Fluid Animation with Explicit Surface Meshes and Boundary-Only Dynamics

by

Tyson Brochu

B.Sc., University of Regina, 2004

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

September 28, 2006

Abstract

An explicit method for fluid surface tracking is presented. The method represents the surface as a triangulated mesh of points in space, rather than as an implicit surface function. Utilizing well-developed algorithms designed for collision detection and resolution in cloth simulation, the system is able to handle topology changes robustly and efficiently. When fluid surfaces collide, we perform topology changes only in the most trivial cases — we reject any degenerate cases and use the cloth algorithm to keep the surfaces separated.

Taking advantage of the explicit surface representation, we introduce new approaches to simulating surface tension and conserving fluid volume. Finally, we propose a boundary element method for enforcing fluid incompressibility which uses only data points on the fluid surface, rather than a full volumetric discretization of the fluid over a grid.

Contents

List of Figures

Acknowledgements

Thanks to my supervisor Robert Bridson for knowing all the answers before I knew the questions.

Thanks to Dinos for rescuing my iPod from Vienna Airport.

Thanks to the whole Imager crew for creating such a cool work environment: Dave, Dinos, Christopher, Biff, Hagit, Vlady, Ken, Tibi, Abhi, Trent, Derek, Brad, Kevin, Peter, Aaron, Llach and anyone else I forgot.

Thanks to my family for all the support, both financial and otherwise. And of course, thanks most of all to Gillian for putting up with a poor, stressed-out grad student for two years.

Chapter 1

Introduction

Despite its potential advantages, explicit surface tracking is generally not used for the animation of fluids, as dynamically changing the connectivity and topology of the surface is deemed too complicated. The potential for degenerate cases in geometric collision detection algorithms adds another layer of complexity. For these reasons, implicit surfaces or particle methods are generally favoured for use in fluid animation.

However, explicit surfaces do offer many benefits, the most obvious being control over topology changes of fluid surfaces. Using implicit surfaces, a fluid in contact with itself will automatically merge. With explicit surfaces, however, we can potentially simulate other physical chemistry surface interactions, such as non-adhesion, much more easily.

Rendering is also time-consuming when using implicit surfaces, since a mesh is usually constructed using marching cubes at each time step. Motion blur is then achieved by moving the explicit mesh back in time for one frame using velocity from the simulation. Our method keeps a persistent triangle mesh throughout the course of the simulation, allowing for quick rendering and motion blur.

A further advantage to using explicit surfaces is the possibility of grid-free, boundary-only fluid dynamics. This approach to fluid simulation is different from both the conventional grid-based and particle-based methods currently favored in fluid animation (see chapter 2). Such a system would use only in-

formation on the fluid surface, rather than on a regular grid or on unorganized points throughout the computational domain. With such a discretization, infinite or large domains such as oceans could potentially be simulated efficiently.

To make explicit surface tracking feasible, we take advantage of existing algorithms developed for the animation of cloth. Utilizing robust collision detection, originally developed to keep cloth free of self-intersections [8], our fluid surfaces are kept in a state guaranteed to be free of intersections. We take a conservative approach to topology change and surface adaptivity, modifying the connectivity only when we can guarantee the result will be collision-free. We describe our method for tracking surfaces in chapter 3.

In modern fluid simulation for animation applications, surface tension effects are usually simulated using potentially non-momentum-conserving approximations to mean curvature. In section 4.3, we introduce a comparatively simple, conservative alternative, based on the actual tension statement of the phenomena.

Boundary Element Methods are used to numerically solve partial differential equations over volumes using data only on boundaries of the computational domains. When dealing with fluids, these boundaries corresponds to the fluid surfaces, and so our surface discretization is a natural framework for applying boundary methods. In chapters 4 and 5, we propose a boundary-only, Lagrangian, incompressible fluid simulation method.

Chapter 2

Previous Work

2.1 Eulerian Schemes

There are two main approaches to fluid simulation in animation: Eulerian and Lagrangian. The Eulerian, or grid-based, approach uses discrete function values throughout the computational domain, on a regular volumetric grid. Foster and Metaxas [16] began the current research thrust in Eulerian fluid simulations for animation. Their method has been used for liquid, smoke [14] and sand simulation [29]. Their technique has been steadily improved with advancements in implicit surface tracking [13, 15], unconditionally stable time integration [26], adaptive grid construction [20], and vorticity preservation [25].

For high quality smooth surfaces, Eulerian methods using level sets are typically used. Here the surface is implicitly represented by the zero level set of a surface function, which is usually the signed distance from the surface. This surface function is then advected by the velocity field defined on the grid (often using particles to avoid mass conservation errors in advection [13]). The obvious drawback with this method is that it cannot reliably resolve any detail at or below the scale of a grid cell: thin sheets and other attractive structures cannot be handled efficiently by the approach, no matter how accurate the advection method.

2.2 Lagrangian Schemes

An alternative to grid-based methods are Lagrangian methods. These methods often track moving samples or "particles" of fluid through the domain. Examples include work by Müller et al. [22] and Clavet et al. [9] based on Smoothed Particle Hydrodynamics [21]. Reconstructing a smooth surface from the particles remains challenging, though recent work on point based level sets by Corbett [10] is promising. Vortex particle and vortex filament methods [1, 2] for the simulation of smoke also fall into this category.

2.3 Explicit Surfaces

Explicit surfaces have seen limited use in animation beyond shallow water simulations. Bargteil et al. [4] recently presented a surface tracking method where an explicit mesh is constructed at each time step from a signed distance function via contouring. However, their method still relies on sampling an implicit surface function on a grid, and cannot overcome the fundamental problem of unreliably tracking features at or beyond the grid resolution.

This subject has received much more attention in the field of computational physics. Recently, Jiao [19] presented a new method of advecting explicit surfaces. This paper also provides a pointer to some of the most recent developments in the area of surface tracking. Their method appears to offer improved tracking in the presence of high-curvature surfaces, but it does not handle on-the-fly topological changes. It would be interesting to combine their method for advection with our method for topology changes.

Handling degeneracies in geometric algorithms has also received some attention, whether through determining consistent orientations in degenerate situations [11], or perturbing the geometry slightly using exact arithmetic [23]. Our

method requires neither of these methods, as our surface geometry is guaranteed to never enter into a degenerate state.

2.4 Boundary Element Methods

Using a surface discretization similar to the one presented in this thesis, James and Pai [18] used a boundary element method to model deformable elastic bodies at interactive rates. We note that the linear elasticity model used in that work becomes degenerate in the incompressible limit: how to adapt that approach to incompressible fluids isn't clear.

An introduction to Boundary Element Methods (BEM) can be found in the textbook [6]. Although there is significantly more sophisticated work in BEM (including a long-running annual conference [5]), this book provides adequate background for our work in chapter 5.

Chapter 3

Explicit Surface Tracking

A high-level view of our surface tracking method is shown in algorithm 1. We begin each time step by performing some operations to locally improve mesh quality (see section 3.3). These operations could change the mesh connectivity and topology. We then update our face velocities, either from an underlying grid simulation, or from a boundary-only dynamics simulation. In the next phase, we predict the locations of the mesh vertices based on these face velocities. Next we resolve any collisions, possibly changing mesh topology (see section 3.2). This collision resolution could potentially adjust the predicted vertex locations, and so we update the face velocities to be the difference between the predicted and current vertex locations. Finally, we move the vertices to their (adjusted) predicted locations and proceed to the next time step.

Algorithm 1 Surface tracking overview

```
perform local mesh improvement
get face velocities
predict vertex locations
resolve collisions
update velocities
update vertex locations
```

3.1 Discretization

We discretize each fluid surface as a polygon in 2D, or polyhedron with triangular faces in 3D. We base our method on the robust collision detection and

handling treatment of Bridson et al. [8], which provides an algorithm for guaranteeing that fixed-connectivity meshes will never suffer (self-)intersection.

We extend this to incorporate the connectivity changes required for fluid simulation (such as mesh adaptation and fluid merging or pinching off), but in a conservative fashion to still guarantee that our explicit surface remains in an intersection-free state (or *legal* state) after each advection step. That is, we handle topological changes selectively, merging and separating fluid only when the resulting surfaces are collision-free, and otherwise sequentially applying repulsion forces, geometric collision impulses and rigid impact forces as needed to resolve the surface collisions, as per cloth simulation.

We also adaptively add and remove vertices or flip edges to maintain a good discretization, again doing so only when the resulting configuration is intersection-free, or legal. That is, we *delay* mesh connectivity changes until we know that they are safe. Otherwise, and for fluid surfaces that shouldn't merge or detach (e.g. between immiscible fluids, or a material with surface characteristics that prevent merging such as flour-coated dough, or viscoelastic properties that resist fracture), we treat the contact just as for cloth—in fact, a convenient mental model for this type of interaction is of bags of fluid coming into contact with each other.

The resulting dynamic fluid surface can simply be advected by a velocity field on a grid in order to capture sub-grid-scale detail, similar to passive marker particles, but the explicit formulation naturally provides new possibilities for surface tension simulation. Furthermore, the topological changes can be easily controlled to allow or prevent fluid merging according to the surface physical chemistry.

3.1.1 Data structures

We initially used a half-edge data structure to allow for efficient traversal of the meshes. We later realized, however, that we must allow an edge to be incident to more than two triangles in some situations (see section 3.3.3). We therefore use a simple set of triangles augmented with a set of auxiliary structures. These auxiliaries store information such as which triangles are adjacent to a given triangle, which triangles are incident to a given vertex, which vertices are adjacent to a given vertex, and which two vertices comprise each edge. These auxiliary structures must be rebuilt whenever the topology or connectivity changes, but they then allow for easy traversal of the mesh structures.

3.2 Collisions

The standard cloth collision resolution algorithm proceeds in three stages, which we will now summarize. In the first stage, proximities in the current positions (e.g. a point very close to a triangle) are handled by repulsion forces, aiming at providing a natural separation distance between disjoint mesh elements. While this handles the majority of cases, high speed impacts may be missed, requiring a second stage where we perform temporal collision detection—i.e. detecting collisions even in the middle of the time step—and apply inelastic collision impulses to colliding pairs of mesh elements. Finally, as a last resort, if there are still collisions to be resolved, we project out the motion of colliding regions ("impact zones") to an affine space. In the original paper [8] this was limited to the space of rigid body motions: we generalize this to include shearing as well (see [7] for more details), which means less kinetic energy is artificially dissipated by the projection while we still are guaranteed collisions are impossible within the impact zone. We extend this in a conservative manner to support the

connectivity changes fluid simulation requires.

Our method extends this cloth collision algorithm to handle topological changes between surfaces. Our collision resolution method is outlined in algorithm 2.

Algorithm 2 Collision resolution

```
detect edge-edge proximities
if non-degenerate edge-edge proximity then
   perform zippering operation to change topology
else
   apply edge-edge repulsion force
end if
detect vertex-triangle proximities
apply vertex-triangle repulsion force
for a small number of iterations or until no collisions detected do
   detect edge-edge and vertex-triangle collisions
   if any collision detected then
      apply collision impulse
   end if
end for
if collisions still exist then
   apply impact zones
end if
```

3.2.1 Topological changes

If two volumes of miscible fluid come into close contact, we attempt to merge the two surfaces. Likewise, when two surfaces of the same fluid volume come into close proximity, we wish to produce a hole in the surfaces, increasing the genus of the surface. Both of these operations can be performed with a similar method. We begin with a discussion of the two-dimensional case, then extend into 3D.

In two dimensions, when a vertex is found to be in close proximity to an edge in the current (guaranteed intersection-free) state, we try the following operation. We delete the vertex and its two incidental edges, as well as the edge

it is penetrating. We are then left with a non-manifold geometric structure, *i.e.* open curves. We resolve this by connecting all boundary vertices to their appropriate counterparts on the opposite boundary (see figure 3.1).

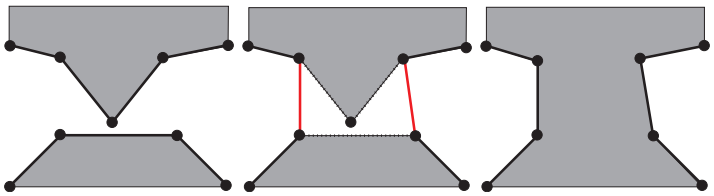

Figure 3.1: Proximity event and response in 2D.

Once these operations are performed, we check the resulting set of new surfaces for intersections. If a new intersection is found, the new configuration is deemed illegal, and we undo the previous delete-reconnect operation, and instead apply a repulsion force to discourage interpenetration (the subsequent cloth collision steps will guarantee this). That is, we delay the topology change to possibly the next time step.

In three dimensions, we have two proximity cases to deal with: a point near a triangle or an edge near another an edge. (For the sake of discussion, we assume in this section that we are dealing with two separate surfaces which are coming into contact, but the algorithms discussed here are also applicable for self-intersection.) In the point-triangle case, we apply repulsion forces and do not attempt to merge surfaces. We have found that it is too difficult to guarantee a collision-free result after a point-triangle merging. In the edge-edge case, we delete the four triangles incident to the edges, then zipper up the gap using eight new triangles. If the new triangles intersect any other mesh elements or each other, we undo this zipper operation.

Figure 3.2 shows the zippering operation. This diagram should not be interpreted as a mesh in 3D space, but rather as a graph showing connectivity. The two interfering edges, shown as dashed lines, are drawn apart for clarity. New

edges are created between the vertices of triangles incident to the interfering edges. These new edges are drawn as arcs in the diagram.

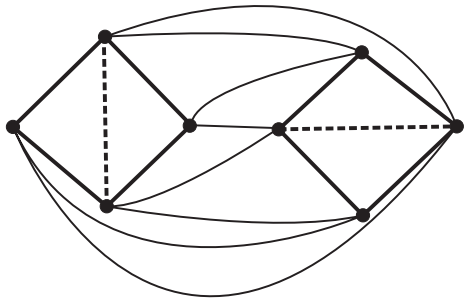

Figure 3.2: Graph view of zippering operation after an edge-edge proximity event

Implicit in the discussion thus far is that the eight vertices on the four incident triangles are distinct. However, we may have edge-edge proximity events where this is not the case (i.e. the edge neighbourhoods share one or more vertices, as in figure 3.3). Rather than trying to foresee all possible configurations of neighbourhood connectivity, we choose to perform zippering only when we know the edge neighbourhoods are distinct, and handle all other situations with the cloth collision resolution. We also avoid the case when the vertices in both neighbourhoods are distinct but are already connected by one or more edges (see figure 3.4), since the result of this operation would be a non-manifold mesh. Therefore, the only possible results of a zippering operation will be an increase in genus (when the edges involved are on the same surface) or a merging (when the edges are on distinct surfaces).

3.3 Local mesh improvement

Since very small or very large edge lengths or face areas can introduce significant numerical error, we adaptively add or remove vertices to keep edge lengths and face areas within a specified range. We also flip edges to maintain a Delaunay

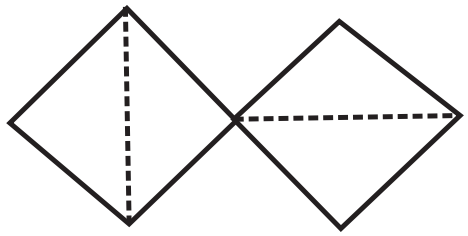

Figure 3.3: Edge-edge proximity with non-distinct neighbourhoods

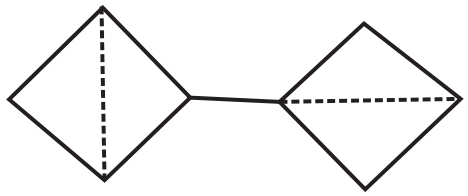

Figure 3.4: Edge-edge proximity with already-connected neighbourhoods

triangulation, and to minimize surface area. These edge flips can result in a change in topology, depending on the connectivity of the edge to be flipped. Our method for mesh improvement is outlined in algorithm 3.

Algorithm 3 Local mesh improvement

```
subdivide long edges
contract short edges
flip edges {may change topology}
handle non-manifold geometry
determine new mesh topology
```

3.3.1 Edge subdivision

Adding a vertex in the middle of a long edge is simple enough: we split the incident triangles in two. Since we are merely tesselating the existing intersection-free geometry more finely, this cannot induce intersections, thus we do not need to check for any.

3.3.2 Edge collapse

To delete vertices, we must take a little more care to ensure that the resulting configuration is legal. When a vertex is scheduled for deletion (because its neighbourhood is too small in area), we check if the pseudo-motion induced in the incident triangles by moving the vertex to its closest neighbour causes any collisions. We stress that this "motion" is not done with the real time step, but rather with the rest of the geometry held fixed and without advancing time. If the pseudo-motion does cause collisions, we do not do anything; otherwise we perform an edge contraction, deleting the vertex. Figure 3.5 illustrates a vertex deletion operation. We check the trajectory shown with an arrow in the middle diagram for collisions before deleting the edge.

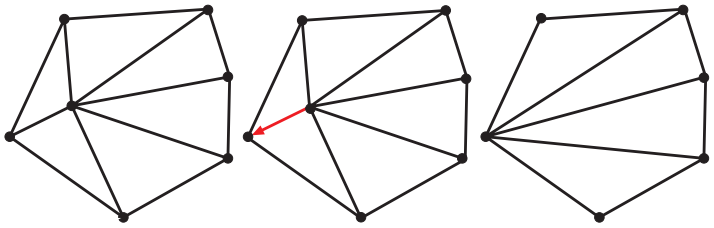

Figure 3.5: Vertex deletion on a 3D mesh

In 2D, an equivalent alternative is to move the doomed vertex to be collinear with its neighbours: if this motion does not cause collisions, we delete the vertex, reconnecting the neighbours: see figure 3.6.

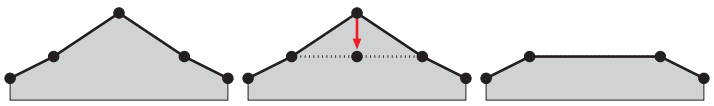

Figure 3.6: Vertex deletion in 2D

3.3.3 Edge flip

We make use of a standard edge flip operation for two purposes: improving triangle aspect ratio and minimizing surface area.

Since we are allowing non-manifold surfaces, we must consider that an edge may be incident to more than two triangles. When dealing with such an edge, we iterate through all pairs of incident triangles which have consistent orientation[1], handling one pair at a time. Each edge flip operation therefore works on an edge and two incident triangles, regardless of how many triangles are actually incident to the edge.

The edge flip operation proceeds as follows: given an edge we wish to flip, we first locate two vertices on the current pair of incident triangles which are not connected by this edge. We construct a "candidate" edge joining these two vertices. Our aim is to delete the given edge and replace it with the candidate edge, but we must first ensure that doing so will not introduce any collisions in the mesh.

To ensure this, we construct the tetrahedron formed by connecting the given and candidate edges, and test to make sure nothing interferes with it. This boils down to ensuring that no vertex lies within the tetrahedron (a simple "point-in-tet" test), and that no edge intersects any face of the tetrahedron ("segment-triangle" test). Once this test is complete, we may delete the given edge and its incident triangles, and add the candidate edge and the new triangles that go with it. This procedure is shown in figure 3.7, where the candidate edge is shown as a dashed line.

[1]For each triangle in our data structure, we store the three vertices in an order which determines the triangle's orientation. To determine if two triangles incident to the same edge have a consistent orientation, we simply check the order of the vertices that make up the shared edge. If the ordering is the same for the two triangles, the triangles do *not* have a consistent orientation. If the vertices are in reverse order on one of the triangles, the triangles have a consistent orientation. Intuitively, we can think of two consistently oriented triangles as being on the same surface passing through the edge, and inconsistent triangles as belonging to different surfaces.

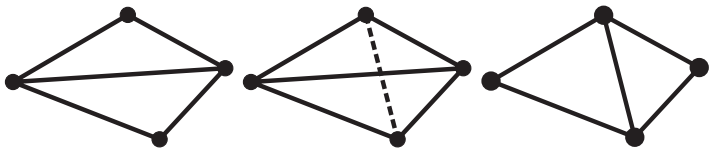

Figure 3.7: Edge flip operation

If the candidate edge already exists, the edge flip operation will result in this candidate edge being incident to more than two triangles. Our data structure can handle this, and indeed this non-manifold situation is necessary to allow for surface separation and genus decrease.

Improving aspect ratio The usual reason for performing an edge flip is to locally improve the aspect ratio of the triangles in a mesh. Flipping an edge when its counterpart candidate edge is shorter produces better aspect ratios, which is desirable for simulation reasons. Poor aspect ratios can produce low-area triangles, which in turn can introduce significant numerical error in the simulation. For this reason, we consider the difference in length between an edge and its candidate edge in our decision of whether or not to flip an edge.

Reducing surface area The other objective that edge flips can help us achieve is the reduction of surface area. We would like to perform an edge flip when it would decrease the surface area of the mesh. This will help achieve a larger goal: simulation of surface tension, which has the effect of minimizing surface area for a given volume. Therefore, we also take into account the potential reduction in surface area when deciding whether to flip an edge.

3.4 Handling non-manifold meshes

Since we do not use a half-edge data structure, we can represent surfaces which are not strictly manifold. In particular, we allow more than two triangles to be

incident to an edge. We do not, however, allow two triangles to share the same three vertices, thus creating a zero volume tetrahedron. After an edge collapse or edge flip, we search the surface meshes for such a situation, and delete the offending triangles. We also delete triangles which may have repeated vertices ("collapsed" triangles).

After this sweep, we deal with surfaces which may be connected at a single vertex. These so-called "standalone" vertices can be detected if their incident triangles are not all connected. If this is the case, we partition the set of incident triangles into connected components. For each component, we create a duplicate vertex and map all triangles in the component to this new vertex. A similar procedure is described in [17]. We also move the duplicate vertices very slightly towards the centroid of their associated triangles to avoid problems with collision detection and resolution.

3.5 Axis-aligned bounding boxes

To increase the efficiency of collision detection, we use a hierarchy of axis-aligned bounding boxes, as in [8]. We begin construction of the hierarchy by first creating bounding boxes for each triangle. These form the leaf nodes of a binary tree hierarchy. We then sweep in alternating directions, pairing up adjacent bounding boxes. For each pair, we create a parent node, taking the union of the children's extents to create the parent's bounding box. We continue until we are left with one root node whose bounding box covers all triangles in the mesh.

When we wish to check a triangle for collisions, we check the triangle's bounding box against the root's bounding box, recursively checking versus the child nodes when the triangle's box overlaps their bounding boxes. We proceed down the binary tree until we end up with a set of leaf nodes — these are the

triangles which could possibly interfere with the triangle being checked. We can then perform edge-edge, point-triangle, or segment triangle collision detection as required.

Chapter 4

Boundary-Only Fluid Dynamics

The explicit surface described so far could be passively advected on an Eulerian grid, and the surface tension force could even be fed back into the underlying simulation. However, one major advantage of explicit surfaces lies with the possibility of a grid-free, purely Lagrangian fluid simulation, using only data points on the fluid surface. In this section, we present our contribution to a *boundary-only* fluid simulation, targeting small-scale phenomena. The benefit of a boundary-only formulation is immediately apparent: we reduce the dimensionality of the problem by one, from the discretization of a full 3D grid to that of a 2D manifold. We can view this as a generalization of the shallow water equations to arbitrary geometry. In particular, we will target flow dominated by the free surface, e.g. thin sheets, small drops, etc. that are especially challenging for volumetric methods.

This chapter will provide a broad overview of the discretization and our approach. The next chapter will cover some of the mathematical details in depth.

4.1 Velocity Discretization

Rather than sampling the velocity of the fluid surface on the vertices, we instead choose to sample on the faces (the edges in 2D, the triangles in 3D). These elements have well-defined normals, which permits us to easily measure the rate of volume change of the fluid. If the fluid occupies a volume Ω with boundary $\partial\Omega$ and moves with velocity $\vec{u}$ then

$$\frac{d}{dt}\text{volume}(\Omega) = \int_{\partial\Omega} \vec{u} \cdot \hat{n} \tag{4.1}$$

This is reminiscent of the staggered MAC grid used in most grid-based incompressible flow solvers, where velocity unknowns are arranged to easily estimate this volume change integral. Note, however, that we sample the full vector velocity on the mesh faces, not just the normal component.

When we need to actually move the mesh we move each vertex with the average of its incident faces' velocities. This motion is corrected by the collision algorithm from the previous section, giving possibly modified end-of-time-step positions for the vertices. We then take the difference between the end-of-time-step and initial positions to find the average velocity over the time step for each vertex. Finally, we set the new velocity of a mesh face to be the average of its incident vertices. For an alternate method of surface advection, see [19].

4.2 Viscosity

Our model of the internal velocity field is such that it should contain no details that are not apparent on the boundary (e.g. no vortices). The most convenient physical approach is to specify that each component of the velocity is harmonically interpolated from the boundary values. That is, the velocity satisfies $\nabla^2\vec{u} = 0$ in the interior. This corresponds to Stokes flow, where we take

the limit as viscosity dominates inertia in the interior of the flow, bringing it into quasi-static equilibrium with the viscous stress. This model is plausible for small-scale phenomena such as water drops. Since we never directly refer to the velocity in the interior, however, we do not actually need to calculate this interior velocity field.

The velocities on the fluid surface, however, should be affected by viscosity. One method for simulating viscosity on the boundary would be to decompose the velocity into rigid and non-rigid components, then applying a damping to the non-rigid components. We have not implemented this method, and thus cannot speak for its effectiveness, nor have we shown that it converges to the usual method for viscosity simulation.

4.3 Surface Tension

Conventional fluid simulation techniques, based on volumetric grids, have severe challenges in modeling surface tension effects, particularly with thin fluid structures. For example, the recent method of Wang et al. [28] suffers from huge computational and memory costs. Ad hoc procedural methods based on blobbies [9] are also inadequate for dealing with thin structures such as sheets. With an explicit surface, we are in a much better position to accurately treat surface tension.

Rather than the conventional approach based on mean curvature driven flow [20, 28], we model surface tension as an *actual tension* per unit length, permitting a more accurate conservative discretization. In two dimensions, we add two forces to each edge, proportional to a surface tension coefficient, parallel to the directions of the two neighboring edges (see figure 4.1). In three dimensions, we add three forces to a given face, corresponding to all neighboring faces. The force is proportional to the surface tension coefficient times the edge length,

in the direction normal to the edge and coplanar to the neighbouring face (see figure 4.2). We note this exactly conserves the momentum of the volume of fluid, unlike other approaches to surface tension, since these forces are always balanced by the opposite forces on neighbouring faces.

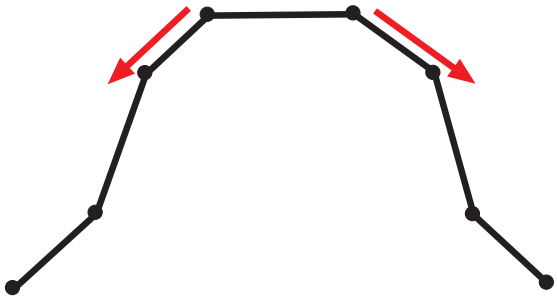

Figure 4.1: 2D surface tension forces acting on the top edge

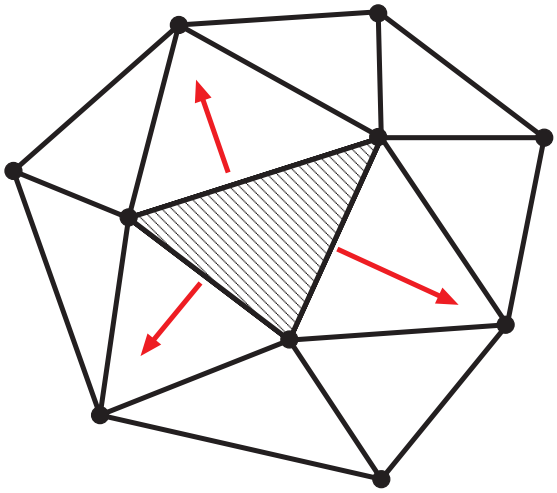

Figure 4.2: 3D surface tension forces acting on the shaded triangle

4.4 Volume conservation

Equation 4.1 gives us a clear constraint on the surface velocities for incompressible flow: the rate of volume change should be zero. We could modify the normal

component of velocities on the faces to ensure that $\int_{\partial\Omega} \vec{u} \cdot \hat{n} = 0$ at each time step. However, to avoid accumulated truncation error in the advection step, we use a more robust technique to modify velocities to preserve volume.

At the start of the simulation, we calculate and store the exact volume of each disjoint region of fluid. To compute the volume of fluid, first notice that the integral of 1 over a closed domain is equal to the total volume of that domain:

$$\text{volume}(\Omega) = \int_{\Omega} 1 \; dx$$

We transform this volume integral into a surface integral using the Gauss divergence theorem, which can be written as:

$$\int_{\Omega} \nabla \cdot \vec{F} \; dV = \int_{\partial\Omega} \vec{F} \cdot \hat{n} \; dS$$

To leverage this theorem, we specify a function whose divergence equals 1, for example:

$$\begin{aligned} \vec{F}(x) &= \frac{\vec{x}}{3} \\ \nabla \cdot \vec{F}(x) &= 1 \end{aligned}$$

We can now apply the divergence theorem to get a formula for volume in terms of a surface integral:

$$\text{volume}(\Omega) = \int_{\Omega} 1 \; dV = \int_{\Omega} \nabla \cdot \frac{\vec{x}}{3} \; dV = \int_{\partial\Omega} \frac{\vec{x}}{3} \cdot \hat{n} \; dS$$

Thus we have a method for computing volume using only surface positions and normals.

At every time step, after adding all other forces such as gravity and surface tension, we move the vertices to candidate new positions and calculate the

volume of each region based on those candidate positions. We measure how different this is from the stored true volume for the region, and add a corrective constant to the normal component of velocity on each face:

$$\Delta u_N = \frac{1}{2}\left(\frac{[\text{true volume}] - [\text{predicted volume}]}{[\text{current surface area}]}\right) \tag{4.2}$$

$$\vec{u}_i^{\text{new}} = \vec{u}_i + \Delta u_N \hat{n}_i \qquad \text{for all } i \tag{4.3}$$

The factor of $1/2$ is there to prevent us overshooting the correct volume in the subsequent time step. This method for volume conservation is illustrated in figure 4.3.

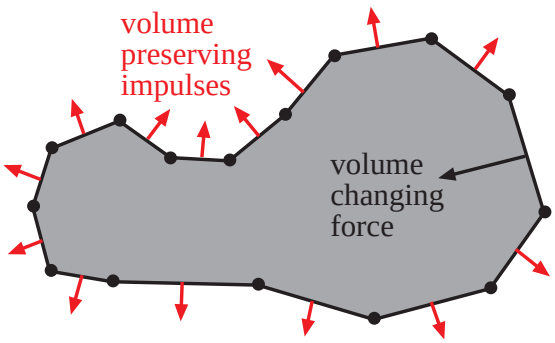

Figure 4.3: Non-physical volume conservation in 2D

In any topological operation, we adjust the true volume stored for the fluid accordingly. If two disjoint regions of fluid merge, we add their true volumes together. If one region detaches into two separate regions, we split the old true volume up between them according to the ratio of their current volumes.

Used by itself, this simple, non-physical heuristic can be surprisingly effective in providing the effect of incompressibility. However, it really is just a post-process and not a true pressure solve—for example, it cannot handle the hydrostatic case of fluid resting in a container under gravity. Thus while we will keep this as a final correction to velocities, we require a more sophisticated

method to constrain the velocities to be incompressible.

4.5 Mesh element masses

This section will introduce a more rigorous approach to volume conservation. More mathematical details will be given in the next chapter. We begin by defining a harmonic partition of unity in the fluid volume. Let the fluid volume be Ω with boundary $\partial\Omega$. For each mesh element i (edge in 2D, triangle in 3D), we define a function $\phi_i(x)$ that is equal to 1 on element i and zero on the rest of $\partial\Omega$, and is harmonic in the interior of the volume: $\nabla^2\phi_i(x) = 0$ for $x \in \Omega$. Note that these are guaranteed to be non-negative functions, and they must add up to 1 inside Ω, since their sum is harmonic and equal to 1 on all of $\partial\Omega$. Thus they are indeed a partition of unity.

We define the mass, m_i, of mesh element i to be the integral of its basis function times the density ρ which we will assume is constant and rescale to be equal to 1.

$$m_i = \int_\Omega \phi_i(x)\, dx \tag{4.4}$$

Note that the sum of the masses is the integral of $\sum \phi_i(x) \equiv 1$ over the fluid, i.e. the total mass of the fluid.

The actual calculation of m_i can be done using the Boundary Element Method, converting the volume integral to boundary integrals which can then be easily performed on the mesh. At no time do we need to calculate anything inside the volume of the fluid. See the next chapter for details on this calculation.

Similar to our basis functions, "Harmonic Coordinates" [27] have recently been used as a generalization to barycentric coordinates. However, use of these coordinates requires the solution of Laplace's equation over the entire domain

using a volumetric grid, whereas our method requires only a boundary discretization of the variables.

4.6 Pressure

One interpretation of pressure in an incompressible fluid is that it is simply the Lagrange multiplier which enforces the divergence-free constraint on the velocity field. Thus if we can appropriately discretize this constraint, and define an appropriate mass to accompany each velocity unknown (as we have done), we can simply use Lagrange multiplier dynamics (e.g. [3]) to get the full effect of pressure. We will thus focus on capturing the incompressibility constraint now.

We calculated the mass m_i of each mesh face above. Intuitively, if we think of that as defining a volume of fluid associated with each face and suppose that the velocity field in the interior is as smooth as possible (i.e. there exist no arbitrary vortices or other features that don't show up on the boundary), it makes sense that we would want this volume to stay constant. We therefore propose the constraint $dm_i/dt = 0$.

We conjecture that dm_i/dt is in fact just a linear combination of the mesh velocities, so we have a simple linear constraint to enforce. Note that summing up these constraints gives us that the total mass of the fluid is conserved, as expected. (However, again due to truncation errors in advection we still apply a post-correction to the velocities described in section 4.4 to closely track the true mass.) In section 5.3 we provide a more mathematical justification for this constraint, relating it to the classical statement $\nabla \cdot \vec{u} = 0$, but admit we are still stymied in making this fully rigorous. Also, in section 5.4 we discuss calculating the coefficients of the face velocities in the expression dm_i/dt, but here again we have not yet worked out all the details.

We further note that if $dm_i/dt = 0$, advecting the faces conserves the linear momentum $m_i\vec{u}_i$ (up to truncation error). As a result, simulations without this constraint sometimes display odd artifacts of momentum loss or gain.

Chapter 5

Incompressibility via the Boundary Element Method

5.1 BEM in two dimensions

We begin with a description of the standard Boundary Element Method in two dimensions (see [6] section 2.6 for an alternate introduction).

Let Ω be a polygonal domain with boundary $\partial\Omega$. Let ϕ be a harmonic basis function, i.e. $\nabla^2\phi(x) = 0$ for $x \in \Omega$. We will assume that ϕ is constant over each boundary segment (polygon edge).

When $x \in \Omega \setminus \partial\Omega$,

$$\phi(x) = \int_\Omega \phi(y)\delta(x-y)dy = \int_\Omega -\phi(y)\nabla^2 G(x-y)dy$$

for Green's function, $G(p) = -\frac{\ln\|p\|}{2\pi}$. Using integration by parts twice yields:

$$\begin{aligned}\phi(x) \quad = \quad & \int_{\partial\Omega} \nabla\phi(y)\cdot\hat{n}(y)G(x-y)dy - \int_{\partial\Omega}\phi(y)\nabla G(x-y)\cdot\hat{n}(y)dy - \\ & \int_\Omega \nabla^2\phi(y)G(y-x)dy\end{aligned}$$

and since $\nabla^2\phi(y) = 0$ for $y \in \Omega$,

$$\phi(x) = \int_{\partial\Omega} \nabla\phi(y)\cdot\hat{n}(y)G(x-y)dy - \int_{\partial\Omega}\phi(y)\nabla G(x-y)\cdot\hat{n}(y)dy \qquad (5.1)$$

When $x \in \partial\Omega$, things are not so simple, however, it can be shown that

$$c(x)\phi(x) = \int_{\partial\Omega} \nabla\phi(y) \cdot \hat{n}(y) G(x-y) dy - \int_{\partial\Omega} \phi(y) \nabla G(x-y) \cdot \hat{n}(y) dy$$

where $c(x)$ is the fraction of the neighbourhood of x that is inside Ω. However, as we shall soon see, it will not be necessary to compute this fraction. Introduce a set of basis functions $\{\psi\}$ such that $\phi(x) = \sum_j \phi_j \psi_j(x)$ where ϕ_j are constant values of ϕ on the boundary segments $\partial\Omega_j$. The previous equation becomes:

$$\begin{aligned} c(x) \sum_j \phi_j \psi_j(x) = \\ \sum_j \left[\left(\int_{\partial\Omega} G(x-y) \psi_j(y) dy \right) \nabla\phi_j \cdot \hat{n} \right] - \\ \sum_j \left[\left(\int_{\partial\Omega} \nabla G(x-y) \cdot \hat{n}(y) \psi_j(y) dy \right) \phi_j \right] \end{aligned}$$

Evaluating at $x = x_i$:

$$\begin{aligned} c_i \sum_j \phi_j \psi_j(x_i) = \\ \sum_j \left[\left(\int_{\partial\Omega} G(y-x_i) \psi_j(y) dy \right) \nabla\phi_j \cdot \hat{n} \right] - \\ \sum_j \left[\left(\int_{\partial\Omega} \nabla G(y-x_i) \cdot \hat{n}(y) \psi_j(y) dy \right) \phi_j \right] \end{aligned}$$

If we let $\psi_j = 1$ on edge $\partial\Omega_j$ and zero on other edges, we can rewrite this equation in matrix notation:

$$\mathbf{Cf} = \mathbf{Gf'} - \hat{\mathbf{H}}\mathbf{f}$$

where

$$\begin{aligned}
\mathbf{f} &= \text{a vector of values of } \phi \text{ at edge midpoints, e.g. } \phi_j \\
\mathbf{f}' &= \text{a vector of values of } \nabla\phi \cdot \hat{n} \text{ at edge midpoints, e.g. } \nabla\phi_j \cdot \hat{n}_j \\
\mathbf{C} &= \text{diag}(c_i) \\
\mathbf{G}_{ij} &= \int_{\partial\Omega_j} G(y - x_i)dy \\
\hat{\mathbf{H}}_{ij} &= \int_{\partial\Omega_j} \nabla G(y - x_i) \cdot \hat{n}(y)dy
\end{aligned}$$

Letting $\mathbf{H} = \hat{\mathbf{H}} + \mathbf{C}$, we get the system

$$\mathbf{Hf} = \mathbf{Gf}'$$

The integrals $\mathbf{G}_{ij}$ and $\hat{\mathbf{H}}_{ij}$ could be approximated with a quadrature rule, however we found that this introduces unacceptable numerical error. Therefore, we compute all of the integrals above analytically, and use theses values for $\mathbf{G}_{ij}$ and $\hat{\mathbf{H}}_{ij}$ except for $\hat{\mathbf{H}}_{ii}$. To compute these diagonal elements, we note that if $\mathbf{f}$ is constant, then $\mathbf{f}' = \mathbf{0}$. In this case, $\mathbf{Hf} = \mathbf{0}$, so $\mathbf{H}$ must have zero row sums. Thus $\mathbf{H}_{ii} = -\sum_{j \neq i} \hat{\mathbf{H}}_{ij}$, eliminating the need for computing both $\mathbf{C}_{ii}$ and $\hat{\mathbf{H}}_{ii}$.

Once these matrices are populated we can rearrange this linear system to solve for any unknown constant values of ϕ and $\nabla\phi \cdot \hat{n}$ on the edges, resulting in a dense linear system:

$$\mathbf{Av} = \mathbf{z}$$

This linear system is NxN in the number of boundary segments. In the sequel, we will be given values for $\mathbf{f}$ (usually a vector with one non-zero entry), and we will solve for values of $\mathbf{f}'$ by inverting $\mathbf{G}$:

$$\mathbf{f'} = \mathbf{G^{-1}Hf}$$

5.2 Mass Calculation

In this section we will demonstrate how to compute a measure of mass for one face of a polytope. We do this by defining a harmonic basis function corresponding to the face, and calculating the integral of this function over the entire computational domain. As we shall see, this integration can be done using only boundary integrals, without having to compute anything inside the domain. In our discussion, the computational domain refers to the volume of the fluid, and the boundary refers to the fluid surface.

Let Ω be the fluid domain with boundary mesh $\partial\Omega$. We define a set of harmonic basis functions $\{\phi\}$ as follows. Each face i has a corresponding ϕ_i which has the value 1 on face i ($\partial\Omega_i$), and the value 0 on all other edges. In other words,

$$\begin{aligned} \nabla^2\phi_i(x) &= 0 \text{ for } x \in \Omega \\ \phi_i(x) &= \begin{cases} 1 & \text{for } x \in \partial\Omega_i \\ 0 & \text{for } x \in \partial\Omega \setminus \partial\Omega_i \end{cases} \end{aligned}$$

Given values for ϕ_i on the boundary in this way, the Boundary Element Method can be used to obtain values for $\nabla\phi_i \cdot \hat{n}$ on the boundary. This gives the following integral expression for ϕ_i in the interior of the fluid:

$$\phi_i(x) = \int_{\partial\Omega} \nabla\phi_i(y) \cdot \hat{n}(y) G(x-y) dy - \int_{\partial\Omega} \phi_i(y) \nabla G(x-y) \cdot \hat{n}(y) dy$$

where G is the fundamental solution of the Laplacian, $G(x) = -\log\|x\|/(2\pi)$ in 2D and $G(x) = 1/(4\pi\|x\|)$ in 3D. We then integrate this over the fluid region

to define our mass measure for the face:

$$m_i = \int_\Omega \phi_i(x)dx \tag{5.2}$$

Subsituting in the boundary integral expression gives:

$$\begin{aligned}
&\int_\Omega \phi_i(x)dx \\
&= \int_\Omega \int_{\partial\Omega} \nabla\phi_i(y)\cdot\hat{n}(y)G(x-y)dydx - \int_\Omega \int_{\partial\Omega} \phi_i(y)\nabla_y G(x-y)\cdot\hat{n}(y)dydx \\
&= \int_{\partial\Omega} \nabla\phi_i(y)\cdot\hat{n}(y)\int_\Omega G(x-y)dxdy - \int_{\partial\Omega}\phi_i(y)\int_\Omega \nabla_y G(x-y)dx\cdot\hat{n}(y)dy
\end{aligned}$$

We can turn these volume integrals into surface integrals using the gradient and divergence theorems:

$$\int_\Omega \nabla_y G(x-y)dx = \int_\Omega -\nabla_x G(x-y)dx = \int_{\partial\Omega} -G(x-y)\hat{n}(x)dx$$

and:

$$\int_\Omega G(x-y)dx = \int_\Omega \nabla_x\cdot F(x-y)dx = \int_{\partial\Omega} F(x-y)\cdot\hat{n}(x)dx$$

where F is an antiderivative of G, the fundamental solution: $F(r) = -\frac{1}{4\pi}\left(r\ln\|r\| - \frac{r}{2}\right)$ in 2D and $F(r) = r/(8\pi\|r\|)$ in 3D. Now we have m_i in terms of (nested) boundary integrals:

$$\begin{aligned}
m_i = &\\
&\int_{\partial\Omega} \nabla\phi_i(y)\cdot\hat{n}(y)\left(\int_{\partial\Omega} F(x-y)\cdot\hat{n}(x)dx\right)dy + \\
&\int_{\partial\Omega}\phi_i(y)\left(\int_{\partial\Omega} G(x-y)\hat{n}(x)dx\right)\cdot\hat{n}(y)dy
\end{aligned}$$

We have closed forms for G and F, ϕ_i is specified on the boundary, and we can obtain values for $\nabla\phi_i \cdot \hat{n}$ on the boundary via the BEM. Using the fact that the ϕ_i functions are constant along faces, we can apply a quadrature rule by evaluating at midpoints x_j:

$$m_i = \sum_j \left[\nabla\phi_i(x_j) \cdot \hat{n}_j A_j \left(\int_{\partial\Omega} F(x - x_j) \cdot \hat{n}(x) dx \right) + \phi_i(x_j) \left(\int_{\partial\Omega} G(x - x_j) \hat{n}(x) dx \right) \cdot \hat{n}_j A_j \right]$$

where n_j are face normals, A_j are face areas (lengths in 2D), and x_j are face midpoints. We compute the integrals of $F(x - y) \cdot \hat{n}(x)$ and $G(x - y)\hat{n}(x)$ analytically, similar to the elements of our BEM matrices.

We now have enough information to compute the volume integral of a face's basis function, giving us the face's mass. Figure 5.1 shows the computed masses for a 2D square which has been subdivided into 20 edges.

5.3 Enforcing Incompressibility

As discussed in section 4.6, we ensure incompressibility of the fluid by enforcing a linear constraint on the velocity of the fluid at the surface. In this section, we will discuss the formulation and properties of such a linear constraint, in particular how it relates to the classical statement of incompressibility, $\nabla \cdot \vec{u} = 0$. Note that we have not yet implemented this method and thus cannot provide support for the conjectures made here. Hypothesized implementation details are covered in the following section.

The continuum statement of our proposed $dm_i/dt = 0$ constraint is as fol-

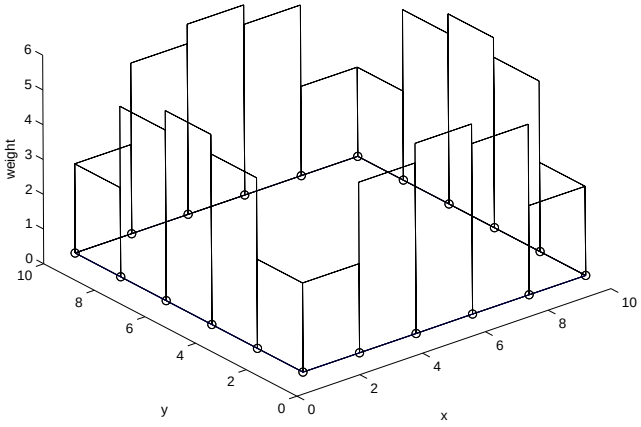

Figure 5.1: Computed element masses for square geometry

lows. Let $\phi(x)$ be harmonically interpolated in Ω from its boundary values, and suppose that those boundary values are advected along with the boundary: $D\phi/Dt = 0$ on $\partial\Omega$. Then we require that the integral of $\phi(x)$ over the volume remains constant:

$$\frac{d}{dt}\int_{\Omega}\phi(x)\,dx = 0 \tag{5.3}$$

Using the Green's function $H(x,y)$ for the problem we can express $\phi(x)$ as a boundary integral:

$$\phi(x) = \int_{\partial\Omega} H(x,y)\phi(y)\,dy \tag{5.4}$$

Here $\nabla_x^2 H(x,y) = 0$ in Ω and $H(x,y) = 2\delta(x-y)$ for x, y on the $\partial\Omega$. Then the constraint is:

$$\begin{aligned} 0 &= \frac{d}{dt}\int_{\Omega}\int_{\partial\Omega} H(x,y)\phi(y)\,dy\,dx \\ &= \int_{\partial\Omega}\left(\int_{\partial\Omega} H(x,y)\phi(y)\,dy\right)\vec{u}(x)\cdot\hat{n}(x)\,dx + \int_{\Omega}\int_{\partial\Omega} H(x,y)\frac{\partial}{\partial t}\phi(y)\,dy\,dx \end{aligned}$$

Using the δ property of H for $x \in \partial\Omega$ in the first integral, and the material derivative $\partial\phi/\partial t + \vec{u} \cdot \nabla\phi = 0$ on the boundary in the second, we get:

$$0 = \int_{\partial\Omega} \phi(x)\vec{u}(x) \cdot \hat{n}(x)\, dx - \int_{\Omega}\int_{\partial\Omega} H(x,y)\vec{u}(y) \cdot \nabla\phi(y)\, dy\, dx$$

Notice the second term is in fact the integral of the harmonic interpolant of the boundary values of $\vec{u}\cdot\nabla\phi$. We assume—but do not have a rigorous justification—that under the assumption of $\vec{u}$ being harmonic itself in the interior, this integral is actually exactly equal to the integral of $\vec{u} \cdot \nabla\phi$. In any case, it is a plausible estimate. We then would have:

$$\begin{aligned} 0 &= \int_{\partial\Omega} \phi(x)\vec{u}(x) \cdot \hat{n}(x)\, dx - \int_{\Omega} \vec{u}(x) \cdot \nabla\phi(x)\, dx \\ &= \int_{\Omega} \phi(x)\nabla \cdot \vec{u}(x)\, dx \end{aligned}$$

where we used integration by parts in the last step. If this is zero for arbitrary ϕ, then $\nabla \cdot \vec{u} = 0$: the fluid is incompressible.

5.4 Discrete Incompressibility

We note again that details presented this section have not yet been implemented, and thus cannot be verified. We have yet to formulate a consistent finite difference method over our piecewise constant function approximations.

We now need to calculate dm_i/dt in terms of the face velocities. From the previous section, taking $\phi = \phi_i$, this is:

$$\begin{aligned} \frac{dm_i}{dt} &= \frac{d}{dt}\int_{\Omega} \phi_i(x)\, dx \\ &= \int_{\partial\Omega} \phi_i(x)\vec{u}(x) \cdot \hat{n}(x)\, dx - \int_{\partial\Omega}\int_{\Omega} H(x,y)\, dx\vec{u}(y) \cdot \nabla\phi_i(y)\, dy \end{aligned}$$

where we switched the order of integration in the second integral. The Green's function $H(x, y)$ integrated over $x \in \Omega$ and y over a mesh face j is nothing other than the mass m_j. Also using the fact ϕ_i is 1 or 0 over the boundary faces, we can approximate the expression as:

$$\frac{dm_i}{dt} = \vec{u}_i \cdot \hat{n}_i - \sum_j m_j \vec{u}_j \cdot \nabla \phi_i(x_j) \tag{5.5}$$

The Boundary Element Method we mentioned earlier lets us determine the normal component of $\nabla \phi_i$, and the tangential component can be estimated by finite differences along the mesh. Exact details for using a finite difference method over a piecewise linear boundary with constant function values have yet to be worked out.

We can express $dm_i/dt = 0$ for all i as a matrix equation $Du = 0$, where the coefficients of D come from the previous equation and u is the vector of all face velocities. To constrain this to zero, we introduce Lagrange multipliers (pressures) and with the diagonal mass matrix M, project as follows:

$$\begin{aligned} DM^{-1}D^T p &= Du \\ u^{\text{new}} &= u - D^T M^{-1} p \end{aligned}$$

which looks exactly like a standard pressure projection step.

Chapter 6

Results

Figures in this section are screen captures from a 3D OpenGL application developed to test our method, except for the "Enright Test" (see below), which was rendered using pbrt [24].

Our surface tracking method can reliably handle basic topological changes, such as merging, separation and genus increase. Theses examples are intended only to highlight our surface tracking algorithm, and thus there is no underlying physics simulation beyond surface tension and the heuristic volume conservation method outlined in section 4.4. Figure 6.1 shows two surfaces merging into one.

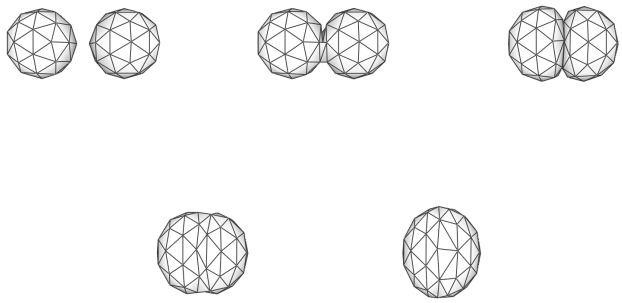

Figure 6.1: Surface merging

Figure 6.2 shows a surface mesh separating.

Figure 6.3 demonstrates genus change: a sphere turning into a torus.

Finally, we subjected our method to the "Enright Test" [12] by advecting

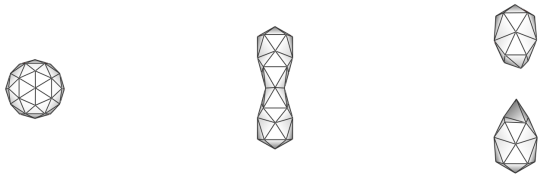

Figure 6.2: Surface separation

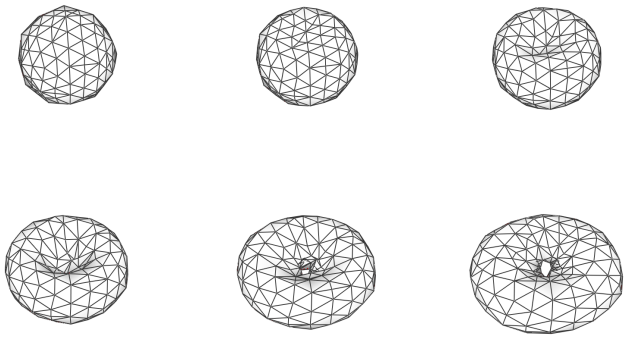

Figure 6.3: Genus change

a surface passively through a velocity field. In this test, vertex velocities were taken directly from a closed-form function, not from triangle velocities as discussed earlier. We also did not include effects due to surface tension, nor did we add the heuristic volume conservation method. We advected the surface using a fourth-order explicit time integration scheme. The test began with around 200 vertices on the surface, and had about 1500 vertices at its most extended (compared to a 100^3 grid with 40 particles per grid cell in the original test). Figure 6.4 shows our results at frames 0, 25, 35, 50, 60, 75, 95, 100, 105, 115, 125 and 150. The volume enclosed by the surface at frame 150 is 98.8% of the volume at frame 0.

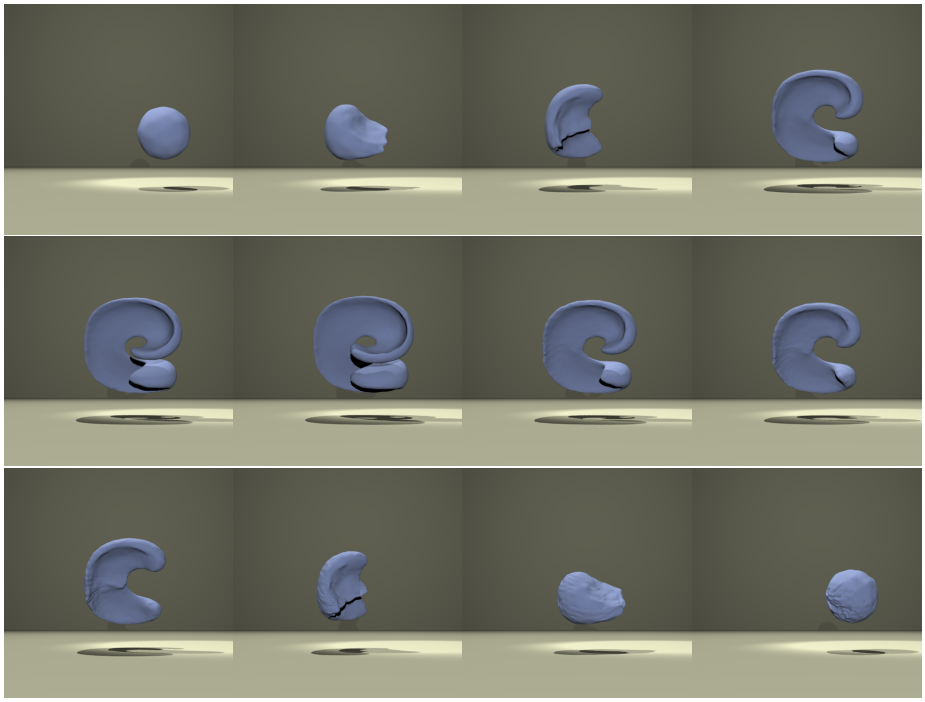

Figure 6.4: The "Enright Test"

Chapter 7

Conclusions

Contributions to explicit surface tracking and incompressible fluid flow via the BEM were presented. A fail-safe scheme for surface separation via cloth collision algorithms allows us to neatly sidestep severely degenerate cases common in most explicit surface tracking methods. This in turn, makes the topological changes necessary in fluid simulation tractable.

Local mesh improvement techniques are employed to maintain an accurate discretization of the fluid, including edge subdivision, edge collapse and edge flipping. We perform these improvement operations only if it does not result in a mesh intersection.

Two techniques for volume conservation were discussed: a simple heuristic adjustment of normal velocities, and a more rigorous scheme using Lagrange multiplier dynamics. Using a Boundary Element Method, we defined a per-face mass function, and conjectured that constraining each of these mass functions to be constant over time would conserve volume. This has not been fully implemented, nor has this constraint formulation been related back to the standard statement of incompressibility. It is not yet clear how the assumptions made on the nature of the fluid flow will restrict the usability of this method for generic cases. We also introduced a method for simulating surface tension, taking advantage of the explicit surface discretization.

7.1 Future work

An obvious advantage of our method is that it provides a framework for adding incremental improvements, while still being able to rely on guaranteed surface separation. This allows us to pick and choose further cases to handle. For example, an immediate next step would be to handle merging from edge-edge collisions where the edge neighbourhoods are not distinct. This could result in some degenerate situations, but we would be able to choose which situations can be handled and which are "too degenerate" and should instead be separated by the cloth collision code.

Some further future steps include:

- Plug the surface tracking method into a grid-based fluid simulation.
- Prove that $\frac{dm}{dt} = 0$ is equivalent to $\nabla \cdot u = 0$.
- Implement the discrete version of incompressibility as in section 5.4.
- Implement the face moving algorithm of [19] for the advection phase.
- Add effects of viscosity on the surface, as described in section 4.2.

Bibliography

[1] Alexis Angelidis and Fabrice Neyret. Simulation of smoke based on vortex filament primitives. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 87–96, New York, NY, USA, 2005. ACM Press.

[2] Alexis Angelidis, Fabrice Neyret, Karan Singh, and Derek Nowrouzezahrai. A controllable, fast and stable basis for vortex based smoke simulation. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*, sep 2006.

[3] David Baraff. Linear-time dynamics using Lagrange multipliers. In *Proc. SIGGRAPH*, volume 30, pages 137–146, 1996.

[4] Adam W. Bargteil, Tolga G. Goktekin, James F. O'Brien, and John A. Strain. A semi-lagrangian contouring method for fluid simulation. *ACM Trans. Graph.*, 25(1), 2006.

[5] Carlos A. Brebbia, editor. *Boundary Elements and Other Mesh Reduction Methods XXVIII, Proceedings of the 28th World Conference on Boundary Elements*, volume 42 of *WIT Transactions on Modelling and Simulation*. WIT Press, 2006.

[6] Carlos A. Brebbia, José C. F. Telles, and Luiz C. Wrobel. *Boundary Element Techniques: Theory and Applications in Engineering*. Springer-Verlag, 1984.

[7] Robert Bridson. *Computational aspects of dynamic surfaces*. PhD thesis, Stanford University, 2003.

[8] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 21(3):594–603, 2002.

[9] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based viscoelastic fluid simulation. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 219–228, 2005.

[10] Richard Corbett. Point-based level sets and progress towards unorganized particle-based fluids. Master's thesis, University of British Columbia, 2005.

[11] Herbert Edelsbrunner and Ernst P. Mucke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. In *Symposium on Computational Geometry*, pages 118–133, 1988.

[12] Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys.*, 183(1):83–116, 2002.

[13] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph. (Proc. SIGGRAPH)*, pages 736–744, 2002.

[14] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proc. SIGGRAPH*, pages 15–22, 2001.

[15] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proc. SIGGRAPH*, pages 23–30, 2001.

[16] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graph. Models Image Process.*, 58(5):471–483, 1996.

[17] André Guéziec, Gabriel Taubin, Francis Lazarus, and Bill Horn. Cutting and stitching: Converting sets of polygons to manifold surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):136–151, 2001.

[18] Doug L. James and Dinesh K. Pai. Artdefo: accurate real time deformable objects. In *SIGGRAPH '99*, pages 65–72, 1999.

[19] Xiangmin Jiao. Face offsetting: a unified framework for explicit moving interfaces. *Journal of Computational Physics*, 2006. under revision.

[20] Frank Losasso, Frederic Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans. Graph.*, 23(3):457–462, 2004.

[21] Joe J. Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30, 1992.

[22] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proc. ACM/Eurographics Symp. Comp. Anim.*, pages 154–159, 2003.

[23] Kuji Ouchi and John Keyser. Exact numerical perturbation. Technical Report 2005-1-2, Texas A & M University, 2005.

[24] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[25] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph.*, 24(3):910–914, 2005.

[26] Jos Stam. Stable fluids. In *Proc. SIGGRAPH*, pages 121–128, 1999.

[27] Mark Meyer Tony DeRose. Harmonic coordinates. Technical Memo 06-02, Pixar Animation Studios, 2006.

[28] Huamin Wang, Peter J. Mucha, and Greg Turk. Water drops on surfaces. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 24(3):921–929, 2005.

[29] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3):965–972, 2005.