Localized bi-Laplacian Solver on a Triangle Mesh and Its Applications

ByungMoon Kim* Jarek Rossignac†

College of Computing
Georgia Institute of Technology

Abstract

Partial differential equations(PDE) defined over a surface are used in various graphics applications, such as mesh fairing, smoothing, surface editing, and simulation. Often these applications involve PDEs with Laplacian or bi-Laplacian terms. We propose a new approach to a finite element method for solving these PDEs that works directly on the triangle mesh connectivity graph that has more connectivity information than the sparse matrix. Thanks to these extra information in the triangle mesh, the solver can be restricted to operate on a sub-domain, which is a portion of the surface defined by user or automatically self-adjusting. Our formulation permits us to solve high order terms such as bi-Laplacian by using a simple linear triangle element. We demonstrate the benefits of our approach on two applications: scattered data interpolation over a triangle mesh(painting), and haptic interaction with a deformable surface.

Keywords: PDE, Triangle Mesh, Interpolation, Deformation, Laplace-Beltrami Operator

1 Introduction

Partial differential equations (PDEs) are used to solve a variety of problems in computer graphics, including haptic rendering, mesh fairing, and physically based simulation. Typical solvers use a finite difference (FDM) or a finite element method (FEM). To use FDM on a triangle mesh of arbitrary topology, one needs to compute a parameterization of the mesh that maps it onto a set of square grids[Stam 2003]. In contrast, FEM does not require such a mapping, and is hence preferred. In a two-dimensional surface in $\mathbb{R}^3$, one can formulate FEM even without any prameterization. An earlier work [Dziuk 1988] exploits such advantage of FEM applied to the elliptic equation of $\nabla^2 u = f$. We use this idea and presents a new approach to FEM for solving PDEs directly on the connectivity graph of the triangle mesh, which has advantages in constructing the localized solver and handling constraints. We use linear elements because of their simplicity and performance. We also show how the linear element formulation of the Laplacian term $\nabla^2 u$ can be used for higher order bi-Laplacian term $\nabla^4 u$, which is needed for smooth interpolation and deformation. To develop this extension, we analyze the stiffness matrix resulting from the Laplacian and show that it can be recursively used with a correction matrix to formulate the bi-Laplacian.

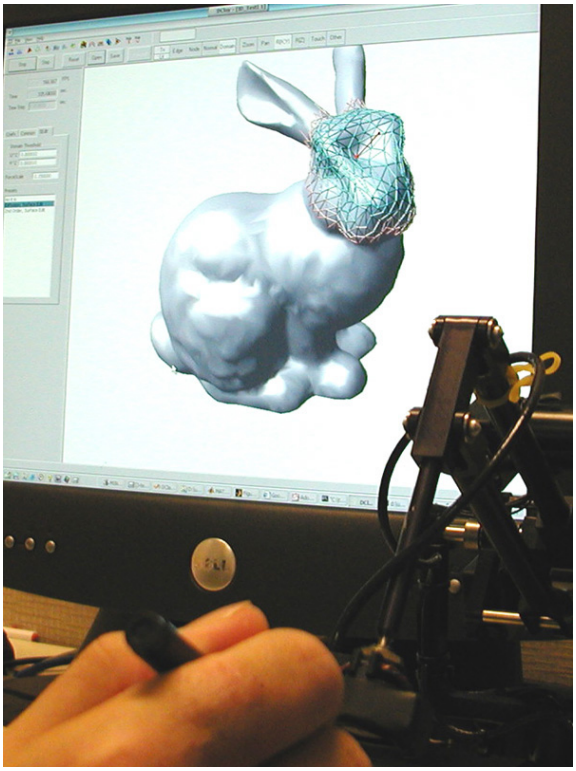

Figure 1: A user poking bunny with PHANToM. The PDE is solved in realtime by using self-adjusting active domain around the contact point.

To demonstrate the power and generality of formulation, we apply it to two problems that have received a fair amount of attention from the graphics, modeling, and haptic communities.

First, we consider a set of constraints specified at a few scattered vertices of the mesh and compute a field that smoothly interpolates these constraints. The field may define at each point of the surface a scalar, such as color, temperature, or a displacement imposed by the user to control a free-form deformation or a vector field on the surface for texture mapping or parameterization[Turk 2001]. We use the term scattered data interpolation to refer to this class of applications.

We also consider a haptic rendering system where the user pokes a surface using a PHANToM. The system must compute the reaction force and a local smooth deformation in real-time. We meet the real-time requirement by solving the PDE only in the neighborhood

*e-mail: bmkim@cc.gatech.edu
†e-mail: jarek@cc.gatech.edu

of the contact point.

1.1 Overview

We use the heat diffusion example to provide an overview of our algorithm. Using the Galerkin approximation[Hughes 2000], the isotropic heat equation is discretized as $\mathbf{M}\{\dot{T}\} + k\mathbf{K}\{T\} = \{q\}$, where k is a constant and $\{T\}, \{\dot{T}\}$ and $\{q\}$ are the long vectors containing the temperature, time derivative, and external heat flux sampled at each vertex. If we use the lumped mass method, $\mathbf{M}$ becomes diagonal. We store it in a 1D array Mii. $\mathbf{K}$ is the sparse stiffness matrix. We store its diagonal terms in the 1D array Kii and off-diagonal terms in another 1D array Kij. Each entry of Mii and Kii is associated with a vertex of the mesh and each entry of Kij is associated with an edge. Hence, by sorting Kii and Kij in the order in which the vertices and edges of the mesh are stored, we can trivially integrate this information with the standard data structures for triangle meshes. The presentation of the algorithm for solving this problem involves three parts. We summarize them here and provide details in subsequent sections.

Part one: Here we show how we compute the Kii, Kij, and Mii terms. The approach is summarized in the following algorithm.

```
Initialize Mii, Kii and Kij to zero.
For each triangle,
  Compute the triangle area Ae
    and element stiffness matrix Ke using (4).
  // Let the vertex ids of the triangle be v0,v1, and v2.
  // Let the edge ids be e0,e1,e2.
  Kii[v0]+=Ke[0][0]; Kii[v1]+=Ke[1][1]; Kii[v3]+=Ke[2][2];
  Kij[e0]+=Ke[1][2]; Kij[e1]+=Ke[0][2]; Kij[e2]+=Ke[1][2];
  Mii[v0]+=Ae/3; Mii[v1]+=Ae/3; Mii[v2]+=Ae/3;
```

Part two: Here we explain how we implement the time integration. We use the backward Euler integration method, which requires a solution to the matrix equation $(\mathbf{M}+\Delta t\mathbf{K})\{T\}^{n+1} = \mathbf{M}\{T\}^n + \Delta t\{q\}^{n+1}$, where $\{T\}^{n+1}$ is the unknown temperature for the next time step, while $\{T\}^n$ and $\{q\}^{n+1}$ are known variables that represent the current temperature and the applied heat constraints. We solve this equation using a conjugate gradient (CG) method, which requires a matrix-vector multiplication: $(\mathbf{M}+\Delta t\mathbf{K})\{T\}$, where $\mathbf{M}\{T\}$ is a trivial dot product, since $\mathbf{M}$ is diagonal. To compute $\mathbf{K}\{T\}$, we can consider $\mathbf{K}$ as a mask applied to $\{T\}$ as shown in the right of Fig. 2.

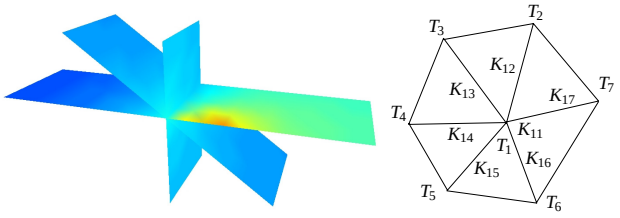

Figure 2: Heat diffusion over a non-manifold triangle mesh(left). An ilustration of the $\mathbf{K}\{T\}$ operation(right).

Each i$^{\text{th}}$element of $\mathbf{K}\{T\}$ is simply $K_{ii}T_i + \sum K_{ij}T_j$ for all the neighboring j$^{\text{th}}$vertices. The matrix-vector multiplication, $(\mathbf{M}+\Delta t\mathbf{K})\{T\}$, is simply plugged in to the standard CG iteration.

Part three: Here we explain how to incorporate the constraints, *i.e.*, the vertices where the temperature has been fixed. We handle constraints by simply not including constrained vertices in the above matrix computations and by solving only for the values at free vertices. However, notice that those fixed temperature values are still used when the neighboring free vertices are evaluated using the mask shown on the right of the Fig. 2. Most importantly, notice that there is no need to reconstruct the sparse matrix $\mathbf{K}$ when a constraint is inserted, deleted or moved to another vertex. When a large portion of the mesh is constrained, the solver needs only to work on the unconstrained, active region. This restriction is important for performance. In situations where the active region is not defined by the user and the effect is local sue to some dissipation mechanism, the domain is adjusted automatically as needed during the simulation. As the temperature is propagated away from a constrained vertex, the active region expands automatically. When the temperature dissipates, the active region shrinks back.

We summarize here the notation used throughout this paper. We will use the terms vertex and node interchangeably, since FEM nodes are the triangle vertices when the linear triangle element is used.

symbol	dimension	
V_i, E_i		i$^{\text{th}}$vertex, edge
E_{ij}		Edge between V_i and V_j
u_i,	$\mathbb{R}$	Value of the variable u at V_i
$\{u\}$	$\mathbb{R}^n$	Long column vector, stack of all u_i
row$_i(\mathbf{K})$	$\mathbb{R}^n$	i$^{\text{th}}$row of matrix $\mathbf{K}$
col$_i(\mathbf{K})$	$\mathbb{R}^n$	i$^{\text{th}}$column of matrix $\mathbf{K}$

2 Previous Works

The Laplacian operator appears when one formulates the PDE for the elementary heat, wave or fluid equation. The mesh fairing and smoothing application also often use Laplacian and bi-Laplacian terms[Taubin 1995; Kobbelt et al. 1998]. The spectral decomposition of the Laplacian matrix can also be used to partition or compress a mesh[Karni and Gotsman 2000]. These authors have used the simple Laplacian matrix defined as

$$\mathbf{L}_{ij} = \begin{cases} 1 & i = j \\ -1/n_{E_i} & n_{E_i} \text{ is valence of vertex } V_i \end{cases} \quad (1)$$

or its slight variations using different weights. Note that the umbrella operator proposed in [Kobbelt et al. 1998] produces the result of multiplying the x, y and z coordinates of the vertices by $\mathbf{L}$. This umbrella operator was refined by Desbrun et al., leading to the discrete Laplace-Beltrami operator [Desbrun et al. 1999; Meyer et al. 2003].

In contrast to the specialized previous approaches, we focus on developing a general purpose tool for solving a broad class of PDEs that involve Laplacian terms over triangle meshes. Following the FEM procedure, we start from the Galerkin approximation using the linear triangle element and construct the stiffness matrix $\mathbf{K}$. Interestingly, this approach produces results identical to those produced using the Laplace-Beltrami operator [Meyer et al. 2003], which was introduced to compute the surface normal scaled by the mean curvature. By showing the connection between this operator and the FEM formulation of the Laplacian, we justify its use for solving a broader class of problems, such as the elementary heat or wave equations over a triangle mesh, haptic rendering, scattered data interpolation over a triangle mesh, as well as mesh smoothing and fairing.

The challenges of haptic rendering [Salisbury et al. 1995] include computing the feedback force, removing force discontinuities [Kim et al. 2002], implementing friction and adding details with haptic texture [Minsky 1995]. In this paper, we address the issue of computing the feedback force, which has been in the past approximated with a simple spring model [Foskey et al. 2002; Kim et al. 2002]. The feedback force can be computed [James and Pai 2001] by solving a volumetric physical model of an elastic deformable body, using the boundary element method (BEM). Their method

fully considers the volume of the object while using the surface mesh only, resulting in much a smaller matrix compared to a volumetric mesh. The resulting matrix is much more dense but it is well conditioned and can be solved efficiently. However, their approach does not scale to larger meshes. To overcome this computational bottleneck, we propose to solve the PDE locally using a dynamically self-adjusting computation domain. We only use a surface mesh and Laplacian and bi-Laplacian terms. However, notice that our idea of locally solving the PDE can be extended to the PDE on a volumetric mesh. Our resulting matrix has the same size as BEM, but it is much more sparse, so it is possible to design a localized solver.

Scattered data interpolation is addressed using radial functions [Dyn 1989], FEM-based approaches of meshing the domain and constructing interpolating patches [Nielson et al. 1997], inverse distance weighted methods [Shepard 1968], multiresolution or hierarchical methods [Lee et al. 1997], and thin-plate models [Litwinowicz and Williams 1994; Lee et al. 1996]. These approaches are surveyed in [Amidror 2002]. Since the minimum of the thin plate energy can be obtained by solving a bi-Laplacian equation, our scheme may be considered to be a thin plate method. Previous publications mostly focused on fitting surfaces through scattered points or finding a higher dimensional field that satisfies a set of constraints. In this paper, we develop an interpolation scheme on a triangulated 3D surface by solving a Laplacian/bi-Laplacian PDE on it. We illustrate the approach by interpolating surface displacement constraints and by computing a vector field over the surface that interpolates tangent vectors specified by the user at a few vertices. The related prior art is discussed in [Turk 2001], where a diffusion strategy was used to solve this problem. Since the author was mostly interested in controlling the direction of a parameterization over the surface, the vector fields were normalized. We show that the interpolation of scattered data can be obtained by simply solving our Laplacian and bi-Laplacian.

The idea of solving a PDE, or an energy-minimizing formulation, over triangulated surface has been used in many other areas. We mention a few examples here. Level set techniques[Osher and Fedkiw 2003] can be applied to a triangulated domain and are especially useful when the domain has complex topology. PDE or corresponding energy formulation on triangle meshes have also been used to simulate a discrete shell[Grinspun et al. 2003]. The idea of localized solver have also been studied. For example, the narrow-band idea used in level set solvers[Adalsteinsson and Sethian 1995].

3 Finite Element Formulation

The Galerkin approximation[Hughes 2000] provides an established theoretical foundation for converting a PDE to its finite element form, which is a large set of algebraic equations, typically involving large sparse matrices and a long column vector of unknowns. In this section, we focus on formulating the Laplacian term $\nabla^2 u = \nabla\cdot\nabla u = u_{,xx}+u_{,yy}+u_{,zz}$[1], where $u = u(x,y,z)$ is a scalar variable defined over the triangle mesh. We use the shorthand notation of $u_{,x}$ for $\partial u/\partial x$ and $u_{,xx}$ for $\partial^2 u/\partial x^2$, etc. Let Ω be the computation domain (i.e., a subset of the surface) and let Γ be its boundary. The weak form of the PDE term $\nabla^2 u$ is obtained by multiplying the weighting function w and integrating over the domain Ω. Integrating by parts and applying the divergence theorem yields

$$\int_\Omega w\nabla^2 u d\Omega = -\int_\Omega \nabla w\cdot\nabla u d\Omega + \int_\Gamma (w\nabla u)\cdot\mathbf{n} d\Gamma \qquad (2)$$

This weak form has the term $\int_\Omega \nabla w\cdot\nabla u d\Omega$ with a reduced order derivative and a boundary integral term $\int_\Gamma (w\nabla u)\cdot\mathbf{n} d\Gamma$ that captures the boundary condition. In this section, we describe how to construct the stiffness matrix for the term $\int_\Omega \nabla w\cdot\nabla u d\Omega$ using a linear triangle element. Then, we show how this *linear* element can be modified to handle the higher order terms such as $\nabla^4 u$. The treatment of boundary condition is discussed in a later section.

3.1 Linear Triangle Element in 3D

The goal of this section is to convert the term $\int_\Omega \nabla w\cdot\nabla u d\Omega$ into a matrix-vector multiplication. We assume that the values of u are computed at the vertices of the triangle mesh. The i$^{\text{th}}$element of the vector $\{u\}$ represents the value of u at V_i. Using the finite element procedure, the integral is approximated by

$$\int_\Omega \nabla w\cdot\nabla u d\Omega \approx \{w\}^T\mathbf{K}\{u\} \qquad (3)$$

$\{w\}$ will be factored out in all PDE terms, yielding a linear equation of $\{u\}$. $\mathbf{K}$ is the global stiffness matrix, which we must construct for the triangle mesh. The finite element procedure permits us to perform the integration in (3) on each of the single triangle elements individually and yields the element stiffness matrix $\mathbf{K}_e \in \mathbb{R}^{3\times 3}$. The global stiffness matrix $\mathbf{K}$ is obtained by simply assembling all the $\mathbf{K}_e$'s.

3.2 Element Stiffness Matrix

We first develop a linear triangle element in 3D. Unlike most triangle elements in the former FEM literature that use a two-dimensional coordinate system in the triangle plane, we use all x,y and z coordinates to embed the coordinate transformation that will be needed otherwise.

To derive the element stiffness matrix, we follow a standard procedure in FEM[Hughes 2000]. Since it is standard approach, we omit its description here and only present the results in this section. However, to help the readers more easily follow our approach, we provide a sketch of derivation in the Appendix. The stiffness matrix of a triangle element $\mathbf{K}_e$ is computed as

$$\begin{aligned}\mathbf{K}_e &= A\mathbf{B}_s\mathbf{B}_s^T \\ \mathbf{B}_s &= \frac{1}{2A}\begin{bmatrix} x_2-x_3 & y_2-y_3 & z_2-z_3 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \\ x_1-x_2 & y_1-y_2 & z_1-z_2 \end{bmatrix}\end{aligned} \qquad (4)$$

where A is the area of the triangle. Note that $\mathbf{B}_s$ is **not** the Jacobian of the natural coordinates ($\mathbf{B}$ in Appendix) that is commonly found in the FEM literature. [2] Interested reader may refer to the Appendix.

3.3 Representation of Global Stiffness Matrix

Once the element stiffness matrix $\mathbf{K}_e$ is computed, we need to assemble it to form the global stiffness matrix $\mathbf{K}$. In this section, we explain this assembly procedure in conjunction with the triangle mesh connectivity. Hence, we devise a mesh-friendly representation of the sparse matrix $\mathbf{K}$ and discuss its advantages.

The symmetric matrix $\mathbf{K}_e \in \mathbb{R}^{3\times 3}$ contains the three diagonal terms corresponding to the triangle vertices, as well as the three off-diagonal terms corresponding to the triangle edges. Each one of the triangles has its own $\mathbf{K}_e$, and the vertices or edges can have multiple triangles that share them. As a result, a vertex or an edge may have multiple contributions from multiple triangles. Notice that the

[1] Notice that $u_{,xx}+u_{,yy}+u_{,zz}$ become a 2D Laplacian $u_{,s_1s_1}+u_{,s_2s_2}$, where s_1, s_2 are tangent direction when we consider local tangent plane approximation of the surface and assume that u only varies on the surface.

[2] To incorporate an additional matrix $\mathbf{E}$ for anisotropy or terms such as $u_{,xy}$, care must be taken since, in general, $\mathbf{K}_e = A\mathbf{B}\mathbf{E}\mathbf{B}^T \neq A\mathbf{B}_s\mathbf{E}\mathbf{B}_s^T$.

assembly procedure in FEM simply sums all of these contributions. The diagonal term $\mathbf{K}_{ii}$ is the summation of all terms corresponding to $\mathbf{p}_i$ in all $\mathbf{K}_e$'s of the triangles that share $\mathbf{p}_i$. Similarly, the off-diagonal term $\mathbf{K}_{ij}$ is the summation of all terms corresponding to the edge E_{ij}, in all $\mathbf{K}_e$'s of the triangles that share E_{ij}.

$\mathbf{K}_{ij}$ is nonzero, if $i = j$ or if the edge E_{ij} exists. This observation leads to a convenient representation of the sparse matrix $\mathbf{K}$. We store the diagonal term $\mathbf{K}_{ii}$ with the corresponding vertex and the off-diagonal term $\mathbf{K}_{ij}$ with the corresponding edge. Notice that this representation does not require that the mesh be manifold.

Using this representation, we consider the multiplication of $\mathbf{K}\{u\}$, where $\{u\}$ is a collection of u defined at vertices. The i^{th}entry of the resulting vector $\mathbf{K}\{u\}$ is the values of u in neighboring vertices multiplied with $\mathbf{K}_{ij}$ plus its own value u multiplied by $\mathbf{K}_{ii}$.

3.4 Higher Order Laplacian

Since we are using a linear element, it is impossible to solve a PDE that make use of the bi-Laplacian. However, we have found that a simple extension can lead to the formulation of higher order terms such as the bi-Laplacian $\nabla^4 u = u_{,xxxx} + u_{,yyyy} + u_{,zzzz} + 2u_{,xxyy} + 2u_{,yyzz} + 2u_{,zzxx}$. The idea is to mix FEM and FDM. We first differentiate u to compute a discrete version of $\nabla^2 u \approx \hat{u}$ at each node. Then, we use the following fact

$$\nabla^4 u = \nabla^2 \left(\nabla^2 u \right) \approx \nabla^2 \hat{u} \tag{5}$$

Since the global stiffness matrix $\mathbf{K}$ is formulating $\nabla^2 \hat{u}$, we only need to compute $\hat{u}$. However, we have found that the same $\mathbf{K}$ can be re-used even for $\hat{u}$ with a diagonal correction matrix $\mathbf{D}$.

$$\mathbf{D} \equiv \mathrm{diag} \left(\frac{3}{\tilde{A}_i} \right) \tag{6}$$

where $\tilde{A}_i$ is the sum of areas of all triangles that contain the i^{th} vertex. The derivation of $\mathbf{D}$ is provided in the Appendix. Notice that the area of a triangle is always computed when $\mathbf{K}_e$ is computed. Thus, $\mathbf{D}$ can be easily computed in a subroutine that assembles $\mathbf{K}$. Using $\mathbf{D}$, the approximation $\hat{u}$ is computed by

$$\nabla^2 u \approx \hat{u} = -\mathbf{DK}\{u\} \tag{7}$$

Consequently, from (5), the bi-Laplacian term is formulated as

$$\nabla^4 u \implies \mathbf{K}\{\hat{u}\} = -\mathbf{KDK}\{u\} \tag{8}$$

The resulting matrix $\mathbf{KDK}$ is still sparse but it is about twice as populated as $\mathbf{K}$. However, we do not need to compute or store $\mathbf{KDK}$. We implement it within the matrix solver using the Conjugate Gradient method. The only operation needed is computing $\mathbf{KDK}\{u\}$, which can be done in three steps, only using $\mathbf{K}$ and the diagonal matrix $\mathbf{D}$, *i.e.*, $\mathbf{K}[\mathbf{D}(\mathbf{K}\{u\})]$.

3.4.1 The Laplace-Beltrami Operator

The Laplace-Beltrami operator computes the normal vector with the magnitude of $\kappa_1 + \kappa_2$, where κ_1, κ_2 are the two principal curvatures. $\kappa_1 + \kappa_2$ can be locally approximated by the Laplacian of the local height field, which can then be transformed back to the global coordinates. Since the Laplacian is invariant under an orthogonal transformation, $-\mathbf{DK}$ should act as a Laplace-Beltrami operator when we multiply it with the coordinates of the vertices.

Assume $\{x\}, \{y\}$ and $\{z\}$ are the vectors containing the x, y and z coordinates of the vertices. Let $\mathrm{row}_i(\mathbf{K}\{x,y,z\})$ be a vector made of the i^{th}rows of $\mathbf{K}\{x\}$, $\mathbf{K}\{y\}$, $\mathbf{K}\{z\}$. We found that the Laplace-Beltrami operator proposed in [Meyer et al. 2003] [3] has close relationship to our formulation

$$\begin{aligned} \Delta_B|_{\mathbf{p}_i} &= \frac{3}{2\tilde{A}} \sum_{j \in N(V_i)} \left(\cot\alpha_{ij} + \cot\beta_{ij} \right) \left(\mathbf{p}_i - \mathbf{p}_j \right) \\ &= -\frac{3}{\tilde{A}} \mathrm{row}_i(\mathbf{K}\{x,y,z\}) \;=\; \mathrm{row}_i(-\mathbf{DK}\{x,y,z\}) \end{aligned} \tag{9}$$

where $\mathbf{p}_i$ are the vertex point of the V_i and α_{ij}, β_{ij} are angles of the corner facing the edge E_{ij} and $N(V_i)$ denotes the set of nodes connected to V_i. One can prove this identity from the fact that $\cot\alpha_{ij}$ and $\cot\beta_{ij}$ can be written as a dot product between edge vectors divided by $2A$. The $\cot(.)$ terms can also be found in [Angenent et al. 1999] in their global stiffness matrix formulations.

Even though we use the formulation of [Meyer et al. 2003], we show different way of defining the Laplace-Beltrami operator in the Appendix. For comparison, we apply these two operators to the sphere meshes and verify that they correctly compute the radius of the sphere. We found a trade-off between consistency and accuracy.

3.4.2 Comparison to Nested Laplacian Operators

It is interesting that even though $\mathbf{K}$ is derived from the FEM formulation, we can use it for a FDM-style formulation, if we discretize $\nabla^4 u$. Using $-\mathbf{DK}$ as a discrete ∇^2 operator, the bi-Laplacian equation can be discretized as

$$\nabla^4 u = 0 \implies \mathbf{DKDK}\{u\} = 0; \tag{10}$$

This formulation differs from the FEM formulation by the left-most term, $\mathbf{D}$, only. This term is a diagonal matrix. Although it can be handled within the standard approach, it increases the condition number of the matrix and can thus increase the number of CG iterations. This side effect may be easily overcome by using the Jacobi pre-conditioner since it divides all equations by the diagonal entries of the matrix $\tilde{A}$.

We notice that the idea of nesting has been already reported in [Kobbelt et al. 1998; Schneider and Kobbelt 2001; Yoshizawa et al. 2003]. However, only a simple Laplacian given in (1) is used.

3.4.3 Our Contribution in This Section

The contribution of our work in the context of previous publications is that we construct a framework to broaden the application of the Laplace-Beltrami operator, which has been previously used for computing the curvature in mesh fairing and smoothing applications. In this paper, we show that this operator can be applied to any property, such as temperature or vector field diffusion, interpolation of data scattered over a (non-)manifold triangle mesh, or haptic rendering.

Since the Laplace-Beltrami operator is not defined in non-manifold mesh, it has no use for it. However, the FEM approach we presented in the earlier chapter does not have such limitations. For example, the heat diffusion problem across non-manifold mesh, as is shown in Fig. 2, can be solved without any additional modification. Thus, we justified the use of the operator (4), or equivalently, (9), even for non-manifold surface. Since using $\mathbf{K}$ given by (4) does not involve $\cot(\cdot)$ terms, it is in a less expensive form than (9). To this reason, we use $\mathbf{K}$.

[3] The original formula used by [Desbrun et al. 1999] was computing 1/3 of the $\kappa_1 + \kappa_2$. A more accurate formula was proposed later by the same group [Meyer et al. 2003], using the idea of non-overlapping area. As others [Ohtake et al. 2000; Schneider and Kobbelt 2001], we devide by a third of the area.

4 Construction of the PDE

We consider a partial differential equation of a scalar variable u that is defined over a triangle mesh.

$$\alpha u_{,tt} - \gamma_1 \nabla^2 u + \gamma_2 \nabla^4 u = f \tag{11}$$

where f is an external forcing term. The discretization of this PDE with an artificial damping term yields

$$\alpha \mathbf{M}\{\ddot{u}\} + \beta \mathbf{C}\{\dot{u}\} + \tilde{\mathbf{K}}\{u\} = \{f\} \tag{12}$$

where $\{f\}$ is the vector of the external forcing term and the stiffness matrix $\mathbf{K}$ is defined as

$$\tilde{\mathbf{K}} = \sum_{i=0}^{n} \gamma_i \mathbf{K}(\mathbf{D}\mathbf{K})^{i-1} = \gamma_1 \mathbf{K} + \gamma_2 \mathbf{K}(\mathbf{D}\mathbf{K}) \tag{13}$$

Notice that the negative sign in the Laplacian term in (11) is cancelled with the negative sign in (8), keeping $\tilde{\mathbf{K}}$ positive definite. For the artificial damping term, we choose Rayleigh damping $\beta\mathbf{C} = \beta_1\mathbf{M} + \beta_2\tilde{\mathbf{K}}$[Hughes 2000].

The backward Euler integration formulation of the above equation is [4]

$$\begin{aligned} \tilde{\mathbf{A}}\{\dot{u}\}^{n+1} &= -\Delta t\tilde{\mathbf{K}}\{u\}^n + \alpha\mathbf{M}\{\dot{u}\}^n + \Delta t\{f\}^{n+1} && (14)\\ &\quad \text{where} \quad \tilde{\mathbf{A}} = (\alpha + \beta_1\Delta t)\mathbf{M} + (\beta_2 + \Delta t)\Delta t\tilde{\mathbf{K}} \\ \{u\}^{n+1} &= \{u\}^n + \{\dot{u}\}^{n+1}\Delta t && (15) \end{aligned}$$

Notice that the CG iteration only needs $\tilde{\mathbf{A}}\{u\}$ in solving (14), which can be done by the matrix-vector product of the diagonal matrix $\mathbf{M}$ and the very sparse matrix $\mathbf{K}$.

4.1 Boundary Conditions

The Galerkin approximation formulates a discrete equation restricted to the free nodes. Boundary conditions appear as extra terms in the formulation. This is not convenient, since whenever the locations of a boundary condition is changed, the resulting system must be reformulated. Therefore, a typical approach is to construct the matrices assuming no boundary conditions and to produce a discrete set of equations that includes nodes that are restricted by boundary conditions as if they were free nodes. Then, the boundary conditions are applied to this discrete equation.

Until now, we have assumed no boundary conditions, *i.e.*, no node where the values u_i or $\dot{u}_i$ are fixed. Now we consider the situation where u_i or $\dot{u}_i$ are only fixed for some nodes. For simplicity, suppose that $\dot{u}_i$ is fixed at the node V_i. We omit the superscript $n+1$ for readability. The equation (14) can be expressed as

$$\left[\begin{array}{c|c|c} \cdots & \mathrm{col}_i(\tilde{\mathbf{A}}) & \cdots \\ \cdots & & \cdots \\ \hline \mathrm{row}_i(\tilde{\mathbf{A}}) & & \\ \hline \cdots & & \cdots \\ \cdots & & \cdots \end{array}\right] \left[\begin{array}{c} \dot{u}_1 \\ \cdots \\ \dot{u}_i \\ \cdots \\ \dot{u}_n \end{array}\right] = \{b\} + \Delta t \left[\begin{array}{c} f_1 \\ \cdots \\ f_i \\ \cdots \\ f_n \end{array}\right] \tag{16}$$

where $\{b\} = -\Delta t\tilde{\mathbf{K}}\{u\}^n + \alpha\mathbf{M}\{\dot{u}\}^n$ and $\mathrm{row}_i(\tilde{\mathbf{A}}), \mathrm{col}_i(\tilde{\mathbf{A}})$ represent the i^{th}row and column of $\tilde{\mathbf{A}}$. Since $\dot{u}_i$ is fixed, f_i should be a unknown force applied to V_i. Indeed f_i is the force holding V_i so that it meets the constraint, which will be used as the feedback force in a haptic rendering application (as discussed later). Thus, the unknowns are $\dot{u}_1, \dot{u}_2, \ldots, \dot{u}_{i-1}, \dot{u}_{i+1}, \ldots, \dot{u}_n$ and f_i. We first compute all

[4] When $\alpha = 0, \beta_2 = 0$, one may compute $\{u\}^{n+1}$ directly using $(\beta_1\mathbf{M} + \tilde{\mathbf{K}}\Delta t)\{u\}^{n+1} = \beta_1\mathbf{M}\{u\}^n + \Delta t\{f\}^{n+1}$. However, we observed that we can still use (14,15) even in this case.

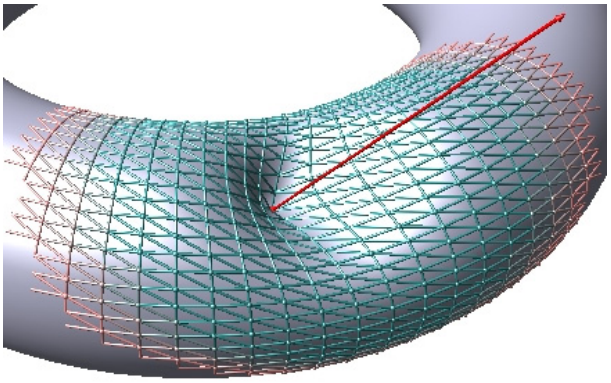

Figure 3: Automatically self-adjusting domain. Blue is the domain and red is a transition domain needed in the intermediate computation stage. All other vertices are not used.

unknown $\dot{u}$. Let $\tilde{\mathbf{A}}^F$ be the $\tilde{\mathbf{A}}$ without $\mathrm{row}_i(\tilde{\mathbf{A}})$ and $\mathrm{col}_i(\tilde{\mathbf{A}})$. Let $\{u\}^F, \{b\}^F, \{f\}^F$ be $\{u\}, \{b\}, \{f\}$ without u_i, b_i, f_i respectively. Then, all rows of (16) except for i^{th}row can be written as

$$\tilde{\mathbf{A}}^F\{u\}^F + \mathrm{col}_i(\tilde{\mathbf{A}})\dot{u}_i = \{b\}^F + \Delta t\{f\}^F \tag{17}$$

Once all the unknown $\dot{u}$ are computed, the unknown force f_i can be computed easily if it is required, as in haptic rendering. This can be trivially generalized to multiple constraints. Notice that this approach reduces the dimension of $\tilde{\mathbf{A}}^F$. Also notice that in equation (14), the constraints on u_i, generate constraints on $\dot{u}_i$, a situation discussed earlier.

However, we do not construct $\tilde{\mathbf{A}}^F$ and pass it to the sparse matrix subroutine. Remember that we do not even formulate $\tilde{\mathbf{A}}$. We only have diagonal matrices $\mathbf{M}, \mathbf{D}$ and a sparse matrix $\mathbf{K}$, and we use these matrices to implement the operation $\tilde{\mathbf{A}}^F\{u\}^F$, which is needed in the CG iteration. We discuss this in the next section.

The filtered CG iteration idea, proposed by Baraff and Witkin in [Baraff and Witkin 1998], is similar to our approach. They modified the CG iteration so that many different types of constraints could be applied inside the CG iteration loop. This approach has was broadly adopted [Choi and Ko 2002; Desbrun et al. 1999]. In contrast, our approach is only able to handle hard constraints on the nodes. However, the resulting CG iteration is designed to operate on the reduced matrix $\tilde{\mathbf{A}}^F$, while [Baraff and Witkin 1998] operates on the full size matrix $\tilde{\mathbf{A}}$. Operating on the full matrix may be acceptable for application such as cloth simulation where most of the nodes are free to move, but it is unnecessarily expensive when a large portion of the mesh is constrained or only a small local region needs to be solved.

4.2 Computation Domain and Localized Solver

Consider the operation $\mathbf{M}\{u\}$. Since $\mathbf{M}$ is a diagonal matrix, its entries are stored in parallel with the vertices. So are the entries of $\{u\}$. Then, $\mathbf{M}\{u\}$ is simply computed by multiplying two properties of vertices. Now we consider $\mathbf{K}\{u\}$. Remember that the diagonal terms $\mathbf{K}_{ii}$ are the properties of vertices and that off-diagonal terms $\mathbf{K}_{ij}$ are the properties of edges. Thus, we can consider $\mathbf{K}\{u\}$ as an operation that computes $\mathbf{K}_{ii}u_i$ plus $\mathbf{K}_{ij}u_j$ fosr all vertices $\mathbf{p}_j$ connected to vertex $\mathbf{p}_i$ by edge E_{ij}.

The above approach simplifies the implementation of (17). Notice that without it, one would have to construct a much denser matrix $\tilde{A}$ and store it somewhere, then construct $\tilde{A}^F$, and finally send it to the sparse matrix subroutine.

We define C as the set of vertices where the values are constrained and use the symbol D for the free vertices of the active region. We further define D_i° as the collection of nodes in the edge distance of i to the region D.[5] We also define $D_i^\uparrow$ as a set of D grown by the edge length of i.[6]

We only need to implement the matrix-vector multiplications, $\mathbf{K}\{u\}$ and $\mathbf{M}\{u\}$, for the nodes that belongs to D. In this computation, the constrained nodes will be accessed only if they are in the neighborhood of $\mathbf{K}\{u\}$, but their node values will not be changed. Hence, the computational domain of $\mathbf{DK}\{u\}$ should include the neighborhood of D, i.e. $D \cup D_1^\circ = D_1^\uparrow$. To do this, we need an intermediate computation, whose result is stored as a parallel array coordinated with the triangle mesh structure.

Now we can evaluate $\tilde{\mathbf{K}}\{\dot{u}\}$, $\mathbf{M}\{\dot{u}\}$ and thus $\tilde{\mathbf{A}}\{\dot{u}\}$ and construct the left-hand side of (17) and plug it into the CG solver. Hence we solve (17) over D, satisfying the constraints.

During interactive editing, local deformation may be sufficient, as is illustrated in Fig. 3. In this case, we only need to solve the equation for the localized domain around the nodes being touched. We denote this active domain by A. Similar to D, we define A_i° and $A_i^\uparrow$. Figure 4 illustrates A_1° required in solving the PDE that involves the bi-Laplacian term. As the deformation happens, the domain needs to be extended. To test for this, we check the values of the boundary of the domain. If the values tends to change, we simply expand the domain to its neighbor. If the values are not changing at the node, we simply remove it. We show the usefulness of this approach in our haptic rendering application.

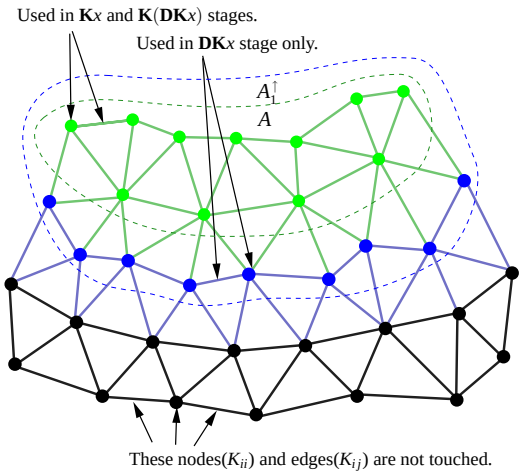

Figure 4: The active domain A and the intermediate domain $A_1^\uparrow$.

5 Applications

In this section, we discuss applications of the proposed technique. The variable u will represent temperature, displacement, or the coordinates of a vertex. We change the coefficients of $\alpha, \beta_1, \beta_2, \gamma_1, \gamma_2$ as needed by the application. We provide nonzero coefficients only. All computation times are measured on a Pentium4 2.5GHz PC.

[5] For example, D_2° is a set of vertices that need to traverse at least two edges to reach D. Also notice that $D_i^\circ \in C$.

[6] $D_i^\uparrow = D \cup \left(\cup_{j=1}^{i} D_j^\circ\right)$.

5.1 Elementary PDEs

First, we demonstrate the two elementary PDEs: the heat diffusion and wave propagation equations over a triangle mesh. Heat diffusion can be easily implemented by setting $\beta_1 = 1, \gamma_1 = 1$. The wave equation can be implemented by setting $\alpha = 1, \gamma_1 = 1$.

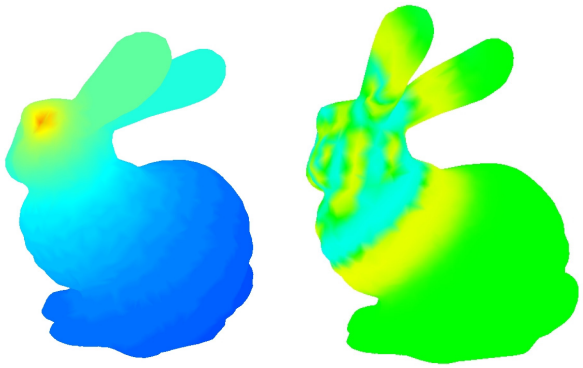

Figure 5: Heat equation(left) : A heat source is applied at the eye of the rabbit. The temperature of the surface after 150 seconds is shown color-coded. Wave equation(right) : An impulse is applied at the eye. The wave propagation after 5 seconds is shown. (Notice that the surface is not smooth and the wave is bouncing.)

5.2 Simple Mesh Fairing and Regularized Membrane Formation

In this section, we use the fact that a simple mesh fairing can be achieved by solving three sets of independent equations on $\{x\}, \{y\}$ and $\{z\}$.[7] The fairing idea we show in this section is simply solving Laplacian and bi-Laplacian equations. Advanced fairing methods are proposed in [Kobbelt et al. 1998; Ohtake et al. 2000; Schneider and Kobbelt 2001].

Figure 6 illustrates fairing an egg shaped model shown at the far left where the lower hemisphere is fixed and upper half is free to move. The center-bottom shows triangulation artifacts since $\mathbf{K}$ computes the Laplacian correctly assuming small deformation only. We can animate the smoothing process by letting $\beta_1 = 1$ and try re-computing $\mathbf{K}$ at each time step. However, as mentioned in [Kobbelt et al. 1998], the process become unstable. We propose an approach to help remedy to this problem. Our approach works robustly for the formation of a membrane($\gamma_1 \neq 0$), but not in the bi-Laplacian equation. We weight the element stiffness $\mathbf{K}_e$ in (4) by its area,*i.e.*, assembling $A\mathbf{K}_e$ instead of $\mathbf{K}_e$, at each time step. This makes small triangles softer and large triangles stiffer. As a result, large stiff triangles pull small soft ones, regularizing the mesh. The Fig. top-right image of 6 shows a regularized membrane mesh produced with this approach. It turned out that this process is very robust. We applied it to the Stanford bunny model where lower half is constrained. The whole process requires 15 seconds

[7] This is due to the fact that the membrane energy $J(h) = \int_S \left(h_{,s_1}^2 + h_{,s_2}^2\right) dS$ has minimum when $\delta J(h) = 0$, which leads to the weak form of the PDE $\nabla^2 h = 0$. h is the local height variable and s_1, s_2 parameterize local tangent plane. Since $J(h)$ is invariant under the orthogonal coordinate transformation, the equation $\nabla^2 h = 0$ and two other trivial equations $\nabla^2 s_1 = 0, \nabla^2 s_2 = 0$ can be transformed back to PDEs on x, y, z, *i.e.*, $\nabla^2 x = \nabla^2 y = \nabla^2 z = 0$. Similarly, the minimum of the plate energy $J(h) = \int_S \left(h_{,s_1 s_1}^2 + h_{,s_2 s_2}^2 + 2h_{,s_1 s_2}^2\right) dS$ [Kobbelt et al. 1998; Turk and O'Brien 1999] is obtained when $\nabla^4 h = 0$ that is solved by $\nabla^4 x = \nabla^4 y = \nabla^4 z = 0$.

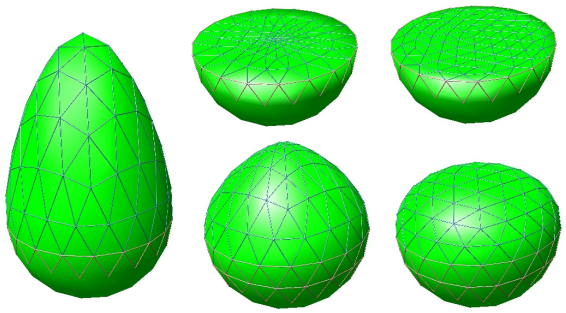

Figure 6: Left is initial shape. Top center : $\gamma_1 = 1$, bottom center :$\gamma_2 = 1$, top right: $\beta_1 = 1, \gamma_1 = 1$, updated $\mathbf{K}$ with area weighting, right bottom: $\beta_1 = 1, \gamma_1 = 2$, updated $\mathbf{K}$ with area weighting, grown from top center.

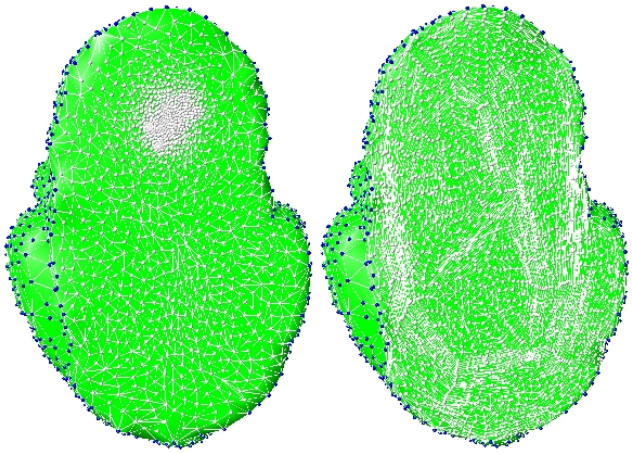

Figure 7: The upper half of the Stanford bunny model is collapsed into a membrane. The high concentration of vertices (left) is regularized (right) using area-based weights.

with $\Delta t = 1, \beta = 0.01, \gamma_1 = 1$. As is show in the Fig. 7, the resulting mesh is regularized. We also applied the same approach to the bi-Laplacian to produce the shape shown in Fig. 6 bottom-right. However, this regularization is not robust and it often fails on more complex models. This remains a future research issue.

5.3 Painting : Scattered Data Interpolation over Triangle Mesh

The Laplacian and bi-Laplacian provide a mechanism that propagates a property on the surface. This can be used to interpolate data scattered over the triangle mesh. Consider the situation when a property is defined for a few vertices of a triangle mesh but it is not defined elsewhere. The user may want to distribute the property over the entire surface, interpolating the values at the already-defined vertices. The diffusion mechanism can be efficiently used in this application.

We show interpolation results for two different type of data: vector fields and displacement fields. Since these are vector-valued properties, we apply the diffusion process on each coordinate of the vector. Figure 8 shows the result of displacement field interpolation on a simple mesh. The Laplacian shows the spike around the constrained point and bi-Laplacian shows a smooth shape with a bump in the middle. When these terms are mixed, these bump tends to decrease. We can also apply this on a non-manifold mesh, shown

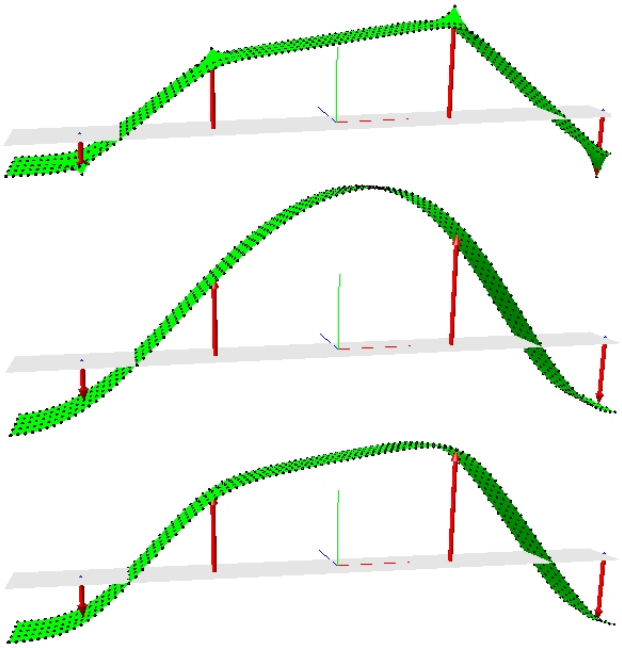

Figure 8: Interpolation of displacement constraints. Laplacian($\gamma_1 =$ 1,top), bi-Laplacian ($\gamma_2 =$ 1,middle) and mixed ($\gamma_1 = 1, \gamma_2 =$ 0.1,bottom)

in the Fig. 9. Figure 10 provides another example of displacement

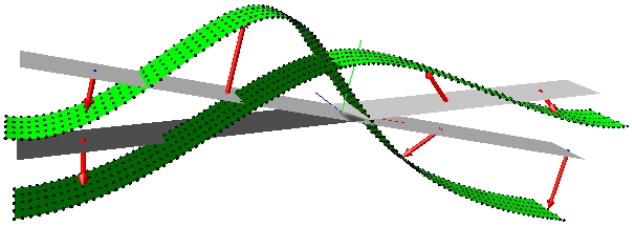

Figure 9: Interpolation of displacement constraints over a non-manifold surface using the bi-Laplacian ($\gamma_2 = 1$)

interpolation over the bunny model. This bunny model has 6578 triangles, 3291 vertices and 9867 edges.

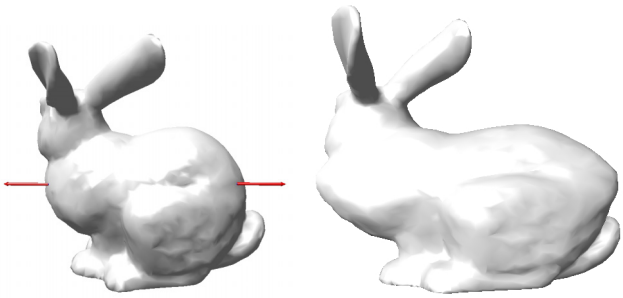

Figure 10: Displacement field interpolation ($\gamma_1 = 1, \gamma = 0.1$, obtained by a single time step that took 19.8 sec)

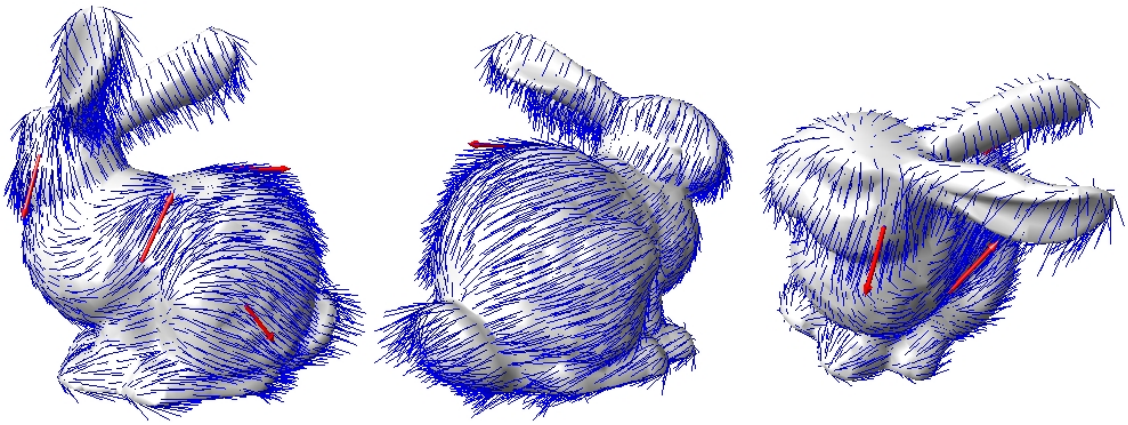

Figure 11: Exact and smooth interpolation of vector field constraints by Laplacian and bi-Laplacian($\gamma_1 = 1$,$\gamma_2 = 0.05$,11.82sec)

Our second experiment is vector field diffusion. In Fig. 11, the four vertices have predefined tangential vectors of arbitrary magnitudes. Since we consider the vector field on the surface, all resulting vectors must be tangent to the surface. Therefore, we project the vector field once the diffusion step is done.

5.4 Haptic Rendering on a Deformable Object

In this section, we discuss how the above self-adjusting computational domain can be used to implement haptic rendering on a deformable object. Notice that in haptic rendering, the deformation typically affects the surface around the point touched by the user. Also, it is reasonable to assume that for a big model, the objects typically have sufficient damping so that the deformation is not propagated to the entire model but instead is dissipated locally.

With this observation in mind, we solve the PDE locally using an automatically self-adjusting active domain of computation as discussed in section 4.2. This idea can be used in all physically-based models that produce a sparse matrix [Grinspun et al. 2002; Muller et al. 2002; Capell et al. 2002] if a proper dissipation term is integrated. In comparison to volumetric models, the Laplacian or bi-Laplacian equations do not represent true physical model, but we show that solving the PDE locally is sufficient to support a haptic rendering system on a complex mesh. In spite of the loss of connection with a physical volumetric model, we found that the resulting system provides the look and feel of physical material. Thus, we advocate this approach, which it does not require the creation of a volumetric mesh.

The computation of the feedback force is simple. Suppose that the i^{th} node is being poked. Since the point poked is attached to the phantom position, it is under constraint and $u_i, \dot{u}_i$ are known. Thus, the i^{th}row of (16) will not be included in (17). Once (17) is evaluated, f_i can be easily evaluated from the i^{th}row of (16). Since we are dealing with a vector-valued displacement field, we need to solve for the three components of the displacement individually.

Figure 12 shows a model being poked by a phantom. The red arrows show the directions of forces. In the right image, the user moved the phantom left and the force is adjusted accordingly. When the user stops poking, the active domain shrinks as the model recovers its initial shape and will finally disappear. In these two images, the user is poking a coarse mesh and the frame rate exceeds 1KHz. When the user pokes a dense mesh, the frame rate is reduced to under 1KHz. However, the frame rate can be kept high if one chose a smaller value for γ_2.

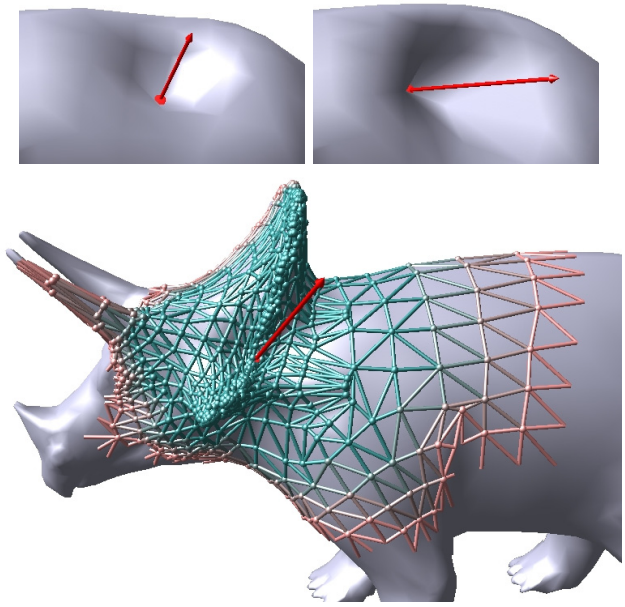

Figure 12: Poking a model with the haptic feedback.

6 Conclusion

We have formulated the Laplacian term using linear triangle elements. We prove that the resulting matrix is identical to the discrete Laplace-Beltrami operator proposed earlier to estimate the mean curvature vector. Since our approach has theoretical foundation in FEM, we show that this Laplace-Beltrami operator can be used in a broad set of PDE applications involving Laplacian and bi-Laplacian terms on a mesh that is not necessarily a manifold. From a geometric interpretation of the resulting FEM matrix, we have developed a way to formulate higher order bi-Laplacian terms in FEM context but using linear element. We also propose a mesh-friendly representation of the FEM matrices and pave a way to consider a matrix operation as a triangle mesh operation. As a result, we provide a way to handle constraints gracefully and propose a self-adjusting active domain, illustrating its benefits for haptic rendering. We also demonstrate the usefulness of these tools for scattered data interpolation over triangle meshes.

7 Acknowledgement

This work was supported by the NSF under the ITR Digital clay grant 0121663.

Appendix

Derivation of $\mathbf{K}_e$

We assume that the values of u are given at vertices of the triangle mesh. We first interpolate these discrete samples of u over each triangle and then compute its gradient.

Consider a thin triangle element with three vertices $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$, where $\mathbf{p}_i = [x_i\ y_i\ z_i]^T$. Let the triangle normal vector to be $\mathbf{n} = (n_x, n_y, n_z)$. Consider the parameterization of this thin triangle with the three barycentric coordinates ξ_1, ξ_2, ξ_3 and another parameter ξ_4 for normal direction, which is introduced to facilitate the computation of analytic inverse mapping that will be discussed shortly.

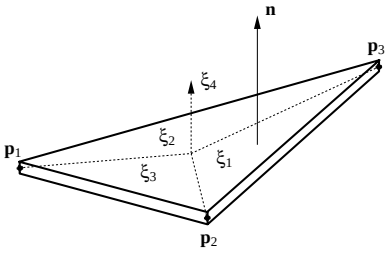

Figure 13: Triangle element in 3D

The mapping from $\xi_{1,2,3,4}$ to x, y, z is

$$\begin{Bmatrix} 1 \\ x \\ y \\ z \end{Bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ x_1 & x_2 & x_3 & n_x \\ y_1 & y_2 & y_3 & n_y \\ z_1 & z_2 & z_3 & n_z \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \end{Bmatrix} \tag{18}$$

The inverse mapping from x, y, z to $\xi_{1,2,3,4}$ can be computed analytically.

$$\begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \end{Bmatrix} = \begin{bmatrix} \times & \xi_{1,x} & \xi_{1,y} & \xi_{1,z} \\ \times & \xi_{2,x} & \xi_{2,y} & \xi_{2,z} \\ \times & \xi_{3,x} & \xi_{3,y} & \xi_{3,z} \\ \times & \times & \times & \times \end{bmatrix} \begin{Bmatrix} 1 \\ x \\ y \\ z \end{Bmatrix} \tag{19}$$

where $\xi_{i,x} = \frac{\partial \xi_i}{\partial x}, \xi_{i,y} = \frac{\partial \xi_i}{\partial y}, \xi_{i,z} = \frac{\partial \xi_i}{\partial z}, i = 1,2,3$. We define $\mathbf{B}$ as the upper right 3×3 block of this inverse mapping matrix.

$$\mathbf{B} = \begin{bmatrix} \xi_{1,x} & \xi_{1,y} & \xi_{1,z} \\ \xi_{2,x} & \xi_{2,y} & \xi_{2,z} \\ \xi_{3,x} & \xi_{3,y} & \xi_{3,z} \end{bmatrix} \tag{20}$$

The entries of $\mathbf{B}$ can be computed analytically using the area of the triangle

$$A = ((\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)) \cdot \mathbf{n}/2 \tag{21}$$

They are product between coordinates and the normal vector entries. For example,

$$\xi_{1,x} = (-y_3 n_z + n_y z_3 + y_2 n_z - z_2 n_y)/(2A) \tag{22}$$

In the linear triangle element, the interpolation functions are simply $\xi_{1,2,3}$, yielding the simple interpolation of

$$u = \xi_1 u_1 + \xi_2 u_2 + \xi_3 u_3 \tag{23}$$

where $u_{1,2,3}$ are the values of u at the node $\mathbf{p}_{1,2,3}$. Since we do not consider the triangle thickness, we simply do not use ξ_4. The gradient of u is computed using the chain rule

$$\nabla u = \begin{bmatrix} u_{,x} \\ u_{,y} \\ u_{,z} \end{bmatrix} = \mathbf{B}^T \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{24}$$

Notice that the property interpolated with ξ has zero gradient along the normal direction, *i.e.*,

$$\mathbf{n}^T\ \mathbf{B}^T = 0 \tag{25}$$

Using the same interpolation rule for the weighting function w, the term $\int_\Omega \nabla w \cdot \nabla u d\Omega$ can be integrated analytically.

$$\int_\Omega \nabla w \cdot \nabla u dA = [w_1\ w_2\ w_3] \left(\int_\Omega \mathbf{B}\mathbf{B}^T dA \right) [u_1\ u_2\ u_3]^T \tag{26}$$

where $w_{1,2,3}$ are the values of w at the node $\mathbf{p}_{1,2,3}$. The matrix $\mathbf{K}_e$ is called the element stiffness matrix which is computed by

$$\mathbf{K}_e = \int_{\text{Triangle}} \mathbf{B}\mathbf{B}^T dA = A\mathbf{B}\mathbf{B}^T \tag{27}$$

where A is the area of the triangle. However, $\mathbf{K}_e$ can be written as a product of simpler matrix $\mathbf{B}_s \neq \mathbf{B}$. This simpler $\mathbf{B}_s$ is given in (4).

Derivation of D

It can be shown that the global stiffness matrix gathers the variation of u around each node. We first develop a geometric interpretation of the element stiffness matrix $\mathbf{K}_e$. Let $\{u_e\} \in \mathbb{R}^3$ be a vector containing the values of u at the three node of a triangle element. Then, we have $\mathbf{K}_e\{u_e\} = A\mathbf{B}\mathbf{B}^T\{u_e\} = A\mathbf{B}\nabla u_e$, where $\nabla u_e \in \mathbb{R}^3$ denotes ∇u, which is constant over the element triangle. In the following discussion, we use the three vector $2A\nabla\xi_i\ , i = 1,2,3$ illustrated in the left of Fig. 14. Notice that three rows of the matrix $2A\mathbf{B} \in \mathbb{R}^{3\times 3}$ are these vectors. Also, we consider $2\mathbf{K}_e\{u_e\}$ instead of $\mathbf{K}_e\{u_e\}$ to make the illustrations in the figure clearer and the following discussions easier.

$2\mathbf{K}_e\{u_e\} = 2A\mathbf{B}\nabla u_e$ is the stack of the three dot products between the gradient ∇u_e and the three vectors $2A\nabla\xi_i\ , i = 1,2,3$. These dot products represent the differences of u along the three direction vectors $2A\nabla\xi_{1,2,3}$. Thus, $2\mathbf{K}_e\{u_e\}$ computes the variation of u from vertex $\mathbf{p}_1$ to the point $\mathbf{q}_{23}$, whose location is shown in the left illustration of Fig. 14. It is on the line perpendicular to $\mathbf{p}_3 - \mathbf{p}_2$ and $|\mathbf{p}_1 - \mathbf{q}_{23}| = |\mathbf{p}_3 - \mathbf{p}_2|$. The same is true for the other two vertices $\mathbf{p}_2$ and $\mathbf{p}_3$. Thus, $2\mathbf{K}_e\{u_e\}$ computes the three variations towards the three nodes.

Now we consider the gloal matrix $\mathbf{K}$. As is shown in the right illustration of Fig. 14, we consider a vertex $\mathbf{p}_i$ and its neighbors $\mathbf{p}_{j_{1,2,3,\ldots,n}}$. For simplicity, we only consider the first triangle with the three vertices $\mathbf{p}_i, \mathbf{p}_{j_1}$ and $\mathbf{p}_{j_2}$. Let $\mathbf{q}_{12}$ be the point in the line passing $\mathbf{p}_i$ and perpendicular to the edge $e_{j_1 j_2}$ and $|\mathbf{p}_i - \mathbf{q}_{12}| = l_{12}$, where $l_{12} \equiv |\mathbf{p}_{j_2} - \mathbf{p}_{j_1}|$. Let u_{12} be the value of u evaluated at $\mathbf{q}_{12}$ using the linear interpolation or extrapolation. From the previous observation, $\mathbf{K}_e\{u_e\}$ returns $(u_i - u_{12})/2$. Since $\mathbf{K}$ is the assembled $\mathbf{K}_e$, the global stiffness matrix $\mathbf{K}$ computes the summation of all such variations corresponding to triangles containing $\mathbf{p}_i$. Now we can write the i^{th} row of $\mathbf{K}\{u\}$ as

$$\text{row}_i(\mathbf{K}\{u\}) = \frac{1}{2}((u_i - u_{12}) + (u_i - u_{23}) + \ldots + (u_i - u_{n1})) \tag{28}$$

Now, we want to prove that this sum of variation leads to the second derivative. Suppose that there exists an approximate parameterization of the local surface $\mathbf{s} = s_1\mathbf{s}_1 + s_2\mathbf{s}_2 + s_n\mathbf{n}$ around $\mathbf{p}_i$, where $\mathbf{s}_{1,2}$

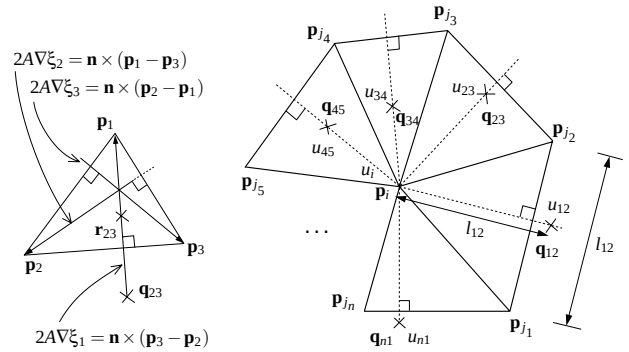

Figure 14: The gemoetric interpretation of $\mathbf{K}_e\{u_e\}$(left) and $\mathbf{K}\{u\}$ (right).

are the two orthonormal local tangent vectors and $\mathbf{n}$ is the normal vector at $\mathbf{p}_i$. Now we can see that each terms of (28) can be approximated using a Taylor series expansion. Let $\mathbf{s} = \mathbf{q}_{12} - \mathbf{p}_i$ then for some constant c, we have $c(u_{12} - u_i) = u(\mathbf{p}_i + c\mathbf{s}) - u(\mathbf{p}_i)$. The taylor series expansion of this yields

$$\begin{aligned} u_{12} - u_i \approx & (u_{,s_1}s_1 + u_{,s_2}s_2) \\ & + \frac{c}{2}\left(u_{,s_1s_1}s_1^2 + 2u_{,s_1s_2}s_1s_2 + u_{,s_2s_2}s_2^2\right) \end{aligned} \tag{29}$$

We assume that u only varies over the surface and has zero gradient along the normal direction. Therefore, $u_{,s_n} \approx 0$. Summing all terms in (28) and simplifying [8]

$$\begin{aligned} \mathrm{row}_i(-\mathbf{K}\{u\}) \approx & \frac{1}{2}\left(u_{,s_1}\sum s_1 + u_{,s_2}\sum s_2\right) \\ & + \frac{c}{4}\left(u_{,s_1s_1}\sum s_1^2 + 2u_{,s_1s_2}\sum s_1s_2 + u_{,s_2s_2}\sum s_2^2\right) \\ = & \frac{cn\bar{l}^2}{8}\left(u_{,s_1s_1} + u_{,s_2s_2}\right) \end{aligned} \tag{30}$$

Note that the operator ∇^2 is invariant under an orthogonal coordinate transform between the global (x,y,z) and the local (s_1,s_2,s_n). We already know $u_{,s_n} \approx 0$. These yields $\nabla^2 u \approx u_{,s_1s_1} + u_{,s_2s_2}$ and consequently

$$\nabla^2 u\Big|_{\mathbf{p}_i} \approx \frac{8}{cn\bar{l}^2}\mathrm{row}_i(-\mathbf{K}\{u\}) \quad = \quad \tilde{c}\,\mathrm{row}_i(-\mathbf{K}\{u\}) \tag{31}$$

Until now, we have shown that the $\mathbf{K}$ can be used as a second derivative. Now we need to decide what value of $\tilde{c}$ to choose. We first considered a local polynomial approximation of u on a flat umbrella and pick an appropriate $\tilde{c}$. This idea yields $\tilde{c}_1 = \frac{4\cos^2(\pi/n_e)}{\tilde{A}}$, where $\tilde{A}$ is the sum of areas of all triangles that share $\mathbf{p}_i$. After noticing the equivalence of the global stiffness matrix $\mathbf{K}$ and the discrete Laplace-Beltrami operator proposed in [Meyer et al. 2003], we found $\tilde{c}_2 = 3/\tilde{A}$ produces identical result. Notice that $\tilde{c}_1 = \tilde{c}_2$ when $n_e = 6$.

We performed experiments on the sphere meshes since the sphere has a mean curvature of $(\kappa_1+\kappa_2)/2 = 1/\rho$, where ρ is the radius. We apply $\tilde{c}_1$ and $\tilde{c}_2$ on several sphere meshes and compare the radius. We chose three types of sphere meshes: the mesh obtained by longitude-latitude discretization and the two other mesh types obtained by subdividing an lcosahedron and a tetrahedron.

We summarize the test in the following table where n_v is the total number of vertices and ρ^* is the true radius of the model, $\bar{\rho}_1$,σ_{ρ_1},$|2\Delta\rho_1|$ are the average, standard deviation, difference between maximum and minimum of the radius computed in all vertices using $\tilde{c}_1$. Similarly $\bar{\rho}_M$, σ_{ρ_M}, $|2\Delta\rho_M|$ are the ones using $\tilde{c}_2$, which corresponds to [Meyer et al. 2003].

As shown in table 1, $\tilde{c}_2$[Meyer et al. 2003] reports a very good estimated radius on average. Interestingly, it recovers the radius of the sphere enclosing the tetrahedron. In comparison, $\tilde{c}_1$ reports a bigger radius, yielding less curvature. This is a drawback since the corner of the tetrahedron is sharp. In contrast, $|\Delta\rho_1|$ is smaller than $|\Delta\rho_2|$ that shows $\tilde{c}_1$ is more consistent. After these investigations, however, we chose $\tilde{c}_2 = 3/\tilde{A}$ in this paper since $\tilde{c}_1$ produces incorrect results at sharp corners.

Discretized by Longitude-latitude

n_v	ρ^*	$\bar{\rho}_1$	σ_{ρ_1}	$\lvert2\Delta\rho_1\rvert$	$\bar{\rho}_2$	σ_{ρ_2}	$\lvert2\Delta\rho_2\rvert$
6	100	**150.0**	8.1e-7	4e-6	100.	5.4e-7	3e-6
14	100	**112.1**	1.00	4.4	99.8	2.28	9.9
26	100	106.7	0.82	6.3	99.3	3.41	**21.6**
42	100	104.7	0.57	7.4	99.2	3.50	**27.2**
422	100	101.7	0.08	14.2	99.8	1.56	**33.1**

Subdivided-Lcosahedron

n_v	ρ^*	$\bar{\rho}_1$	σ_{ρ_1}	$\lvert2\Delta\rho_1\rvert$	$\bar{\rho}_2$	σ_{ρ_2}	$\lvert2\Delta\rho_2\rvert$
12	5	5.73	4.20e-9	1e-6	5.00	3.66e-9	0.00
92	5	5.08	3.44e-3	0.045	4.99	5.37e-3	0.65
252	5	5.03	1.03e-3	0.047	5.00	8.92e-4	0.67

Subdivided-Tetrahedron

n_v	ρ^*	$\bar{\rho}_1$	σ_{ρ_1}	$\lvert2\Delta\rho_1\rvert$	$\bar{\rho}_2$	σ_{ρ_2}	$\lvert2\Delta\rho_2\rvert$
4	5	**15.00**	7.88e-9	0.00	5.00	2.63e-9	0.00
10	5	**7.04**	2.38e-1	1.88	4.86	4.51e-1	**3.56**
34	5	5.39	5.43e-2	0.84	4.93	2.43e-2	**3.96**
202	5	5.06	1.16e-2	0.49	4.99	6.87e-3	**3.67**
802	5	5.02	1.81e-3	0.27	5.00	1.22e-3	**3.55**

Table 1: Comparison of Laplace-Beltrami operators

References

ADALSTEINSSON, D., AND SETHIAN, J. 1995. A fast level set method for propagating interfaces. *Journal Computational Physics 118*, 2, 269–277.

AMIDROR, I. 2002. Scattered data interpolation methods for electronic imaging systems: a survey. *Journal of Electronic Imaging 11*, 2 (April), 157–176.

ANGENENT, S., HAKER, S., TANNENBAUM, A., AND KIKINIS, R. 1999. On the laplace-beltrami operator and brain surface flattening. *IEEE Transactions on Medical Imaging 18*, 8 (August), 700–711.

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *Proceedings of ACM SIGGRAPH*, 43–54.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIC, Z. 2002. A multiresolution framework for dynamic deformations. In *Proceedings of Symposium on Computer Animation*, 41–48.

CHOI, K.-J., AND KO, H.-S. 2002. Stable but responsive cloth. In *Proceedings of ACM SIGGRAPH*, 604–611.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of ACM SIGGRAPH*, 317–324.

[8] We assume that $\mathbf{p}_{j_{1,2,\ldots,n}}$ are evenly distributed orbiting around $\mathbf{p}_i$, *i.e.*, $\mathbf{s} \approx \bar{l}\cos(\theta_0 + 2\pi i/n)\mathbf{s}_1 + \bar{l}\sin(\theta_0 + 2\pi i/n)\mathbf{s}_2$, where $\bar{l} = (l_{12} + l_{23} + \ldots + l_{n1})/n$. Then, we have $\sum s_1 \approx 0$, $\sum s_2 \approx 0$, $\sum s_1 s_2 \approx 0$ and $\sum s_1^2 \approx \sum s_2^2 \approx n\bar{l}^2/2$.

Also, notice the following identities for $n = 3,4,\ldots$ and a constant θ_0

$$\sum_{i=0}^{n}\sin\left(\theta_0 + \frac{2\pi i}{n}\right) = \sum_{i=0}^{n}\cos\left(\theta_0 + \frac{2\pi i}{n}\right) = \sum_{i=0}^{n}\sin\left(\theta_0 + \frac{2\pi i}{n}\right)\cos\left(\theta_0 + \frac{2\pi i}{n}\right) = 0$$

$$\sum_{i=0}^{n}\sin^2\left(\theta_0 + \frac{2\pi i}{n}\right) = \sum_{i=0}^{n}\cos^2\left(\theta_0 + \frac{2\pi i}{n}\right) = \frac{n}{2}$$

DYN, N. 1989. *Interpolation and Approximation by Radial and Related Functions*. Academic press, Boston.

DZIUK, G. 1988. Finite elements for the beltrami operator on arbitrary surfaces. *Lecture Notes in Math. 1357*, 142–155.

FOSKEY, M., OTADUY, M. A., AND LIN, M. C. 2002. Artnova: Touch-enabled 3d model design. In *Preceedings of IEEE Virtual Reality Conference*, 119–126.

GRINSPUN, E., KRYSL, P., AND SCHRÖDER, P. 2002. Charms: a simple framework for adaptive simulation. In *Proceedings of ACM SIGGRAPH*, 281–290.

GRINSPUN, E., HIRANI, A., DESBRUN, M., AND SCHRÖDER, P. 2003. Discrete shells. In *Proceedings of Symposium on Computer Animation*, 62–67.

HUGHES, T. J. R. 2000. *The Finite Element Method – Linear Static and Dynamic Finite Element Analysis*. Dover Publishers, New York.

JAMES, D. L., AND PAI, D. K. 2001. A unified treatment of elastostatic contact simulation for real time haptics. *Haptics-e, The Electronic Journal of Haptics Research (www.haptics-e.org) 2*, 1 (September).

KARNI, Z., AND GOTSMAN, C. 2000. Spectral compression of mesh geometry. In *Proceedings of ACM SIGGRAPH*, 279–286.

KIM, L., KYRIKOU, A., DESBRUN, M., AND SUKHATME, G. 2002. An implicit-based haptic rendering technique. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots*.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of ACM SIGGRAPH*, 105–114.

LEE, S.-Y., CHWA, K.-Y., HAHN, J., AND SHIN, S. Y. 1996. Image morphing using deformation techniques. *The Journal of Visualization and Computer Animation 7*, 1, 3–23.

LEE, S., WOLBERG, G., AND SHIN, S. Y. 1997. Scattered data interpolation with multilevel b-splines. *IEEE Transactions on Visualization and Computer Graphics 3*, 3, 228–244.

LITWINOWICZ, P., AND WILLIAMS, L. 1994. Animating images with drawings. In *ACM SIGGRPH*, 409–412.

MEYER, M., DESBRUN, M., SCHRÖDER, P., AND BARR, A. H. 2003. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, 35–57.

MINSKY, M. D. R. 1995. *Computational Haptics: The Sandpaper System for Synthesizing Texture for a Force-Feedback Display*. PhD thesis, MIT, June.

MULLER, M., MCMILLAN, L., DORSEY, J., JAGNOW, R., AND CUTLER, B. 2002. Stable real-time deformations. In *Proceedings of Symposium on Computer Animation*, 49–54.

NIELSON, G. M., HANGEN, H., AND MULLER, H. 1997. *Scientific Visualization*. IEEE, Newyork.

OHTAKE, Y., BELYAEV, A. G., AND BOGAEVSKI, I. A. 2000. Polyhedral surface smoothing with simultaneous mesh regularization. In *Proceedings of the Geometric Modeling and Processing*, 229–237.

OSHER, S., AND FEDKIW, R. 2003. *Level Set Methods and Dynamic Implicit Surfaces*. Springer.

SALISBURY, K., BROCK, D., MASSIE, T., SWARUP, N., AND ZILLES, C. 1995. Haptic rendering: Programming touch interaction with virtual objects. In *Preceedings of the 1995 Symposium on Interactive 3D Graphics*, 123–130.

SCHNEIDER, R., AND KOBBELT, L. 2001. Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design 18*, 4, 359–379.

SHEPARD, D. 1968. A two dimensional interpolation function for irregularily spaced data. In *Proceedings of the 23rd ACM National Conference*, 517–524.

STAM, J. 2003. Flows on surfaces of arbitrary topology. In *Proceedings of ACM SIGGRAPH*.

TAUBIN, G. 1995. Signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH*, 351–358.

TURK, G., AND O'BRIEN, J. F. 1999. Shape transformation using variational implicit functions. In *Visualization and Mathematics III*, 335–342.

TURK, G. 2001. Texture synthesis on surfaces. In *ACM SIGGRAPH*, 347–354.

YOSHIZAWA, S., BELYAEV, A. G., AND SEIDEL, H.-P. 2003. Free-form skeleton-driven mesh deformations. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, 247–253.