Volume xx (200y), Number z, pp. 1–10

Fast Mean-Curvature Flow via Finite-Elements Tracking

Ming Chuang and Michael Kazhdan

Johns Hopkins University

Abstract

In this paper, we present a novel approach for efficiently evolving meshes using mean-curvature flow. We use a finite-elements hierarchy that supports an efficient multigrid solver for performing the semi-implicit time-stepping. Though expensive to compute, we show that it is possible to track this hierarchy through the process of surface evolution. As a result, we provide a way to efficiently flow the surface through the evolution, without requiring a costly initialization at the beginning of each time-step. Using our approach, we demonstrate a factor of nearly seven-fold improvement over the non-tracking implementation, supporting the evolution of surfaces consisting of 1M triangles at a rate of just a few seconds per update.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Geometric algorithms, languages, and systems—

1. Introduction

Surface evolution, or surface flow, has many important applications in geometry processing, including surface parameterization [JKG07], surface fairing [DMSB99, BS05], skeleton extraction [ATC*08], and mesh editing [CDR00, HP04]. In these applications, the surface is evolved by prescribed rules, often driven by the gradients of a nonlinear energy, as in cases like mean-curvature flow and Willmore flow.

Such dynamic systems mainly consist of two iterated steps: *evaluation* of the current state of the system, and *solution* for the next state of the system. In the context of mean-curvature flow, the first step involves estimating the current mass matrix and Laplace-Beltrami operator, and the second requires the solution of a linear system. The efficiency of surface evolution is determined by the speed with which these two steps can be performed.

One promising way to implement surface flow efficiently is to leverage the recent work of Chuang *et al.* [CLB*09]. In that work, the authors propose an octree-based finite-elements system that can be used to define a fast multigrid solver. Unfortunately, the approach is limited by the initial start-up cost of setting up the multigrid framework, requiring the construction of an adaptive octree and the partitioning of the model's triangles to the faces of the octree cells. As a result, using this approach in a setting where the geometry is changing at each time-step (requiring the computation of a new octree and a re-partitioning of the model at every frame) would seem to be impractical.

Focusing on mean-curvature flow, we show how to avoid the overhead of rebuilding the system at every time-step. The key idea behind our approach is to evolve the octree defining the finite-elements system with the deforming surface. Figure 1 shows a visualization of this approach for a 2D Raptor model undergoing a swirl deformation (top row). Because the embedding of the Raptor evolves, a direct application of [CLB*09] would require fitting a new octree hierarchy at each step (middle row). Instead, we adapt the hierarchy by evolving the octree with the deforming surface (bottom row). As we will show, this approach provides a significant speed-up, supporting the evolution of high-resolution surfaces at a rate of just a few seconds per update.

The remainder of the paper is organized as follow: We start with a brief survey of related work in Section 2, and review mean-curvature flow, finite-elements solvers, and the octree-based finite-elements system in Section 3. After that, we describe our tracking scheme and its implementation in Section 4. We show results in Section 5 and conclude with a summary and discussion of future directions in Section 6.

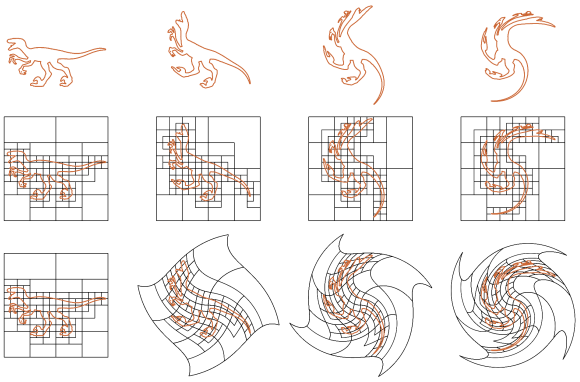


Figure 1: *A 2D raptor model undergoing a "swirl" deformation (top). Computing a quadtree independently for each deformation, we obtain a temporally-varying spatial indexing structure (middle). Tracking the quadtree with the deformed surface, the indexing structure remains constant, allowing us to reuse information from frame to frame (bottom).*

2. Related Work

Geometric flow Geometric flow is an important technique that supports numerous applications in geometry processing and has included well-known flows such as mean-curvature flow [DMSB99], Willmore flow [BS05], and Ricci flow [JKG07]. Other variants have also been used for volume-controlled fairing [EPT*07], anisotropic diffusion for detail preserving smoothing [CDR00, HP04], and Gaussian curvature flow [ZGX06].

In addition to providing methods for smoothing a mesh, geometric flows have also played an important role in other graphics applications. Pinching tubular regions, mean-curvature flow has been used for skeleton extraction [ATC*08]. Flowing surfaces to match prescribed curvature normals, the shearing resulting from differential coordinate editing [SCOL*04, YZX*04] can be corrected [ATLF06]. And, using boundary position and normal information, fourth-order flows have been used for filling in holes [BS05, XPB06].

Multigrid Solvers Evolving the surface with the flow often requires the solution of a sparse linear system of equations defined over the mesh. Although the construction of efficient multigrid solvers is a reasonably straight-forward task when considering (homogenous) linear systems over regular grids (e.g. [BHM00]), the task is more challenging in the context of meshes where the unconstrained triangulation does not directly admit a hierarchical structure.

A variety of methods have been proposed for addressing this challenge. These have ranged from "black-box" algebraic multigrid solvers that rely solely on the algebraic information encoded in the system matrices [RS87] (as applied in mesh decomposition [CGR*04] and flow-field clustering [GPR*04]) to more geometry-driven solvers that support multigrid through the design of mesh hierarchies [KCVS98, CDR00, SK01, RL03, AKS05, SYBF06, SBZ09]. More recently, a multigrid solver has been proposed that avoids the creation of a mesh hierarchy by embedding the mesh within a regular 3D grid and using the regularity to define the multigrid hierarchy on the surface [CLB*09].

3. Review of Mean-Curvature Flow

We begin by reviewing the temporal discretization of mean-curvature flow using semi-implicit integration. Then, we describe how finite-elements systems are used to perform the spatial discretization and outline the finite-elements system on which our method is based.

In our discussion, the evolution of the surface is defined with respect to a hierarchy of functions defined on a triangle mesh M (imbued with some Riemannian metric structure) and is expressed by the functions $X_t : M \to \mathbb{R}^3$ giving the embedding of M into three-space at time t. Though this is not the "natural" formulation for describing the octree-based finite-elements system of Chuang *et al.* (which defines test-functions over the embedded surface $X(M)$ rather than the triangle mesh M), it is precisely this interpretation that enables our adaptation of the octree-based finite-elements to the context of evolving surfaces.

This approach is similar in spirit to the *logical nesting* of Kornhuber and Yserntant [KY08], where we replace triangles as the atomic units with the patches obtained by intersecting the surface with the nodes of the octree. However, we stress that we lack the convergence proofs provided in [KY08] and validation of our approach is solely empirical.

3.1. Temporal Discretization

Given an embedding $X : M \to \mathbb{R}^3$ of the mesh into three-space, mean-curvature flow is a smoothing process evolving the embedding subject to the differential equation:

$$\frac{\partial}{\partial t} X = -\vec{H}_X = -\Delta_X X$$

where $\vec{H}_X$ is the mean-curvature vector of the embedded surface and Δ_X is the Laplace-Beltrami operator, defined with respect to the embedding.

Perhaps the most straightforward approach for performing mean-curvature flow is to use explicit integration:

$$\frac{X_{t+\delta} - X_t}{\delta} \approx -\Delta_t X_t \quad \Rightarrow \quad X_{t+\delta} \approx X_t - \delta \Delta_t X_t .$$

(Here, Δ_t is the Laplace-Beltrami operator defined with respect to the embedding X_t.) However, the difficulty with using explicit integration is that unless the step-size δ is very small, the flow exhibits instability (manifest as the unwanted

negation and amplification of high-frequency content) making the approach impractical for long-term flows.

As described by Desbrun *et al.* [DMSB99], the instability can be addressed by using a semi-implicit solver:

$$\frac{X_{t+\delta}-X_t}{\delta} \approx -\Delta_t X_{t+\delta} \;\Rightarrow\; (\mathrm{Id}+\delta\Delta_t)X_{t+\delta} \approx X_t. \quad (1)$$

(The integrator is *semi*-implicit because the Laplace-Beltrami operator is still defined with respect to the embedding at time t.) Although this method has the same convergence properties as the explicit approach, it is stable even in the presence of larger time-steps, making it more suitable in practice for simulating large-scale flows.

3.2. Spatial Discretization

To perform the semi-implicit time-step described in Equation 1, the PDE needs to be discretized in space. This is most often done using the finite-elements approach. A set of test-functions is defined on the base mesh, $\{B_I : M \to \mathbb{R}\}$, and the embeddings $X_t : M \to \mathbb{R}^3$ are assumed to reside within the span of these functions:

$$X_t(p) = \sum_I x_t^I B_I(p) \qquad \text{with } x_t^I \in \mathbb{R}^3.$$

Using these test-functions, the embedding at time $t+\delta$ is found by projecting the solution to Equation 1 onto the span of the test-functions. That is, the coefficients at time $t+\delta$ are obtained by solving the linear system of equations:

$$\langle B_J(p), (\mathrm{Id}+\delta\Delta_t)X_{t+\delta}\rangle_t = \langle B_J(p), X_t\rangle_t \quad \forall J$$

where $\langle\cdot,\cdot\rangle_t$ is the inner-product on the space of functions, defined by the metric induced by the embedding X_t.

Denoting by $\vec{x}_t$ the coefficients of the embedding X_t with respect to the test-functions, the above becomes:

$$\left(D^t + \frac{\delta}{2}L^t\right)\vec{x}_{t+\delta} = D^t\vec{x}_t \quad (2)$$

where D^t and L^t are the mass and Laplacian matrices, defined with respect to the embedding X_t:

$$\begin{aligned} D^t_{IJ} &= \int_M B_I(p)\cdot B_J(p)\sqrt{|g_t(p)|}\,dp \quad (3)\\ L^t_{IJ} &= -\int_M (\nabla_M B_I(p))^T g_t^{-1}(p)(\nabla_M B_J(p))\sqrt{|g_t(p)|}\,dp. \end{aligned}$$

Here, $g_t = dX_t^T dX_t$ is the metric tensor and $\nabla_M B_I$ is the gradient of B_I. Note that though the integrands are defined in terms of a particular Riemannian structure on M, the integrals themselves are independent of the structure as all areas and angles are computed by pulling back the inner product from the embedding $X_t(M)$.

Octree-based Finite-Elements System

While a semi-implicit solver provides a stable way for taking larger time-steps, its use in surface flow (particularly in the context of high-resolution meshes) depends on the ability to construct and solve the system in Equation 2 efficiently.

The approach we pursue in this work builds on the recent work of Chuang *et al.* [CLB*09]. Instead of defining an intrinsic function space over the abstract mesh M, Chuang *et al.* define an extrinsic function space over the embedded manifold, $\mathscr{M} = X(M)$. First, a function space is defined on $\mathbb{R}^3$ by centering a set of degree d B-splines, $\{\tilde{B}^h_I : \mathbb{R}^3 \to \mathbb{R}\}$, within a regular 3D grid of resolution h:

$$\begin{aligned} \tilde{B}^h_{i,j,k}(p) &= \tilde{B}\left(\frac{p}{h} - (i,j,k) - \left(\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)\right) && \text{for } d \text{ even}\\ \tilde{B}^h_{i,j,k}(p) &= \tilde{B}\left(\frac{p}{h} - (i,j,k)\right) && \text{for } d \text{ odd} \end{aligned}$$

where $\tilde{B}(p)$ is the untransformed B-spline of degree d (obtained by convolving the indicator function of the domain $[-0.5,0.5]^3$ with itself d times). Then, a function space is defined on the embedded surface by considering the restriction of the functions $\tilde{B}^h_I$ to the points on $\mathscr{M}$.

Since the space of B-splines defined by a regular grid of resolution $2h$ is nested within the space of B-splines defined by a regular grid of resolution h and since function restriction is a linear operator, this definition of test-functions provides a nesting hierarchy supporting a multigrid solver. In particular, one can use the same prolongation/restriction operators used for performing the refinement/coarsening of B-splines in the 3D case to perform the refinement/coarsening of test-functions restricted to the embedded surface.

Finally, since the supports of the B-splines associated with most cells do not overlap the surface, the test-functions can be indexed by an adaptive octree.

To compute the integrals defining the system coefficients in Equation 3, the authors use a cubature approach. They clip the triangles in the original mesh to the faces of the grid cells (so that the test-functions are polynomial on each clipped triangle) and then sample at appropriate cubature positions. Using the set of cubature points $\mathscr{P} \subset \mathscr{M}$, the computation of the matrix coefficients is performed by setting:

$$\begin{aligned} D^h_{IJ} &= \sum_{p\in\mathscr{P}} \tilde{B}^h_I(p)\cdot\tilde{B}^h_J(p)\cdot|\mathscr{T}(p)|\cdot\omega_p\\ L^h_{IJ} &= -\sum_{p\in\mathscr{P}} \langle\nabla_{\mathscr{M}}\tilde{B}^h_I(p), \nabla_{\mathscr{M}}\tilde{B}^h_J(p)\rangle\cdot|\mathscr{T}(p)|\cdot\omega_p \end{aligned}$$

where $0 \le \omega_p \le 1$ is the cubature weight associated to the sample p, $|\mathscr{T}(p)|$ is the area of the triangle in $\mathscr{M}$ that contains p, and $\nabla_{\mathscr{M}}$ is the gradient operator restricted to the embedded manifold (obtained by computing the 3D gradient of $\tilde{B}^h_I$ and projecting it onto the tangent plane of $\mathscr{M}$ at p).

4. Approach

The challenge of using the octree-based finite-elements system for performing surface evolution is that the test-functions depend on the embedding of the triangle mesh

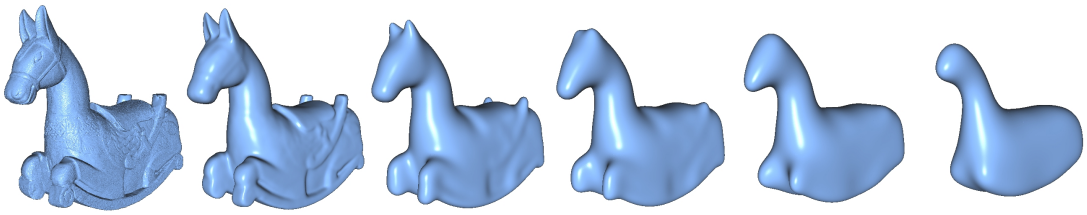


Figure 2: *Mean-curvature flow of the Isidore Horse after* 0, 5, 25, 50, 100, *and* 200 *iterations with step-size* $\delta = 1 \times 10^{-5}$.

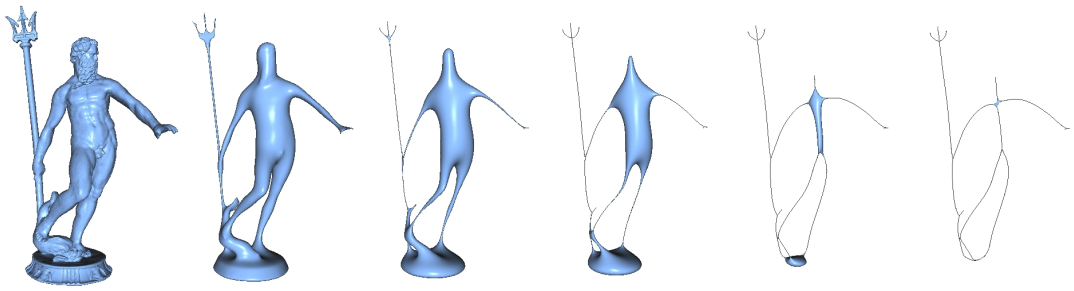


Figure 3: *Mean-curvature flow of Neptune after* 0, 5, 10, 20, 40, *and* 80 *iterations with step-size* $\delta = 5 \times 10^{-4}$.

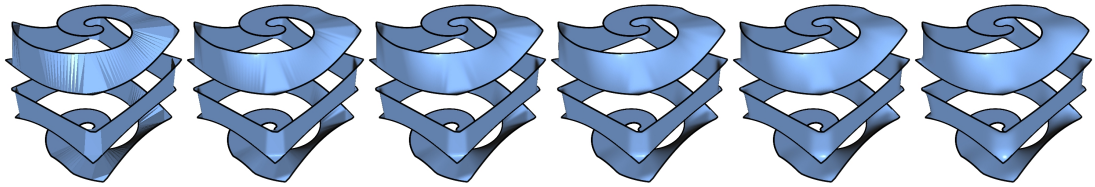


Figure 4: *Mean-curvature flow for a cubical spiral after* 0, 5, 10, 20, 40, *and* 80 *iterations with step-size* $\delta = 1 \times 10^{-5}$.

within $\mathbb{R}^3$. As a result, a new set of test-functions would need to be defined at each time-step, requiring the costly construction of the octree and partitioning of the input triangles to the faces of the octree nodes.

We propose a new method for performing surface evolution that deforms the test-functions *with* the evolving surface (Figure 1). The advantage of our approach is that the nesting structure of the test-functions initially supporting the multigrid solver persists throughout the evolution. As a result, the same hierarchical structure constructed at $t = 0$ can be used for all values of t, alleviating the need for the computationally expensive octree initialization.

Although this approach may appear impractical due to the complexity of the deformed test-functions (function degree increases under composition), we show that it is in fact easy to implement. The key to our implementation lies in the observation that though the function space on the embedded surface $\mathscr{M}_t = X_t(M)$ is extrinsic, we can turn this into an intrinsic function space on M using the pull-back of the embedding. As a result, we obtain a set of test-functions on M that is independent of the embedding/deformation.

4.1. General Approach

In our approach, we use the initial embedding of the mesh, $X_0 : M \to \mathbb{R}^3$, to define a nested hierarchy of functions on M. Then, rather than constructing the test-functions from scratch at each time-step t, we use the test-functions defined by the initial embedding, and only use the embedding X_t to define the metric structure against which we integrate.

Specifically, if we set $\{B_I : M \to \mathbb{R}\}$ to be the pull-back of the test-functions on $X_0(M)$ (setting $B_I = \tilde{B}_I \circ X_0$) and we set $P = X_0^{-1}(\mathscr{P})$ to be the pre-image of the cubature points on M, then the coefficients of the linear system become:

$$D^t_{IJ} = \sum_{p \in P} B_I(p) \cdot B_J(p) \cdot \sqrt{|g_t(p)|} \cdot |T(p)| \cdot \omega_p \tag{4}$$

$$L^t_{IJ} = -\sum_{p \in P} (\nabla_M B_I(p))^T g_t^{-1}(p)(\nabla_M B_J(p)) \cdot \cdot \sqrt{|g_t(p)|} \cdot |T(p)| \cdot \omega_p$$

where $|T(p)|$ is the area of the triangle in M that contains p, computed with respect to the Riemannian structure on M.

4.2. Data Re-Use

The advantage of the formulation in Equation 4 is that it allows for a computation of the matrix entries that re-uses much of the same information. In particular, since the values $B_I(p)$ and $\nabla_M B_I(p)$ are independent of the embedding, we only need to compute these once. It is only the metric g_t that changes throughout the evolution.

To compute the metric tensor, we use the fact that the solution from the previous time-step provides the coefficients, x_t^I, of the embedding with respect to the test-functions:

$$X_t(p) = \sum_I x_t^I B_I(p) \qquad \text{with } x_t^I \in \mathbb{R}^3.$$

Using these coefficients, the differential of the embedding can then be expressed as:

$$dX_t\big|_p = \sum_I x_t^I \left(\nabla_M B_I(p)\right)^T.$$

Thus, given the values and gradients of the test-functions (intrinsic information) and given the coefficients of the embedding at time-step t (extrinsic information), we have the necessary information for computing the linear system that defines the embedding at time-step $t+\delta$.

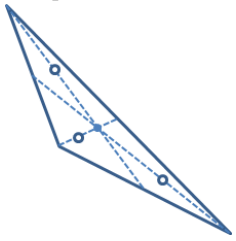
Tracking Data with First-Order B-Splines

We define the test-functions by setting the $\{\tilde{B}_I\}$ to be first-order B-splines centered at the vertices of a regular voxel grid. Because they are first-order, each B-spline is supported within the eight grid-cells adjacent to the vertex and every point $p \in P$ is contained in the support of exactly eight test-functions. Thus, we perform the surface evolution by storing the following values with each cubature point:

- The weight of the sample: $w_p = |T(p)| \cdot \omega_p$ $(1 \times |P|)$,
- The values of the test-functions: β_p^I $(8 \times |P|)$,
- The values of the gradients, w.r.t. an orthonormal frame for the tangent space T_pM: (η_p^I, μ_p^I) $(16 \times |P|)$.

In addition, for each vertex $v \in M$, we also store the values of the eight test-functions supported at v, denoted β_v^I.

We approximate the integrals by using a 3-point cubature formula with equally weighted samples on the mid-points of the line segments connecting the vertices of a triangle to its centroid (visualized by the white points in the inset). While the correct computation of the Laplace-Beltrami operator would require using a fourth-order integrator and the computation of the mass-matrix entries would require a sixth-order integrator (e.g. the 11-point integrator of Day and Taylor [DT07]), we have found that our three-point formula is sufficiently accurate and does not lead to perceptible errors.



4.3. Mean-Curvature Flow

We implement mean-curvature flow in three logical steps. First, in a pre-processing step, the values ω_p, β_p^I, η_p^I and μ_p^I are computed. Then, at each time-step t, we compute the system matrices and define the linear system. Finally, we solve the linear system and use the solution to evaluate the position of the vertices on the new mesh.

Pre-Processing To set the values of β_p^I, we use the embedding X_0 to map p into $\mathbb{R}^3$. We identify the octree node in which the sample falls and evaluate the eight B-splines centered on the corners of the node at the point $X_0(p)$.

Defining the values $|T(p)|$ and (η_p^I, μ_p^I) requires choosing a Riemannian structure for M. We do this by pulling back the differential structure from the inital embedding X_0: For a point p contained in a triangle $T \subset M$, we set $|T(p)|$ to the area of $X_0(T)$. Similarly, we set (η_p^I, μ_p^I) by choosing an orthonormal basis for the plane tangent to $X_0(T)$ and projecting $\nabla\tilde{B}_I$ onto this basis. (This makes the initial metric-tensor, g_0, equal the identity.)

Computing the System Matrices To compute the matrices D and L from the coefficients of the embedding, $\vec{x}$, we perform the following steps:

Algorithm: SetSystemMatrices($\vec{x}$)

```
D , L ← 0
for p ∈ P
  dX ← ComputeDifferential(p, x⃗)
  g ← dX^T dX
  for I ∈ NeighborCorner(p)
    for J ∈ NeighborCorner(p)
      D_IJ ← D_IJ + β_p^I · β_p^J · √|g| · w_p
      L_IJ ← L_IJ + (η_p^I, μ_p^I) g^{-1} (η_p^J, μ_p^J)^T · √|g| · w_p
return ( D^t , L^t )
```

Here, NeighborCorner(p) returns the indices of the eight grid corners whose associated B-splines contain $X_0(p)$ in their support and ComputeDifferential$(p, \vec{x})$ is the function that uses the coefficients of the current embedding to evaluate the differential of the embedding map at p:

Algorithm: ComputeDifferential(p , $\vec{x}$)

```
dX ← 0
for I ∈ NeighborCorner(p)
  dX ← dX + x^I (η_p^I, μ_p^I)
return dX
```

Solving and Updating the System Given the system matrices and the coefficients of the embedding $\vec{x}_t$, we set $\vec{x}_{t+\delta}$ to the solution of the linear system in Equation 2, obtained using the multigrid solver described in [CLB*09].

Lastly, to determine the new embedding of the mesh M, we need to evaluate the embedding $X_{t+\delta}$ at each of the vertices of M. To this end, we use an approach similar to the

one used for computing the differential, invoking the function EvaluateVertex$(v, x_{t+\delta})$ at each vertex $v \in M$:

Algorithm: EvaluateVertex(v , $\vec{x}$)

$q \leftarrow 0$
for $I \in$ NeighborCorner(v)
  $q \leftarrow q + \beta_v^I x^I$
return q

5. Results

To evaluate our approach, we demonstrate its stability and measure its accuracy. We also discuss extensions to minimal surface flow and summarize the performance characteristics. We conclude by discussing limitations of our approach. (All models except the sphere are scaled to fit into a unit-cube.)

5.1. Stability

To evaluate the stability of our approach, we evolved two different models under the action of mean-curvature flow. The results of these flows can be seen in Figures 2 and 3, which show the evolution of the Isidore Horse and the Neptune models. The figures highlight the robustness of our method. Using small time-steps, our solution remains stable even after many iterations of semi-implicit integration (Isidore). Using the stability of semi-implicit time-stepping, we can also take larger time-steps (Neptune), to quickly arrive at the "skeleton" of the mesh, as described in [ATC*08].

Note that in pinching regions, the cubature points do not contribute to the integrals. As a result, the associated solution coefficients remain unchanged by the iterative solver. Since we use the solution from the previous time-step to initialize the solver, this has the effect of anchoring vertices.

5.2. Accuracy

To evaluate the correctness of our approach, we consider three questions: How does our solver compare to existing methods? What is its multigrid behavior? And, how stable is the integration?

Comparison to Existing Methods

We compare the results of using our tracked-finite-elements with two other implementations of mean-curvature flow. In the first, we use the octree-based finite-elements system of Chuang *et al.* [CLB*09], fitting a new octree at each time-step and using the associated functions to define the coefficients of the linear system. In the second, we use the cotangent-weight Laplacian operator, combined with (non-diagonally-lumped) mass-matrix, defined by integrating products of per-vertex hat functions [Dzi88] (and derivatives) over the mesh.

Figure 5 shows results for the Bimba model after one, ten, and one hundred steps of mean-curvature flow with step-size $\delta = 1 \times 10^{-5}$. Visually, the results of our tracking implementation (bottom) are indistinguishable from the results of both the non-tracking (middle) and cotangent-weight Laplacian (top) solutions.

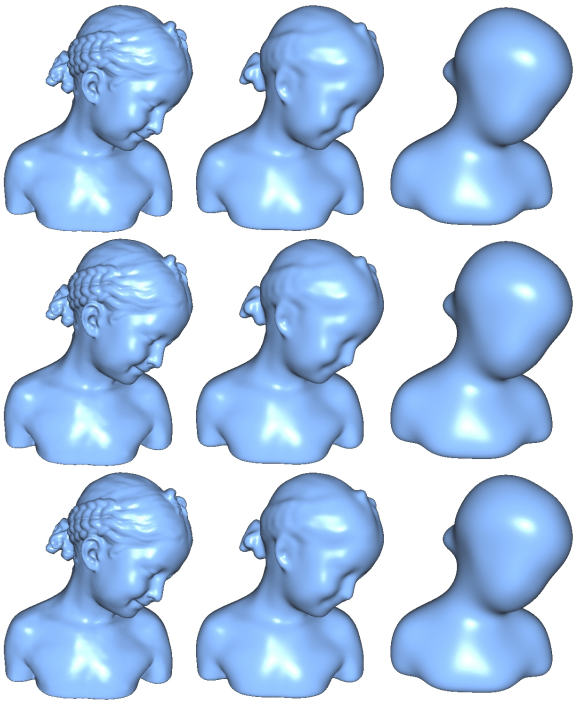


Figure 5: *The Bimba model, evolved through one, ten, and one hundred time-steps of mean-curvature flow using a step-size of* $\delta = 1 \times 10^{-5}$. *The figures compare the results obtained using the cotangent-weight Laplacian (top), Chuang* et al.*'s octree-based system without frame-to-frame tracking (center), and our tracking adaptation (bottom).*

We quantify these results using the Metro [Met07] tool, computing the distances from the solutions obtained using the octree-based systems at depth 8 to the solution obtained using the cotangent-weight Laplacian. For the octree-based flows, we solve Equation 2 using one W-cycle with five Gauss-Seidel iterations per level. For the cotangent-weight flows, we used the LIS library [LIS10] to solve with a Conjugate-Gradients solver, pre-conditioned with Smoothed Aggregation AMG, and run for at most 3000 iterations per time-step or until the residual was below 1×10^{-10}.

Results of these experiments are shown in the left plots of Figure 6, with the maximum (Hausdorff) distance to the cotangent-Laplacian solution drawn with dashed lines and the RMS distance drawn with solid lines. As the plots show, the error obtained using our tracked octree method (orange) tend to be lower than the errors obtained by naively rebuild-

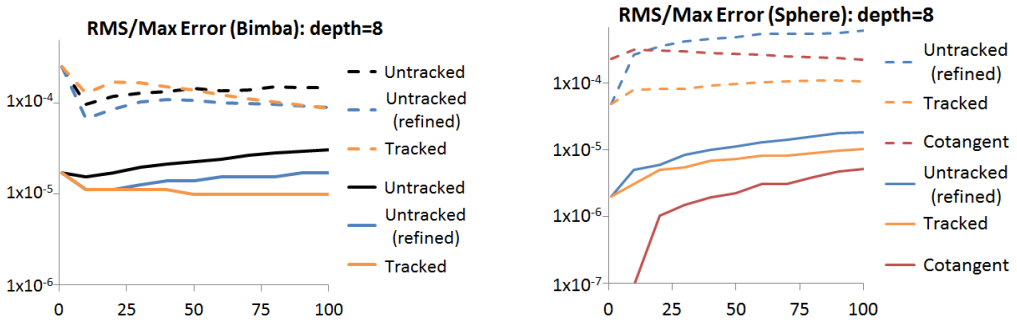


Figure 6: *Error plots for mean-curvature flow of the Bimba model (left) and the sphere (right), showing the maximum (dashed lines) and RMS (solid lines) errors. For the Bimba model, the distances are measured to the solution obtained using the cotangent-weight Laplacian. For the sphere, the distances are measured from the ground-truth, semi-implicit solution.*

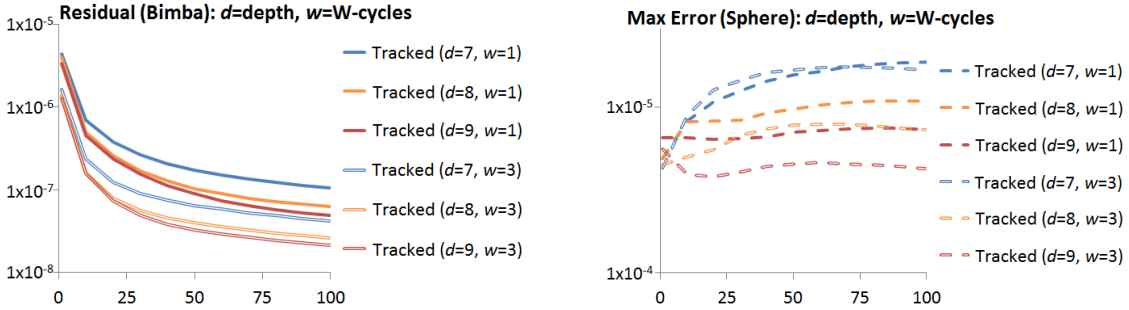


Figure 7: *Plots for mean-curvature flow, showing the residuals for the Bimba (left) and Hausdorff distances for the sphere (right), as a function of the depth of the octree and the number of W-cycles.*

ing the octree at each time-step (black). We believe that this is largely due to the clipping of triangles to the faces of the octree nodes. Although both methods estimate the integrals over the refined triangulation, the non-tracking solution "flattens" the resulting geometry at the end of each time-step by only sampling the flow at the original vertices. In contrast, the tracking solution works with the same refined triangulation, allowing us to capture more fine-grained properties.

To test this hypothesis, we re-ran the non-tracking flow, but this time we applied a 1-to-4 face-split to the initial mesh, thereby allowing the flow to evolve a higher-resolution surface. As the plots show (blue), refining the mesh before running the non-tracked solver improves the accuracy.

We also evaluated the accuracy of the three methods when performing one hundred steps of mean-curvature flow on a unit sphere, using a step-size of $\delta = 1 \times 10^{-3}$. These results are shown in the right plots of Figure 6. In this case, the ground-truth solution is known so we compare all three methods against it. Although the analytic solution for the radius of the sphere acted on by mean-curvature flow is:

$$r'(t) = -\frac{2}{r(t)} \quad \Longrightarrow \quad r(t+\delta) = \sqrt{r^2(t) - 4\delta},$$

this is not the ground-truth solution that we compare against. The reason is that comparing against the analytic solution captures errors that arise due to both temporal and spatial discretization. However, the choice of finite-elements system only affects the spatial discretization. To isolate the effects of spatial discretization, we compare the results to the sphere whose radius is predicted by a semi-implicit solution:

$$r(t) = (1 + \delta\Delta_t)\, r(t+\delta) \quad \Longrightarrow \quad r(t+\delta) = \frac{r^3(t)}{r^2(t) + 2\delta}.$$

As the plots on the right of Figure 6 show, the results efficiently obtained using a multigrid solver with our tracked system (orange) are comparable to the results obtained using a more expensive pre-conditioned conjugate-gradient solver (red) and are slightly more accurate than the results obtained using non-tracking octrees (blue), even when the mesh used for the non-tracking octrees is initially refined.

Multigrid Behavior

To evaluate the multigrid behavior of our solver, we consider the accuracy of the solution, as a function of the depth of the octree and the number of W-cycles used. For the Bimba model, where a ground-truth solution is not available, we measure the accuracy in terms of the norm of the residual.

For the sphere, we use the Hausdorff distance. The results of these experiments are shown in Figure 7.

Though we fix the number of Gauss-Seidel iterations within a resolution to five, there is no deterioration in solver performance as the resolution is increased. Additionally, for the Bimba, we see reduction in residual as the number of W-cycles goes up. For the sphere, we do not see an analogous reduction in error when the finite-elements are defined over a low-resolution octree. We believe that this is because the limiting factor in reducing the Hausdorff distance is the precision of the test functions, not the performance of the solver. (We validated this by examining the residuals, which exhibit similar trends to the ones shown for the Bimba model.)

Note that since we are using a multigrid solver with W-cycles rather than V-cycles, solving at a higher depth implies that coarser resolutions are relaxed more often. As a result, when the system is predominantly low-frequency (e.g. after the mean-curvature flow has been run for a few time-steps and the geometry has been smoothed) the solution obtained using the higher resolution system tends to give lower residual than the solutions from the lower resolution systems.

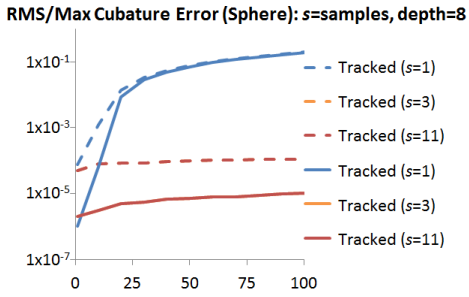


Figure 8: *Plots showing the accuracy of our tracked solver as a function of the number of cubature samples used. Though the one-point cubature (blue) provides an inaccurate estimate of the integral, our three-point cubature (orange) gives results identical to those of the full 11-point cubature (red) (and is, consequently, obscured in the plot).*

Stability of the integration

To evaluate the accuracy of the integrals obtained using our three-point cubature, we compare to the results obtained using a simple one-point estimate and a correct 11-point estimate [DT07]. The results of these experiments for the sphere (run at depth 8, with one V-cycle, and 5 Gauss-Seidel iterations per resolution) are shown in Figure 8. As the plots demonstrate, the assumption that the functions are piecewise constant on the triangles is too simplifying and results in large error. However, the use of an 11-point cubature for estimating the integrals of degree six polynomials is also unnecessary, as the results obtained using our three-point cubature are indistinguishable.

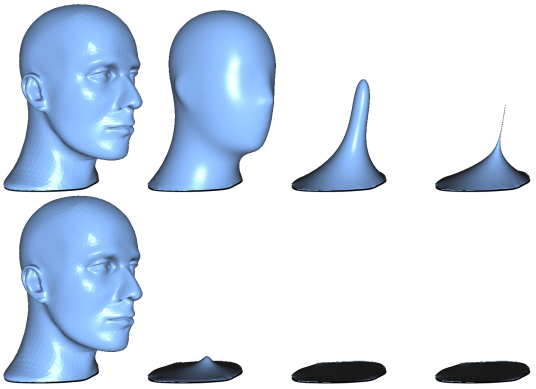


Figure 9: *Evolving the head with short time-steps (top) results in undesirable surface pinching. Using larger time-steps (bottom) approximates the minimization of the Dirichlet Energy and quickly converges to a minimal area surface.*

5.3. Minimal Area Surfaces

For non-water-tight surfaces, we extend our approach by constraining the solver to lock the positions of points on the boundary. We do this by fixing the coefficients of test functions whose support overlaps the boundary. At the finest resolution, we use the fact that the identity function can be expressed as a linear combination of test functions:

$$p = \sum_{i,j,k} (i,j,k)\tilde{B}_{i,j,k}(p)$$

and set the coefficient of a test function overlapping the boundary, indexed by corner (i,j,k), to be (i,j,k). This has the effect of reproducing the identity function in nodes containing the boundary. At coarser resolutions, we use the fact that the boundary constraints are guaranteed to be met at the finest resolution and lock the coefficients of functions whose support overlaps the boundary to zero. Figure 4 shows an example of the solver for a cubical helix, demonstrating how the initial surface evolves towards a minimal area surface.

Examining Equation 1, we observe that as the time-step δ get larger, the system converges to:

$$\begin{aligned} \Delta_t X_{t+\delta} &= 0 \\ X_{t+\delta}(p) &= X_0(p) \qquad \forall p \in \partial M \end{aligned}$$

Noting that this is the minimizer of the Dirichlet energy, we see that (for large time-steps) our approach approximates Pinkall and Polthier's [PP93] method for computing minimal area surfaces. As described in their work, the advantage of using the minimizer of the Dirichlet Energy is that it avoids unwanted surface pinching. This can be seen in Figure 9 which shows two evolutions of the Mannequin head. Using short time-steps (top) the surface pinches at the neck while using large time-steps (bottom) we quickly converge to the minimal surface.

Model	Depth	Input Vertices	Split Triangles	Dimension	System Set-up Time	System Update + Solve Time
Isidore Horse	9	1.1×10^6	1.0×10^7	1.2×10^6	16 (+10)	3.7 + 0.47
Neptune	9	5.0×10^5	4.3×10^6	4.7×10^5	7 (+4)	1.3 + 0.20
Cubical Spiral	9	5.3×10^5	1.1×10^7	1.4×10^6	15 (+10)	3.4 + 0.48
Bimba	8	3.0×10^5	2.7×10^6	3.4×10^5	4 (+3)	0.9 + 0.14
Sphere	8	3.7×10^5	3.8×10^6	5.1×10^5	6 (+3)	1.4 + 0.22
Mannequin	7	2.7×10^4	3.7×10^5	7.8×10^4	1 (+0.5)	0.1 + 0.05

Table 1: *Performance of mean-curvature flow for the models in Figures 2–9, showing the depth of the octree, the number of vertices in the input model, the number of triangles after splitting to the octree cells, the dimension of the linear system, the pre-processing time for initializing the octree (plus the time it would have taken to construct the system), and the average times for setting up and solving the linear system at each frame. All timings are in seconds.*

5.4. Performance

The performance of our system is summarized in Table 1, showing the depth of the octree, the size of the input triangle mesh, the size of the mesh obtained after splitting to the faces of the octree cells, the dimension of the linear system (defined by the number of finest-level octree nodes), the time required to initialize the octree, and the times required to set up and solve the linear system at each semi-implicit time-step. All of these results were run using a single W-cycle with five Gauss-Seidel iterations per level. (The depth was chosen so that the dimension of the system would roughly match the number of input vertices.)

As the table indicates, the time required for adapting an octree and re-partitioning the mesh markedly outweighs the combined time for constructing the system matrices at time t and solving for the embedding at time $t+\delta$. In particular, we see that by tracking the finite-elements system, we are able to get a nearly 7-fold improvement in the processing time.

5.5. Limitations

Though our approach provides an effective method for tracking a multigrid solver with an evolving surface, the use of cubature for performing the integration can come at a cost in both space and time. Using 3-point cubature, we require the storage of $3 \cdot 25$ floating point values with each clipped triangle and, even with the use of first-order B-splines, each of these cubature points contributes to as many as 64 different matrix coefficients. While we have been unsuccessful in developing a stable implementation using Monte-Carlo integration with many fewer samples, this is an area of research we hope to continue pursuing in the future.

Additionally, we would like to stress that the contribution of our work is *not* a solver that outperforms existing techniques. Comparisons to the performance described by Shi *et al.* [SYBF06] indicate that though our solver is faster, we spend more time setting up the system matrices, resulting in comparable overall run-time performance. Rather, we believe that the contribution of our work is its exploration and adaptation of a recently developed finite-elements solver to the context of evolving surfaces, providing an alternate approach to defining and solving linear systems over meshes.

Also, we note that our finite-elements tracking scheme assumes that the test-functions pulled back by the initial embedding provide a function space rich enough to describe signals over embeddings in future time-steps. While this is true in the context of mean-curvature flow, which preserves the general shape of the geometry throughout the evolution, this may not be true for other flows (e.g. flows morphing one shape into another) and requires further investigation.

Finally, as discussed in [CLB*09], using test functions that are defined in 3D has the disadvantage of "supplanting geodesic distances with Euclidian ones, at the resolution of the ... functions". In the context of surface evolution, this limitation is manifest in two ways. First, when a surface pinches under the action of mean-curvature flow, then even if the singularity is identified and the topology of the surface is changed to split the surface at the pinch region (as in [PP93]), the test functions do not provide the resolution to allow the separated surfaces to evolve in different directions. Second, in evolving surfaces to have minimal area, the constraints that the boundary remain locked are implemented at the resolution of the test functions, so the position of *any* point falling within a node intersected by the boundary will remain unchanged.

6. Conclusion

In this work, we presented an approach for tracking a hierarchical finite-elements system with a surface evolving under mean-curvature flow. Using our approach, we have shown that it is possible to significantly reduce the computational overhead of adapting the linear system to the geometry of the evolving surface without sacrificing either stability or accuracy. We have demonstrated the utility of this approach in several simulations, demonstrating that our method supports the deformation of high-resolution models at a rate of just a few seconds per frame.

In future work, we would like to explore extensions of our approach to higher-orders flows such as those defined by the Willmore, membrane, and flexural energies.

References

[AKS05] AKSOYLU B., KHODAKOVSKY A., SCHRÖDER P.: Multilevel solvers for unstructured surface meshes. *SIAM Journal of Scientific Computing 26* (2005), 1146–1165. 2

[ATC*08] AU O., TAI C., CHU H., COHEN-OR D., LEE T.: Skeleton extraction by mesh contraction. *ACM Transactions on Graphics (SIGGRAPH '08) 27*, 3 (2008). 1, 2, 6

[ATLF06] AU O., TAI C., LIU L., FU H.: Dual Laplacian editing for meshes. *IEEE Transactions on Visualization and Computer Graphics 12* (2006), 386–395. 2

[BHM00] BRIGGS W., HENSON V., MCCORMICK S.: *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, 2000. 2

[BS05] BOBENKO A., SCHRÖDER P.: Discrete Willmore flow. In *EG Symposium on Geometry Processing* (2005), pp. 101–110. 1, 2

[CDR00] CLARENZ U., DIEWALD U., RUMPF M.: Anisotropic geometric diffusion in surface processing. In *Visualization '00: Proceedings of the 11th IEEE Visualization 2000 Conference (VIS 2000)* (2000), pp. 497–405. 1, 2

[CGR*04] CLARENZ U., GRIEBEL M., RUMPF M., SCHWEITZER M. A., TELEA A.: Feature sensitive multiscale editing on surfaces. *The Visual Computer 20*, 5 (2004), 329–343. 2

[CLB*09] CHUANG M., LUO L., BROWN B., RUSINKIEWICZ S., KAZHDAN M.: Estimating the Laplace-Beltrami operator by restricting 3D functions. *Computer Graphics Forum (Symposium on Geometry Processing)* (2009), 1475–1484. 1, 2, 3, 5, 6, 9

[DMSB99] DESBRUN M., MEYER M., SCHRÖDER P., BARR A.: Implicit fairing of irregular meshes using diffusion and curvature flow. In *ACM SIGGRAPH Conference Proceedings* (1999), pp. 317–324. 1, 2, 3

[DT07] DAY D., TAYLOR M.: A new 11 point degree 6 cubature formula for the triangle. *Sixth International Congress on Industrial Applied Mathematics (ICIAM07) and GAMM Annual Meeting 7* (2007), 1022501–1022502. 5, 8

[Dzi88] DZIUK G.: Finite elements for the Beltrami operator on arbitrary surfaces. In *Partial Differential Equations and Calculus of Variations, Lecture Notes in Mathematics*, vol. 1357. 1988, pp. 142–155. 6

[EPT*07] ECKSTEIN I., PONS J., TONG Y., KUO C., DESBRUN M.: Generalized surface flows for mesh processing. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing* (2007), pp. 183–192. 2

[GPR*04] GRIEBEL M., PREUSSER T., RUMPF M., SCHWEITZER M. A., TELEA A.: Flow field clustering via algebraic multigrid. In *Visualization* (2004), IEEE Computer Society, pp. 35–42. 2

[HP04] HILDEBRANDT K., POLTHIER K.: Anisotropic filtering of non-linear surface features. *Computer Graphics Forum 23* (2004), 391–400. 1, 2

[JKG07] JIN M., KIM J., GU X.: Discrete surface Ricci flow: Theory and applications. In *IMA Conference on the Mathematics of Surfaces* (2007), pp. 209–232. 1, 2

[KCVS98] KOBBELT L., CAMPAGNA S., VORSATZ J., SEIDEL H.: Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), pp. 105–114. 2

[KY08] KORNHUBER R., YSERENTANT H.: Multigrid methods for discrete elliptice problems on triangular surfaces. *Computing and Visualization in Science 11* (2008), 251–257. 2

[LIS10] LIS V.1.2.52: `http://www.ssisc.org/lis/`, 2010. 6

[Met07] METRO V.4.07: `http://vcg.soureforge.net/index.php/metro`, 2007. 6

[PP93] PINKALL U., POLTHIER K.: Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics 2* (1993), 15–36. 8, 9

[RL03] RAY N., LEVY B.: Hierarchical least squares conformal map. In *Pacific Graphics* (2003), p. 263. 2

[RS87] RUGE J., STUEBEN K.: Algebraic multigrid. 73–130. 2

[SBZ09] SHI X., BAO H., ZHOU K.: Out-of-core multigrid solver for streaming meshes. *ACM Transactions on Graphics (SIGGRAPH Asia '09) 28*, 5 (2009). 2

[SCOL*04] SORKINE O., COHEN-OR D., LIPMAN Y., ROSSL C., SEIDEL H.: Laplacian surface editing. In *Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (2004), pp. 179–188. 2

[SK01] SCHNEIDER R., KOBBELT L.: Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design 18* (2001), 359–379. 2

[SYBF06] SHI L., YU Y., BELL N., FENG W.-W.: A fast multigrid algorithm for mesh deformation. *ACM Transactions on Graph (SIGGRAPH '06) 25*, 3 (2006), 1108–1117. 2, 9

[XPB06] XU G., PAN Q., BAJAJ C.: Discrete surface modeling using partial differential equations. *Computer Aided Geometric Design 23* (2006), 125–145. 2

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with Poisson-based gradient field manipulation. *ACM Transactions on Graphics (SIGGRAPH '04) 23* (2004), 644–651. 2

[ZGX06] ZHAO H., G., XU: Triangular surface mesh fairing via Gaussian curvature flow. *Journal of Computational and Applied Mathematics 195* (2006), 300–311. 2