View metadata, citation and similar papers at core.ac.uk
brought to you by CORE

Acta Polytechnica **57**(2):105–115, 2017
doi:10.14311/AP.2017.57.0105

© Czech Technical University in Prague, 2017
available online at http://ojs.cvut.cz/ojs/index.php/ap

IMPLEMENTATION OF A 3D VOF TRACKING ALGORITHM BASED ON BINARY SPACE-PARTITIONING

Filip Kolařík*, Bořek Patzák

CTU in Prague, Faculty of Civil Engineering, Department of mechanics, Thákurova 7, 166 29 Praha 6; Czech Republic

* corresponding author: filip.kolarik@fsv.cvut.cz

Abstract. The paper focuses on modeling free surface flow. The interface is modeled using the Volume-Of-Fluid method, where the advection of volume fractions is treated by a purely geometrical method. The novelty of the work lies in the way that it incorporates Binary Space-Partitioning trees for computing the intersections of polyhedra. Volume-conserving properties and shape-preserving properties are presented on two benchmarks and on a simulation of the famous broken dam problem.

Keywords: free surface; two-fluid flow; VOF; Binary Space-Partitioning.

1. Introduction

Free surface flows appear in a wide range of industrial applications including molten metal production in steel processing [1], resin-infusion processes in structural manufacturing [2], water waves in ship hydrodynamics [3], and fresh concrete casting in civil engineering [4]. The development of numerical techniques for describing the evolution of a free surface can be tracked back to the early 1980s. A good review of the early development of this field can be found in [5]. Since that time, a number of different approaches and methods have been proposed.

The existing methods can be divided into so-called interface-tracking and interface-capturing methods [6]. In the interface tracking techniques, the underlying discretization follows the interface in a Lagrangian manner (Lagrangian formulation is typically employed). Because of the large mesh distortions during the flow, the mesh needs to be updated as the interface evolves. The need for frequent re-meshing can sometimes involve an excessive expense, especially in the case of complex 3D flows. This disadvantage is balanced by the natural representation of the interface.

Interface-capturing methods are usually formulated in an Eulerian context, where the computations are performed on a fixed mesh. The free surface flow is then realized as the flow of two immiscible fluids, where one is the fluid of interest (the reference fluid) and the other represents the ambient air. The interface between the fluids is determined with the help of a suitably chosen (scalar) interface function, defined on the underlying mesh. A direct consequence of this assumption is that the interface is captured within the resolution of the underlying mesh. Fundamental examples of interface-capturing methods are the Level Set method [7–9], the Volume-Of-Fluid (VOF) method [5, 10], and a combination of these methods CLSVOF [11]. In Level Set methods, the interface is typically represented as a zero contour of the interface signed distance function, which is defined over the whole computational domain. The interface is then tracked by solving the transport equation which is of hyperbolic type, and therefore introduces some computational difficulties. In Volume-Of-Fluid, the volume fractions of the representative fluid are tracked. Many different numerical schemes have been developed in this area. Most of them are based on Finite Difference schemes [8], or employ stabilized Finite Elements (SUPG), see [12, 13]. The work presented here employs the Volume-Of-Fluid method which, within the interface capturing methods, belongs to the family of so-called volume tracking methods [5]. Here, instead of tracking the interface itself according to the transport equation (as in the Level-Set method), we track the material volume fraction inside each cell along the streamlines. The interface is then reconstructed by projecting these advected volumes onto the original mesh. The volume fraction f coincides with the characteristic function of the domain occupied by the reference fluid, and can therefore be advected in the same fashion as in the Level Set methods. In the case of incompressible fluid, this approach leads to numerical schemes where it is necessary to compute the fluxes across the element boundaries [14]. This approach is sometimes referred to as Eulerian [15].

A different treatment of volume fraction advection has been proposed by Dukowicz and Baumgardner in [16]. Their method is based on two simple facts. The first is that the volume of a reference fluid inside a sub-domain is equal to its volume fraction integrated over that sub-domain. The second states that a certain volume of the reference fluid cannot change in time (mass conservation). By requiring that an arbitrary volume projected along the streamlines at time t^{n+1} is equal to the same volume at time t^n, they derived a simple method that can be based purely on geometrical procedures, i.e. intersections of polygons and polyhedrons.

Our approach in this work follows Dukowicz and Baumgardner [16], with later developments by Shahbazi [15]. The novelty lies in incorporating so-called Binary Space-Partitioning (BSP) trees. BSP trees were originally developed in connection with 3D computer graphics and are nowadays heavily used in the gaming industry. An early mention of BSP trees can be found in [17], and a modern exposition, together with implementation notes, is contained in [18]. The main idea behind BSP trees lies in subdividing space into convex sets by hyper-planes. This subdivision can be represented by a binary tree structure, which is used in our work for efficient implementation of polyhedra intersections. The main advantages of this formulation include natural geometrical interpretation, mass conservation, and applicability to structured and unstructured meshes.

The paper is organized as follows. The first section provides an overview of the governing equations of incompressible immiscible flow, together with discretization using the finite element method. Then the VOF-based interface tracking method is presented in detail, covering the interface reconstruction based on volume fraction distribution, a mass conserving interface update using a three step procedure, and description of the BSP algorithm for truncating advected volumes of the representative fluid. Finally, the interface tracking technique is presented on the basis of several examples that illustrates its capabilities and its performance.

2. A description of the fluid

This section provides a description of the governing equations for the flow of two immiscible fluids. As the problem is by nature fully transient, the Navier-Stokes equations (NSE) govern the motion of both fluids. Let us denote $\Omega \subset \mathbb{R}^3$ as a three dimensional domain which is completely filled with the two immiscible fluids, occupying the corresponding subsets $\Omega_1(t)$ and $\Omega_2(t)$, where t stresses the time dependence of the two subsets. The boundary of the whole domain, denoted as $\partial\Omega$, can be decomposed into two mutually disjoint parts, Γ_D and Γ_N, on which the so-called Dirichlet and Neumann boundary conditions are prescribed, respectively. The interface between the two fluids is denoted as $\Sigma(t)$. Then, for each phase $j = 1, 2$ in the domain Ω and on its boundary $\partial\Omega$, and for each time $t \in [0, T]$, the problem can be formulated as follows, see [19]

$$\begin{aligned}
\rho_j\Big(\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v}\cdot\boldsymbol{\nabla})\boldsymbol{v} - \boldsymbol{b}\Big) - \boldsymbol{\nabla}\cdot\boldsymbol{\sigma} &= \boldsymbol{0} && \text{in } \Omega_j,\\
\boldsymbol{\nabla}\cdot\boldsymbol{v} &= 0 && \text{in } \Omega_j,\\
\boldsymbol{v} &= \boldsymbol{g} && \text{on } \Gamma_D,\\
\boldsymbol{\sigma}\cdot\boldsymbol{n} &= \boldsymbol{h} && \text{on } \Gamma_N,\\
[\boldsymbol{v}]_{\Sigma(t)} &= 0 && \text{on } \Sigma,\\
[\boldsymbol{n}\cdot\boldsymbol{\sigma}]_{\Sigma(t)} &= 0 && \text{on } \Sigma,\\
\boldsymbol{v}_{t=0} &= \boldsymbol{v_0} && \text{in } \Omega_j.
\end{aligned} \tag{1}$$

In the equations presented above, we have denoted as $\boldsymbol{n}$ the normal vectors to both $\partial\Omega$ and $\Sigma(t)$, since it will be clear from the context which normal vector we have in mind. The unknowns are velocity field $\boldsymbol{v}$ and pressure field p. Density ρ, body forces $\boldsymbol{b}$ and functions $\boldsymbol{g}$, $\boldsymbol{h}$ and $\boldsymbol{v_0}$ are assumed to be known. The square brackets $[\cdot]_{\Sigma(t)}$ in the interface conditions on $\Sigma(t)$ denote a jump in the velocity and the normal stress components. In the case of a fluid with surface tension, the jump in the normal stress will be proportional to the curvature of the interface $\Sigma(t)$. Standard decomposition of stress tensor $\boldsymbol{\sigma}$ into deviatoric stress $\boldsymbol{\tau}$ and hydrostatic pressure p is used

$$\boldsymbol{\sigma} = \boldsymbol{\tau} - p\boldsymbol{\delta}. \tag{2}$$

Both fluids are considered as Newtonian fluids, so the constitutive equations read

$$\boldsymbol{\tau} = \mu\boldsymbol{D}, \tag{3}$$

where $\boldsymbol{D}$ denotes the strain rate tensor, which is defined as a symmetric part of the velocity gradient

$$\boldsymbol{D} = \frac{1}{2}\big(\nabla\boldsymbol{v} + (\nabla\boldsymbol{v})^T\big) \tag{4}$$

and μ is the dynamic viscosity of the fluid.

The numerical solution of (1) is based on stabilized Finite Elements as introduced in [12]. Provided that suitable finite dimensional sub-spaces $\boldsymbol{\mathcal{S}}^h \subset \boldsymbol{\mathcal{S}}$, $\boldsymbol{\mathcal{V}}^{\boldsymbol{h}} \subset \boldsymbol{\mathcal{V}}$ and $\mathcal{Q}^h \subset \mathcal{Q}$ are defined, the discretized problem states: find $\boldsymbol{v^h} \in \boldsymbol{\mathcal{S}}^h$ and $p^h \in \mathcal{Q}^h$ such that $\forall \boldsymbol{w^h} \in \boldsymbol{\mathcal{V}}^{\boldsymbol{h}}, \forall q^h \in \mathcal{Q}^h$:

$$\begin{aligned}
&\int_\Omega \rho_j \boldsymbol{w^h}\frac{\partial \boldsymbol{v^h}}{\partial t}\,\mathrm{d}x + \int_\Omega \rho_j \boldsymbol{w^h}\cdot(\boldsymbol{v^h}\cdot\nabla)\boldsymbol{v^h}\,\mathrm{d}x\\
&+ \int_\Omega \nabla\boldsymbol{w^h} : \boldsymbol{\tau}\big(\boldsymbol{D}(\boldsymbol{v^h})\big)\,\mathrm{d}x - \int_\Omega (\nabla\cdot\boldsymbol{w^h})p^h\,\mathrm{d}x\\
&- \int_\Omega \boldsymbol{w^h}\cdot\boldsymbol{b}\,\mathrm{d}x - \int_{\Gamma_N} \boldsymbol{w^h}\cdot\boldsymbol{h}\,\mathrm{d}s + \int_\Omega q^h(\nabla\cdot\boldsymbol{v^h})\,\mathrm{d}x\\
&+ \sum_{el}\int_{\Omega_e} \tau_{SUPG}(\boldsymbol{v^h}\cdot\nabla)\boldsymbol{w^h}\cdot\boldsymbol{\mathcal{R}}(\boldsymbol{v^h},p^h)\,\mathrm{d}x\\
&+ \sum_{el}\int_{\Omega_e} \tau_{PSPG}\frac{1}{\rho}\nabla q^h\cdot\boldsymbol{\mathcal{R}}(\boldsymbol{v^h},p^h)\,\mathrm{d}x = 0,
\end{aligned} \tag{5}$$

where $\mathcal{R}(\boldsymbol{v^h}, p^h)$ denotes the residuum of the linear momentum balance part of (1). The need for stabilization follows from the use of linear tetrahedrons (i.e. both velocity and pressure fields are approximated by linear functions) in order to circumvent issues with the LBB condition, see [20] for details. Note that the effective density ρ and also other material parameters such as viscosity μ, have to be averaged in elements which are cut by the interface. Averaging is performed using the original material parameters of both fluids, which are weighted by the respective volume fractions of the fluids in the cut element. The averaging is computed as

$$\mu = f\mu_1 + (1-f)\mu_2, \tag{6}$$

$$\rho = f\rho_1 + (1-f)\rho_2, \tag{7}$$

where f stands for the VOF value at the cut element. Additional stabilization terms are defined only element-wise, and are added as the sum over all elements Ω_e. Parameters τ_{SUPG} and τ_{PSPG} depend on the velocity and material parameters. Details on how they are determined can be found in [12, 13].

3. Evolution of the interface

This section presents in detail the computational representation of the interface between the two fluids, based on the VOF technique. First, we present the algorithms for interface reconstruction and tracking. Then there is a description of the method for truncating the polyhedra, based on BSP Trees. It should be noted that there is only one-way coupling between solving (5) and interface evolution; the interface advection is computed with the given velocity field obtained as a solution to (5).

The position of the interface can be obtained from the VOF spatial distribution. The interface is located either between two elements, when one is completely filled with the reference fluid (VOF = 1) while the other is fully filled with the second fluid (VOF = 0), or it passes through the partially filled element. On every partially filled element, we adopted a piece-wise linear approximation of the interface by the line segment in 2D and the planar segment in 3D. This is sometimes referred to as PLIC-VOF [5].

The interface evolution algorithm is required in order to update the interface position with the flow. The algorithm presented here is based on the paper by Shahbazi et al. (see [15]), and consists of three parts (see also Figure 1):

(1.) The Lagrangian part, in which the original mesh is projected along trajectories. This is achieved by advecting the nodal position with the flow. The time integration is based on forward trajectory remapping to evolve positions and volumes in time. To determine the trajectories, the velocity field is integrated from the original grid to the updated grid

$$\boldsymbol{x}_L(t^{t+\Delta t}) = \boldsymbol{x}(t) + \int_t^{t+\Delta t} \boldsymbol{v}\,\mathrm{d}t, \tag{8}$$

where $\boldsymbol{x}_L$ is the coordinate of the updated (Lagrangian) grid and the $\boldsymbol{x}$ is the original coordinate. The discretized form of equation (8) employing the midpoint Runge-Kutta method is

$$\begin{aligned} \boldsymbol{k}_1 &= \Delta t \boldsymbol{v}(t, \boldsymbol{x}(t)), \\ \boldsymbol{k}_2 &= \Delta t \boldsymbol{v}(t + \Delta t, \boldsymbol{x}(t) + \tfrac{1}{2}\boldsymbol{k}_1), \\ \boldsymbol{x}_L^{t+\Delta t} &= \boldsymbol{x}(t) + \boldsymbol{k}_2. \end{aligned}$$

Time step Δt is taken to be equal to the time step used for solving (5). In order to satisfy the CFL condition, we used adaptive time stepping where the time step is determined by the element with the most unfavorable ratio $h/||\boldsymbol{v}^h||$, given the velocity field previously obtained by solving (5).

(2.) The reconstruction part. In this step the fluid volumes are reconstructed on the updated grid, assuming that the VOF values remain constant during the Lagrangian phase. The 3D polyhedron representation of the volume occupied by the reference fluid is established on the basis of the spatial VOF distribution in each cell and its neighborhood. This includes the calculation of the interface segment normal, the segment constant, and the material volume truncation at each Lagrangian cell, as described in Section 3.1.

(3.) The remapping part. This involves assembling the polyhedral representation of a reference fluid material for each cell on the updated (Lagrangian) grid and redepositing it back on the target grid (i.e. on the original grid) to obtain updated VOF values. This phase is performed by a series of 3D polyhedron intersection procedures between the polyhedron representing the reference fluid on the updated grid and the polyhedra representing the original mesh cells, yielding the contributions to the updated reference fluid volumes in particular cells. The details of this algorithm are given in Section 3.2.

3.1. Interface Reconstruction

As was mentioned above, the interface at each partially filled cell (so-called interface cell) is represented as a planar segment, which can be described as

$$\boldsymbol{n} \cdot \boldsymbol{x} - c = 0, \tag{9}$$

where $\boldsymbol{n}$ is the unit normal to the planar surface representing the interface, $\boldsymbol{x}$ is a position vector, and c is the plane constant. The interface segment normal in each cell can be determined from the spatial volume fraction distribution in the neighborhood of the cell. The segment constant is then determined from the volume conservation requirement.

The method for determining the normal is based on an extension to Young's second method, developed originally by Rider and Kothe [5]. The idea is to form a Taylor series expansion of volume fraction f_i^{TS} for each interface cell i with VOF value f_i to each of its adjacent cell with k with VOF value f_k. Then, the sum of $(f_i^{TS} - f_k)^2$ over all cells neighboring cell i

$$\sum_k \big(f_i^{TS}(\boldsymbol{x}_i) + \boldsymbol{\nabla} f_i^{TS}(\boldsymbol{x}_i) \cdot (\boldsymbol{x}_k - \boldsymbol{x}_i) - f_k(\boldsymbol{x}_k)\big)^2 \tag{10}$$

is minimized in the least square sense. The minimization yields the volume fraction gradient $\boldsymbol{\nabla} f_i^{TS}$ corresponding to the interface segment normal $\boldsymbol{n}_i$ for cell i as the solution for the resulting system of normal equations. All coordinates are evaluated in the mass center of the cells.

This method guarantees exact reconstruction of the gradients for a linear function of f. However, even for the linear interface, the distribution of f through tessellation is not linear. This method is therefore only first order accurate. In [15], a second order accurate method is proposed for the 2D case. This method is based on geometric minimization of the differences between the real volume fractions and the fractions determined by the given line.

The value of planar segment constant c is determined from the volume conservation requirement. Its value is constrained such that the resulting planar segment (with the fixed normal determined in the previous step) passes through the cell with a truncation volume equal to the cell material volume V. The planar segment constant is determined from the following relation

$$F(c) = V(c) - V = 0, \tag{11}$$

where $V(c)$ is the material volume in the cell bounded by the planar interface segment with constant c and the portion of the cell boundary surfaces within the material. The cell material volume V is equal to the total cell volume multiplied by the volume fraction. Brent's method has been used to find zero of the $F(c)$ function (and to determine the volume conserving segment constant). This method uses inverse parabola interpolation, so it is suitable for finding the root of $F(c)$, as V often varies quadratically with c.

3.2. A BSP tree based approach to remapping the tetrahedral mesh

The remapping part of the proposed algorithm is of purely a geometric nature. It involves establishing a representation of the reference fluid volume in each cell (in the form of polyhedra) on the updated grid, projecting it onto the original grid and summing up the contributions of the truncated volumes (representing the contributing volume fractions) in each cell. The pseudo-code is presented in Algorithm 1

The overall procedure requires two basic operations. The first operation is polyhedron formation based on knowledge of the interface segment that intersects a given cell. The second necessary operation is to compute the intersection between two (convex) polyhedra and to compute the volume of the intersection.

Polyhedron formatting is represented by the TruncatePoly procedure in Algorithm 1. Once the tetrahedron and the corresponding intersecting plane are given, truncating the polyhedron is a trivial task, at least from the theoretical point of view. The only difficulty lies in the need to draw a correct distinction for each case of tetrahedron-plane intersection. The TruncatePoly procedure consists of a loop over the tetrahedron faces, where each of the faces is cut by the intersecting plane. Each such plane-face intersection results in a sub-face that belongs to the newly formed polyhedron, so that the new sub-face lies on the side of the interface plane filled with the reference fluid. Lastly, the closing sub-face of the new polyhedron (the sub-face coincident with the interface plane) is formed as a cross-section originating from the union of the intersecting lines. The FormPoly function

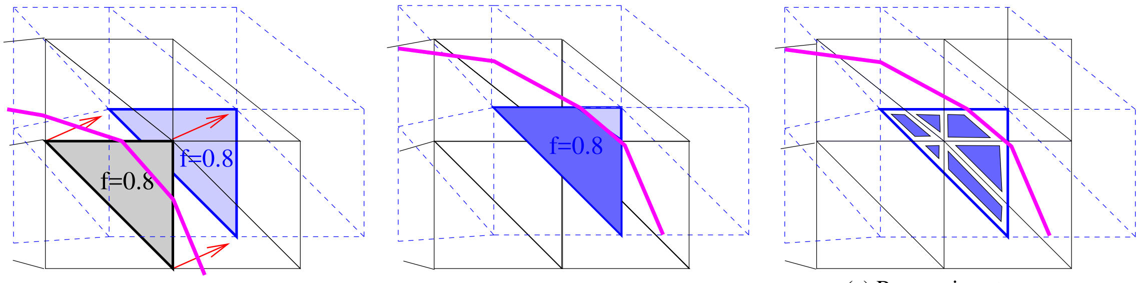
Algorithm 1 Remmaping algorithm

```
 1: procedure doInterfaceRemmaping
 2:     for el := 1 to $n_{el}$ do
 3:         if el is cut by interface then
 4:             n = el.GetPlaneNormal();
 5:             c = el.GetPlaneConstant();
 6:             P = el.TruncatePoly(n, c);
 7:             for all neighbours of el do
 8:                 Q = neighbour.FormPoly();
 9:                 vol = IntersectPoly(P, Q);
10:                 neighbour.AddVolume(vol);
11:             end for
12:         end if
13:     end for
14: end procedure
```

forms a polyhedron from the plain (not intersected) element.

Although an problem evaluation of the intersection of two arbitrary polyhedrons is theoretically trivial, implementing it presents an interesting problem. In this work, the intersections are computed with the help of so-called Binary Space-Partitioning (BSP) trees. The Original idea of space partitioning with binary trees comes from a paper by Fuchs, Kedem and Naylor [17] in 1980. Binary Space-Partitioning is, generally speaking, a method for recursively subdividing a space into convex sets by hyper-planes. This subdivision of the space enables polyhedra (in general not necessarily convex polyhedra) to be represented within the space by means of a tree data structure known as a BSP tree, see [18].

The idea is that a given plane $\boldsymbol{n}\cdot\boldsymbol{x}-c = 0$ partitions the space (here, we assume that the space is bounded) into two sub-spaces. According to the plane normal $\boldsymbol{n}$, one of the sub-spaces can be called positive (the points in that subspace lies on the side to which $\boldsymbol{n}$ points) and the other sub-space can be called negative. Note that points on the positive subspace satisfy the inequality $\boldsymbol{n} \cdot \boldsymbol{x} - c > 0$, and vice versa. Both the positive sub-spaces and the negative sub-spaces can be further partitioned by another plane, and so on. By applying this recursive procedure, one obtains the partitioning of the original space represented by a binary tree. Tho nodes of the tree represent the splitting planes. Hence, a left child of the node corresponds to a positive subspace created by the plane that the node represents; a right child corresponds to the negative subspace. The leaf nodes then represents convex regions obtained by partitioning. The idea is illustrated in Fig. 2, which is for clarity represented only in 2D. Splitting lines are denoted as P_i, convex regions are denoted as C_j. The sign over each edge of the tree indicates the positive and negative subspace. The structure of the BSP tree for partitioning the space can easily be used for partitioning a polyhedron into convex sub-polyhedra. This can be done by creating the nodes of the tree in such a way that each node represents a suitable



(a) Lagrangian step (b) Reconstruction step (c) Remapping steps

Figure 1. Illustration of the three phases of the interface update algorithm.

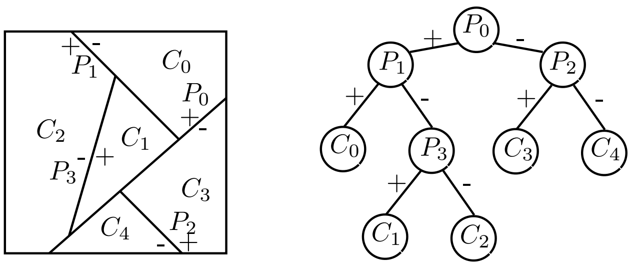


Figure 2. BSP tree - 2D illustration. The nodes correspond to splitting lines denoted as $P's$. The leaf nodes represent final convex sub-regions, denoted as $C's$

Algorithm 2 BSP tree construction

```
 1: procedure ConstructBSPTree(FaceList)
 2:     P = GetPlaneFromFace(FaceList.First())
 3:     for all Faces in FaceList do
 4:         type = ClassifyFace(P, Face)
 5:         if type = crosses then
 6:             (posFace, negFace) = SplitFace(P, Face);
 7:             AddToPosList(posFace);
 8:             AddToNegList(negFace);
 9:         else if type = positive then
10:             AddToPosList(Face);
11:         else if type = negative then
12:             AddToNegList(Face);
13:         else if type = coincident then
14:             AddToCoincident(Face);
15:         end if
16:     end for
17:     ConstructTree(PosList);
18:     ConstructTree(NegList);
19: end procedure
```

Algorithm 3 Intersection of Polyhedra

```
1: procedure IntersectPoly(P, Q)
2:     Q.ConstructBSPTree(Q.FaceList);
3:     P.IntersectWith(Q);
4:     P.ConstructBSPTree(P.FaceList);
5:     Q.IntersectWith(P);
6: end procedure
```

splitting plane (that splitting plane may or may not coincide with actual face of the polyhedron). Other faces are split at that node by using the corresponding splitting plane. All the faces that are on the positive side of that plane are sent to the positive child, and all the faces on the negative side are sent to the negative child of the tree. The process is then repeated, so the binary tree is computed recursively. A face which accidentally lies fully on the splitting plane is also stored at the corresponding node, and it is denoted as a coincident face. The pseudo-code for constructing a BSP tree, motivated by [18], is shown below. The key stage of BSP tree construction is classification of the mutual position between the splitting plane and the given face, represented by the ClassifyFace function. The classification is performed with the help of standard methods used in analytic geometry. However, due to the round-off errors, it is necessary to set up appropriate tolerance ε under which we consider the face and the splitting plane as coincident. Our experience is that ε depends on the problem that is to be solved (i.e. element size and time step) and a typical value is 10^{-11}.

3.2.1. Intersection of polyhedra

The intersection of two polyhedra is theoretically a trivial task, and can be computed in a straightforward manner. However, implementation in floating point arithmetic requires careful treatment of some special situations. We will go into this matter below. Let us say that we want to intersect polyhedron P with polyhedron Q. The idea is to intersect each face of P with polyhedron Q and to keep all portions of the intersected faces of P that lies inside Q as a part of the intersection between P and Q. Then, we proceed the other way around, i.e. we intersect all faces of Q with polyhedron P and keep all portions of the intersected faces of Q lying inside P. If we were test for intersection of each face of P with each face of Q, then the resulting algorithm would have quadratic complexity. However, the use of a BSP tree reduces the number of comparisons, because a face on one side of a splitting plane does not need to be tested with faces on the other side of the splitting plane [18]. The main part of Algorithm 3 is contained in the IntersectWith function. Basically, this function

Algorithm 4 Intersection with Polyhedron

```
1: procedure IntersectWith(P)
2:     for all Faces of Q do
3:         T = P.GiveBSPTree();
4:         (In, Coin) = GetPartition(T, Face);
5:     end for
6: end procedure
```

takes the faces of one polyhedron and sends them in a loop for further proceeding into the BSP tree for the other polyhedron, see Algorithm 4. The intersection of the two polyhedra can be composed of two types of faces/sub-faces. To the first type belong those faces of polyhedron P, which lies inside of polyhedron Q (and symmetrically faces of Q lying inside of P). To the second type belong faces of P, which accidentally coincide with some faces of Q (and vice versa). Therefore, the output of the IntersectWith function is composed of the two sets of faces, namely In and Coin. In contains sub-faces and faces of Q which lies inside polyhedron P, Coin contains sub-faces and faces of Q which coincide with some of the faces of P. The intersection value is computed from these two sets of faces, as will be described below.

The GetPartition function is the core part of the IntersectWith procedure. Its purpose is to partition a given face by the BSP tree into sub-faces which belong to a positive sub-space, or to a negative sub-space, or which are coincident with some of the splitting planes. The face is processed at each node (representing the splitting plane). According to the result of the ClassifyFace function, the face is sent to the positive child or to the negative child for further processing. If the face is crossed by the splitting plane, one part of the face lies on the positive side and the other part of the face lies on the negative side of the splitting plane. Therefore the part of the face on the positive side is sent to the positive child, and the other part is sent to the negative child. The most complicated case is when the face lies in the splitting plane, i.e. when the face and the splitting plane are coincident. In this case, the overlapping portion of the processed face and the face which served as a basis for the splitting plane has to be computed. Note that in the case of convex polyhedra, it is enough to keep just the overlapping part of the face; the rest is of no importance. In the case of general polyhedra, the non-overlapping portions of coincident faces would have to be processed by the positive or negative child of the BSP tree. The modification of algorithm proposed in [18] is a contained in pseudo-code Algorithm 5. In general, the result of GetPartition procedure is a representation of a face in form of a union of the segments (sub-faces) which are contained in positive, negative or coincident sub-spaces according to the BSP tree. However, all we need here is to compute the volume of the polyhedra intersection. Therefore, only the segments in the negative and coincident sub-

Algorithm 5 Partitioning of a Face

```
1: procedure GetPartition(Tree, Face)
2:     P = GetPlaneFromFace(Tree.CoinFace())
3:     type = ClassifyFace(P, Face)
4:     if type = crosses then
5:         (posFace, negFace) = SplitFace(P, Face);
6:         SendToPosChild(posFace);
7:         SendToNegChild(negFace);
8:     else if type = positive then
9:         SendToPosChild(Face);
10:    else if type = negative then
11:        SendToNegChild(Face);
12:    else if type = coincident then
13:        overlap = Intersect(CoinFace, Face);
14:        StoreIntersection(overlap);
15:    end if
16: end procedure
```

spaces of the BSP tree are needed. They are denoted as In and Coin, see algorithm 4. Finally, the volume is computed with the help of the Stokes formula

$$\int_{V_p} \nabla \cdot \boldsymbol{F} \, \mathrm{d}x = \int_{S_p} \boldsymbol{F} \cdot \boldsymbol{n} \, \mathrm{d}s, \tag{12}$$

which transforms the volume integral of divergence of vector field $\boldsymbol{F}$ to a surface integral of the normal component of that field. V_p and S_p denote the volume and the surface of the polyhedron, respectively. In order to compute the volume V of the polyhedron resulting from the intersection, we can proceed as follows

$$\begin{aligned} V &= \int_{V_p} 1 \, \mathrm{d}x = \int_{V_p} \nabla \cdot \left(\frac{\boldsymbol{x}}{3}\right) \mathrm{d}x = \frac{1}{3} \int_{S_p} \boldsymbol{x} \cdot \boldsymbol{n} \, \mathrm{d}s \\ &= \frac{1}{3} \sum_{F_i \in \{\textsc{In} \cup \textsc{Coin}\}} \int_{F_i} \boldsymbol{x} \cdot \boldsymbol{n}_i \, \mathrm{d}s \\ &= \frac{1}{3} \sum_{F_i \in \{\textsc{In} \cup \textsc{Coin}\}} \int_{F_i} c_i \, \mathrm{d}s = \frac{1}{3} \sum_{F_i \in \{\textsc{In} \cup \textsc{Coin}\}} c_i A_i, \end{aligned} \tag{13}$$

where $\boldsymbol{n}_i$ and c_i are used to denote the normal vector and the plane constant of face F_i, respectively. The area of face F_i is denoted as A_i. The volume of the intersection of the two polyhedra then reduces to an evaluation of the areas and the plane constants of the faces contained in the set $\{\textsc{In} \cup \textsc{Coin}\}$.

All the procedures were implemented into the OOFEM code [21, 22]. All the results presented in the following section were also obtained with OOFEM.

4. Numerical examples

In this section, we present several numerical examples that illustrate the capability of the proposed BSP tree-based VOF method. The first example focuses on the shape-preserving ability of the method, and also on conservation of volume. The second example

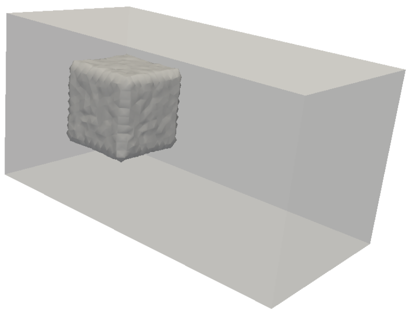


Figure 3. Cube translation — initial position, $t = 0\,\mathrm{s}$.

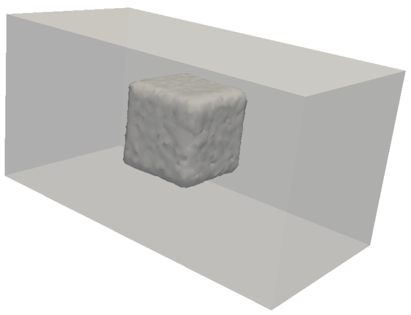


Figure 4. Cube translation — middle position, $t = 5\,\mathrm{s}$.

models the famous experiment performed by Martin and Moyce in the early 1950s [23].

4.1. Cube propagation through the unstructured mesh

The example presented here considers the propagation of a cube through the unstructured mesh in the case of two basic motions of a fluid: translation and rotation. In both cases, the velocity field is prescribed in the whole domain, so we can focus exclusively on the VOF method itself. The focus here is on the shape-preserving and volume conservation properties.

4.1.1. Cube translation

The computational domain is a $10 \times 10 \times 20$ prismatic block with a $5 \times 5 \times 5$ cube inside, see Figure 3. The velocity field is prescribed as a uniform field, i.e. $(1, 0, 0)$, along the longest side of the block in the x-direction. For the computation presented here, we used an unstructured mesh with 27910 nodes and 150909 elements. For shape preservation, Figures 3–5 show the initial position, the middle position and the final position of the cube; it can be seen that the shape preservation is very decent. Figures 6 display slices through the cube by planes with the normal vectors in x, y and z direction respectively, at times $t = 0\,\mathrm{s}$ (initial position), $t = 5\,\mathrm{s}$ (middle position) and $t = 10\,\mathrm{s}$ (final position). Visually, there is no significant error in the original shape of the cube. Detailed information about shape preservation is provided in

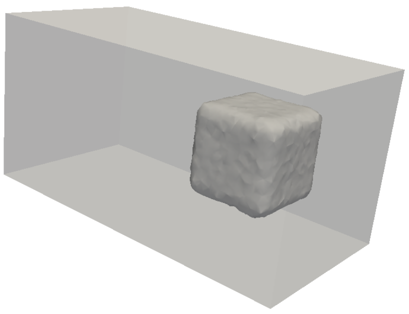


Figure 5. Cube translation — end position, $t = 10\,\mathrm{s}$.

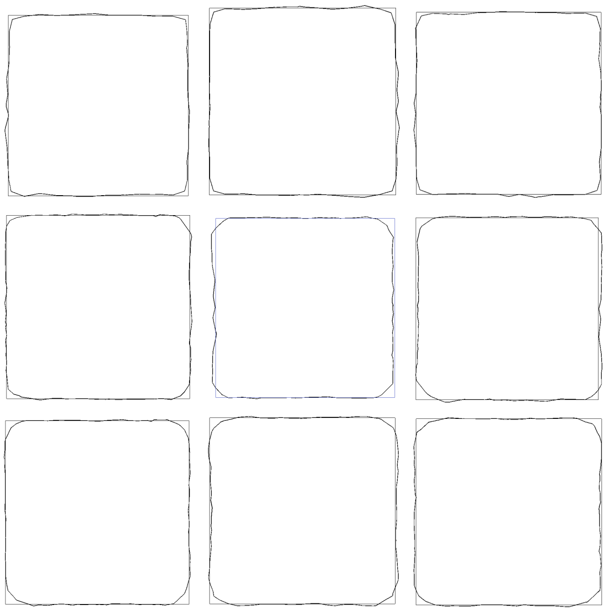


Figure 6. Cube translation. Slices with the normal in the x, y, z-direction (columns) and times $t = 0, 5, 10\,\mathrm{s}$ (rows).

Table 1, which presents a quantitative evaluation of the VOF values at time $t = 0\,\mathrm{s}$ and $t = 10\,\mathrm{s}$; the first column shows the relative error of the VOF values (denoted as H) on each side of the cube. The mean value of VOF is shown in the middle column, and the standard deviation is in the third column. The theoretically exact value $H_{ex} = 0.5$. It can be seen that there are no significant changes in the mean value, and the first column shows an average error of approximately 12 %. There is a slight increment in the standard deviation, which indicates non-uniform distribution of VOF values across the planes of the cubes and increasing fluctuations. Other tests, which are not reported here, show that these fluctuations are reduced when finer meshes are used.

The volume conservation is also excellent. The ratio between the final and initial the volume is only 1.000076, which represents an increase in volume of less than 0.1 ‰.

	Cube side	$\lvert\mathbb{E}(H) - H_{ex}\rvert / H_{ex}$		$\mathbb{E}(H)$		$\sigma(H)$	
	Time	t_0	t_{10}	t_0	t_{10}	t_0	t_{10}
n_x	Front	0.122	0.005	0.439	0.503	0.121	0.143
	Back	0.127	0.186	0.437	0.407	0.121	0.125
n_y	Left	0.144	0.108	0.428	0.446	0.118	0.141
	Right	0.141	0.121	0.429	0.439	0.121	0.133
n_z	Top	0.123	0.113	0.439	0.443	0.129	0.146
	Bottom	0.132	0.136	0.434	0.432	0.124	0.152

Table 1. Cube propagation — shape preservation evaluation.

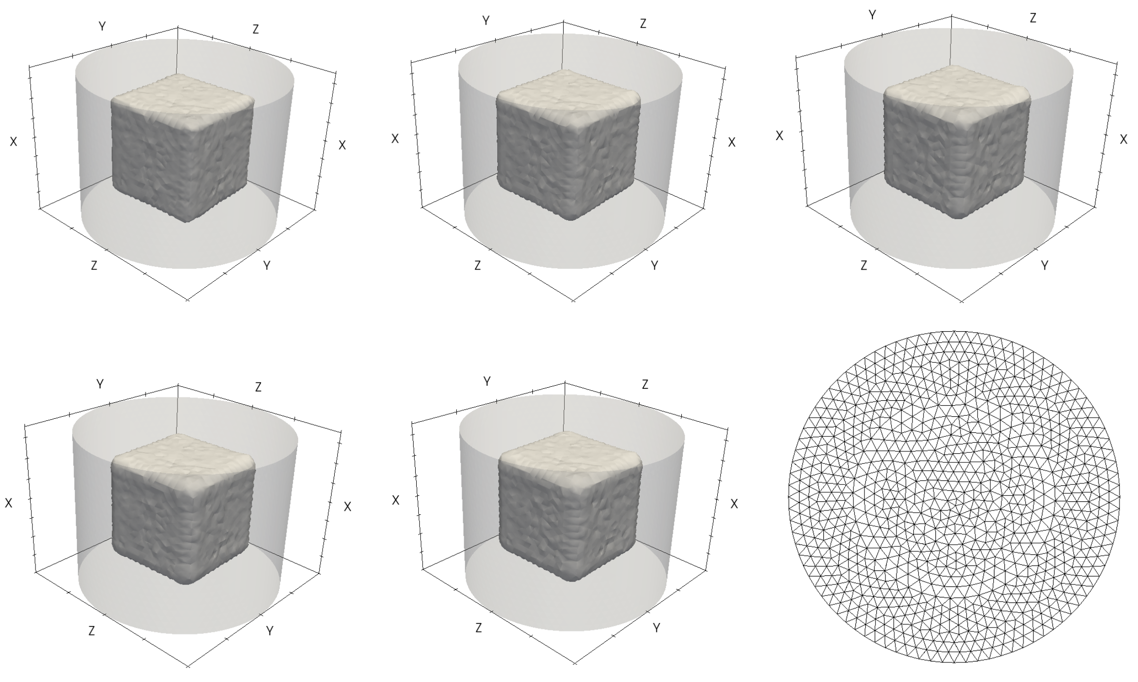


Figure 7. Cube rotation. From left to right and from top to bottom there are rotation angles $\alpha = 0°, 90°, 180°, 270°, 360°$. The bottom-right figure shows the top view of the fully unstructured mesh.

4.1.2. Cube rotation

The second example performed in order to investigate the shape and volume preserving capabilities deals with rotation of the cube. The geometry is formed as a $1 \times 1 \times 1$ cube inside a cylinder with radius 1 and height 1.5. The velocity changes linearly from 0 at the axis of rotation to 1 at the surface of the cylinder. The computational mesh consists of 24448 nodes and 132971 elements, and it is fully unstructured. The whole setup is shown in Figure 7, where the cube is presented in its initial position and then after 90°, 180°, 270° and 360° of rotation. The last figure shows the top view on the unstructured mesh. The conservation of volume is again excellent. The ratio between the initial and the final position of the cube (after 360 deg rotation) is 0.9998012 which represents an error of less than 0.2 ‰ error. Figure 8 shows slices through the cube by planes with the normal in the x, y and z direction, again for the initial position and after 90°, 180°, 270° and 360° rotation, showing shape preservation. Quantitative results connected to shape preservation are summarized in Table 2. Similarly as in the cube translation, the first column shows the relative error in the VOF values on each side of the cube for the initial position and after 360° rotation. The second column shows the mean VOF values, and the third column shows the standard deviation. Note that the trend is the same as in cube translation, i.e. the VOF values do not change significantly on an average, but, as the cube rotates, the fluctuations around the mean value increase slightly.

4.2. The Martin and Moyce broken dam experiment

The final example makes use of the famous Martin and Moyce experiment with the problem of a broken dam (also known as the water column collapse problem), see [23].

The setup of the experiment is illustrated in Figure 9 in the top-left picture. A rectangular column

	Cube side	$\lvert\mathbb{E}(H) - H_{ex}\rvert/H_{ex}$		$\mathbb{E}(H)$		$\sigma(H)$	
	Angle	0 deg	360 deg	0 deg	360 deg	0 deg	360 deg
n_x	Top	0.139	0.186	0.431	0.407	0.135	0.171
	Bottom	0.135	0.199	0.433	0.400	0.136	0.175
n_y	Left	0.059	0.077	0.471	0.461	0.100	0.158
	Right	0.058	0.061	0.471	0.469	0.101	0.156
n_z	Front	0.054	0.019	0.473	0.491	0.100	0.181
	Back	0.059	0.040	0.470	0.480	0.104	0.187

Table 2. Cube rotation — shape preservation evaluation.

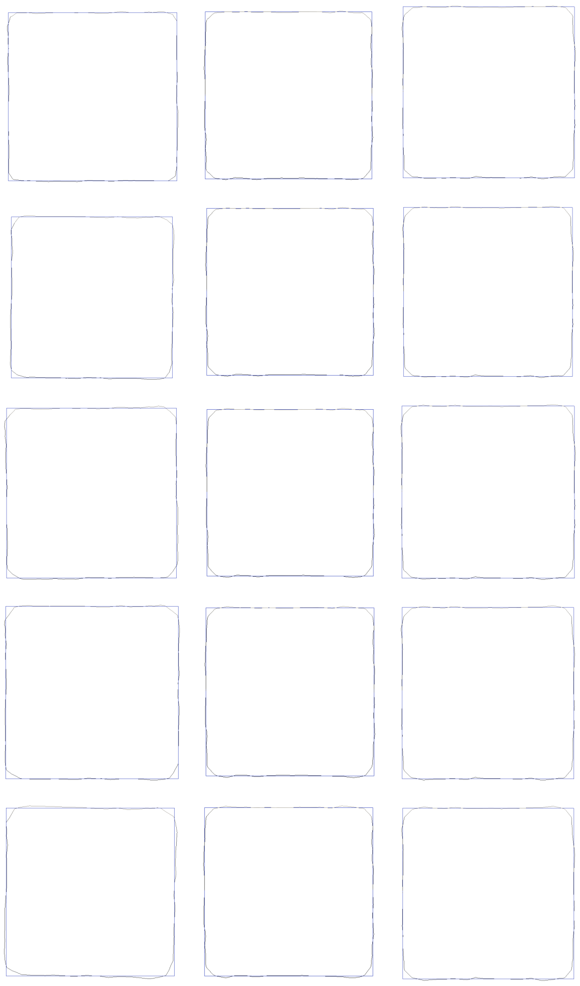


Figure 8. Cube rotation. Slices with the normal in the x, y, z-direction (column) and rotation angles $\alpha = 0°, 90°, 180°, 270°, 360°$. (row).

of water is initially in a state of hydrostatic equilibrium. After time t_0, the water column starts to collapse due to its gravity by forming an advancing water wave, as can be seen in the rest of the pictures of Figure 9. The initial configuration is chosen as a cubic water block with edge-length $a = 0.05175$ m, and the bounding box is a prismatic block with dimensions of $0.2286 \times 0.05715 \times 0.085725$ m. Frictionless boundary conditions are assumed on the bottom and on the vertical walls. The density and the viscosity of water are taken as $1000\,\mathrm{kg/m^3}$ and 1×10^{-3} Pa s. Ambient air with density equal to $1\,\mathrm{kg/m^3}$ and viscosity 1×10^{-5} Pa s is assumed. Four different discretizations with 2454, 12589, 19111 and 29604 nodes have been used to verify the objectivity of the description with respect to mesh size. The position of the water front and the residual water column height with respect to time are compared with the experimental results in Figure 10 and Figure 11, respectively. For convenience, the dimensionless lengths $x^* = x/a$, $z^* = z/a$ and time $t^* = t\sqrt{g/a}$ are used in the figures.

It can be seen from Figure 10 that numerical results converge to the experimental results with increasing fineness of the mesh. In case of the residual height of the column in Figure 11, the results are less sensitive to the mesh and all the meshes that were performed seem to provide reasonably accurate results.

5. Conclusion

The research presented here has dealt with numerical simulations of flow with a free surface. The free surface is handled using the Volume-Of-Fluid method, and the strategy employed for advancing the interface is based on purely geometric algorithms. In particular, we have proposed a new approach for interface reconstruction and for remapping phases of the advecting algorithms, which is based on Binary Space-Partitioning. To the best of our knowledge, this approach has not been published before.

The presented examples show very good volume-conserving properties. The shape preserving properties are also satisfactory. We believe that this is achieved mainly due to the geometrical nature of the algorithm, which is implemented in a very effective manner with the help of the idea of Binary Space-Partitioning. Although this work is considered as a proof of concept, and additional efforts will be needed in order to develop a more complex code, the results presented here are more than promising. In particular, more accurate time integration in the Lagrangian phase of the interface advecting algorithm, and an analysis of its influence on the volume conserving properties, would be beneficial.

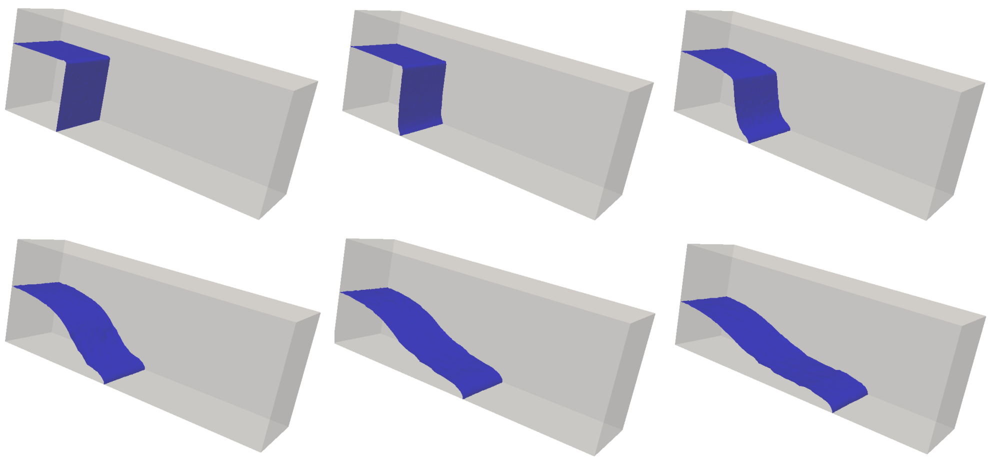


FIGURE 9. Martin and Moyce. From left to right and from top to bottom, the times are $t = 0, 0.025, 0.05, 0.075, 0.1$ and 0.125 s.

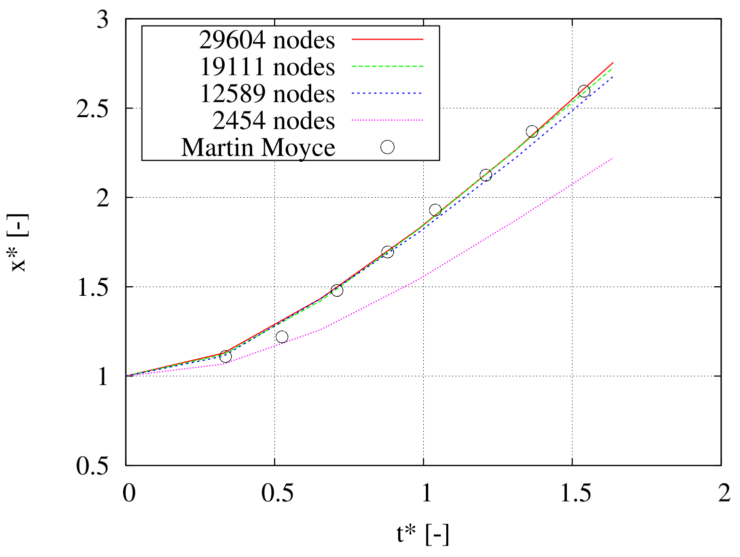


FIGURE 10. Martin and Moyce. Dependence of the wave front on time for different meshes and experimental results.

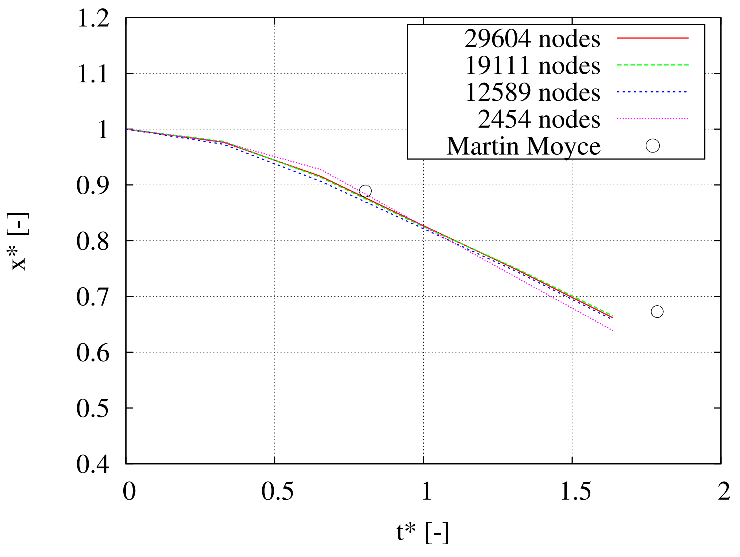


FIGURE 11. Martin and Moyce. Dependence of the residual height of the column on time for different meshes and experimental results.

Acknowledgements

The authors would like to acknowledge the support provided by the Czech Science Foundation under project 13-23584S.

References

[1] G. Trapaga, E. F. Matthys, J. J. Valencia, J. Szekely. Fluid flow, heat transfer, and solidification of molten metal droplets impinging on substrates: Comparison of numerical and experimental results. *Metallurgical and Materials Transactions B* **23**, 1992. doi:10.1007/bf02656450.

[2] W. Brouwer, E. van Herpt, M. Labordus. Vacuum injection moulding for large structural applications. *Composites Part A: Applied Science and Manufacturing* **34**, 2003. doi:10.1016/s1359-835x(03)00060-5.

[3] I. Akkerman, Y. Bazilevs, D. J. Benson, et al. Free-surface flow and fluid-object interaction modeling with emphasis on ship hydrodynamics. *Journal of Applied Mechanics* **79**, 2012. doi:10.1115/1.4005072.

[4] B. Patzák, Z. Bittnar. Modeling of fresh concrete flow. *Computers & Structures* **87**(15–16):962–969, 2009. doi:10.1016/j.compstruc.2008.04.015.

[5] W. J. Rider, D. B. Kothe. Reconstructing volume tracking. *Journal of Computational Physics* **141**, 1998. doi:10.1006/jcph.1998.5906.

[6] E. Stein, R. D. Borst, T. J. Hughes. *Encyclopedia of computational mechanics*, vol. Volume 3. John Wiley, 1st edn., 2004.

[7] S. Osher, J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics* **79**, 1988. doi:10.1016/0021-9991(88)90002-2.

[8] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2nd edn., 1999.

[9] R. F. a. Stanley Osher. *Level Set Methods and Dynamic Implicit Surfaces*. Applied Mathematical Sciences 153. Springer-Verlag New York, 1st edn., 2003.

[10] C. Hirt, B. Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. *Journal of Computational Physics* **39**, 1981. doi:10.1016/0021-9991(81)90145-5.

[11] M. Sussman, E. G. Puckett. A coupled level set and volume-of-fluid method for computing 3d and axisymmetric incompressible two-phase flows. *Journal of Computational Physics* **162**, 2000. doi:10.1006/jcph.2000.6537.

[12] T. E. Tezduyar. Stabilized Finite Element Formulations for Incompressible Flow Computations. *Advances in Applied Mechanics* **28**:1–44, 1991.

[13] T. E. Tezduyar, Y. Osawa. Finite Element Stabilization parameters computed from element matrices and vectors. *Computer Methods in Applied Mechanics and Engineering* **190**:411–430, 2000.

[14] J. E. Pilliod, E. G. Puckett. Second-order accurate volume-of-fluid algorithms for tracking material interfaces. *Journal of Computational Physics* **199**, 2004. doi:10.1016/j.jcp.2003.12.023.

[15] K. Shahbazi, M. Paraschivoiu, J. Mostaghimi. Second order accurate volume tracking based on remapping for triangular meshes. *Journal of Comput Physics* **188**:100–122, 2003.

[16] J. K. Dukowicz, J. R. Baumgardner. Incremental remapping as a transport/advection algorithm. *Journal of Computational Physics* **160**, 2000. doi:10.1006/jcph.2000.6465.

[17] H. Fuchs, M. Kedem, B. F. Naylor. On visible surface generation by a priori tree structures. *ACM SIGGRAPH Computer Graphics* **14**, 1980. doi:10.1145/965105.807481.

[18] P. Schneider, D. H. Eberly. *Geometric tools for computer graphics*. The Morgan Kaufmann series in computer graphics and geometric modeling. Boston :, Morgan Kaufmann Publishers, 2003.

[19] H. Sauerland, T. P. Fries. The extended finite element method for two-phase and free-surface flows: A systematic study. *Journal of Computational Physics* **230**:3369–3390, 2011.

[20] I. Babuska. The finite element method with Lagrangian multipliers. *NumMath* **20**:179–192, 1973.

[21] B. Patzák. OOFEM home page, 2000. Http://www.oofem.org.

[22] B. Patzák, Z. Bittnar. Design of object oriented finite element code. *Advances in Engineering Software* **32**:759–767, 2001.

[23] J. C. Martin, W. J. Moyce. Part iv. an experimental study of the collapse of liquid columns on a rigid horizontal plane. *Philosophical Transactions Mathematical Physical and Engineering Sciences* **244**, 1952. doi:10.1098/rsta.1952.0006.