Fluid Surface Reconstruction from Particles

by

Brent Warren Williams

Hons. B.Math, The University of Waterloo, 1995

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

(Vancouver)

Abstract

Outlined is a new approach to the problem of surfacing particle-based fluid simulations. The key idea is to construct a surface that is as smooth as possible while remaining faithful to the particle locations. We describe a mesh-based algorithm that expresses the surface in terms of a constrained optimization problem. Our algorithm incorporates a secondary contribution in Marching Tiles, a generalization of the Marching Cubes isosurfacing algorithm. Marching Tiles provides guarantees on the minimum vertex valence, making the surface mesh more amenable to numerical operators such as the Bilaplacian.

Table of Contents

List of Figures

Acknowledgments

First thanks to Prof. Robert Bridson for remarkable insights and patience. I could not have asked for a better advisor.

Thanks as well to Prof. Ian Mitchell and Robert McGovern for many helpful comments in the preparation of this thesis.

This work would have not have been possible without the generosity and good will of employers past and present, in particular I'd like to thank Greg Brill, DeBorah Johnson and Randy Pyburn.

Finally I'd like to thank my close-knit but far-flung family for their support and encouragement, and those I missed and who missed me on many a long evening.

Chapter 1

Introduction

In computer animations complex dynamic materials such as hair, cloth and fluid are difficult to model directly. Physically-based simulations provide an alternative by exercising the governing physical equations. We are interested here in simulations of table-top fluid mechanics such as pouring water into a glass. Fluid simulations may be divided into two broad categories, Eulerian (grid-based) and Lagrangian (particle-based). As with all physical simulations there are three basic steps involved — performing the simulation, extracting a renderable representation, and performing the rendering.

Eulerian simulations suffer from a number of numerical difficulties including diffusion of small-scale structure, mass loss and spatial discretization complexities. However extracting a high quality surface is straightforward, a critical advantage over particle-based approaches and one of the primary reasons for the current domination of the Eulerian approach.

Our primary contribution lies in the field of Lagrangian simulations, specifically extracting a renderable fluid surface from particle locations. This problem has been little addressed in the literature and is something of an Achilles heel of the Lagrangian approach. The seminal work of Enright et al. [19] advocates the Eulerian approach in part because "Particle methods,

while quite versatile, pose difficulties when trying to reconstruct a smooth water surface from the locations of the particles alone". Premože et al. [47] echoes the lament: "The biggest disadvantage of using particle methods is the question of surface representation". Our particle surfacing algorithm is described in chapter 3.

One step in our surfacing algorithm calls for isosurfacing, i.e. generating a surface of constant value from a volume of scalars. The standard isosurfacing algorithm is Marching Cubes, which, along with its many derivatives, generates surfaces that contain many vertices with a valence (degree) of four. Valence four vertices represent a relatively poor sampling of the underlying surface and do not behave well when numerical procedures such as smoothing, normal extraction, simplification and so on are applied. We introduce a new algorithm dubbed Marching Tiles which guarantees a minimum valence of five on the generated isosurface, allowing for improvements in the quality of our fluid surfaces, with application to other areas in computer graphics. The theory and implementation behind Marching Tiles is described in chapter 4.

Our fluid surfacing algorithm is motivated by perceived advantages in Lagrangian approaches over Eulerian approaches in fluid simulations. We begin by justifying this premise.

Chapter 2

Fluid Simulation

2.1 Eulerian Approaches

Eulerian simulations are characterized by a generally rectangular domain within which fluid properties such as velocity and pressure are tracked at fixed points in space. The values of these properties are determined by a governing equation such as Navier-Stokes [7] which is discretized and integrated across time and/or space. The properties determine the behaviour of the fluid in terms of compressibility, viscosity, elasticity and so on.

It is generally necessary to model fluid throughout the entire domain, although in computer graphics we are usually only interested in the location of the fluid surface, or *interface*. In Eulerian simulations the interface is encoded implicitly, for example within a level set $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}$. At each point ϕ contains the signed distance to the nearest interface. By convention, locations where $\phi > 0$ lie outside the fluid and locations where $\phi < 0$ lie inside the fluid [45]. Implicitly the surface itself lies at $\phi = 0$. There are of course alternatives to level sets in the Eulerian framework, such as the Volume-of-Fluid (VOF) method; however, they suffer from the same sampling problem of any Eulerian method discussed below, and generally don't fare as well when extracting a quality surface for rendering.

The use of level sets in grid-based approaches makes extracting a renderable representation straightforward. The zero-level isosurface can be generated from the level set in the form of a mesh using the simple and well-established Marching Cubes algorithm ([36], see also section 4.1). It is also possible to render level sets using ray tracing, although volume primitives are not always supported natively nor do most renderers rely on ray-tracing exclusively. We will restrict our consideration to the mesh generation approach usable with all rendering engines.

Although extracting a mesh for use in rendering the fluid surface is straightforward, the grid-based approach introduces a number of difficulties in performing the simulation itself. One problem lies in the difficulty of representing small-scale features of size comparable to or smaller than the grid spacing. This can be addressed using adaptive techniques such as octrees [38] which 'zoom in' on features near the interface. Octrees may be supplemented with the particle level set method (PLS [17]) which sprinkles Lagrangian 'shepherd' particles on either side of the interface in order to correct (and be corrected by) the position of the interface within the level set (see figure 2.1).

While techniques such as octrees and PLS represent an improvement over ordinary level sets, fundamentally small features cannot be represented on a grid and so are not reliably sampled. This undersampling inevitably leads to artifacts such as mass loss in the form of holes that suddenly open up in thin surfaces, and voxelization in the limits of the spatial refinement (figure 2.2). The problem of mass loss is inherent to level set approaches and is exacerbated as the surface grows in complexity [39]. The undersampling

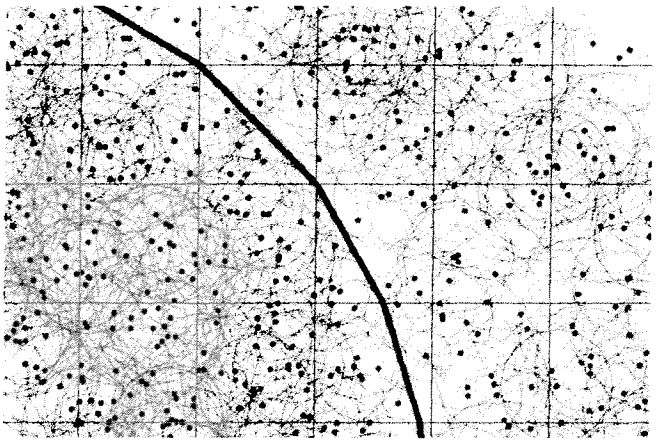

Figure 2.1: The Particle Level Set (PLS) method adds Lagrangian particles both inside (blue points) and outside (red points) the level set interface (blue line) to help reduce diffusion of small-scale structure (see [17] for details). The grid shows the resolution of the Eulerian simulation, emphasizing the number of additional samples required to implement PLS, especially for complex surfaces. Image from [57].

also makes diffusion or smearing of small-scale structures a recurrent problem, somewhat mitigated but not eliminated by approaches such as PLS and vorticity confinement [20].

Further difficulties are brought about by the spatial discretization which requires complex codes such as ENO [27, 28] and WENO [32, 35] in order to reach beyond basic first order accuracy. High-order schemes are particularly necessary for Eulerian approaches as level sets cannot preserve surface features under even simple transformations such as translations and rotations. Large-kernel spatial discretizations schemes add further complexity when combined with the non-periodic boundary conditions generally desirable in computer graphics.

Level sets also tend to depart from signed distance over time, requir-

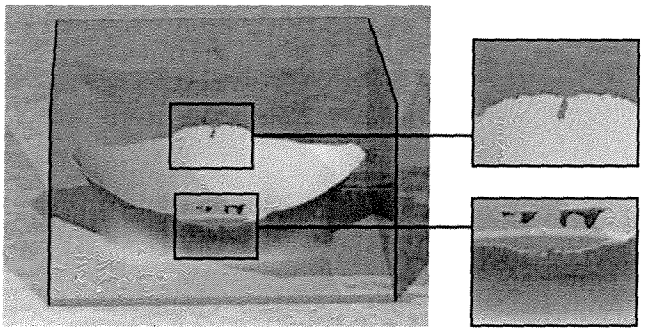

Figure 2.2: This image from Losasso et al. [38] shows unexpected mass loss (top right) as well as undesirable voxelization at the limits of the octree (top right and bottom right).

ing a reinitialization procedure every few time steps to maintain favourable numerical qualities [11].

Clearly there is a price to be paid in the simulation when selecting Eulerian approaches for their simplicity in rendering. We will now examine the Lagrangian alternative intended for use with our surfacing algorithm.

2.2 Lagrangian Approaches

Lagrangian simulations track fluid properties at points or *particles* that move with the fluid rather than at fixed points in space as with Eulerian systems. There are many variations on the Lagrangian approach including Smoothed Particle Hydrodynamics (SPH [41]), Particle-in-Cell and FLuid-Implicit-Particle (PIC and FLIP [60]), marker particle methods [26] as well as various adaptive variations [1, 31]. See Bridson [5] for a comprehensive review of Lagrangian fluid simulations.

Lagrangian approaches share a number of benefits over Eulerian ap-

proaches in terms of simplicity and correctness. For example, SPH does not involve a discretization on a spatial grid, allowing for large or irregularly shaped domains such as meandering rivers. Lagrangian approaches generally reduce or eliminate the need for complex discretizations in space such as ENO and WENO. Lagrangian approaches generally have little or no problem with mass preservation or spatial diffusion. Indeed the utility of particle systems is evidenced by their frequent employment to correct or supplement non-particle-based codes ([10, 17, 37], see also figure 2.3).

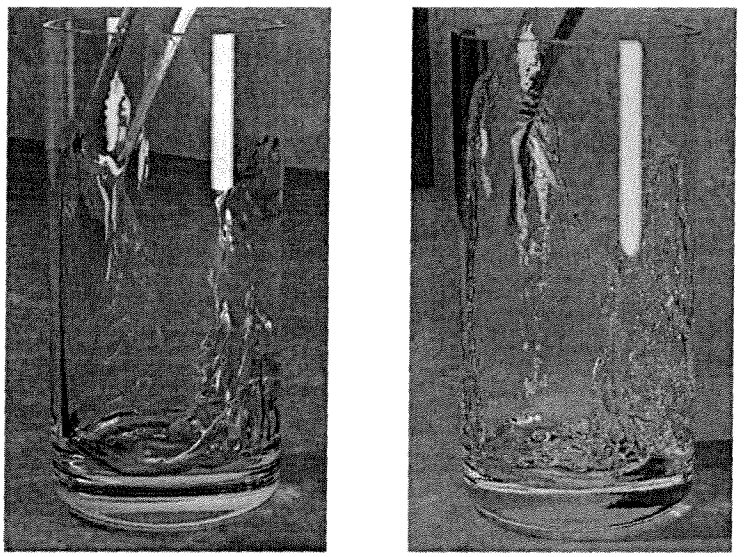

Figure 2.3: (Left) An early grid-based paper [17] demonstrates reasonable quality in capturing relatively thin fluid structures. (Right) Six years later an update [37] by some of the same authors resolves finer scales only by coupling the grid-based solver with a particle-based solver.

Leveraging particle systems directly in a fluid simulation requires an approach to constructing a high-quality surface mesh for rendering. Although a wealth of literature considers particle-based simulations surprisingly little

attention has been devoted to surface reconstruction.

2.3 Hybrids and Alternatives

In practice, modern approaches to fluid simulation and rendering in computer graphics tend to blur the line between Eulerian and Lagrangian. The approaches are combined in the hybrid FLIP codes of Zhu et al. [60], while Losasso et al. [37] bidirectionally couples independent Eulerian simulations for liquids with Lagrangian simulations for less-dense foam and spray. Different approaches altogether include fluid-oriented Finite Element Models (FEM [2]), boundary-only element dynamics [10], wave particles [59], Lattice Boltzmann techniques [54] and procedural approaches [6]. Our surfacing algorithm may in general be used with any fluid simulation that expresses the final location of the fluid as a collection of discrete locations.

2.4 Previous Work

One of the earliest papers on particle surface reconstruction comes from Blinn [4] which outlines an approach commonly referred to as *blobbies*. From the particle locations Blinn builds an additive scalar potential field which is subsequently isocontoured to produce the final surface. The method was designed for, and is well suited to, the task of visualizing molecular densities. However, when applied to a large number of macroscopic fluid particles, the resulting surface is quite naturally very blobby. The blobbiness highlights the particle-oriented nature of the underlying simulation rather than masking it as desired.

Another early approach is *surface splatting* [43, 61] whereby billboards (textures orthogonal to the axis of the camera) are rendered at the particle locations. While straightforward to implement the approach requires an exorbitantly high particle count to avoid the fluid visually breaking up at the edges [48]. Further, the cost of the method increases with frame resolution.

More recently, Zhu et al. [60] construct a scalar field like that used with blobbies but isosurface a *moving average* on the field. This approach smooths out the contribution of each particle to a given surface vertex. Unfortunately, the discretization also results in the production of spurious blobs. These blobs can be eliminated through subsequent upsampling and smoothing but at the expense of small surface detail (figure 2.4).

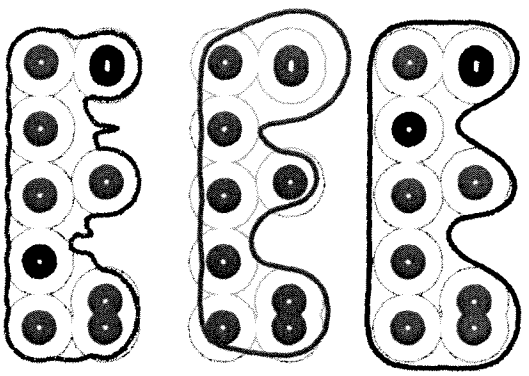

Figure 2.4: Particle surfacing comparisons in 2D. (Left) Zhu et al. [60] generates spurious blobs in concave regions and relatively poor approximations of flat surfaces, both of which require unconstrained smoothing at the expense of genuine features. (Middle) The refinement offered by Adams et al. [1] produces somewhat flatter surfaces but is adversely affected when particle distribution deviates from the average such as in the concentration of particles at the upper right. (Right) Our surfacing algorithm generates perfectly flat surfaces, smooth convex and concave surfaces, and is not adversely affected by the particle distribution.

The approach of Zhu et al. was adapted by Adams et al. [1] to track particle-surface distance for each particle. This was necessary for their adaptively-sized particle simulations and has the nice side effect of further smoothing and improving the surface. However, as with any kernel-based approach no long-range smoothing is possible, which makes it difficult to represent truly flat surfaces. Furthermore the Zhu et al. and Adams et al. approaches both rely on a final particle-unaware smoothing step that cannot distinguish genuine small-scale structure from noise. Consequently, both approaches diffuse small-scale structure during surface extraction (just as Eulerian approaches diffuse small-scale structure during simulation).

Premože et al. [47] constructs and isosurfaces a scalar field that is advected by particle-driven forces. Although the approach is an order of magnitude slower than those cited above and was illustrated using high resolution grids (larger than 200^3), their results show noticeable particle artifacts. The authors conclude their method "has some inherent problems with creating sharp boundaries when the fluid is in contact with a solid object or another fluid" and suggest they have abandoned the approach.

Chapter 3

Particle Surfacing

We propose constructing surfaces around particles which are as smooth as possible while remaining *faithful* to the particle locations in a technical sense we define in section 3.4.1. We have chosen to formalize this approach as a problem of constrained nonlinear optimization. A quadratic objective function measuring surface energy is used to maximize smoothness. To keep the surface faithful to the underlying particles, we enforce a constraint during the optimization stage that restricts each surface vertex to lie outside the union of radius r_{inner} particle-centered spheres and inside the union of radius r_{outer} particle-centered spheres (see figure 3.1). Our algorithm generates a compliant initial surface using Marching Tiles at r_{outer} (chapter 4) and optimizes the surface using a sparse iterative Gauss-Seidel solver [29]. The algorithm is outlined in figure 3.2 and described in detail below.

3.1 Smoothness

In order to pose a constrained optimization problem we require a scalar objective function measuring surface smoothness. We select here the thin-plate energy of the surface i.e. the integral of the surface Laplacian squared. Following [14] we consider a discrete approximation to the Laplacian about

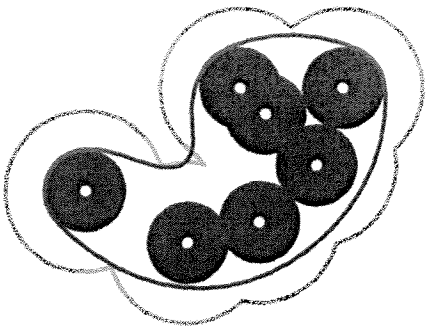

Figure 3.1: The key idea in our surface definition is constructing a surface as smooth as possible that remains outside the union of radius r_{inner} particle-centered spheres and inside the union of radius r_{outer} particle-centered spheres. The resulting surface captures all true features of the fluid without introducing spurious features in the interpretation. We enforce the constraint on mesh vertices but not mesh edges, explaining the small violation visible above.

vertex i:

$$(\nabla^2 x)_i \approx \frac{1}{D_{ii}} \sum_j W_{ij} x_j \tag{3.1}$$

where D is a diagonal normalization matrix and W a zero row-sum weighting matrix. All matrices encoding the surface graph connectivity including D and W are sparse.

We will use two quadratic objective functions, $x^T Ax$ and $x^T Bx$ where A represents an unweighted approximation to the graph Laplacian and B a weighted approximation to the surface Bilaplacian[1].

[1] Also known as the biharmonic operator

Surfacing Algorithm
Calculate distance field ϕ from particles P on a background grid.
Generate initial x_i by Marching Tiles for isocontour $\phi = r_{outer}$.
Find unweighted graph Laplacian matrix A for x_i.
Run five steps of Gauss-Seidel on x_i using $x^T Ax$ with particle constraints.
Find integrated Bilaplacian matrix B given current x_i.
Run 25 steps of Gauss-Seidel on x_i using $x^T Bx$ with particle constraints.
Run five steps of Gauss-Seidel on x_i using $x^T Bx$ without particle constraints.
Optionally 'erode' x_i back along the surface normal.

Figure 3.2: Outline of the surface reconstruction algorithm.

3.2 Unweighted Graph Laplacian

For the unweighted Laplacian case, we set W_{ij} equal to negative one for each 1-ring neighbour j of i, and W_{ii} to the valence of vertex x_i as in Taubin [53]. We don't use the weighting matrix D, instead forming the discrete quadratic objective $x^T Ax$ with $A = W$. Although we form A explicitly, it is straightforward to apply the operator directly to structure of the mesh. Optimization with this objective is often used in mesh generation, where it is called 'mesh smoothing', as it generally leads to improved triangle shapes. In the absence of constraints, a minimizing solution would satisfy $Ax = 0$, which can be interpreted as requiring that a vertex be at the centroid of its neighbours.

3.3 Weighted Surface Bilaplacian

For the weighted Bilaplacian case, following Desbrun et al. [14] we set W_{ij} to the sum of the cotangents of opposite angles from the pair of triangles

incident on vertices x_i and x_j. D_{ii} is set to the sum of the area of all triangles incident on vertex x_i. Our approximation to the integral of the Laplacian squared is therefore:

$$\begin{aligned} \iint_S \|\nabla^2 x\|^2 &\approx \sum_i (D^{-1}Wx)_i^2 D_{ii} \\ &= x^T((D^{-1}W)^T D D^{-1}W)x \\ &= x^T(W^T D^{-1}W)x \end{aligned} \tag{3.2}$$

To make use of the Bilaplacian in our optimization problem we wish to converge on a mesh where the gradient is zero (i.e. a stable point). Taking the gradient of equation (3.2) gives

$$2(W^T D^{-1}W)x \tag{3.3}$$

and so our quadratic approximation of the Bilaplacian is $B = W^T D^{-1}W$. Note that B is symmetric positive semi-definite which guarantees convergence for Gauss-Seidel, although this is not actually an issue for us as we iterate with a fixed upper bound (see section 3.8).

3.4 Constraints

3.4.1 Particle Faithfulness

We consider a surface vertex x_i to be *faithful* if it lies within a bounded distance of the nearest particle P_j

$$r_{inner} \leq \min_i \|x - P_i\| \leq r_{outer} \tag{3.4}$$

A faithful surface is one that lies outside the union of spheres of radius r_{inner} and inside the union of spheres of radius r_{outer} (figure 3.1). Heuristically, this forces the surface to never vary much from the distribution of the particles, while allowing it enough freedom to smooth over individual particles and thus capture smooth and flat surfaces when appropriate.

The surface is kept faithful to the particle locations by snapping surface vertices to within $[r_{inner}, r_{outer}]$ of the closest particle after each iteration of the solver. We employ a simple sparse spatial hash on particle locations to accelerate the particle lookup.

Enforcing constraints represents approximately 10% of the cost of the entire surfacing procedure per frame. This cost can be reduced by skipping the enforcement every second or third step or employing a more refined space-partitioning structure such as a kd-tree. The remaining costs in the surfacing algorithm are 10% for the smoothing (section 3.1) and 80% in generating the initial conforming isosurface (chapter 4).

Note that we require only the surface vertices to be faithful, not the edges (figure 3.1). We investigated enforcing faithfulness on edges as well but did

not detect an improvement in surface quality that justified the added cost.

3.4.2 Solid-Liquid-Air Intersections

Under gravity, fluids tend to rest heavily and uniformly against the bottom and sides of a container, or boundary. To ensure our algorithm generates a surface that does not cross boundaries we snap surface vertices outside the boundaries to the nearest point on the boundary after each solver iteration. The result is a fluid surface that aligns precisely with the container walls (see figure 3.3).

If rendering the container and an opaque liquid, one may choose to move the liquid slightly further inwards to prevent rendering artifacts such as 'Z-fighting' that might be caused when two surfaces are rendered at the same location (even if the normals point in opposite directions). This simple approach may be appropriate if the camera positioning won't unduly reveal the 'gap' (as in figure 3.4), although vertex manipulation of this kind may result in non-manifold surfaces (section 3.9).

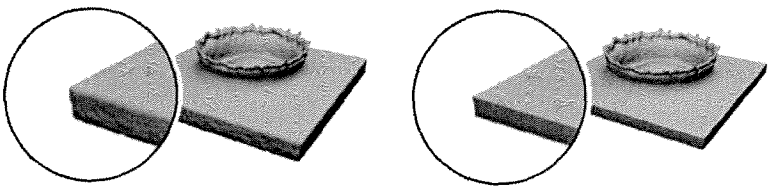

Figure 3.3: Handling fluid-boundary intersections. (Left) The initial mesh may violate boundary conditions such as an enclosing square dish. (Right) By snapping each violating surface vertex to the nearest boundary after creation of the initial mesh and each smoothing step our final mesh conforms to the boundary exactly.

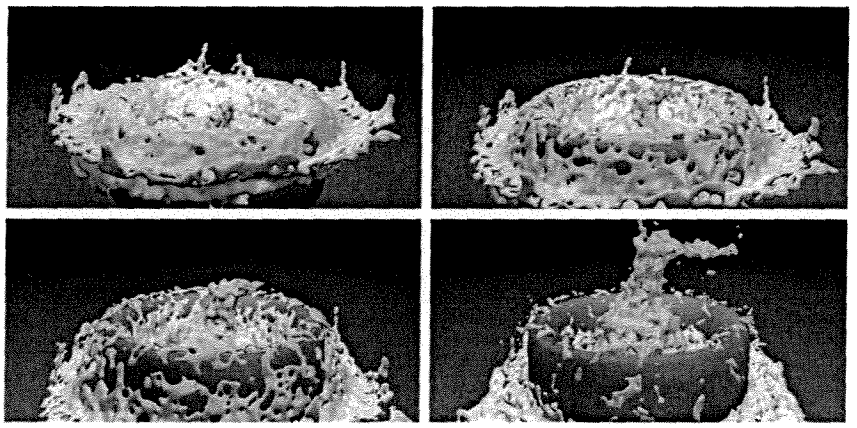

Figure 3.4: Surfacing with a complex boundary. The clay dish was encoded as a level set using geometric boolean operations on a plane, sphere and cylinder. During surfacing the level set representation provided inside-outside tests which were combined with distance-to-primitive measurements to prevent the final fluid surface from penetrating the dish.

For transparent liquids, and to better support arbitrary camera positions, a more accurate rendering can be achieved by explicitly identifying the surfaces representing each of the three surface pairs (fluid-air, fluid-boundary and boundary-air) and rendering each pair individually. This approach was used to render the fish tank in figure 3.5 in which the correct dielectric ratio [46] was applied to each surface pair.

To isolate the surface pairs we first consider the fluid-air surface, which is easily obtained by intersecting the surface mesh with four inward-facing planes coincident with the inner sides of the tank (see [16] for the relevant triangle-plane intersection test) and accepting all fragments on the inside of all four planes. To obtain the fluid-boundary surface we intersect the mesh with each plane in turn and collect the resulting linear fragments into closed curves outlining the location of the fluid as it lies against the tank

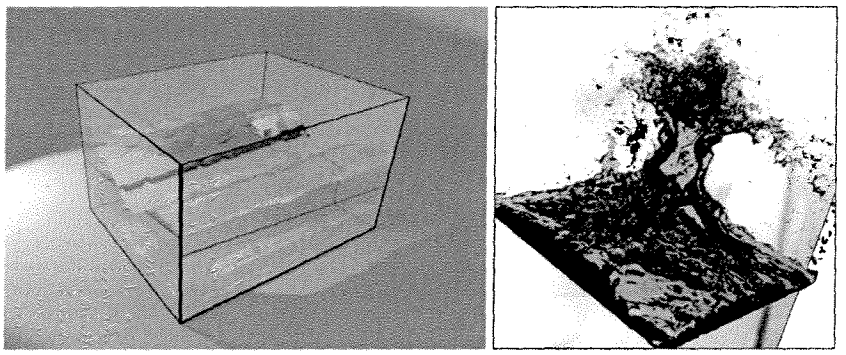

Figure 3.5: (Left) Isolation of fluid-air-boundary surface pairs. Through intersection and triangulation procedures the fluid-air, fluid-boundary and boundary-air surfaces were isolated, allowing interface-specific rendering parameters such as dielectric ratios to be applied to each surface pair. (Right) Another rendering of a transparent fluid surfaced using our algorithm.

wall. We triangulate these curves about an interior point [50] to obtain the boundary-fluid surface. We triangulate about an exterior point in a domain bounded by the inner tank wall to obtain the boundary-air surface.

Identifying surface-pairs for non-planar boundaries would follow the same procedure but substitute triangulation with, for example, NURBS[2] trimming.

3.5 Gauss-Seidel Solver

We used a sparse implementation of the Gauss-Seidel iterative solver to condition and smooth our surface mesh. The (nonlinear) constraints could not be encoded directly into the problem statement so we enforced them explicitly on the surface mesh after each step of the solver.

[2]Non-Uniform Rational B-Spline

3.6 Barzilai-Borwein Solver

As a potentially faster alternative to Gauss-Seidel we investigated the interesting Barzilai-Borwein solver [3] and implemented several variations including Cyclic Barzilai-Borwein [13, 22], Preconditioned BB [40] and Alternating Step [12].

We found the BB variants to be at least competitive with Gauss-Seidel in generic convergence tests. However, the distinguishing characteristic of the Barzilai-Borwein family of techniques is their *nonmonotonic* behaviour [9, 25]. Numerical experiments by Fletcher [21] indicate the range of step lengths can vary in magnitude by as much as 10^5, a fact noted as "disconcerting" (see figure 3.6). When applied to our surfacing algorithm the occasionally large step lengths of these solvers caused our surface vertices to temporarily move close to distant particles, which then 'captured' them by virtue of the constraints and thereby destroyed the manifold property of the mesh. The highly nonlinear nature of our constraints unfortunately seems to mandate the use of a slow-and-steady iterative solver such as Gauss-Seidel. See also the discussion on damping in section 3.8.

3.7 Radii

We have not described how r_{inner} and r_{outer} are actually determined. r_{inner} is typically set to the radius used in the fluid simulation (e.g. SPH, PIC, FLIP) or the average inter-particle spacing where the simulation method does not support an explicit radii (e.g. some procedural and ad-hoc methods). r_{outer} must be chosen large enough to fill air pockets in fluid, but no so large that

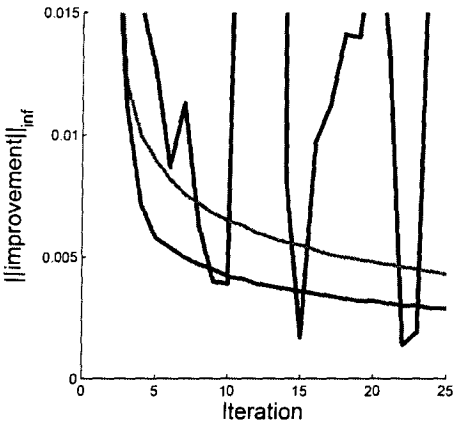

Figure 3.6: A comparison of step lengths for Jacobi (red), Gauss-Seidel (green) and Barzilai-Borwein (blue) in a generic convergence test. Although Barzilai-Borwein is seen to converge faster, the tremendous range of observed step lengths prevented us from effectively enforcing our constraints with the BB solver. Image from [56].

it obscures surface detail. By default we strike a balance at $r_{outer} = 2r_{inner}$.

An important drawback to existing local approaches such as Zhu et al. [60] and Adams et al. [1] is the inability to generate very flat surfaces. These are especially important for still and slow-moving liquids. We point out that given any r_{inner} and considering surfacing particles arranged on square grid of size d our algorithm will construct a planar surface if the following condition on r_{outer} is met

$$r_{outer}^2 > \frac{d^2}{2} + r_{inner}^2 \tag{3.5}$$

This follows setting the intersection forming the 'spikes' at the outer surface to the inner radius, thereby allowing a planar surface to lie between each. The finding is illustrated in figure 3.7 which depicts the inner and out

spheres when the difference in equation (3.5) is some small ϵ. Similarly for a triangular grid where the distance between neighours is d we require only

$$r_{outer}^2 > \frac{d^2}{3} + r_{inner}^2 \tag{3.6}$$

We contrast this result with Adams et al. [1] where a perfectly flat surface is only reachable in the limit as the particle density becomes infinite.

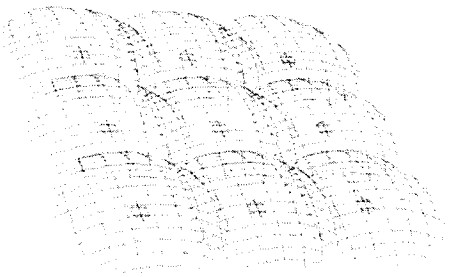

Figure 3.7: The r_{inner} particle-centered spheres (green) are just visible at the point where the r_{outer} particle-centered spheres touch all neighbouring r_{outer} spheres under the condition that $r_{outer}^2 = \frac{d^2}{2} + r_{inner}^2$, when the particles are arranged in a square lattice. When $r_{outer}^2 \geq \frac{d^2}{2} + r_{inner}^2$ a surface constructed with our algorithm will be planar as desired.

3.8 Algorithm

We now have all the pieces in place (objective functions, constraints and solver) to describe our full algorithm.

As in any optimization problem the first step is determining an initial guess that satisfies the constraints. We use Marching Tiles (chapter 4) to generate compliant surface meshes with good valences that allow for improved smoothing, although any isosurfacing algorithm will do.

We select the r_{outer} isosurface as our initial guess, as opposed to, for example, r_{inner} or $\frac{1}{2}(r_{inner} + r_{outer})$ as the final surface was often found to be closer to r_{outer} than r_{inner}. Also, the surface at r_{inner} frequently contains deep ingresses which demand a prohibitive amount of smoothing to work the surface up to a point where all the constraints come into effect. These cavities also create clusters of unnecessarily close vertices in the final mesh (which smooths them over). These problems cannot appear at the r_{outer} isosurface as the surface is stretched uniformly over the r_{inner} spheres as the smoothing progresses. Furthermore, we observe that vertices created at r_{outer} already lie within roughly $r_{outer} - r_{inner}$ of their final location, so significant localized clustering does not occur.

When surfacing, we select a tile size that results in an average inter-sample distance of $h = .8r_{inner}$. We observe that fewer samples do not allow the resulting surface to be smoothed in areas of high curvature and additional samples produce unnecessary tessellation that slows down the smoothing process. Further tessellation, if desired, is better performed later on using subdivision.

We perform a preliminary normalization of triangle areas on the sampled mesh by running five steps of Gauss-Seidel using $x^T Ax$ with constraints enabled as described in section 3.4. This ensures that each vertex has no degenerate triangles that would lead to a near-undefined B (which divides by the sum of areas of adjoining triangles in its construction, see section 3.3). We use an under-relaxation parameter of 0.5 during the normalization stage to reduce the possibility of creating degenerate triangles which would otherwise destroy the manifold property of the mesh.

Next we smooth the mesh with 25 steps of Gauss-Seidel with $x^T Bx$ and constraints enabled. A finishing polish is then applied through five additional iterations with $x^T Bx$ without constraints. We observe that the final polish improves the smoothness of the surface normals without altering the location of the surface significantly, and is very inexpensive to perform.

The iteration counts were selected, after limited experimentation, for giving consistently good results, although they may of course be tweaked for a desired quality/cost trade-off. More sophisticated convergence criteria are unnecessary due to the selection of the r_{outer} isosurface initial condition, which permits roughly the same amount of high-frequency 'noise' regardless of the particle distribution, thereby requiring roughly the same number of Gauss-Seidel iterations.

Finally we provide for an optional post-smoothing *surface erosion* to tighten and sharpen mesh features.

3.9 Surface Erosion

Just as Eulerian approaches are limited in the dimensions of the rendered features to the grid cell size, our algorithm supports flat contiguous structures only to a minimum thickness of r_{inner}. Reducing r_{inner} to support thinner structures necessitates simulating additional particles in order to maintain surface continuity. Furthermore, if $r_{outer} \gg r_{inner}$ a considerable number of Gauss-Seidel iterations may be required for the initial surface at r_{outer} to 'find' all the relevant r_{inner} particle constraints.

We therefore support *surface erosion* as an optional final step, which

generates very thin structures without simulating very large numbers of particles or requiring excessive smoothing. Surface erosion is the process of stepping all the surface vertices back along the negative surface normal by δr_{inner} for some small $0 < \delta \ll 1$ (see figure 3.8). Note that while this technique is both fast and easy to implement, there is a risk of creating degenerate triangles in the final mesh which may in turn cause rendering artifacts. Furthermore, features with diameter less than δ will actually be thickened by the erosion. These problems can be overcome by detecting triangle intersections across imaginary time $(0, \delta]$ and reducing δ and iterating as required.

It is interesting to contrast our surface thinning with the deliberate thickening stage intended to prevent mass loss of Chentznez et al. [10].

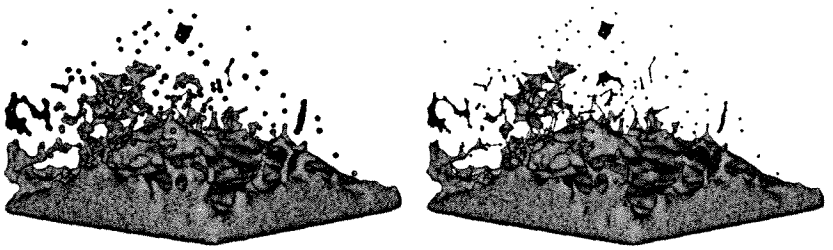

Figure 3.8: (Left) Surface mesh without erosion. (Right) Surface mesh with erosion where $\delta = \frac{1}{2} r_{inner}$. While features are thinned and sharpened as desired the resulting mesh is in general no longer manifold without additional processing.

Chapter 4

Marching Tiles

Isosurfacing is the process of extracting a surface of constant value from a regular scalar field. The classic approach to isosurfacing is Marching Cubes (MC [36]).

4.1 Marching Cubes

Classic MC operates by considering each composite cubical volume of the field in turn and categorizing each according to whether the values at each of the eight corners are above or below (and therefore lie outside or inside) the isosurface value. There are therefore 2^8 possible configurations for each cube, reducible to 15 by taking advantage of rotational and reflective symmetries. Each of the 15 cases is assigned a particular configuration of triangles representing the interpolating surface (figure 4.1). The triangle vertices are linearly interpolated between values at the corners. The isosurface is simply the union of all such triangles.

As pointed out by Durst [15], MC can produce non-manifold surfaces due to ambiguities introduced by assigning fixed surface configurations. Consequently there are variations that break the cubical cells into tetrahedra [42], consider each face of the cube in turn [30], superimpose cubes in a body-

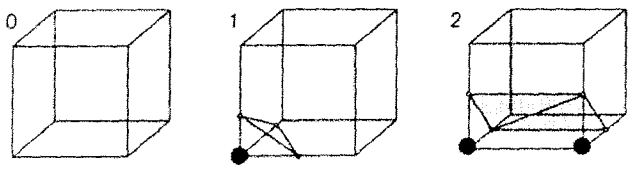

Figure 4.1: Several Marching Cubes cell configurations [36].

centered cubic (BCC) lattice [8, 33], or consider the dual of MC [44]. However, an essentially cubical lattice is common to each of these approaches, and consequently each incorporate directly or indirectly composite tetrahedra with dihedral angles greater or equal to 90°. This allows for the possibility of valences as low as four, which, as seen in figure 4.2, can be substantial in number.

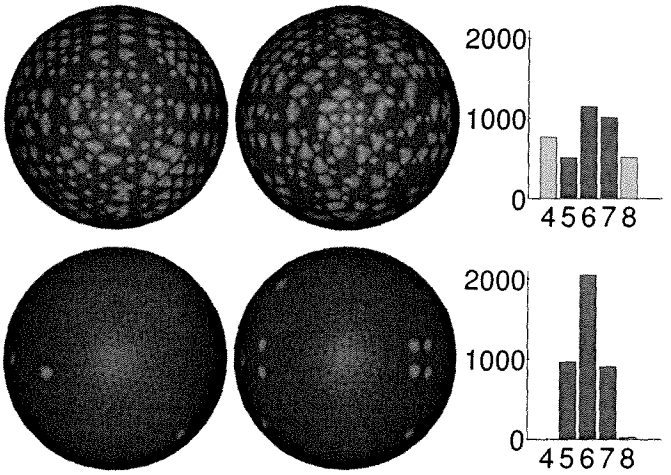

Figure 4.2: Marching Tiles valence histogram comparison. (Top) The front and back of a sphere tessellated with Marching Cubes along with the associated valence histogram. Vertices with neighbor counts outside the range $[5, 6, 7]$ are shaded lighter. (Bottom) Our solution, with the majority of the vertices having the ideal valence of six, and none below five.

For many purposes, valence four vertices represent a poor sampling of the surface at that vertex. In our surfacing algorithm, valence four nodes are seen to oscillate during smoothing with our weighted Bilaplacian from equation (3.2) and often form unwanted pockmarks in the final surface. Valence four vertices also cause smoothness problems for many subdivision schemes, and hinder convergence for FEM and related methods.

4.2 Marching Tiles

Our secondary contribution is a generalization of Marching Cubes to arbitrary space-filling tiles, and one tile in particular that ensures a minimum valence of five and is amenable to spatial hashing. The resulting isosurfacing algorithm provides a better minimum sampling of the surface than Marching Cubes, resulting in better behaved numerical operators such as smoothing. To generalize Marching Cubes we must first identify non-cubical space-filling tiles.

Space-filling tiles have been investigated in the context of minimal surface energy configurations in physical phenomena such as foam and crystals [52]. For any given tile, valence four vertices occur only when the isosurface crosses an edge incident on four tetrahedra. Valence four vertices can therefore be eliminated by requiring that at least five tetrahedra are incident on every edge. This is the case when all tetrahedra have acute (less than 90°) dihedral angles, as at least five tetrahedra must therefore share every edge.

Üngör [55] recently investigated tilings with acute tetrahedra. There are

at least several infinite sets of such tiles [24] including tetrahedrally close-packed (TCP) cells so-named because the surface of the Voronoi cell of each consists of only pentagonal and hexagonal faces [52]. One of the four basic TCP cells is designated A15 and happens to encode the atomic structure of Chromium Silicide (Cr_3Si). We selected A15 for use with our Marching Tiles algorithm as it involves only integral values in its construction. However as A15 is somewhat irregular we shuffled around tetrahedra to arrive at a modified roughly cubical A15 that is more amenable to (regular) spatial hashes and as well reduces the size of the enclosing domain (therefore requiring fewer covering tiles). The final modified tile is shown in figure 4.3. Numerical values for this tile may be found in appendix A.

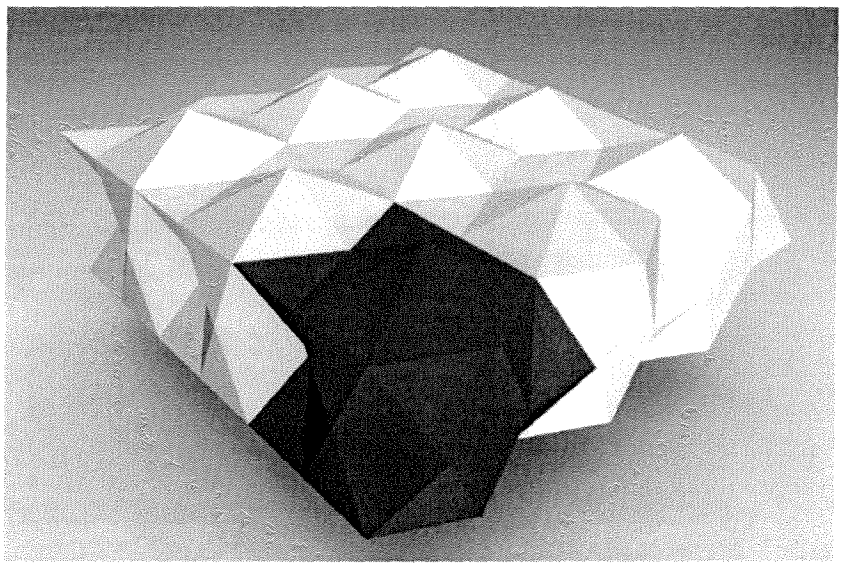

Figure 4.3: The modified A15 space-filling tile used with Marching Tiles to construct an initial conforming mesh for the particle surfacing algorithm.

We note that the tile designated C15 has even lower minimum dihedral angles than A15. However we did not pursue this tiling as its construction is somewhat more complicated, and further valence five vertices do not present nearly as serious difficulties to our Bilaplacian smoothing as valence four vertices. However a tile based on C15 or some other tile with a lower maximum dihedral angle may provide significant additional benefit in conjunction with other numerical operators.

One might ask if we can eliminate valence five vertices altogether. Unfortunately this is not possible as the average valence for any manifold mesh is six (appendix B) and it is trivial to show a perfect valence-six graph does not exist in general. Once valence four vertices have been eliminated only incremental improvements to mesh valence distributions are possible.

Chapter 5

Results

5.1 Enright Deformation Test

A problem common to all fluid representations is maintaining a continuous surface for thin sheets formed, for example, when water splashes against a smooth surface. Enright et al. [17] considered several such reference cases including a deformation commonly known as the Enright Test. The test deforms a sphere of radius .15 centered at $(.35, .35, .35)$ under a velocity field defined by LeVeque [34]

$$\begin{aligned} u(x,y,z) &= 2sin^2(\pi x)sin(2\pi y)sin(2\pi z) \\ v(x,y,z) &= -sin(2\pi x)sin^2(\pi y)sin(2\pi z) \\ w(x,y,z) &= -sin(2\pi x)sin(2\pi y)sin^2(\pi z) \end{aligned} \tag{5.1}$$

The velocity is reversed at $t = .5T$, so the sphere should be exactly restored at time T. The difficulty in performing the test lies in maintaining a continuous surface as the sphere is flattened in the Z axis. The Eulerian approach must also use a high-order spatial discretization scheme to prevent the sphere deforming during the integration. Figure 5.1 shows several published level set results alongside results obtained using our surfacing algorithm.

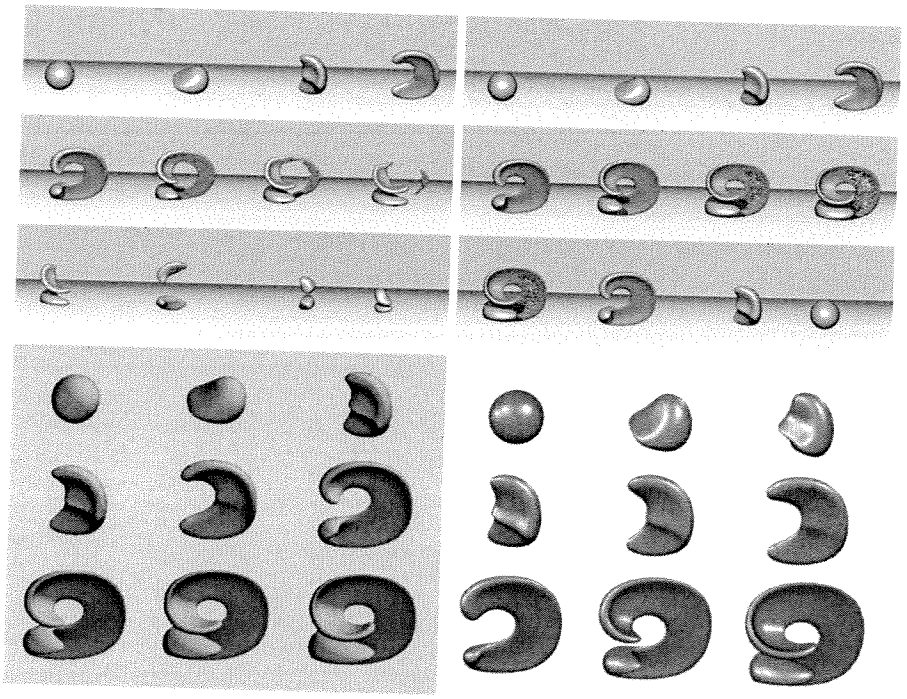

Figure 5.1: (Top Left) On a 100^3 grid plain level sets "fail severely" [17]. (Top Right) The PLS technique introduced in [17] better resolves the surface but does not maintain surface continuity. The top images also demonstrate the inability of the spatial discretization to accurately restore the sphere at time T. (Bottom Left) In [18] Enright et al. first demonstrated continuity using level sets by leveraging the additional sampling required by PLS and adaptive octree codes. (Bottom Right) Our surfacing algorithm demonstrates continuity with only the 100^3 samples used in the plain level set test (top left) and without using any complex numerical machinery.

5.2 Fluid Simulations

We applied our surfacing algorithm to particle data from a number of actual fluid simulations generated with the FLIP codes of Zhu et al. [60].

The sequence in figure 5.2 shows an early rendering demonstrating flat and smooth surfaces. The simulation required one million particles and was surfaced on a grid equivalent in size to 333^3. The sequence in figure 5.3 shows an oily jet of fluid undergoing complex topological changes and required at most 480,000 particles per frame and a 110^3 grid for surfacing. See [58] for further details on how these images were generated.

Although our algorithm renders all isolated particles as identical spheres we did not find the sameness to be distracting as long as the particle resolution was relatively high ($\gtrapprox$ 50K particles). A physically-based drop oscillation model similar to that used in [23] might be usefully employed to further the realism of the dynamics of fluid spray.

All code was developed and run on a multi-core Intel® processor including fluid simulations and rendering. Images were created using NVIDIA® Gelato®, a Python/C programmable GPU-accelerated off-line renderer. Surfacing was performed on up to four frames in parallel.

Our fluid surfacing algorithm has been incorporated into Double Negative's production fluid simulator SQUIRT and applied to shots in 10,000 B.C. (figure 5.4), HELLBOY II (figure 5.5) and other feature films currently in development. The ability to surface frames in parallel permitted a drop in turn-around time from hours to minutes compared to off-the-shelf surfacing codes.

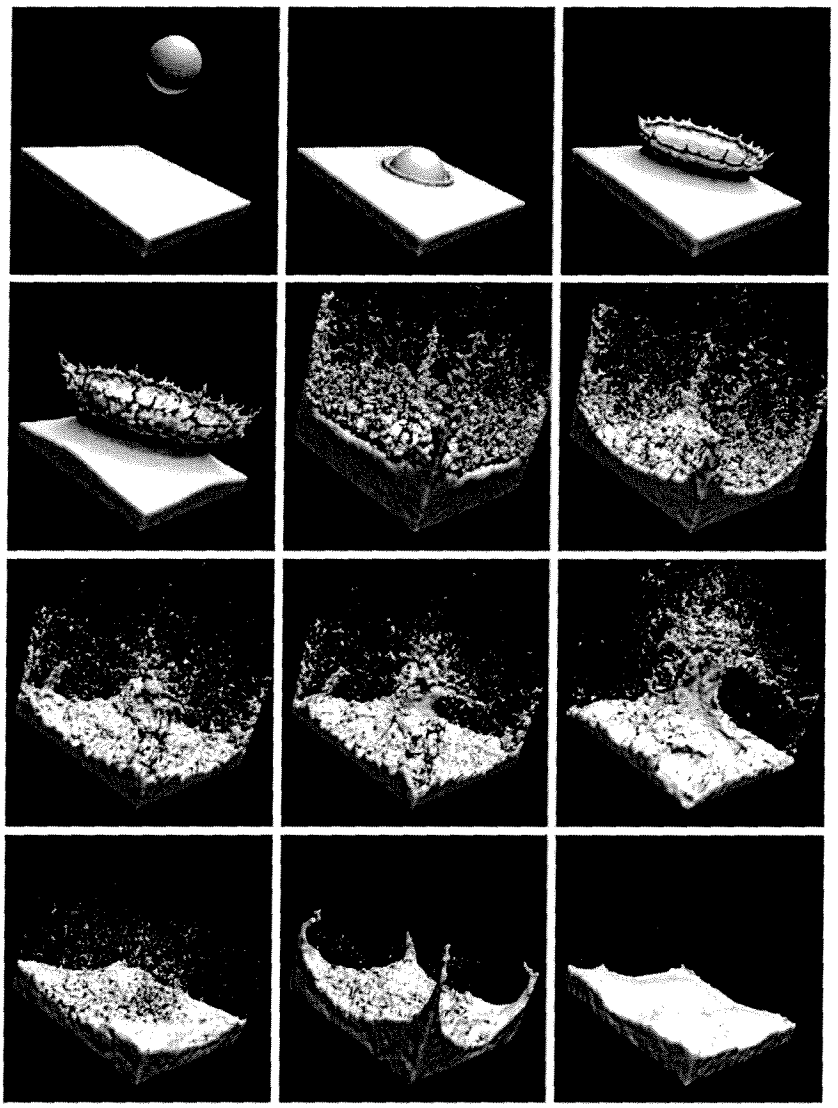

Figure 5.2: Sequence from a test fluid simulation. The top left image shows a flat initial surface unobtainable using previous approaches. Note the thin sheets of fluid generated by the initial splash and varied topologies as the liquid transitions from sheets to tendrils to droplets.

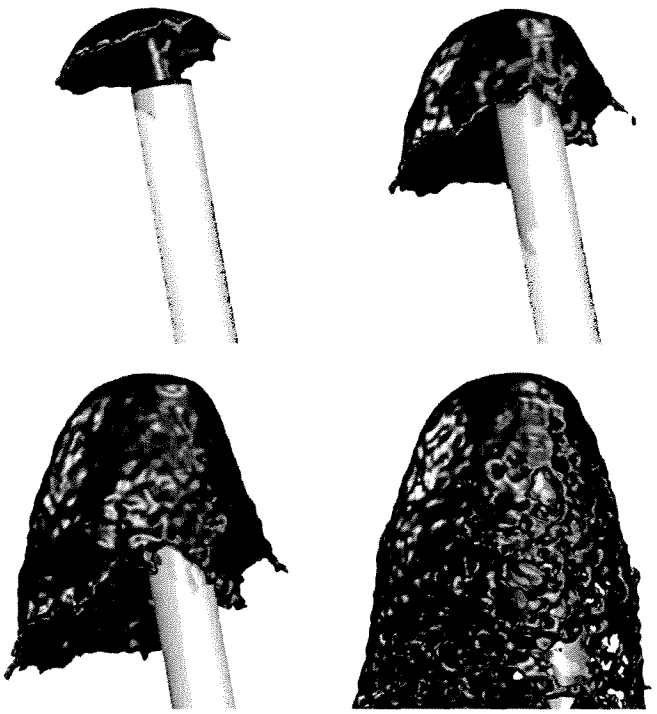

Figure 5.3: A viscous oily fluid spilling from a tube.

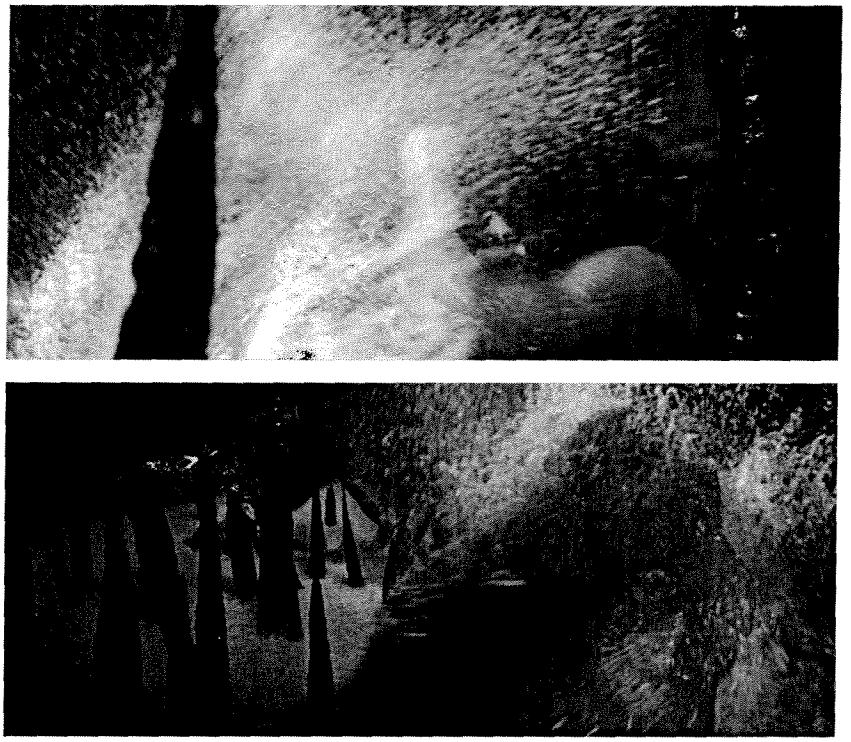

Figure 5.4: Scenes from 10,000 B.C. incorporating fluid surfaced using our algorithm. © Warner Bros.

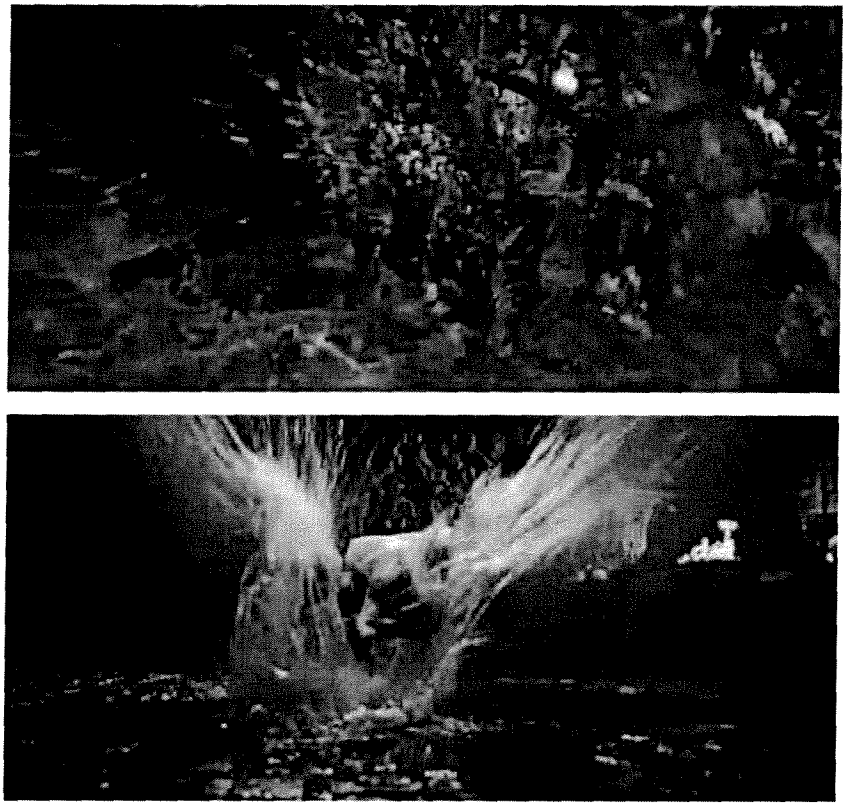

Figure 5.5: Scenes from HELLBOY II incorporating fluid surfaced using our algorithm. © Universal Studios

Chapter 6

Conclusion

A new algorithm for surfacing particle fields was presented. The surface generated is, by design, globally smooth and strictly adheres to the particle locations, properties lacking in existing approaches. Our algorithm is treats recent advances in mesh smoothing within the context of a iterative solver, permitting trade-off between processing time and final surface quality.

Our fluid surfacing algorithm generates isosurfaces with Marching Tiles, which produces meshes with superior valence distributions over those of Marching Cubes and derivatives. By substituting the cubical lattice with a general space-filling tile consisting of tetrahedrons with acute dihedral angles we are able to guarantee that at least five tetrahedrons share every edge of our tile. The resulting surface vertices have a minimum valence of five which permits an improved worst-case sampling of the underlying surface, and consequently better-behaved numerical operators such as the Bilaplacian.

6.1 Future Work

The problem of temporal coherence might usefully be addressed by replacing surface meshes with level sets. This could be accomplished by discretizing

equations (3.2) using finite differences, for example, and boosting our surface constraint equation (3.4) across $\phi(x, y, z)$

$$d(x, y, z) - r_{outer} \leq \phi(x, y, z) \leq d(x, y, z) - r_{inner} \tag{6.1}$$

Level sets might also offer a way forward in obtaining a time-independent surface parameterization to better support texture-based surface detail such as foam and floating particulates [49].

Marching Tiles could be further improved by refining the tile employed. We selected A15 over other tiles with acute dihedral angle tetrahedral decompositions primarily for the simplicity of working with only integral values in its construction. However an alternative might without additional runtime cost reduce the average number of valence five vertices, thereby improving (if incrementally) the performance of applied numerical operators.

The primary costs for our algorithm lie in initial surfacing (80%), and to a lesser extent, constraint enforcement (10%). Both costs might be reduced using the spatial acceleration structures described in Rosenberg and Birdwell [48].

Bibliography

[1] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. Adaptively sampled particle fluids. *ACM Trans. Graph.*, 26(3):48, 2007.

[2] Adam W. Bargteil, Chris Wojtan, Jessica K. Hodgins, and Greg Turk. A finite element method for animating large viscoplastic flow. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 16. ACM, 2007.

[3] Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

[4] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.

[5] Robert Bridson. *Fluid Simulation.* AK Peters, 2008.

[6] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. *ACM Trans. Graph.*, 26(3):46, 2007.

[7] Robert Bridson and Matthias Müller-Fischer. Fluid simulation course notes. In *ACM SIGGRAPH 2007 Courses*, pages 1–81, 2007.

[8] Hamish Carr, Thomas Theußl, and Torsten Möller. Modified marching octahedra for optimal regular meshes. In *SIGGRAPH '02: ACM*

SIGGRAPH 2002 conference abstracts and applications, pages 219–219. ACM, 2002.

[9] R.M. Chamberlain, M.J.D. Powell, C. Lemarechal, and H.C. Pedersen. The watchdog technique for forcing convergence in algorithms for constrained optimization. *Math. Program.*, 24:113–116, 1982.

[10] N. Chentanez, B. Feldman, F. Labelle, J. O'Brien, and J. Shewchuk. Liquid simulation on lattice-based tetrahedral meshes. In *ACM SIGGRAPH/Eurographics 2007 Symposium on Computer Animation*, pages 219–228, 2007.

[11] David L. Chopp. Computing minimal surfaces via level set curvature flow. *J. Comput. Phys.*, 106(1):77–91, 1993.

[12] Yu-Hong Dai. Alternate step gradient method. *Optimization*, 52(4–5):395–415, 2003.

[13] Yu Hong Dai, Williams W. Hager, Klaus Schittkowski, and Hongchao Zhang. The cyclic Barzilai-Borwein method for unconstrained optimization. *IMA Journal of Numerical Analysis*, 26(3):604–627, July 2006.

[14] M. Desbrun, M. Meyer, P. Schröder, and A. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324, 1999.

[15] M. J. Dürst. Additional reference to "marching cubes" (letters). *Comput. Graph*, 22(4):72–73, 1988.

[16] D. Eberly. *3D Game Engine Design.* Morgan Kaufmann, second edition, 2007.

[17] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comp. Phys.*, 183:83–116, 2002.

[18] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-lagrangian particle level set method. *Computers and Structures*, 83:479–490, 2005.

[19] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 21(3):736–744, 2002.

[20] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *Proc. ACM SIGGRAPH.* Association of Computing Machinery, 2001.

[21] R. Fletcher. On the Barzilai-Borwein method. Technical report, Department of Mathematics, University of Dundee, 2001.

[22] A. Friedlander, J. M. Martínez, B. Molina, and M. Raydan. Gradient method with retards and generalizations. *SIAM Journal on Numerical Analysis*, 36(1):275–289, 1999.

[23] Kshitiz Garg and Shree K. Nayar. Photorealistic rendering of rain streaks. *ACM Trans. Graph.*, 25(3):996–1002, 2006.

[24] M. Goldberg. Three infinite families of tetrahedral space-fillers. *J. Combin. Theory*, 16:348–354, 1974.

[25] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986.

[26] F. Harlow and J. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids*, 8:2182–2189, 1965.

[27] A Harten and S Osher. Uniformly high-order accurate nonoscillatory schemes. *SIAM Journal on Numerical Analysis*, 24(2):279–309, 1987.

[28] A. Harten, S. Osher, B. Engquist, and S. R. Chakravarthy. Some results on uniformly high-order accurate essentially nonoscillatory schemes. *Appl. Numer. Math.*, 2(3-5):347–378, 1986.

[29] Michael T. Heath. *Scientific Computing: An Introductory Survey.* McGraw-Hill, second edition, 2002.

[30] Chien-Chang Ho, Fu-Che Wu, Bing-Yu Chen, Yung-Yu Chuang, and Ming Ouhyoung. Cubical marching squares: adaptive feature preserving surface extraction from volume data. In *Proceedings of Eurographics 2005*, volume 24, pages 537–545. Eurographics, 2005.

[31] W. Hong, D. House, and J. Keyser. Adaptive particles for incompressible fluid simulation. Technical report, Texas A&M Computer Science, 2007.

[32] Guang-Shan Jiang and Chi-Wang Shu. Efficient implementation of weighted ENO schemes. *J. Comput. Phys.*, 126(1):202–228, 1996.

[33] Thomas Theußl, Torsten Möller, and Meister Eduard Gröller. Optimal regular volume sampling. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 91–98. IEEE Computer Society, 2001.

[34] R. LeVeque. High-resolution conservative algorithms for advection in incompressible flow. *SIAM J. Num. Anal.*, 33:627–665, 1996.

[35] S. Liu and T. Chan. Weighted essentially non-oscillatory schemes. *J. Comput. Phys.*, 115:200–212, 1994.

[36] William E. Lorensen and Harvey E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In *Proc. ACM SIGGRAPH*, volume 21, pages 163–169, 1987.

[37] F. Losasso, J. Talton, N. Kwatra, and R. Fedkiw. Two-way coupled SPH and particle level set fluid simulation. *IEEE TVCG*, 2008 (in print).

[38] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 457–462, New York, NY, USA, 2004. Association of Computing Machinery, ACM Press.

[39] Frank Losasso, Tamar Shinar, Andrew Selle, and Ron Fedkiw. Multiple interacting liquids. *ACM Trans. Graph.*, 25(3):812–819, 2006.

[40] Brigida Molina and Marcos Raydan. Preconditioned Barzilai-Borwein method for the numerical solution of partial differential equations. *Journal of Numerical Algorithms*, 13(1):45–60, 1996.

[41] J. J. Monaghan. Smoothed particle hydrodynamics. *Annu. Rev. Astron. Astrophys.*, 30:543–574, 1992.

[42] Heinrich Müller and Michael Wehle. Visualization of implicit surfaces using adaptive tetrahedrizations. In *DAGSTUHL '97: Proceedings of the Conference on Scientific Visualization*, page 243. IEEE, IEEE Computer Society, 1997.

[43] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 154–159, 2003.

[44] Gregory M. Nielson. Dual marching cubes. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 489–496. IEEE Visualization, 2004.

[45] Stanley Osher and Ron Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces.* Springer, 2003.

[46] Matt Pharr and Greg Humphreys. *Physically Based Rendering.* Elsevier, 2004.

[47] S. Premože, T. Tasdizen, J. Bigler, A. Lefohn, and R. Whitaker. Particle based simulation of fluids. *Computer Graphics Forum*, 22(3):401–410, 2003.

[48] Ilya D. Rosenberg and Ken Birdwell. Real-time particle isosurface extraction. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 35–43. ACM, 2008.

[49] Chen Shen and Apurva Shah. Extracting and parametrizing temporally coherent surfaces from particles. In *ACM SIGGRAPH '07 Sketches*, page 66, 2007.

[50] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[51] H. Si. TetGen, a quality tetrahedral mesh generator and three-dimensional delaunay triangulator, v1.3, user's manual. Technical Report 9, Weierstrass Institute for Applied Analysis and Stochastics, 2004.

[52] J. M. Sullivan. The geometry of bubbles and foams. In *Proc. NATO Advanced Study Institute on Foams, Emulsions and Cellular Materials*, pages 379–402. Kluwer Academic Publishers, 1998.

[53] Gabriel Taubin. A signal processing approach to fair surface design. In *Proc. ACM SIGGRAPH*, pages 351–358, 1995.

[54] Nils Thürey and Ulrich Rüde. Optimized free surface fluids on adaptive grids with the Lattice Boltzmann method. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Posters*, page 112. ACM, 2005.

[55] Alper Üngör. Tiling 3D euclidean space with acute tetrahedra. In *Proc. 13th Canadian Conference on Computational Geometry (CCCG'01)*, pages 169–172, 2001.

[56] B. Williams. The CBB method for unconstrained optimization. Technical report, Univ. of British Columbia Dept. of Comp. Sci., 2007.

[57] B. Williams. Particle level set method: implementation and results. Technical report, Univ. of British Columbia Dept. of Comp. Sci., 2007.

[58] B. Williams, R. Bridson, and M. Nordenstam. Smooth surface reconstruction from particles. In *Proceedings of SIGGRAPH '08: ACM SIGGRAPH 2008 Papers*. Association of Computing Machinery, ACM Press, 2008. To appear.

[59] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. *ACM Trans. Graph.*, 26(3):99, 2007.

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

[61] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *Proc. ACM SIGGRAPH*, pages 371–378, 2001.

Appendix A

Modified A15 Tile

Following Üngör [55] we construct tile A15 by first covering the XY plane with a 4x4 grid as shown in figure A.1.

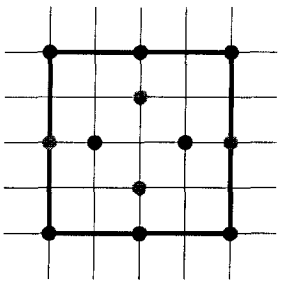

Figure A.1: Structure of the A15 tile in the XY plane

We define additional red vertices along the Z axis every $2k-1$ units, additional blue vertices every $4k-1$ units and additional green vertices every $4k+1$ units for all $k \in Z$. Next perform a Delaunay triangulation [51] on the point-set and identify and extract the resulting repeated tile. As shown in figure A.2 we shuffle the tetrahedra so the final tile is roughly cubical.

The modified A15 tile contains 27 vertices:

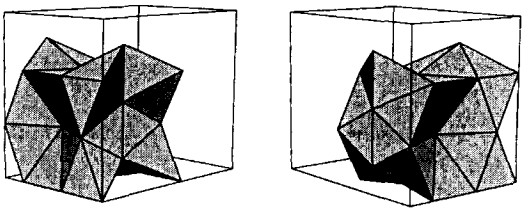

Figure A.2: The A15 variant from several viewpoints

0: (1,0,0)	7: (0,2,1)	14: (2,5,2)	21: (0,0,2)
1: (2,2,0)	8: (0,2,3)	15: (4,2,1)	22: (5,0,4)
2: (1,4,0)	9: (2,2,4)	16: (4,2,3)	23: (4,0,2)
3: (3,4,0)	10: (1,4,4)	17: (5,4,4)	24: (3,0,0)
4: (1,0,4)	11: (3,4,4)	18: (4,4,2)	25: (5,0,0)
5: (3,0,4)	12: (0,4,2)	19: (0,2,5)	26: (5,4,0)
6: (2,1,2)	13: (2,3,2)	20: (4,2,5)	

The vertices are connected with 46 tetrahedra:

0: (2,3,14,13)	12: (8,6,4,21)	24: (8,13,6,7)	36: (13,8,6,9)
1: (2,14,12,13)	13: (7,6,8,21)	25: (8,13,10,9)	37: (10,8,9,19)
2: (5,20,16,9)	14: (13,11,16,18)	26: (1,7,0,6)	38: (6,8,4,9)
3: (5,16,22,23)	15: (11,20,9,16)	27: (13,7,1,6)	39: (16,5,22,20)
4: (11,13,16,9)	16: (13,8,12,7)	28: (11,14,18,13)	40: (5,6,4,9)
5: (0,24,1,6)	17: (7,2,13,1)	29: (18,26,15,3)	41: (14,11,10,13)
6: (20,11,17,16)	18: (13,16,15,18)	30: (16,11,17,18)	42: (15,13,1,6)
7: (3,13,15,18)	19: (16,13,6,9)	31: (13,8,10,12)	43: (6,16,23,15)
8: (3,14,13,18)	20: (15,6,24,23)	32: (13,11,10,9)	44: (25,23,15,24)
9: (13,15,1,3)	21: (16,5,6,23)	33: (2,7,13,12)	45: (13,15,16,6)
10: (0,6,7,21)	22: (10,14,13,12)	34: (5,16,6,9)	
11: (6,15,24,1)	23: (3,2,1,13)	35: (4,8,19,9)	

Appendix B

Average Mesh Valence

The average valence in a mesh can be shown to be six using the Euler-Poincaré formula:

$$v + f - e = 2(c - g) - b$$

where

v = #vertices

f = #faces

e = #edges

c = #connected components

g = genus

b = #boundaries

In a closed manifold mesh

$$f = 2e/3$$

Therefore

$$v + 2e/3 - e = 2 - 2g$$

from which we can isolate

$$e = 3(v - 2 + 2g)$$

$$f = 2(v - 2 + 2g)$$

The required value is

$$\begin{aligned} average(degree) &= 2e/v \\ &= \frac{6(v - 2 + 2g)}{v} \end{aligned}$$

where

$$\lim_{v \to \infty} \frac{6(v - 2 + 2g)}{v} = 6$$