Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computational Physics 208 (2005) 228–252

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

Adaptive tetrahedral meshing in free-surface flow

Meizhong Dai, David P. Schmidt *

Department of Mechanical and Industrial Engineering, University of Massachusetts, Amherst Massachusetts 01003, USA

Received 8 April 2004; received in revised form 16 February 2005; accepted 16 February 2005
Available online 13 April 2005

Abstract

An unstructured, three-dimensional, moving-mesh algorithm is developed for free-surface flows with the capability of simulating large deformation. A robust edge swapping algorithm and an optimization-based mesh smoothing algorithm are used to improve mesh quality as the interface deforms. The edge swapping algorithm uses an optimal swap sequence to transform configurations with several tetrahedra sharing a common edge. The optimization-based mesh smoothing relocates vertex position so that the minimum quality of the incident cells is maximized. Also, edge contraction and bisection are used to adjust the mesh resolution according to surface curvature. These local refinements effectively maintain good mesh quality and appropriate mesh size while avoiding global re-meshing. This scheme's robustness and accuracy are demonstrated with deforming liquid ligaments, periodic jets, and colliding droplets.
© 2005 Elsevier Inc. All rights reserved.

PACS: 65P45; 76T99

Keywords: Moving-mesh; Dynamic meshing; ALE; Edge swapping; Mesh smoothing; Free-surface flow

1. Introduction

Many important engineering applications can be simulated using a free-surface model. This work is particularly concerned with liquid atomization. The details of this process are difficult to observe due to its high speed and small scale. Therefore, the ability to do direct numerical simulation will greatly help develop physical insights into this phenomenon and improve the accuracy of current spray models. This paper presents an incompressible three-dimensional moving-mesh algorithm with the capability of simulating free-surface flow with large deformation.

* Corresponding author. Tel.: +1 413 545 1393; fax: +1 413 545 1027.
E-mail address: schmidt@ecs.umass.edu (D.P. Schmidt).

0021-9991/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2005.02.012

Nomenclature

$I_n(\cdot)$	modified Bessel function of the first kind of order n
$J_n(\cdot)$	Bessel function of the first kind of order n
$\det(\cdot)$	determinant of a square matrix
k	wave number, $2\pi/\lambda$
r_0	initial radius
S	deformation of droplet collision: surface area/initial surface area
t_c	characteristic time, $(\rho r_0^3/\sigma)^{1/2}$
J	$=\sigma r_0/(\rho v^2) = 1/Oh^2$
Oh	Ohnesorge number, $\mu/(\rho r_0 \sigma)^{1/2}$
Re	Reynolds number
We	Weber number
ε	disturbance amplitude of jet radius, non-dimensionalized by r_0
ϕ	velocity potential
λ	wavelength
μ	dynamic viscosity
v	kinematic viscosity
ρ	density
σ	surface tension
Φ	dissipation

In moving-mesh methods the domain deforms in order to follow the interface in a Lagrangian manner. Lagrangian methods explicitly track the interface position by following the fluid motion. A two-dimensional example is Welch's moving-mesh scheme [1]. Welch used a triangular mesh to simulate compressible bubble dynamics with mass transfer. Compared with other multi-phase methods, the moving-mesh scheme has the following advantages. First, because moving-mesh schemes deform the mesh to follow the interface motion, the interface is exactly tracked and is not smeared or diffused. In contrast, this feature is difficult to achieve in a volume-of-fluid (VOF) method [2], which is an Eulerian surface capturing method. VOF methods use a volume fraction or color function to represent the portion of a cell filled with a particular phase. This volume fraction function is advected based on mass conservation to capture the interface. One of the advantages of VOF is that no special treatment is necessary for interface reconnection or breakup, and hence topological robustness is easily obtained (like [3], which used a characteristics-based method and a triangular grid). The main difficulty of these methods is that the interface is not smooth when reconstructed from the color function. Brackbill et al.'s [4] continuum surface force (CSF) method does not need to reconstruct the interface for surface tension calculation, and uses a carefully selected kernel to smooth the color function in the transition region. However, the interface profile may be diffused. The VOF method with quadtree mesh adaptation [5,6] reduces the interface diffusion by refining the mesh around the whole interface. This method splits a cell if it is close to the interface regardless of the local geometry, while the present moving-mesh method only refines the mesh when the local surface curvature is high. So generally the total number of cells for VOF method with adaptation is more than those in the moving-mesh method. For example, if five levels of refinement are used to get ideal resolution, then the interface mesh size will be 2^5 times smaller than the original mesh, which may considerably increase the computational effort for three-dimensional flows. Second, moving-mesh methods can achieve mass conservation easily, since the interface is moved using the fluid velocity. In the level set methods, the interface is represented by the zero contour of a characteristic function. Like the VOF method, level set is also an Eulerian interface capturing method. However, because the zero contour of the characteristic function does not have a strong mass conservation property as VOF's volume fraction function does, level set methods usually must use some extra treatments to conserve mass (such as Lagrangian particles [7]). Third, in the moving-mesh scheme, resolution can be easily adjusted using edge bisection and edge contraction methods. Because an unstructured mesh is used, these operations will not introduce any extra difficulties. Finally, moving-mesh methods do not need to average the interface quantities. Front-tracking methods, which are also Lagrangian methods, use a structured fixed mesh to solve the fluid equations, and use tracking particles to track the interface [8]. These tracking particles are connected so they form a mesh with a dimension one less than the fixed mesh. Then during the solution process, the quantities at the interface, such as surface tension forces, are converted from the front tracking mesh to the nearby nodes of the fixed mesh by a weighting function. In front-tracking methods, the interface is actually treated like a narrow transition strip with finite width, but the width of the interface will not diffuse as in VOF methods. In a moving-mesh scheme one single mesh is used to solve both the interface motion and the fluid equations. The interface moves with the fluid, avoiding diffusion of the underlying fluid properties. Therefore, there is no need to convert or average interface quantities.

Some boundary integral methods also move the mesh to track the interface [9,10]. In these methods, the flow solution in the entire domain is obtained only from the boundary meshes. Since boundary integral methods use a surface mesh, only two-dimensional mesh improvement techniques are necessary. Because these methods can only solve purely elliptic problems, they are restricted to inviscid flows and creeping flows.

Compared with other multi-phase methods, the major disadvantage of moving-mesh methods is that the mesh may quickly become distorted and tangled, leading to large numerical error. Thus Welch's simulations [1] were limited to weakly deformed bubbles. To overcome this difficulty, most moving-mesh methods utilize some kind of mesh improvement techniques. Fukai et al. [11,12] studied the deformation of an axisymmetric liquid droplet colliding with a flat surface. In their work, a new mesh was generated when the distortion of the existing mesh exceeded a threshold value. This new mesh was generated in a way that only the nodes close to the droplet interface were different from the original mesh. This technique avoids global re-meshing, but the algorithm is difficult to generalize to three dimensions and other boundary conditions. Hu et al. [13] also used a moving-mesh to simulate particle interaction. In their work, node insertion was used for mesh refinement when two particles approached each other, and the changes in the cell volume and aspect ratio from the initial mesh were used as the mesh quality measure. When some cells' qualities became unacceptable, a new mesh was generated globally with no correspondence to the old one, and then all fluid parameters were projected to the new mesh.

In practice, mesh improvement without global re-meshing is desirable in order to obtain better computational accuracy, stability and efficiency. Two commonly used improvements for triangular meshes are edge swapping (flipping) and Laplacian mesh smoothing. Edge swapping changes the topological structure of the existing mesh by local reconnections. Since these changes only affect local elements, large-scale interpolation of the flow field is avoided. The reconnection capability is essential for a deforming mesh, since if two vertices are moving in opposite directions, then at some time they must be allowed to disconnect. Fyfe et al. [14] used two-dimensional edge swapping and node addition/removal to transform obtuse angles, and simulated internal gravity and capillary waves, droplet oscillations, and viscous shear layers. The Laplacian mesh smoothing method relocates the vertex position without changing mesh topology. Helenbrook [15] used mesh smoothing and simulated an inviscid droplet oscillating in a vacuum and a viscous droplet falling in gas. Baker et al. [16] used both edge swapping and mesh smoothing methods, together with edge contraction and node insertion to control mesh resolution. Instead of eliminating obtuse angles, Baker et al. used the Delaunay criterion as the mesh quality measure. The Delaunay criterion means that no vertex falls in the interior of the circumcircle/circumsphere of any triangle/tetrahedron in the triangulation. Because two-dimensional Delaunay triangulation has the property that it can maximize the minimum angle among all triangulations of a planar point set, two-dimensional moving-mesh schemes can tolerate significant mesh distortion. Using this method Baker et al. successfully simulated a circle translating through a channel and a square rotating in a circle. From these demonstrations one can see that some of the global re-meshing steps in Hu et al.'s [13] two-dimensional simulations may be avoided using edge swapping and mesh smoothing. Although in two dimensions Delaunay triangulation normally generates a good-quality mesh, in three dimensions the Delaunay criterion fails to identify a kind of harmful cell called "sliver cells". These cells have a moderate circum-radius (compared to the edge length), and thus can often survive in a Delaunay triangulation. Because of this, the desirable properties of the two-dimensional Delaunay criterion cannot be generalized to three dimensions. Therefore, in the past, moving-mesh methods have been limited by the robustness and accuracy of re-meshing.

This paper presents a new three-dimensional dynamic meshing scheme using an edge swapping algorithm developed by Briere de L'isle and George [17], Klincsek [18] and Shewchuk [19], and an optimization-based mesh smoothing algorithm by Freitag et al. [20,21]. The three-dimensional edge swapping algorithm was originally developed by Briere de L'isle et al., who proposed to use the optimal configuration to transform the local structure of several tetrahedra sharing a common edge. They found this configuration by enumerating all possible cases. Shewchuk used Klincsek's dynamic programming technique and developed a way to find such a configuration in polynomial time. In the present work Shewchuk's algorithm is revised to avoid the temporary inverted cells, so that the geometric calculation is easy to implement. In addition, a robust interpolation scheme is developed to handle the degenerate cells during the edge swapping process. Another essential technique for maintaining mesh quality is mesh smoothing. Unlike the Laplacian smoothing method, Freitag et al.'s optimization-based mesh smoothing relocates a vertex so that the minimum quality of the incident tetrahedra is maximized. Therefore, optimization-based smoothing guarantees that the minimum and overall mesh qualities are always improved. The present work applies these recently developed techniques to a moving-mesh method, and develops a numerical scheme that can simulate three-dimensional free-surface flow with large deformation.

The outline of the paper is as follows: Section 2 describes the governing equations and discretization scheme, and Sections 3 and 4 describe the mesh reconnection and optimization-based smoothing algorithms. Some numerical tests and simulations are presented in Sections 5 and 6. These examples demonstrate that this dynamic meshing scheme can simulate large deformation in three dimensions.

2. Governing equations and discretization

This work uses an exact fractional step method and a staggered mesh [22,23] to solve the incompressible Navier–Stokes equations. Consider a moving control volume V with surface A and normal $\mathbf{n}$. Let $\mathbf{u}$ be the fluid velocity, and $\mathbf{u}_{\text{mesh}}$ be the mesh velocity. The incompressible continuity and momentum equations for this control volume are:

$$\oint_A \mathbf{u} \cdot \mathbf{n} \, \mathrm{d}A = 0, \tag{1}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_V \rho \mathbf{u} \, \mathrm{d}V + \oint_A \rho \mathbf{u} (\mathbf{u} - \mathbf{u}_{\text{mesh}}) \cdot \mathbf{n} \, \mathrm{d}A = \int_V (\rho \mathbf{g} - \nabla p) \, \mathrm{d}V + \oint_A \mu \nabla \mathbf{u} \cdot \mathbf{n} \, \mathrm{d}A. \tag{2}$$

Now define the velocity area integrals and the average cell velocity as:

$$U = \int_A \mathbf{u} \cdot \mathbf{n} \, \mathrm{d}A, \tag{3}$$

$$U_{\text{mesh}} = \int_A \mathbf{u}_{\text{mesh}} \cdot \mathbf{n}\, dA, \tag{4}$$

$$\mathbf{u}_{\text{c}} = \frac{1}{V} \int_V \mathbf{u}\, dV. \tag{5}$$

Then the discrete versions of Eqs. (1) and (2) on a tetrahedra cell are (with the Crank–Nicholson scheme applied to the diffusion terms):

$$\sum_{\text{faces}} U = 0, \tag{6}$$

$$\rho \frac{\mathbf{u}_{\text{c}}^{n+1} V^{n+1} - \mathbf{u}_{\text{c}}^{n} V^{n}}{\Delta t} + \rho \sum_{\text{faces}} \mathbf{u}_{\text{f}} (U - U_{\text{mesh}}) = -V^{n+1} (\nabla p - \rho \mathbf{g})^{n+1} + \sum_{\text{faces}} 0.5 \left[(\mu \nabla \mathbf{u}_{\text{f}} \cdot \mathbf{n} A)^{n} + (\mu \nabla \mathbf{u}_{\text{f}} \cdot \mathbf{n} A)^{n+1} \right]. \tag{7}$$

Because in a staggered mesh scheme the primary unknown is the face mass flux U, the cell velocity $\mathbf{u}_{\text{c}}$ must be reconstructed from U:

$$\mathbf{u}_{\text{c}} = \frac{1}{V} \sum_{\text{faces}} U (\mathbf{x}_{\text{f}} - \mathbf{x}_{\text{c}}), \tag{8}$$

where $\mathbf{x}_{\text{f}}$ and $\mathbf{x}_{\text{c}}$ are the face and cell centers of gravity, and the mass flux U is positive when flowing out of this cell. The face velocity $\mathbf{u}_{\text{f}}$ and face velocity gradient $\nabla \mathbf{u}_{\text{f}} \cdot \mathbf{n}$ in Eq. (7) are approximated by interpolating the cell velocities $\mathbf{u}_{\text{c}}$ from the neighbor cells.

The calculation of the face integral of the mesh velocity U_{mesh} in Eq. (7) is shown in Fig. 1. Consider a face $\mathbf{x}_1^n \mathbf{x}_2^n \mathbf{x}_3^n$ at time n. Assume this face moves to a new position $\mathbf{x}_1^{n+1} \mathbf{x}_2^{n+1} \mathbf{x}_3^{n+1}$ at time $n + 1$. Then U_{mesh} is the volume of the prism swept by this face divided by the time interval Δt. Assuming the three vertices moves with a constant velocity, Perot and Nallapati [22] integrated this process and found an exact analytical solution

$$U_{\text{mesh}} = \frac{1}{\Delta t} (\mathbf{x}_{\text{f}}^{n+1} - \mathbf{x}_{\text{f}}^{n}) \cdot \left\{ \frac{1}{2} (\mathbf{n}^{n+1} A^{n+1} + \mathbf{n}^{n} A^{n}) - \frac{1}{12} [\mathbf{v}_{n1} \times \mathbf{v}_{n2} + \mathbf{v}_{n2} \times \mathbf{v}_{n3} + \mathbf{v}_{n3} \times \mathbf{v}_{n1}] \right\}, \tag{9}$$

where A^n and A^{n+1} are the face area at time n and $n + 1$, and $\mathbf{v}_{ni}$ is the node vector $\mathbf{x}_i^{n+1} - \mathbf{x}_i^n$. Note that the side faces of the prism are curved, so its volume cannot be exactly calculated by splitting it into polyhedrons. Because Eq. (9) is an exact expression for the volume swept by a face, then U_{mesh} exactly satisfies the following geometric constraint:

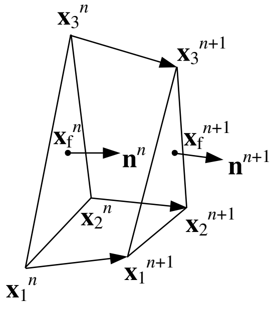

Fig. 1. Calculation of U_{mesh}: volume swept by face $\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$.

$$\frac{V^{n+1} - V^n}{\Delta t} = \sum_{\text{faces}} U_{\text{mesh}}. \tag{10}$$

The convection term in Eq. (7) is solved explicitly, using central differencing in time and a predictor–corrector technique:

$$\rho \sum_{\text{faces}} \mathbf{u}_{\text{f}}(U - U_{\text{mesh}}) \approx \rho \sum_{\text{faces}} 0.5\left[(\mathbf{u}_{\text{f}}(U - U_{\text{mesh}}))^n + (\mathbf{u}_{\text{f}}(U - U_{\text{mesh}}))^{n+1}\right]. \tag{11}$$

To this point Eqs. (6) and (7) are based on the primary unknown U. This work solves this system using an exact fractional step method [23]. When fully discretized over the whole domain, the continuity equation (6) becomes $\mathbf{DU} = \mathbf{0}$, where $\mathbf{D}$ is the divergence operator (matrix) defined by Eq. (6), and $\mathbf{U}$ is the vector of all face mass fluxes U. Let $\tilde{\mathbf{s}}$ be the stream function vector in 3D, l be an edge, and $\mathbf{l}$ be the edge vector of edge l. Then the line integral of $\tilde{\mathbf{s}}$ along edge l is

$$s = \int_l \tilde{\mathbf{s}} \cdot \mathrm{d}\mathbf{l}. \tag{12}$$

Then the face mass flux U can be exactly calculated as:

$$U = \sum_{\text{edges}} s \cdot \text{SIGN}(\mathbf{l}, \mathbf{n}). \tag{13}$$

The function SIGN($\mathbf{l}$,$\mathbf{n}$) is positive if the edge vector $\mathbf{l}$ points in the counterclockwise direction of the face normal $\mathbf{n}$ using the right-hand rule. Let $\mathbf{s}$ be the vector of all the edge integrals s. When fully discretized over the whole domain, Eq. (13) becomes $\mathbf{U} = \mathbf{Cs}$. It is easily verified that the matrix $\mathbf{C}$ constructed this way is a null space of $\mathbf{D}$, and the continuity equation (6) is satisfied exactly. Now the primary unknown changes to $\mathbf{s}$, and the momentum equation (7) becomes the only constraint on $\mathbf{s}$. Let l be an edge shared by several cells. Then divide Eq. (7) by volume V^{n+1} and take a line integral around edge l. This step removes the pressure unknown, and adjusts the number of constraints to be the same number as unknown $\mathbf{s}$.

The boundary condition on the free-surface is

$$-p\mathbf{n} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^{\text{T}}) \cdot \mathbf{n} = \frac{\sigma}{R}\mathbf{n}, \tag{14}$$

where R is the radius of surface curvature. The surface curvature is calculated using a surface fitting method. A parabolic surface is first fitted through the nearby nodes, and then the curvature is calculated using the derivatives of the surface [9,24].

To summarize the numerical scheme, the solution procedure is given below:

```
Solution procedure at each time step:
  $t = t + \Delta t$
  calculate the initial right hand side of Eq. (7) using current flow variables
  do $i = 1, 2$
    move the interface with the fluid velocity
    perform mesh smoothing using methods described in Section 4
    calculate $U_{\text{mesh}}$ using Eq. (9)
    solve Eq. (7) using the exact fractional step method
    update the velocity field
    update the convection term according to Eq. (11)
  perform mesh reconnection and interpolation using methods described in Section 3
  proceed to the next time step.
```

3. Mesh reconnection algorithm

3.1. Edge swapping algorithm

The idea of using edge swapping to construct two-dimensional (2D) Delaunay triangulation is from Lawson and Charles [25]. Lawson also generalized the concept of edge swapping to three and higher dimensions [26]. The difference between 2D and 3D swapping is that there are many more variants in 3D than 2D. In 3D, when the common face of two convex tetrahedra is swapped, three new tetrahedra are created instead of two. Therefore, there has to be two kinds of swapping in 3D: 2-3 swapping and 3-2 swapping, as shown in Fig. 2. The 3-2 swap can also be considered as a supplement for a 2-3 swap when the latter is not applicable. The 2-3 swap is only applicable when the two tetrahedra incident on their common face are convex on all common edges, like in Fig. 2. When they are not convex on one or more of their common edges (Fig. 3(a)), the 2-3 swap cannot be performed because the transformation will cause one new tetrahedron to be outside the original hull and have a negative volume. This kind of negative cell is usually called an "inverted" or "tangled" cell. In this case, if there are only three tetrahedra on the non-convex edge and they form a convex polyhedron, then the 3-2 swap can be applied to remove their common edge and create two good cells (Fig. 3(b) and (c)).

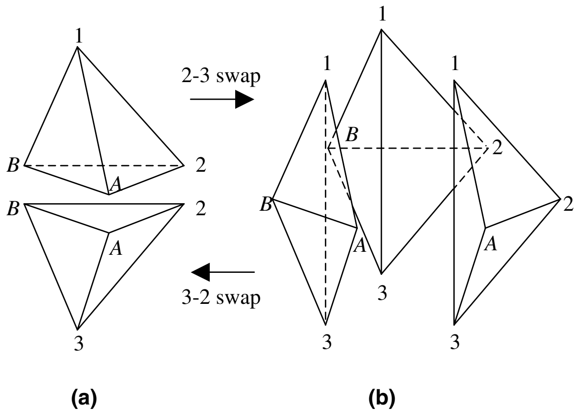

Fig. 2. Two kinds of swapping in 3D.

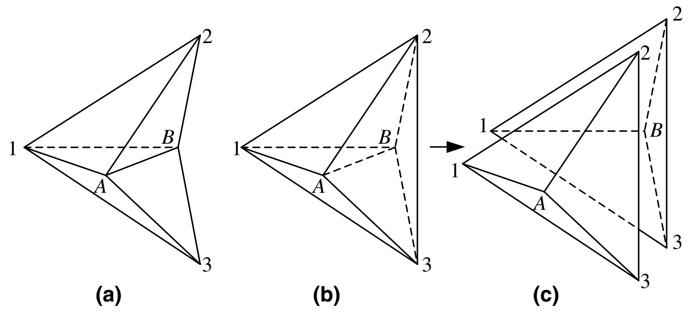

Fig. 3. 3-2 Swap in 3D: (a) 2-3 swap is not applicable because edge AB is not convex; (b) 3-2 swap is applicable if there are only three tetrahedra incident on the non-convex edge; (c) two new tetrahedral are created after a 3-2 swap.

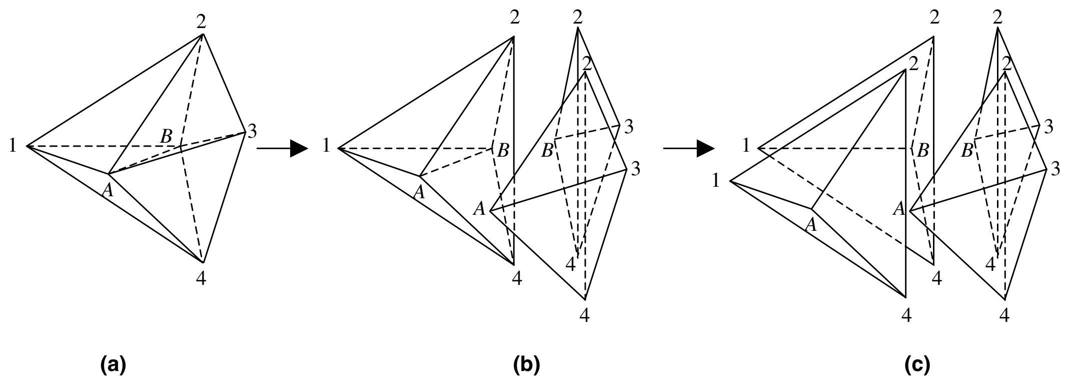

Fig. 4. Four tetrahedra share edge AB: (a) original configuration; (b) 2-3 swap to remove face AB 3; (c) 3-2 swap to remove edge AB. Polygon 1234 is defined as the "equatorial polygon".

However, when there are more than three tetrahedra sharing a common edge, as in Fig. 4(a), both 2-3 and 3-2 swaps can be locally stuck. For convenience define polygon 1234 as the "equatorial polygon". In this situation, a more complex reconnection scheme is needed to improve the mesh quality. Joe [27] showed that considering the effect of two or more consecutive swaps generally gives better results than just using 2-3 and 3-2 swaps separately, and thus the case with four tetrahedra incident to a common edge can be handled effectively. This work uses an edge swapping algorithm originally developed from Briere de L'isle and George [17] "edge removal" algorithm. An example of edge removal is shown in Fig. 4(c). Equatorial polygon 1234 is first triangulated into triangles 243 and 241, and then vertices A and B are connected to these two triangles. After these reconnections, edge AB is removed and four new tetrahedra are formed. There are many ways to triangulate a simple polygon (the total number is known as the Catalan number for a convex polygon). The optimal triangulation maximizes the minimum quality of the triangles. Here, the quality of a triangle is defined as the smaller quality of the tetrahedra on its two sides, formed by connecting vertices A and B to it as described above. Freitag and Ollivier-Gooch [21] used an algorithm that stored all the possible triangulations of the equatorial polygon, and then compared all these configurations to find the best one. To reduce memory usage, the triangulations of the equatorial polygon are stored as a rotation of many fewer "topological classes". For example, there are five possible triangulations for a 5-sided polygon, but they are all rotations of one topological class. Freitag et al. also found that it was sufficient to consider only the cases with up to seven tetrahedra incident to a common edge, therefore, triangulations of polygons with more than seven sides do not need to be stored.

Shewchuk [19] noticed that this edge removal process was equivalent to a series of 2-3 swaps followed by one last 3-2 swap. Fig. 4(b) shows such an example: one 2-3 swap removing face AB3, followed by one 3-2 swap removing edge AB. This process is equivalent to the edge removal process described above: triangulating polygon 1234 into 243 and 241, and then connecting vertices A and B to these two triangles. In the general cases with $n \geqslant 3$ tetrahedra incident to a common edge, there are $(n - 3)$ 2-3 swaps followed by one 3-2 swap. This edge swapping algorithm can be divided into several independent 2-3 and 3-2 swaps, each of which involves fewer mesh elements to reconnect. Therefore, this algorithm is easier to implement than the edge removal algorithm. When the optimal triangulation is found, it can be transformed to its equivalent swap sequence and produce the same configuration.

Shewchuk [19] also used Klincsek's dynamic programming technique [18] and developed a faster algorithm to find the optimal triangulation of the equatorial polygon, which ran in polynomial time $O(n^3)$. This algorithm does not force an upper bound on n, therefore, it can handle cases with arbitrary number of tetrahedra incident to a common edge. This feature is useful for robustness. Although Freitag and Ollivier-

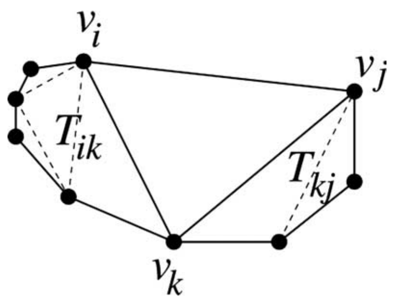

Fig. 5. Decomposition of an optimization triangulation problem (from Shewchuk [19]).

Gooch [21] found that the number of transformations that can improve the mesh quality declines dramatically with increasing n, all kinds of unusual shapes can appear in a moving mesh with new vertices inserted frequently. Since the whole numerical calculation can be destroyed by just one bad cell, it is best to be able to consider every possible configuration to provide robustness. In practice, one can certainly set an upper limit on n to save computation time, as long as the mesh quality is acceptable to the numerical calculation.

The detailed algorithm for edge swapping is described below. Assume n tetrahedra share an interior edge AB. Let R be the set of edges forming the equatorial polygon, and R_{ij} be a closed ring of edges with vertices $v_i, v_{i+1}, \ldots, v_j$ with $j \geqslant i+2$, i.e., R_{ij} is a closed ring of $v_i \to v_{i+1} \to \cdots \to v_j \to v_i$. Let T be the optimal triangulation of R, and T_{ij} be the optimal triangulation of R_{ij}. Obviously T is also T_{1n}. Define $Q[i,j]$ as a two-dimensional array. Each entry of Q stores the quality of T_{ij}, which is the minimum quality of the triangles in T_{ij}. Then the quality of T_{1n} is $Q[1,n]$, which is also the upper-right corner entry if Q is considered as a two-dimensional dynamic programming table. Dynamic programming finds this value by recursively decomposing the main problem into a series of sub-problems: finding the optimal triangulation T_{ij}. The decomposition process is shown in Fig. 5. Let $K[i,j]$ be a two-dimensional array, with each entry corresponding to $Q[i,j]$ and storing the decomposition vertex v_k defined in Fig. 5. Here v_k is a vertex of triangle t, with t belonging to the final optimal triangulation T_{ij} and containing edge v_iv_j, and $v_k \neq v_i, v_j$. Vertex v_k separates T_{ij} into two sub-problems T_{ik} and T_{kj}. Array K actually represents how the equatorial polygon is triangulated, and the final triangulation can be extracted from it using an algorithm described below. Then the quality of T_{ij} is calculated by

$$Q[i,j] = \max_{k \in [i+1, j-1]} \min(Q[i,k],\ \text{quality of } \Delta v_i v_k v_j, Q[k,j]). \tag{15}$$

Although Eq. (15) looks like a recursive process, entries of $Q[i,j]$ can actually be filled in polynomial time if arranged in a special filling sequence shown in Algorithm 1. For a detailed explanation of Algorithm 1 readers can refer to [19].

Algorithm 1. *FILLTABLES*(A,B,R) (from Shewchuk [19]):

```
do $i = n - 2$, down to 1
  do $j = i + 2$, $n$
    do $k = i + 1$, $j - 1$
      set $q = QUALITY\_TRIANGLE(v_i, v_k, v_j)$
      if $k < j - 1$
        set $q = \min(q, Q[k,j])$
      if $k > i + 1$
        set $q = \min(q, Q[i,k])$
      if $k = i + 1$ or $q > Q[i,j]$
         set $Q[i,j] = q$
         set $K[i,j] = k$
return $(Q,K)$
```

Algorithm 2. *QUALITY_TRIANGLE*(v_i,v_k,v_j)

```
set $q$ = $QUALITY\_TET(A, v_i,v_k,v_j)$
if $q < q_{\text{old}}$
   return $q$
else
   set $q = \min(q, QUALITY\_TET(v_i,v_k,v_j,B))$
   return $q$
```

Algorithm 3. *QUALITY_TET*(v_1,v_2,v_3,v_4)

```
return quality of tetrahedron $v_1v_2v_3v_4$
```

Algorithm 4. recursive *EXTRACT_TRI*(TRI,v_i,v_j,K) (adapted from [19])

```
if $(v_j - v_i >= 2)$
   set $v_k = K(v_i,v_j)$
   set $TRI\ (1,v_j - v_i - 1) = v_i$; $TRI\ (2,v_j - v_i - 1) = v_k$; $TRI\ (3,v_j - v_i - 1) = v_j$
   if $(v_k - v_i >= 2)$
      call $EXTRACT\_TRI(TRI(:, 1 : v_k - v_i - 1), v_i,v_k, K)$
   if $(v_j - v_k >= 2)$
      call $EXTRACT\_TRI(TRI(:, v_k - v_i : v_j - v_i - 2), v_k,v_j, K)$
```

Algorithm 2 *QUALITY_TRIANGLE* returns the quality of a triangle $\Delta v_iv_kv_j$. In cases where the mesh quality measure is expensive to calculate (e.g. dihedral angles), Freitag and Ollivier-Gooch's [21] technique is used to save computation time. They noticed that the tetrahedra in the final configurations appear in pairs above and below the equatorial triangles, such as tetrahedra $234A$ and $234B$ in Fig. 4(c). Thus, if one of them has a worse quality than q_{old}, the original configuration's minimum quality, these two cells will not be part of the final configuration, and it is not necessary to calculate the quality of the tetrahedron on the other side.

Algorithm 3 *QUALITY_TET* returns the quality of a tetrahedron. It also needs to detect inverted tetrahedra. Looking at Fig. 4, it is easy to get the wrong impression that the equatorial polygon 1234 is flat, and a valid tetrahedron can always be formed by connecting vertex A or B to a triangle in it. In general, polygon 1234 can be skewed and/or non-convex, and a new tetrahedron formed this way may certainly be an inverted one. Algorithm 1 assumes that qualities of inverted cells are negative, therefore, it does not perform any convexity test. This arrangement guarantees that if the old mesh does not have inverted cells, no such cells will be created in the new mesh.

Algorithm 1 returns two arrays: Q and K. The value of $Q[1,n]$ gives the mesh quality after edge swapping. If it is better than the old quality, edge swapping is applicable, and the optimal triangulation is extracted from array K using Algorithm 4 (adapted from [19]). Then the swap sequence can be arranged using Algorithms 5 and 6. In Shewchuk's original design [19], the reconnection procedure does not identify if a triangle t corresponds to a 2-3 swap or 3-2 swap. The reconnection procedure may do a 2-3 swap on a triangle that is supposed to be a 3-2 swap, so an inverted tetrahedron may be created transiently. After the final 3-2 swap, this inverted tetrahedron will be deleted and the final mesh is still valid. This approach is not adopted

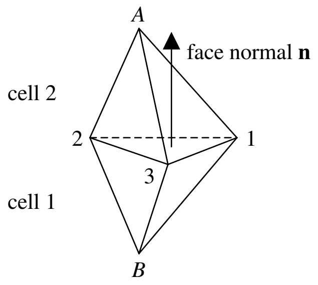

Fig. 6. Definition of a face normal from cell 1 to cell 2.

here, because inverted cells have very different geometric properties than regular cells, and their geometric calculation is much more complicated. An inverted cell is usually identified by its negative orientation. The orientation of a tetrahedron is defined by which side a specific vertex lies on, separated by the plane formed by the other three vertices. When this vertex lies on the negative side, then this tetrahedron is inverted. Inverted cells require more complex geometric calculations. For example, the calculation of the face normal is different for inverted cells. The face normal is defined as the direction from cell 1 to cell 2, as in Fig. 6. With regular cells, the face normal direction can be calculated using the vector from the center of cell 1 to the center of cell 2. However, if cell 1 is inverted, and vertex A falls to the lower side of plane 123, then the vector between the two cell centers can be arbitrary and does not give the correct direction (from cell 1 to cell 2). When this inverted cell is deleted by the final 3-2 swap, the boundary faces of this transformation (like faces A12, A23, or A31 in Fig. 3(c)) that are not involved in the edge swapping may have wrong normals.

Based on above analysis, this work identifies the kind (2-3 or 3-2) of each swap first so that the proper swap sequence can be applied. The identification method used here for a triangle t is to test if the open segment AB intersects the interior of t or the interior of any edge of t. If the answer is yes, then t corresponds to a 3-2 swap; otherwise t corresponds to a 2-3 swap. This process is summarized in Algorithm 5.

Algorithm 5. *SWAP32(TRI,A,B)*

```
do $m = 1, n - 2$
    set $v_i = TRI(1, m)$; $v_j = TRI(2, m)$; $v_k = TRI(3, m)$;
    if edge $AB$ intersects $\Delta v_i v_k v_j$ in its interior or the interior of any of its edges
        return $m$
```

Finally, the main function *EDGE_SWAP*(·) is designed to assemble these subroutines together, as in Algorithm 6. The swap of each edge is performed by calling this function.

Algorithm 6. *EDGE_SWAP(A,B)*

```
calculate $q_{\text{old}}$
set $(Q,K) = FILLTABLES(A,B,R)$
if $Q(1,n) > q_{\text{old}}$
    call $EXTRACT\_TRI(TRI,1,n,K)$
    set $t_{32} = SWAP32(TRI,A,B)$
    set $j = 0$
```

```
        do while ($j < n - 3$)
            do $i = 1, n - 2$
                if($i! = t_{32}$)
                    set $v_1 = TRI(1,m)$; $v_2 = TRI(2,m)$; $v_3 = TRI(3,m)$;
                    if triangle $\Delta v_1v_2v_3$ lies on the boundary
                        set $j = j + 1$
                        perform 2-3 swap on triangle $i$
                        interpolate flow variables for 2-3 swap
    do 3-2 swap on triangle $t_{32}$
    interpolate flow variables for 3-2 swap
```

Algorithm 6 first calculates the quality of the worst cell q_{old} which will be used to evaluate the quality of each equatorial triangle. Then the entries of arrays Q and K are filled by calling *FILLTABLES*(·). If the new quality $Q(1,n)$ is better than q_{old}, edge swapping is applicable. To set up the correct sequence, the triangulation is first extracted from K using *EXTRACT_TRI*(·), and then *SWAP*32(·) is called to decide which triangle corresponds to a 3-2 swap. This 3-2 swap will be performed as the last step after all the 2-3 swaps are done. The implementation is illustrated as in Fig. 7. This figure shows a case of six tetrahedra sharing a common edge. Vertices 1 through 6 form the equatorial polygon. Suppose the partition shown in Fig. 7 is the optimal triangulation. Let the shaded triangle Δ145 correspond to a 3-2 swap, and the rest of the triangles correspond to 2-3 swaps. A 2-3 swap triangle is ready to be swapped when it lies on the boundary of the equatorial polygon. Algorithm 6 goes through the list of all the triangles and swaps 2-3 ones when they are ready. Eventually all the 2-3 swaps will be performed in a layered fashion, and the 3-2 swap is performed as the last step.

This edge swapping procedure can be applied to every edge in the interior region of the computational domain. A stack is used to store the edges that need to be checked. Once a reconnection is performed, all the affected neighbor edges are added back to the stack to be checked again. Because for each edge swap, the minimum quality of all the elements cannot decrease, this process will not fall into a cyclic loop. Although this process usually gives only locally optimal results and may be far from global optimum, in practice, it can often transform a poor mesh into a good one. The running time of each edge is cubic in *n*. However, the main concern is with the running time based on total number of elements in the whole mesh. If *n* has a bound, then the total cost of edge swapping will remain linear in the number of edges in the domain. In the small-scale calculations in this work, edge swapping takes about 20% of CPU time. Since no matter what iteration method is used to solve the Navier–Stokes equations, the time cost cannot be

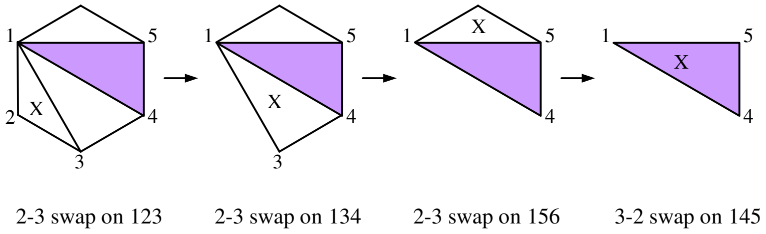

Fig. 7. Edge swap sequence illustration: a case with 6 cells sharing one common edge. Vertices 1–6 form the equatorial polygon. The shaded triangle corresponds to a 3-2 swap, and the rest of the triangles correspond to 2-3 swaps. A 2-3 swap triangle is ready to be swapped when it lies on the boundary of the equatorial polygon. The 3-2 swap is performed as the last step.

lower than linear, the CPU cost of edge swapping will be a smaller fraction for larger problems. Also, it is observed that in each time step less than 0.1% edges need to be reconnected, causing only trivial interpolation errors.

3.2. Edge bisection and edge contraction

In free-surface flow simulations, the surface curvature changes throughout the computation. This is demonstrated in the results of the capillary jet simulations in Section 6. If the mesh resolution is not high enough to catch up with the local length scale, the numerical accuracy of the curvature calculation declines dramatically. This is contrary to the Eulerian surface capturing methods, such as VOF and level set, in which all the curvature information smaller than the mesh resolution will be lost and will not cause a catastrophic result. On the other hand, if the mesh resolution is too high, there will be more than enough elements in the mesh to solve the problem, wasting CPU time. To keep the mesh resolution consistent with the local length scale, edge bisection and edge contraction are used to adjust mesh resolution. The surface mesh smoothing technique described in Section 4 can also adapt to length scale changes, while keeping the total number of nodes constant. Obviously new vertices are needed when the flow is depleting the existing ones. Successive edge bisection could cause the mesh quality to deteriorate [16]. However, with the edge swapping and mesh smoothing methods modifying both connection and vertex position, the mesh quality will be maintained.

Edge bisection is illustrated in Fig. 8. This figure shows a case of four tetrahedra sharing edge AB. If the length of AB is too long, it is split in half by a new vertex C, and the incident tetrahedra are also split in halves by this vertex. A 2D example of edge contraction is illustrated in Fig. 9. In this figure edge AB is shared by two incident cells. If AB is too short, it can be collapsed into a new vertex C, and the two incident cells are also collapsed to two edges. Edge contraction does not always produce a topologically valid new mesh. For example, edge AB of Fig. 10 cannot be collapsed. Otherwise, the resulting mesh will have two different triangles with exactly the same three vertices. Dey et al. [28] studied the conditions under which the topological structure is preserved after edge contraction. They proved that in 2D and 3D, when the intersection of the links of two vertices A and B of the edge to be collapsed equal the links of this edge, the topological type will be conserved after edge contraction. The links of a simplex t (such as a vertex, edge, face or cell) are defined as all the simplices contained in those simplices containing t, except those directly attached to t.

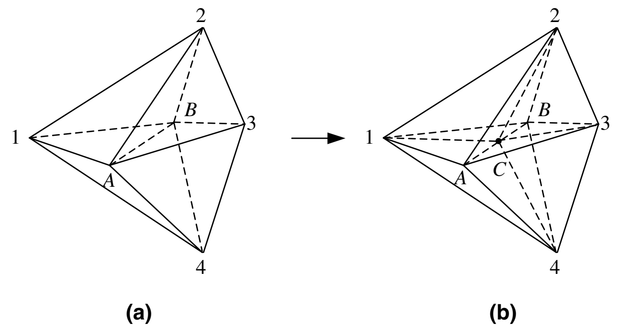

Fig. 8. A 3D example of edge bisection.

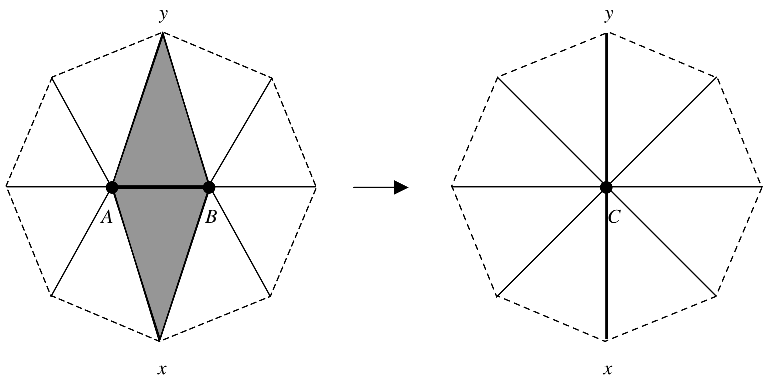

Fig. 9. A 2D example of edge contraction. Edge AB is collapsed to a new vertex C. In the left figure: Link(a) ∩ Link(b) = Link(edge AB) = $\{x,y\}$ (from Dey et al. [28]).

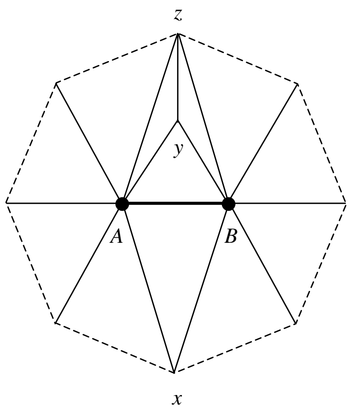

Fig. 10. A 2D example of an edge that cannot be collapsed. Link(a) ∩ Link(b) = $\{x,y,z,yz\} \neq$ Link(edge AB) = $\{x,y\}$ (from Dey et al. [28]).

3.3. Flow variable interpolation after reconnection

Because mesh reconnection creates new mesh elements, the flow variables need to be projected to these new elements. From the description of the numerical scheme and the solution procedure in Section 2, one can see that only the line integral s of the stream function needs to be interpolated. Because stream function s is defined on each edge, only cases with edge changes need interpolation.

There are four reconnection cases to consider: 2-3 swap, 3-2 swap, edge bisection, and edge contraction. Because the 3-2 swap (Fig. 3) does not create new edges, no interpolation is necessary.

Now consider the edge bisection case (Fig. 8). Assume there are n tetrahedra incident to edge AB. Then after edge bisection, there are n new edges that are incident to C, and AB is split into AC and CB. Because s is defined as a line integral, edges AC and CB will have a portion of the s value of AB. For the other new edges that are incident to C, the stream functions are calculated as follows. First the velocities of the new cells are set according to the old cells. Because every new cell is part of an old cell, they should have the same velocity as the corresponding old ones. Then the stream function on the new edges are approximated from these velocities and the relation between s and $\mathbf{u}_c$ described in Section 2. Usually, the number of un-

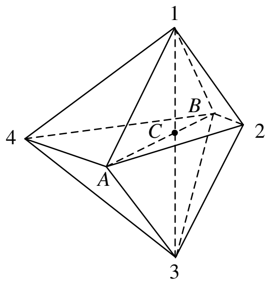

Fig. 11. A degenerate 2-3 swap on face $AB2$.

knowns (stream function) and constraints (cell velocity) are not equal, and a least squares fitting method is used in these situations.

For the edge contraction case, assume edge AB is collapsed to a new vertex C. Then all the edges incident to C need a new stream function value. The interpolation is handled like mesh smoothing, which can be illustrated using the 2D example in Fig. 9. For the un-shaded cells that are not collapsed, their structures will remain the same but their boundaries are moved, because vertices A and B are moved to a new position C. So for these cells the operation of edge contraction is just like mesh smoothing, and the momentum flux can be calculated using U_{mesh} from Eq. (9) and $\mathbf{u}_c$ from the neighbor cells. Then the velocity of each affected cell can be calculated using momentum conservation, and the stream function can be calculated from the cell velocities. Similar to edge bisection, a least squares fitting method is used here if necessary. If all these affected cells (i.e., incident to the new vertex C) are set to have the same average velocity of the original cells, this is a first order approximation. So by considering the flux caused by mesh motion, this interpolation method is approximately second order accurate.

For the interpolation for the 2-3 swap, consider the example in Fig. 2(a) and (b). In the original configuration, two tetrahedra $AB21$ and $AB23$ share a common face $AB2$. After the 2-3 swap, a new edge 13 is created, so the stream function s_{13} needs to be interpolated on edge 13. The cell velocities of the three new cells are set based on momentum conservation, and then s_{13} is calculated using a similar least squares fitting method described above. If this 2-3 swap is one of a sequence of edge swaps and the common edge is AB, then the new tetrahedron $AB13$ is a temporary cell and will be deleted after the final 3-2 swap. Therefore there is no need to consider this cell's velocity when calculating s_{13}. Also this cell may not have a better quality than q_{old} and including it in the interpolation process may cause the condition number of the least squares matrix to deteriorate. In the worst case, cell $AB13$ could be a degenerate cell, when the new edge 13 intersects edge AB, as shown in Fig. 11. In this case cell $AB13$ may have zero volume. Up to the machine round-off error, this volume can also be negative. However, this negative volume will not cause the same geometric effect as the inverted cells, since it will not give a wrong value to the face normal. This can be illustrated using Fig. 6. In this figure, if vertex A falls onto the plane 123, or a little below due to the round-off error, the difference between the two cell centers still gives the correct direction of face normal: from cell 1 to cell 2. So this degenerate cell will not cause error in the geometric calculation. However, since it has zero volume, including it in the interpolation process may cause a catastrophic error for s_{13}. Therefore, this cell should definitely be excluded in order to achieve robustness.

4. Mesh smoothing

Mesh smoothing methods relocate the vertex position to improve the mesh quality without changing mesh topology, as shown in Fig. 12. Laplacian smoothing is the simplest and fastest smoothing method,

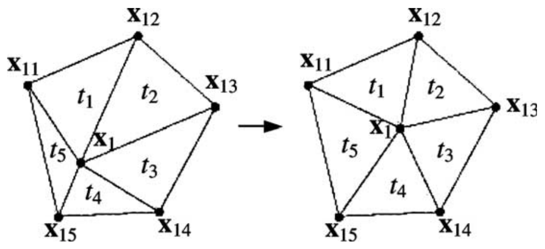

Fig. 12. Mesh smoothing: a local submesh consisting of a free vertex $\mathbf{x}_1$, its direct neighbors $\mathbf{x}_{11},\ldots,\mathbf{x}_{15}$, and its incident elements $t_1,\ldots,t_5$. The right figure shows the mesh after vertex $\mathbf{x}_1$ is smoothed (from Freitag and Ollivier-Gooch [21]).

and is widely used in moving-mesh schemes [15,16,22]. This method adjusts the location of each mesh vertex to the geometric center of its neighbor vertices. Consider an interior vertex $\mathbf{x}_i$ and its direct neighbors $\mathbf{x}_{ij}$. The Laplacian scheme and its weighted-average version can be expressed as:

$$\sum_j (\mathbf{x}_{ij} - \mathbf{x}_i^{\text{new}}) = 0, \tag{16}$$

$$\sum_j k_{ij}(\mathbf{x}_{ij} - \mathbf{x}_i^{\text{new}}) = 0, \tag{17}$$

where k is usually called a spring constant. Eq. (17) can also be converted to a parabolic system

$$\mathbf{x}_i^{\text{new}} - \mathbf{x}_i^{\text{old}} = \sum_j k_{ij}(\mathbf{x}_{ij} - \mathbf{x}_i). \tag{18}$$

Similar ideas can be applied to the surface mesh. Consider a free vertex $\mathbf{x}_i$ and its direct neighbors $\mathbf{x}_{ij}$, all on the interface. Let $\hat{\mathbf{n}}$ be the surface normal. To conserve volume and the interface profile, the free vertex should only move in the tangential plane of the surface

$$\mathbf{x}_i^{\text{new}} - \mathbf{x}_i^{\text{old}} = (\mathbf{I} - \hat{\mathbf{n}}\hat{\mathbf{n}}^{\text{T}}) \sum_j k_{ij}(\mathbf{x}_{ij} - \mathbf{x}_i), \tag{19}$$

Eq. (19) can also be solved implicitly to obtain better stability. Since $\hat{\mathbf{n}}\hat{\mathbf{n}}^{\text{T}}$ is non-linear in terms of $\mathbf{x}_i$, Eq. (19) is solved by a predictor–corrector scheme

$$\mathbf{x}_i^{\text{new}} - \mathbf{x}_i^{\text{old}} = \sum_j k_{ij}(\mathbf{x}_{ij}^{\text{new}} - \mathbf{x}_i^{\text{new}}) - \mathbf{f}_n, \quad \mathbf{f}_n = \left[\hat{\mathbf{n}}\hat{\mathbf{n}}^{\text{T}} \sum_j k_{ij}(\mathbf{x}_{ij} - \mathbf{x}_i)\right]^{\text{new}}, \tag{20}$$

where the term $\mathbf{f}_n$ is iteratively solved using updated values of $\mathbf{x}_i$ until convergence is reached. The reason that the lagged term is $\mathbf{f}_n$ instead of just $\hat{\mathbf{n}}\hat{\mathbf{n}}^{\text{T}}$ is because the different coordinates in Eq. (20) can be decoupled and separated into smaller independent problems. To adjust mesh resolution, the spring constant k can also be set according to the local length scale L

$$k_{ij} = c\frac{|\mathbf{x}_{ij} - \mathbf{x}_i|}{L}. \tag{21}$$

The disadvantage of Laplacian smoothing is that it is not directly related to a well-behaved element quality measure. Laplacian smoothing can make all the vertices distributed uniformly, since Eq. (17) is equivalent to minimizing $\sum_{\text{edges}} k_{\text{e}} l_{\text{e}}^2$ over the whole domain, where l_{e} is the edge length. But Laplacian smoothing may cause some cells in the mesh to become badly shaped or even inverted, as demonstrated by Freitag and Ollivier-Gooch [21]. In 3D, this problem is more common since each vertex has more neighbors than in 2D.

Hence Freitag et al.'s [20,21] local optimization-based smoothing method is more attractive, which is adopted in this work. This method directly optimizes the mesh quality, so it guarantees that a better mesh will always be generated after smoothing. The local optimization-based smoothing method adjusts the vertices one by one, and only solves a local optimization problem to maximize the minimum quality of local cells. Consider Fig. 12, where the positions of vertices $\mathbf{x}_{11}$–$\mathbf{x}_{15}$ are fixed, and the free vertex $\mathbf{x}_1$ can be moved to a new position. The objective is to move vertex $\mathbf{x}_1$ to such a position that the minimum quality of the local elements t_1–t_5 is maximized. Let function $f_i(\mathbf{x}_1)$ be the quality of triangle t_i. Then the mesh optimization problem can be formulated as the following system:

$$\begin{array}{ll} \max & \phi(\mathbf{x}_1) = \min f_i(\mathbf{x}_1) \\ \text{such that} & \mathbf{x}_1 \in \text{feasible region}, \end{array} \tag{22}$$

Freitag et al. used a steepest descent method to solve this problem. When several cells share the same minimum quality, a non-smooth optimization problem needs to be solved to find the direction along which the qualities of all these cells can be improved. If no such direction exists, the optimal position is reached. Freitag et al. converted this non-smooth optimization problem to a convex (positive definite) quadratic programming problem, which is known to have a polynomial solution and has many well-developed algorithms [29–31].

Freitag et al.'s optimization-based smoothing method can also be used for surface mesh smoothing, using the similar idea of fixing the free vertex in the tangential plane of the surface. However, since this is a local method, it may move a surface vertex to a relatively longer distance than the Laplacian method of Eq. (20). This causes disturbances on the surface because the tangential plane is only a first order approximation of the surface profile. Since the surface mesh is essentially two-dimensional, where Laplacian smoothing can be acceptable, this work uses the Laplacian method for surface smoothing, and the optimization-based method for interior smoothing. Unlike mesh reconnections described in Section 3, there is no need to do flow variable interpolation after mesh smoothing. This can be seen from the description of the numerical scheme and the solution procedure in Section 2. The flux change caused by mesh motion is already included in the momentum equation (7).

5. Numerical tests

The mesh quality measure used in this work is Knupp's algebraic quality metric for a tetrahedron [32]. For consistency purposes this metric is used for both edge swapping and optimization-based mesh smoothing. Let $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ be the coordinates of four vertices of a tetrahedron. Define its Jacobian matrix $\mathbf{A}$ as

$$\mathbf{A} = (\mathbf{x}_3 - \mathbf{x}_0, \mathbf{x}_2 - \mathbf{x}_0, \mathbf{x}_1 - \mathbf{x}_0). \tag{23}$$

Let λ_{ij} be the entries of matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$. Then the quality of this tetrahedron is defined as:

$$q = \frac{3(\det(\mathbf{A})\sqrt{2})^{2/3}}{1.5(\lambda_{11} + \lambda_{22} + \lambda_{33}) - (\lambda_{12} + \lambda_{23} + \lambda_{13})}. \tag{24}$$

This quality measure is based on the transformation between a real cell and an ideal cell, therefore, it can effectively identify all kinds of bad cells.

The 3D test cases presented here are very similar to Baker et al.'s [16] 2D demonstrations. Fig. 13 shows a sphere translating through a channel at a constant velocity, controlled by a CFL number of 0.3. Fig. 14 shows a cube rotating in a sphere with constant angular velocity, also at a CFL number of 0.3. In both cases significant deformations are simulated with acceptable mesh qualities, minimum dihedral angle being 14.4° and 8.27°, respectively.

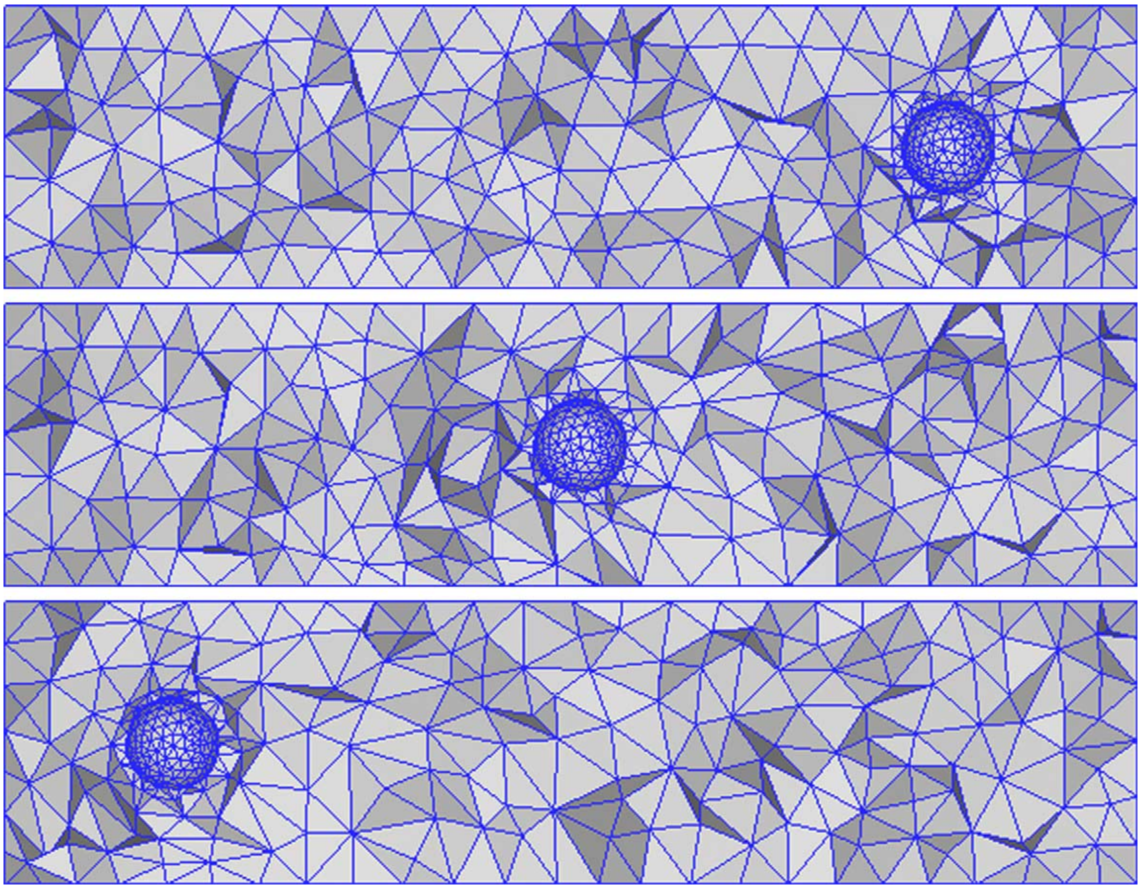

Fig. 13. The slice view of a sphere translating through a channel. The minimum dihedral angle during the process is 14.4°.

6. Numerical simulations

Three simulations are presented to demonstrate the robustness and accuracy of this scheme for flows with large deformation: ligament collapsing, capillary jet breakup, and collision of two equal size droplets.

6.1. Ligament collapsing

The first simulation is a liquid ligament collapsing into a droplet under surface tension. This case is important in spray modeling since ligaments are frequently formed in droplet collisions, and they may break into satellite drops or collapse into a single drop. The initial condition is a cylinder with a hemisphere at each end. Under the force of surface tension, the ligament is pulled into a spherical shape. The sphere's oscillations quickly die out due to the damping effect of viscosity. In this case, the ligament is short enough that it does not pinch into two droplets. For this calculation the Ohnesorge number, $Oh = \mu/(\rho r_0 \sigma)^{1/2}$ is 0.2 and the characteristic time, $t_c = (\rho r_0^3/\sigma)^{1/2}$ is 1.150 ms. A sequence of images is shown in Fig. 15.

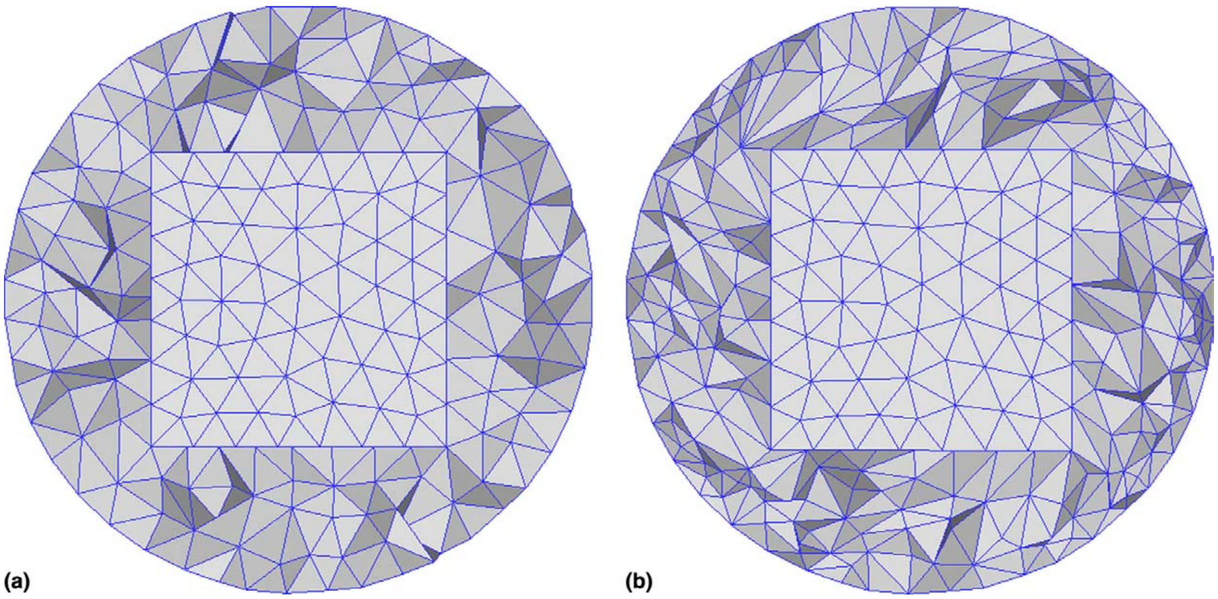
Fig. 14. The slice view of a cube rotating in a sphere: (a) original mesh; (b) after 360° rotation. The minimum dihedral angle during the process is 8.27°.

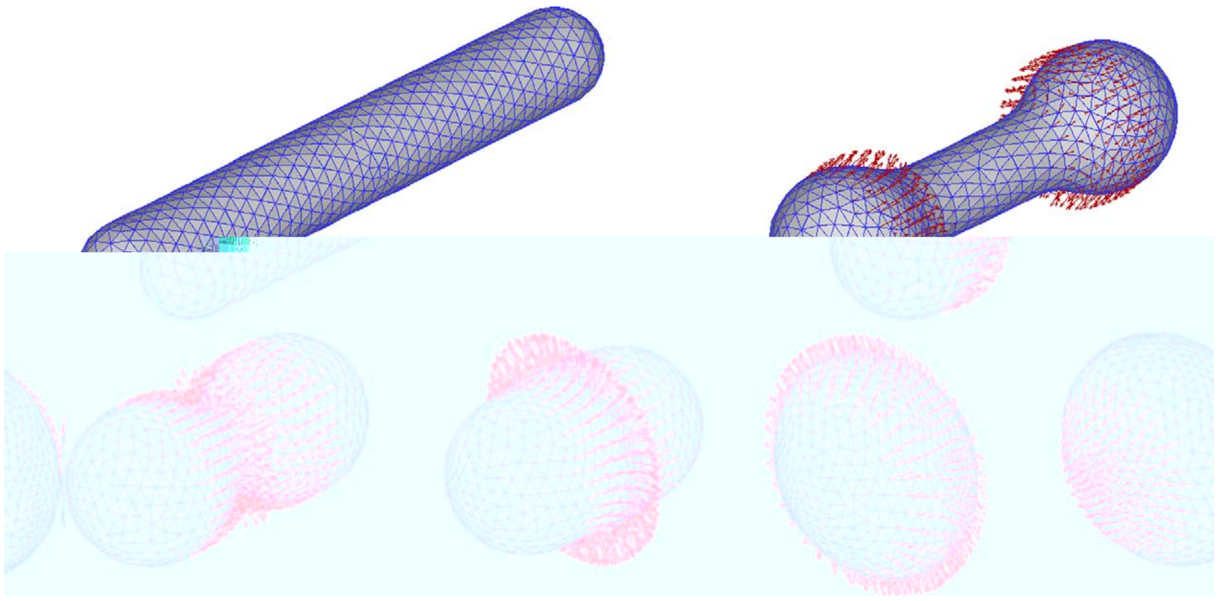
Fig. 15. Ligament collapsing into a droplet under surface tension. Vectors indicate velocity of the surface. The Ohnesorge number, $Oh = \mu/(\rho r_0 \sigma)^{1/2}$ is 0.2. Figures are taken at times: 0, 3.48, 5.74, 6.59, 7.63, 9.89, non-dimensionalized by $t_c = (\rho r_0^3/\sigma)^{1/2} = 1.150$ ms.

6.2. Capillary jet breakup

The capillary instability of a cylindrical jet is simulated with a periodic boundary condition. The mechanism of a liquid jet breaking up into droplets is of fundamental importance for spray simulations. An axisymmetric numerical analysis of this problem was done by Ashgriz and Mashayek [33]. Three different cases are presented here to show the capability of this dynamic meshing scheme. In the first case, the initial disturbance ε of the jet profile is 2.0% of the initial radius of the cylinder r_0. The characteristic time, $t_c = (\rho r_0^3/\sigma)^{1/2}$ is 0.0364 s. The wavelength λ of the disturbance is ten times as large as r_0. The non-dimensional wave number kr_0 is 0.628, which falls into the unstable regime of the linear theory. Fig. 16 shows the jet images at different times of this simulation. The calculation is stopped when the minimum radius reaches 5% of r_0, and this point is defined as the breakup point. The breakup time for this case is 13.01, non-dimensionalized by the characteristic time. This figure shows that the dynamic meshing scheme can adapt well to the surface length scales, while the curvature is continuously increasing in the neck region.

Fig. 17 shows the comparison of the disturbance growth between the numerical and the analytical solutions. The analytical solutions are taken from Lord Rayleigh's inviscid linear theory [34], Weber's viscous linear analysis [35], and Bogy's viscous linear stability analysis [36]. In order to be comparable to linear analysis, the initial velocity potential ϕ is set to be exactly the same as in the linear analysis

$$\phi = BJ_0(ikr)\sin(kx), \tag{25}$$

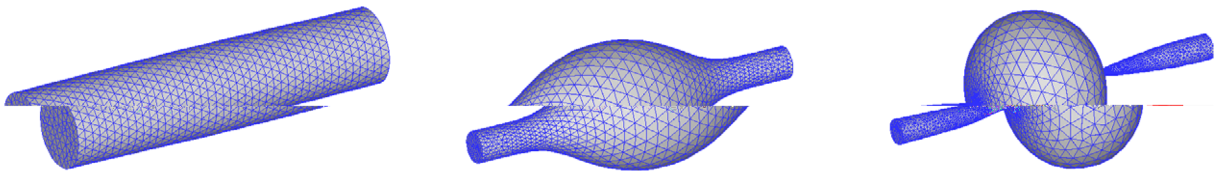

Fig. 16. Capillary jet breakup: $kr_0 = 0.628$; $\varepsilon = 0.02$. Images are at non-dimensional times of 0, 11.37, and 13.01 (breakup time).

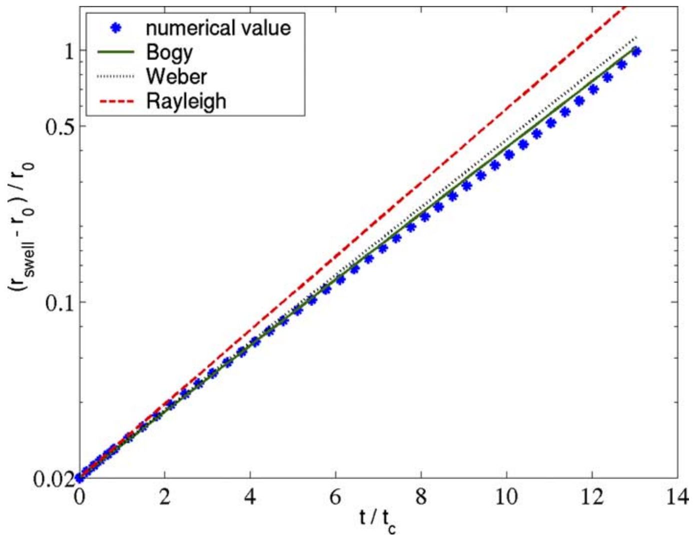

Fig. 17. Disturbance growth process: comparison between numerical and analytical solutions. $kr_0 = 0.628$; $\varepsilon = 0.02$; $J = \sigma r_0/(\rho v^2) = 238.34$.

where the disturbance amplitude B is given by:

$$B = -\frac{\varepsilon r_0 c}{\mathrm{i}kJ_1(\mathrm{i}kr_0)}, \tag{26}$$

$$c^2 = \frac{\sigma}{\rho r_0^3}\frac{I_1(kr_0)}{I_0(kr_0)}kr_0(1 - k^2r_0^2), \tag{27}$$

Eq. (27) is from Lord Rayleigh [34], and is represented as the dashed line in Fig. 17. The constant c represents the disturbance growth rate. Weber's viscous solution [35] is given by Eq. (28), and is represented as the dotted line in Fig. 17

$$c^2 + c\frac{3v}{r_0^2}k^2r_0^2 - \frac{\sigma}{2\rho r_0^3}(1 - k^2r_0^2)k^2r_0^2 = 0. \tag{28}$$

Bogy used the results from Green's Cosserat theory [37], and performed linear stability analysis for viscous jet [36]. His solution is given in Eq. (29) and represented as the solid line in Fig. 17. In Eq. (29), the velocity u_0 can be eliminated, so the growth rate c is still independent of u_0

$$\left(c\frac{r_0}{u_0}\right)^2 + \left(c\frac{r_0}{u_0}\right)\frac{24k^2r_0^2 + k^4r_0^4}{Re(8 + k^2r_0^2)} + \frac{4k^4r_0^4 - k^2r_0^2}{We(8 + k^2r_0^2)} = 0, \quad Re = r_0u_0/v, \quad We = \rho r_0u_0^2/\sigma, \tag{29}$$

Fig. 17 shows that when the disturbance is small, the numerical growth rate agrees very well with Bogy's solutions. The reason for the exact match at the starting point $t = 0.0$ is because the analytical growth rate c is used in the initial condition (27). However, the numerical simulation still reveals that the growth process is exponential and can follow the analytical solution until the disturbance has grown quite large. Fig. 18 shows a similar case with a different wave number $kr_0 = 0.4312$. This simulation is also in good agreement with Bogy and Weber's solutions.

Fig. 19 shows another simulation for jet breakup. The initial condition in this case is a perfect cylinder with a sinusoidal disturbance in the velocity field. The initial Reynolds number, $Re = r_0u_0/v$, is 18 and

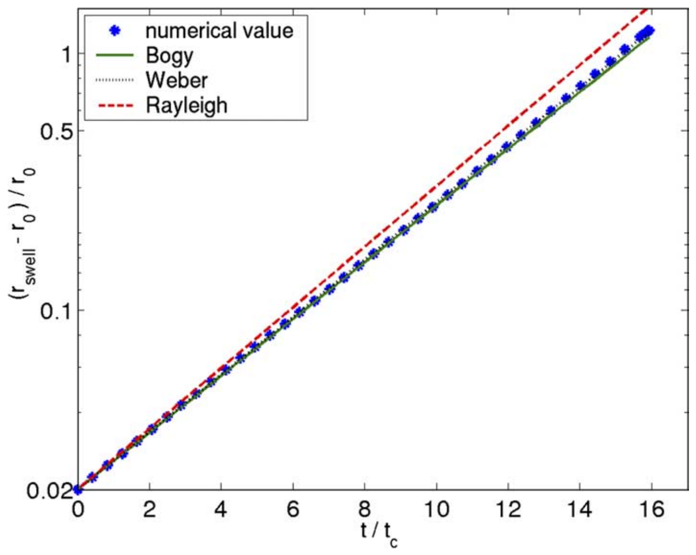

Fig. 18. Disturbance growth process: comparison between numerical and analytical solutions. $kr_0 = 0.4312$; $\varepsilon = 0.02$; $J = \sigma r_0/(\rho v^2) = 238.34$.

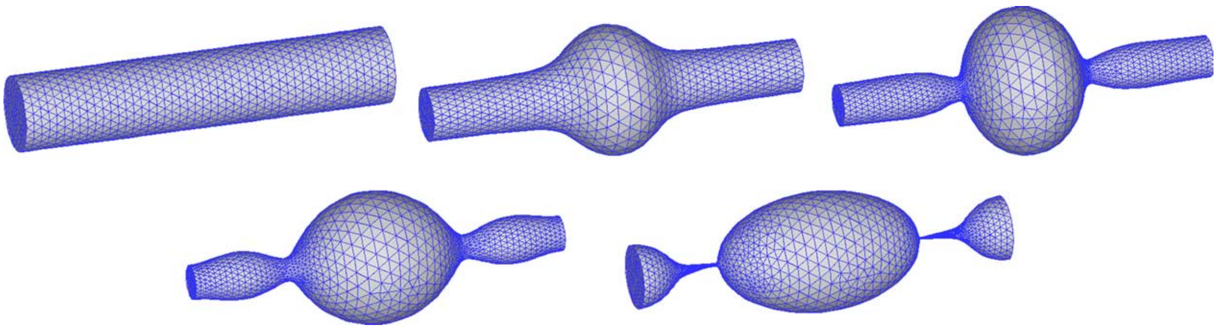
Fig. 19. Capillary jet breakup: $kr_0 = 0.628$, initial $Re = r_0u_0/v = 18$ and $We = \rho r_0 u_0^2/\sigma = 1.4$. Images are at non-dimensional times of 0, 1.49, 3.23, 4.49, and 6.49 (breakup time).

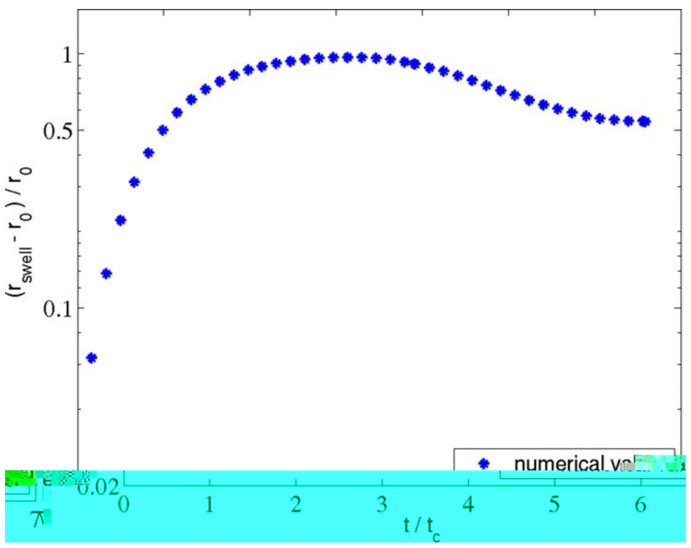

Fig. 20. Disturbance growth process. $kr_0 = 0.628$, initial $Re = r_0u_0/v = 18$ and $We = \rho r_0 u_0^2/\sigma = 1.4$. The dimensionless breakup time is 6.49.

Weber number, $We = \rho r_0 u_0^2/\sigma$, is 1.4. The wavelength λ of the disturbance is also $10r_0$. Fig. 20 shows the disturbance growth process. The dimensionless breakup time is 6.49. Because a much larger initial velocity is used in this case, the breakup time is much shorter than the analytical solutions for the same wave number, and the deformation growth process is also very different from the above two cases.

6.3. Droplet collision

Fig. 21 shows the images of head-on collision between two equal size droplets. The initial condition is two spherical droplets connected by a thin bridge. The initial radius r_0 of a single droplet is 0.168 mm. Let u_0 be the initial velocity of a single droplet. The collision Weber number and Reynolds number are defined using the initial diameter and the relative velocity

$$Re = 2d_0u_0/v, \quad We = 4\rho d_0 u_0^2/\sigma. \tag{30}$$

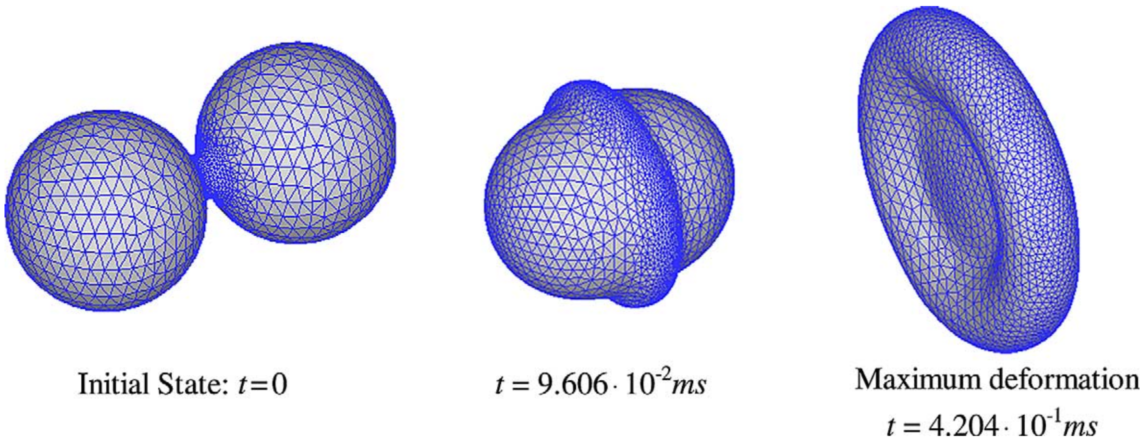

Fig. 21. Collision of two equal size droplets. Droplet radius $r_0 = 0.168$ mm, $Re = 2d_0u_0/\nu = 296.5$, $We = 4\rho d_0 u_0^2/\sigma = 61.4$.

The Weber number for this simulation is 61.4, and the Reynolds number is 296.5. Define the dissipated energy Φ and deformation S as:

$$\Phi = \frac{1}{2}\mu \int_0^t \mathrm{d}t \int \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)^2 \mathrm{d}V, \tag{31}$$

$$S = \frac{\text{Surface area}}{8\pi r_0^2}. \tag{32}$$

The total non-dimensional dissipation until maximum deformation is $\Phi/(2\pi\rho r_0^3 u_0^2) = 0.3642 \sim \mathrm{O}(1)$. The maximum deformation is $S = 1.407$. These results agree well with the experimental and analytical results from Jiang et al. [38]: $\Phi/(2\pi\rho r_0^3 u_0^2) \sim \mathrm{O}(1)$ and $S \approx 1.4$. For numerical results of the viscosity effect on maximum deformation during head-on collisions, readers can refer to Dai and Schmidt [39].

7. Conclusions

A three-dimensional unstructured moving-mesh scheme for free-surface flows is presented in this work. This scheme is capable of simulating large deformation and exactly tracking the free-surface profile. An edge swapping algorithm and an optimization-based mesh smoothing algorithm are used to maintain mesh quality as the domain deforms. The robustness and accuracy of this scheme are demonstrated with simulations of deforming liquid ligaments, periodic jets, and colliding droplets. Comparison with the experimental and analytical results shows good agreement.

Though demonstrated for free-surface flows, the re-meshing scheme can also be used to simulate flows with moving control surfaces, fluid–structure interaction, reciprocating machinery, etc. In these circumstances, the movement of boundaries can be exactly solved or prescribed with a moving-mesh method. This algorithm is also useful where adaptive mesh resolution is needed, e.g. capturing shocks in transient supersonic flows. For such problems, the region of high resolution can be dynamically captured. Other interesting problems, such as topological changes, multi-phase flows, and parallelization will be addressed in future work. Preliminary results were published in [40], in which mesh separation and two-phase flow simulations were presented.

Acknowledgments

The authors thank Professor J. Blair Perot at University of Massachusetts Amherst for sharing both software and hardware resources for this project. The authors also thank Professor Stephen de Bruyn Kops for his expertise and help. This work is performed under ONR Contract N00014-02-1-0507.

References

[1] S.W.J. Welch, Local simulation of two-phase flows including interface tracking with mass-transfer, J. Comput. Phys. 121 (1) (1995) 142–154.
[2] B. Lafaurie, C. Nardone, R. Scardovelli, S. Zaleski, G. Zanetti, Modeling merging and fragmentation in multiphase flows with SURFER, J. Comput. Phys. 113 (1) (1994) 134–147.
[3] Y. Zhao, H.H. Tan, B.L. Zhang, A high-resolution characteristics-based implicit dual time-stepping VOF method for free surface flow simulation on unstructured grids, J. Comput. Phys. 183 (1) (2002) 233–273.
[4] J.U. Brackbill, D.B. Kothe, C. Zemach, A continuum method for modeling surface-tension, J. Comput. Phys. 100 (2) (1992) 335–354.
[5] A. Theodorakakos, G. Bergeles, Simulation of sharp gas–liquid interface using VOF method and adaptive grid local refinement around the interface, Int. J. Numer. Meth. Fluids 45 (4) (2004) 421–439.
[6] J.P. Wang, A.G.L. Borthwick, R.E. Taylor, Finite-volume-type VOF method on dynamically adaptive quadtree grids, Int. J. Numer. Meth. Fluids 45 (5) (2004) 485–508.
[7] D. Enright, R. Fedkiw, J. Ferziger, I. Mitchell, A hybrid particle level set method for improved interface capturing, J. Comput. Phys. 183 (1) (2002) 83–116.
[8] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, Y.J. Jan, A front-tracking method for the computations of multiphase flow, J. Comput. Phys. 169 (2) (2001) 708–759.
[9] A.Z. Zinchenko, M.A. Rother, R.H. Davis, A novel boundary-integral algorithm for viscous interaction of deformable drops, Phys. Fluids A – Fluid Dynam. 9 (6) (1997) 1493–1511.
[10] V. Cristini, J. Blawzdziewicz, M. Loewenberg, An adaptive mesh algorithm for evolving surfaces: Simulations of drop breakup and coalescence, J. Comput. Phys. 168 (2) (2001) 445–463.
[11] J. Fukai, Z. Zhao, D. Poulikakos, C.M. Megaridis, O. Miyatake, Modeling of the deformation of a liquid droplet impinging upon a flat surface, Phys. Fluids A – Fluid Dynam. 5 (11) (1993) 2588–2599.
[12] J. Fukai, Y. Shiiba, T. Yamamoto, O. Miyatake, D. Poulikakos, C.M. Megaridis, Z. Zhao, Wetting effects on the spreading of a liquid droplet colliding with a flat surface – experiment and modeling, Phys. Fluids A – Fluid Dynam. 7 (2) (1995) 236–247.
[13] H.H. Hu, N.A. Patankar, M.Y. Zhu, Direct numerical simulations of fluid–solid systems using the arbitrary Lagrangian–Eulerian technique, J. Comput. Phys. 169 (2) (2001) 427–462.
[14] D.E. Fyfe, E.S. Oran, M.J. Fritts, Surface-tension and viscosity with Lagrangian hydrodynamics on a triangular mesh, J. Comput. Phys. 76 (2) (1988) 349–384.
[15] B.T. Helenbrook, A two-fluid spectral-element method, Comp. Meth. Appl. Mech. Eng. 191 (3–5) (2001) 273–294.
[16] Timothy J. Baker, Peter A. Cavallo, Dynamic adaptation for deforming tetrahedral meshes, AIAA Paper (1999) 99–3253.
[17] Eric Briere de L'isle, Paul Louis George, Optimization of tetrahedral meshes, Model. Mesh Generation Adaptive Numer. Meth. Partial Different. Eq., IMA Vol. 72 (1995) 97–127.
[18] G. Klincsek, Minimal triangulations of polygonal domains, Ann. Discrete Math. 9 (1980) 121–123.
[19] Jonathan R. Shewchuk, Two discrete optimization algorithms for the topological improvement of tetrahedral meshes, Manuscript, 2002. Available from: http://www-2.cs.cmu.edu/~jrs/jrspapers.html.
[20] Lori Freitag, Mark Jones, Paul Plassmann, An efficient parallel algorithm for mesh smoothing, in: Fourth International Meshing Roundtable Proceedings, Sandia National Laboratories, Albuquerque, New Mexico, 1995.
[21] L.A. Freitag, C. Ollivier-Gooch, Tetrahedral mesh improvement using swapping and smoothing, Int. J. Numer. Meth. Eng. 40 (21) (1997) 3979–4002.
[22] B. Perot, R. Nallapati, A moving unstructured staggered mesh method for the simulation of incompressible free-surface flows, J. Comput. Phys. 184 (1) (2003) 192–214.
[23] C. Wang, F. Giraldo, B. Perot, Analysis of an exact fractional step method, J. Comput. Phys. 180 (1) (2002) 183–199.

[24] M.Z. Dai, H. Wang, B. Perot, D.P. Schmidt, Direct interface tracking of droplet deformation, Atomization Spray 12 (5–6) (2002) 721–735.

[25] Charles L. Lawson, Software for C1 surface interpolation, in: Mathematical Software III, Proceedings of a Symposium Conducted by the Mathematics Research Center, The University of Wisconsin–Madison, Academic Press, New York, 1977.

[26] Charles L. Lawson, Properties of *n*-dimensional triangulations, Comp. Aided Geom. Des. 3 (4) (1986) 231–246.

[27] B. Joe, Construction of 3-dimensional improved-quality triangulations using local transformations, SIAM J. Sci. Comput. 16 (6) (1995) 1292–1307.

[28] T.K. Dey, H. Edelsbrunner, S. Guha, D.V. Nekhayev, Topology preserving edge contraction, Publ. Inst. Math. (Beograd) (N.S.) 66 (1999) 23–45.

[29] J.R. Bunch, L. Kaufman, A computational method for the indefinite quadratic-programming problem, Linear Algebra Appl. 34 (1980) 341–370.

[30] N.I.M. Gould, M.E. Hribar, J. Nocedal, On the solution of equality constrained quadratic programming problems arising in optimization, SIAM J. Sci. Comput. 23 (4) (2001) 1375–1394.

[31] N.I.M. Gould, P.L. Toint, An iterative working-set method for large-scale nonconvex quadratic programming, Appl. Numer. Math. 43 (1–2) (2002) 109–128.

[32] P.M. Knupp, Algebraic mesh quality metrics for unstructured initial meshes, Finite Elem. Anal. Des. 39 (3) (2003) 217–241.

[33] N. Ashgriz, F. Mashayek, Temporal analysis of capillary jet breakup, J. Fluid Mech. 291 (1995) 163–190.

[34] Lord Rayleigh, On the instability of jets, Proc. Lond. Math. Soc. 10 (1879) 4–13.

[35] C. Weber, Zum zerfall eines flussigkeitsstrahles, Z. Angew. Math. Mech. 11 (1931) 136–141.

[36] D.B. Bogy, Use of one-dimensional cosserat theory to study instability in a viscous-liquid Jet, Phys. Fluids 21 (2) (1978) 190–197.

[37] A.E. Green, On the non-linear behavior of fluid jets, Int. J. Eng. Sci. 14 (1) (1976) 49–63.

[38] Y.J. Jiang, A. Umemura, C.K. Law, An experimental investigation on the collision behavior of hydrocarbon droplets, J. Fluid Mech. 234 (1992) 171–190.

[39] Meizhong Dai, David P. Schmidt, Numerical simulation of head-on droplet collision: Effect of viscosity on maximum deformation, Physics of Fluids 17 (4) (2005).

[40] Meizhong Dai, Shaoping Quan, David P. Schmidt, Direct numerical simulation of multiphase flows, in: 17th Annual Conference of ILASS – Americas: Institute for Liquid Atomization and Spray Systems, Arlington, VA, 2004.