Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow

Mathieu Desbrun Mark Meyer Peter Schröder Alan H. Barr

Caltech*

Abstract

In this paper, we develop methods to rapidly remove rough features from irregularly triangulated data intended to portray a smooth surface. The main task is to remove undesirable noise and uneven edges while retaining desirable geometric features. The problem arises mainly when creating high-fidelity computer graphics objects using imperfectly-measured data from the real world.

Our approach contains three novel features: an *implicit integration* method to achieve efficiency, stability, and large time-steps; a scale-dependent Laplacian operator to improve the diffusion process; and finally, a robust curvature flow operator that achieves a smoothing of the shape itself, distinct from any parameterization. Additional features of the algorithm include automatic exact volume preservation, and hard and soft constraints on the positions of the points in the mesh.

We compare our method to previous operators and related algorithms, and prove that our curvature and Laplacian operators have several mathematically-desirable qualities that improve the appearance of the resulting surface. In consequence, the user can easily select the appropriate operator according to the desired type of fairing. Finally, we provide a series of examples to graphically and numerically demonstrate the quality of our results.

1 Introduction

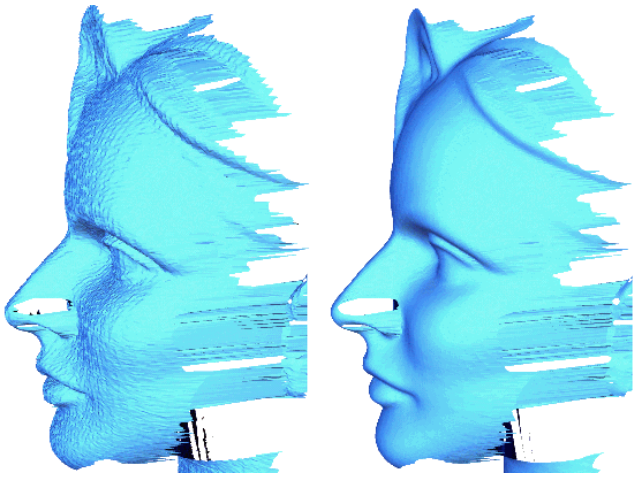
While the mainstream approach in mesh fairing has been to enhance the smoothness of triangulated surfaces by minimizing computationally expensive functionals, Taubin [Tau95] proposed in 1995 a signal processing approach to the problem of fairing arbitrary topology surface triangulations. This method is linear in the number of vertices in both time and memory space; large arbitrary connectivity meshes can be handled quite easily and transformed into visually appealing models. Such meshes appear more and more frequently due to the success of 3D range sensing approaches for creating complex geometry [CL96].

Taubin based his approach on defining a suitable generalization of frequency to the case of arbitrary connectivity meshes. Using a discrete approximation to the Laplacian, its eigenvectors become the "frequencies" of a given mesh. Repeated application of the resulting linear operator to the mesh was then employed to tailor the frequency content of a given mesh.

Closely related is the approach of Kobbelt [Kob97], who considered similar discrete approximations of the Laplacian in the construction of fair interpolatory subdivision schemes. In later work this was extended to the arbitrary connectivity setting for purposes of multiresolution editing [KCVS98].

The success of these techniques is largely based on their simple implementation and the increasing need for algorithms which can process the ever larger meshes produced by range sensing techniques. However, a number of issues in their application remain open problems in need of a more thorough examination.

*{mathieu|mmeyer|ps|barr}@cs.caltech.edu.



(a) (b)

Figure 1: *(a): Original 3D photography mesh (41,000 vertices). (b): Smoothed version with the scale-dependent operator in two integration step with $\lambda dt = 5 \cdot 10^{-5}$, the iterative linear solver (PBCG) converges in 10 iterations. All the images in this paper are flat-shaded to enhance the faceting effect.*

The simplicity of the underlying algorithms is based on very basic, uniform approximations of the Laplacian. For irregular connectivity meshes this leads to a variety of artifacts such as geometric distortion during smoothing, numerical instability, problems of slow convergence for large meshes, and insufficient control over global behavior. The latter includes shrinkage problems and more precise shaping of the frequency response of the algorithms.

In this paper we consider more carefully the question of numerical stability by observing that Laplacian smoothing can be thought of as time integration of the heat equation on an irregular mesh. This suggests the use of *implicit integration* schemes which lead to unconditionally stable algorithms allowing for very large time steps. At the same time the necessary linear system solvers run faster than explicit approaches for large meshes. We also consider the question of mesh parameterization more carefully and propose the use of discretizations of the Laplacian which take the underlying parameterization into account. The resulting algorithms avoid many of the distortion artifacts resulting from the application of previous methods. We demonstrate that this can be done at only a modest increase in computing time and results in smoothing algorithms with considerably higher geometric fidelity. Finally a more careful analysis of the underlying discrete differential geometry is used to derive a curvature flow approach which satisfies crucial geometric properties. We detail how these different operators act on meshes, and how users can then decide which one is appropriate in their case. If the user wants to, at the same time, smooth the shape of an object and equalize its triangulation, a scale-dependent diffusion must be used. On the other hand, if only the shape must be filtered without affecting the sampling rate, then curvature flow has all the desired properties. This allows us to propose a novel class of efficient smoothing algorithms for arbitrary connectivity meshes.

2 Implicit fairing

In this section, we introduce *implicit fairing*, an implicit integration of the diffusion equation for the smoothing of meshes. We will demonstrate several advantages of this approach over the usual ex-

plicit methods. While this section is restricted to the use of a linear approximation of the diffusion term, implicit fairing will be used as a robust and efficient numerical method throughout the paper, even for non-linear operators. We start by setting up the framework and defining our notation.

2.1 Notation and definitions

In the remainder of this paper, X will denote a mesh, x_i a vertex of this mesh, and e_{ij} the edge (if existing) connecting x_i to x_j. We will call $N_1(i)$ the "neighbors" (or 1-ring neighbors) of x_i, i.e., all the vertices x_j such that there exists an edge e_{ij} between x_i and x_j (see Figure 9(a)).

In the surface fairing literature, most techniques use constrained energy minimization. For this purpose, different fairness functionals have been used. The most frequent functional is the total curvature of a surface $\mathcal{S}$:

$$\mathcal{E}(\mathcal{S}) = \int_{\mathcal{S}} \kappa_1^2 + \kappa_2^2 \, d\mathcal{S}. \tag{1}$$

This energy can be estimated on discrete meshes [WW94, Kob97] by fitting local polynomial interpolants at vertices. However, principal curvatures κ_1 and κ_2 depend non-linearly on the surface $\mathcal{S}$. Therefore, many practical fairing methods prefer the membrane functional or the thin-plate functional of a mesh X:

$$\mathcal{E}_{membrane}(X) = \frac{1}{2}\int_{\Omega} X_u^2 + X_v^2 \; dudv \tag{2}$$

$$\mathcal{E}_{thin\ plate}(X) = \frac{1}{2}\int_{\Omega} X_{uu}^2 + 2X_{uv}^2 + X_{vv}^2 \; dudv. \tag{3}$$

Note that the thin-plate energy turns out to be equal to the total curvature only when the parameterization (u,v) is isometric. Their respective variational derivatives corresponds to the Laplacian and the second Laplacian:

$$\mathcal{L}(X) = X_{uu} + X_{vv} \tag{4}$$

$$\mathcal{L}^2(X) = \mathcal{L} \circ \mathcal{L}(X) = X_{uuuu} + 2X_{uuvv} + X_{vvvv}. \tag{5}$$

For smooth surface reconstruction in vision, a weighted average of these derivatives has been used to fair surfaces [Ter88]. For meshes, Taubin [Tau95] used signal processing analysis to show that a combination of these two derivatives of the form: $(\lambda+\mu)\mathcal{L} - \lambda\mu\mathcal{L}^2$ can provide a Gaussian filtering that minimizes shrinkage. The constants λ and μ must be tuned by the user to obtain this non-shrinking property. We will refer to this technique as the $\lambda|\mu$ algorithm.

2.2 Diffusion equation for mesh fairing

As we just pointed out, one common way to attenuate noise in a mesh is through a *diffusion process*:

$$\frac{\partial X}{\partial t} = \lambda \mathcal{L}(X). \tag{6}$$

By integrating equation 6 over time, a small disturbance will disperse rapidly in its neighborhood, smoothing the high frequencies, while the main shape will be only slightly degraded. The Laplacian operator can be linearly approximated at each vertex by the umbrella operator (we will use this approximation in the current section for the sake of simplicity, but will discuss its validity in section 4), as used in [Tau95, KCVS98]:

$$\mathcal{L}(x_i) = \frac{1}{m}\sum_{j \in N_1(i)} x_j - x_i \tag{7}$$

where x_j are the neighbors of the vertex x_i, and $m = \#N_1(i)$ is the number of these neighbors (valence). A sequence of meshes (X^n) can be constructed by integrating the diffusion equation with a simple *explicit Euler* scheme, yielding:

$$X^{n+1} = (I + \lambda dt\, \mathcal{L})X^n. \tag{8}$$

With the umbrella operator, the stability criterion requires $\lambda dt < 1$. If the time step does not satisfy this criterion, ripples appear on the surface, and often end up creating oscillations of growing magnitude over the whole surface. On the other hand, if this criterion is met, we get smoother and smoother versions of the initial mesh as n grows.

2.3 Time-shifted evaluation

The implementation of this previous explicit method, called *forward Euler method*, is very straightforward [Tau95] and has nice properties such as linear time and linear memory size for each filtering pass. Unfortunately, when the mesh is large, the time step restriction results in the need to perform hundreds of integrations to produce a noticeable smoothing, as mentioned in [KCVS98].

Implicit integration offers a way to avoid this time step limitation. The idea is simple: if we approximate the derivative using the new mesh (instead of using the old mesh as done in explicit methods), we will get to the equilibrium state of the PDE faster. As a result of this time-shifted evaluation, stability is obtained unconditionally [PTVF92]. The integration is now: $X^{n+1} = X^n + \lambda dt\, \mathcal{L}(X^{n+1})$. Performing an implicit integration, this time called *backward Euler method*, thus means solving the following linear system:

$$(I - \lambda dt\, \mathcal{L})X^{n+1} = X^n. \tag{9}$$

This apparently minor change allows the user not to worry about practical limitations on the time step. Consequent smoothing will then be obtained safely by increasing the value λdt. But solving a linear system is the price to pay.

2.4 Solving the sparse linear system

Fortunately, this linear system can be solved efficiently as the matrix $A = I - \lambda dt\, \mathcal{L}$ is sparse: each line contains approximately six non-zero elements if the Laplacian is expressed using Equ. (7) since the average number of neighbors on a typical triangulated mesh is six. We can use a preconditioned bi-conjugate gradient (PBCG) to iteratively solve this system with great efficiency[1]. The PBCG is based on matrix-vector multiplies [PTVF92], which only require linear time computation in our case thanks to the sparsity of the matrix A. We review in Appendix A the different options we chose for the PBCG in order to have an efficient implementation for our purposes.

2.5 Interpretation of the implicit integration

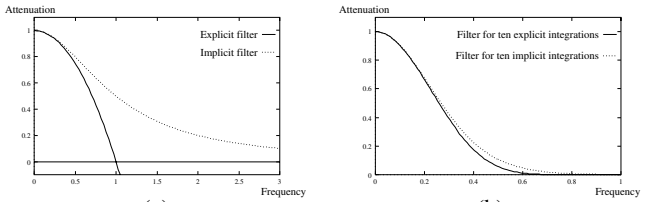
Although this implicit integration for diffusion is sound as is, there are useful connections with other prior work. We review the analogies with signal processing approaches and physical simulation.

2.5.1 Signal processing

In [Tau95], Taubin presents the explicit integration of diffusion with a signal processing point of view. Indeed, if X is a 1D signal of a given frequency ω: $X = e^{i\omega}$, then $\mathcal{L}(X) = -\omega^2 X$. Thus, the transfer function for Equ. (8) is $1 - \lambda dt\,\omega^2$, as displayed in Figure 2(a) as a solid line. We can see that the higher the frequency ω, the stronger the attenuation will be, as expected.

The previous filter is called FIR (for Finite Impulse Response) in signal processing. When the diffusion process is integrated using implicit integration, the filter in Equ. (9) turns out to be an Infinite Impulse Response filter. Its transfer function is now $1/(1+\lambda dt\,\omega^2)$, depicted in Figure 2(a) as a dashed line. Because this filter is always in $[0,1]$, we have unconditional stability.

[1] We use a bi-conjugate gradient method to be able to handle non symmetric matrices, to allow the inclusion of constraints (see Section 2.7).



(a) (b)

Figure 2: *Comparison between (a) the explicit and implicit transfer function for $\lambda dt = 1$, and (b) their resulting transfer function after 10 integrations.*

By rewriting Equ. (9) as: $X^{n+1} = (I - \lambda dt\, \mathcal{L})^{-1} X^n$, we also note that our implicit filtering is equivalent to $I + \lambda dt\, \mathcal{L} + (\lambda dt)^2 \mathcal{L}^2 + ...$, i.e., standard explicit filtering plus an infinite sequence of higher order filtering. Contrary to the explicit approach, one single implicit filtering step performs global filtering.

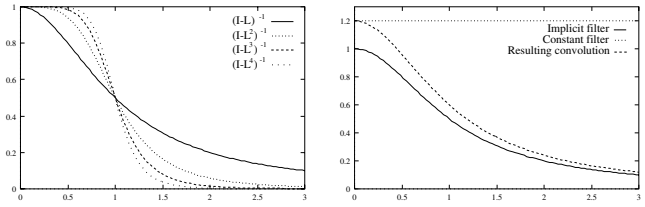
2.5.2 Mass-spring network

Smoothing a mesh by minimizing the membrane functional can be seen as a physical simulation of a mass-spring network with zero-rest length springs that will shrink to a single point in the limit. Recently, Baraff and Witkin [BW98] presented an implicit method to allow large time steps in cloth simulation. They found that the use of an implicit solver instead of the traditional explicit Euler integration considerably improves computational time while still being stable for very stiff systems. Our method compares exactly to theirs, but used for meshes and for a different PDE. We therefore have the same advantages of using an implicit solver over the usual explicit type: *stability* and *efficiency* when significant filtering is called for.

2.6 Filter improvement

Now that the method has been set up for the usual diffusion equation, we can consider other equations that may be more appropriate or may give better visual results for smoothing when we use implicit integration.

We have seen in Section 2.1 that both $\mathcal{L}$ and $\mathcal{L}^2$ have been used with success in prior work [Ter88, Tau95, KCVS98]. When we use implicit integration, as Figure 3(a) shows, the higher the power of the Laplacian, the closer to a *low-pass filter* we get. In terms of frequency analysis, it is a better filter. Unfortunately, the matrix becomes less and less sparse as more and more neighbors are involved in the computation. In practice, we find that $\mathcal{L}^2$ is a very good trade-off between efficiency and quality. Using higher orders affects the computational time significantly, while not always producing significant improvements. We therefore recommend using $(I + \lambda dt\, \mathcal{L}^2) X^{n+1} = X^n$ for implicit smoothing (a precise definition of the umbrella-like operator for $\mathcal{L}^2$ can be found in [KCVS98]).



(a) (b)

Figure 3: *(a): Comparison between filters using $\mathcal{L}$, $\mathcal{L}^2$, $\mathcal{L}^3$, and $\mathcal{L}^4$. (b): The scaling to preserve volume creates an amplification of all frequencies; but the resulting filter (diffusion+scaling) only amplifies low frequencies to compensate for the shrinking of the diffusion.*

We also tried to use a linear combination of both $\mathcal{L}$ and $\mathcal{L}^2$. We obtained interesting results like, for instance, amplification of low or middle frequencies to exaggerate large features (refer to [GSS99] for a complete study of feature enhancement). It is not appropriate in the context of a fixed mesh, though: amplifying frequencies requires refinement of the mesh to offer a good discretization.

2.7 Constraints

We can put hard and soft constraints on the mesh vertex positions during the diffusion. For the user, it means that a vertex or a set of vertices can be fixed so that the smoothing happens only on the rest of the mesh. This can be very useful to retain certain details in the mesh.

A vertex x_i will stay fixed if we impose $\mathcal{L}(x_i) = 0$. More complicated constraints are also possible [BW98]. For example, vertices can be constrained along an axis or on a plane by modifying the PBCG to keep these constraints enforced during the linear solver iterations.

We can also easily implement *soft constraints*: each vertex can be weighted according to the desired smoothing that we want. For instance, the user may want to smooth a part of a mesh less than another one, in order to keep desirable features while getting a smoother version. We allow the assignment of a smoothing value between 0 and 1 to attenuate the smoothing spatially: this is equivalent to choosing a variable λ factor on the mesh, and happens to be very useful in practice. Entire regions can be "spray painted" interactively to easily assign this special factor.

2.8 Discussion

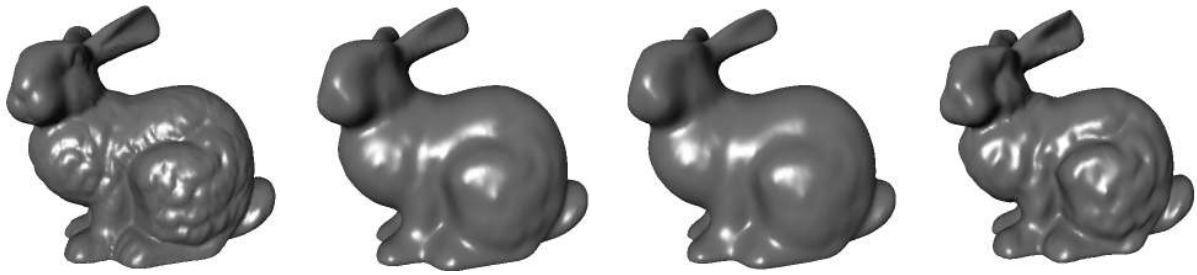
Even if adding a linear solver step to the integration of the diffusion equation seems to slow down the problem at first glance, it turns out that we gain significantly by doing so. For instance, the implicit integration can be performed with an arbitrary time step. Since the matrix of the system is very sparse, we actually obtain computational time similar or better than the explicit methods. In the following table, we indicate the number of iterations of the PBCG method for different meshes and it can be seen that the PBCG is more efficient when the smoothing is high. These timings were performed on an SGI High Impact Indigo2 175MHz R10000 processor with 128M RAM.

Mesh	Nb of faces	$\lambda dt = 10$	$\lambda dt = 100$
Horse	42,000	8 iterations (2.86s)	37 iterations (12.6s)
Dragon	42,000	8 iterations (2.98s)	39 iterations (13.82s)
Isis	50,000	9 iterations (3.84s)	37 iterations (15.09s)
Bunny	66,000	7 iterations (4.53s)	35 iterations (21.34s)
Buddha	290,000	5 iterations (13.78s)	28 iterations (69.93s)

To be able to compare the results with the explicit method, one has to notice that one iteration of the PBCG is only slightly more time consuming than one integration step using an explicit method. Therefore, we can see in the following results that our implicit fairing takes about 60% less time than the explicit fairing for a filtering of $\lambda dt = 100$, as we get about 33 iterations compared to the 100 integration steps required in the explicit case. We have found this behavior to be true for all the other meshes as well. The advantage of the implicit method in terms of computational speed becomes more obvious for *large meshes* and/or *high smoothing* value. In terms of quality, Figure 4(b) and 4(c) demonstrate that both implicit and explicit methods produce about the same visual results, with a slightly better smoothness for the implicit fairing. Note that we use 10 explicit integrations of the umbrella operator with $\lambda dt = 1$, and 1 integration using the implicit integration with $\lambda dt = 10$ to approximate the same results. Therefore, there is a definite advantage in the use of implicit fairing over the previous explicit methods. Moreover, the remainder of this paper will make heavy use of this method and its stability properties.

3 Automatic anti-shrinking fairing

Pure diffusion will, by nature, induce shrinkage. This is inconvenient as this shrinking may be significant for aggressive smoothing. Taubin proposed to use a linear combination of $\mathcal{L}$ and $\mathcal{L} \circ \mathcal{L}$ to amplify low frequencies in order to balance the natural shrinking. Unfortunately, the linear combination depends heavily on the mesh in practice, and this requires fine tuning to ensure both stable



(a) (b) (c) (d)

Figure 4: *Stanford bunnies: (a) The original mesh, (b) 10 explicit integrations with $\lambda dt = 1$, (c) 1 implicit integration with $\lambda dt = 10$ that takes only 7 PBCG iterations (30% faster), and (d) 20 passes of the $\lambda|\mu$ algorithm, with $\lambda = 0.6307$ and $\mu = -0.6732$. The implicit integration results in better smoothing than the explicit one for the same, or often less, computing time. If volume preservation is called for, our technique then requires many fewer iterations to smooth the mesh than the $\lambda|\mu$ algorithm.*

and non-shrinking results. In this section, we propose an automatic solution to avoid this shrinking. We preserve the zeroth moment, i.e., the volume, of the object. Without any other information on the mesh, we feel it is the most reasonable invariant to preserve, although surface area or other invariants can be used.

3.1 Volume computation

As we have a mesh given in terms of triangles, it is easy to compute the interior volume. This can be done by summing the volumes of all the oriented pyramids centered at a point in space (the origin, for instance) and with a triangle of the mesh as a base. This computation has a linear complexity in the number of triangles [LK84]. For the reader's convenience, we give the expression of the volume of a mesh in the following equation, where x_k^1, x_k^2 and x_k^3 are the three vertices of the kth triangle:

$$V = \frac{1}{6} \sum_{k=1}^{nbFaces} g_k \cdot N_k \qquad (10)$$

$$\text{where } g = (x_k^1 + x_k^2 + x_k^3)/3 \text{ and } N_k = \vec{x_k^1 x_k^2} \wedge \vec{x_k^1 x_k^3}$$

3.2 Exact volume preservation

After an integration step, the mesh will have a new volume V^n. We then want to scale it back to its original volume V^0 to cancel the shrinking effect. We apply a simple scale on the vertices to achieve this. By multiplying all the vertex positions by $\beta = (V^0/V^n)^{1/3}$, the volume is guaranteed to go back to its original value. As this is a simple scaling, it is harmless in terms of frequencies. To put it differently, this scaling corresponds to a convolution with a scaled Dirac in the frequency domain, hence it amplifies all the frequencies in the same way to change the volume back. The resulting filter, after the implicit smoothing and the constant amplification filter, amplifies the low frequencies of the original mesh to *exactly* compensate for the attenuation of the high frequencies, as sketched on Figure 3(b).

The overall complexity for volume preservation is then linear. With such a process, we do not need to tweak parameters: the anti-shrinking filter is *automatically* adapted to the mesh and to the smoothing, contrary to previous approaches. Note that hard constraints defined in the previous section are applied before the scaling and do not result in fixed points anymore: scaling alters the absolute, but not the relative position.

We can generalize this re-scaling phase to different invariants. For instance, if we have to smooth height fields, it is more appropriate to take the invariant as being the volume enclosed between the height field and a reference plane, which changes the computations only slightly. Likewise, for surfaces of revolution, we may change the way the scaling is computed to exploit this special property. We can also preserve the surface area if the mesh is a non-closed surface. However, in the absence of specific characteristics, preserving the volume gives nice results. According to specific needs, the user can select the appropriate type of invariant to be used.

3.3 Discussion

When we combine both methods of implicit integration and anti-shrinking convolution, we obtain an automatic and efficient method for fairing. Indeed, no parameters need be tuned to ensure stability or to have exact volume preservation. This is a major advantage over previous techniques. Yet, we retain all of the advantages of previous methods, such as constraints [Tau95] and the possibility of accelerating the fairing via multigrid [KCVS98], while additionally offering stability and efficiency. This technique also dramatically reduces the computing time over Taubin's anti-shrinking algorithm: as demonstrated in Figure 4(c) and 4(d), using the $\lambda|\mu$ algorithm may preserve the volume after fine tuning, but one iteration will only slightly smooth the mesh. The rest of this paper exploits both automatic anti-shrinking and implicit fairing techniques to offer more accurate tools for fairing.

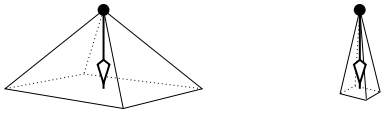
4 An accurate diffusion process

Up to this section, we have relied on the umbrella operator (Equ. (7)) to approximate the Laplacian on a vertex of the mesh. This particular operator does not truly represent a Laplacian in the physical meaning of this term as we are about to see. Moreover, simple experiments on smooth meshes show that this operator, using explicit or implicit integration, can create bumps or "pimples" on the surface, instead of smoothing it. This section proposes a sounder simulation of the diffusion process, by defining a new approximation for the Laplacian and by taking advantage of the implicit integration.

4.1 Inadequacy of the umbrella operator

The umbrella operator, used in the previous sections corresponds to an approximation of the Laplacian in the case of a specific parameterization [KCVS98]. This means that the mesh is supposed to have edges of length 1 and all the angles between two adjacent edges around a vertex should be equal. This is of course far from being true in actual meshes, which contain a variety of triangles of different sizes.

Treating all edges as if they had equal length has significant undesired consequences for the smoothing. For example, the Laplacian can be the same for two very different configurations, corresponding to different frequencies as depicted in Figure 5. This distorts the filtering significantly, as high frequencies may be considered as low ones, and vice-versa. Nevertheless, the advantage of the umbrella operator is that it is normalized: the time step for integration is always 1, which is very convenient. But we want a more accurate diffusion process to smooth meshes consistently, in order to more carefully separate high from low frequencies.



(a) (b)

Figure 5: *Frequency confusion: the umbrella operator is evaluated as the vector joining the center vertex to the barycenter of its neighbors. Thus, cases (a) and (b) will have the same approximated Laplacian even if they represent different frequencies.*

We need to define a discrete Laplacian which is scale dependent, to better approximate diffusion. However, if we use explicit integration [Tau95], we will suffer from a very restricted stability criterion. It is well known [PTVF92] that the time step for a parabolic PDE like Equ. (6) depends on the square of the smallest length scale (here, the smallest edge length $min(|e|)$):

$$dt \leq \frac{min(|e|)^2}{2\,\lambda}$$

This limitation is a real concern for large meshes with small details, since an enormous number of integration steps will have to be performed to obtain noticeable smoothing. This is *intractable* in practice.

With implicit integration explained in Section 2, we can overcome this restriction and use a much larger time step while still achieving good smoothing, saving considerable computation. In the next two paragraphs we present one design of a good approximation for the Laplacian.

4.2 Simulation of the 1D heat equation

The 1D case of a diffusion equation corresponds to the heat equation $x_t = x_{uu}$. It is therefore worth considering this example as a test problem for higher dimensional filtering. To do so, we use Milne's test presented in [Mil95]. Milne compared two cases of the same initial problem: first, the problem is solved on a regular mesh on $[0,1]$, and then on an irregular mesh, taken to consist of a uniform coarse grid of cells on $[0,1]$ with each of the cells in $[\frac{1}{2},1]$ subdivided into two fine cells as depicted in Figure 6(a) and 6(b). With such a configuration, classical finite difference coefficients for second derivatives can be used on each cell, except for the middle one which does not have centered neighbors. Milne shows that if no particular care is taken for this "peripheral" cell, it introduces a *noise term* that creates large inaccuracies — larger than if the mesh was represented uniformly at the coarser resolution! But if we fit a quadratic spline at this cell to approximate the second derivative, then the noise source disappears and we get more accurate results than with a constant coarse resolution (see the errors created in each case in one iteration of the heat equation in Figure 6(c)).

This actually corresponds to the extension of finite difference computations for irregular meshes proposed by Fornberg [For88]: to compute the FD coefficients, just fit a quadratic function at the sample point and its two immediate neighbors, and then return the first and second derivative of that function as the approximate derivatives. For three points spaced Δ and δ apart (see Figure 6(d)), we get the 1D formula:

$$(x_{uu})_i = \frac{2}{\delta+\Delta}\left(\frac{x_{i-1}-x_i}{\delta}+\frac{x_{i+1}-x_i}{\Delta}\right).$$

Note that when $\Delta = \delta$, we find the usual finite difference formula.

4.3 Extension to 3D

The umbrella operator suffers from this problem of large inaccuracies for irregular meshes as the same supposedly constant parameterization is used (Figure 7 shows such a behavior). Surprisingly, a simple generalization of the previous formula valid in 1D corresponds to a known approximation of the Laplacian. Indeed, Fujiwara [Fuj95] presents the following formula:

$$\mathcal{L}(x_i) = \frac{2}{E}\sum_{j\in N_1(i)}\frac{x_j - x_i}{|e_{ij}|}, \quad \text{with } E = \sum_{j\in N_1(i)}|e_{ij}|. \qquad (11)$$

where $|e_{ij}|$ is the length of the edge e_{ij}. Note that, when all edges are of size 1, this reduces to the umbrella operator (7). We will then denote this new operator as the *scale-dependent umbrella operator*.

Unfortunately, the operator is no longer linear. But during a typical smoothing, the length of the edges does not change dramatically. We thus make the approximation that the coefficients of the matrix $A = (I - \lambda dt\,\mathcal{L})$ stay constant during an integration step. We can compute them initially using the current edges' lengths and keep their values constant during the PBCG iterations. In practice, we have not noted any noticeable drawbacks from this linearization. We can even keep the same coefficients for a number of (or all) iterations: it will correspond to a filtering "relative" to the initial mesh instead if the current mesh. For the same reason as before, we also recommend the use of the second Laplacian for higher quality smoothing without significant increase in computation time. As demonstrated in Figure 7, the scale-dependent umbrella operator deals better with irregular meshes than the umbrella operator: no spurious artifacts are created. We also applied this operator to noisy data sets from 3D photography to obtain smooth meshes (see Figure 1 and 12).

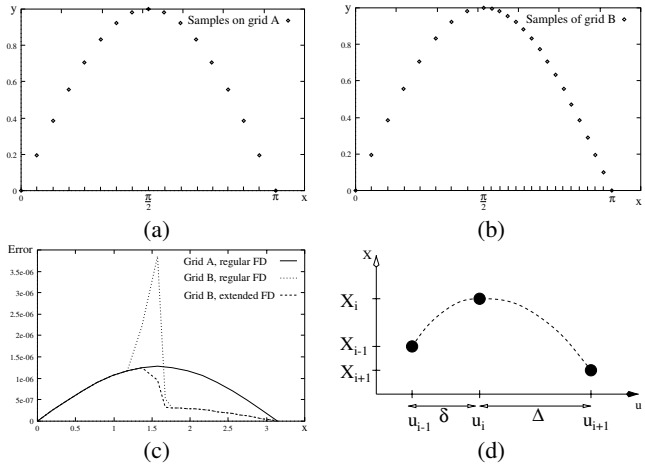


Figure 6: *Test on the heat equation: (a) regular sampling vs. (b) irregular sampling. Numerical errors in one step of integration (c): using the usual FD weight on an irregular grid to approximate second derivatives creates noise, and gives a worse solution than on the coarse grid, whereas extended FD weights offer the expected behavior. (d) Three unevenly spaced samples of a function and corresponding quadratic fitting for extended FD weights.*

The number of iterations needed for convergence depends heavily on the ratio between minimum and maximum edge lengths. For typical smoothing and for meshes over 50000 faces, the average number of iterations we get is 20. Nevertheless, we still observe undesired behavior on flat surfaces: vertices in flat areas still slide during smoothing. Even though this last formulation generally reduces this problem, we may want to keep a flat area *intact*. The next section tackles this problem with a new approach.

5 Curvature flow for noise removal

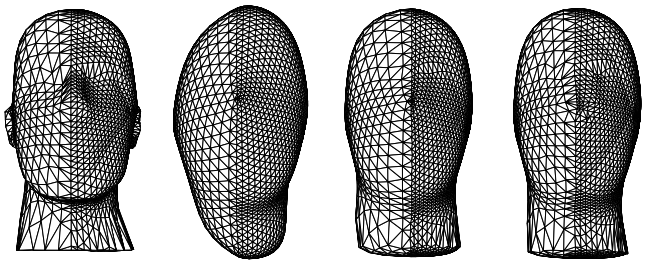
In terms of differential equations, diffusion is a close relative of curvature flow. In this section, we first explore the advantages of using curvature flow over diffusion, and then propose an efficient algorithm for noise removal using curvature flow.

5.1 Diffusion vs. curvature flow

The Laplacian of the surface at a vertex has both normal and tangential components. Even if the surface is locally flat, the Laplacian approximation will rarely be the zero vector [KCVS98]. This introduces undesirable drifting over the surface, depending on the parameterization we assume. We in effect fair the parameterization of the surface as well as the shape itself (see Figure 10(b)).

We would prefer to have a noise removal procedure that does not depend on the parameterization. It should use only *intrinsic properties* of the surface. This is precisely what curvature flow does. Curvature flow smoothes the surface by moving along the surface normal $\mathbf{n}$ with a speed equal to the mean curvature $\overline{\kappa}$:

$$\frac{\partial x_i}{\partial t} = -\overline{\kappa}_i\,\mathbf{n}_i. \qquad (12)$$



(a) (b) (c) (d)

Figure 7: *Application of operators to a mesh: (a) mesh with different sampling rates, (b) the umbrella operator creates a significant distortion of the shape, but (c) with the scale-dependent umbrella operator, the same amount of smoothing does not create distortion or artifacts, almost like (d) when curvature flow is used. The small features such as the nose are smoothed but stay in place.*

Other curvatures can of course be used, but we will stick to the mean curvature: $\overline{\kappa} = (\kappa_1 + \kappa_2)/2$ in this paper. Using this procedure, a sphere with different sampling rates should stay spherical under curvature flow as the curvature is constant. And we should also not get any vertex "sliding" when an area is flat as the mean curvature is then zero.

There are already different approaches using curvature flow [Set96], and even mixing both curvature flow and volume preservation [DCG98] to smooth object appearance, but mainly in the context of level-set methods. They are not usable on a mesh as is. Next, we show how to approximate curvature consistently on a mesh and how to implement this curvature flow process with our implicit integration for efficient computations.

5.2 Curvature normal calculation

It seems that all the formulations so far have a non-zero tangential component on the surface. This means that even if the surface is flat around a vertex, it may move anyway. For curvature flow, we don't want this behavior. A good idea is to check the divergence of the normal vector, as it is the definition of mean curvature ($\overline{\kappa} = div\ \mathbf{n}$): if all the normals of the faces around a vertex are the same, this vertex should not move then (zero curvature). Having this in mind, we have selected the following differential geometry definition of the *curvature normal* $\overline{\kappa}\,\mathbf{n}$:

$$\frac{\nabla \mathcal{A}}{2\,\mathcal{A}} = \overline{\kappa}\,\mathbf{n} \tag{13}$$

where $\mathcal{A}$ is the area of a small region around the point P where the curvature is needed, and ∇ is the derivative with respect to the (x,y,z) coordinates of P. With this definition, we will have the zero vector for a flat area. As proven in Figure 8, we see that moving the center vertex x_i on a flat surface does not change the surface area. On the other hand, moving it above or below the plane will always increase the local area. Hence, we have the desired property of a null area gradient for a locally flat surface, whatever the valence, the aspect ratio of the adjacent faces, or the edge lengths around the vertex.

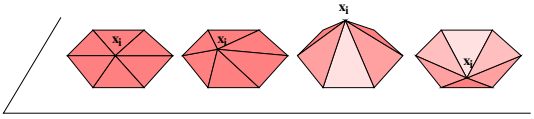
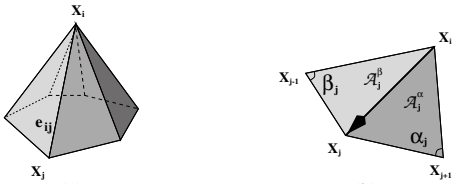


Figure 8: *The area around a vertex x_i lying in the same plane as its 1-ring neighbors does not change if the vertex moves within the plane, and can only increase otherwise. Being a local minimum, it thus proves that the derivative of the area with respect to the position of x_i is zero for flat regions.*

To derive the discrete version of this curvature normal, we select the smallest area around a vertex x_i that we can get, namely the area of all the triangles of the 1-ring neighbors as sketched in Figure 9(a). Note that this area $\mathcal{A}$ uses cross products of adjacent edges, and thus implicitly contains information on local normal vectors. The complete derivation from the continuous formulation to the discrete case is shown in Appendix B. We find the following discrete expression through basic differentiation:

$$-\overline{\kappa}\,\mathbf{n} = \frac{1}{4\,\mathcal{A}} \sum_{j \in N_1(i)} (cot\ \alpha_j + cot\ \beta_j)(x_j - x_i) \tag{14}$$

where α_j and β_j are the two angles opposite to the edge in the two triangles having the edge e_{ij} in common (as depicted in Figure 9(b)), and $\mathcal{A}$ is the sum of the areas of the triangles having x_i as a common vertex.



(a) (b)

Figure 9: *A vertex x_i and its adjacent faces (a), and one term of its curvature normal formula (b).*

Note the interesting similarity with [PP93]. We obtain almost the same equation, but with a completely different derivation than theirs, which was using energies of linear maps. The same remark stands for [DCDS97] since they also find the same kind of expression as Equ. (14) for their functional, but using this time piecewise linear harmonic functions.

5.3 Boundaries

For non-closed surfaces or surfaces with holes, we can define a special treatment for vertices on boundaries. The notion of mean curvature does not make sense for such vertices. Instead, we would like to smooth the boundary, so that the shape of the hole itself gets rounder and rounder as iterations go. We can then use for instance Equ. (11) restricted to the two immediate neighbors which will smooth the boundary curve itself.

Another possible way is to create a virtual vertex, stored but not displayed, initially placed at the barycenter of all the vertices placed on a closed boundary. A set of faces adjacent to this vertex and connecting the boundary vertices one after the other are also virtually created. We can then use the basic algorithm without any special treatment for the boundary as now, each vertex has a closed area around it.

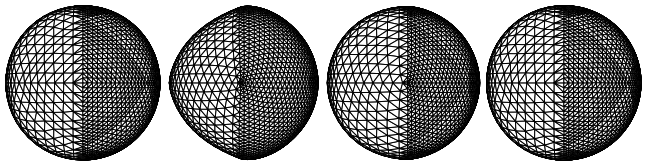
5.4 Implementation

Similarly to Section 4, we have a non-linear expression defining the curvature normal. We can however proceed in exactly the same way, as the changes induced in a time step will be small. We simply compute the non-zero coefficients of the matrix $I - \lambda dt K$, where K represents the matrix of the curvature normals. We then successively solve the following linear system:

$$(I - \lambda dt K)\, X^{n+1} = X^n.$$

We can use preconditioning or constraints, just as before as everything is basically the same except for the local approximation of the speed of smoothing. As shown on Figure 10, a sphere with different triangle sizes will remain the same sphere thanks to both the curvature flow and the volume preservation technique.

In order for the algorithm to be robust, an important test must be performed while the matrix K is computed: if we encounter a face of zero area, we must skip it. As we divide by the area of the face, degenerate triangles are to be treated specially. Mesh decimation to eliminate all degenerate triangles can also be used as suggested in [PP93].



(a) (b) (c) (d)

Figure 10: *Smoothing of spheres: (a) The original mesh containing two different discretization rates. (b) Smoothing with the umbrella operator introduces sliding of the mesh and unnatural deformation, which is largely attenuated when (c) the scale-dependent version is used, while (d) curvature flow maintains the sphere exactly.*

5.5 Normalized version of the curvature operator

We can now write the equivalent of the umbrella operator, but for the curvature normal. Since the new formulation has nice properties, we can create a normalized version that could be used in an explicit integration for quick smoothing. The normalization will bring the eigenvalues back in $[-1,0]$ so that a time step up to 1 can be used in explicit integration methods. Its expression is simply:

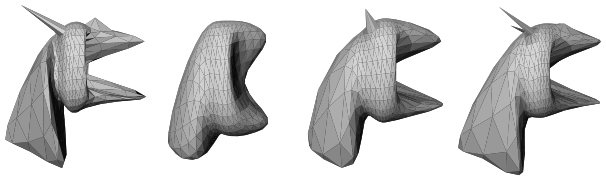
$$(\overline{\kappa}\,\mathbf{n})_{normalized} = \frac{1}{\sum_j (cot\ \alpha_j^l + cot\ \alpha_j^r)} \sum_j (cot\ \alpha_j^l + cot\ \alpha_j^r)(X_i - X_j)$$

5.6 Comparison of results

Figures 7, 10, and 11 compare the different operators we have used:

- For significant fairing, the umbrella operator changes the shape of the object substantially: triangles drift over the surface and tend to be uniformly distributed with an equal size.
- The scale-dependent umbrella operator allows the shape to stay closer to the original shape even after significant smoothing, and almost keeps the original distribution of triangle sizes.
- Finally, the curvature flow just described achieves the best smoothing with respect to the shape, as no drift happens and only geometric properties are used to define the motion.

Knowing these properties, the user can select the type of smoothing that fits best with the type of fairing that is desired. Diffusion will smooth the shape along with the parameterization, resulting in a more regular triangulation. If only the shape is to be affected, then the curvature operator should be used.



(a) (b) (c) (d)

Figure 11: *Significant smoothing of a dragon: (a) original mesh, (b) implicit fairing using the umbrella operator, (c) using the scale-dependent umbrella operator, and (d) using curvature flow.*

6 Discussion and conclusion

In this paper, we have presented a comprehensive set of tools for mesh fairing. We first presented an *implicit fairing* method, using implicit integration of a diffusion process that allows for both efficiency, quality, and stability. Additionally we guarantee volume preservation during smoothing. Since the umbrella operator used in the literature appears to have serious drawbacks, we defined a new scale-dependent umbrella operator to overcome undesired effects such as large distortions on irregular meshes. Finally, since using a diffusion process leads always to vertex "sliding" on the mesh, we developed a curvature flow process. The same implicit integration is used for this new operator that now offers a smoothing only depending on intrinsic geometric properties, without sliding on flat areas and with preserved curvature for constant curvature areas. The user can make use of all these different tools according to the mesh to be smoothed.

We believe the computational time for this approach can still be improved upon. We expect that multigrid preconditioning for the PBCG in the case of the scale-dependent operator for diffusion and for curvature flow would speed up the integration process. This multigrid aspect of mesh fairing has already been mentioned in [KCVS98], and could be easily extended to our method. Likewise, subdivision techniques can be directly incorporated into our method to refine or simplify regions according to curvature for instance. Other curvature flows, for example along the principal curvature directions, are also worth studying.

Acknowledgements

The original 3D photography mesh was provided by Jean-Yves Bouguet, the mannequin head and spock dataset by Hugues Hoppe, the bunny and buddha models by Stanford University, and additional test meshes by Cyberware. The authors would like to thank John T. Reese for the initial implementation and the dragon mesh, and Konrad Polthier for interesting comments. This work was supported by the Academic Strategic Alliances Program of the Accelerated Strategic Computing Initiative (ASCI/ASAP) under subcontract B341492 of DOE contract W-7405-ENG-48. Additional support was provided by NSF (ACI-9624957, ACI-9721349, DMS-9874082, and ASC-89-20219 (STC for Computer Graphics and Scientific Visualization)), Alias|wavefront and through a Packard Fellowship.

References

[Bar89] Alan H. Barr. The Einstein Summation Notation: Introduction and Extensions. In *SIGGRAPH 89 Course notes #30 on Topics in Physically-Based Modeling*, pages J1–J12, 1989.

[BW98] David Baraff and Andrew Witkin. Large Steps in Cloth Simulation. In *SIGGRAPH 98 Conference Proceedings*, pages 43–54, July 1998.

[CL96] Brian Curless and Marc Levoy. A Volumetric Method for Building Complex Models from Range Images. In *SIGGRAPH 96 Conference Proceedings*, pages 303–312, 1996.

[DCDS97] Tom Duchamp, Andrew Certain, Tony DeRose, and Werner Stuetzle. Hierarchical computation of PL harmonic embeddings. Technical report, University of Washington, July 1997.

[DCG98] Mathieu Desbrun and Marie-Paule Cani-Gascuel. Active Implicit Surface for Computer Animation. In *Graphics Interface (GI'98) Proceedings*, pages 143–150, Vancouver, Canada, 1998.

[For88] Bengt Fornberg. Generation of Finite Difference Formulas on Arbitrarily Spaced Grids. *Math. Comput.*, 51:699–706, 1988.

[Fuj95] Koji Fujiwara. Eigenvalues of Laplacians on a closed riemannian manifold and its nets. In *Proceedings of AMS 123*, pages 2585–2594, 1995.

[GSS99] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution Signal Processing for Meshes. In *SIGGRAPH 99 Conference Proceedings*, 1999

[KCVS98] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive Multi-Resolution Modeling on Arbitrary Meshes. In *SIGGRAPH 98 Conference Proceedings*, pages 105–114, July 1998.

[Kob97] Leif Kobbelt. Discrete Fairing. In *Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces '97*, pages 101–131, 1997.

[LK84] S. Lien and J. Kajiya. A Symbolic Method for Calculating the Integral Properties of Arbitrary Nonconvex Polyhedra. *IEEE CG&A*, 4(9), October 1984.

[Mil95] Roger B. Milne. An Adaptive Level-Set Method. *PhD Thesis*, University of California, Berkeley, December 1995.

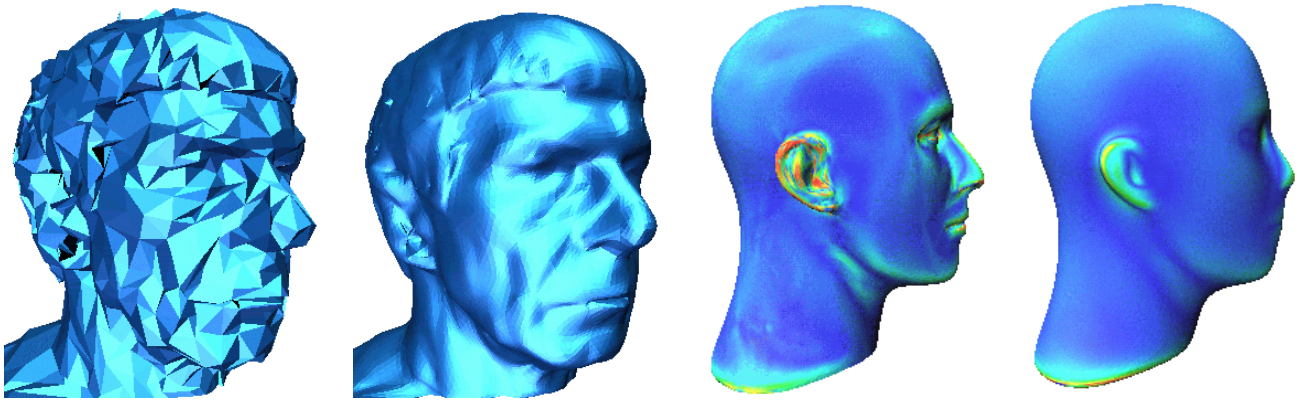
[PP93] Ulrich Pinkall and Konrad Polthier. Computing Discrete Minimal Surfaces and Their Conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[PTVF92] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. *Numerical Recipes in C, second edition*. Cambridge University Press, New York, USA, 1992.

[Set96] James A. Sethian. *Level-Set Methods: Evolving Interfaces in Geometry, Fluid Dynamics, Computer Vision, and Material Science*. Cambridge Monographs on Applied and Computational Mathematics, 1996.

[Tau95] Gabriel Taubin. A Signal Processing Approach to Fair Surface Design. In *SIGGRAPH 95 Conference Proceedings*, pages 351–358, August 1995.

[Ter88] Demetri Terzopoulos. The Computation of Visible-Surface Representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4), July 1988.



(a) (b) (c) (d)

Figure 12: *Faces: (a) The original decimated Spock mesh has 12,000 vertices. (b) We linearly oversampled this initial mesh (every visible triangle on (a) was subdivided in 16 coplanar smaller ones) and applied the scale-dependent umbrella operator, observing significant smoothing. One integration step was used, $\lambda dt = 10$, converging in 12 iterations of the PBCG. Similar results were achieved using the curvature operator. (c) curvature plot for the mannequin head (obtained using our curvature operator), (d) curvature plot of the same mesh after a significant implicit integration of curvature flow (pseudo-colors).*

[WW94] Willian Welch and Andrew Witkin. Free-form shape design using triangulated surfaces. In *SIGGRAPH 94 Conference Proceedings*, pages 247–256, July 1994.

Appendix

A Preconditioned Bi-Conjugate Gradient

In this section, we enumerate the different implementation choices we made for the PBCG linear solver.

A.1 Preconditioning

A good preconditioning, and particularly a multigrid preconditioning, can drastically improve the convergence rate of conjugate gradient solver. The umbrella operator (7) has all its eigenvalues in $[-1,0]$: in turn, the matrix A is always well conditioned for typical values of λdt. In practice, the simpler the conditioning the better. In our examples, we used the usual diagonal preconditioner $\tilde{A}$ with: $\tilde{A}_{ii} = 1/A_{ii}$, which provides a significant speedup with almost no overhead.

A.2 Convergence criterion

Different criteria can be used to test whether or not further iterations are needed to get a more accurate solution of the linear system. We opted for the following stopping criterion after several tests: $||AX^{n+1} - X^n|| < \varepsilon ||X^n||$, where $||.||$ can be either the L_2 norm, or, if high accuracy is needed, the L_∞ norm.

A.3 Memory requirements

An interesting remark is that we don't even need to store the matrix A in a dedicated data structure. The mesh itself provides a sparse matrix representation, as the vertex x_i and its neighbors are the only non-zero locations in A for row i. Computations can thus be carried directly within the mesh structure. Computing AX can be implemented by gathering values from the 1-ring neighbors of each vertex, while $A^T X$ can be achieved by "shooting" a value to the 1-ring neighbors.

With these simple setups, we obtain an efficient linear solver for the implicit integration described in Section 2.

B Curvature normal approximation

From the continuous definition of the curvature normal (Equ. (13)), we must derive a discrete formulation when the surface is given as a mesh. Let's consider a point P of the mesh. Its neighbors, in counterclockwise order around P, are the points $\{Q^n\}$. An adjacent face is then of the form P, Q^n, Q^{n+1}. The edge vector PQ^n is the difference between Q^n and P:

$$PQ^n = Q^n - P.$$

Now, we take the neighboring area as being the union of the adjacent faces. The total adjacent area $\mathcal{A}$ is then equal to the sum of every adjacent face's area: $\mathcal{A} = \sum_n \mathcal{A}_n$, the area of each adjacent face being: $\mathcal{A}_n = \frac{1}{2}||PQ^n \times PQ^{n+1}||$. So, using Einstein summation notation [Bar89], we have:

$$\mathcal{A}_n^2 = \frac{1}{4}\varepsilon_{ijk}\, PQ_j^n\, PQ_k^{n+1}\, \varepsilon_{ilm}\, PQ_l^n\, PQ_m^{n+1},$$

where ε_{ijk} is the permutation symbol. Using the Kronecker delta δ_{ij}, and using $\frac{\partial P_i}{\partial P_q} = \delta_{iq}$ as well as $\nabla = \partial/\partial P_q$, we derive:

$$\begin{aligned}\frac{\partial \mathcal{A}_i^2}{\partial P_q} &= 2\,\mathcal{A}_i\,\frac{\partial \mathcal{A}_i}{\partial P_q} \\ &= \frac{1}{4}\varepsilon_{ijk}\varepsilon_{ilm}\Big[-\delta_{jq}\,PQ_k^{n+1}\,PQ_l^n PQ_m^{n+1} - \delta_{kq}\,PQ_j^n\,PQ_l^n\,PQ_m^{n+1} \\ &\quad -\delta_{lq}\,PQ_j^n\,PQ_k^{n+1}\,PQ_m^{n+1} - \delta_{mq}\,PQ_j^n\,PQ_k^{n+1}\,PQ_l^{n+1}\Big]\end{aligned}$$

Using the ε-δ rule stating $\varepsilon_{ijk}\varepsilon_{ilm} = \delta_{jl}\delta_{km} - \delta_{jm}\delta_{kl}$, we obtain:

$$\begin{aligned}\frac{\partial \mathcal{A}_i^2}{\partial P_q} &= \frac{1}{2}\Big[-||PQ^{n+1}||^2\,PQ^n + (PQ^n\cdot PQ^{n+1})\,PQ^{n+1} \\ &\quad -||PQ^n||^2\,PQ^{n+1} + (PQ^{n+1}\cdot PQ^n)\,PQ^n\Big]_q \\ &= \frac{1}{2}\Big[(PQ^{n+1}\cdot Q^{n+1}Q^n)\,PQ^n + (PQ^n\cdot Q^nQ^{n+1})\,PQ^{n+1}\Big]_q.\end{aligned}$$

Consequently:

$$\frac{\partial \mathcal{A}_i}{\partial P} = \frac{1}{4\,\mathcal{A}_i}\Big((PQ^{n+1}\cdot Q^{n+1}Q^n)\,PQ^n + (PQ^n\cdot Q^nQ^{n+1})\,PQ^{n+1}\Big). \tag{15}$$

Using Equ. (13), we find:

$$\frac{\nabla \mathcal{A}}{2\,\mathcal{A}} = \frac{1}{2\,\mathcal{A}}\sum_i \frac{\partial \mathcal{A}_i}{\partial P} \tag{16}$$

From equations (15) and (16), we find the equations used in Section 5.2 since the dot product of PQ^n by Q^nQ^{n+1} divided by their cross product simplifies into a cotangent.