original positions, was less successful. Consequently, writing an improved Gaussian smoothing algorithm is the primary area that future work within this field should focus on

2) Methodology

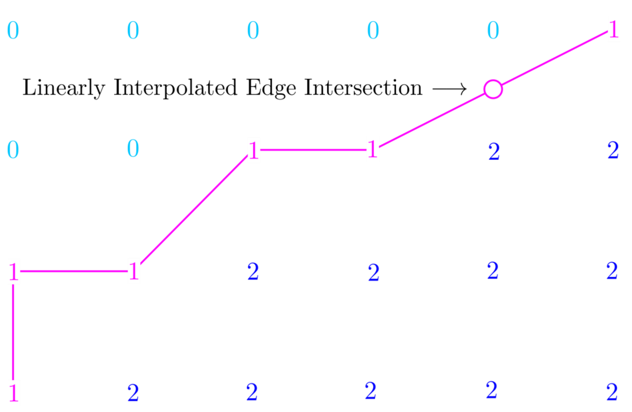
2.1) Surface Delineation

The first algorithm uses two look-up tables and two binary indices to detect how a surface with a set value (an isosurface and isovalue, respectively) passes through the volume that is mapped by the dataset. It produces triangles that are connected to form a mesh that models the isosurface. The loops are shown in Algorithm 1, but the main steps are:

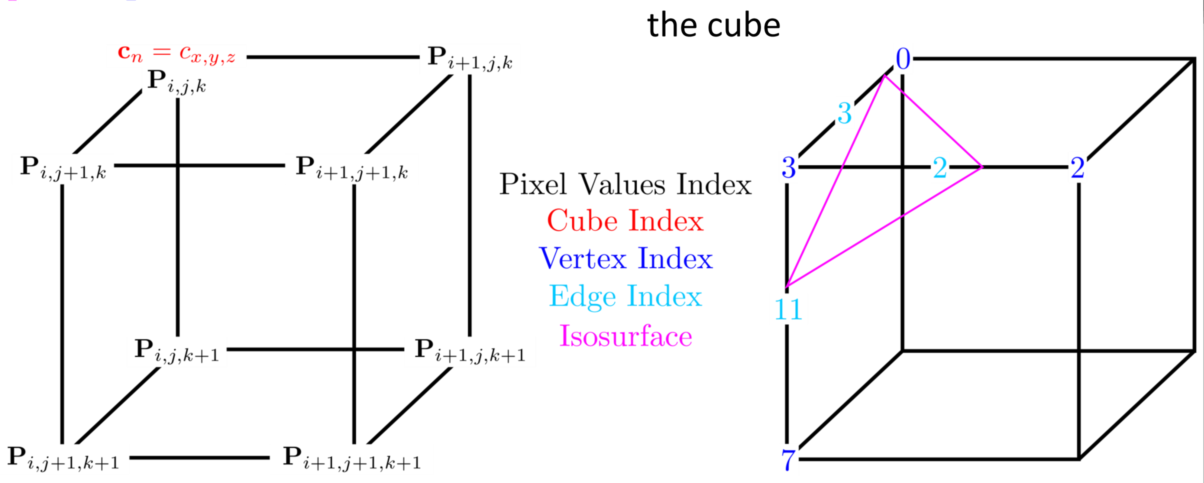
1) Read in the data (**P**) and split it up into a series of cubes, with numbered vertices and edges
2) Determine which vertices of each cube are above/below the isovalue
3) If the surface contains vertices that are above and below the isovalue, the surface passes through the cube. Linearly interpolate the position of the surface along each cube edge the surface bisects
4) Store the coordinates of the mesh (in matrix **I**) and store which coordinates are connected to form a triangle (in matrix **T**).

The declaration of the binary *cube-vertex index* and *cube-edge index* for the n^{th} cube is:

$$\forall \mathbf{c}_n \in (C \subsetneq \mathbb{N}_1^3)\ \exists! \psi_n, \omega_n$$



Left: The coordinates for the vertices of the triangle are calculated through linear interpolation when required. Below: An example of a triangle generated when vertex 3 is below the isovalue but all other vertices are above it, or vice versa, and the coordinate location of vertices and the cube



Algorithm 1: Marching Cubes

Input: Matrix **P** representing layers of rasters from the micro-CT image
Output: Matrix of isosurface coordinates (**I**) and matrix of triangles (**T**)

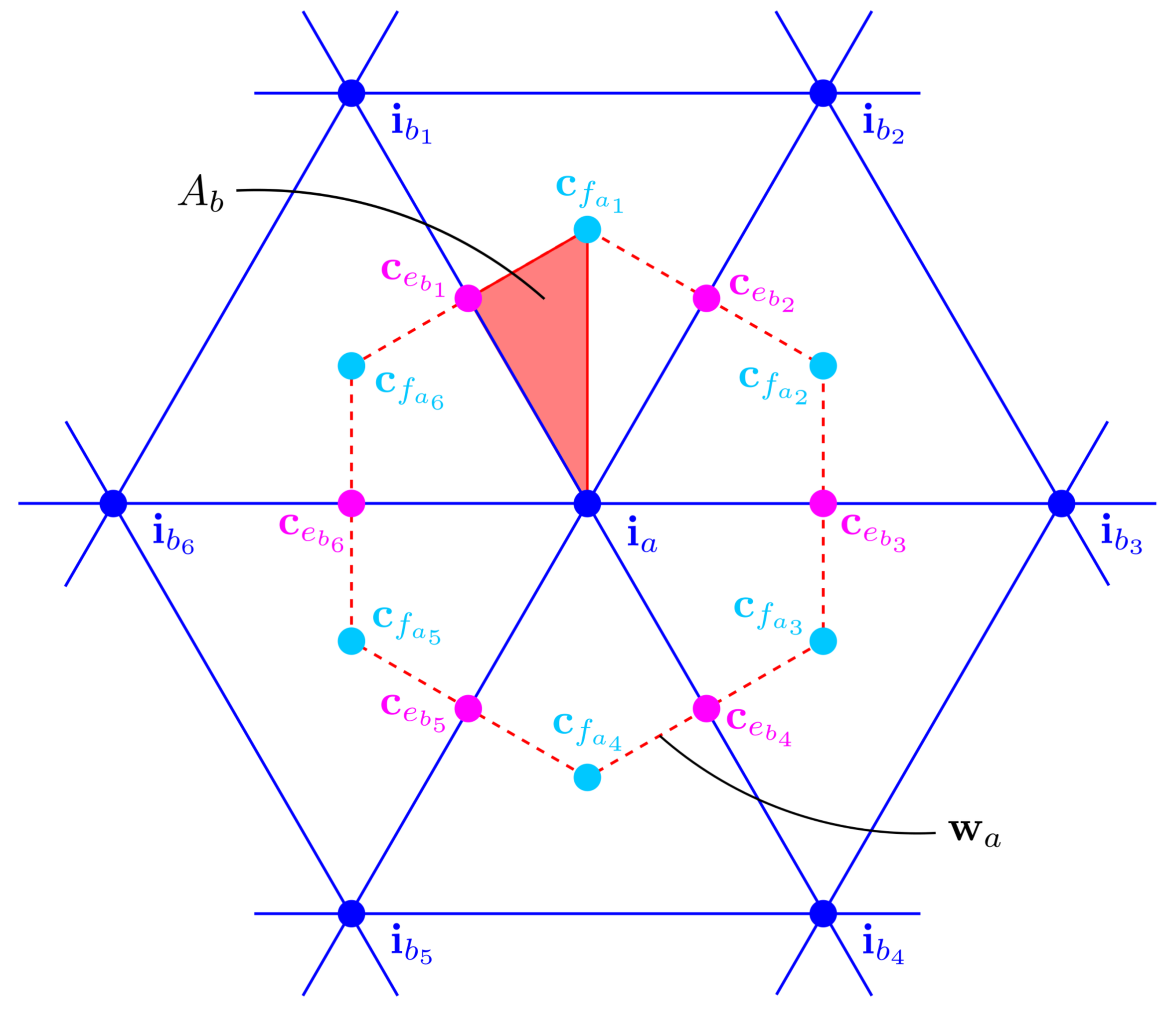
for each cube {
  Parse data from **P** into the n^{th} cube from the set of cubes ($\forall \mathbf{c}_n \in C$).
  Determine 8-bit binary *cube-vertex index* (ψ_n).
  Determine 12-bit binary *cube-edge index* (ω_n) from Vertex-Edge lookup table using *cube-vertex index*
  if (*cube-vertex index* $= 0$) **{** //binary false ('0') shows isosurface is not inside cube
    continue //move to next cube
  }
  else { //isosurface passes through cube
    /* If an edge is bisected, calculate interpolated position */
    for each edge {
      if (edge is marked as bisected in *cube-edge index*) **{** //binary true ('1') used for marking
        Calculate interpolated position along edge and save to **I**
      }
    }
  }
  Cube-edge index is passed to the Edge-Triangle lookup table which returns a list of edges (**t**) that form each triangle that is required to model the isosurface (up to 5 triangles required) in the cube.
  for ($m \in \{0, 3, 6, 9, 12, 15\}$) **{** //for every third index of **t**, with indexing starting at 0
    if ($\mathbf{t}_m = -1$) **{** //no subsequent triangles
      continue //move to next cube
    }
    else { //there are subsequent triangles
      Add the triangle's three interpolated positions to **T**, which are stored in $\mathbf{I}_{\mathbf{t}_m}$, $\mathbf{I}_{\mathbf{t}_{m+1}}$ and $\mathbf{I}_{\mathbf{t}_{m+2}}$.
    }
  }
}

2.2) Isosurface Smoothing

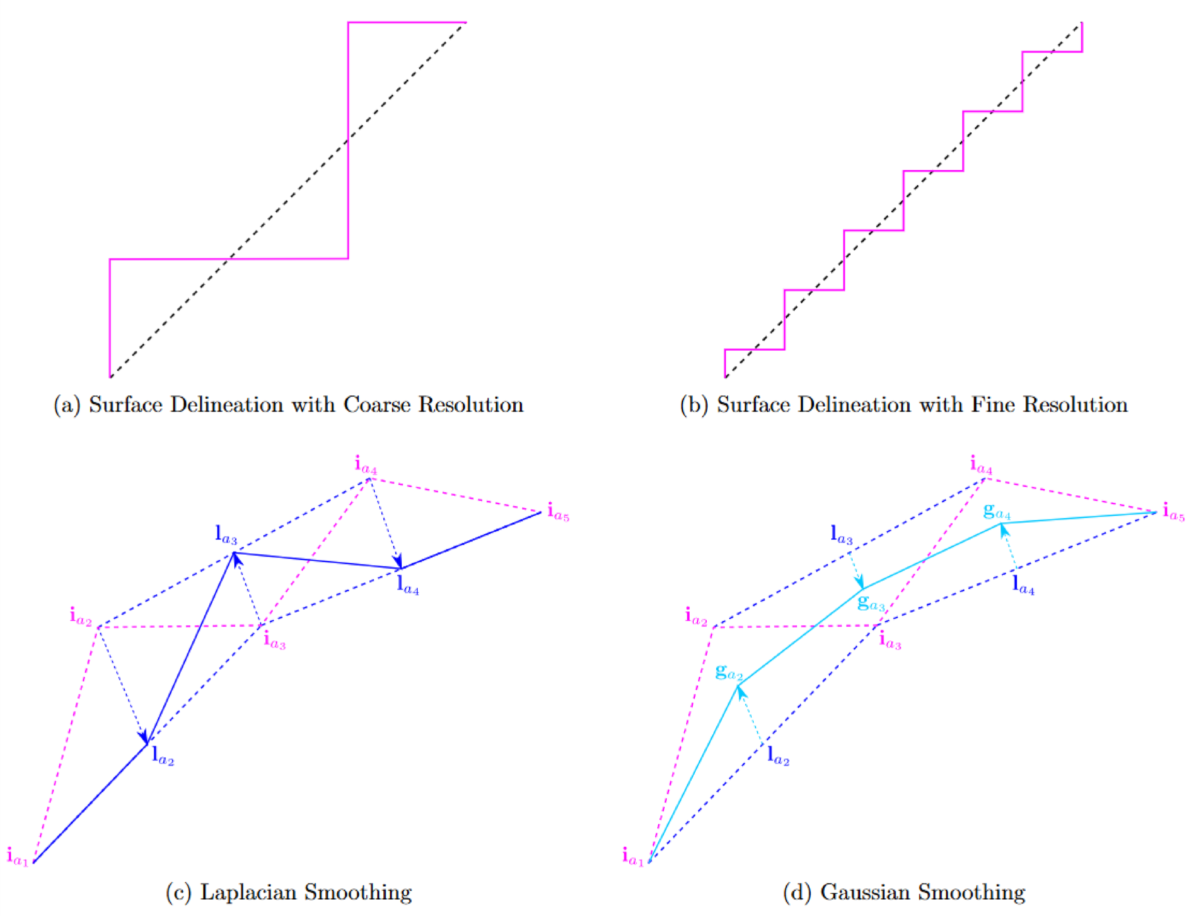
The second algorithm aims to migrate the vertices of the triangular mesh using two types of smoothing:

- Laplacian smoothing migrates the vertices to an average of their adjacent points to improve smoothness
- Gaussian smoothing migrates vertices back towards their old positions to help preserve fluid volume,

The parameters retrieved from the triangular mesh, and two calculated parameters, are shown below:



A schematic showing how increased resolution doesn't necessarily improve accuracy, and how the two types of smoothing work, is shown below. The smoothing algorithm is shown underneath.



Algorithm 2: Surface Smoothing

Input: Matrices **I** and **T**, representing the output of the marching cubes algorithm
Output: Updated matrix **I** containing smoothed isosurface vertices

/* Loop for each Gaussian iteration */
for ($x \in \{1, ..., iterations_G\} \subsetneq \mathbb{N}_1$) **{**
  /* Loop for each Laplacian iteration */
  for ($y \in \{1, ..., iterations_L\} \subsetneq \mathbb{N}_1$) **{**
    for all ($\mathbf{i}_a \in \mathbf{I}$) **{**
      if ($adj(\mathbf{i}_a) = 0$) **{** //no adjacent points
        continue //move to next coordinate
      }
      else { //there are adjacent vertices
        /* Calculate preliminary variables */
        for all ($e_b \in E$) **{**
          $\mathbf{c}_{e_b} = \frac{1}{2}(\mathbf{i}_a + \mathbf{i}_b), \quad \mathbf{i}_a, \mathbf{i}_b \in e_b$ //calculate edge centres
        }
        for all ($f_a \in F$) **{**
          $\mathbf{c}_{f_a} = \frac{1}{n_{\mathbf{i}_b}} \sum_{b \in f_a} \mathbf{i}_b, \quad \mathbf{i}_b \in f_a$ //calculate face centres
          $A_b = \frac{1}{2}\left(\overrightarrow{\mathbf{i}_a \mathbf{c}_{f_a}} \times \overrightarrow{\mathbf{c}_{e_b} \mathbf{c}_{f_a}}\right)$ //calculate triangular areas
        }
        $\mathbf{w}_a = \sum_{b \in adj(\mathbf{i}_a)} A_b$ //calculate vertex weighting
        $\mathbf{n}_a = \frac{\mathbf{w}_a}{|\mathbf{w}_a|}$ //calculate vector normal
        $\alpha_b = |\mathbf{w}_{\mathbf{o}_j}| + 0.3|\mathbf{w}_{\mathbf{o}_a}|$ //calculate weight factor
        $\mathbf{d}_a = \frac{\sum_{b \in adj(\mathbf{i}_a)} \alpha_b \mathbf{o}_b}{\sum_{b \in adj(\mathbf{i}_a)} \alpha_b} - \mathbf{o}_a$ //calculate displacement to average position of adjacent vertices
        /* Migrate vertices using Laplacian smoothing to augment smoothness of isosurface model */
        $\mathbf{l}_a = \mathbf{o}_a + 0.8\beta(\mathbf{d}_a \cdot \mathbf{n}_a)\mathbf{n}_a + 0.2\beta\mathbf{d}_a$ //$\mathbf{o}_a \mapsto \mathbf{l}_a$
      }
    }
    $\mathbf{s}_a = \frac{\sum_{b \in adj(\mathbf{i}_a)} \alpha_b(\mathbf{q}_b - \mathbf{o}_b)}{\sum_{b \in adj(\mathbf{i}_a)} \alpha_b}$ //calculate average displacement during Laplacian smoothing
  }
  /* Migrate vertices back towards initial positions using Gaussian smoothing to preserve volume */
  $\mathbf{g}_a = \mathbf{l}_a - (0.3(\mathbf{s}_a + (\mathbf{s}_a \cdot \mathbf{n}_a)\mathbf{n}_a))$ //$\mathbf{l}_a \mapsto \mathbf{g}_a$
}

2.3) Curvature Calculation

The third algorithm calculates curvature from a (smoothed) triangular mesh using triangle barycentres and contains three sub-algorithms:

- Project Curvature Tensor: Alter curvature tensor coordinate system
- Rotate Coordinate System: Alter vertex coordinate system
- Diagonalise Curvature Tensor: Ensures desired eigenvalue and eigenvector calculation to yield principal curvatures and directions.

The curvature calculation focuses on the expression of normal curvature (κ_n) using the second fundamental tensor (SFT, $\mathbf{II}$), and placing restrictions on the SFT to solve using a least squares solver.

$$\kappa_n = \begin{pmatrix} \gamma & \varepsilon \end{pmatrix} \begin{bmatrix} e & f \\ f & g \end{bmatrix} \begin{pmatrix} \gamma \\ \varepsilon \end{pmatrix} = \begin{pmatrix} \gamma & \varepsilon \end{pmatrix} \mathbf{II} \begin{pmatrix} \gamma \\ \varepsilon \end{pmatrix}$$

The SFT can be diagonalised to express normal curvature in terms of two principal curvatures and principal directions, γ_p and ε_p:

$$\kappa_n = \begin{pmatrix} \gamma_p & \varepsilon_p \end{pmatrix} \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} \begin{pmatrix} \gamma_p \\ \varepsilon_p \end{pmatrix} = \kappa_1 \gamma_p^2 + \kappa_2 \varepsilon_p^2$$

The SFT for a smooth surface can be expressed in terms of the direction derivatives of the surface normal using an orthonormal coordinate system in the tangent plane ($\mathbf{u}', \mathbf{v}'$):

$$\mathbf{II} = [\nabla_{\mathbf{u}'} \mathbf{n}_s \ \nabla_{\mathbf{v}'} \mathbf{n}_s] = \begin{bmatrix} \frac{\partial \mathbf{n}_s}{\partial \mathbf{u}'} \cdot \mathbf{u}' & \frac{\partial \mathbf{n}_s}{\partial \mathbf{v}'} \cdot \mathbf{u}' \\ \frac{\partial \mathbf{n}_s}{\partial \mathbf{u}'} \cdot \mathbf{v}' & \frac{\partial \mathbf{n}_s}{\partial \mathbf{v}'} \cdot \mathbf{v}' \end{bmatrix}$$

The final steps of the curvature calculation algorithm are below:

Algorithm 2: Final Part of Curvature Calculation

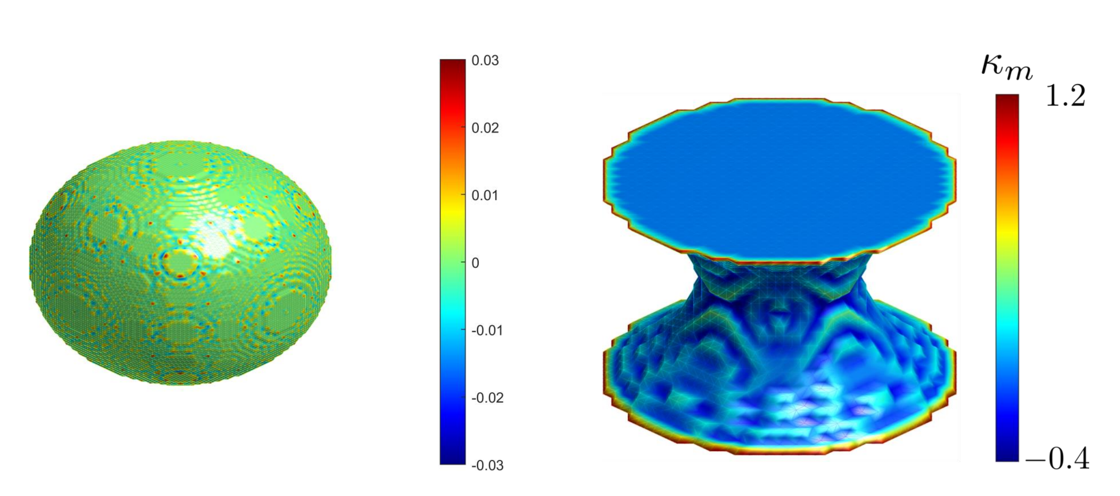
Input: Matrices **I** and **T**, representing a surface mesh
Output: Principal curvatures κ_1 and κ_2

/* Calculate Curvature (continued...) */
for all ($t_x \in \mathbf{T}$) **{**
  //Create least squares problem of form $\mathbf{Ax} = \mathbf{b}$ to solve for **x**, which contains the elements of $\mathbf{II}$
  $\mathbf{u}_{t_x} = \hat{\mathbf{e}}_{x_1}$ //set direction of $\mathbf{u}_{t_x}$
  $\mathbf{v}_{t_x} = \frac{\hat{\mathbf{n}}_{t_x} \times \mathbf{u}_{t_x}}{|\hat{\mathbf{n}}_{t_x} \times \mathbf{u}_{t_x}|}$ //set direction of $\mathbf{v}_{t_x}$
  $\mathbf{A} = \begin{bmatrix} \mathbf{e}_{x_1} \cdot \mathbf{u}_{t_x} & \mathbf{e}_{x_1} \cdot \mathbf{v}_{t_x} & 0 \\ 0 & \mathbf{e}_{x_1} \cdot \mathbf{u}_{t_x} & \mathbf{e}_{x_1} \cdot \mathbf{v}_{t_x} \\ \mathbf{e}_{x_2} \cdot \mathbf{u}_{t_x} & \mathbf{e}_{x_2} \cdot \mathbf{v}_{t_x} & 0 \\ 0 & \mathbf{e}_{x_2} \cdot \mathbf{u}_{t_x} & \mathbf{e}_{x_2} \cdot \mathbf{v}_{t_x} \\ \mathbf{e}_{x_3} \cdot \mathbf{u}_{t_x} & \mathbf{e}_{x_3} \cdot \mathbf{v}_{t_x} & 0 \\ 0 & \mathbf{e}_{x_3} \cdot \mathbf{u}_{t_x} & \mathbf{e}_{x_3} \cdot \mathbf{v}_{t_x} \end{bmatrix}$ //Define **A**
  $\mathbf{b} = \begin{bmatrix} (\hat{\mathbf{n}}_{\mathbf{i}_{x_3}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_2}}) \cdot \mathbf{u}_{t_x} \\ (\hat{\mathbf{n}}_{\mathbf{i}_{x_3}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_2}}) \cdot \mathbf{v}_{t_x} \\ (\hat{\mathbf{n}}_{\mathbf{i}_{x_1}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_3}}) \cdot \mathbf{u}_{t_x} \\ (\hat{\mathbf{n}}_{\mathbf{i}_{x_1}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_3}}) \cdot \mathbf{v}_{t_x} \\ (\hat{\mathbf{n}}_{\mathbf{i}_{x_2}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_1}}) \cdot \mathbf{u}_{t_x} \\ (\hat{\mathbf{n}}_{\mathbf{i}_{x_2}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_1}}) \cdot \mathbf{v}_{t_x} \end{bmatrix}$ //Define **b**
  Solve for **x**. //uses GSL solver
  $\mathbf{II}_{t_x} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 \\ \mathbf{x}_2 & \mathbf{x}_3 \end{bmatrix}$ //set values for the triangle's second fundamental tensor
  $\kappa_{n_{t_x}} = \mathbf{x}_1$ //optional - not used further on
  //Calculate curvature per vertex
  for all ($\mathbf{i}_{x_y} \in t_x$) **{**
    $V_{\mathbf{i}_{x_y}} = \frac{\xi_{\mathbf{i}_{x_y}}}{V_{\mathbf{i}_a}}$ //calculate corner Voronoi weights
    Project curvature tensor using sub algorithms 'Project Curvature Tensor' and 'Rotate Coordinate System' to produce $\mathbf{II}_{\mathbf{i}_{x_y}}$.
    $\mathbf{II}_{\mathbf{i}_a} = \mathbf{II}_{\mathbf{i}_a} + V_{\mathbf{i}_{x_y}} \mathbf{II}_{\mathbf{i}_{x_y}}$ //Multiply corner curvature tensor with corner weights and sum for each use of the vertex as a face corner to produce the vertex curvature tensor
  }
}
for all ($\mathbf{i}_a \in \mathbf{I}$) **{**
  Retrieve principal curvatures (κ_1 and κ_2) and principal directions (γ_p and ε_p) for each vertex using sub-algorithm 'Diagonalise Curvature Tensor'.
}

3) Results

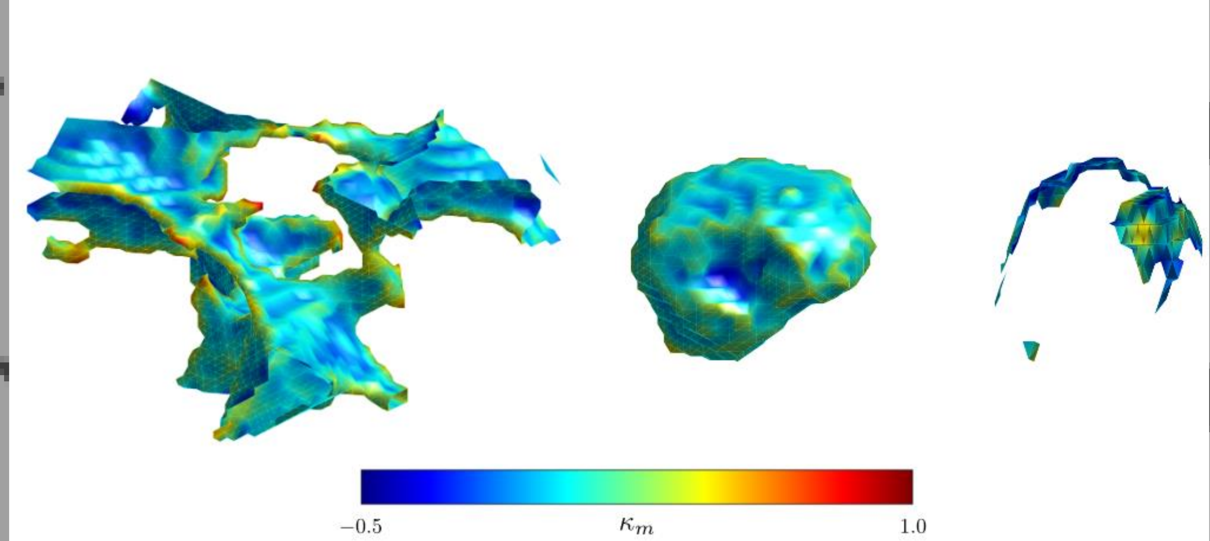
3.1) Analytical Surfaces

The analytical surfaces chosen for this study were the sphere and the catenoid, as both these shapes have exact mean and Gaussian curvatures. Laplacian smoothing was sufficient for accurate calculation of sphere curvature, with results far exceeding competing algorithms (see overview for exact results). However, when catenoids were used to test the Gaussian smoothing section of the algorithm, curvatures failed to tend to their expected values due to a phenomenon known as oversmoothing. This is the principal area of this study which needs further development. The initial calculated Gaussian curvature of a sphere and mean curvature of a catenoid are shown below, prior to any smoothing:

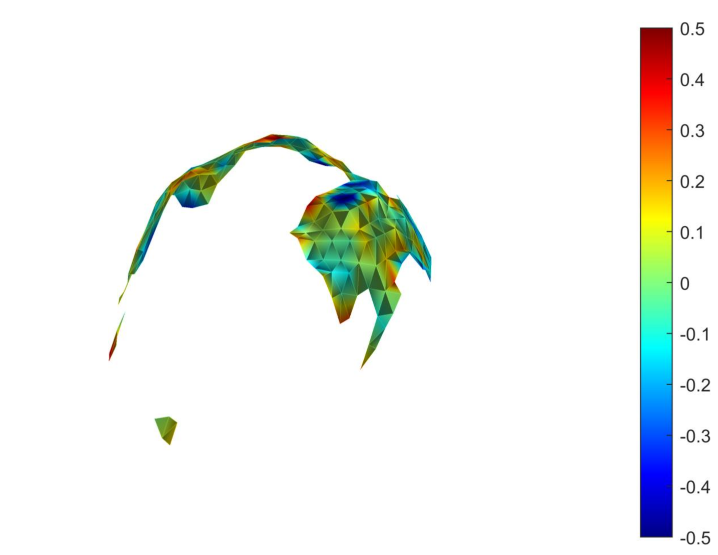


3.2) Sandstone Micro-CT Datasets

Due to the inaccuracy of the Gaussian smoothing algorithm, the Laplacian smoothing algorithm was chosen for use on real datasets. An example of the curvature calculated from an unsmoothed water-wet sample of Bentheimer sandstone is shown below, where the brine phase is on the left, and the oil phase in the centre and the oil-water on the right:



The oil-water interface shown below is after 8 iterations of Laplacian smoothing. This was successful in altering the mean curvature from negative to positive (as expected in a water-wet system).



4) Conclusions

The *C/C++* implementations of the marching cubes algorithm and curvature calculation algorithm were extremely successful. The Laplacian smoothing algorithm worked well but is prone to producing inaccurate results and so an improved Gaussian smoothing algorithm is required to improve the accuracy of calculated curvatures for real micro-CT images of porous media.

5) Key References

- Akai, T., Lin, Q., Alhosani, A., Bijeljic, B. & Blunt, M. J. (2019), 'Quantification of uncertainty and best practice in computing interfacial curvature from complex pore space images', Materials 12(13), 2138.
- AlRatrout, A., Raeini, A. Q., Bijeljic, B. & Blunt, M. J. (2017), 'Automatic measurement of contact angle in pore-space images', Advances in Water Resources 109, 158–169.
- Rusinkiewicz, S. (2004), Estimating curvatures and their derivatives on triangle meshes, in 'Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004.', IEEE, pp. 486–493.