THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Robust intersection of hexahedral meshes and triangle meshes with applications in finite volume methods

Frida Svelander

Department of Mathematical Sciences
Division of Mathematics
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2016

Robust intersection of hexahedral meshes and triangle meshes with applications in finite volume methods
Frida Svelander

Department of Mathematical Sciences
Division of Mathematics
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg, Sweden
Telephone: +46 (0)31-772 10 00

Fraunhofer-Chalmers Research Centre for Industrial Mathematics
Chalmers Science Park
SE-412 88 Gothenburg, Sweden
Telephone: +46 (0)31-772 42 39
frida.svelander@fcc.chalmers.se

Printed by Chalmers Reproservice
Gothenburg, Sweden, 2016

Robust intersection of hexahedral meshes and triangle meshes with applications in finite volume methods

FRIDA SVELANDER
Department of Computational Engineering and Design
Fraunhofer-Chalmers Research Centre for Industrial Mathematics
and
Department of Mathematical Sciences
Division of Mathematics
Chalmers University of Technology and University of Gothenburg

Abstract

The topic of this thesis is the intersection of a structured hexahedral grid and one or more triangle meshes. The interest in the problem has arisen in connection with a finite volume method for simulation of conjugated heat transfer. In the particular finite volume method, axis-aligned hexahedra are used for the discretization of the simulation domain, and solids are represented by triangle meshes. The heat equation is discretized over the hexahedral cells. Special treatment is needed in the cells that are intersected by the surface of the solid. In these cells, the solid temperature is found after discretization of the heat equation over the solid part of the cell.

To implement the above, it is of great importance to find the geometry of the cut cells. Of particular interest is the solid volume fraction of a cell, and the solid area fraction of the cell faces. The solid volume fraction is defined as the fraction of the hexahedral cell that is intersected by the solid. Similarly, the solid area fraction is defined for each cell face as the fraction of the face that is intersected by the solid.

Two algorithms for calculation of the solid volume fraction and the solid

area fractions are presented. One algorithm is exact, and the other is approximate. The algorithms are extended to handle double surfaces, which is a common mesh degeneracy in engineering applications. A double surface is two layers of coplanar triangles, formed when the triangles are put on top of each other.

The handling of double surfaces is an extension of similar algorithms, which only handle non degenerate triangle meshes. This work is a step towards an algorithm that can be used with such meshes without preprocessing through a repair algorithm. A mesh repair method could be adopted, if available, but that is not always desirable since the existing repair algorithms could fail in removing the degeneracies without introducing unwanted side effects. This motivates the need for an algorithm that handles degenerate triangle meshes.

The algorithms are validated against a geometry from an industrial application, which includes a double surface. It is concluded that the exact algorithm is independent of cell size, while the approximate algorithm is second order accurate for the test case that has been studied. It is further concluded that the methods handle the major problems with double surfaces.

Finally, it is described how the algorithms are used in a finite volume framework for simulation of conjugated heat transfer.

Keywords: triangle-hex intersection, overlapping triangles, degenerate mesh, volume fraction, area fraction, cut cell, conjugated heat transfer.

Acknowledgements

I would like to thank my supervisors, Dr Andreas Mark at the Fraunhofer-Chalmers Research Centre and Professor Anders Logg at Chalmers University of Technology, for their guidance during this thesis work. Also, special thanks to Gustav Kettil, Dr Tomas Johnson, Associate Professor Fredrik Edelvik, and everyone else who helped me during my research at the Fraunhofer-Chalmers Research Centre.

Preface

This work has been carried out at the Fraunhofer-Chalmers Research Centre for Industrial Mathematics within the graduate program Advanced Engineering Mathematics. The work has resulted in one manuscript, which has been submitted for publication. It is appended in this thesis. The result of the paper is two algorithms for the volume of intersection of an axis-aligned rectangular hexahedron with one or more triangle meshes. The work has also been presented at the 19th European Conference for Mathematics in Industry (ECMI) in Santiago de Compostela (Spain), June 2016.

The algorithms described in the paper are designed to be used in a finite volume method for simulation of conjugated heat transfer. Such a simulation framework has been developed at the Fraunhofer-Chalmers Research Centre. The conjugated heat transfer solver has mainly been developed by Dr Andreas Mark and Dr Tomas Johnson. It is described in this thesis to put the geometric algorithms in the context they are used in.

This work was supported in part by the Sustainable Production Initiative and the Production Area of Advance at Chalmers.

Contents

1. Introduction

1.1 Outline

The outline of this thesis is as follows. The current chapter continues with the background and motivation to the work, presented in Section 1.2 and Section 1.3, respectively. In Section 1.4, a formal problem statement is given, and in Section 1.5 previous work is reviewed. In Chapter 2, some mathematical tools used to solve the problem are discussed. In Chapter 3, the proposed algorithms are summarized, and in Chapter 4 the algorithms are validated against an industrial test case. In Chapter 5, it is described how the algorithms are used in a finite volume framework for simulation of conjugated heat transfer. In Chapter 6, advantages and drawbacks of the algorithms are discussed. Finally, in Chapter 7, some conclusions are drawn. A manuscript describing the algorithms in more detail is found in the appendix.

1.2 Background

Researchers at the Fraunhofer-Chalmers Centre for Industrial Mathematics (FCC) have developed a multiphase flow simulation software called IBOFlow (Immersed Boundary Octree Flow Solver) [20]. It is built on a finite volume discretization of the Navier-Stokes equations, and an immersed boundary method [27, 28] is used for the modeling of solid-fluid interactions. In the particular finite volume method, axis-aligned hexahedra are used for the discretization of the simulation domain, and solids are represented by triangle meshes.

A solid and a fluid heat transfer solver is available for simulation of conjugated heat transfer. Both solvers are built on a finite volume discretization of the heat equation. The solid solver is used in the solid cells, and the fluid solver is used in the fluid cells. When the fluid temperature is solved for, the fluid and the solid are coupled through an immersed boundary condition [29]. When the solid temperature is solved for, the two phases are coupled through a heat flux condition. The heat flux coupling is done in the mixed cells, which are cells cut by the surface of the solid. The mixed cells are also called cut cells.

The solid-fluid heat flux in a mixed cell is found by summing up the fluxes through each triangle that intersects the cell, and the solid heat equation is discretized and solved on the solid part of the cell. For these reasons, it is of great importance to find the geometry of a cut cell. From the cut cell geometry, the solid volume fraction of the cell and the solid area fractions of the cell faces can be determined. The solid volume fraction is the fraction of the hexahedral cell that is intersected by the solid. Similarly, the solid area fraction is defined for each cell face as the fraction of the face that is intersected by the solid. That information is needed when the heat equation is discretized over a mixed cell. Also included in the cut cell geometry is the intersection between each triangle and the cell. That information is needed in the solid-fluid heat flux coupling.

1.3 Goal of the project

The goal of this thesis work is to develop a geometric algorithm for the intersection between an axis-aligned hexahedral cell and one or more triangle meshes. The triangle meshes represent the surface of one or more solids. Of particular interest is the solid volume fraction of a cut cell, and the solid area fractions of the cell faces.

The information generated by the algorithm is to be used in a finite volume method for simulation of conjugated heat transfer, as described in Section 1.2. The triangle meshes used in the simulation framework often originate from industrial applications, where mesh degeneracies such as large scale triangle overlaps are common. It is therefore of interest to make sure that the algorithms handle such overlaps. Large scale triangle overlaps will throughout this thesis also be called double surfaces. A double surface is formed when two or more coplanar triangles are put on top of each other. A formal statement of the

geometric problem is found in Section 1.4.

1.4 Mathematical problem formulation

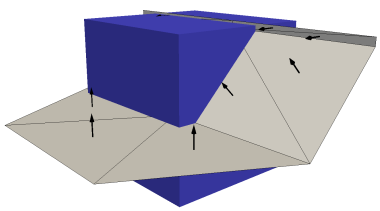
From a geometric point of view, the problem formulation is as follows. Let $\mathcal{T} = \{T_i\}_{i=1}^{n_T}$ be a triangle mesh, where $T_i \subset \mathbb{R}^3$, $i = 1, \ldots, n_T$, are triangles. The triangular mesh is oriented by the triangle normals $\{\mathbf{n}_i\}_{i=1}^{n_T}$ pointing out of $\mathcal{T}$. If $\mathcal{T}$ encloses a bounded volume, the interior of $\mathcal{T}$ is denoted $\Omega_{\mathcal{T}}$.

Let $\mathcal{C} \subset \mathbb{R}^3$ be a Cartesian cell, i.e. an axis-aligned cuboid consisting of its boundary $\partial\mathcal{C}$ and interior $\Omega_{\mathcal{C}}$. Then $\partial\mathcal{C} = \bigcup_{i=1}^{6} F_i$, where $\{F_i\}_{i=1}^{6}$, are the rectangular cell faces.

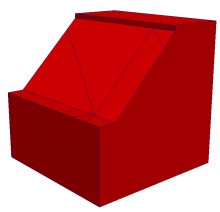
We are interested in the following:

- the **solid volume fraction** of $\mathcal{C}$, defined as the fraction of the cell $\mathcal{C}$ inside the triangle mesh $\mathcal{T}$,
- the **solid area fraction** of F_i, $i = 1, \ldots, 6$, defined as the fraction of the cell face F_i inside the triangle mesh $\mathcal{T}$.

A cell $\mathcal{C}$ intersecting a triangular mesh $\mathcal{T}$ is seen in Figure 1.1(a), and the part of the cell inside the triangle mesh is seen in Figure 1.1(b). The part of the cell inside the triangle mesh is given by $\overline{\Omega_{\mathcal{T}}} \bigcap \mathcal{C}$, where $\overline{\Omega_{\mathcal{T}}}$ denotes the closure of the interior of $\mathcal{T}$, and $\overline{\Omega_{\mathcal{T}}} \bigcap \mathcal{C}$ denotes the intersection between $\overline{\Omega_{\mathcal{T}}}$ and $\mathcal{C}$.



(a) A cell $\mathcal{C}$ intersecting a triangular mesh $\mathcal{T}$. The arrows represent triangle normals pointing out of $\Omega_{\mathcal{T}}$.



(b) The polyhedron $\overline{\Omega_{\mathcal{T}}} \bigcap \mathcal{C}$ resulting from intersecting $\mathcal{C}$ and $\mathcal{T}$.

Figure 1.1: Intersection between a triangle mesh $\mathcal{T}$ and a Cartesian cell $\mathcal{C}$.

We also want the algorithms to be robust against geometrically degenerate triangle meshes. In particular, we want them to handle double surfaces, since that is a common mesh degeneracy in engineering applications.

To define the concept of a double surface, let $\mathcal{T}_1 = \{T_i^1\}_{i=1}^{n_T^1}$ and $\mathcal{T}_2 = \{T_i^2\}_{i=1}^{n_T^2}$ be proper triangle meshes (without geometric degeneracies). Let $\mathcal{S}_1 \subset \mathcal{T}_1$ and $\mathcal{S}_2 \subset \mathcal{T}_2$ be such that $\bigcup_{T_i \in \mathcal{S}_1 \bigcup \mathcal{S}_2} T_i$ lie in a plane and $(\bigcup_{T_i \in \mathcal{S}_1} T_i) \bigcap (\bigcup_{T_i \in \mathcal{S}_2} T_i) \neq \emptyset$. Then $\mathcal{D} = (\bigcup_{T_i \in \mathcal{S}_1} T_i) \bigcap (\bigcup_{T_i \in \mathcal{S}_2} T_i)$ is a double surface. A typical double surface is seen in Figure 1.2.

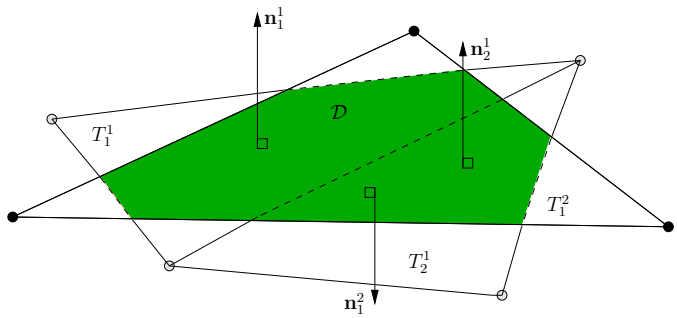


Figure 1.2: Triangles from different meshes overlap in a double surface. The double surface is the intersection of triangles T_1^1 and T_2^1 from one triangle mesh $\mathcal{T}_1$, and triangle T_1^2 from a second triangle mesh $\mathcal{T}_2$.

In a double surface, $\mathcal{T}_1$ and $\mathcal{T}_2$ could be two separate meshes, but the two meshes could also be merged to a single mesh $\mathcal{T}$. In the following we mainly consider double surfaces on a single mesh $\mathcal{T} = \mathcal{T}_1 \bigcup \mathcal{T}_2$.

1.5 Previous work

The problem formulated in Section 1.4 is interesting from a geometrical point of view, but is also used in more practical applications. In particular, it emerges as a problem in computational fluid dynamics (CFD) when cut cell methods are used to model the presence of solid bodies in a fluid. This is also where we find the most relevant previous work.

In CFD, and in particular in the finite volume method, the geometries in-

volved are commonly represented by discrete meshes, consisting of polygonal or polyhedral objects. One approach is to model the fluid by structured hexahedra [39], and represent the surface of solid objects by triangle meshes. The fluid mesh can be refined in the vicinity of a solid to better fit the surface of the solid [8, 9, 17, 39], resulting in unstructured or block-structured grid arrangements.

An alternative to Cartesian (axis-aligned) hexahedral meshes is body-fitted meshes [14, 37, 39], where the fluid cells are formed to fit the surface of the geometry. Another alternative is the cut cell method [1, 2, 15, 21, 23, 36, 38, 40]. In the cut cell method, a Cartesian fluid grid is used as a base grid, and new irregular cells are formed where the surface of the solid intersects the fluid. An important step in a cut cell method is to find the geometry of the cut cells. The same step is needed to calculate the solid area and volume fractions in Section 1.4, which makes previous research within cut cell methods relevant to this work. Figure 1.1(b) shows an example of a cut cell.

Approximate cut cells have been used for example by [36, 40]. In [36], a two-dimensional cut cell method is introduced, where the intersection between the mesh and the cell is approximated by a line. This approximation is motivated by the fact that a more complex intersection indicates that the grid is not fine enough to resolve the solid properly. In [40], Yang et al. presents a three-dimensional cut cell method where the intersection between the mesh and the cell is represented by a quadrilateral. They find the area of the quadrilateral and each cut face and use Gauss's divergence theorem to calculate the volume of the part of the cell located inside the triangle mesh.

In [1, 2] Aftosmis et al. present an approach for completely resolving the geometry of a cut cell in three dimensions. They introduce triangle-polygons, face-polygons and face-segments to describe the polyhedron that represents a cut cell. To construct the triangle-polygons they use the Sutherland-Hodgman algorithm [35] for clipping each triangle against the cell boundary. They mention that face-polygons are easily formed by connecting face-segments with the edges of the cut cell. Cieslak et al. [15] also present a cut cell method that preserves the real geometry by finding the exact cut out polygons and polyhedron. They find the face polygons through connectivity criteria such as common faces or common triangles.

In [15, 36, 40] little or nothing is mentioned about geometrically degenerate

triangle meshes. In engineering applications the triangle mesh often includes some artifact, such as hanging nodes (T-vertices), gaps, cracks, overlapping meshes, or overlapping triangles. These artifacts can emerge for example when a CAD surface is triangulated and used as a triangle mesh. If the degeneracies are removed before the mesh is given as input to a method that calculates the cut cell information, there is no need to modify the method to account for geometric artifacts.

Several methods for removing geometric degeneracies are described in the literature. They can in general be classified as either surface oriented or volumetric. Surface oriented methods such as [6, 19, 26] operate directly on the degenerate geometry. Volumetric methods such as [11, 22, 32] create an intermediate volumetric representation of the geometry that is used to create a new mesh without degeneracies. Both classes of methods have drawbacks [16]. Volumetric methods destroy connectivity structures of the input geometry and can lead to loss of model features. Surface oriented methods can fail in resolving certain types of artifacts and suffer from robustness problems for other types such as large scale triangle overlaps.

Aftosmis et al. [1, 2] adopt a volumetric approach to degenerate and overlapping triangle meshes. An alternative is to let the cut cell method handle geometric degeneracies, as is done for double surfaces in this thesis.

2. Mathematical tools

We here give a short overview of parts of the mathematical tools needed to solve the problem described in Section 1.4. We also discuss the intersection between a triangle mesh and an axis-aligned hexahedral cell. It is needed in the solid volume fraction algorithms, but is not in focus in the appended paper.

2.1 Volume and area of polyhedrons and polygons

The intersection between a triangle mesh and a hexahedral cell is a polyhedron, consisting of several polygonal faces. To calculate the area and volume fractions we need to be able to calculate the area of a polygon and the volume of a polyhedron. A formula for the volume of a polyhedron P can be derived from the divergence theorem. When the theorem is applied with the vector field $\mathbf{F}(x,\, y,\, z) = \dfrac{1}{3}(x,\, y,\, z)$ we get

$$V(P) = \frac{1}{3} \sum_i \mathbf{c}_i \cdot \hat{\mathbf{n}}_i \, A_i, \tag{2.1}$$

where $V(P)$ is the volume of the polyhedron P, $\mathbf{c}_i$ is the centroid, $\hat{\mathbf{n}}_i$ the unit normal and A_i the area of the i:th face S_i of the surface of P. To see this, note

that $\nabla \cdot \mathbf{F} = 1$ and

$$\begin{aligned} V(P) &= \int_P dV = \int_P \nabla \cdot \mathbf{F}\, dV = \int_{\partial P} \mathbf{F} \cdot \hat{\mathbf{n}}\, dS \\ &= \frac{1}{3} \sum_i \int_{S_i} (x,\, y,\, z) \cdot \hat{\mathbf{n}}_i\, dS_i \\ &= \frac{1}{3} \sum_i \hat{\mathbf{n}}_i \cdot \int_{S_i} (x,\, y,\, z)\, dS_i \\ &= \frac{1}{3} \sum_i \hat{\mathbf{n}}_i \cdot \mathbf{c}_i\, A_i. \end{aligned} \tag{2.2}$$

The area A and centroid $\mathbf{c} = (c_x,\, c_y)$ of a polygon with vertices $(x_0,\, y_0),\, \ldots,\, (x_{n-1},\, y_{n-1}),\, (x_n,\, y_n)$ are given by [7]

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i), \tag{2.3}$$

and

$$\begin{aligned} c_x &= \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i), \\ c_y &= \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i), \end{aligned} \tag{2.4}$$

where $(x_n,\, y_n) = (x_0,\, y_0)$.

2.2 Triangle-cell intersections

To calculate the area and volume fractions, we first need to find the intersection between the triangle mesh and the hexahedral cell. This step is not addressed in the appended paper, and is therefore discussed briefly in the following.

What we basically want to do is to intersect a number of triangles and an axis-aligned hexahedron in a robust way. By robustness we mean that we want to locate the points of intersection, we do not want to miss an intersection due to numerical imprecision, and we want related intersection tests to give consistent

results. The latter requirement could fail for example if an edge that is shared between two triangles is not represented consistently between the two triangles [18]. We also want unique intersections, meaning that an intersection point on a triangle edge is shared between the two triangles meeting at the edge, and that an intersection point in a triangle vertex is shared between all triangles that meet in the vertex.

The above robustness criteria makes the triangle-box intersection problem hard to solve. Determining whether a triangle and an axis-aligned box intersect or not is a well studied problem, discussed for example in [3]. Robustness issues are discussed by among others [18, 33]. It gets more complicated when the locations of the intersection points are also required. One approach to this is to perform a sequence of simpler intersection tests, where the triangle edges are intersected with the rectangular faces of the box, and the box edges are intersected with the triangle. Such line segment-rectangle and line segment-triangle intersection tests are common in several applications, often build upon the parametric representation of the geometric objects, and summarized for example in [18].

To get unique intersections, we have introduced vertices and edges that are shared between triangles. Similarly, we have introduced box vertices and box edges that are shared between the rectangular faces of the box. These joint edges and vertices are then used in a sequence of simpler intersection tests, as described above. Before the intersection points are calculated numerically, we have tried to discretely determine how many intersections there are by using coordinate comparisons similar to the outcodes described in [1,2]. The coordinate comparison approach is appealing since the cell is axis-aligned, and the x-, y- and z-coordinates can be considered separately. Neither are there any numerical issues with the coordinate comparisons, since no floating point calculations have to be performed.

3. Algorithms

Two algorithms have been developed and implemented to solve the problem in Section 1.4, one exact algorithm and one approximate algorithm. The two algorithms are summarized in this chapter, and presented in detail in the appended paper. The exact algorithm is summarized in Section 3.1, and the approximate algorithm is summarized in Section 3.2. In Section 3.3, we briefly describe how the algorithms handle triangle meshes including double surfaces. For a more thorough explanation, we refer to the appended paper.

3.1 Exact algorithm

In the exact algorithm, we use a methodology similar to [1, 2, 15] for extracting the polyhedron of intersection $\overline{\Omega_{\mathcal{T}}} \bigcap \mathcal{C}$ between the mesh and the cell. The area and volume fractions can then be calculated after the volume of the polyhedron and the area of the polyhedral faces are found. We note that the solid volume fraction α can be defined as

$$\alpha = \frac{V(\overline{\Omega_{\mathcal{T}}} \bigcap \mathcal{C})}{V(\mathcal{C})}, \tag{3.1}$$

where $V(X)$ denotes the volume of a subset X of $\mathbb{R}^3$. Similarly, the solid area fraction β_i of face F_i can be defined as

$$\beta_i = \frac{A(\overline{\Omega_{\mathcal{T}}} \bigcap F_i)}{A(F_i)}, \tag{3.2}$$

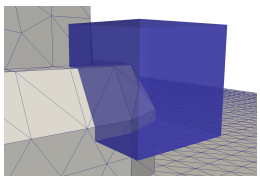
where $A(X)$ is the area of a two-dimensional object X. The area of the possibly non-convex polygon(s) $\overline{\Omega_{\mathcal{T}}} \bigcap F_i$, $i = 1, \ldots, 6$, in (3.2), are calculated from (2.3). The volume of the polyhedron $\overline{\Omega_{\mathcal{T}}} \bigcap \mathcal{C}$ in (3.1) is calculated from Gauss's divergence theorem as in (2.1).

The required steps can broadly be summarized as follows:

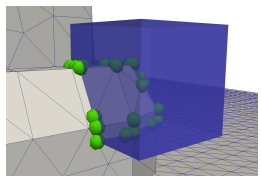
Main steps in the exact algorithm:

1. Intersect the triangle mesh and the cell
2. For each cell face, use the intersections from step 1 and the face vertices to find the face polygon(s) $\overline{\Omega_{\mathcal{T}}} \bigcap F_i$
3. For each triangle, use the intersections from step 1 to find the polygon of intersection between the triangle and the cell
4. For each cell face, use the face polygons from step 2 and the area of the face to calculate the area fraction of the face according to (3.2)
5. Use the face polygons from step 2 and the triangle polygons from step 3 to calculate the solid volume fraction from (2.1) and (3.1).

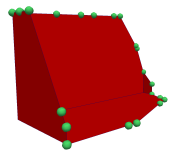
A sketch of the different steps in the exact method is found in Figure 3.1. For more details we refer to the appended paper.



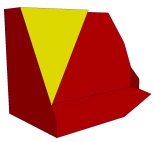
(a) A Cartesian cell $\mathcal{C}$ intersected by a triangle mesh $\mathcal{T}$.



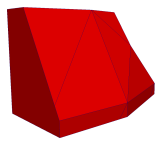
(b) The intersection points between the mesh and the cell are found.



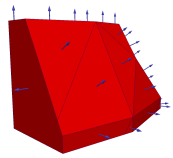
(c) For each face F_i of $\mathcal{C}$, the intersection points on that face are connected into polygons.



(d) For each triangle, the intersection points on that triangle are connected into a polygon.



(e) When all intersecting triangles are processed, the polyhedron $\overline{\Omega_{\mathcal{T}}} \bigcap \mathcal{C}$ is complete.



(f) The volume fraction is calculated according to (3.1) and (2.1), where normals, areas and centroids are required.

Figure 3.1: Description of how the polyhedron of intersection is found in the exact algorithm.

3.2 Approximate algorithm

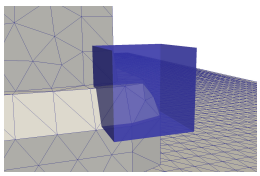
In the approximate algorithm, the polyhedron of intersection $\overline{\Omega_{\mathcal{T}}} \bigcap \mathcal{C}$ in Section 3.1 is not exactly resolved. Instead, $\overline{\Omega_{\mathcal{T}}} \bigcap \mathcal{C}$ is approximated by a simpler, convex, polyhedron P, with convex polygonal faces. This is done by fitting a total least squares plane [31] to the intersection between the cell $\mathcal{C}$ and the triangle mesh $\mathcal{T}$. The plane splits the cell in two parts, of which one is the approximate polyhedron of intersection.

The required steps can broadly be summarized as follows:

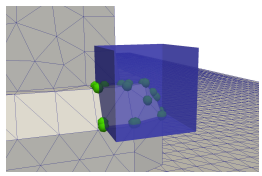
Main steps in the approximate algorithm:

1. Intersect the triangle mesh and the cell
2. Fit a total least squares plane to the intersections from step 1
3. Cut the cell in two by intersecting the least squares plane from step 2 with the cell
4. Identify the approximate polyhedron of intersection among the two parts in step 3
5. For each cell face, calculate the area of the intersection between the face and the polyhedron in step 4 and divide the result by the area of the face to calculate area fraction
6. Apply (2.1) to the approximate polyhedron of intersection and divide the result by the volume of the whole cell to calculate the solid volume fraction.

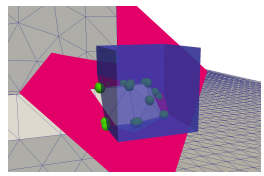
A sketch of the different steps in the approximate method is found in Figure 3.2. For more details we refer to the appended paper.



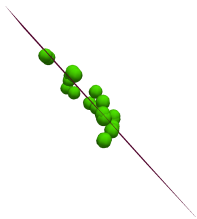
(a) A Cartesian cell $\mathcal{C}$ intersected by a triangle mesh $\mathcal{T}$.



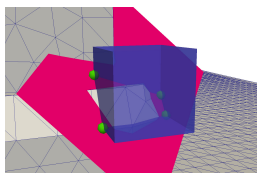
(b) The intersection points between the mesh and the cell are found.



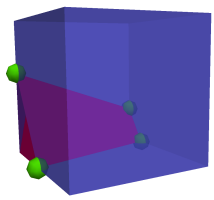
(c) A least squares plane is fitted to the intersection points.



(d) The least squares plane and the intersection points from another view angle.



(e) The intersection points between the plane and the cell are located.



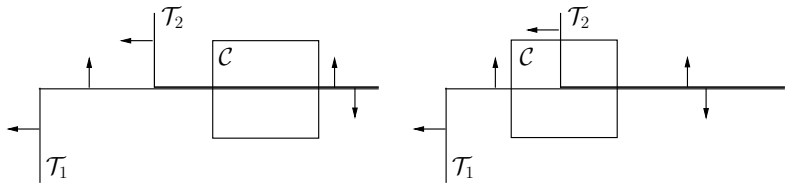
(f) The approximate polyhedron of intersection P is defined by the intersection points in (e) and the normal of the plane.

Figure 3.2: Description of how the polyhedron of intersection is found in the approximate algorithm.

3.3 Geometric degeneracy

As mentioned in Section 1.4, the algorithms should handle large scale triangle overlaps, or double surfaces. An example of a mesh including a double surface is seen in cross section in Figure 3.3. The mesh consists of two merged triangle meshes $\mathcal{T}_1$ and $\mathcal{T}_2$, with triangles from the two meshes overlapping in a double surface.

Both algorithms are modified to handle double surfaces. In the exact algorithm, the double surface could cause problems when the intersections on the cell faces are connected into polygons. A condition is therefore added at this stage of the algorithm, which makes sure that the correct polygons are created. In the approximate algorithm, the aim is to detect double surfaces before any intersection points are calculated. A double surface is present if all triangles



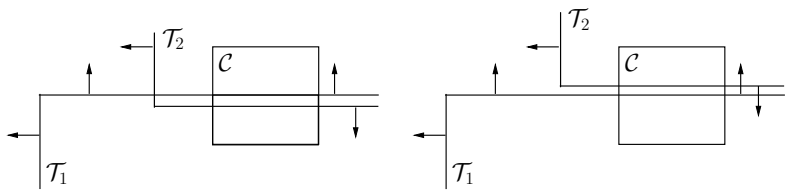
(a) A cell $\mathcal{C}$ intersecting a double surface.

(b) A cell $\mathcal{C}$ intersecting both a double surface and another part of the mesh.

Figure 3.3: Cross section of typical cases of intersection between a cell and a double surface.

intersecting the cell are coplanar and there are triangle normals pointing in opposite directions. For more details we refer to the appended paper.

An issue that is not discussed in the appended paper is how the algorithms handle the numerical representation of a double surface. When floating points are used, the triangles in a double surface could slightly overlap or be slightly separated, as in Figure 3.4. The algorithms proposed in the appended paper work even if the triangles in the double surface are slightly separated and not overlapping. However, an extra condition has to be added if small overlaps are to be handled. Problems with numerical overlaps have not been encountered in the studied test case, but we still discuss how they could be solved.



(a) A cell $\mathcal{C}$ intersecting an overlapping double surface.

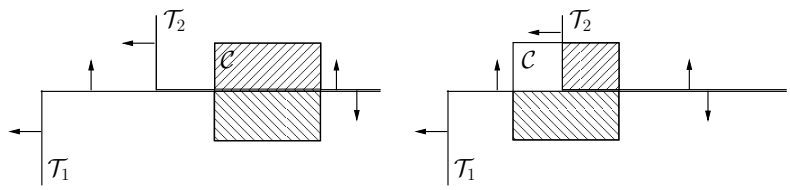
(b) A cell $\mathcal{C}$ intersecting a double surface with a small separation between the triangles.

Figure 3.4: Cross section of typical cases of intersection between a cell and a double surface. The double surface includes small overlaps or small separations between triangles.

To handle numerical overlaps, a limit ε for the maximal overlap allowed can be introduced and used to indicate when a double surface is found. Two intersection points at a separation distance smaller than ε are assumed to lie on a double surface if they also belong to triangles with opposite normals. Due to numerical issues, it will be necessary to have a limit on the oppositeness of normals. In the following we assume that two normals are opposite if $\pi - \theta < \delta$, where θ is the angle between the normals and δ is a prescribed limit.

In the approximate method, it is enough to use the limits to determine when two triangles with opposite normals are coplanar. In the exact method, the limits are used in the face polygon connection step to indicate whether a certain intersection point can be taken as the next vertex in the face polygon or not. The polygon connection step corresponds to step 2 in the numbered list in Section 3.1, and is exemplified in Figure 3.1(c). In the following, we discuss the treatment of double surfaces in the exact algorithm.

When a double surface intersects a cell, as in Figure 3.3, we wish to connect the intersection points on $\mathcal{T}_1$ to one polyhedron of intersection, and the intersection points on $\mathcal{T}_2$ to another polyhedron of intersection. How this is done is explained in the appended paper. The result after a correct polygon connection step is visualized in Figure 3.5, where the different striped patterns correspond to the different polyhedrons of intersection, one for $\mathcal{T}_1$ and one for $\mathcal{T}_2$.



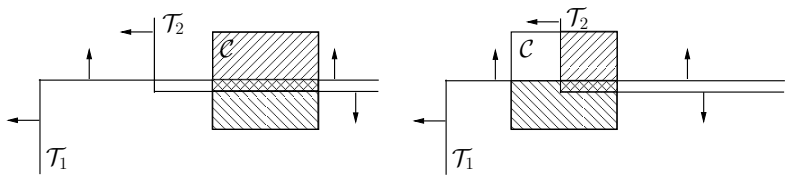
(a) A cell $\mathcal{C}$ intersecting a double surface.

(b) A cell $\mathcal{C}$ intersecting both a double surface and another part of the mesh.

Figure 3.5: Cross section of typical cases of intersection between a cell and a double surface. The intersections have been connected into the correct polyhedrons of intersection, one for $\mathcal{T}_1$ and one for $\mathcal{T}_2$.

For an overlapping double surface, the aim is to get to the result in Figure 3.6. The two meshes $\mathcal{T}_1$ and $\mathcal{T}_2$ still contribute with one polyhedron of intersection each, but there will now be an overlap between the polyhedrons. This

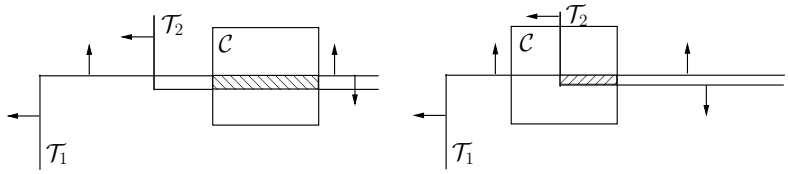
overlap corresponds to the overlap of the double surface. The area and volume of the overlap is counted twice in the solid area and solid volume fraction calculations. To get to the result in Figure 3.6, the face polygon connection step has to be modified, or the result will be as in Figure 3.7 where only a fraction of the correct polyhedrons of intersection is found.



(a) A cell $\mathcal{C}$ intersecting a double surface.

(b) A cell $\mathcal{C}$ intersecting both a double surface and another part of the mesh.

Figure 3.6: Cross section of typical cases of intersection between a cell and an overlapping double surface. The intersections have been connected into the correct polyhedrons of intersection, one for $\mathcal{T}_1$ and one for $\mathcal{T}_2$.



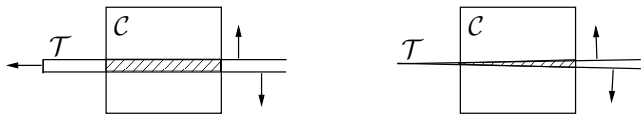
(a) A cell $\mathcal{C}$ intersecting a double surface.

(b) A cell $\mathcal{C}$ intersecting both a double surface and another part of the mesh.

Figure 3.7: Cross section of typical cases of intersection between a cell and a double surface. The intersections have in both cases been connected erroneously to only one polyhedron of intersection.

The problem with the cases in Figure 3.7 is that the algorithm can not distinguish the overlapping double surface from a thin or sharp-edged part of the mesh. Examples of thin and sharp-edged meshes are found in Figure 3.8(a) and Figure 3.8(b), respectively. A sharp edge is in this case formed when two triangles meet at an angle less than δ. For the cases in Figure 3.8, it is correct

to connect the two close intersections at the left and right cell faces, while it is wrong to do the same thing for the cases in Figure 3.7.



(a) A cell $\mathcal{C}$ intersecting a thin mesh $\mathcal{T}$.

(b) A cell $\mathcal{C}$ intersecting a sharp-edged mesh $\mathcal{T}$.

Figure 3.8: Meshes that can not be distinguished from an overlapping double surface in the method based on a maximal limit of the allowed overlap.

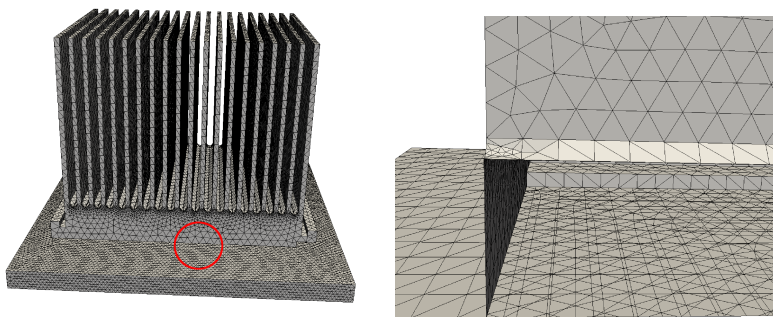
To avoid the problem demonstrated in Figure 3.7, the maximal overlap limit ε is used as an upper limit on how close two nearby vertices of a face polygon can be. If the distance between the current vertex and a candidate for the next vertex is closer than ε, and the two vertices belong to triangles with opposite normals, the candidate can not be taken as the next vertex. With this modification of the polygon connection algorithm, the result will be as in Figure 3.6. A consequence is that sharp-edged meshes and meshes thinner than ε can not be handled.

The above modification of the exact algorithm is necessary only if the double surface belongs to a single mesh $\mathcal{T} = \mathcal{T}_1 \bigcup \mathcal{T}_2$. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are handled separately, the double surface is resolved by finding the face polygons and polyhedrons of intersection for one mesh at a time. The area and volume fractions from the different meshes are then added.

For separate meshes $\mathcal{T}_1$ and $\mathcal{T}_2$, the overlaps that can be handled are not restricted to double surfaces. It is even possible to handle arbitrary overlaps, as long as it is reasonable to count the overlap volume and area twice.

4. Numerical results

The exact and approximate volume fraction algorithms have been validated and compared against a geometry including a double surface. The geometry (Figure 4.1), which is taken from an industrial application, represents a heat sink used to cool a CPU. The whole mesh is seen in Figure 4.1(a), and in Figure 4.1(b) we have zoomed in on the part of the mesh that is marked by a circle. The triangle pattern in the lower right part of the zoom-in reveals that the geometry contains a double surface.



(a) Test geometry representing a heat sink.

(b) Zoom in on the part in the left figure marked by a circle. The overlapping triangles indicate that there is a double surface.

Figure 4.1: The test case.

The algorithms have been implemented in the C++-based multiphase flow framework IBOFlow [20]. Grid convergence and CPU times were analysed and

compared. The fluid grid was stored in an octree and initially consisted of $1\,800$ cubical cells with side $4.16 \cdot 10^{-3}\,\mathrm{m}$. It was refined up to five times around the triangular mesh, resulting in a finest grid with $8\,438\,270$ cells of size $1.3{\cdot}10^{-4}\,\mathrm{m}$ or larger. At each refinement level, the smallest cell size was halved and eight new cells were formed. The test setup resulting after the fluid grid was refined three times is demonstrated in Figure 4.2. All computations were carried out on a 3.50GHz Intel Core i7 processor (5930K) and 64 GB of RAM (1066 GHz DDR4).

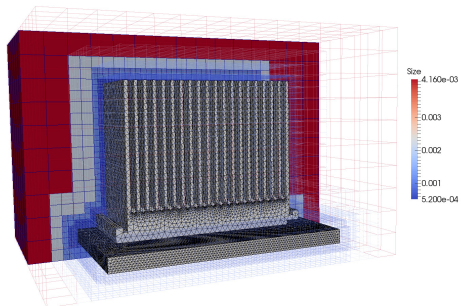
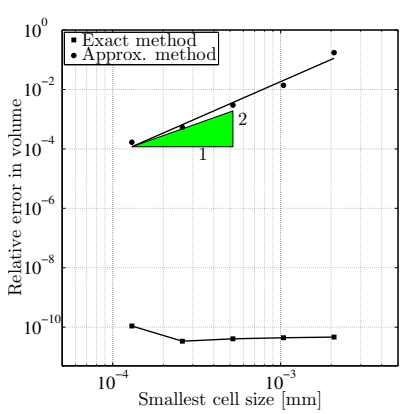
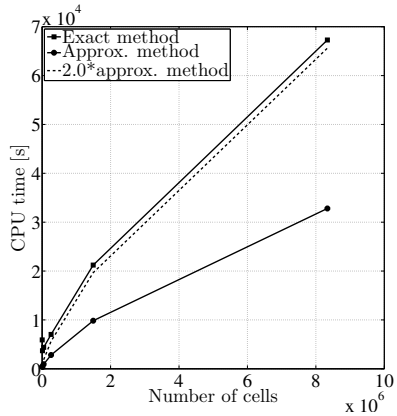


Figure 4.2: Test setup for volume fraction calculation. The colour of the cells represents cell size, where blue is smallest and red is largest.

Results of the grid study are presented in Figure 4.3(a). The total volume of the interior of the geometry was calculated by running the exact and approximate volume fraction algorithms for each cell. This was repeated for each refinement level. The calculated volume was compared to the real volume $1.45 \cdot 10^{-5}\,\mathrm{m}^3$, which was found by applying (2.1) to the whole triangle mesh. A plot of the CPU time against the number of fluid cells is seen in Figure 4.3(b). The times are the average results from 10 runs. In summary, the exact method is independent of cell size, while the approximate method is second order accurate. The exact method is a constant factor slower than the approximate method. The constant factor is close to 2 when the number of cells is large.



(a) Grid convergence of volume fraction algorithms run on the heat sink case.



(b) CPU time of volume fraction algorithms run on the heat sink case.

Figure 4.3: Results from grid and CPU time studies.

5. Applications in conjugated heat transfer

The volume fraction algorithms described in the appended paper and summarized in Chapter 3 are developed to be used together with an immersed boundary method [28, 29] for simulation of conjugated heat transfer with IBOFlow [20]. In this chapter it is described how the volume fraction is used in the discretization and numerical solution of the heat equations. In Section 5.1, the physics of heat transfer are discussed. In Section 5.2, the equations governing the temperature field in a fluid and a solid are presented. In Section 5.3 it is described how a finite volume approach is used to discretize the governing equations, and how the volume fraction enters in the discretized equations. Finally, in Section 5.4, a numerical example of forced convection cooling of a CPU and a heat sink is described.

5.1 Heat transfer

There are many situations in which it is of interest to describe or predict the temperature field in a physical system. Some well known examples are building insulation [4, 41] and cooling of electronics [13]. The temperature field in the building or electronic equipment can be described by equations derived from fundamental principles of physics. The equations governing the temperature field in a fluid or solid are described in Section 5.2. In this section the physical mechanisms that affect the temperature field are discussed.

Heat transfer is the mechanism that causes change in temperature, as heat is transferred from a location of higher temperature to a location of lower temperature. The mechanism can be classified into one of three main classes: conduction, convection, or radiation [30]. These modes of heat transfer are described briefly below.

5.1.1 Conduction

Conduction is the transfer of heat by direct surface contact. At a molecular scale, it occurs as molecules with higher kinetic energy collide with or vibrate against less kinetic molecules. The speed of the less kinetic molecules is increased and the speed of the molecules with higher kinetic energy is decreased, thus increasing the temperature in the cooler area and decreasing the temperature in the hotter area. Since the process is due to molecular collisions, it is not dependent on a bulk velocity. Conduction is therefore present in both solids and fluids. Heat conduction is in general more effective in solids due to the molecular structure [25].

Heat conduction can be described by Fourier's law

$$q_i = -k\frac{\partial T}{\partial x_i}, \tag{5.1}$$

where q_i is the heat flux in $\frac{W}{m^2}$, T is the temperature, and k is the thermal conductivity of the material.

5.1.2 Convection

In contrast to conduction, convection occurs as a result of movements of fluids. Hence there is no convection in solids. There are two main modes of convection: natural convection and forced convection.

Natural or free convection occurs in fluids when buoyancy forces cause fluid movement. This occurs in gravitational fields when the fluid has varying temperature and therefore varying density. The hot fluid is less dense and moves against the gravity [25]. This happens for example when a radiator is heating a room [24].

In forced convection it is assumed that temperature differences do not af-

fect the fluid motion. Instead, heat is transported from one place in the fluid to another by the fluid flow [25]. An example is air motion caused by a fan. Another example is the wind, which transports hot or cold air from one location to another.

Mixed convection is a combination of forced convection and natural convection.

5.1.3 Radiation

Thermal radiation is the transfer of energy by electromagnetic waves. All bodies at non-zero temperature emit energy in the form of thermal radiation [12]. A black body (a body which absorbs all incoming electromagnetic radiation) emits energy according to Stefan-Boltzmann's law:

$$E_b = \sigma T_b^4, \tag{5.2}$$

where E_b is the heat flow per unit surface area and second, $\sigma = 5.67 \cdot 10^{-8}$ is the Stefan-Boltzmann constant, and T_b is the temperature of the black body.

A surface that is not a black body has emissive power E satisfying

$$E = \varepsilon \sigma T^4, \tag{5.3}$$

where ε is the surface emissivity of the non-black body, and T is its temperature [39].

5.2 Governing equations

The equations governing the temperature field of an incompressible fluid are the continuity equation (5.4), the momentum equations (5.5), and the transport

equation for temperature (5.6), given by

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{5.4}$$

$$\rho \frac{\partial u_i}{\partial t} + \rho \frac{\partial (u_i u_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu \frac{\partial u_i}{\partial x_j}\right) - \rho_0 g_i \beta (T - T_0), \tag{5.5}$$

$$\rho c_p \frac{\partial T}{\partial t} + \rho c_p \frac{\partial (T u_j)}{\partial x_j} = \frac{\partial}{\partial x_j}\left(k \frac{\partial T}{\partial x_j}\right) + q_{ext}. \tag{5.6}$$

Here u_i is the fluid velocity field, T is the fluid temperature, ρ is the fluid density, p is the pressure, μ is the dynamic viscosity of the fluid, ρ_0 is the fluid density at the reference temperature T_0, g_i is the gravitational field, β is the coefficient of thermal expansion of the fluid, k is the thermal conductivity of the fluid, and q_{ext} denotes external heat sources. Together, (5.4) and (5.5) form the Navier-Stokes equations.

Boussinesq approximation is assumed for the coupling between the momentum equation (5.5) and the heat equation (5.6). The approximation is a model for fluid flow driven by density differences. The fluid density is considered constant, except in the gravitational fluid body force $f_i = \rho g_i$ where it is assumed that $\rho = \rho(T) = \rho_0 - \rho_0 \beta (T - T_0)$. Inserting this relation in the formula for the gravitational fluid body force, we get $f_i = \rho_0 g_i - \rho_0 g_i \beta (T - T_0)$. The contribution from $-\rho_0 g_i \beta (T - T_0)$ is included in (5.5) as an explicit source term, while $\rho_0 g_i$ is included in the definition of the pressure term $\dfrac{\partial p}{\partial x_i}$. Boussinesq approximation implies that $\rho = \rho_0$ in (5.5) and (5.6).

The temperature field in a homogeneous solid is also governed by the heat equation, but the advective term is zero since the velocity field vanishes inside the solid. The equation reads:

$$\rho_s c_{p,s} \frac{\partial T_s}{\partial t} = \frac{\partial}{\partial x_j}\left(k_s \frac{\partial T_s}{\partial x_j}\right) + q_{ext,s}, \tag{5.7}$$

where T_s is the solid temperature, ρ_s is the solid density, $c_{p,s}$ is the specific heat capacity of the solid, k_s is the thermal conductivity of the solid, and $q_{ext,s}$ denotes external heat sources.

The different terms in the Navier-Stokes equations (5.4)-(5.5) and the heat equations (5.6)-(5.7) can be related to the modes of heat transfer discussed in

Section 5.1. In Section 5.1, conduction was described by Fourier's law, $q_i = -k\frac{\partial T}{\partial x_i}$. In the heat equation it enters through the term $\frac{\partial}{\partial x_j}\left(k\frac{\partial T}{\partial x_j}\right)$. Natural convection was in Section 5.1 described by buoyancy forces in a gravitational field causing fluid to move from hot to cold areas. It enters Navier-Stokes equation through the Boussinesq approximation and the term $\rho_0 g_i \beta(T - T_0)$. Forced convection was in Section 5.1 described as the transport of heat by the bulk flow. It enters the heat equation for the fluid through the term $\rho c_p \frac{\partial (T u_j)}{\partial x_j}$. Finally, eventual radiation enters the heat equations through the source terms q_{ext} and $q_{ext,s}$.

When it is of interest to describe the heat transfer between a solid and a fluid, the two heat equations (5.6) and (5.7) have to be coupled. This is done by prescribing the heat flux between the solid and the fluid at the solid-fluid interface. The heat flux is due to conduction and radiation, since there is no velocity field at the interface that could transport heat by convection. The flux coupling is therefore described as

$$q_{tot} = q_{cond} + q_{rad}, \tag{5.8}$$

where q_{tot} is the total heat flux entering or leaving the solid, q_{cond} is the conductive heat flux, and q_{rad} is the radiative heat flux.

The conductive heat flux q_{cond} is governed by Fourier's law (5.1). An approximate way of describing the conductive heat transfer is Newton's law of cooling [10]. The law states that the heat flux is proportional to the temperature difference between the body and the surroundings. This can be written

$$q_{cond} \approx h\left(T_\infty - T_s\right), \tag{5.9}$$

where h is the heat transfer coefficient, T_∞ is the temperature of the surroundings, and T_s is the temperature of the solid body. The heat transfer coefficient depends on the geometry of the solid, the fluid, and the flow. The boundary layer thickness and the Nusselt number are two important factors. The boundary layer is the location close to the solid surface where there is a significant velocity gradient. The Nusselt number describes the relative importance of convective and conductive heat transfer. The heat transfer coefficient can be defined in different ways, but has to be chosen in conjunction with the ambient

temperature T_∞ [5].

5.3 Discretization and numerical solution

In IBOFlow, the governing equations in Section 5.2 are discretized and solved on a dynamically refined octree grid. The fluid is represented by an anisotropic Cartesian hexahedral grid, and the solids are represented by triangle meshes. Other solid representations are possible, but these are not considered here.

The finite volume method is used for the discretization of the governing equations. Roughly, each fluid cell is either marked as fluid or solid. A cell is marked as solid if it has a non-zero solid volume fraction. A cell in which both the solid volume fraction and the fluid volume fraction is non-zero is called a mixed cell. A two-dimensional visualization of the different kinds of cells is seen in Figure 5.1, where cell 3 is a standard fluid cell, and cell 4 is a standard solid cell. The rest of the cells are intersected by the solid triangle mesh $\mathcal{T}$, and have both a solid and a fluid part. These are the mixed cells.

The momentum equations, the continuity equation and the fluid heat equation are discretized on the fluid cells, while the heat equation for the solid is discretized on the solid cells. The mixed cells have both a solid and a fluid temperature. All fluid and solid properties are stored in the cell centers.

The governing equations are coupled through the velocity field u_i and the temperature T. In each time step the continuity equation (5.4) and the momentum equations (5.5) are first solved for the velocity field through the SIMPLEC [34] method. The fluid velocity at the IB is constrained to the velocity of the IB by an implicit immersed boundary condition [27, 28]. The velocity in the solid is set by a Dirichlet boundary condition to the velocity at the IB.

Next, and in the same time step, the heat equations are solved. The heat equation for the solid is first solved in the solid cells. After that, the heat equation for the fluid is solved. The mixed cells require special treatment, and this is where the solid area and volume fraction is used. In the mixed cells the control volume used in the finite volume discretization is the polyhedron of intersection between the triangle mesh and the cell, instead of the whole cell. The solid area fractions are thus used to ensure physical fluxes over the cell faces, and the solid volume fraction enters in the formula for the size of the control volume. There is also a heat flux from solid to fluid or from fluid to solid, which enters

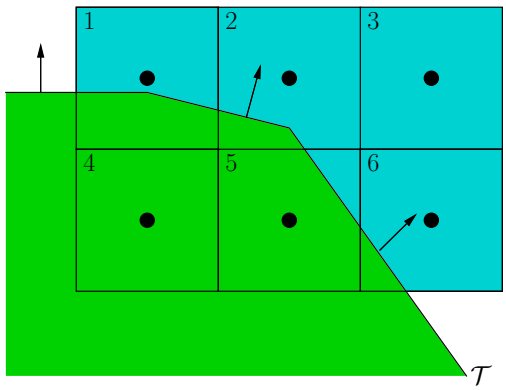


Figure 5.1: Two dimensional representation of a fluid grid intersected by a triangle mesh $\mathcal{T}$. The triangle mesh represents the boundary of a solid body. Cells 3 and 4 are standard fluid and standard solid cells respectively. The remaining cells have both a fluid and a solid volume fraction. They are the mixed cells.

and leaves through the triangles that intersect the cell. Referring to the volume fraction algorithm in the appended paper, heat flows between solid and fluid through the cell polygons.

In Figure 5.2 a two-dimensional visualization of a mixed cell is seen. In the discretized equations, heat flows through the cut faces and triangles in the positive or negative direction of the arrows. The direction of the heat flow is determined by the temperature difference according to Fourier's law or Newton's law of cooling.

If Fourier's law is used for the heat flux, direct numerical simulation (DNS) is used to calculate q_{cond} in (5.8). The temperature gradient in (5.1) is approximated by a forward difference using the fluid temperature interpolated to a point close to the solid surface and the fluid temperature at the surface. If Newton's law of cooling is used, the approximate relation in (5.9) is employed with the predefined heat transfer coefficient h and the reference temperature T_∞.

The solid area and volume fractions are used in a similar way in the momentum equations to ensure physical momentum fluxes.

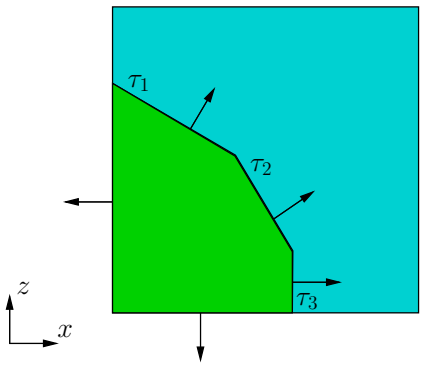


Figure 5.2: Two-dimensional representation of a mixed cell cut by the triangles τ_1, τ_2 and τ_3. Heat flows from the solid to the fluid through the triangles. Heat flows through the cell faces with non-zero solid area fraction to the solid in the neighboring cells.

5.4 Numerical example

The conjugated heat transfer solver has been used to simulate air cooling of a CPU by forced convection. The CPU was mounted on a rectangular block, and a fin heat sink was placed on top to increase the heat transfer from the CPU to the surrounding air. The block and the heat sink together form the geometry in Section 4, which was used to validate the solid volume fraction algorithms. The block, the heat sink and the CPU together are henceforth referred to as the geometry.

The simulation setup is seen in Figure 5.3. The rectangular box, which measures $0.25\,\mathrm{m} \times 0.03\,\mathrm{m} \times 0.025\,\mathrm{m}$, represents the simulation domain. The geometry was placed in the middle of the simulation domain. An anisotropic fluid grid was used, with base cell size $0.0033\,\mathrm{m} \times 0.0017\,\mathrm{m} \times 0.0005\,\mathrm{m}$. The grid was refined two times around the geometry. For each refinement level, the cell size was halved and eight new cells were formed, resulting in a total of $628\,445$ fluid cells.

The heat sink and the block were made of an aluminum alloy with density $2800\,\mathrm{kg/m^3}$, specific heat capacity $900\,\mathrm{J/(kgK)}$, and thermal conductivity $205\,\mathrm{W/(mK)}$. For the air, the corresponding values were $1161\,\mathrm{kg/m^3}$,

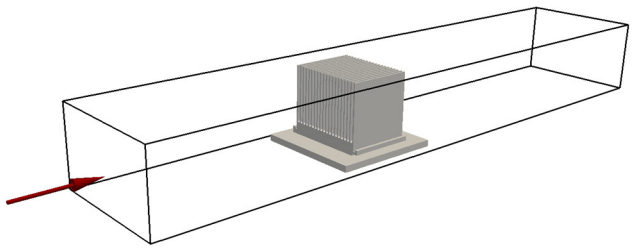


Figure 5.3: Simulation setup for forced convection air cooling of a CPU mounted on a rectangular block, and with a heat sink on top.

$1005\,\mathrm{J/(kgK)}$, and $0.026\,\mathrm{W/(mK)}$, respectively.

To model the heating of a CPU in work, an external heat source of $50\,\mathrm{W}$ was applied between the block and the heat sink. Air was let in with a speed of $10\,\mathrm{m/s}$ through a short side of the simulation domain, to simulate the presence of a blowing fan. In Figure 5.3, the air inlet and the direction of the flow is indicated by the arrow.

At the inlet, Dirichlet boundary conditions were used for the velocity and the temperature, and a zero Neumann boundary condition was used for the pressure. The temperature of the inlet air was set to $293\,\mathrm{K}$. An outlet was located at the short side opposite to the inlet. At the outlet, zero Neumann conditions were used for the velocity and the temperature, and a zero Dirichlet boundary condition was used for the pressure. For the remaining boundaries, symmetry conditions (zero Neumann) were used for the temperature and the pressure, and the no-slip boundary condition was used for the velocity.

A steady state artificial time-stepping method was used. The resulting temperature field in the solid and a slice of the fluid is illustrated in Figure 5.4. As seen from the figure, the solid temperature is highest close to the heated CPU, which is located between the block and the heat sink. It is also seen how heat is spread from the source to the block and the heat sink, where heat is exchanged with the surrounding air. The fins are cooled, and the surrounding air is heated and convected by the flow towards the outlet. In this way, overheating of the CPU can be avoided.

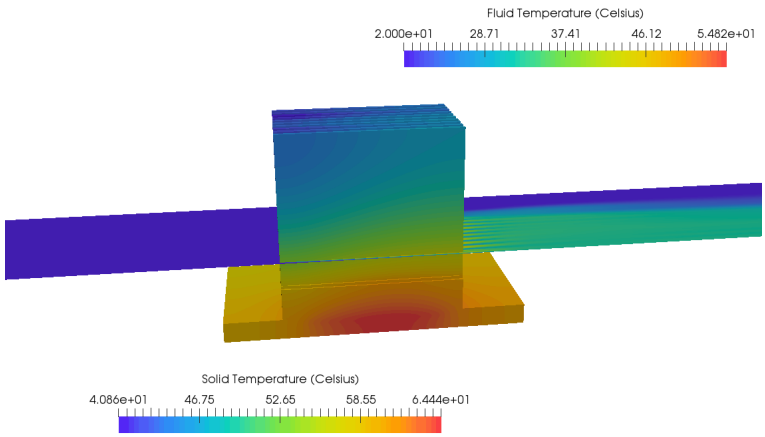


Figure 5.4: Temperature field in the solid and the fluid, resulting from the conjugated heat transfer simulation.

6. Discussion

The advantages and drawbacks of the implemented algorithms are here discussed. We also consider the robustness problems in the triangle-cell intersection routine from Section 2.2, and briefly discuss the extra condition introduced in Section 3.3 to handle overlapping double surfaces. We start the discussion by addressing the robustness issue.

When intersecting the triangle mesh and the cell, the aim is to use a robust routine, as noted in Section 2.2. However, the intersection method we have discussed and used is not completely robust. To ensure robustness, an alternative would be to use exact arithmetics, interval arithmetics or other similar approach. For a discussion of exact arithmetics and interval arithmetics, see for example [33]. We have not done this, but instead tried to as far as possible discretely determine how many intersections there are and where they are located. Still, there is more work to be done to make the intersection routine completely robust. It would be interesting to study the intersection routine further and see how accurately the intersection points can be located without introducing numerical errors. This could possibly include use of interval arithmetics.

If the intersections are not calculated correctly, we might not be able to find the correct faces of the polyhedron of intersection in Section 3.1. This might cause the polyhedron of intersection to be degenerate. For example, a polyhedral face could be missing, which would create a hole in the surface of the polyhedron. This is a problem, since formula (2.1) for the volume of a polyhedron is sensitive to such geometrical degeneracies. How the polyhedral faces are found is discussed in more detail in the appended paper.

The approximate algorithm, which is not as sensitive to numerical robust-

ness issues, was therefore implemented. It is more robust against numerical issues since the intersection between the triangle mesh and the cell is approximated by a least squares plane before the faces of the approximate polyhedron of intersection are found. A missed or erroneously calculated intersection point will make the approximating plane less accurate, which affects the size and shape of the approximate polyhedron of intersection, but it does not make it degenerate. The approximate algorithm is also easier to implement. Moreover, it has been shown to be faster than the exact algorithm, and second order accurate for the test case in Section 4. Second order accuracy is enough in many applications.

The approximate algorithm is not as accurate for double surfaces as the exact algorithm, as shown in the appended paper. Another advantage of the exact algorithm is its exactness, as demonstrated by the convergence results in Section 4. Also, since the exact algorithm is only a constant factor slower than the approximate algorithm, it is likely to be preferred.

The convergence result shows that the occurrence of numerical issues is overall negligible. However, it can be of importance that the solid volume and area fractions in individual cells are accurate as well. This could fail due to the previously discussed numerical issues. Our solution to this problem is to accept potential robustness problems at the intersection stage, and use the approximate volume fraction algorithm if the exact algorithm should fail.

Finally, we mention something about the overlap of double surfaces, which was discussed in Section 3.3. A drawback of the proposed solution is that meshes including sharp edges or thin parts can not be handled, since the polygon connection algorithm can not distinguish these details from an overlapping double surface. On the other hand, overlaps due to numerical imprecision should be small in comparison to the size of the mesh, unless the details of the mesh are very fine. There could be other reasons to discuss if meshes with sharp edges or thin parts are valid as well. It would be interesting to investigate if there is a discrete solution to the problem, in contrast to the proposed numeric one. There could be a discrete constraint on the order in which the intersections are connected to polygons that solves the problem. A clever choice of the start vertex might also be a possible solution.

7. Conclusion

In this thesis work, two algorithms for calculation of the solid volume fraction and solid area fractions of an axis-aligned heaxahedral cell intersected by a triangle mesh have been presented. The triangle mesh, which is allowed to include double surfaces, represents the surface of a solid. The first algorithm is in principle exact, and the second algorithm is based on a least squares plane fit to the intersection between the triangle mesh and the cell.

The algorithms have been implemented in the multiphase flow framework IBOFlow [20], where they are used in a finite volume method for simulation of conjugated heat transfer. The solid volume fraction describes how much of each fluid cell that is intersected by the solid. Similarly, the solid area fractions describe how much of each cell face that is intersected by the solid. This information is needed in the conjugated heat transfer solver when the heat equation is discretized over a cell that is intersected by the triangle mesh.

It is concluded that the exact algorithm is independent of cell size, while the approximate algorithm is second order accurate for the test case in study. Besides being more accurate, the exact algorithm is better at handling double surfaces. The handling of double surfaces is an extension of similar algorithms, which only handle non degenerate triangle meshes. The triangle meshes used in engineering applications are often degenerate, and this work is a step towards an algorithm that can be used with such meshes without preprocessing through a repair algorithm. A mesh repair method could be adopted, but that is not always desirable since the existing repair algorithms could fail in removing the degeneracies without introducing unwanted side effects. This motivates the need for an algorithm that handles degenerate triangle meshes.

The approximate algorithm is a constant factor faster than the exact algorithm, and more robust against numerical imprecisions. The exact algorithm could fail due to numerical robustness problems at the stage of intersecting the triangle mesh and the cell. These problems are avoided in the approximate algorithm by approximating the intersection by a least squares plane.

The aim is to use an intersection routine that is numerically robust, though the currently used intersection routine does not completely fulfill this requirement. It would be interesting to study the intersection routine to see if robustness can be ensured. Meanwhile, to assure robustness when the exact algorithm is used, the approximate algorithm is utilized in case of failure.

As a next step towards an algorithm that is robust against geometric degeneracies, the exact algorithm could be extended to account for overlapping triangle meshes and hanging nodes. Overlaps between meshes could be found by handling each mesh separately. To solve the problem with hanging nodes the triangle edges would need to have subedges, and these would have to be accounted for in the polygon connection algorithms.

Bibliography

[1] Aftosmis, M.J., Berger, M.J., Melton, J.E.: Robust and Efficient Cartesian Mesh Generation for Component-Based Geometry. AIAA Journal 36(6), 952–960 (1998)

[2] Aftosmis, M.J., Berger, M.J., Melton, J.E.: Adaptive Cartesian Mesh Generation. In: Weatherill, N.P., Soni, B.K., Thompson, J.F. (eds.) Handbook of Grid Generation, pp. 22-1–22-21. CRC Press, Boca Raton (1998)

[3] Akenine-Möller, T.: Fast 3D Triangle-Box Overlapping Test. Journal of Graphics Tools 6(1), 29–33 (2001)

[4] Asan, H., Sancaktar, Y.S.: Effects of Wall's Thermophysical Properties on Time Lag and Decrement Factor. Energy and Buildings 28, 159–166 (1998)

[5] Azar, K., Tavassoli, B. (eds): Understanding Heat Transfer Coefficient. Qpedia Thermal eMagazine 1(1-12), 32–35 (2008)

[6] Barequet, G., Sharir, M.: Filling gaps in the boundary of a polyhedron. Computer-Aided Geometric Design 12(2), 207–229 (1995)

[7] Bashein, G., Detmer, P.R.: Centroid of a polygon. In: P.S. Heckbert (ed.) Graphics Gems IV, pp. 3–6. Academic, New York (1994)

[8] Berger, M.J., Oliger, J.: Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. Journal of Computational Physics 53, 485–512 (1984)

[9] Berger, M.J., Colella, P.: Local Adaptive Mesh Refinement for Shock Hydrodynamics. Journal of Computational Physics 82, 64–84 (1989)

[10] Besson, U.: The History of the Cooling Law: When the Search for Simplicity can be an Obstacle. Science and Education 21(8), 1085–1110 (2012)

[11] Bischoff, S. Pavic, D., Kobbelt, L.: Automatic restoration of polygon models. Transactions on Graphics 24(4), 1332–1352 (2005)

[12] Blundell, S., Blundell, K.: Concepts in Thermal Physics. Oxford University Press, New York (2006)

[13] Çengel, A. J., Ghajar, A. J.: Cooling of Electronic Equipment (web chapter). In: Heat and Mass Transfer: Fundamentals and Applications (5th ed.). McGraw Hill Education, New York (2015). Retrieved from highered.mheducation.com/sites/dl/free/0073398187/835451/Chapter15.pdf. Accessed 8 July 2016.

[14] Chesshire, G., Henshaw, W.D.: Composite Overlapping Meshes for the Solution of Partial Differential Equations. Journal of Computational Physics, 1–64 (1990)

[15] Cieslak, S., Ben Khelil, S., Choquet, I., Merlen, A.: Cut cell strategy for 3-D blast waves numerical simulations. Shock Waves 10, 421–429 (2001)

[16] Botsch, M., Pauly, M., Kobbelt, L., Alliez, P., Levy, B.: Geometric modeling based on polygonal meshes. http://lgg.epfl.ch/publications/2008/botsch_2008_GMPeg.pdf (2007). Accessed 27 March 2016

[17] De Zeeuw, D., Powell, K.G.: An Adaptively Refined Cartesian Mesh Solver for the Euler Equations. Journal of Computational Physics 104, 56–68 (1993)

[18] Ericson C.: Real-Time Collision Detection. Morgan Kaufmann, Burlington, Massachusetts (2004)

[19] Guéziec, A., Taubin, G., Lazarus, F., Horn, B.: Cutting and stitching: Converting sets of polygons to manifold surfaces. IEEE Transactions on Visualization and Computer Graphics 7(2), 136–151 (2001)

[20] IBOFlow. http://www.ipsiboflow.com. Accessed 30 May 2016

[21] Ingram, D.M., Causon, D.M., Mingham, C.G.: Developments in Cartesian Cut Cell Methods. Mathematics and Computer Simulations 61, 561–572 (2003)

[22] Ju, T.: Robust repair of polygonal models. ACM Transactions on Graphics (TOG) 23(3) 888–895 (2004)

[23] Kirkpatrick, M.P., Armfield, S.W., Kent, J.H.: A representation of curved boundaries for the solution of the NavierâĂŞStokes equations on a staggered three-dimensional Cartesian grid. Journal of Computational Physics 184, 1–36 (2003)

[24] Lankhorst, A.M., Hoogendoorn, C.J.: Numerical Computation of High Rayleigh Number Natural Convection and Prediction of Hot Radiator Induced Room Air Motion. Applied Scientific Research 47, 301–322 (1990)

[25] Lienhard IV, J.H., Lienhard V, J.H.: A Heat Transfer Textbook 4. Phlogiston Press, Cambridge, Masssachusetts (2015)

[26] Liepa, P. : Filling holes in meshes. Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing 200–205 (2003)

[27] Mark, A., van Wachem, B.: Derivation and Validation of a Novel Implicit Second Order Accurate Immersed Boundary Method. Journal of Computational Physics 227(13), 6660–6680 (2008)

[28] Mark, A., Rundqvist, R., Edelvik, F.: Comparison Between Different Immersed Boundary Conditions for Simulation of Complex Fluid Flows. Fluid Dynamics and Material Processing 7(3), 241–258 (2011)

[29] Mark, A., Svenning, E., Edelvik, F.: An Immersed Boundary Method for Simulation of Flow with Heat Transfer. International Journal of Heat and Mass Transfer 56, 424–435 (2013)

[30] McCormack, P.: Physical Fluid Dynamics. Academic Press, New York (1973)

[31] Nievergelt, Y.: Total Least Squares: State-of-the-Art Regression in Numerical Analysis. SIAM Review 36(2), 258–264 (1994)

[32] Nooruddin, F.S., Turk, G.: Simplification and repair of polygonal models using volumetric techniques. IEEE Transactions on Visualization and Computer Graphics 9(2), 191–205 (2003)

[33] Schirra, S.: Robustness and Precision Issues in Geometric Computation. In: Sack, J.-R., Urrutia, J. Handbook of Computational Geometry, pp. 597–632 Elsevier (1997)

[34] Van Dormal, J.P., Raithby, G.D.: Enhancements of the Simple Method for Predicting Incompressible Fluid Flows. Numerical Heat Transfer 7, 147–163 (1984)

[35] Sutherland, I.E., Hodgman, G.W.: Reentrant polygon clipping. Comm of the ACM 17(1), 32–42 (1974)

[36] Quirk, J.J.: An alternative to unstructured grids for computing gas dynamic flows around arbitrarile complex two-dimensional bodies. Computer Fluids 23, 125–142 (1994)

[37] Thompson, J.F., Thames, F.C., Mastin, C.W.: Automatic Numerical Generation of Body-Fitted Curvilinear Coordinate System for Field Containing Any Number of Arbitrary Two-Dimensional Bodies. Journal of Computational Physics 15, 299–319 (1974)

[38] Tucker, P.G., Pan, Z.: A Cartesian Cut Cell Method for Incompressible Viscous Flow. Applied Mathematics Modelling 24, 591–606 (2000)

[39] Versteeg, H.K., Malalasekera, W.: An Introduction to Computational Fluid Dynamics: the Finite Volume Method. Pearson Education (2007)

[40] Yang, G., Causon, D.M., Ingram, D.M.: Calculation of compressible flows about complex moving geometries using a three-dimensional Cartesian cut cell method. International Journal for Numerical Methods in Fluids 33, 1121–1151 (2000)

[41] Zhang, Y., Chen, Q., Zhang, Y., Wang, X.: Exploring Buildings' Secrets: The Ideal Thermophysical Properties of a Building's Wall for Energy Conservation. International Journal of Heat and Mass Transfer 65, 265–273 (2013)