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

Chamitha Gomez

January 2022

Supervisor: Martin Blunt

This report is submitted as part requirement for the MSci Degree in Geophysics at Imperial College London. It is substantially the result of my own work except where explicitly indicated in the text.

This MSci project was conducted between September 2021 and January 2022, and encompassed research funded by a 6-week UROP bursary in July and August 2021.

The main body of this dissertation has 5,489 words.

Contents

List of Algorithms

Abstract

The analysis of fluid displacement within porous media has a wide range of potential applications, including the construction of personal protective equipment to limit the spread of COVID-19 and attenuating anthropogenic climate change through the use of carbon dioxide capture and storage. Within rock samples, the configuration of fluid phases elucidates the pore-scale physics within the rock. Accurate automated computation of the properties of the fluid-fluid interface remains an area of intense research.

Although there have been many recent studies that have automated the calculation of the contact angle between the fluid-fluid interface and the solid rock wall, there are few which automate the calculation of interfacial curvature. Successful automated calculation of curvature will provide further insight into flow in porous media by avoiding issues with contact angle analysis such as hysteresis and surface heterogeneity.

This study produces a methodology for the automated calculation of interfacial curvature from pore-scale images which comprises three principal sections: firstly, an isosurface is delineated using the marching cubes algorithm; secondly, the surface mesh is smoothed to produce a more accurate model of the fluid-fluid interface; thirdly, the curvature of the mesh is calculated and analysed. The methodology is open-source to promote independent error analysis and provide potential for improvement in future work.

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 neighbouring 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.

1 Introduction

The study of fluid dynamics within porous media has numerous applications across a variety of fields, ranging from reducing the spread of COVID-19 through the use of face masks (Dbouk & Drikakis 2020, Mittal et al. 2020) to attenuating anthropogenic climate change through carbon dioxide capture and storage (CCS) (Boot-Handford et al. 2014, Börner et al. 2013). The interfacial curvature of the meniscus between two fluid phases is a fundamental property governing immiscible fluid displacement within porous media, as demonstrated by its inclusion in the Young-Laplace equation (Young 1805, Laplace 1805, Blunt 2017). However, current automated curvature measurements often rely on commercial image analysis software (Iltis et al. 2011, Scanziani et al. 2017, Akai et al. 2019). These softwares are often designed for use in specific industries and use proprietary methodologies, hampering independent error-analysis and potential improvements. This study aims to produce an open-source methodology for accurate and robust surface delineation, isosurface smoothing and computation of interfacial curvature in porous media, which is tested on artificial datasets with analytical solutions and subsequently tested on three-dimensional X-ray datasets of sandstones.

The marching cubes algorithm from Lorensen & Cline (1987) is one of the most widespread surface delineation algorithms and has formed the basis of numerous studies which have attempted to resolve some of its limitations, such as the marching tetrahedra and dual contouring adaptations (Ju et al. 2002, Lu & Chen 2012, Lopes & Brodlie 2003). However, the simplicity of the original algorithm and its low computational cost due to the use of lookup tables have meant that it has remained more popular than other algorithms, and so is used in this study (Newman & Yi 2006).

Smoothing a delineated isosurface to improve its geometric properties so it more accurately represents the interface it models is ubiquitous in the literature (Taubin 1995, Jones et al. 2003, Shen & Barner 2004). Laplacian smoothing relocates the vertices within a discretised surface to new positions based on an average of the location of neighbouring vertices (Field 1988, Ohtake et al. 2001). However, there are several limitations of this method, most notably shrinkage and feature loss (Liu et al. 2001, Desbrun et al. 1999, Ji et al. 2005), which are areas of ongoing research (Wei et al. 2013, Liu et al. 2017, Xi et al. 2021). This study uses the robust approach of Vollmer et al. (1999), which reduces shrinkage and feature loss from the Laplacian smoothing step by adding a Gaussian smoothing step to migrate the vertices back towards their original positions.

As curvature is a continuous variable across a surface, it must be approximated over a piece-wise linear mesh (Razdan & Bae 2005, Dong & Wang 2005). There are various techniques available for computing curvature from triangular meshes, such as normal curvature estimation (Chen & Schmitt 1992, Theisel et al. 2004) and patch-fitting approaches (Cazals & Pouget 2005, Huang et al. 2005). Curvature calculation in this study uses the method of Rusinkiewicz (2004), which places linear constraints on the curvature tensor and then solves using least-squares, which has been shown to produce accurate results on micro-CT datasets (Shabat & Fischer 2015) and produce exact curvatures for analytical surfaces (Rusinkiewicz 2004).

Non-destructive X-ray micro-computed tomography (micro-CT), which utilises X-rays to image materials with micron resolution in three dimensions, such as a rock sample and the fluids within it, is now routinely used for many earth science applications (Wildenschild et al. 2002, Arns et al. 2005). Nanometre resolution is available using alternative techniques, although these methods are often destructive, time-consuming or limited to two-dimensions and may not be necessary when investigating pore-scale processes (which usually operate at the 10s of microns scale) (Blunt et al. 2013, Wildenschild & Sheppard 2013). The advent of micro-CT imaging has revolutionised our understanding of flow in porous media by providing unprecedented, laboratory-reproducible insight into the displacement processes. For slow, capillary dominated flow, which is typical in

almost all subsurface applications, interfacial forces are the most prevalent (Hazlett 1995, Blunt 2017). To enhance our understanding of the fluids' configurations due to these interfacial forces, micro-CT images have been used to measure the contact angle between fluid phases manually (Andrew et al. 2014, Khishvand et al. 2016). As contact angle can be related to the interfacial tension using the Young equation, calculation of curvature is extremely important as when combined with the interfacial tension it can be used to calculate capillary pressure (Young 1805, Blunt 2017).

There is a plethora of recent research on the automatic calculation of contact-angles within porous media (Klise et al. 2016, AlRatrout et al. 2017, Scanziani et al. 2017, Ibekwe et al. 2020, Yang & Zhou 2020, Zankoor et al. 2021). However, characterisation of interfaces through the contact angle along a three-phase contact line could produce large errors due to contact-angle hysteresis and surface heterogeneity (Blunt 2017, Klise et al. 2016, Zankoor et al. 2021). As a result, the extension of these studies to the automated calculation of curvature, which considers the entire delineated surface, has the potential to circumvent these problems and so improve the accuracy of fluid displacement models within porous media.

In this study, *C/C++* implementations of the marching cubes algorithm, Gaussian smoothing algorithm and curvature calculation algorithms were written to delineate, manipulate and determine the properties of an isosurface within a variety of datasets. The algorithms were initially tested on analytical surfaces to identify areas of potential improvement within the methodology. Once they were modified, the algorithms were tested on a series of datasets generated from micro-CT images of sandstones and compared to results in the literature that were generated using alternative methods. Thus, the accuracy of this study's methodology will be tested using both synthetic and real datasets, with the aim of elucidating the pore-scale physics that occur within porous media.

2 Methodology

Calculating curvature from pore-scale images requires two fundamental steps: the interface between the fluids must be delineated from the micro-CT scan and then the curvature can be calculated from this interface. Initially, the 'Lego-brick' style segmented image is analysed to locate the faces where a phase change occurs. This is achieved by setting the isovalue to a fluid's value within the image - as soon as this value is encountered, this marks the beginning of the phase so this location forms part of the fluid's surface. When these interfaces are combined they form an isosurface, which is the surface that connects all points which have a value equal to the isovalue. Once the surface has been detected, the curvature of the mesh that models the surface can be calculated. An additional step between surface delineation and curvature calculation is to smooth the isosurface so it will represent the topology of the fluids more realistically. These three primary steps were implemented using *C/C++* algorithms, which were written for this study to facilitate precise error analysis and are open-source to provide the possibility of enhancement in the future.

2.1 Isosurface Delineation

Voxel image data is initially read into the pixel values matrix $\mathbf{P}$, whose points ($\mathbf{P}_{i,j,k} \in \mathbf{P}$) have their indices stored in the set $P \subsetneq \mathbb{N}_1^3$ and their values (a measure of brightness) stored in the set $B \subsetneq \mathbb{Q}^+$. An isosurface passing through the dataset is delineated using the marching cubes algorithm, modelling how the points that form the interface are connected throughout the volume mapped by P (Lorensen & Cline 1987). This approach involves generating a series of triangles which connect the isosurface coordinates that approximate the position of the isosurface as it passes through the dataset, and is outlined in Algorithm 1.

A new set is defined which contains all the cube indices, $C \subsetneq \mathbb{N}_1^3$. Cubes are indexed by their top-left vertex on their upper layer (see Figure 1). To ensure that all the cubes are inside the dataset, the indices within C do not include indices that correspond to the right column or bottom row of any layer in $\mathbf{P}$, nor the lower layer of $\mathbf{P}$. Thus, for the indices that correspond to the bottom-right point or cube on the lower layer of the dataset, $\mathbf{p}_{i_{max},j_{max},k_{max}} \in P$ and $c_{x_{max},y_{max},z_{max}} \in C$:

$$\begin{aligned} x_{max} &= i_{max} - 1, \\ y_{max} &= j_{max} - 1, \\ z_{max} &= k_{max} - 1, \end{aligned} \tag{1}$$

$$\therefore C \subsetneq P. \tag{2}$$

To detect whether the isosurface passes through the volume of each cube, every cube is analysed in isolation. For the n^{th} cube ($c_{x,y,z} = \mathbf{c}_n \in C$) that is parsed from $\mathbf{P}$, two new sets, P_n and B_n, are created. P_n contains the coordinates of each of the eight vertices of the cube ($\mathbf{p}_{1,\dots,8} \in P_n \subset P$). B_n contains the values of $\mathbf{P}$ at each vertex of the cube ($v_{1,\dots,8} \in B_n \; for \; \mathbf{p}_{1,\dots,8} \in P_n$).

The marching cubes algorithm begins by comparing the value of each vertex of the cube to the isovalue that defines the isosurface ($isovalue = v_{isosurface}$). If all the values are above or below the isovalue ($\forall v \in B_n, v > v_{isosurface} \vee v < v_{isosurface}$), the cube is entirely inside or outside the volume of the fluid being detected and the isosurface does not pass through the cube. However, if some of the values are above the isovalue whilst others are below it, the isosurface must pass through the volume of the cube (see Figure 2 and Appendix A.1).

The index of each vertex within the isosurface is stored in an 8-bit number, called the 'cube-vertex index'

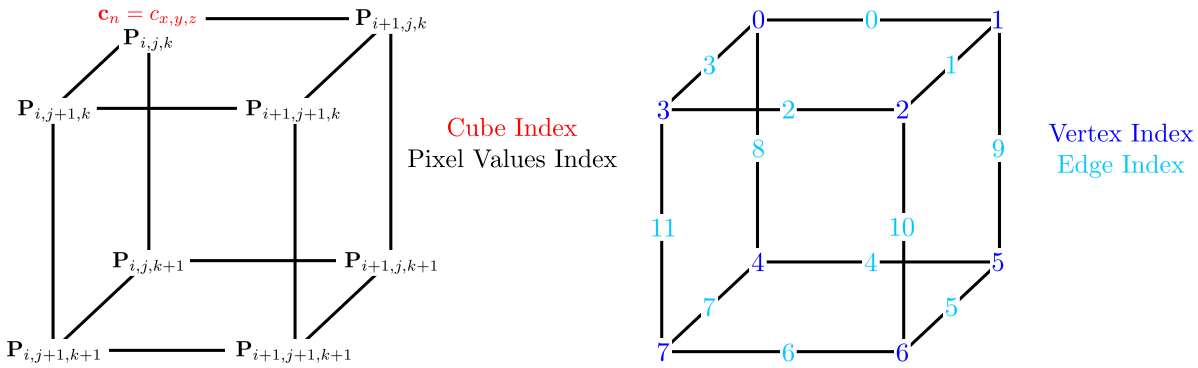


Figure 1: Indexing convention used to parse a single cube from the 3-D pixel values matrix and how this corresponds to the vertices' and edges' indices of a cube in the marching cubes algorithm. Vertex 0 and coordinate $P_{i,j,k}$ are in the upper-left corner of the top layer of the cube.

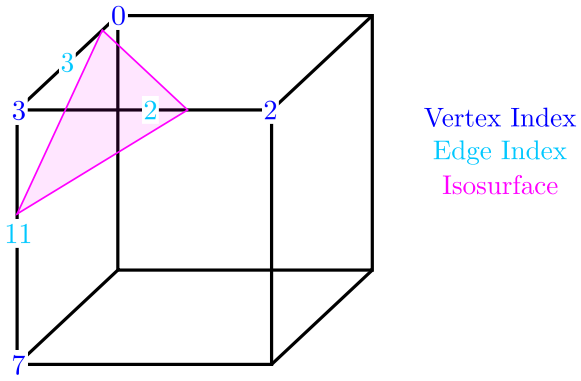


Figure 2: An example of a triangle generated when vertex 3 is below the isovalue but all other vertices are above it, or vice versa. The coordinates for the vertices of the triangle are calculated through linear interpolation. For further examples where several triangles are used within one cube, see Appendix A.1.

($\forall \mathbf{c}_n \in C \; \exists! [1] \psi_n$), with a binary true for every vertex within the fluid volume and binary false for every vertex outside the fluid. The cube-vertex index is passed to the 'Vertex-Edge' lookup table (Appendix A.2), which returns the 12-bit 'cube-edge index' ($\forall \mathbf{c}_n \in C \; \exists! \omega_n$). This stores the indices of each edge that is bisected by the isosurface, with a binary true for every edge that is bisected by the isosurface and a binary false for every edge which the isosurface does not pass through. ψ_n is passed to the 'Edge-Triangle' lookup table (Appendix A.3), which returns the 'triangles' vector ($\mathbf{t}$) that lists the edges that form each triangle that is required to replicate the shape of the isosurface as it passes through the cube. When there are no more triangles required, every subsequent element of $\mathbf{t}$ is set to the termination character -1. As there are a maximum of five triangles required to represent a combination of vertices that are within the fluid volume, $\mathbf{t}$ has a length of $(5 \times 3) + 1 = 16$.

As the 'triangles' list only returns the edges that are needed to model the isosurface, but not the coordinates of the isosurface along each edge, linear interpolation is used to determine the exact position where the isosurface bisects each edge of the cube. For an edge between two points in the pixel values matrix ($\mathbf{P}_1, \mathbf{P}_2 \in \mathbf{P}$), the interpolated position is calculated as:

$$\mathbf{p}_{interpolated} = \mathbf{p}_1 + \frac{v_{isosurface} - v_1}{v_2 - v_1}(\mathbf{p}_2 - \mathbf{p}_1). \tag{3}$$

For each cube, the coordinates of the isosurface are stored in a matrix $\mathbf{I}$ and the triangles that connect these coordinates are stored in a matrix $\mathbf{T}$. Once all the cubes have been iterated over, these two matrices will contain a model of how an isosurface with $v_{isosurface} = isovalue$ passes through the volume of the dataset.

[1] '$\exists!$' can be read as '*there exists one and only one*'

Algorithm 1: Marching Cubes

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

for all ($\mathbf{c}_n \in C$) **{**
  Parse data from $\mathbf{P}$ to P_n and B_n.
  Determine ψ_n using B_n.
  ω_n = Vertex-Edge[ψ_n]
  if ($\omega_n = 0$) **{** //isosurface is not inside cube
    continue //move to next cube
  }
  else { //isosurface passes through cube
    /* If an edge is bisected, calculate interpolated position */
    for all ($e_m \in \{0, ..., 11\} \subsetneq \mathbb{N}_0$) **{** //for each edge index, with indexing starting at 0
      if (ω_n & 2^m) **{** //& is bitwise 'and'
        $\mathbf{p}_{interpolated} = \mathbf{p}_1 + \frac{v_{isovalue} - v_1}{v_2 - v_1}(\mathbf{p}_2 - \mathbf{p}_1)$ //Calculate $\mathbf{p}_{interpolated}$ and save to $\mathbf{I}$
      }
    }
  }
  $\mathbf{t}$ = Edge-Triangle[ω_n]
  for ($m \in \{0, 3, 6, 9, 12, 15\}$) **{** //for every third index of $\mathbf{t}$, 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 $\mathbf{T}$, which are stored in $\mathbf{I}_{\mathbf{t}_m}$, $\mathbf{I}_{\mathbf{t}_{m+1}}$ and $\mathbf{I}_{\mathbf{t}_{m+2}}$.
    }
  }
}

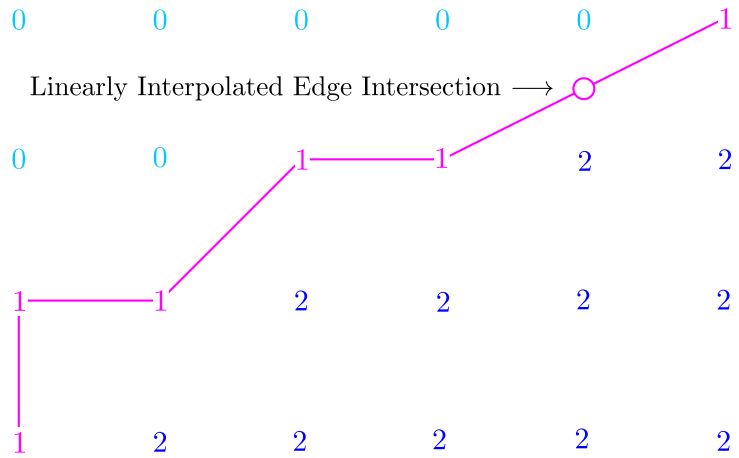


Figure 3: An example of a delineated isosurface without smoothing passing through a 2-D slice of $\mathbf{P}$, where $v_{isovalue} = 1$ and one linearly interpolated edge intersection is calculated. Although the fluids have a smooth surface, voxelisation of the micro-CT scan has lead to discretisation errors, which cause abrupt changes in the orientation of the model of the surface.

2.2 Surface Smoothing

Having delineated an isosurface using the marching cubes algorithm, the isosurface is subsequently smoothed using Algorithm 2 to increase the accuracy of the calculated curvature by removing the artificial stair-step and other sharp features in the interface (Taubin et al. 1996). This is achieved by reducing the discretisation errors involved when voxelising the micro-CT scan and segmenting the image into different fluids (see Figure 5).

The smoothing algorithm has been adapted from AlRatrout et al. (2017) and comprises two stages. Firstly, Laplacian smoothing migrates the vertices of the isosurface to the average of their adjacent points to improve the smoothness of the delineated surface (Figure 5c). If Gaussian smoothing is also used, a second manipulation occurs where the vertices are migrated back towards their old positions to help preserve the volume of the fluids (Vollmer et al. 1999) (Figure 5d).

Each coordinate $\mathbf{i} \in \mathbf{I}$ is assigned a label, $\mathbf{i}_a$. Any face containing this point is labelled as f_a, any coordinate adjacent to this point is labelled as $\mathbf{i}_b$ and any edge containing this point is labelled as e_b. The migration of vertices during Laplacian smoothing uses the weight factors of adjacent vertices (α_b), which requires the surrounding face centres ($\mathbf{c}_f$), edge centres ($\mathbf{c}_e$) and vertex area weights ($\mathbf{w}$) to be determined, so these parameters must be calculated first.

For any coordinate $\mathbf{i}_a \in \mathbf{I}$, the set of edges containing this coordinate (E) is defined as $\forall \mathbf{i}_b \in \mathbf{I}\ \exists! e_b \in E$ and the set of faces containing this coordinate (F) is defined as $\forall f_a \ni \mathbf{i}_a, f_a \in F$. The centre of an edge ($\mathbf{c}_{e_b}$) is calculated by averaging the two coordinates at each end of the edge. The centre of a face ($\mathbf{c}_{f_a}$) is calculated by summing the position of each coordinate that belongs to the face and then dividing by the number of coordinates that form the face.

$$\mathbf{c}_{e_b} = \frac{1}{2}(\mathbf{i}_{b_1} + \mathbf{i}_{b_2}), \qquad \mathbf{i}_{b_1}, \mathbf{i}_{b_2} \in e_b \tag{4}$$

$$\mathbf{c}_{f_a} = \frac{1}{n_{\mathbf{i}_b}} \sum_{b \in f_a} \mathbf{i}_b, \qquad \mathbf{i}_b \in f_a \tag{5}$$

To calculate the vertex area weights, two vectors are defined that connect the face centre to the coordinate and the edge centre, respectively. As e_b forms part of the boundary of f_a, both vectors are contained within

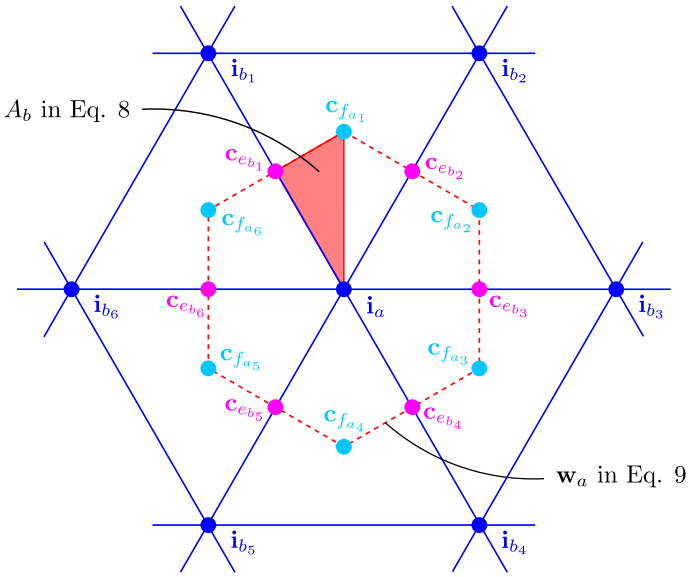


Figure 4: An example of a 2-D slice of the unsmoothed triangular mesh for $\mathbf{i}_a, \mathbf{i}_{b_x} \in \mathbf{I}$ and $f_{a_x} \in \mathbf{T}$, adapted from AlRatrout et al. (2017). The adjacent edge centres $\mathbf{c}_{e_{b_x}}$ and face centres $\mathbf{c}_{f_{a_x}}$ have been marked, which are used to calculate the areas A_b and $\mathbf{w}_a$ in Equation 8 and Equation 9, respectively.

f_a. The triangular area (A_b) between these two vectors (see Figure 4) is also contained within f_a.

$$\overrightarrow{\mathbf{i}_a\mathbf{c}_{f_a}} = \mathbf{i}_a - \mathbf{c}_{f_a}, \qquad \mathbf{i}_a, \mathbf{c}_{f_a} \in f_a \tag{6}$$

$$\overrightarrow{\mathbf{c}_{e_b}\mathbf{c}_{f_a}} = \mathbf{c}_{e_b} - \mathbf{c}_{f_a}, \qquad \mathbf{c}_{e_b}, \mathbf{c}_{f_a} \in f_a \tag{7}$$

$$A_b = \frac{1}{2}\left(\overrightarrow{\mathbf{i}_a\mathbf{c}_{f_a}} \times \overrightarrow{\mathbf{c}_{e_b}\mathbf{c}_{f_a}}\right) \tag{8}$$

The number of $\mathbf{i}_b$ or e_b for any $\mathbf{i}_a$ is counted using $\mathbf{T}$ and is labelled as $adj(\mathbf{i}_a)$. The vertex area weighting is the total of the triangular area $\forall adj(\mathbf{i}_a)$. The vertex normals are then calculated according to Thürmer (2001).

$$\mathbf{w}_a = \sum_{b \in adj(\mathbf{i}_a)} A_b \tag{9}$$

$$\mathbf{n}_a = \frac{\mathbf{w}_a}{|\mathbf{w}_a|} \tag{10}$$

The weight factor for each adjacent point (α_b) can now be determined and used to calculate the displacement to the average position of the adjacent vertices ($\mathbf{d}_a$).

$$\alpha_b = \left|\mathbf{w}_{\mathbf{o}_j}\right| + 0.3|\mathbf{w}_{\mathbf{o}_a}| \tag{11}$$

$$\mathbf{d}_a = \frac{\sum_{b \in adj(\mathbf{i}_a)} \alpha_b \mathbf{o}_b}{\sum_{b \in adj(\mathbf{i}_a)} \alpha_b} - \mathbf{o}_a \tag{12}$$

The migration vector for moving vertices during Laplacian smoothing ($\mathbf{l_a}$) is determined using the original positions of adjacent vertices ($\mathbf{o}_b$), their respective weight factors (α_b) and their vector normals ($\mathbf{n}_b$). The Laplacian smoothing step ($\mathbf{o}_a \mapsto \mathbf{l}_a$) also uses a scalar $\beta \in \{0 \to 1\}$ to control the migration, which is set to 0.1 in this study (AlRatrout et al. 2017).

$$\mathbf{l}_a = \mathbf{o}_a + 0.8\beta\left(\mathbf{d}_a \cdot \mathbf{n}_a\right)\mathbf{n}_a + 0.2\beta\mathbf{d}_a \tag{13}$$

To perform Gaussian smoothing, the average displacement during Laplacian smoothing ($\mathbf{s}_a$) is calculated:

$$\mathbf{s}_a = \frac{\sum_{b \in adj(\mathbf{i}_a)} \alpha_b \left(\mathbf{q}_b - \mathbf{o}_b\right)}{\sum_{b \in adj(\mathbf{i}_a)} \alpha_b}. \tag{14}$$

The coordinates are then migrated back towards their original positions in the Gaussian smoothing step ($\mathbf{l}_a \mapsto \mathbf{g}_a$), with the migration vector $\mathbf{g}_a$ calculated as:

$$\mathbf{g}_a = \mathbf{l}_a - \left(0.3\left(\mathbf{s}_a + \left(\mathbf{s}_a \cdot \mathbf{n}_a\right)\mathbf{n}_a\right)\right). \tag{15}$$

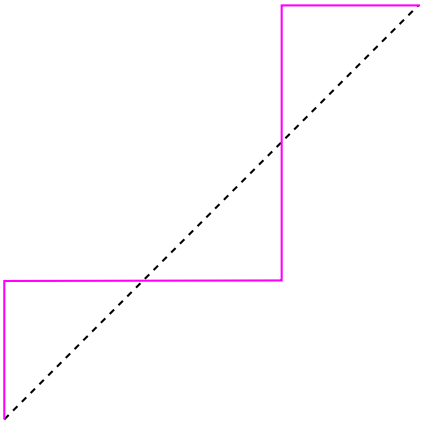
The migrations according to $\mathbf{l}_a$ or $\mathbf{g}_a$ are the final processes in each iteration of the smoothing algorithm, which attenuates shrinkage that occurs during Laplacian smoothing alone (Vollmer et al. 1999, Ji et al. 2005). The smoothing process, which updates $\mathbf{I}$ with smoothed coordinates, can be repeated for the number of Laplacian iterations ($iterations_L$) and Gaussian iterations ($iterations_G$) chosen.

2.3 Curvature Calculation

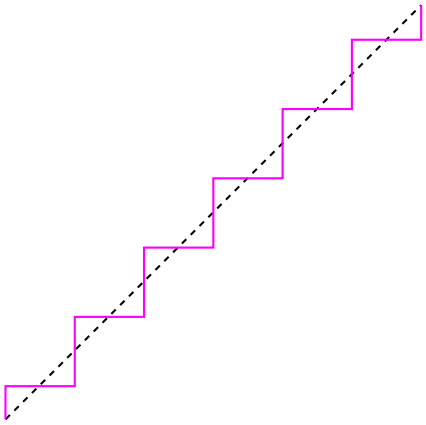
An arbitrary point within a surface has two principal curvatures (κ_1 and κ_2), which are orthogonal to each other and are defined as the maximum and minimum values of normal curvature (κ_n). The two principal curvatures are used to define the mean curvature (κ_m) and Gaussian curvature (κ_G) at a point within a surface.

$$\kappa_m = \frac{\kappa_1 + \kappa_2}{2} \tag{16}$$

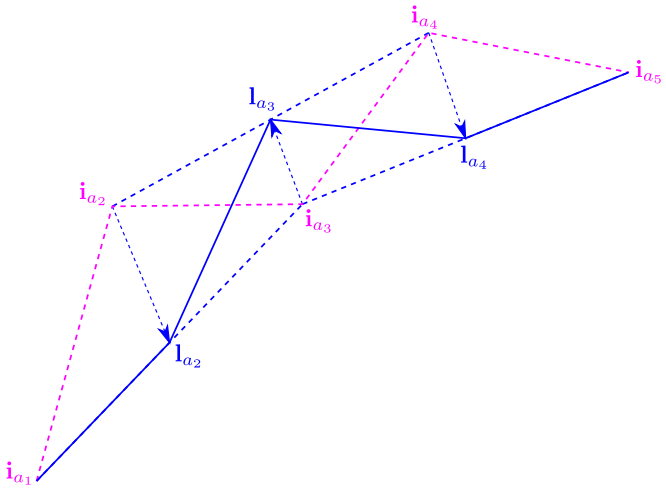
$$\kappa_G = \kappa_1\kappa_2 \tag{17}$$



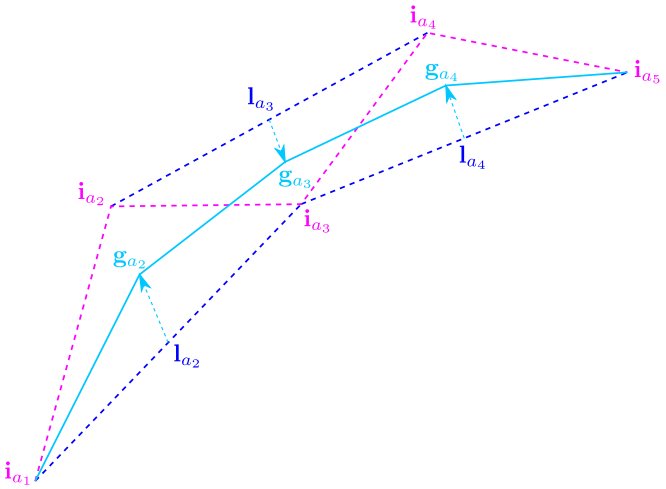
(a) Surface Delineation with Coarse Resolution



(b) Surface Delineation with Fine Resolution



(c) Laplacian Smoothing



(d) Gaussian Smoothing

Figure 5: A series of surface models, demonstrating the benefits isosurface smoothing. Figures a and b show that increasing the resolution of the surface model (pink) does not imply an increase in accuracy of the calculated curvature of the isosurface, due to the 'stair-step' features which may occur when modelling surfaces using voxelised images (here, both resolutions will produce the same calculated curvatures). However, a more accurate curvature may be produced when utilising Laplacian smoothing (c), which migrates the vertices of the isosurface to the average of their adjacent vertices using Equation 13 ($\mathbf{o}_a \mapsto \mathbf{l}_a$, navy arrow). When Gaussian smoothing (d) is utilised in conjunction with Laplacian smoothing, the vertices are migrated back towards their original positions using Equation 15 ($\mathbf{l}_a \mapsto \mathbf{g}_a$, light blue arrow). The new isosurface model after Gaussian smoothing (light blue) aims to preserve the volume of the fluids and surface features that may be lost when using Laplacian smoothing only (navy). The isosurface model may be updated iteratively to further improve accuracy.

The algorithm used to calculate the principal curvatures for every vertex within a surface is based on Rusinkiewicz (2004) and Shabat & Fischer (2015) and comprises four steps: calculating face normals for the surface, calculating vertex normals for the surface, calculating the per-face curvature and finally determining the principal curvatures for each vertex.

The calculation of face normals is achieved through calculating the cross product of any two vectors that lie within the plane that contains the face. Each of the faces output from the marching cubes algorithm is a triangle. Thus, for a triangle $t_x \in \mathbf{T}$, edge vectors $\mathbf{e}_{x_1}$ and $\mathbf{e}_{x_2}$ are calculated as:

$$\mathbf{e}_{x_1} = \mathbf{i}_{x_3} - \mathbf{i}_{x_2}, \qquad \mathbf{i}_{x_3}, \mathbf{i}_{x_2} \in t_x \qquad \textit{and} \tag{18}$$

$$\mathbf{e}_{x_2} = \mathbf{i}_{x_1} - \mathbf{i}_{x_3}, \qquad \mathbf{i}_{x_1}, \mathbf{i}_{x_3} \in t_x. \tag{19}$$

Algorithm 2: Surface Smoothing

Input: Matrices $\mathbf{I}$ and $\mathbf{T}$, representing the output of the marching cubes algorithm
Output: Updated matrix $\mathbf{I}$ 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 $(e_b \in E)$ **{**
          $\mathbf{c}_{e_b} = \frac{1}{2}(\mathbf{i}_a + \mathbf{i}_b), \qquad \mathbf{i}_a, \mathbf{i}_b \in e_b$ //calculate edge centres
        }
        for all $(f_a \in F)$ **{**
          $\mathbf{c}_{f_a} = \frac{1}{n_{\mathbf{i}_b}} \sum_{b \in f_a} \mathbf{i}_b, \qquad \mathbf{i}_b \in f_a$ //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
        $\alpha_b = \left|\mathbf{w}_{\mathbf{o}_j}\right| + 0.3|\mathbf{w}_{\mathbf{o}_a}|$ //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 */
        $\mathbf{l}_a = \mathbf{o}_a + 0.8\beta\left(\mathbf{d}_a \cdot \mathbf{n}_a\right)\mathbf{n}_a + 0.2\beta\mathbf{d}_a$ //$\mathbf{o}_a \mapsto \mathbf{l}_a$
      }
    }
    $\mathbf{s}_a = \frac{\sum_{b \in adj(\mathbf{i}_a)} \alpha_b(\mathbf{q}_b - \mathbf{o}_b)}{\sum_{b \in adj(\mathbf{i}_a)} \alpha_b}$ //calculate average displacement during Laplacian smoothing
  }
  /* Migrate vertices back towards initial positions using Gaussian smoothing to preserve volume */
  $\mathbf{g}_a = \mathbf{l}_a - (0.3\left(\mathbf{s}_a + (\mathbf{s}_a \cdot \mathbf{n}_a)\,\mathbf{n}_a\right))$ //$\mathbf{l}_a \mapsto \mathbf{g}_a$
}

The face normal is then calculated as the normalized vector of the cross product of these two edge vectors:

$$\hat{\mathbf{n}}_{t_x} = \frac{\mathbf{e}_{x_1} \times \mathbf{e}_{x_2}}{|\mathbf{e}_{x_1} \times \mathbf{e}_{x_2}|}. \tag{20}$$

The vertex normals are calculated using the common method of averaging the adjacent face normals. Firstly, all edge vectors are calculated for each triangle. Edge vectors $\mathbf{e}_{x_1}$ and $\mathbf{e}_{x_2}$ are calculated using equations 18 and 19 respectively, while $\mathbf{e}_{x_3}$ is calculated as:

$$\mathbf{e}_{x_3} = \mathbf{i}_{x_2} - \mathbf{i}_{x_1}, \qquad \mathbf{i}_{x_2}, \mathbf{i}_{x_1} \in t_x. \tag{21}$$

Each edge vector is then normalised:

$$\hat{\mathbf{e}}_{x_y} = \frac{\mathbf{e}_{x_y}}{|\mathbf{e}_{x_y}|}, \qquad \forall \mathbf{e}_{x_y} \in t_x. \tag{22}$$

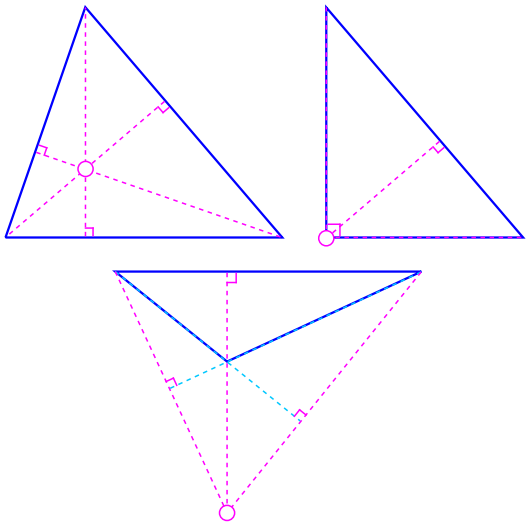


Figure 6: A schematic of the altitudes (pink lines) and orthocentres (circles) of an acute, right-angled and obtuse triangle. The light blue lines for the obtuse triangle are extended edge vectors. Whether the location of the orthocentre is interior, exterior or on the right-angled vertex of the triangle can be used to determine the triangle's shape.

Additionally, the triangle area (A_t) is calculated as:

$$A_t = \frac{1}{2}|\mathbf{e}_{x_1} \times \mathbf{e}_{x_2}|. \tag{23}$$

Secondly, the lengths of each edge are calculated, squared and then used to calculate the coordinates of the triangle's orthocentre in barycentric coordinates ($\forall t_x \in \mathbf{T}\ \exists! H_x$).

$$\begin{aligned} H_{x_1} &= |\mathbf{e}_{x_1}|^2 \times \left(|\mathbf{e}_{x_2}|^2 + |\mathbf{e}_{x_3}|^2 - |\mathbf{e}_{x_1}|^2\right) \\ H_{x_2} &= |\mathbf{e}_{x_2}|^2 \times \left(|\mathbf{e}_{x_3}|^2 + |\mathbf{e}_{x_1}|^2 - |\mathbf{e}_{x_2}|^2\right) \\ H_{x_3} &= |\mathbf{e}_{x_3}|^2 \times \left(|\mathbf{e}_{x_1}|^2 + |\mathbf{e}_{x_2}|^2 - |\mathbf{e}_{x_3}|^2\right) \end{aligned} \tag{24}$$

Thirdly, each vertex needs a weighting for its vertex normal ($\forall \mathbf{i}_{x_y} \in t_x\ \exists! w_{x_y}$), which is calculated according to Max (1999) as the area of t_x divided by the squares of the lengths of the two edges that contain $\mathbf{i}_{x_y}$.

$$\begin{aligned} w_{x_1} &= \frac{A_{t_x}}{|\mathbf{e}_{x_2}|^2 \times |\mathbf{e}_{x_3}|^2}, & \mathbf{i}_{x_1} \in t_x \\ w_{x_2} &= \frac{A_{t_x}}{|\mathbf{e}_{x_1}|^2 \times |\mathbf{e}_{x_3}|^2}, & \mathbf{i}_{x_2} \in t_x \\ w_{x_3} &= \frac{A_{t_x}}{|\mathbf{e}_{x_2}|^2 \times |\mathbf{e}_{x_1}|^2}, & \mathbf{i}_{x_3} \in t_x \end{aligned} \tag{25}$$

During the vertex normal section of the curvature calculation algorithm, these weightings are multiplied by the normal of the face that is currently being iterated over. This product is summed with the current vertex normal to produce the new vertex normal:

$$\hat{\mathbf{n}}_{\mathbf{i}_{x_y}} = \sum_{y \in \{1,2,3\},\ t_x \in \mathbf{T}} w_{x_y} \hat{\mathbf{n}}_{t_x} \tag{26}$$

$$= \hat{\mathbf{n}}_{\mathbf{i}_{x_y}} + w_{x_y} \hat{\mathbf{n}}_{t_x}, \qquad \forall \mathbf{i}_{x_y} \in t_x, \forall t_x \in \mathbf{T}. \tag{27}$$

Equation 26 has been rewritten as Equation 27 to allow $\hat{\mathbf{n}}_{\mathbf{i}_{x_y}}$ to be carried over the outermost loop in Algorithm 3, which is $\forall t_x \in \mathbf{T}$.

The weighting for each corner's share of the face curvature ($\forall \mathbf{i}_{x_y} \in t_x\ \exists! \xi_{\mathbf{i}_{x_y}}$) is calculated according to

Meyer et al. (2002) as the fraction of A_{t_x} that lies within the Voronoi polygon of that corner for that face. The weightings depend on the location of the triangle's orthocentre, as this governs the triangle's shape (see Figure 6). An acute triangle's orthocentre lies within the triangle, which is indicated by the orthocentre having all positive values ($H_{x_y} > 0 \ \forall y \in \{1, 2, 3\}$). Since the orthocentre of a right-angled triangle lies on the vertex with the right angle, and the vertices of a triangle are given as $(0, 0, 1)$, $(0, 1, 0)$ or $(1, 0, 0)$ in barycentric coordinates, the orthocentre of a right-angled triangle will have two zero values. If any value of the orthocentre in barycentric coordinates is negative, the orthocentre lies exterior to the triangle and the triangle is obtuse. The possible weighting calculations are given in algorithms 3 and 4.

The area of the Voronoi polygon for each vertex ($\forall \mathbf{i}_a \in \mathbf{I} \ \exists! V_{\mathbf{i}_a}$) is calculated as the sum of all $\xi_{\mathbf{i}_{x_y}}$ at the same location as the vertex, such that $\mathbf{i}_a = \mathbf{i}_{x_y}$:

$$V_{\mathbf{i}_a} = \sum_{x_y = a} \xi_{\mathbf{i}_{x_y}} \tag{28}$$

$$= V_{\mathbf{i}_a} + \xi_{\mathbf{i}_{x_y}}, \qquad \forall x_y = a. \tag{29}$$

Similarly to the calculations for $\hat{\mathbf{n}}_{\mathbf{i}_{x_y}}$, Equation 28 has been rewritten as Equation 29 to allow the running total for $V_{\mathbf{i}_a}$ to be carried over all $t_x \in \mathbf{T}$ in Algorithm 4.

The final part of vertex normal calculation involves defining an orthonormal coordinate system $(\mathbf{u}_{t_x}, \mathbf{v}_{t_x})$ in the tangent plane of the surface at t_x so that each adjacent face normal can be expressed in terms of a coordinate system for t_x. The initial values of $\mathbf{u}_{\mathbf{i}_{x_y}}$ are set for each face as:

$$\begin{aligned} \mathbf{u}_{\mathbf{i}_{x_1}} &= \hat{\mathbf{e}}_{x_3}, \\ \mathbf{u}_{\mathbf{i}_{x_2}} &= \hat{\mathbf{e}}_{x_1}, \\ \mathbf{u}_{\mathbf{i}_{x_3}} &= \hat{\mathbf{e}}_{x_2}. \end{aligned} \tag{30}$$

A loop over all $\mathbf{i}_a \in \mathbf{I}$ then creates the coordinate system $(\mathbf{u}_{\mathbf{i}_a}, \mathbf{v}_{\mathbf{i_a}})$ by updating and normalising $\mathbf{u}_{\mathbf{i}_a}$ and then defining $\mathbf{v}_{\mathbf{i_a}}$ (see Algorithm 4).

Although normal curvature varies with direction, it can be calculated using the second fundamental tensor ($\mathbf{\Pi}$), which in turn is determined using a least squares solver. For a smooth surface and for any unit vector $(\gamma, \ \varepsilon)$ in the local tangent plane, Rusinkiewicz (2004) define normal curvature as:

$$\kappa_n = \begin{pmatrix} \gamma & \varepsilon \end{pmatrix} \begin{bmatrix} e & f \\ f & g \end{bmatrix} \begin{pmatrix} \gamma \\ \varepsilon \end{pmatrix} = \begin{pmatrix} \gamma & \varepsilon \end{pmatrix} \mathbf{\Pi} \begin{pmatrix} \gamma \\ \varepsilon \end{pmatrix}. \tag{31}$$

The second fundamental tensor can be diagonalised by rotating the local coordinate system, which yields:

$$\kappa_n = \begin{pmatrix} \gamma_p & \varepsilon_p \end{pmatrix} \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} \begin{pmatrix} \gamma_p \\ \varepsilon_p \end{pmatrix} = \kappa_1 \gamma_p^2 + \kappa_2 \varepsilon_p^2 \tag{32}$$

where γ_p and ε_p are the principal directions in which κ_n is maximised or minimised.

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

$$\begin{aligned} \mathbf{\Pi} &= \left[\nabla_{\mathbf{u}'} \mathbf{n}_s \ \nabla_{\mathbf{v}'} \mathbf{n}_s \right] \\ &= \begin{bmatrix} \dfrac{\partial \mathbf{n}_s}{\partial \mathbf{u}'} \cdot \mathbf{u}' & \dfrac{\partial \mathbf{n}_s}{\partial \mathbf{v}'} \cdot \mathbf{u}' \\ \dfrac{\partial \mathbf{n}_s}{\partial \mathbf{u}'} \cdot \mathbf{v}' & \dfrac{\partial \mathbf{n}_s}{\partial \mathbf{v}'} \cdot \mathbf{v}' \end{bmatrix}. \end{aligned} \tag{33}$$

Multiplying the second fundamental tensor by an arbitrary vector in the tangent plane τ yields the directional

Algorithm 3: Curvature Calculation - Part 1

Input: Matrices $\mathbf{I}$ and $\mathbf{T}$, representing a triangular surface mesh
Output: Principal curvatures κ_1 and κ_2

for all $(t_x \in \mathbf{T})$ **{**
  /* Calculate Face Normals */
  $\mathbf{e}_{x_1} = \mathbf{i}_{x_3} - \mathbf{i}_{x_2}, \qquad \mathbf{i}_{x_3}, \mathbf{i}_{x_2} \in t_x$
  $\mathbf{e}_{x_2} = \mathbf{i}_{x_1} - \mathbf{i}_{x_3}, \qquad \mathbf{i}_{x_1}, \mathbf{i}_{x_3} \in t_x$
  $\hat{\mathbf{n}}_{t_x} = \frac{\mathbf{e}_{x_1} \times \mathbf{e}_{x_2}}{|\mathbf{e}_{x_1} \times \mathbf{e}_{x_2}|}$ //calculate face normals using two edge vectors
  /* Calculate Vertex Normals */
  $\mathbf{e}_{x_3} = \mathbf{i}_{x_2} - \mathbf{i}_{x_1}, \qquad \mathbf{i}_{x_2}, \mathbf{i}_{x_1} \in t_x$
  for all $(\mathbf{e}_{x_y} \in t_x)$ **{**
    $\hat{\mathbf{e}}_{x_y} = \frac{\mathbf{e}_{x_y}}{|\mathbf{e}_{x_y}|}$ //normalise all edge vectors
  }
  $A_t = \frac{1}{2}|\mathbf{e}_{x_1} \times \mathbf{e}_{x_2}|$ //calculate triangle area vector
  //Calculate components of triangle's orthocentre in barycentric coordinates
  $H_{x_1} = |\mathbf{e}_{x_1}|^2 \times \left(|\mathbf{e}_{x_2}|^2 + |\mathbf{e}_{x_3}|^2 - |\mathbf{e}_{x_1}|^2\right)$
  $H_{x_2} = |\mathbf{e}_{x_2}|^2 \times \left(|\mathbf{e}_{x_3}|^2 + |\mathbf{e}_{x_1}|^2 - |\mathbf{e}_{x_2}|^2\right)$
  $H_{x_3} = |\mathbf{e}_{x_3}|^2 \times \left(|\mathbf{e}_{x_1}|^2 + |\mathbf{e}_{x_2}|^2 - |\mathbf{e}_{x_3}|^2\right)$
  //Calculate vertex normal weightings
  $w_{x_1} = \frac{A_{t_1}}{|\mathbf{e}_{x_2}|^2 \times |\mathbf{e}_{x_3}|^2}, \qquad \mathbf{i}_{x_1} \in t_x$
  $w_{x_2} = \frac{A_{t_1}}{|\mathbf{e}_{x_1}|^2 \times |\mathbf{e}_{x_3}|^2}, \qquad \mathbf{i}_{x_2} \in t_x$
  $w_{x_3} = \frac{A_{t_1}}{|\mathbf{e}_{x_2}|^2 \times |\mathbf{e}_{x_1}|^2}, \qquad \mathbf{i}_{x_3} \in t_x$
  for all $(\mathbf{i}_{x_y} \in t_x)$ **{**
    $\hat{\mathbf{n}}_{\mathbf{i}_{x_y}} = \hat{\mathbf{n}}_{\mathbf{i}_{x_y}} + w_{x_y}\hat{\mathbf{n}}_{t_x}$ //sum vertex normals across each iteration of outermost loop $(\forall t_x \in \mathbf{T})$
  }
  //Calculate weights $\xi_{\mathbf{i}_{x_y}}$ according to the location of triangle's orthocentre
  if $(H_{x_1} \leq 0)$ **{**
    $\xi_{\mathbf{i}_{x_2}} = \frac{-0.25 \times |\mathbf{e}_{x_3}|^2 \times A_t}{\mathbf{e}_{x_1} \cdot \mathbf{e}_{x_3}}$
    $\xi_{\mathbf{i}_{x_3}} = \frac{-0.25 \times |\mathbf{e}_{x_2}|^2 \times A_t}{\mathbf{e}_{x_1} \cdot \mathbf{e}_{x_2}}$
    $\xi_{\mathbf{i}_{x_1}} = A_t - \xi_{\mathbf{i}_{x_2}} - \xi_{\mathbf{i}_{x_3}}$
  }
}

derivative of the normal in that direction:

$$\mathbf{\Pi}\tau = \nabla_\tau \mathbf{n}_s. \tag{34}$$

This definition of the second fundamental tensor for smooth surfaces can be approximated for use on the discretised triangular mesh output of the marching cubes algorithm. Combining the edge vectors of a triangle with the differences in vertex normals in the direction of the edge vectors (see Figure 7) provides a set of linear constraints for the second fundamental tensor of each triangle ($\mathbf{\Pi}_{t_x}$), which can then be solved using a least squares approach. In Algorithm 5, *'QRPT decompositon'* from GNU's GSL library is used to solve the least

Algorithm 4: Curvature Calculation - Part 2

Input: Matrices $\mathbf{I}$ and $\mathbf{T}$, representing a triangular surface mesh
Output: Principal curvatures κ_1 and κ_2

for all $(t_x \in \mathbf{T})$ **{**
  //Continue possible calculations of $\xi_{\mathbf{i}_{x_y}}$ according to the location of the triangle's orthocentre
  else if $(H_{x_2} \leq 0)$ **{**
    $\xi_{\mathbf{i}_{x_3}} = \frac{-0.25 \times \left|\mathbf{e}_{x_1}\right|^2 \times A_t}{\mathbf{e}_{x_2} \cdot \mathbf{e}_{x_1}}$
    $\xi_{\mathbf{i}_{x_1}} = \frac{-0.25 \times \left|\mathbf{e}_{x_3}\right|^2 \times A_t}{\mathbf{e}_{x_2} \cdot \mathbf{e}_{x_3}}$
    $\xi_{\mathbf{i}_{x_2}} = A_t - \xi_{\mathbf{i}_{x_1}} - \xi_{\mathbf{i}_{x_3}}$
  }
  else if $(H_{x_3} \leq 0)$ **{**
    $\xi_{\mathbf{i}_{x_1}} = \frac{-0.25 \times \left|\mathbf{e}_{x_2}\right|^2 \times A_t}{\mathbf{e}_{x_3} \cdot \mathbf{e}_{x_2}}$
    $\xi_{\mathbf{i}_{x_2}} = \frac{-0.25 \times \left|\mathbf{e}_{x_1}\right|^2 \times A_t}{\mathbf{e}_{x_3} \cdot \mathbf{e}_{x_1}}$
    $\xi_{\mathbf{i}_{x_3}} = A_t - \xi_{\mathbf{i}_{x_1}} - \xi_{\mathbf{i}_{x_2}}$
  }
  else { //all $H_{x_y} > 0 \therefore t_x$ is acute
    $H_{x_s} = \frac{0.5 \times A_t}{H_{x_1} + H_{x_2} + H_{x_3}}$ //scale factor
    $\xi_{\mathbf{i}_{x_1}} = H_{x_s} \times (H_{x_2} + H_{x_3})$
    $\xi_{\mathbf{i}_{x_2}} = H_{x_s} \times (H_{x_1} + H_{x_3})$
    $\xi_{\mathbf{i}_{x_3}} = H_{x_s} \times (H_{x_1} + H_{x_2})$
  }
  $V_{\mathbf{i}_a} = V_{\mathbf{i}_a} + \xi_{\mathbf{i}_{x_y}}, \qquad a = x_y$ //sum area of Voronoi polygon for each vertex across each iteration of outermost loop $(\forall t_x \in \mathbf{T})$

  //Set initial values of $\mathbf{u}_{\mathbf{i}_{x_y}}$
  $\mathbf{u}_{\mathbf{i}_{x_1}} = \hat{\mathbf{e}}_{x_3}$
  $\mathbf{u}_{\mathbf{i}_{x_2}} = \hat{\mathbf{e}}_{x_1}$
  $\mathbf{u}_{\mathbf{i}_{x_3}} = \hat{\mathbf{e}}_{x_2}$
}
//Update and normalise $\mathbf{u}_{\mathbf{i}_a}$ and calculate $\mathbf{v}_{\mathbf{i}_a}$
for all $(\mathbf{i}_a \in \mathbf{I})$ **{**
  $\mathbf{u}_{\mathbf{i}_a} = \mathbf{u}_{\mathbf{i}_a} \times \hat{\mathbf{n}}_{\mathbf{i}_a}$
  $\hat{\mathbf{u}}_{\mathbf{i}_a} = \frac{\mathbf{u}_{\mathbf{i}_a}}{|\mathbf{u}_{\mathbf{i}_a}|}$
  $\mathbf{v}_{\mathbf{i}_a} = \hat{\mathbf{n}}_{\mathbf{i}_a} \times \hat{\mathbf{u}}_{\mathbf{i}_a}$
}

squares problem for the second fundamental tensor (GNU 2021).

$$
\begin{aligned}
\mathbf{\Pi}_{t_x} \begin{pmatrix} \mathbf{e}_{x_1} \cdot \mathbf{u}_{t_x} \\ \mathbf{e}_{x_1} \cdot \mathbf{v}_{t_x} \end{pmatrix} &= \begin{pmatrix} \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_3}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_2}}\right) \cdot \mathbf{u}_{t_x} \\ \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_3}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_2}}\right) \cdot \mathbf{v}_{t_x} \end{pmatrix} \\
\mathbf{\Pi}_{t_x} \begin{pmatrix} \mathbf{e}_{x_2} \cdot \mathbf{u}_{t_x} \\ \mathbf{e}_{x_2} \cdot \mathbf{v}_{t_x} \end{pmatrix} &= \begin{pmatrix} \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_1}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_3}}\right) \cdot \mathbf{u}_{t_x} \\ \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_1}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_3}}\right) \cdot \mathbf{v}_{t_x} \end{pmatrix} \\
\mathbf{\Pi}_{t_x} \begin{pmatrix} \mathbf{e}_{x_3} \cdot \mathbf{u}_{t_x} \\ \mathbf{e}_{x_3} \cdot \mathbf{v}_{t_x} \end{pmatrix} &= \begin{pmatrix} \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_2}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_1}}\right) \cdot \mathbf{u}_{t_x} \\ \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_2}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_1}}\right) \cdot \mathbf{v}_{t_x} \end{pmatrix}
\end{aligned}
\tag{35}
$$

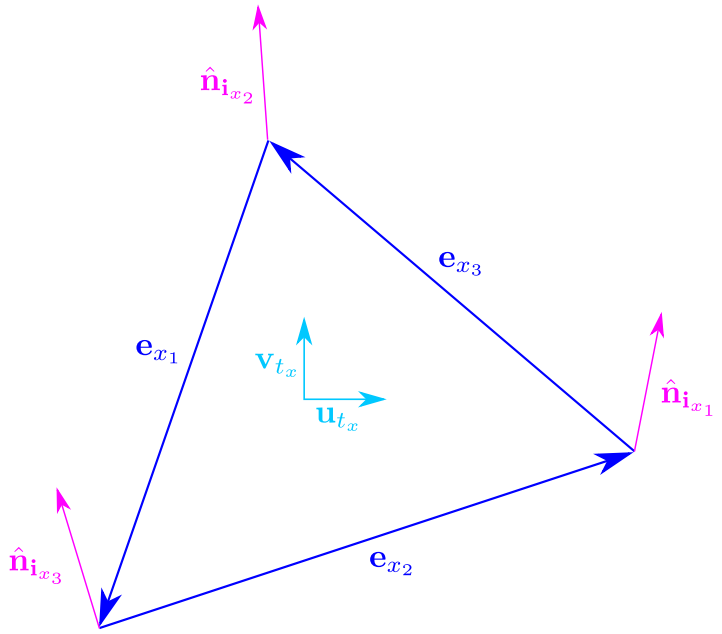


Figure 7: A diagram of a triangular face showing the components of the linear constraints that are placed on the triangle's second fundamental tensor $\mathbf{\Pi}_{t_x}$ in Equation 35 (adapted from Rusinkiewicz (2004)).

Once the second fundamental tensor for faces has been calculated, curvature per vertex can be determined. This uses Voronoi weightings for each corner of a given triangle $\left(\forall \mathbf{i}_{x_y} \in t_x \; \exists! V_{\mathbf{i}_{x_y}}\right)$, which are calculated as:

$$V_{\mathbf{i}_{x_y}} = \frac{\xi_{\mathbf{i}_{x_y}}}{V_{\mathbf{i}_a}}, \qquad \mathbf{i}_{x_y} \in t_x. \tag{36}$$

To calculate the curvature tensor for the corner of each face ($\mathbf{\Pi}_{\mathbf{i}_{x_y}}$), the face curvature tensor must be projected to the coordinate system of the corner (see Algorithm 6). In turn, this projection requires the rotation of the coordinate systems at the corners of the face to the plane of the triangle that they belong to (see Algorithm 7). One the corner curvature tensor has been calculated, it is multiplied by the corner Voronoi weightings for each face and then summed over all faces to produce the vertex curvature tensor ($\mathbf{\Pi}_{\mathbf{i}_a}$).

The vertex curvature tensor ($\mathbf{\Pi}_{\mathbf{i}_a}$) is calculated as the sum of all the weighted corner curvature tensors for every time the same vertex is used as a corner ($\forall \mathbf{i}_a = \mathbf{i}_{x_y}$):

$$\mathbf{\Pi}_{\mathbf{i}_a} = \sum_{x_y = a} V_{\mathbf{i}_{x_y}} \mathbf{\Pi}_{\mathbf{i}_{x_y}} \tag{37}$$

$$= \mathbf{\Pi}_{\mathbf{i}_a} + V_{\mathbf{i}_{x_y}} \mathbf{\Pi}_{\mathbf{i}_{x_y}}, \qquad \forall x_y = a. \tag{38}$$

Here, Equation 37 has been rewritten as Equation 38 to allow the total $\mathbf{\Pi}_{\mathbf{i}_a}$ to be calculated in Algorithm 5.

To retrieve the principal curvatures and directions for each vertex, the eigenvalues and eigenvectors could be computed from the vertex's second fundamental tensor using an appropriate solver form an external library. However, Equation 32 is computed using a manual implementation of Jacobi rotation to diagonalise the second fundamental tensor, which ensures that the calculated principal directions are perpendicular to the vertex normal (see Algorithm 8) (Rutishauser 1966, Jacobi 1846).

Once the principal curvatures and have been calculated, they can be used in equations 16 and 17 to calculate mean curvature and Gaussian curvature, respectively.

Algorithm 5: Curvature Calculation - Part 3

Input: Matrices $\mathbf{I}$ and $\mathbf{T}$, representing a surface mesh
Output: Principal curvatures κ_1 and κ_2

/* Calculate Curvature */
for all $(t_x \in \mathbf{T})$ **{**
  //Create least squares problem of form $\mathbf{Ax} = \mathbf{b}$ to solve for $\mathbf{x}$, which contains the elements of $\mathbf{\Pi}$
  $\mathbf{u}_{t_x} = \hat{\mathbf{e}}_{x_1}$ //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 $\mathbf{A}$
  $\mathbf{b} = \begin{bmatrix} \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_3}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_2}}\right) \cdot \mathbf{u}_{t_x} \\ \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_3}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_2}}\right) \cdot \mathbf{v}_{t_x} \\ \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_1}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_3}}\right) \cdot \mathbf{u}_{t_x} \\ \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_1}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_3}}\right) \cdot \mathbf{v}_{t_x} \\ \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_2}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_1}}\right) \cdot \mathbf{u}_{t_x} \\ \left(\hat{\mathbf{n}}_{\mathbf{i}_{x_2}} - \hat{\mathbf{n}}_{\mathbf{i}_{x_1}}\right) \cdot \mathbf{v}_{t_x} \end{bmatrix}$ //Define $\mathbf{b}$
  Solve for $\mathbf{x}$. //uses GSL solver
  $\mathbf{\Pi}_{t_x} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 \\ \mathbf{x}_2 & \mathbf{x}_3 \end{bmatrix}$ //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 Algorithm 6 to produce $\mathbf{\Pi}_{\mathbf{i}_{x_y}}$.
    $\mathbf{\Pi}_{\mathbf{i}_a} = \mathbf{\Pi}_{\mathbf{i}_a} + V_{\mathbf{i}_{x_y}} \mathbf{\Pi}_{\mathbf{i}_{x_y}}$ //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 Algorithm 8.
}

Algorithm 6: Project Curvature Tensor

Input: Face coordinate system $\left(\mathbf{u}_{t_x}, \mathbf{v}_{t_x}\right)$, face normal $\hat{\mathbf{n}}_{t_x}$, face curvature tensor $\mathbf{\Pi}_{t_x}$ and vector coordinate system $\left(\mathbf{u}_{\mathbf{i}_{x_y}}, \mathbf{v}_{\mathbf{i}_{x_y}}\right)$

Output: Corner curvature tensor $\mathbf{\Pi}_{\mathbf{i}_{x_y}}$

Rotate coordinate system using Algorithm 7 with inputs of $\mathbf{u}_{\mathbf{i}_{x_y}}$, $\mathbf{v}_{\mathbf{i}_{x_y}}$ and $\hat{\mathbf{n}}_{t_x}$ to produce temporary coordinate system $\left(\mathbf{u}_{\mathbf{i}_{t_x}}, \mathbf{v}_{\mathbf{i}_{t_x}}\right)$.

//Calculate dot products for use when determining the corner's second fundamental tensor

$u_1 = \mathbf{u}_{\mathbf{i}_{t_x}} \cdot \mathbf{u}_{t_x}$

$v_1 = \mathbf{u}_{\mathbf{i}_{t_x}} \cdot \mathbf{v}_{t_x}$

$u_2 = \mathbf{v}_{\mathbf{i}_{t_x}} \cdot \mathbf{u}_{t_x}$

$v_2 = \mathbf{v}_{\mathbf{i}_{t_x}} \cdot \mathbf{v}_{t_x}$

$$\mathbf{\Pi}_{\mathbf{i}_{x_y}} = \begin{bmatrix} (u_1\ v_1)\ \mathbf{\Pi}_{t_x} \begin{pmatrix} u_1 \\ v_1 \end{pmatrix} & (u_1\ v_1)\ \mathbf{\Pi}_{t_x} \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} \\ (u_1\ v_1)\ \mathbf{\Pi}_{t_x} \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} & (u_2\ v_2)\ \mathbf{\Pi}_{t_x} \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} \end{bmatrix}$$ //calculate the corner's second fundamental tensor

Algorithm 7: Rotate Coordinate System

Input: Coordinate system vectors $\mathbf{u}_{input}$ and $\mathbf{v}_{input}$, and face or vector normal $\hat{\mathbf{n}}_{input}$

Output: Rotated coordinate system $(\mathbf{u}_{output}, \mathbf{v}_{output})$

$$\hat{\mathbf{n}}_{old} = \frac{\mathbf{u}_{input} \times \mathbf{v}_{input}}{|\mathbf{u}_{input} \times \mathbf{v}_{input}|}$$ //calculate old normal

$\phi = \hat{\mathbf{n}}_{input} \cdot \hat{\mathbf{n}}_{old}$ //calculate difference between normals

if $(\phi \leq -1)$ **{** //normals are in opposite directions ($\leq$ not $=$ for floating-point errors)

  //Reverse coordinate system

  $\mathbf{u}_{output} = -\mathbf{u}_{input}$

  $\mathbf{v}_{output} = -\mathbf{v}_{input}$

}

else { //calculate new coordinate system

  $\mathbf{p} = \hat{\mathbf{n}}_{input} - \phi \hat{\mathbf{n}}_{old}$

  $$\mathbf{dp} = \frac{\hat{\mathbf{n}}_{old} + \hat{\mathbf{n}}_{input}}{1 + \phi}$$

  $\mathbf{u}_{output} = \mathbf{u}_{input} - (\mathbf{p} \cdot \mathbf{u}_{input})\,\mathbf{dp}$

  $\mathbf{v}_{output} = \mathbf{v}_{input} - (\mathbf{p} \cdot \mathbf{v}_{input})\,\mathbf{dp}$

}

Algorithm 8: Diagonalise Curvature Tensor

Input: Second fundamental tensor $\mathbf{\Pi}_{\mathbf{i}_a}$ and vector coordinate system $\left(\mathbf{u}_{\mathbf{i}_a}, \mathbf{v}_{\mathbf{i}_a}\right)$
Output: Principal curvatures (κ_1 and κ_2) and directions (γ_p and ε_p) for vertex $\mathbf{i}_a \in \mathbf{I}$

Rotate coordinate system using Algorithm 7 with inputs of $\mathbf{u}_{\mathbf{i}_a}$, $\mathbf{v}_{\mathbf{i}_a}$ and $\hat{\mathbf{n}}_{\mathbf{i}_a}$ to produce temporary coordinate system $\left(\mathbf{u}_{\mathbf{i}_{a_p}}, \mathbf{u}_{\mathbf{i}_{a_p}}\right)$
//If its non-diagonal elements are not 0, use Jacobi rotation to diagonalise the vertex's curvature tensor
if $(\mathbf{\Pi}_{\mathbf{i}_{a_{1,2}}} = \mathbf{\Pi}_{\mathbf{i}_{a_{2,1}}} \neq 0)$ **{**
  $h = \dfrac{\mathbf{\Pi}_{\mathbf{i}_{a_{2,2}}} - \mathbf{\Pi}_{\mathbf{i}_{a_{1,1}}}}{2 \times \mathbf{\Pi}_{\mathbf{i}_{a_{1,2}}}}$
  if $(h < 0)$ **{**
    $tt = \dfrac{1}{h - \sqrt{1 + h^2}}$
  }
  else {
    $tt = \dfrac{1}{h + \sqrt{1 + h^2}}$
  }
  $c = \dfrac{1}{\sqrt{1 + tt^2}}$
  $s = tt \times c$
}
else { //the vertex's curvature tensor is already a diagonal matrix
  $tt = 0$
  $c = 1$
  $s = 0$
}
//Calculate temporary principal curvatures
$\kappa_{1_{temp}} = \mathbf{\Pi}_{\mathbf{i}_{a_{1,1}}} - \left(tt \times \mathbf{\Pi}_{\mathbf{i}_{a_{1,2}}}\right)$
$\kappa_{2_{temp}} = \mathbf{\Pi}_{\mathbf{i}_{a_{2,2}}} + \left(tt \times \mathbf{\Pi}_{\mathbf{i}_{a_{1,2}}}\right)$
//Calculate first principal direction and final principal curvatures
if $(\left|\kappa_{1_{temp}}\right| \geq \left|\kappa_{2_{temp}}\right|)$ **{**
  $\gamma_p = \left(c \times \mathbf{u}_{\mathbf{i}_{a_p}}\right) - \left(s \times \mathbf{v}_{\mathbf{i}_{a_p}}\right)$
  $\kappa_1 = \kappa_{1_{temp}}$
  $\kappa_2 = \kappa_{2_{temp}}$
}
else {
  $\gamma_p = \left(s \times \mathbf{u}_{\mathbf{i}_{a_p}}\right) + \left(c \times \mathbf{v}_{\mathbf{i}_{a_p}}\right)$
  //Swap principal curvatures
  $\kappa_1 = \kappa_{2_{temp}}$
  $\kappa_2 = \kappa_{1_{temp}}$
}
$\varepsilon_p = \hat{\mathbf{n}}_{\mathbf{i}_a} \times \gamma_p$ //calculate second principal direction

3 Results and Discussion

To enable use of these algorithms in the estimation of curvature within micro-CT datasets, the accuracy of the algorithms was investigated by quantifying the uncertainty in calculated curvatures as a function of resolution. This is because one of the principal issues with automated curvature calculation occurs in the common instance where image resolution is similar to the radius of curvature of the interfaces. To facilitate the quantification of uncertainty, analytical surfaces with known exact values for mean curvature and Gaussian curvature were used as synthetic datasets. Having investigated error margins for these analytical surfaces, the methodology was extended to cover some basic analysis of real image datasets.

3.1 Analytical Surfaces

The analytical surfaces chosen for this study were the sphere and the catenoid, due to their well-known mean and Gaussian curvatures. Additionally, these shapes are representative of interfaces found in real pore spaces for two-phase displacement (Blunt et al. 2019). The sphere can be used to model the oil/water interface in a water-wet or oil-wet dataset, while the catenoid can approximate the interface in a mixed-wet sample (Akai et al. 2019).

3.1.1 Spheres

The equation of a sphere in Cartesian (x, y, z) coordinates, with radius r centred at (x_0, y_0, z_0), is:

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2. \tag{39}$$

The principal curvatures for any point on the surface of a sphere are:

$$\begin{aligned} \kappa_1^{sphere} &= \frac{1}{r}, \\ \kappa_2^{sphere} &= \frac{1}{r}. \end{aligned} \tag{40}$$

The mean curvature and Gaussian curvature can then be defined using the radius as:

$$\kappa_m^{sphere} = \frac{1}{r} \qquad and \tag{41}$$

$$\kappa_G^{sphere} = \frac{1}{r^2}. \tag{42}$$

Spherical testing began by generating datasets for the pixel values matrix $\mathbf{P}$ with an input radius (r_{input}) and voxel size (Δx). For all spheres represented in the subsequent figures, $\Delta x = 1$. A variety of input radii were used to represent datasets where resolution is similar to the radius of curvature (coarse resolution) as well as when the radius of curvature is much greater than the voxel size (fine resolution). Initially, the spheres had their surfaces delineated using Algorithm 1 but their curvature was calculated without any Gaussian or Laplacian smoothing. The mean curvature and Gaussian curvature of these 'unsmoothed' spheres was normalised by multiplying the calculated curvatures by r and r^2 respectively, and then subtracting 1 so the analytical solution for both the normalised curvatures is 0. These were plotted against the dimensionless quantity 'inverse resolution' $\left(\frac{\Delta x}{r}\right)$ for comparison between different potential interfaces (see figures 8 and 9 respectively). For unsmoothed spheres, both mean curvature and Gaussian curvature diverge from their analytical solutions as resolution increases. Although this is the inverse of the expected relationship for a perfect sphere, it can be explained due to the increased proportion of flat triangles, and therefore vertices lying along flat edges, as resolution increases when smoothing is omitted. This results in $\kappa_m^{normalised} \to -1$ and $\kappa_G^{normalised} \to -1$ rather than $\kappa_m^{normalised} \to 0$ and

$\kappa_G^{normalised} \to 0$ as resolution increases, which explains the divergence. Consequently, Algorithm 2 was written to improve the isosurface model before curvature calculations were undertaken.

To smooth the surface of the spheres so their mean and Gaussian curvatures converge as resolution increases, Laplacian smoothing is used. This smoothing mechanism migrates the vertices towards the centre of the sphere to improve surface smoothness, albeit while failing to preserve volume. The error generated in the approximation of spheres ($E_{Spheres}$) was analysed by comparing the analytical radius of curvature (R) to the computed mean radius of curvature (R_{est}) as follows:

$$E_{Spheres} = \sqrt{\frac{(R - R_{est})^2}{\Delta x^2}}. \tag{43}$$

Figures 10 and 11 show how the error margin is reduced for Laplacian-smoothed spheres. This smoothing approach is sufficient for addressing the issue of flat interfaces within unsmoothed spherical surfaces, as shown by the drastic reduction in the percentage error for the calculation of the radius of curvature from 8-9% to 1-1.5% for smaller spheres and $\approx 0.5\%$ for larger spheres when the optimal combination smoothing iterations is used. However, this approach is prone to oversmoothing, which occurs when the vertices are migrated too far and the isosurface loses its resemblance to a sphere. This can be seen in Figure 12, when the smoothing algorithm creates surfaces where $\kappa_m^{normalised} > 0$. If Laplacian smoothing were to continue beyond these optimal parameters, the mean of mean curvature would continue to rise. Thus, for more complex interfaces, feature and volume preservation are required to accurately model the interface between the fluids.

Overall, the uncertainty in calculated curvature is dependent on the resolution of the dataset. For smaller spheres with a radius of curvature of < 7 grid blocks $\left(\text{inverse resolution } \frac{\Delta x}{r} > 0.15\right)$, the curvature is accurate to within 2% of the analytical curvature within 3 iterations of Laplacian smoothing. For larger spheres with finer resolutions, where the radius of curvature ≥ 7 grid blocks $\left(\frac{\Delta x}{r} \leq 0.15\right)$, the calculated curvature is accurate to within 1.5% of the correct curvature within 5 iterations of Laplacian smoothing. These error margins are significantly lower than studies which use alternative smoothing algorithms. Akai et al. (2019) use a similar Laplacian smoothing approach but their estimated curvature is only accurate to within 10% for the larger spheres which were investigated $\left(0.025 \leq \frac{\Delta x}{r} \leq 0.2\right)$. Although Li et al. (2018) improved the calculated mean curvature by modifying the equation used for calculating mean curvature, when they used Equation 41 their error was $\geq 6\%$, regardless of the resolution. Thus, this Laplacian smoothing algorithm provides surface smoothing that reduces error margins relative to alternative methods found in the literature.

3.1.2 Catenoids

A catenoid is a minimal surface, meaning that its mean curvature is zero (Euler 1952, Colding & Minicozzi 2006). The two other common minimal surfaces are the plane (which is a trivial case therefore would not rigorously test the methodology) and a helicoid (which can be continuously deformed into a catenoid through a simple transformation) (Meeks III & Pérez 2011). Since the catenoid can approximate the interface between fluids in a mixet-wet rock, the catenoid was selected as the appropriate minimal surface for testing the algorithms (Hoffman et al. 1993, Akai et al. 2019).

A discretised dataset of pixel values ($\mathbf{P}$) that represents a catenoid can be considered a series of cocentric circles with varying radii for each layer. When the marching cubes algorithm is run on the catenoid datasets, the vertices $\mathbf{i}_a \in \mathbf{I}$ model the edge of the circle of that layer, and the triangles $t_x \in \mathbf{T}$ form the faces that connect each layer of circles within the surface of the catenoid. Thus, a catenoid can be expressed as:

$$x^2 + y^2 = c^2 \cosh^2\left(\frac{z}{c}\right), \qquad c \in \mathbb{R}. \tag{44}$$

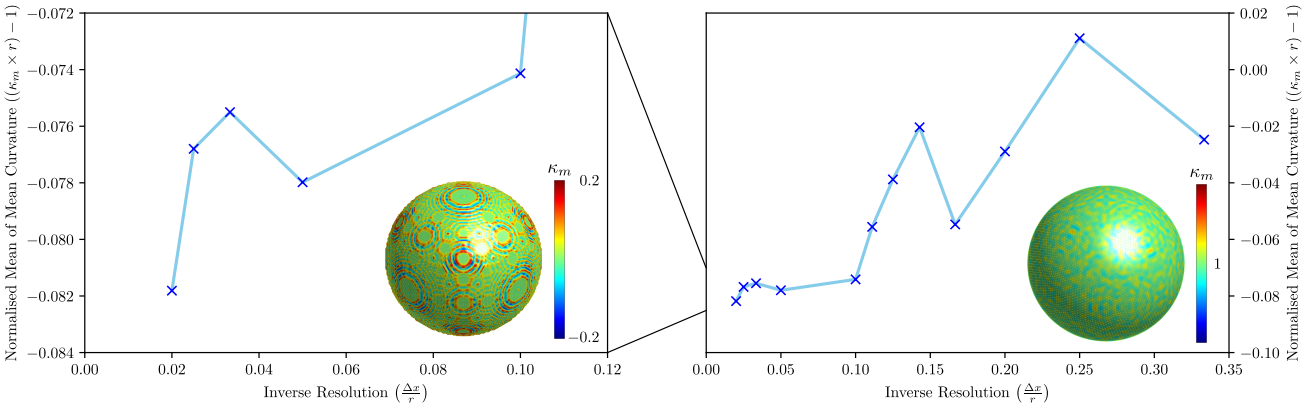


Figure 8: The normalised mean of mean curvature for unsmoothed spheres, plotted against inverse resolution. An expanded section is shown on the left, which focuses on spheres with larger radii. As inverse resolution tends to 0 and resolution tends to ∞, the analytical normalised mean curvature tends to 0. However, the normalised mean of mean curvature for these unsmoothed spheres generally diverges from the analytical solution as resolution increases. This is caused by the number of flat interfaces increasing and the proportion of vertices that form 'stair-step' interfaces in the surface decreasing. The negative y-values occur because the number of flat interfaces increases with resolution, which pushes the mean curvatures towards 0 and so the normalised mean curvatures below 0 (due to the '-1' in each normalisation formula). In some instances, an increase in resolution will produce a surface mesh that has a greater curvature without as large a rise in flat interfaces, leading to an increase in mean curvature with resolution, as can be seen between spheres with an inverse resolution of ≈ 0.14 to 0.2 ($r = 5$ to $r = 7$) and ≈ 0.03 to 0.05 ($r = 20$ to $r = 30$). For very coarse resolutions, the surface approximation of a sphere breaks down and can even cause the mean of mean curvature to exceed the analytical solution. Nonetheless, the overall divergence from the analytical mean curvature demonstrates the need for isosurface smoothing prior to curvature calculation. A sphere with inverse resolution = 0.02 ($r = 50$) is shown on the left and coloured by mean curvature at each vertex. As spheres with smaller radii relative to the voxel size do not approximate a sphere well, a unit geodesic polyhedron with 5 subdivisions per triangle (a common graphical approximation for a sphere often called an icosphere) is shown and coloured by computed mean curvature on the right (Barbosa & Colares 1997, Magid et al. 2007).

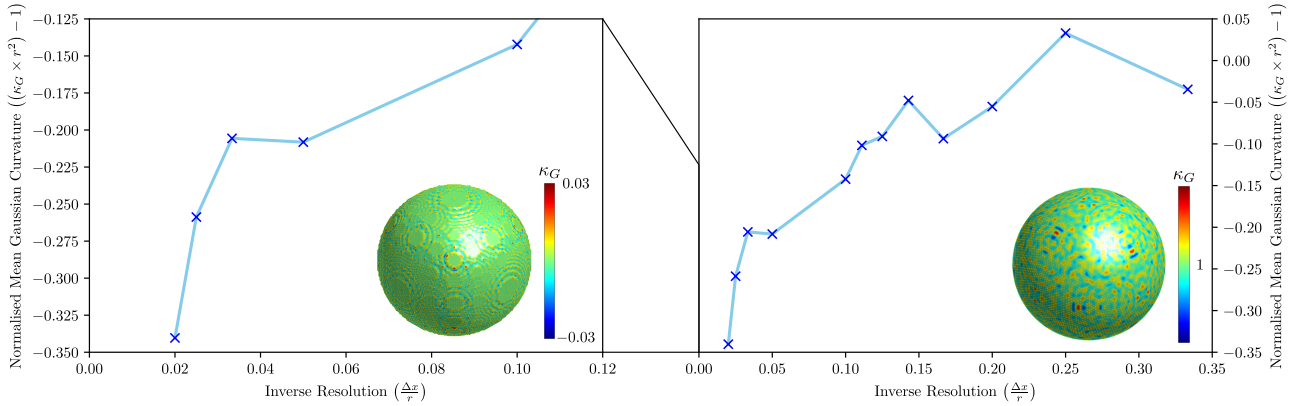
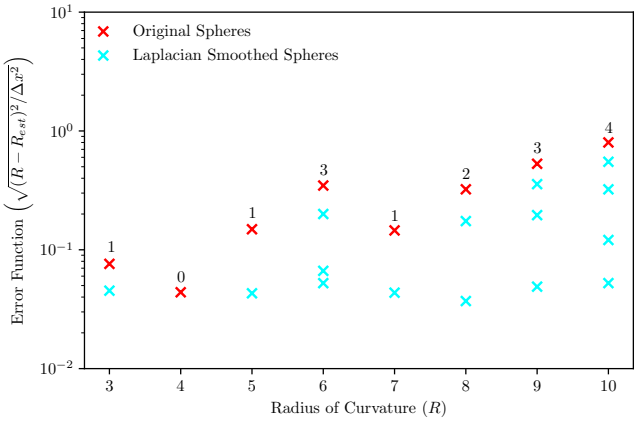
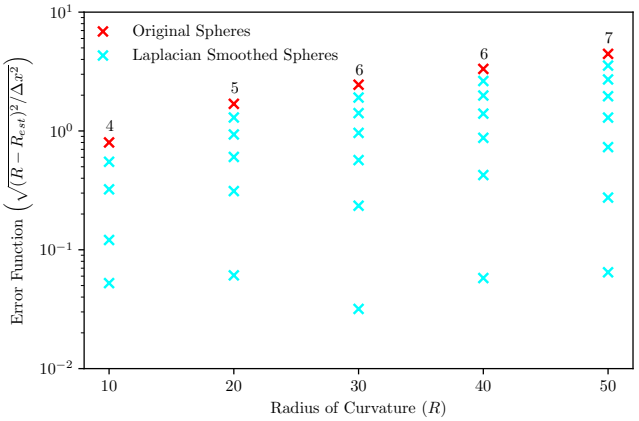


Figure 9: The normalised mean of Gaussian curvature for unsmoothed spheres, plotted against inverse resolution. Similarly to Figure 8, an expanded section is shown on the left, which focuses on spheres with larger radii and as inverse resolution tends to 0 and resolution tends to ∞, the analytical solution tends to 0. Like with mean curvature, the normalised mean of Gaussian curvature for these unsmoothed spheres generally diverges from the analytical solution as resolution increases, due to an increasing proportion of flat interfaces. In some instances, where an increase in resolution between spheres leads to an increase in the mean of mean curvature, mean Gaussian curvature also increases. However, there is a comparable overall divergence from the analytical mean Gaussian curvature as resolution increases, further demonstrating the need for isosurface smoothing prior to curvature calculation. The sphere with inverse resolution = 0.02 ($r = 50$) is shown and coloured by Gaussian curvature at each vertex, as is the unit icosphere with 5 subdivisions per triangle.

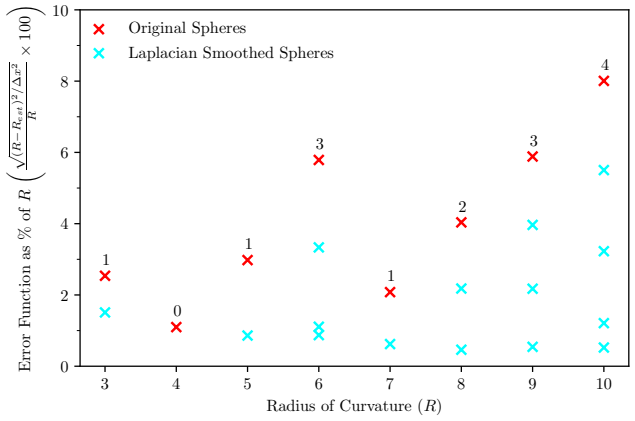


(a) Absolute Error Function - Small Spheres

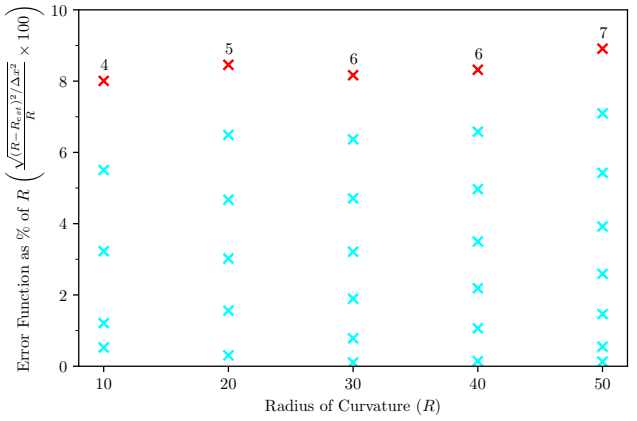


(b) Absolute Error Function - Large Spheres

Figure 10: Absolute error analysis for spheres that have undergone Laplacian smoothing using Equation 43, which shows that successive iterations of smoothing reduce the error margin of the calculated radius of curvature. The number of iterations before the error function is minimised $\left(iterations_L^{optimal}\right)$ is shown for each sphere. For smaller spheres and so coarser resolutions, the amount of smoothing is somewhat unpredictable due to the poor quality of the model of the sphere. However, once the model of a sphere uses a sufficiently high resolution (for $r \geq 7$), the amount of smoothing required is proportional to the radius of curvature $\left(iterations_L^{optimal} \propto R\right)$. This is principally because larger unsmoothed spheres have less accurate curvatures. Furthermore, once a sufficient degree of Laplacian smoothing is applied, all spheres have $E_{sphere} \leq 0.1$.



(a) Error Function as a Percentage of R- Small Spheres



(b) Error Function as a Percentage of R- Large Spheres

Figure 11: Error analysis using the absolute error function as a percentage of the radius of curvature for spheres that have undergone Laplacian smoothing, with the number of iterations before the percentage error is minimised $\left(iterations_L^{optimal}\right)$ shown for each sphere. As with the absolute error function, successive iterations of smoothing reduce percentage error of the calculated radius of curvature. For larger spheres, the unsmoothed percentage error is ≈ 8 to 9%. The proprotionality between $\left(iterations_L^{optimal}\right)$ and R is explained because the reduction in percentage error for a single iteration of smoothing reduces as the radius of curvature increases. The minimised percentage error is higher for smaller spheres which undergo less smoothing, although still below 2%. For the larger spheres ($r \geq 7$), the minimised percentage error is $< 0.7\%$.

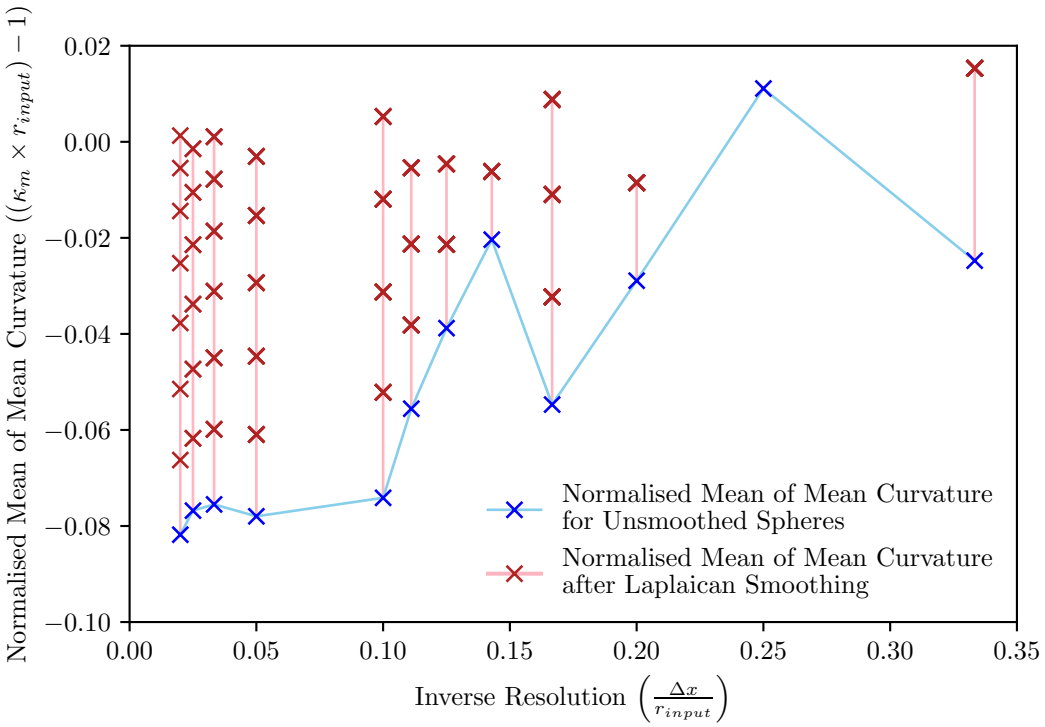


Figure 12: A comparison of the normalised mean of mean curvature with inverse resolution for spheres that have undergone optimised Laplacian smoothing, such that the difference between the calculated and analytical mean curvature is minimised. For smaller spheres, the maximum error is ≈ 0.015 (for $r = 3$). For larger spheres ($r \geq 7$), the maximum error is ≈ 0.006 (for $r = 7$). The larger spheres have mean curvature's closest to the analytical solution, even though their radii are larger, because the percentage error for larger spheres is lower than for smaller spheres.

where x, y and z are Cartesian coordinates and c is a non-zero constant. A derivation of Equation 44 can be found in Appendix B. The principal curvatures and mean curvature of a catenoid can be defined parametrically in terms of $v \in \mathbb{R}$ and c as follows:

$$\begin{aligned} \kappa_1^{catenoid} &= \frac{1}{c}\operatorname{sech}^2\left(\frac{v}{c}\right), \qquad c \neq 0 \\ \kappa_2^{catenoid} &= -\frac{1}{c}\operatorname{sech}^2\left(\frac{v}{c}\right), \qquad c \neq 0 \end{aligned} \tag{45}$$

$$\kappa_m^{catenoid} = 0. \tag{46}$$

Synthetic catenoid datasets were constructed using Equation 44, which was evaluated for a new z value at each layer. The limits on the z values were set to ± 1 in this study ($z_{max} = 1$). The difference between the z value for two adjacent layers was set by the voxel size $\Delta x = 0.1$; hence, there were 21 layers in each dataset, with each circle stored in one layer of the three-dimensional matrix $\mathbf{P}$. The constant c was varied to create catenoids with differing resolutions, as the radius of each circle is a function of z and c. The inverse resolution is $c\Delta x$ as decreasing c with a fixed $\Delta x = 0.1$ will increase the number of voxels used to model each circle within a layer of $\mathbf{P}$.

The mean curvature for unsmoothed catenoids is plotted in Figure 13 and demonstrates the complexity of modelling a catenoid. Although mean curvature eventually decreases with increased resolution, this is due to the increased proportion of flat interfaces and not the quality of the catenoid model. This is because increasing resolution alone will create 'stair-step' features in the isosurface model due to discretisation errors that are

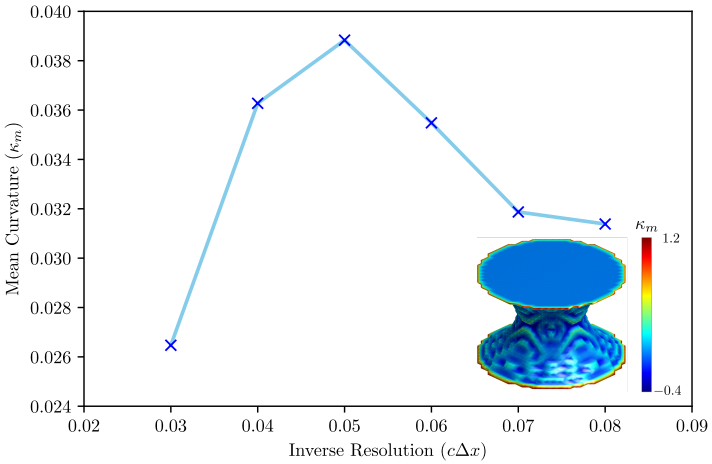


Figure 13: The mean curvature of unsmoothed catenoids, plotted against inverse resolution. The denominator of the inverse resolution, which is the radius of the circle that models the catenoid for a given z, is derived in Appendix B. The mean curvature for a catenoid is 0 when inverse resolution = 0. Initially, as resolution increases and inverse resolution $\to$ 0, mean curvature diverges from the analytical solution. This is because as resolution increases initially, the proportion of interfaces that were depicted as flat will decrease, so the mean curvature rises. However, once a sufficient resolution is reached, the inverse relationship occurs. Because the number of layers is constant, increasing the resolution will lead to flat interfaces along the surface of each circle before enough voxels have passed such that the z value increases. This effectively reintroduces 'stair-step' interfaces as resolution increases as with spherical datasets. Consequently, smoothing the catenoids is required to better approximate a minimal surface. An unsmoothed catenoid with $c = 0.7$ is shown in the bottom-right corner, coloured by mean curvature at each vertex. Although the top and bottom layers are coloured, these are only included to enable smoothing - these layers are excluded from calculations of means of curvature.

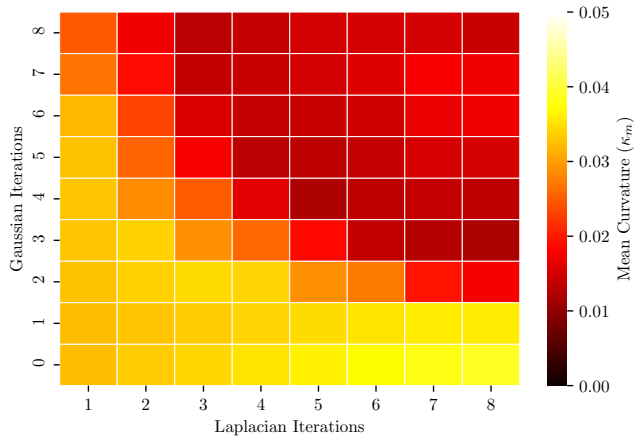


Figure 14: An example of a heatmap of the calculated mean curvature for a catenoid with $c = 0.7$ over different combinations of smoothing iterations, where the analytical mean curvature is 0. For isosurface smoothing with low levels of Gaussian smoothing (such as Laplacian smoothing or Gaussian iterations = 1), the mean curvature increases with additional Laplacian iterations, demonstrating the need for additional Gaussian iterations. The heatmap also shows instances of oversmoothing for Gaussian smoothing: for higher Gaussian iterations, the mean curvature initially decreases towards the analytical solution as Laplacian iterations increase but then increases with further Laplacian iterations, due to oversmoothing creating sharp interfaces. This also explains why mean curvature may even begin to decrease again for very high smoothing iterations (when the oversmoothing is severe). For this catenoid, the optimal smoothing parameters were $iterations_G^{optimal} = 5$ and $iterations_L^{optimal} = 2$, where $\kappa_m \approx 0.025$. Thus, although Gaussian smoothing has helped to improve the calculated curvature, it must be utilised while simultaneous monitoring of oversmoothing is in place.

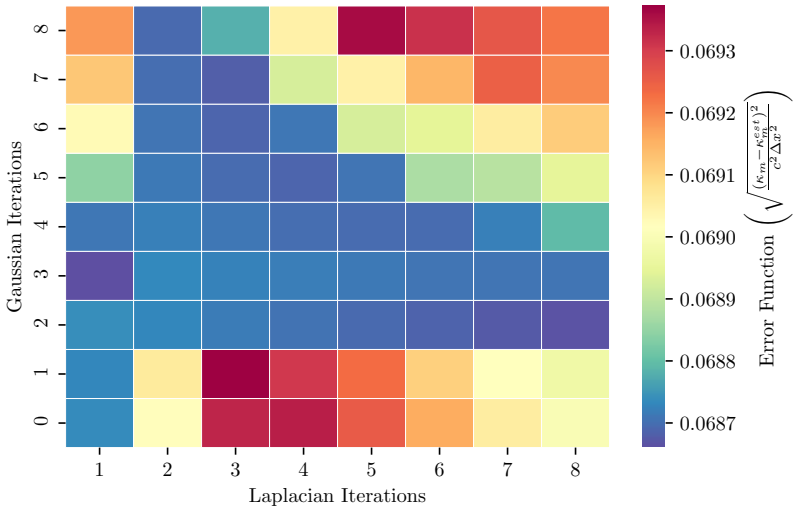


Figure 15: The error function for a catenoid with $\Delta x = 0.7$, calculated according to Equation 47. The error function shows that for lower iterations of Gaussian smoothing, the smoothing is not sufficient to significantly reduce the error of calculated curvatures. For higher iterations of smoothing, the threat of oversmoothing is shown by the larger errors. Through manual inspection of the surface model post-smoothing, optimised parameters for smoothing for this catenoid are $iterations_G = 5$ and $iterations_L = 2$, where $E_{catenoid} \approx 0.048$.

produced when using a finite resolution. Thus, smoothing of the catenoid surfaces is required to improve the calculated curvatures.

Laplacian smoothing was initially used to smooth the catenoids. However, because this approach alone reduces the number of flat interfaces within the surface without adequate feature preservation, using Laplacian smoothing causes an increase in the calculated mean curvature for these surfaces. Thus, Gaussian smoothing is required to smooth the isosurface while reducing feature loss.

Figure 14 shows a heatmap of the combinations of Gaussian and Laplacian smoothing iterations which were tested on the catenoid with $\Delta x = 0.7$. Laplacian smoothing, shown in the bottom row where Gaussian iterations are 0, is shown to cause a divergence from the analytical solution. Gaussian smoothing initially causes the mean curvature to converge to 0, although this approach is still prone to oversmoothing and does not reach the analytical solution. This highlights that although the Gaussian smoothing can help improve the curvature properties of a surface, it does not perfectly prevent feature loss and so must be deployed carefully to ensure the geometry of the isosurface is preserved.

$$E_{catenoid} = \sqrt{\frac{(\kappa_m - \kappa_m^{est})^2}{\alpha^2 \Delta x^2}} \tag{47}$$

Although higher iterations of smoothing reduce the mean curvature, inspection of the smoothed surface reveals this is due to oversmoothing where the model no longer resembles a catenoid and so should be disregarded as an appropriate solution. Analysis of the error function of a catenoid was conducted using Equation 47 and is shown for the catenoid with $\Delta x = 0.7$. As with spherical datasets, oversmoothing can be seen for higher iterations of smoothing. However, unlike with spherical interfaces, increased resolution does not reduce the error margins post-smoothing. The failure to sufficiently improve the calculated curvatures of the catenoids using the Laplcian and Gaussian smoothing approaches emphaise that for surfaces with greater complexity, an improved smoothing approach must be utilised.

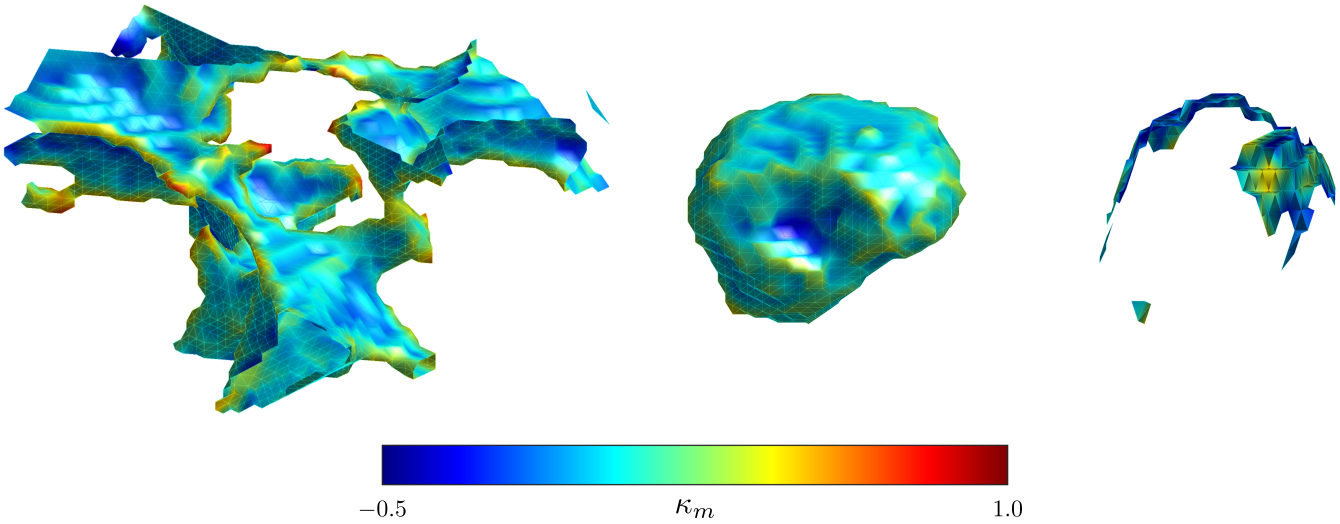


Figure 16: An example of surface delineation in a water-wet sample of Bentheimer sandstone, for the brine phase (left), oil phase (centre) and oil-water interface (right). These unsmoothed interfaces demonstrate that the marching cubes algorithm works well on real micro-CT images, and can be used to successfully delineate the interface between fluids within a pore space. However, the mean curvature for the unsmoothed interface is negative (≈ -0.025), when water-wet pore spaces are expected to have positive mean curvatures. Thus, smoothing of the interface is required to produce a more accurate curvature calculation.

3.2 Sandstone micro-CT Datasets

To test the methodology on real datasets of flow in porous media, the isosurface delineation, surface smoothing and curvature calculation algorithms were applied to micro-CT images of oil/water interfaces within Bentheimer sandstones, which were collected as part of Lin et al. (2019) and have $\Delta x = 3.578\mu$m. The pore-scale physics is related to the total curvature ($\kappa_{total} = \kappa_1 + \kappa_2$) through the Young-Laplace equation (Blunt 2017):

$$P_c = \sigma \left(\frac{1}{r_1} + \frac{1}{r_2} \right) = \kappa_{total} \sigma. \tag{48}$$

The mean curvature can be used to relate the delineated interface to capillary pressure and interfacial tension based on Equation 48 (Akai et al. 2019):

$$P_c = 2\sigma\kappa_m. \tag{49}$$

It is expected that the water-wet case will have a positive Gaussian curvature, while the mixed-wet case will have a negative Gaussian curvature (Akai et al. 2019).

The surface delineation for a water-wet rock of the brine phase, oil phase and oil-water interface is shown and coloured by computed mean curvature for each vertex in Figure 16. For this real micro-CT image, the marching cubes algorithm is successful in separating each phase within the micro-CT image, whilst also facilitating the detection of the oil-water interface by locating the faces and vertices which occur in both the brine surface and oil surface. Here, the marching cubes algorithm and the curvature calculation algorithm are shown to be robust, as they are able to handle issues that may arise within real datasets, such as disconnected interfaces. However, the mean curvature for the unsmoothed interface is negative (≈ -0.025), so smoothing of the isosurface is required for accurate curvature computation.

Laplacian smoothing is successful in increasing the mean curvature to positive values (after $iterations_L = 8$, the mean of mean curvature ≈ 0.002, see Figure 17. However, when considering that this image has roughly $0.05 \leq \frac{\Delta x}{r} \leq 0.1$, it is likely this level of smoothing means the mean curvature only reaches positive values due to shrinkage.

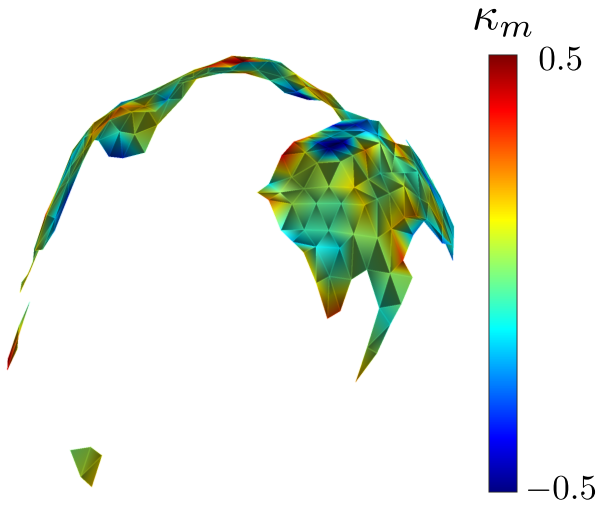


Figure 17: The oil-water interface after 8 iterations of Laplacian smoothing, where the mean of mean curvature ≈ 0.002. Although this is a positive mean curvature as is expected in a water-wet system, it is likely this is achieved due to shrinkage as volume is not preserved when using Laplacian smoothing alone.

For Gaussian smoothing, the mean curvature remains negative for lower smoothing iterations, and at higher iterations oversmoothing renders these surfaces unsuitable. Thus, in a similar fashion to the catenoid datasets, the analysis of this water-wet sample demonstrates that the smoothing algorithm needs to be improved to produce accurate curvatures and prevent the surface from being oversmoothed.

4 Conclusion

This study was undertaken to produce an open-source methodolgy for surface delineation, isosurface smoothing and curvature calculation. These three principal algorithms were successfully written in *C/C++* and have been tested on synthetic datasets of analytical surfaces and real micro-CT images of rocks.

The implementations of the marching cubes algorithm and the curvature calculation algorithm were very successful. Both are able to handle complex geometries and are robust even when being used on real datasets, where issues such as discontinuous data arise.

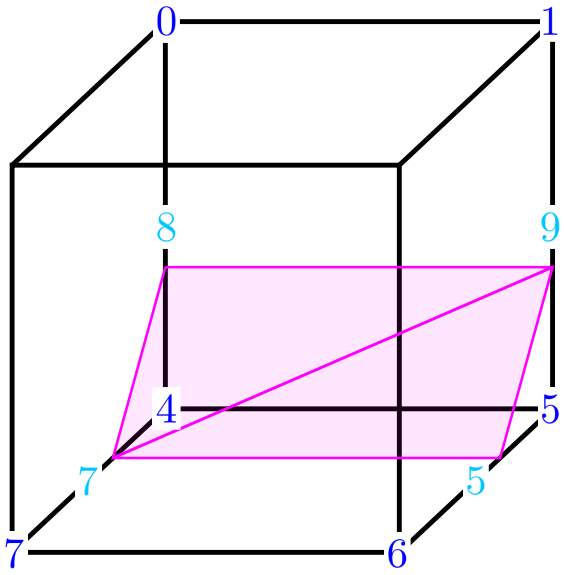
The surface detection algorithm produced mixed results. When using Laplacian smoothing only on simple spherical geometries, the algorithm produced very accurate curvatures. The error in calculated mean curvature was shown to be ¡2% of the radius of curvature for coarse resolutions and ¡1.5% of the radius of curvature for fine resolutions, both of which are significantly lower than comparable error margins in studies that employ alternative methodologies for Laplacian smoothing.

However, testing on complicated synthetic datasets and real images emphasised that the Gaussian smoothing algorithm has significant potential enhancements. Primarily, the smoothing algorithm should incorporate an analysis of oversmoothing, which would assist in choosing the optimal smoothing parameters. Additionally, future work may improve the smoothing algorithm by performing a detailed sensitivity analysis on all parameters that were used in this study. Once these changes are incorporated into the smoothing algorithm, this methodology has the potential to provide a robust and accurate approach for the calculation of interfacial curvature in pore-scale images.

A Lookup Tables

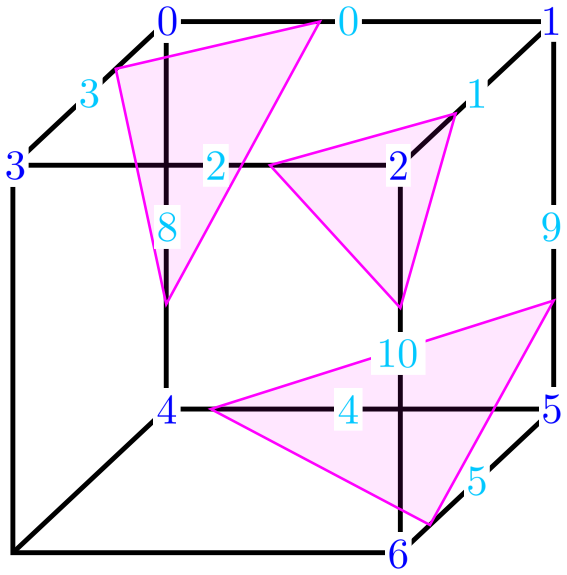
A.1 Examples of Triangular Isosurface Modelling within a Single Cube

These are some examples of isosurface models generated by the marching cubes algorithm (see Subsection 2.1) which require more than one triangle within a single cube; for an example of a single triangle modelling the isosurface within a single cube, see Figure 2. These figures are a visualisation of the output of the Edge-Triangle table in Appendix A.3.

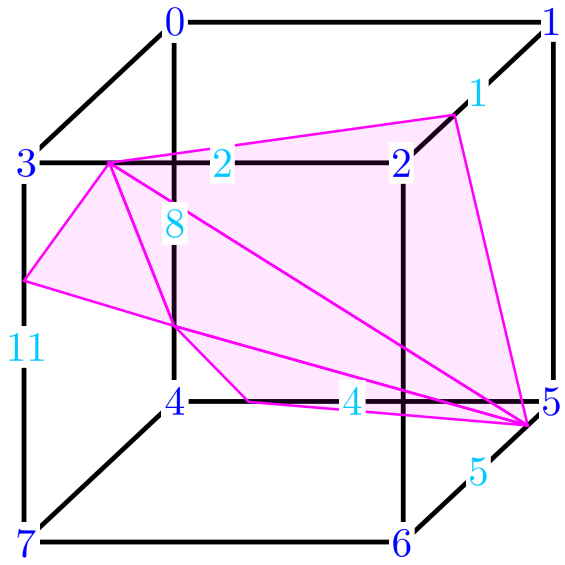


(a) An example of two connected triangles which are generated when vertices 4 and 5 are below the isovalue but all other vertices are above it, or vice versa. Depending on the linear interpolation, the triangles may be coplanar (therefore forming one face of the isosurface model) or may form two distinct faces.

Vertex Index
Edge Index
Isosurface

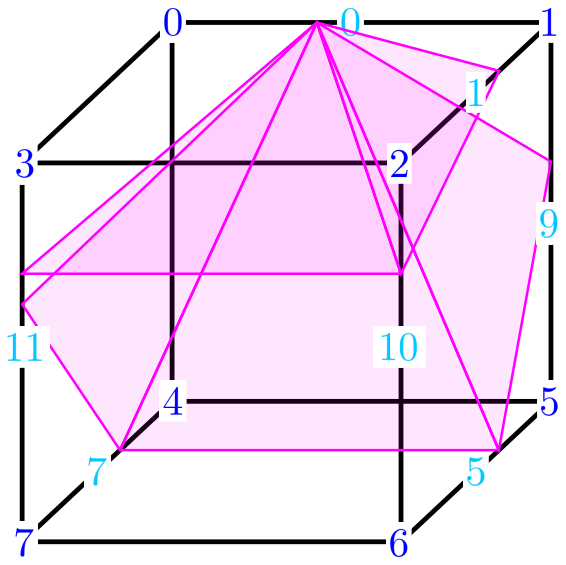


(b) An example of three triangles which are generated when vertices 0, 2 and 5 are below the isovalue but all other vertices are above it, or vice versa. Here, the isosurface has three distinct faces within the cube.



(c) An example of four connected triangles which are generated when vertices 0, 1, 3 and 5 are below the isovalue but vertices 2, 4, 6 and 7 are above it, or vice versa. Depending on the linear interpolation, the triangles may be coplanar (therefore forming one face of the isosurface model) or may form up to four distinct faces.

Vertex Index
Edge Index
Isosurface



(d) An example of five triangles which model two faces of the isosurface as it passes through the cube. Here, vertices 0, 2, 3, 4 and 5 are all above the isovalue while vertices 1, 6 and 7 are below it, or vice versa. Two 'upper' triangles are used to separate vertices 0, 2 and 3 from vertices 1, 6 and 7 whilst three additional 'lower' triangles are used to separate vertices 4 and 5 from vertices 1, 6 and 7. Depending on the linear interpolation, the two 'upper' triangles may be coplanar, as may the three 'lower' triangles, or the triangles may model up to five distinct faces of the isosurface.

Figure A.1: Examples of isosurface models which require more than one triangle. Depending on the linear interpolation, the triangles in Figure A.1a and Figure A.1c may be coplanar and so form one face of the isosurface model. Figure A.1d shows an example of multiple faces comprising more than one triangle being contained within the same cube, and demonstrates that even modelling within a single cube can become complicated. It is worth noting that even if faces are modelled by only a single triangle within a cube, such as in Figure A.1b, they may contain numerous coplanar triangles which together will construct a singular face of the isosurface in adjacent cubes.

A.2 Vertex-Edge Table

This is the Vertex-Edge lookup table, which is a simple list that is $2^{number\ of\ cube\ edges} = 2^8 = 256$ elements long, read along each column before proceeding to the row below. The 8-bit binary cube-vertex index (ψ_n for $\mathbf{c}_n \in C$) is passed as the index of the table, with a binary true stored for every vertex of a cube within the fluid volume and a binary false for every vertex outside the fluid. The table returns the 12-bit binary cube-edge index (ω_n for $\mathbf{c}_n \in C$), with a binary true stored for every edge of the cube that is bisected by the isosurface and a binary false for every edge that the isosurface does not pass through. The table below shows the hexadecimal form of the 12-bit binary output ω_n. The table returns 0 if none of the edges are bisected by the isosurface, which occurs when $\psi_n = 0$ (all vertices are below the isosurface) or $\psi_n = 0\text{xff} = 11111111 = 255$ (all vertices are above the isosurface).

0x0	0x109	0x203	0x30a	0x406	0x50f	0x605	0x70c
0x80c	0x905	0xa0f	0xb06	0xc0a	0xd03	0xe09	0xf00
0x190	0x99	0x393	0x29a	0x596	0x49f	0x795	0x69c
0x99c	0x895	0xb9f	0xa96	0xd9a	0xc93	0xf99	0xe90
0x230	0x339	0x33	0x13a	0x636	0x73f	0x435	0x53c
0xa3c	0xb35	0x83f	0x936	0xe3a	0xf33	0xc39	0xd30
0x3a0	0x2a9	0x1a3	0xaa	0x7a6	0x6af	0x5a5	0x4ac
0xbac	0xaa5	0x9af	0x8a6	0xfaa	0xea3	0xda9	0xca0
0x460	0x569	0x663	0x76a	0x66	0x16f	0x265	0x36c
0xc6c	0xd65	0xe6f	0xf66	0x86a	0x963	0xa69	0xb60
0x5f0	0x4f9	0x7f3	0x6fa	0x1f6	0xff	0x3f5	0x2fc
0xdfc	0xcf5	0xfff	0xef6	0x9fa	0x8f3	0xbf9	0xaf0
0x650	0x759	0x453	0x55a	0x256	0x35f	0x55	0x15c
0xe5c	0xf55	0xc5f	0xd56	0xa5a	0xb53	0x859	0x950
0x7c0	0x6c9	0x5c3	0x4ca	0x3c6	0x2cf	0x1c5	0xcc
0xfcc	0xec5	0xdcf	0xcc6	0xbca	0xac3	0x9c9	0x8c0
0x8c0	0x9c9	0xac3	0xbca	0xcc6	0xdcf	0xec5	0xfcc
0xcc	0x1c5	0x2cf	0x3c6	0x4ca	0x5c3	0x6c9	0x7c0
0x950	0x859	0xb53	0xa5a	0xd56	0xc5f	0xf55	0xe5c
0x15c	0x55	0x35f	0x256	0x55a	0x453	0x759	0x650
0xaf0	0xbf9	0x8f3	0x9fa	0xef6	0xfff	0xcf5	0xdfc
0x2fc	0x3f5	0xff	0x1f6	0x6fa	0x7f3	0x4f9	0x5f0
0xb60	0xa69	0x963	0x86a	0xf66	0xe6f	0xd65	0xc6c
0x36c	0x265	0x16f	0x66	0x76a	0x663	0x569	0x460
0xca0	0xda9	0xea3	0xfaa	0x8a6	0x9af	0xaa5	0xbac
0x4ac	0x5a5	0x6af	0x7a6	0xaa	0x1a3	0x2a9	0x3a0
0xd30	0xc39	0xf33	0xe3a	0x936	0x83f	0xb35	0xa3c
0x53c	0x435	0x73f	0x636	0x13a	0x33	0x339	0x230
0xe90	0xf99	0xc93	0xd9a	0xa96	0xb9f	0x895	0x99c
0x69c	0x795	0x49f	0x596	0x29a	0x393	0x99	0x190
0xf00	0xe09	0xd03	0xc0a	0xb06	0xa0f	0x905	0x80c
0x70c	0x605	0x50f	0x406	0x30a	0x203	0x109	0x0

A.3 Edge-Triangle Table

This is the Edge-Triangle lookup table, which is a table of $2^{number\ of\ cube\ edges} = 2^8 = 256$ rows and 16 columns. Each row represents a possible configuration of the triangles that are required to model the isosurface within a cube. Each triangle within the cube is defined using its vertices, as each vertex lies along an edge of the cube which is bisected by the isosurface. The indices of these edges are listed in the table in groups of three for each triangle. Once all the triangles within the cube have been listed, the remainder of the row is filled with the termination character -1. The isosurface model within a cube may require up to 5 triangles and a termination character, hence each row is 16 elements long (see Figure 2 and Appendix A.1 for example figures of the triangular models). The 12-bit binary cube-edge index (ω_n for $\mathbf{c}_n \in C$) is passed as the index to this table, which returns the 'triangles' vector $\mathbf{t}$ that is simply the entire ω^{th} row of the table.

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	8	3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	1	9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	8	3	9	8	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	2	10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	8	3	1	2	10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	2	10	0	2	9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	8	3	2	10	8	10	9	8	-1	-1	-1	-1	-1	-1	-1
3	11	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	11	2	8	11	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	9	0	2	3	11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	11	2	1	9	11	9	8	11	-1	-1	-1	-1	-1	-1	-1
3	10	1	11	10	3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	10	1	0	8	10	8	11	10	-1	-1	-1	-1	-1	-1	-1
3	9	0	3	11	9	11	10	9	-1	-1	-1	-1	-1	-1	-1
9	8	10	10	8	11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	7	8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	3	0	7	3	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	1	9	8	4	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	1	9	4	7	1	7	3	1	-1	-1	-1	-1	-1	-1	-1
1	2	10	8	4	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	4	7	3	0	4	1	2	10	-1	-1	-1	-1	-1	-1	-1
9	2	10	9	0	2	8	4	7	-1	-1	-1	-1	-1	-1	-1
2	10	9	2	9	7	2	7	3	7	9	4	-1	-1	-1	-1
8	4	7	3	11	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	4	7	11	2	4	2	0	4	-1	-1	-1	-1	-1	-1	-1
9	0	1	8	4	7	2	3	11	-1	-1	-1	-1	-1	-1	-1
4	7	11	9	4	11	9	11	2	9	2	1	-1	-1	-1	-1
3	10	1	3	11	10	7	8	4	-1	-1	-1	-1	-1	-1	-1
1	11	10	1	4	11	1	0	4	7	11	4	-1	-1	-1	-1
4	7	8	9	0	11	9	11	10	11	0	3	-1	-1	-1	-1
4	7	11	4	11	9	9	11	10	-1	-1	-1	-1	-1	-1	-1

9	5	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	5	4	0	8	3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	5	4	1	5	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	5	4	8	3	5	3	1	5	-1	-1	-1	-1	-1	-1	-1
1	2	10	9	5	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	0	8	1	2	10	4	9	5	-1	-1	-1	-1	-1	-1	-1
5	2	10	5	4	2	4	0	2	-1	-1	-1	-1	-1	-1	-1
2	10	5	3	2	5	3	5	4	3	4	8	-1	-1	-1	-1
9	5	4	2	3	11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	11	2	0	8	11	4	9	5	-1	-1	-1	-1	-1	-1	-1
0	5	4	0	1	5	2	3	11	-1	-1	-1	-1	-1	-1	-1
2	1	5	2	5	8	2	8	11	4	8	5	-1	-1	-1	-1
10	3	11	10	1	3	9	5	4	-1	-1	-1	-1	-1	-1	-1
4	9	5	0	8	1	8	10	1	8	11	10	-1	-1	-1	-1
5	4	0	5	0	11	5	11	10	11	0	3	-1	-1	-1	-1
5	4	8	5	8	10	10	8	11	-1	-1	-1	-1	-1	-1	-1
9	7	8	5	7	9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	3	0	9	5	3	5	7	3	-1	-1	-1	-1	-1	-1	-1
0	7	8	0	1	7	1	5	7	-1	-1	-1	-1	-1	-1	-1
1	5	3	3	5	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	7	8	9	5	7	10	1	2	-1	-1	-1	-1	-1	-1	-1
10	1	2	9	5	0	5	3	0	5	7	3	-1	-1	-1	-1
8	0	2	8	2	5	8	5	7	10	5	2	-1	-1	-1	-1
2	10	5	2	5	3	3	5	7	-1	-1	-1	-1	-1	-1	-1
7	9	5	7	8	9	3	11	2	-1	-1	-1	-1	-1	-1	-1
9	5	7	9	7	2	9	2	0	2	7	11	-1	-1	-1	-1
2	3	11	0	1	8	1	7	8	1	5	7	-1	-1	-1	-1
11	2	1	11	1	7	7	1	5	-1	-1	-1	-1	-1	-1	-1
9	5	8	8	5	7	10	1	3	10	3	11	-1	-1	-1	-1
5	7	0	5	0	9	7	11	0	1	0	10	11	10	0	-1
11	10	0	11	0	3	10	5	0	8	0	7	5	7	0	-1
11	10	5	7	11	5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
10	6	5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	8	3	5	10	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	0	1	5	10	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	8	3	1	9	8	5	10	6	-1	-1	-1	-1	-1	-1	-1
1	6	5	2	6	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	6	5	1	2	6	3	0	8	-1	-1	-1	-1	-1	-1	-1
9	6	5	9	0	6	0	2	6	-1	-1	-1	-1	-1	-1	-1
5	9	8	5	8	2	5	2	6	3	2	8	-1	-1	-1	-1
2	3	11	10	6	5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	0	8	11	2	0	10	6	5	-1	-1	-1	-1	-1	-1	-1
0	1	9	2	3	11	5	10	6	-1	-1	-1	-1	-1	-1	-1

5	10	6	1	9	2	9	11	2	9	8	11	-1	-1	-1	-1
6	3	11	6	5	3	5	1	3	-1	-1	-1	-1	-1	-1	-1
0	8	11	0	11	5	0	5	1	5	11	6	-1	-1	-1	-1
3	11	6	0	3	6	0	6	5	0	5	9	-1	-1	-1	-1
6	5	9	6	9	11	11	9	8	-1	-1	-1	-1	-1	-1	-1
5	10	6	4	7	8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	3	0	4	7	3	6	5	10	-1	-1	-1	-1	-1	-1	-1
1	9	0	5	10	6	8	4	7	-1	-1	-1	-1	-1	-1	-1
10	6	5	1	9	7	1	7	3	7	9	4	-1	-1	-1	-1
6	1	2	6	5	1	4	7	8	-1	-1	-1	-1	-1	-1	-1
1	2	5	5	2	6	3	0	4	3	4	7	-1	-1	-1	-1
8	4	7	9	0	5	0	6	5	0	2	6	-1	-1	-1	-1
7	3	9	7	9	4	3	2	9	5	9	6	2	6	9	-1
3	11	2	7	8	4	10	6	5	-1	-1	-1	-1	-1	-1	-1
5	10	6	4	7	2	4	2	0	2	7	11	-1	-1	-1	-1
0	1	9	4	7	8	2	3	11	5	10	6	-1	-1	-1	-1
9	2	1	9	11	2	9	4	11	7	11	4	5	10	6	-1
8	4	7	3	11	5	3	5	1	5	11	6	-1	-1	-1	-1
5	1	11	5	11	6	1	0	11	7	11	4	0	4	11	-1
0	5	9	0	6	5	0	3	6	11	6	3	8	4	7	-1
6	5	9	6	9	11	4	7	9	7	11	9	-1	-1	-1	-1
10	4	9	6	4	10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	10	6	4	9	10	0	8	3	-1	-1	-1	-1	-1	-1	-1
10	0	1	10	6	0	6	4	0	-1	-1	-1	-1	-1	-1	-1
8	3	1	8	1	6	8	6	4	6	1	10	-1	-1	-1	-1
1	4	9	1	2	4	2	6	4	-1	-1	-1	-1	-1	-1	-1
3	0	8	1	2	9	2	4	9	2	6	4	-1	-1	-1	-1
0	2	4	4	2	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	3	2	8	2	4	4	2	6	-1	-1	-1	-1	-1	-1	-1
10	4	9	10	6	4	11	2	3	-1	-1	-1	-1	-1	-1	-1
0	8	2	2	8	11	4	9	10	4	10	6	-1	-1	-1	-1
3	11	2	0	1	6	0	6	4	6	1	10	-1	-1	-1	-1
6	4	1	6	1	10	4	8	1	2	1	11	8	11	1	-1
9	6	4	9	3	6	9	1	3	11	6	3	-1	-1	-1	-1
8	11	1	8	1	0	11	6	1	9	1	4	6	4	1	-1
3	11	6	3	6	0	0	6	4	-1	-1	-1	-1	-1	-1	-1
6	4	8	11	6	8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
7	10	6	7	8	10	8	9	10	-1	-1	-1	-1	-1	-1	-1
0	7	3	0	10	7	0	9	10	6	7	10	-1	-1	-1	-1
10	6	7	1	10	7	1	7	8	1	8	0	-1	-1	-1	-1
10	6	7	10	7	1	1	7	3	-1	-1	-1	-1	-1	-1	-1
1	2	6	1	6	8	1	8	9	8	6	7	-1	-1	-1	-1
2	6	9	2	9	1	6	7	9	0	9	3	7	3	9	-1

7	8	0	7	0	6	6	0	2	-1	-1	-1	-1	-1	-1	-1
7	3	2	6	7	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	3	11	10	6	8	10	8	9	8	6	7	-1	-1	-1	-1
2	0	7	2	7	11	0	9	7	6	7	10	9	10	7	-1
1	8	0	1	7	8	1	10	7	6	7	10	2	3	11	-1
11	2	1	11	1	7	10	6	1	6	7	1	-1	-1	-1	-1
8	9	6	8	6	7	9	1	6	11	6	3	1	3	6	-1
0	9	1	11	6	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
7	8	0	7	0	6	3	11	0	11	6	0	-1	-1	-1	-1
7	11	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
7	6	11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	0	8	11	7	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	1	9	11	7	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	1	9	8	3	1	11	7	6	-1	-1	-1	-1	-1	-1	-1
10	1	2	6	11	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	2	10	3	0	8	6	11	7	-1	-1	-1	-1	-1	-1	-1
2	9	0	2	10	9	6	11	7	-1	-1	-1	-1	-1	-1	-1
6	11	7	2	10	3	10	8	3	10	9	8	-1	-1	-1	-1
7	2	3	6	2	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
7	0	8	7	6	0	6	2	0	-1	-1	-1	-1	-1	-1	-1
2	7	6	2	3	7	0	1	9	-1	-1	-1	-1	-1	-1	-1
1	6	2	1	8	6	1	9	8	8	7	6	-1	-1	-1	-1
10	7	6	10	1	7	1	3	7	-1	-1	-1	-1	-1	-1	-1
10	7	6	1	7	10	1	8	7	1	0	8	-1	-1	-1	-1
0	3	7	0	7	10	0	10	9	6	10	7	-1	-1	-1	-1
7	6	10	7	10	8	8	10	9	-1	-1	-1	-1	-1	-1	-1
6	8	4	11	8	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	6	11	3	0	6	0	4	6	-1	-1	-1	-1	-1	-1	-1
8	6	11	8	4	6	9	0	1	-1	-1	-1	-1	-1	-1	-1
9	4	6	9	6	3	9	3	1	11	3	6	-1	-1	-1	-1
6	8	4	6	11	8	2	10	1	-1	-1	-1	-1	-1	-1	-1
1	2	10	3	0	11	0	6	11	0	4	6	-1	-1	-1	-1
4	11	8	4	6	11	0	2	9	2	10	9	-1	-1	-1	-1
10	9	3	10	3	2	9	4	3	11	3	6	4	6	3	-1
8	2	3	8	4	2	4	6	2	-1	-1	-1	-1	-1	-1	-1
0	4	2	4	6	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	9	0	2	3	4	2	4	6	4	3	8	-1	-1	-1	-1
1	9	4	1	4	2	2	4	6	-1	-1	-1	-1	-1	-1	-1
8	1	3	8	6	1	8	4	6	6	10	1	-1	-1	-1	-1
10	1	0	10	0	6	6	0	4	-1	-1	-1	-1	-1	-1	-1
4	6	3	4	3	8	6	10	3	0	3	9	10	9	3	-1
10	9	4	6	10	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	9	5	7	6	11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

0	8	3	4	9	5	11	7	6	-1	-1	-1	-1	-1	-1	-1
5	0	1	5	4	0	7	6	11	-1	-1	-1	-1	-1	-1	-1
11	7	6	8	3	4	3	5	4	3	1	5	-1	-1	-1	-1
9	5	4	10	1	2	7	6	11	-1	-1	-1	-1	-1	-1	-1
6	11	7	1	2	10	0	8	3	4	9	5	-1	-1	-1	-1
7	6	11	5	4	10	4	2	10	4	0	2	-1	-1	-1	-1
3	4	8	3	5	4	3	2	5	10	5	2	11	7	6	-1
7	2	3	7	6	2	5	4	9	-1	-1	-1	-1	-1	-1	-1
9	5	4	0	8	6	0	6	2	6	8	7	-1	-1	-1	-1
3	6	2	3	7	6	1	5	0	5	4	0	-1	-1	-1	-1
6	2	8	6	8	7	2	1	8	4	8	5	1	5	8	-1
9	5	4	10	1	6	1	7	6	1	3	7	-1	-1	-1	-1
1	6	10	1	7	6	1	0	7	8	7	0	9	5	4	-1
4	0	10	4	10	5	0	3	10	6	10	7	3	7	10	-1
7	6	10	7	10	8	5	4	10	4	8	10	-1	-1	-1	-1
6	9	5	6	11	9	11	8	9	-1	-1	-1	-1	-1	-1	-1
3	6	11	0	6	3	0	5	6	0	9	5	-1	-1	-1	-1
0	11	8	0	5	11	0	1	5	5	6	11	-1	-1	-1	-1
6	11	3	6	3	5	5	3	1	-1	-1	-1	-1	-1	-1	-1
1	2	10	9	5	11	9	11	8	11	5	6	-1	-1	-1	-1
0	11	3	0	6	11	0	9	6	5	6	9	1	2	10	-1
11	8	5	11	5	6	8	0	5	10	5	2	0	2	5	-1
6	11	3	6	3	5	2	10	3	10	5	3	-1	-1	-1	-1
5	8	9	5	2	8	5	6	2	3	8	2	-1	-1	-1	-1
9	5	6	9	6	0	0	6	2	-1	-1	-1	-1	-1	-1	-1
1	5	8	1	8	0	5	6	8	3	8	2	6	2	8	-1
1	5	6	2	1	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	3	6	1	6	10	3	8	6	5	6	9	8	9	6	-1
10	1	0	10	0	6	9	5	0	5	6	0	-1	-1	-1	-1
0	3	8	5	6	10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
10	5	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	5	10	7	5	11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	5	10	11	7	5	8	3	0	-1	-1	-1	-1	-1	-1	-1
5	11	7	5	10	11	1	9	0	-1	-1	-1	-1	-1	-1	-1
10	7	5	10	11	7	9	8	1	8	3	1	-1	-1	-1	-1
11	1	2	11	7	1	7	5	1	-1	-1	-1	-1	-1	-1	-1
0	8	3	1	2	7	1	7	5	7	2	11	-1	-1	-1	-1
9	7	5	9	2	7	9	0	2	2	11	7	-1	-1	-1	-1
7	5	2	7	2	11	5	9	2	3	2	8	9	8	2	-1
2	5	10	2	3	5	3	7	5	-1	-1	-1	-1	-1	-1	-1
8	2	0	8	5	2	8	7	5	10	2	5	-1	-1	-1	-1
9	0	1	5	10	3	5	3	7	3	10	2	-1	-1	-1	-1
9	8	2	9	2	1	8	7	2	10	2	5	7	5	2	-1

1	3	5	3	7	5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	8	7	0	7	1	1	7	5	-1	-1	-1	-1	-1	-1	-1
9	0	3	9	3	5	5	3	7	-1	-1	-1	-1	-1	-1	-1
9	8	7	5	9	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
5	8	4	5	10	8	10	11	8	-1	-1	-1	-1	-1	-1	-1
5	0	4	5	11	0	5	10	11	11	3	0	-1	-1	-1	-1
0	1	9	8	4	10	8	10	11	10	4	5	-1	-1	-1	-1
10	11	4	10	4	5	11	3	4	9	4	1	3	1	4	-1
2	5	1	2	8	5	2	11	8	4	5	8	-1	-1	-1	-1
0	4	11	0	11	3	4	5	11	2	11	1	5	1	11	-1
0	2	5	0	5	9	2	11	5	4	5	8	11	8	5	-1
9	4	5	2	11	3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	5	10	3	5	2	3	4	5	3	8	4	-1	-1	-1	-1
5	10	2	5	2	4	4	2	0	-1	-1	-1	-1	-1	-1	-1
3	10	2	3	5	10	3	8	5	4	5	8	0	1	9	-1
5	10	2	5	2	4	1	9	2	9	4	2	-1	-1	-1	-1
8	4	5	8	5	3	3	5	1	-1	-1	-1	-1	-1	-1	-1
0	4	5	1	0	5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	4	5	8	5	3	9	0	5	0	3	5	-1	-1	-1	-1
9	4	5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	11	7	4	9	11	9	10	11	-1	-1	-1	-1	-1	-1	-1
0	8	3	4	9	7	9	11	7	9	10	11	-1	-1	-1	-1
1	10	11	1	11	4	1	4	0	7	4	11	-1	-1	-1	-1
3	1	4	3	4	8	1	10	4	7	4	11	10	11	4	-1
4	11	7	9	11	4	9	2	11	9	1	2	-1	-1	-1	-1
9	7	4	9	11	7	9	1	11	2	11	1	0	8	3	-1
11	7	4	11	4	2	2	4	0	-1	-1	-1	-1	-1	-1	-1
11	7	4	11	4	2	8	3	4	3	2	4	-1	-1	-1	-1
2	9	10	2	7	9	2	3	7	7	4	9	-1	-1	-1	-1
9	10	7	9	7	4	10	2	7	8	7	0	2	0	7	-1
3	7	10	3	10	2	7	4	10	1	10	0	4	0	10	-1
1	10	2	8	7	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	9	1	4	1	7	7	1	3	-1	-1	-1	-1	-1	-1	-1
4	9	1	4	1	7	0	8	1	8	7	1	-1	-1	-1	-1
4	0	3	7	4	3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	8	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	10	8	10	11	8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	0	9	3	9	11	11	9	10	-1	-1	-1	-1	-1	-1	-1
0	1	10	0	10	8	8	10	11	-1	-1	-1	-1	-1	-1	-1
3	1	10	11	3	10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	2	11	1	11	9	9	11	8	-1	-1	-1	-1	-1	-1	-1
3	0	9	3	9	11	1	2	9	2	11	9	-1	-1	-1	-1
0	2	11	8	0	11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

3	2	11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	3	8	2	8	10	10	8	9	-1	-1	-1	-1	-1	-1	-1
9	10	2	0	9	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	3	8	2	8	10	0	1	8	1	10	8	-1	-1	-1	-1
1	10	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	3	8	9	1	8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	9	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	3	8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

B Analytical Expression of a Catenoid

The equation for a catenoid is commonly given in parametric form as:

$$x = c\cosh\left(\frac{v}{c}\right)\cos u, \tag{B.1}$$

$$y = c\cosh\left(\frac{v}{c}\right)\sin u, \tag{B.2}$$

$$z = v \tag{B.3}$$

where v is a real variable ($v \in \mathbb{R}$), u is a variable within a range of 2π (such as $u \in \{-\pi, \pi\}$) and c is a non-zero constant ($c \in \mathbb{R}, c \neq 0$) (Akai et al. 2019). The equation of a circle in Cartesian coordinates is:

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{B.4}$$

for a circle with radius r that is centred at (x_0, y_0). If the centre of the circles is set to $(0, 0)$, the equation becomes $x^2 + y^2 = r^2$. As mentioned previously, a discretised dataset of pixel values ($\mathbf{P}$) that represents a catenoid can be considered a series of cocentric circles with varying radii for each layer. This can be shown by rearranging the parametric equations for a catenoid (B.1, B.2 and B.3) as follows:

Squaring B.1 and B.2 whilst substituting z for v based on B.3:

$$x^2 = c^2\cosh^2\left(\frac{z}{c}\right)\cos^2 u, \tag{B.5}$$

$$y^2 = c^2\cosh^2\left(\frac{z}{c}\right)\sin^2 u. \tag{B.6}$$

Adding B.5 and B.6 together:

$$x^2 + y^2 = c^2\cosh^2\left(\frac{z}{c}\right)\cos^2 u + c^2\cosh^2\left(\frac{z}{c}\right)\sin^2 u \tag{B.7}$$

which shows the catenoid can be considered a series of cocentric circles due to the LHS of $x^2 + y^2$. Simplifying from B.7 whilst using the trigonometric identity $\cos^2 u + \sin^2 u \equiv 1$ produces:

$$x^2 + y^2 = c^2\cosh^2\left(\frac{z}{c}\right)(\cos^2 u + sin^2 u) \tag{B.8}$$

$$= c^2\cosh^2\left(\frac{z}{c}\right) \tag{B.9}$$

which is the same form of the equation of a catenoid that is given in Equation 44. In turn, the radius of the circle that models a layer of the catenoid can be calculated by square-rooting B.9:

$$r(c, z) = c\cosh\left(\frac{z}{c}\right) \tag{B.10}$$

which is used when calculating the inverse resolution in Figure 13.

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.

Andrew, M., Bijeljic, B. & Blunt, M. J. (2014), 'Pore-scale contact angle measurements at reservoir conditions using x-ray microtomography', *Advances in Water resources* **68**, 24–31.

Arns, C. H., Bauget, F., Limaye, A., Sakellariou, A., Senden, T., Sheppard, A., Sok, R. M., Pinczewski, V., Bakke, S., Berge, L. I. et al. (2005), 'Pore scale characterization of carbonates using x-ray microtomography', *Spe Journal* **10**(04), 475–484.

Barbosa, J. L. M. & Colares, A. G. (1997), 'Stability of hypersurfaces with constant r-mean curvature', *Annals of Global Analysis and Geometry* **15**(3), 277–297.

Blunt, M. J. (2017), *Multiphase flow in permeable media: A pore-scale perspective*, Cambridge University Press.

Blunt, M. J., Bijeljic, B., Dong, H., Gharbi, O., Iglauer, S., Mostaghimi, P., Paluszny, A. & Pentland, C. (2013), 'Pore-scale imaging and modelling', *Advances in Water resources* **51**, 197–216.

Blunt, M. J., Lin, Q., Akai, T. & Bijeljic, B. (2019), 'A thermodynamically consistent characterization of wettability in porous media using high-resolution imaging', *Journal of colloid and interface science* **552**, 59–65.

Boot-Handford, M. E., Abanades, J. C., Anthony, E. J., Blunt, M. J., Brandani, S., Mac Dowell, N., Fernández, J. R., Ferrari, M.-C., Gross, R., Hallett, J. P. et al. (2014), 'Carbon capture and storage update', *Energy & Environmental Science* **7**(1), 130–189.

Börner, J. H., Herdegen, V., Repke, J.-U. & Spitzer, K. (2013), 'The impact of co2 on the electrical properties of water bearing porous media–laboratory experiments with respect to carbon capture and storage', *Geophysical Prospecting* **61**, 446–460.

Cazals, F. & Pouget, M. (2005), 'Estimating differential quantities using polynomial fitting of osculating jets', *Computer Aided Geometric Design* **22**(2), 121–146.

Chen, X. & Schmitt, F. (1992), Intrinsic surface properties from surface triangulation, *in* 'European Conference on Computer Vision', Springer, pp. 739–743.

Colding, T. H. & Minicozzi, W. P. (2006), 'Shapes of embedded minimal surfaces', *Proceedings of the National Academy of Sciences* **103**(30), 11106–11111.

Dbouk, T. & Drikakis, D. (2020), 'On respiratory droplets and face masks', *Physics of Fluids* **32**(6), 063303.

Desbrun, M., Meyer, M., Schröder, P. & Barr, A. H. (1999), Implicit fairing of irregular meshes using diffusion and curvature flow, *in* 'Proceedings of the 26th annual conference on Computer graphics and interactive techniques', pp. 317–324.

Dong, C.-s. & Wang, G.-z. (2005), 'Curvatures estimation on triangular mesh', *Journal of Zhejiang University-Science A* **6**(1), 128–136.

Euler, L. (1952), *Methodus inveniendi lineas curvas maximi minimive proprietate gaudentes sive solutio problematis isoperimetrici latissimo sensu accepti*, Vol. 1, Springer Science & Business Media.

Field, D. A. (1988), 'Laplacian smoothing and delaunay triangulations', *Communications in applied numerical methods* **4**(6), 709–712.

GNU (2021), 'GSL - GNU scientific library'. https://www.gnu.org/software/gsl/ [Accessed: 04-10-21].

Hazlett, R. (1995), 'Simulation of capillary-dominated displacements in microtomographic images of reservoir rocks', *Transport in porous media* **20**(1), 21–35.

Hoffman, D., Wei, F. S. & Karcher, H. (1993), 'Adding handles to the helicoid', *Bulletin of The American Mathematical Society* **29**(1), 77–84.

Huang, A., Summers, R. M. & Hara, A. K. (2005), Surface curvature estimation for automatic colonic polyp detection, *in* 'Medical Imaging 2005: Physiology, Function, and Structure from Medical Images', Vol. 5746, International Society for Optics and Photonics, pp. 393–402.

Ibekwe, A., Pokrajac, D. & Tanino, Y. (2020), 'Automated extraction of in situ contact angles from micro-computed tomography images of porous media', *Computers & Geosciences* **137**, 104425.

Iltis, G. C., Armstrong, R. T., Jansik, D. P., Wood, B. D. & Wildenschild, D. (2011), 'Imaging biofilm architecture within porous media using synchrotron-based x-ray computed microtomography', *Water Resources Research* **47**(2).

Jacobi, C. G. J. (1846), 'Über ein leichtes verfahren die in der theorie der säcularstörungen vorkommenden gleichungen numerisch aufzulösen*'.

Ji, Z., Liu, L. & Wang, G. (2005), A global Laplacian smoothing approach with feature preservation, *in* 'Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)', IEEE, pp. 6–pp.

Jones, T. R., Durand, F. & Desbrun, M. (2003), Non-iterative, feature-preserving mesh smoothing, *in* 'ACM SIGGRAPH 2003 Papers', Association for Computing Machinery, pp. 943–949.

Ju, T., Losasso, F., Schaefer, S. & Warren, J. (2002), Dual contouring of hermite data, *in* 'Proceedings of the 29th annual conference on Computer graphics and interactive techniques', pp. 339–346.

Khishvand, M., Alizadeh, A. & Piri, M. (2016), 'In-situ characterization of wettability and pore-scale displacements during two-and three-phase flow in natural porous media', *Advances in Water Resources* **97**, 279–298.

Klise, K. A., Moriarty, D., Yoon, H. & Karpyn, Z. (2016), 'Automated contact angle estimation for three-dimensional x-ray microtomography data', *Advances in water resources* **95**, 152–160.

Laplace, P. (1805), 'Traite de mecanique celeste (gauthier-villars, paris, 1839), suppl. au livre x, 1805 and 1806, resp', *Oeuvres compl* **4**.

Li, T., Schlüter, S., Dragila, M. I. & Wildenschild, D. (2018), 'An improved method for estimating capillary pressure from 3d microtomography images and its application to the study of disconnected nonwetting phase', *Advances in Water Resources* **114**, 249–260.

Lin, Q., Bijeljic, B., Berg, S., Pini, R., Blunt, M. J. & Krevor, S. (2019), 'Minimal surfaces in porous media: Pore-scale imaging of multiphase flow in an altered-wettability bentheimer sandstone', *Physical Review E* **99**(6), 063105.

Liu, T., Chen, M., Song, Y., Li, H. & Lu, B. (2017), 'Quality improvement of surface triangular mesh using a modified laplacian smoothing approach avoiding intersection', *PLoS One* **12**(9), e0184206.

Liu, X., Bao, H., Heng, P., Wong, T. & Peng, Q. (2001), Constrained fairing for meshes, *in* 'Computer Graphics Forum', Vol. 20, Wiley Online Library, pp. 115–123.

Lopes, A. & Brodlie, K. (2003), 'Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing', *IEEE Transactions on Visualization and Computer Graphics* **9**(1), 16–29.

Lorensen, W. E. & Cline, H. E. (1987), 'Marching cubes: A high resolution 3d surface construction algorithm', *ACM siggraph computer graphics* **21**(4), 163–169.

Lu, T. & Chen, F. (2012), 'Quantitative analysis of molecular surface based on improved marching tetrahedra algorithm', *Journal of Molecular Graphics and Modelling* **38**, 314–323.

Magid, E., Soldea, O. & Rivlin, E. (2007), 'A comparison of gaussian and mean curvature estimation methods on triangular meshes of range image data', *Computer Vision and Image Understanding* **107**(3), 139–159.

Max, N. (1999), 'Weights for computing vertex normals from facet normals', *Journal of graphics tools* **4**(2), 1–6.

Meeks III, W. & Pérez, J. (2011), 'The classical theory of minimal surfaces', *Bulletin of the American Mathematical Society* **48**(3), 325–407.

Meyer, M., Barr, A., Lee, H. & Desbrun, M. (2002), 'Generalized barycentric coordinates on irregular polygons', *Journal of graphics tools* **7**(1), 13–22.

Mittal, R., Ni, R. & Seo, J.-H. (2020), 'The flow physics of covid-19', *Journal of fluid Mechanics* **894**.

Newman, T. S. & Yi, H. (2006), 'A survey of the marching cubes algorithm', *Computers & Graphics* **30**(5), 854–879.

Ohtake, Y., Belyaev, A. & Bogaevski, I. (2001), 'Mesh regularization and adaptive smoothing', *Computer-Aided Design* **33**(11), 789–800.

Razdan, A. & Bae, M. (2005), 'Curvature estimation scheme for triangle meshes using biquadratic bézier patches', *Computer-Aided Design* **37**(14), 1481–1491.

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.

Rutishauser, H. (1966), 'The Jacobi method for real symmetric matrices', *Numerische Mathematik* **9**(1), 1–10.

Scanziani, A., Singh, K., Blunt, M. J. & Guadagnini, A. (2017), 'Automatic method for estimation of in situ effective contact angle from x-ray micro tomography images of two-phase flow in porous media', *Journal of colloid and interface science* **496**, 51–59.

Shabat, Y. B. & Fischer, A. (2015), 'Design of porous micro-structures using curvature analysis for additive-manufacturing', *Procedia CIRP* **36**, 279–284.

Shen, Y. & Barner, K. E. (2004), 'Fuzzy vector median-based surface smoothing', *IEEE Transactions on Visualization and Computer Graphics* **10**(3), 252–265.

Taubin, G. (1995), Curve and surface smoothing without shrinkage, *in* 'Proceedings of IEEE international conference on computer vision', IEEE, pp. 852–857.

Taubin, G., Zhang, T. & Golub, G. (1996), Optimal surface smoothing as filter design, *in* 'European Conference on Computer Vision', Springer, pp. 283–292.

Theisel, H., Rossi, C., Zayer, R. & Seidel, H.-P. (2004), Normal based estimation of the curvature tensor for triangular meshes, *in* '12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.', IEEE, pp. 288–297.

Thürmer, G. (2001), Smoothing normal vectors on discrete surfaces while preserving slope discontinuities, *in* 'Computer Graphics Forum', Vol. 20, Wiley Online Library, pp. 103–114.

Vollmer, J., Mencl, R. & Mueller, H. (1999), Improved Laplacian smoothing of noisy surface meshes, *in* 'Computer graphics forum', Vol. 18, Wiley Online Library, pp. 131–138.

Wei, M., Shen, W., Qin, J., Wu, J., Wong, T.-T. & Heng, P.-A. (2013), 'Feature-preserving optimization for noisy mesh using joint bilateral filter and constrained laplacian smoothing', *Optics and Lasers in Engineering* **51**(11), 1223–1234.

Wildenschild, D. & Sheppard, A. P. (2013), 'X-ray imaging and analysis techniques for quantifying pore-scale structure and processes in subsurface porous medium systems', *Advances in Water resources* **51**, 217–246.

Wildenschild, D., Vaz, C., Rivers, M., Rikard, D. & Christensen, B. (2002), 'Using x-ray computed tomography in hydrology: systems, resolutions, and limitations', *Journal of Hydrology* **267**(3-4), 285–297.

Xi, N., Sun, Y., Xiao, L. & Mei, G. (2021), 'Designing parallel adaptive laplacian smoothing for improving tetrahedral mesh quality on the gpu', *Applied Sciences* **11**(12), 5543.

Yang, J. & Zhou, Y. (2020), 'An automatic in situ contact angle determination based on level set method', *Water Resources Research* **56**(7), e2020WR027107.

Young, T. (1805), 'III. an essay on the cohesion of fluids', *Philosophical transactions of the royal society of London* **95**, 65–87.

Zankoor, A., Khishvand, M., Mohamed, A., Wang, R. & Piri, M. (2021), 'In-situ capillary pressure and wettability in natural porous media: Multi-scale experimentation and automated characterization using x-ray images', *Journal of Colloid and Interface Science* **603**, 356–369.