Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2012)
P. Kry and J. Lee (Editors)

Linear-Time Smoke Animation with Vortex Sheet Meshes

Tyson Brochu[1†] Todd Keeler[1‡] Robert Bridson[1,2§]

[1]University of British Columbia, Canada [2]Exotic Matter, Sweden

Abstract

We present the first quality physics-based smoke animation method which runs in time approximately linear in the size of the rendered two-dimensional visual detail. Our fundamental representation is a closed triangle mesh surface dividing space between clear air and a uniformly smoky region, on which we compute vortex sheet dynamics to accurately solve inviscid buoyant flow. We handle arbitrary moving no-stick solid boundaries and by default handle an infinite domain. The simulation itself runs in time linear to the number of triangles thanks to the use of a well-conditioned integral equation treatment together with a Fast Multipole Method for linear-time summations, providing excellent performance. Basic zero-albedo smoke rendering, with embedded solids, is easy to implement for interactive rates, and the mesh output can also serve as an extremely compact and detailed input to more sophisticated volume rendering.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

1. Introduction

Tantum videt, quantum computato: as much as one sees, that much one should compute. While the last decade and a half has seen a revolution in the quality of cinematic smoke animation through the use of physical simulation, scalability remains a serious impediment to high quality real-time smoke effects and interactive artistic design. To achieve $\sim n \times n$ visual detail in 2D renders, all prior methods have needed at least $O(n^3)$ time per step, due to the simulation or the rendering or both. In this paper we develop a purely Lagrangian vortex sheet model of smoke simulation, and argue that in the most useful asymptotic limit it captures all essential dynamic and visual detail in a closed 2D surface. We discretize the model with an adaptive dynamic triangle mesh, implement time integration using an optimal linear-time Fast Multipole Method (FMM), and render the result at interactive rates — computing only what is seen. Additionally, our method handles arbitrary no-stick moving solid boundaries and provides output for more sophisticated offline rendering.

Previous smoke simulation methods in graphics can loosely be divided between velocity-pressure formulations and vorticity approaches. The classic example of a velocity-pressure solver is Stam's Stable Fluids approach [Sta99]: this requires an n^3 grid of the volume, and even with an optimal linear-time multigrid solver for the pressure projection, is an order of magnitude more costly than the desired $O(n^2)$ rendered output. Vortex methods use a much richer representation whereby in principle many fewer elements (such as our vortex sheet of $O(n^2)$ triangles) can replicate the same detailed dynamics. However, so far in graphics the solvers hit a simulation bottleneck when calculating velocity from the vorticity and solid boundary conditions, resulting in a dense $n^2 \times n^2$ matrix solve which can be as bad as $O(n^4)$ storage and $O(n^6)$ run-time using LU factorization. Prior vortex methods also have used simpler volumetric soot particle tracking for providing output to the renderer, which introduces another expensive $O(n^3)$ bottleneck. For film visual effects, this strains the file system and network capacity in particular, as simulation is generally run separately from rendering, and so all particle data must be stored and transferred to the renderer.

† tbrochu@cs.ubc.ca
‡ keelert@cs.ubc.ca
§ rbridson@cs.ubc.ca

DOI: 10.2312/SCA/SCA12/087-095

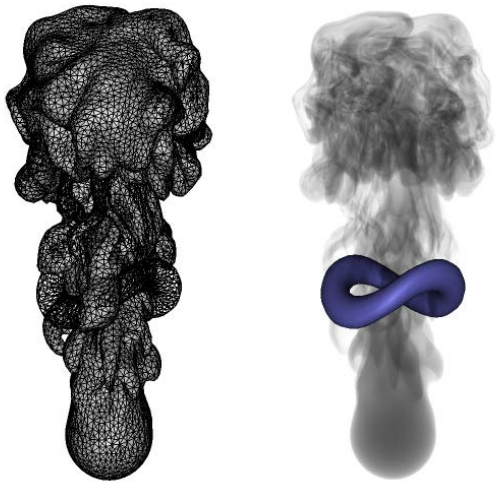

Figure 1: *We represent smoke with an adaptive triangle mesh both for linear-time simulation, as a vortex sheet, and linear-time interactive rendering as the boundary of the smoky region. Here a deforming torus influences the buoyant smoke flowing around and through it.*

Our basic model in this paper treats air as an incompressible inviscid fluid with the Boussinesq buoyancy approximation,

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \alpha \vec{g} \tag{1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2}$$

where $\alpha\vec{g}$ is the buoyancy force, with $\alpha = 0$ in the ambient clear air, and α in the smoky region depending on the ratio of smoke temperature to ambient temperature as well as the density of soot particles. At solid boundaries we take the no-stick boundary condition,

$$\vec{u} \cdot \hat{n} = \vec{u}_{\text{solid}} \cdot \hat{n}, \tag{3}$$

and at infinity the far-field condition $\vec{u} = 0$. We assume $\vec{u} = 0$ initially. Without viscosity, the vorticity $\vec{\omega} = \nabla \times \vec{u}$ remains zero apart from where buoyancy creates it; vorticity is advected and stretched by the flow, but doesn't diffuse away from smoke/temperature gradients.

Just as the viscosity in air is virtually negligible for most practical scenarios, hence the inviscid equations are an excellent model, the rate of molecular diffusion of temperature and soot particles in air is vanishingly small, so they are just advected through the velocity field. Under the assumption that smoke sources have uniform temperature and soot concentration, and also using the incompressibility condition, the air is precisely divided into uniformly clear and uniformly smoky regions. This in turn implies buoyant vorticity is only generated at the surface separating the two, where α jumps in the normal direction, hence all vorticity is concentrated in that surface, making a *vortex sheet*. As Stock et al. show [SDT08], the vorticity is further constrained to remain tangent to this sheet — in section 3, we show that our discretization identically enforces this constraint.

2. Related Work

Fluid simulation for computer animation has become a broad topic in recent years. A thorough survey of techniques is beyond the scope of this paper, so we refer the reader to the textbook by Bridson [Bri08], and focus on work related to smoke. Foster and Metaxas animated smoke using a one-phase, grid-based Navier-Stokes solver [FM97]. They used marker particles to track smoke through the simulation, transferring particle density onto a grid at render time to make use of volumetric rendering techniques. Stam introduced an unconditionally stable time integration scheme to computer graphics [Sta99], and his method was later adapted to use a staggered grid with vorticity confinement [FSJ01]. His method uses a grid-based scalar density field to track the concentration of smoke. Recent work has focused on reducing the damping effect of numerical diffusion by introducing improved integration schemes [ZB05, KLLR07, SFK*08], adding sub-grid-scale turbulence on top of a simulation [KTJG08, SB08, NSCL08], or through the use of procedural methods [SRF05, BHN07].

Losasso et al. improved the scalability somewhat with the replacement of the uniform grid by an octree structure, but adaptivity in velocity-pressure formulation is difficult as much of the volume must still be finely gridded for adequate behaviour [LGF04]. Other fluid solvers (e.g. [Exo12]) only run a solve on a sparse tiled grid instantiated in a bounded envelope around the smoke, but need a substantial envelope to approximate an unbounded domain, and run at full volumetric resolution inside the smoke, likewise limiting scalability.

The use of triangulated surface meshes in fluid animation has traditionally been eschewed in favour of grid- or particle-based surface tracking methods, such as the particle level set method [EFFM02]. Recently, however, some researchers in the computer graphics community have begun turning to triangle mesh-based surface tracking for liquid simulation [WTGT09, Mül09, BBB10, WTGT10, YWTY12]. These authors generally use explicit surfaces to track the fluid interface in a free-surface simulation, whereas in this paper, we use surfaces themselves as simulation elements.

Mesh surfaces have been used for visualizing 3D vector fields since Hultquist generalized streamlines to stream surfaces [Hul92]. More recently, *streak surfaces* (moving open surfaces seeded continuously from a curve), and *time surfaces* (seeded instantaneously from a surface) have been treated using surface tracking methods similar to the one

used in this paper. Krishnan et al. [KGJ09] recently introduced an algorithm which advects an adaptive mesh through a 3D vector field to treat both of these categories of surface. A similar visualization method which explicitly uses a smoke metaphor was introduced by von Funck et al. [vFWTS08]. Their method does not use adaptivity, but instead gradually reduces the opacity of poor quality triangles, giving the appearance of dissipating smoke or steam.

Brochu & Bridson [BB09a] used surface tracking to visualize a procedural smoke simulation. Park et al. [PSCN10] constructed NURBS surfaces from particle streamlines to achieve thin, sheet-like surface renderings of smoke simulations. These techniques are useful for visualizing an external simulation, however neither approach uses the surface elements for simulation, as we propose in this paper.

Vortex methods for fluid dynamics were developed during the 1980s as an efficient and high-order, purely Lagrangian method of numerically solving the Euler equations for fluid flow [BM82,AG85]. Subsequent work was done on developing viscous vortex methods for the Navier-Stokes equations and semi-Lagrangian contexts [CK00]. Though the grid distortion in fully Lagrangian vortex methods makes them less useful for scientific application, their efficiency in preserving rotational flow makes them appealing for fluid animation.

In computer graphics, early 2D simulations discretized vorticity on a grid [YUM86], or used vortex-in-cell methods, advecting Lagrangian parcels of vorticity through the domain but relying on a grid-based Eulerian solver [CMTM94, GLG95]. Park and Kim [PK05] introduced a fully Lagrangian vortex particle method, using the Biot-Savart formula to compute velocity on the particles without a grid. Recently, Weißmann and Pinkall [WP10] have demonstrated a complete vortex filament method, with adaptive filaments. Both approaches discretize solid boundaries with vortex sheets and rely on static geometry to enable precomputation for efficiently handling boundary conditions. In comparison, our linear-time approach can handle arbitrary moving solid boundaries, as demonstrated in section 4.

Several papers have used vortex primitives to augment Eulerian fluid simulations, including vortex particles [SRF05] and Eulerian vortex sheets [KSK09]. Concurrent with this work, Pfaff et al. have introduced a Lagrangian vortex sheet method for augmenting an Eulerian fluid solver [PTG12]. Their vortex sheet discretization is similar to the one presented in this paper; however their vortex sheet interactions are purely local, with global flow handled by an underlying grid-based, Eulerian fluid solver. Their use of compact kernels avoids the need for FMM, but the reliance on the grid-based solver limits scalability and complicates the simulation process.

Boundary integral equations were initially developed a century ago as a tool for proving the uniqueness and existence for solutions to PDEs, particularly Laplace's equation. They have had a modern renaissance as effective numerical methods for solving a variety of PDEs [GGM93]. This has been primarily motivated by the construction of effective summation methods such as the FMM which allows rapid solution of the dense linear equations resulting from the numerical treatment of many integral equations.

3. The Vortex Sheet Mesh

Our discretization of the vortex sheet closely follows Stock et al. [SDT08] and Pfaff et al. [PTG12] (though unlike their methods which rely on a 3D background grid for bulk velocity computation, we directly compute it in linear time on the surface itself with FMM, including arbitrary solid boundaries, as we will see in section 5). In particular, we discretize the vortex sheet surface with a triangle mesh, and store circulation values, Γ_i, on each mesh edge i. The per-triangle vortex sheet strength is then:

$$\vec{\gamma}_p = \frac{1}{a_p} \sum_j \vec{e}_j \Gamma_j, \tag{4}$$

for triangle edges j, where $\vec{e}_j$ is the vector between the end points of edge j, and where a_p is the area of triangle p.

The vorticity on a triangle is then computed as required with the formula:

$$\vec{\omega}_p = a_p \vec{\gamma}_p = \sum_j \vec{e}_j \Gamma_j. \tag{5}$$

for triangle edges j, where $\vec{e}_j$ is the vector between the end points of edge j. The main advantage of using these scalar circulations as our fundamental variable is that they are conserved along particle paths. The vorticity stretching is thus implicitly handled by the stretching of the vortex sheet mesh. Recomputing the vortex sheet strength when needed for the velocity update is done according to (5). Using (5) also guarantees that $\vec{\omega}$ is tangent to the triangle plane.

We apply a buoyancy force $\alpha\vec{g} = \langle 0, \alpha, 0 \rangle$ to each triangle as $\Delta\vec{\omega}_i = \alpha\vec{g} \times \vec{n}_i A_i$ for triangle normal $\vec{n}_i$ and area A_i. We then compute the change in circulation for each triangle edge by solving the overdetermined system of equations 5 for Γ as suggested by Stock et al. [SDT08]. We solve this in the least-squares sense. We note that recomputing the *total* edge circulations in this way would introduce numerical damping manifesting as undesirable loss of vorticity, so we only solve for the *change* in edge vorticity due to the buoyancy force.

With no solid boundaries, the velocity at any point in the domain can be computed explicitly with the Biot-Savart law, with S being the vortex sheet surface, and K being a (mollified) fundamental solution to Laplace's equation:

$$\vec{u}_{\text{fluid}}(\vec{x}) = \int_S \nabla K(\vec{x} - \vec{y}) \times \vec{\omega}(\vec{y}) dy \tag{6}$$

This is discretized on a mesh surface using midpoint

quadrature as:

$$\vec{u}_{\text{fluid}}(\vec{x}) \approx \sum_j \nabla K(\vec{x} - \vec{c}_j) \times \vec{\omega}_j \tag{7}$$

where $\vec{c}_j$ is the barycentre of triangle j, and $\vec{\omega}_j$ is the constant vorticity on triangle j.

Since we require the velocity at each vertex, and every velocity evaluation depends on the entire surface, this direct summation has computational complexity $O(M^2)$ in the number of mesh vertices. Later in this paper we will discuss using the Fast Multipole Method to achieve an approximation which runs in $O(M)$ time.

The gradient of the fundamental solution $\nabla K(\vec{x} - \vec{y})$ is singular when $\vec{x} = \vec{y}$ and becomes large as $\vec{x}$ approaches $\vec{y}$, therefore a mollified approximation is often used. Our implementation follows Beale and Majda's form [BM85]:

$$\nabla \tilde{K}(\vec{x} - \vec{y}) = -(\vec{x} - \vec{y}) \frac{1 - e^{(r/\beta)^3}}{r^3}, \tag{8}$$

where $r = ||\vec{x} - \vec{y}||$, and β is a mollification parameter.

As our mesh evolves due to this self-imparted velocity, it naturally expands and curls up. Without some form of surface mesh refinement, the simulation would quickly lose accuracy and visual appeal. On the other hand, we wish to keep the number of mesh elements as low as possible, to keep the computational cost down. Therefore, removing small elements (recoarsening) is also desirable. For mesh refinement and recoarsening operations, we use the open-source El Topo surface tracking library [BB09b]. This library also has the ability to perform changes in topology, such as merging and pinching. By merging nearby surface patches, we can eliminate very thin sheets, thereby further reducing the total number of mesh triangles while still maintaining a good quality simulation.

3.1. Circulation redistribution

El Topo's surface remeshing operations add and remove edges. As we are storing scalar circulation values on the edges, we must update these values as the mesh connectivity changes. One way of doing so is to compute per-triangle vorticity values using equation 5, performing the remeshing operations, then solving equation 5 for Γ in a least-squares sense. However, we have found that repeatedly solving this overdetermined system leads to diffusion of vorticity.

An alternative is to explicitly update the edge circulation values using some rules of thumb when a remeshing operation is performed. Though heuristic, we have found these rules reduce vorticity diffusion as compared to repeatedly projecting back and forth between edge and triangle values. Of the local remeshing operations available in El Topo, we use only edge splitting, edge collapsing, and mesh merging via deletion and creation of new triangles.

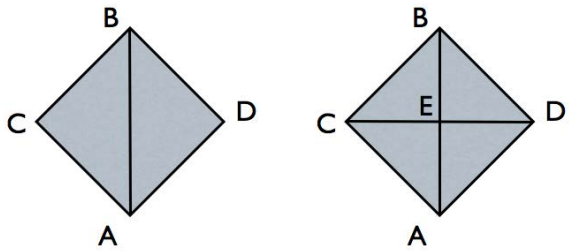

Figure 2: *Edge split operation. New edges AE and EB are assigned circulation from original edge AB.*

3.1.1. Edge split

As shown in figure 2, splitting edge AB with opposite vertices C and D introduces a new vertex E. We store the circulation value on the original edge AB, and assign it to both of the new edges:

$$\Gamma_{\text{AE}} = \Gamma_{\text{EB}} = \Gamma_{\text{AB}}. \tag{9}$$

The two other new edges, CE and ED, receive no circulation. This ensures that the sum of vorticities over the new triangles (by equation 5) equals the sum of vorticities on the new triangles:

$$\vec{\omega}_{\text{ABC}} + \vec{\omega}_{\text{DBA}} = \vec{\omega}_{\text{AEC}} + \vec{\omega}_{\text{CEB}} + \vec{\omega}_{\text{DBE}} + \vec{\omega}_{\text{DEA}} \tag{10}$$

3.1.2. Edge collapse

When collapsing an edge, we simply allow the circulation on that edge to disappear from the system. Since we collapse only very short edges, the loss in circulation is generally minimal.

3.1.3. Mesh merge

El Topo merges surface meshes by removing two nearby edges and their incident triangles, and introducing new triangles, forming a neck between the surfaces [BB09b]. From the new edges introduced, we pick the two edges whose associated vectors (difference of end points) best match the removed edges, i.e. finding new edges i and j such that

$$||\vec{e}_i - \vec{e}_0|| + ||\vec{e}_j - \vec{e}_1||$$

is minimized, where edges 0 and 1 are the removed edges. We assign the original edge circulations to these "best match" new edges (see figure 3).

4. Solid Boundaries

Boundary conditions for Lagrangian vorticity-based fluid animation have generally been constructed by defining a vortex sheet on the surface of the solid boundaries. Park and Kim [PK05] implemented boundary conditions in 3D by enforcing the tangential and normal components of the fluid

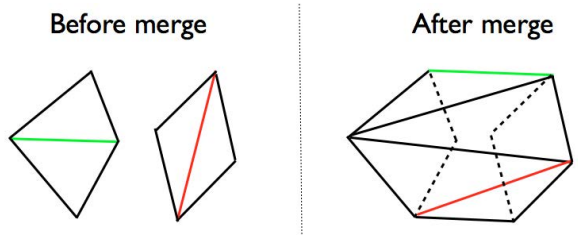

Figure 3: *Edge merge operation. Left: The red and green edges and their incident triangles are removed from the mesh, leaving two quad-shaped holes. Right: The resulting holes are zippered with new triangles. Suppose the red and green edges in the right diagram are the closest matches to the corresponding red and green edges in the left diagram (comparing Euclidean distances of associated edge vectors). These new edges are then assigned the circulation values from the deleted edges.*

velocity to be equal to those of the boundary using a vortex sheet, which resulted in an over-determined matrix solve. To avoid finding an optimal solution each time-step, they restricted themselves to a simple static boundary for which they precomputed a pseudo-inverse, requiring only a matrix-vector multiplication instead of a matrix solve. Weißmann and Pinkall [WP10] defined their vortex sheet in terms of a scalar potential, avoiding the extra dimensionality of the tangential vorticity field. They regularized the resulting matrix and also precomputed a factorization, again limiting their domain to a single boundary with static geometry.

We incorporate solid boundaries by adding an irrotational, divergence-free vector field that corrects the boundary velocity induced by the vorticity to account for solids. To do so we define the irrotational flow by the gradient of a scalar potential function Φ that satisfies Laplace's equation $\nabla^2\Phi = 0$ in the domain. The potential due to a single point charge with value σ, at point $\vec{y}$, is:

$$\frac{\sigma}{4\pi|\vec{x}-\vec{y}|}. \tag{11}$$

We represent Φ by a continuous distribution of point charges $\{\sigma(\vec{y})|y \in S\}$ where S is the boundary of our domain.

$$\Phi(\vec{x}) = \int_S \frac{\sigma(\vec{y})}{4\pi|\vec{x}-\vec{y}|}dy. \tag{12}$$

This representation of Φ is called the single layer potential. The normal derivative of the single layer potential has a discontinuity across the domain boundary. When satisfying boundary conditions, this leads to the following Fredholm integral equation of the second kind for σ [Kel29, HS62]. Given the normal derivative f of Φ on the boundary we have:

$$f = -\sigma(\vec{x})/2 + \int_S \sigma(\vec{y})\frac{\partial}{\partial n_x}\frac{1}{4\pi|\vec{x}-\vec{y}|}dy, \quad \vec{x},\vec{y} \in S. \tag{13}$$

Fredholm equations of the second kind are integral equations consisting of the identity operator plus an integral operator that is "compact". While the definition for compact requires an understanding of functional analysis and is therefore beyond the scope of this paper, an integral operator is compact if the multiplicative terms in the integral are themselves square integrable, which is the case in (13). The solvability of Fredholm equations of the second kind can be determined by examining the null-space of the equation, similar to determining the solvability of linear systems. For smooth boundaries, (13) has only a trivial null-space and thus its solution exists and is unique for any right hand side f. In addition, solvable Fredholm equations of the second kind tend to lead to well conditioned numerical approximations. While our description of integral equations is cursory and incomplete, there exists ample mathematical literature for the interested. Solving (13) for σ and evaluating Φ from equation (12) satisfies the exterior Neumann problem for Laplace's equation,

$$\nabla^2\Phi = 0 \tag{14}$$

$$\frac{\partial}{\partial n}\Phi(x) = f, \quad x \in S. \tag{15}$$

Let $\vec{u}_{\text{fluid}}$ be the velocity due to the fluid, and $\vec{u}_{\text{solid}}$ be the velocity of the solid object. Setting $f = (\vec{u}_{\text{solid}} - \vec{u}_{\text{fluid}}) \cdot \vec{n}$, solving for potential Φ, and calculating the gradient yields a potential flow which will cancel the normal component of velocity at the solid boundary, effecting a free-slip, no-entry boundary condition.

The advantages of this scalar formulation over the previously described vortex sheet boundary implementations are that it is solvable for arbitrary smooth solid boundaries and is particularly amenable to numerical solutions based on iterative matrix solvers and fast summation methods. Integral equations for implementing boundaries based on vortex sheets are more difficult to implement, and though one can derive Fredholm integral equations of the second kind for finding a boundary vorticity [CK00], these formulations become singular if the domain is multiply-connected.

Discretizing (13) with midpoint quadrature yields the following equation for triangle i:

$$f_i \approx -\sigma_i/2 + \sum_j \sigma_j \frac{\partial}{\partial n_i}\frac{1}{4\pi|\vec{c}_i - \vec{c}_j|}A_j \tag{16}$$

Directly evaluating this summation for all triangles yields an $M \times M$ linear system where M is the number of triangles on the solid boundary. In practice, this matrix is very well-conditioned; for such problems, iterative Krylov solvers such as BiCGSTAB have been observed to converge in $O(1)$ iterations, resulting in total complexity of $O(M^2)$ for the solid solve. We can additionally accelerate this to $O(M)$ by using the FMM to compute the matrix-vector product in the iterative solve rather than explicitly constructing the system.

Once we have solved for a density σ, we can compute Φ as in equation (12). The gradient of Φ is then evaluated as:

$$\nabla_x \Phi(\vec{x}) = \int_S \frac{-\sigma(\vec{y})}{4\pi|\vec{x}-\vec{y}|^3}(\vec{x}-\vec{y})dy \tag{17}$$

The midpoint quadrature rule for this integral yields:

$$\nabla_x \Phi(\vec{x}) \approx \sum_j \frac{-\sigma_j A_j}{4\pi|\vec{x}-\vec{c}_j|^3}(\vec{x}-\vec{c}_j) \tag{18}$$

We use this to modify our fluid surface vertex velocities:

$$\vec{u}_{\text{final}}(\vec{x}) = \vec{u}_{\text{fluid}}(\vec{x}) + \nabla\Phi(\vec{x}) \tag{19}$$

5. Fast Multipole Method

The Fast Multipole Method (FMM), initially introduced by Greengard and Rhoklin [GR87], reduces the asymptotic complexity of the N-body problem from $O(N^2)$ to $O(N)$. The FMM is used in two different areas of our fluid simulation. Computing the velocity from the vorticity using the Biot-Savart law can be accomplished using an FMM for each component of the vorticity. In addition, computing the matrix multiply in the iterative solver for the solid-boundary potential flow can be reduced from $O(M^2)$ to $O(M)$ by using the FMM to approximate Φ:

$$\Phi(\vec{x}_i) = \sum_j \frac{\sigma_j}{|\vec{x}_i - \vec{x}_j|}, \quad i,j = 1..N, \tag{20}$$

and its gradient,

$$\nabla\Phi(\vec{x}_i) = \sum_j \frac{-\sigma_j(\vec{x}_i - \vec{x}_j)}{|\vec{x}_i - \vec{x}_j|^3} \tag{21}$$

for a set of points $\vec{x}_i$ and "charges" σ_i. Our 3d FMM implementation follows that described by Cheng et al. [CGR99].

Mild controversy surrounds the FMM's claim to $O(N)$ complexity, since straightforward computation of the octree data structure required by the FMM is $O(N \log N)$. We compute our octree in $O(N)$ complexity. To do so, we specify beforehand the maximum depth of the octree. We then compute unsigned integer coordinates for each particle that correspond to the grid cell of the finest octree subdivision that contains the particle. A single key is then created by interleaving the bits of each of the three coordinates creating a 3D Morton ordering of the occupied grid cells. We sort the particles by these interleaved keys using a radix sort in $O(N)$. The result of the sort is that the particles are arranged in a linear octree, from which the hierarchical octree structure can be constructed in $O(N)$.

The log-log plot in figure 4 shows the order of complexity for computing velocity due to vorticity using the direct summation method vs. the FMM. Performance plots for the solid solver are shown in figure 5.

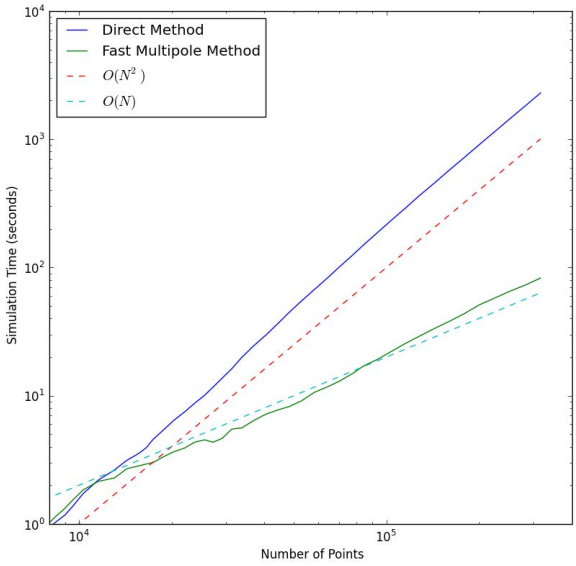

Figure 4: *Simulation time per frame for computing velocity via the Biot-Savart law, for both the direct method and Fast Multipole Method*

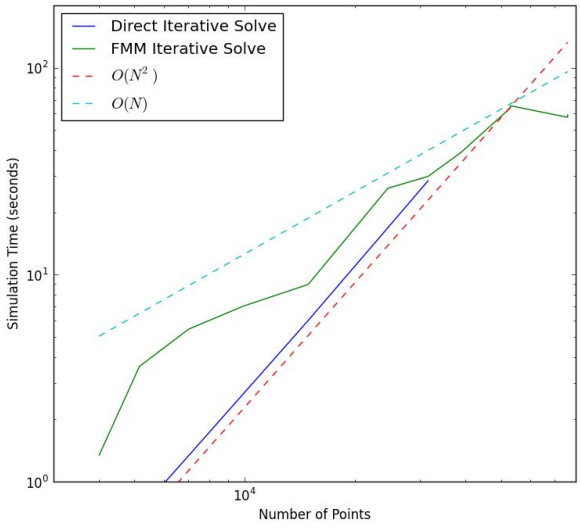

Figure 5: *Simulation time per frame for computing the single layer potential on a sphere using BiCGSTAB, computing the matrix multiply with both the direct method and Fast Multipole Method. Note that the direct method is nearly exactly $O(N^2)$ showing the $O(1)$ convergence of BiCGSTAB*

6. Rendering

Our interactive-rate smoke rendering is accomplished through a simple set of OpenGL shaders, which compute the optical depth of the smoke volume. We then apply a zero-albedo absorption model based on this depth, taking into account the solid objects in the scene. More sophisticated real-time smoke rendering methods do exist (cf. [ZRL*08]), and exploring self-shadowing and light scatter for triangle mesh models in real time is an area of potential future work.

As Brochu and Bridson [BB09b] point out, for high quality cinematic smoke rendering in a typical film pipeline, the critical bottleneck is often the load on the file system and network from tracking smoke per frame with hundreds of millions of soot particles, or similarly high resolution grids. The surface mesh representation is vastly more efficient for capturing the volumetric bulk of the smoke, and as Brochu and Bridson note can be directly ray-traced for self-shadowing smoke rendering. We extend their idea noting that the interior of the surface mesh, at render time, can easily be rasterized into a lower resolution 3D texture for scattering computation, at least using a box filter via polygon clipping; if this grid is aligned with the light source or the view, especially efficient calculation of single-scattering is possible. The direct render can use triangle rasterization of the mesh, storing depths in an A-buffer, looking up into the 3D texture for smooth lighting calculations. Going even further, we can track the age of mesh elements since their emission from a source, and splat this into another render channel to provide age-proportional blurring, thus giving the visual effect of smoke dissipation from turbulent mixing or diffusion at render time, for every frame in parallel. Figure 6 illustrates the effect of an image-space age-dependent blur.

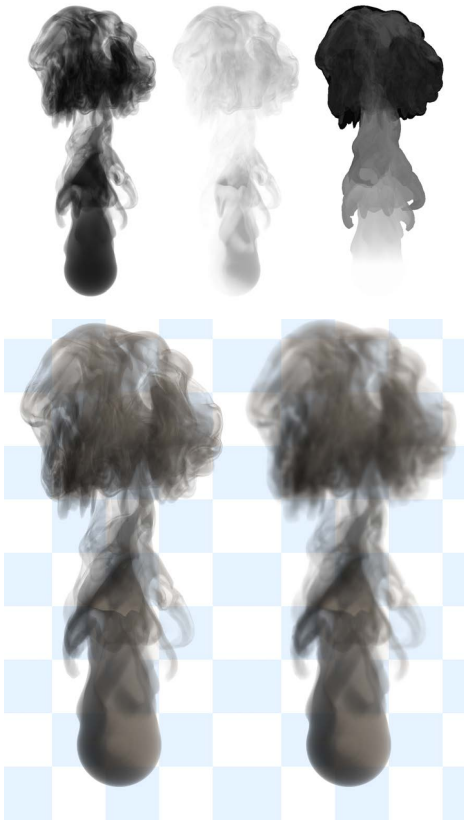

Figure 6: *Our proof-of-concept volume renderer produces, from left to right on the top row, an alpha channel from the smoke mesh, a scattering channel with self-shadowing, and an image-space age channel. Below on the left is a sharp composite of the first two, and below on the right includes an age-dependent blur, simulating diffusion in image-space at render-time.*

7. Results

Our FMM-accelerated vortex sheet solver was implemented in C++, using El Topo for surface tracking, and tested on several Linux and Mac OS X computers. Figures 4 and 5 show that our tests have empirically near-linear time complexity. Since we do not rely on precomputing the solution to the exterior Neumann problem, we can easily handle non-rigidly deforming solid objects, as shown in figure 1.

Our interactive smoke renderer implemented in GLSL runs at 4 FPS for 168K triangles on an AMD Radeon 6770M with 512 MB of memory (shown in figure 1). Our proof-of-concept software renderer uses CPU-based volumetric rasterization and self-shadowing to produce the images shown in figure 6. These images were produced from 126K triangles at 1600×1200 resolution with a $167 \times 125 \times 82$ shadow texture in around 5.5 s on a single thread of a 2.3GHz Intel Core i7 CPU, including all file IO.

7.1. Mesh simplification

As mentioned in section 3, remeshing operations are crucial for allowing the surface mesh to move freely while keeping the number of triangles manageable. The effectiveness of edge splitting and collapsing to control the explosion of mesh elements has been shown by Stock et al. [SDT08] and others, but we also make use of El Topo's topology change operations — specifically the merging and pinching of mesh surfaces. We have found this to be an effective tool in mitigating the explosion of surface area (and number of mesh elements), which is a known problem in vortex sheet simulations.

As an example, we ran a simulation twice, once with topology changes enabled, and once with these changes disabled. The number of triangles per time step is shown in figure 7. Note that without topology changes enabled, the number of triangles grows exponentially, but with aggres-

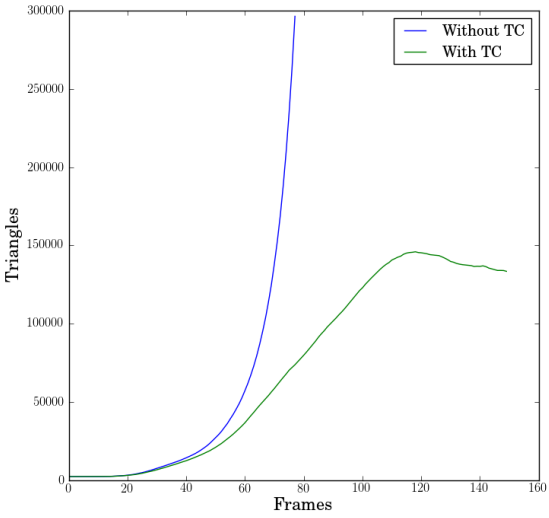

Figure 7: *Geometry creation when simulating a smoke plume without topological changes (Without TC) and with topological changes (With TC).*

sive topological merging and splitting, the number of triangles remains bounded (see the accompanying video).

Pfaff et al. [PTG12] also address this explosion in the number of mesh elements. Their approach identifies triangles which are deep within the volume of smoke (using a grid-based signed distance field), or otherwise occluded from the camera view, and marks them for deletion. By contrast, our approach uses only Lagrangian mesh-based operations, without additional structures or heuristics.

8. Future Work

We have demonstrated that interactive-rate, near-cinematic-quality smoke simulation *and* rendering is within reach, but there are many avenues to explore further:

- Code optimization and parallelization of the FMM solver to achieve interactive simulation rates.
- Porting our software A-buffer rasterizer to run self shadowing and diffusion in GPU hardware.
- View-dependent, level-of-detail mesh refinement, coarsening, and topology changes.
- Incorporating the creation of vorticity from solid interactions (vortex shedding).
- Simulation of flame front propagation to achieve vortex sheet based fire simulation, combined with real-time lighting techniques on the rendering side.
- Simulation of ocean wave free surfaces — much of our vortex sheet work could apply to rough and vast simulations, which are currently infeasible with volumetric simulation methods.

Acknowledgements

This work was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada. We also thank Mary Bridson for the Latin turn of phrase.

References

[AG85] ANDERSON C., GREENGARD C.: On vortex methods. *SIAM J. Num. Anal. 22*, 3 (June 1985), 413–440. 3

[BB09a] BROCHU T., BRIDSON R.: Animating smoke as a surface. Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim. (posters and demos), 2009. 3

[BB09b] BROCHU T., BRIDSON R.: Robust topological operations for dynamic explicit surfaces. *SIAM J. Sci. Comput. 31*, 4 (2009), 2472–2493. 4, 7

[BBB10] BROCHU T., BATTY C., BRIDSON R.: Matching fluid simulation elements to surface geometry and topology. *ACM Trans. Graph. (Proc. SIGGRAPH) 29*, 4 (2010), 47:1–47:9. 2

[BHN07] BRIDSON R., HOURIHAM J., NORDENSTAM M.: Curl-noise for procedural fluid flow. *ACM Trans. Graph. (Proc. SIGGRAPH) 26*, 3 (2007), 46:1–46:3. 2

[BM82] BEALE J. T., MAJDA A.: Vortex methods I: Convergence in three dimensions. *Mathematics of Computation 39*, 159 (1982), 1–27. 3

[BM85] BEALE J., MAJDA A.: High order accurate vortex methods with explicit velocity kernels. *Journal of Computational Physics 58*, 2 (1985), 188–208. 4

[Bri08] BRIDSON R.: *Fluid Simulation for Computer Graphics*. A K Peters, 2008. 2

[CGR99] CHENG H., GREENGARD L., ROKHLIN V.: A fast adaptive multipole algorithm in three dimensions. *J. Comp. Phys. 155*, 2 (1999), 468–498. 6

[CK00] COTTET G.-H., KOUMOUTSAKOS P. D.: *Vortex Methods: Theory and practice*. Cambridge University Press, 2000. 3, 5

[CMTM94] CHIBA N., MURAOKA K., TAKAHASHI H., MIURA M.: Two-dimensional visual simulation of flames, smoke and the spread of fire. *J. Visualization and Comp. Anim. 5*, 1 (1994), 37–53. 3

[EFFM02] ENRIGHT D., FEDKIW R., FERZIGER J., MITCHELL I.: A hybrid particle level set method for improved interface capturing. *J. Comp. Phys. 183*, 1 (2002), 83–116. 2

[Exo12] EXOTIC MATTER: Naiad 0.6 documentation. 2

[FM97] FOSTER N., METAXAS D.: Modeling the motion of a hot, turbulent gas. In *Proc. SIGGRAPH* (New York, NY, USA, 1997), pp. 181–188. 2

[FSJ01] FEDKIW R., STAM J., JENSEN H. W.: Visual simulation of smoke. In *Proc. SIGGRAPH* (2001), pp. 15–22. 2

[GGM93] GREENBAUM A., GREENGARD L., MCFADDEN G. B.: Laplace's equation and the Dirichlet-Neumann map in multiply connected domains. *J. Comp. Phys. 105*, 2 (1993), 267–348. 3

[GLG95] GAMITO M. N., LOPES P. F., GOMES M. R.: Two-dimensional simulation of gaseous phenomena using vortex particles. In *Proc. Eurographics Workshop on Computer Animation and Simulation* (1995), pp. 3–15. 3

[GR87] GREENGARD L., ROKHLIN V.: A fast algorithm for particle simulations. *J. Comp. Phys. 73*, 2 (1987), 325–348. 6

[HS62] HESS J. L., SMITH A.: *Calculation of non-lifting potential flow about arbitrary three-dimensional bodies*. Tech. rep., DTIC Document, 1962. 5

[Hul92] HULTQUIST J. P. M.: Constructing stream surfaces in steady 3D vector fields. In *Proc. IEEE Visualization* (1992), pp. 171–178. 2

[Kel29] KELLOGG O. D.: *Foundations of Potential Theory*. Dover Publications, 1929. 5

[KGJ09] KRISHNAN H., GARTH C., JOY K. I.: Time and streak surfaces for flow visualization in large time-varying data sets. In *Proc. IEEE Visualization* (2009). 3

[KLLR07] KIM B., LIU Y., LLAMAS I., ROSSIGNAC J.: Advections with significantly reduced dissipation and diffusion. *IEEE Trans. Vis. Comp. Graph. 13*, 1 (2007), 135–144. 2

[KSK09] KIM D., SONG O.-Y., KO H.-S.: Stretching and wiggling liquids. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* (2009), 120. 3

[KTJG08] KIM T., THÜREY N., JAMES D., GROSS M.: Wavelet turbulence for fluid simulation. *ACM Trans. Graph. (Proc. SIGGRAPH)* (2008), 1–6. 2

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. *ACM Trans. Graph. (Proc. SIGGRAPH) 23*, 3 (2004), 457–462. 2

[Mül09] MÜLLER M.: Fast and robust tracking of fluid surfaces. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.* (2009), pp. 237–245. 2

[NSCL08] NARAIN R., SEWALL J., CARLSON M., LIN M.: Fast animation of turbulence using energy transport and procedural synthesis. *ACM Trans. Graphics (Proc. SIGGRAPH Asia) 27*, 5 (2008), 166:1–166:8. 2

[PK05] PARK S. I., KIM M. J.: Vortex fluid for gaseous phenomena. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.* (2005), pp. 261–270. 3, 4

[PSCN10] PARK J., SEOL Y., CORDIER F., NOH J.: A smoke visualization model for capturing surface-like features. *Computer Graphics Forum (Proc. Eurographics) 29*, 8 (2010), 2352–2362. 3

[PTG12] PFAFF T., THUEREY N., GROSS M.: Lagrangian vortex sheets for animating fluids. *ACM Trans. Graph. (Proc. SIGGRAPH) 31*, 4 (2012), 112:1–112:8. 3, 8

[SB08] SCHECHTER H., BRIDSON R.: Evolving sub-grid turbulence for smoke animation. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.* (2008). 2

[SDT08] STOCK M. J., DAHM W. J., TRYGGVASON G.: Impact of a vortex ring on a density interface using a regularized inviscid vortex sheet method. *J. Comp. Phys. 227*, 21 (2008), 9021 – 9043. 2, 3, 7

[SFK*08] SELLE A., FEDKIW R., KIM B., LIU Y., ROSSIGNAC J.: An unconditionally stable MacCormack method. *SIAM J. Sci. Comput. 35*, 2-3 (2008), 350–371. 2

[SRF05] SELLE A., RASMUSSEN N., FEDKIW R.: A vortex particle method for smoke, water, and explosions. *ACM Trans. Graph. (Proc. SIGGRAPH) 24*, 3 (2005), 910–914. 2, 3

[Sta99] STAM J.: Stable fluids. In *Proc. SIGGRAPH* (1999), pp. 121–128. 1, 2

[vFWTS08] VON FUNCK W., WEINKAUF T., THEISEL H., SEIDEL H.-P.: Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. *IEEE Trans. Vis. Comp. Graph. 14*, 6 (2008), 1396–1403. 3

[WP10] WEISSMANN S., PINKALL U.: Filament-based smoke with vortex shedding and variational reconnection. *ACM Trans. Graph. (Proc. SIGGRAPH) 29*, 4 (2010), 115:1–115:12. 3, 5

[WTGT09] WOJTAN C., THÜREY N., GROSS M., TURK G.: Deforming meshes that split and merge. *ACM Trans. Graph. (Proc. SIGGRAPH) 28*, 3 (2009), 1–10. 2

[WTGT10] WOJTAN C., THÜREY N., GROSS M., TURK G.: Physics-inspired topology changes for thin fluid features. *ACM Trans. Graph. (Proc. SIGGRAPH) 29*, 4 (2010), 50:1–50:8. 2

[YUM86] YAEGER L., UPSON C., MYERS R.: Combining physical and visual simulation–creation of the planet jupiter for the film "2010". *SIGGRAPH Comput. Graph. 20*, 4 (Aug. 1986), 85–93. 3

[YWTY12] YU J., WOJTAN C., TURK G., YAP C.: Explicit mesh surfaces for particle based fluids. *Computer Graphics Forum (Proc. Eurographics) 31*, 2 (2012). 2

[ZB05] ZHU Y., BRIDSON R.: Animating sand as a fluid. *ACM Trans. Graph. (Proc. SIGGRAPH)* (2005), 965–972. 2

[ZRL*08] ZHOU K., REN Z., LIN S., BAO H., GUO B., SHUM H.-Y.: Real-time smoke rendering using compensated ray marching. *ACM Transactions on Graphics (Proc. SIGGRAPH) 27*, 3 (2008), 36:1–36:12. 7