Appeared in Visualization 2000, Salt Lake City, Utah

# Fairing Of Non-Manifolds For Visualization

Andreas Hubeli* Markus Gross*
ETH Zurich

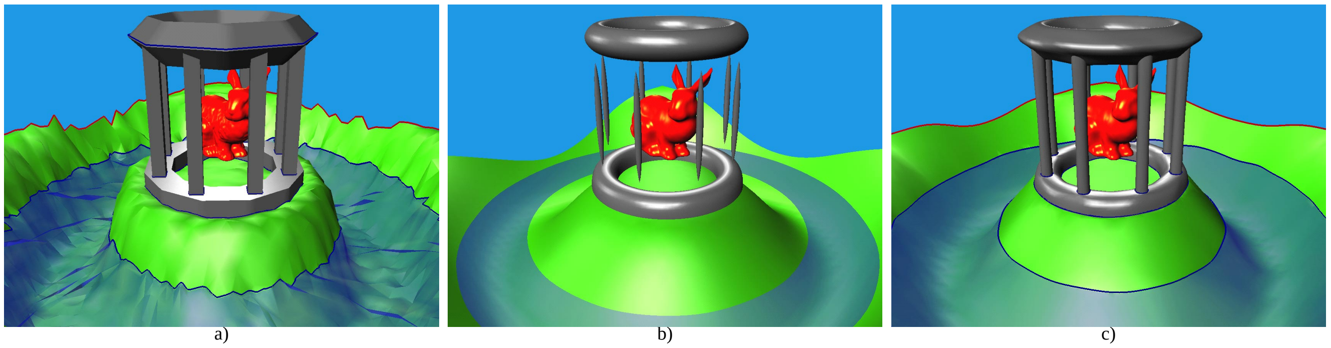
a) b) c)

***Figure 1:*** Fairing of a simple scene (see also CP1):
a) Original input scene.
b) Scene faired in the two-manifold setting: the intersections between water and land and between the columns and the top/bottom of the shrine are lost.
c) Scene faired in the non-manifold setting: the topological type of the scene is preserved, and the model is smooth both along and across intersection curves.

## Abstract

The concept of fairing applied to irregular triangular meshes has become more and more important. Previous contributions constructed better fairing operators, and applied them both to multiresolution editing tools and to multiresolution representations of meshes. In this paper, we generalize these powerful techniques to handle non-manifold models. Our framework computes a multilevel fairing of models by fairing both the two-manifold surfaces that define the model, the so-called two-features, and all the boundary and intersection curves of the model, the so-called one-features. In addition we introduce two extensions that can be used in our framework as well as in manifold fairing concepts: an exact local volume preservation strategy and a method for feature preservation. Our framework works with any of the manifold fairing operators for meshes.

**CR Descriptors:** Boundary Representations, Surface Representations, Non-manifold, Fairing, Geometric Modeling, Triangle Decimation, Multiresolution Models.

## 1 Introduction

### 1.1 Motivation: Model-Centric Graphics

In recent years, with ubiquitous low-priced, high-performance graphics hardware conquering the desktop, the models of many graphics applications are becoming ever more complex. Driven by the need to manage model complexity there has been a convergence of computer graphics and modeling technologies. Rather than maintaining separate representations for modeling and rendering, researchers strive towards *model-centric graphics*, the benefits of which include improved workflows and reduced data loss [1], [10].

Most core fields of computer graphics, such as animation, have concentrated their efforts on working with manifold surfaces, since they can be handled more easily: non-manifold models are inherently more complex to construct and to maintain. The rationale behind this choice has been that the visual quality of a product is of paramount importance, and other considerations, such as the topological consistency of a model, were not thought of as priorities. By taking a model-centric view we free the artist from concerns about topological inconsistencies and construct automatisms that allow him to concentrate on the creative part of his work. We advocate the use of non-manifold models, as built in an advanced modeling framework. As an example, consider figure 1-a that depicts a non-manifold graphics model, where the water, the land, the columns, the top and bottom of the shrine and the bunny were modeled separately as manifold surfaces. If these manifolds are faired independently (figure 1-b), severe artifacts become visible. For instance, the water does not wash against the terrain and the top and bottom of the shrine are not connected to the columns. If the same model is faired in a non-manifold model-centric setting (figure 1-c), the topological type of the model is preserved, and some of its features, such as the shape of the top of the shrine, are preserved more accurately.

A significant step towards model-centric graphics are editing frameworks that build multiresolution hierarchies directly from triangle meshes. As a core feature, users can interactively edit and manipulate meshes at different levels of resolution. A key ingredient of these frameworks is discrete mesh fairing, applying signal processing techniques to meshes. However, advanced modeling frameworks typically build non-manifold models. We address the issue of applying *fairing to non-manifold models*.

In order to further understand the problem of non-manifold fairing, consider the example given in figure 2-a, where two triangular meshes intersect. Applying an adaptation of a manifold fairing method will either remove the intersection completely, depicted in figure 2-b, or it will generate a non-smooth model, as shown in figure 2-c. Our non-manifold fairing method, by contrast, smooths the entire model including the two partial surfaces and the intersection line, as presented in figure 2-d (see section 2.3).

* {hubeli, grossm}@inf.ethz.ch
ETH Zentrum
CH - 8092 Switzerland

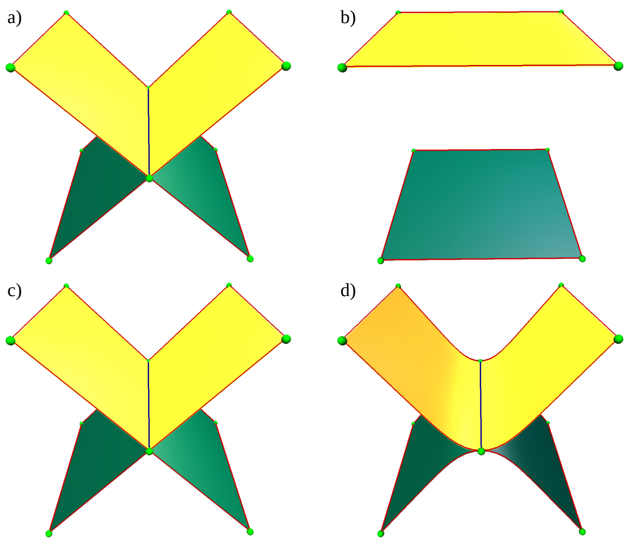

***Figure 2:*** The problem of non-manifold fairing:
a) Initial model.
b) + c) Results obtained using simple adaptations of manifold smoothers.
d) Smooth model computed with our framework.

By extending conventional fairing operators to non-manifolds we provide a framework that can be used to build advanced multiresolution and model-centric graphics representations supporting constraints and other useful functionalities. In fairing non-manifold models we approach a system capable of *automation*.

### 1.2 Related Work

Among the most popular concepts of multiresolution editing frameworks we mention [17]. The authors used Loop subdivision for the estimation of the high resolution mesh from the coarse representation. Another elegant system was devised by [9], who was the first to demonstrate the advantages of a discrete fairing method as a fairing operator for mesh editing. He combined a very fast multilevel smoother with a progressive mesh algorithm for mesh simplification. These two examples show that the key ingredients to design multiresolution mesh representations include both a *fairing or subdivision method* and a *mesh reduction* algorithm.

Of the variety of mesh reduction methods, the most popular ones encompass the progressive mesh of [7] that computes a sequence of progressively refineable meshes by successive application of an edge collapse operator. In combination with appropriate data structures [8] and error metrics [4] this method provides a very powerful representation for triangle meshes. Other popular methods comprise [14] who proposes a vertex removal strategy with a local remeshing method to successively simplify an initially dense mesh.

In order to efficiently build such multiresolution mesh hierarchies, mesh fairing has often been used as a core technology to enhance the smoothness of a mesh. Unlike geometrically motivated approaches to fairing that involve the costly minimization of fairing functionals, [16] pioneered a signal processing approach to mesh fairing. This approach generalizes the notion of *"frequency"* to meshes of arbitrary connectivity by taking the eigenfunctions of the discretized Laplacian. Hence, mesh smoothing can be accomplished by attenuation of the eigenvalues associated to the *"high frequencies"* of the mesh. This type of *"low-pass"* filtering bandlimits the mesh and produces visually appealing models. Since the storage and computational cost is linear in the number of vertices, this approach has become popular for mesh filtering. While mesh fairing can be considered as a diffusion of the perturbations over the mesh surface, [2] proposed a fast and robust implicit fairing scheme using a backwards Euler integration. Another important aspect relates to the quality of the estimation of the surface curvature. Rather than using discretized Laplacians [2] proposed a discrete curvature flow operator. A further important extension of the signal processing approach to triangle meshes has been given in [6]. They elegantly combined non-uniform subdivision with a fairing algorithm to transform an arbitrary mesh into a multiresolution representation, where the details influence the mesh on an increasingly coarse scale. By manipulation of individual scales, they obtain low-, high-, and bandpass behavior, with very impressive results.

We note, however, that most of the described research has been directed towards the handling of arbitrary, but *two-manifold*, meshes. In contrast, relatively little work has been conducted towards multiresolution editing of *non-manifold* models. By abandoning topologically simple models, such as spheres and manifolds, and by tolerating non-manifold geometry, we obtain a new dimension of modeling features and bring in more of the classical Computer Aided Design functionality into graphics modeling. Although non-manifold representations are being widely used in modeling, there has not been any framework for multiresolution editing of non-manifolds introduced so far.

In order to devise such a framework, the described key components have to be extended to tolerate non-manifold geometry. Multiresolution meshes in principle can accommodate topological changes [4] and extend to multidimensional simplices [12]. We are, however, not aware of any fairing framework for non-manifolds.

### 1.3 Our Contribution

In this paper we present a framework for multiresolution fairing of non-manifold models - one of the key ingredients needed to build multiresolution editors. We demonstrate the usefulness of our methods by taking various examples, including subsurface models from the domain of geoscience. The framework comprises the following components:

- A *multilevel fairing algorithm* based on a multiresolution representation of non-manifold models. The flexibility of the basic fairing algorithm allows us to use any discrete Laplacian, curvature flow, or second order difference operator.
- The underlying boundary representation data structure allows us to model *basic constraints*, such as boundary conditions and intersection curves. These constraints are usually extracted automatically from models, but they can also be specified manually by the user.
- The data structure, in conjunction with the basic fairing algorithm, enables us to define *complex constraints*, such as volume preservation and feature preservation easily. The volume preservation strategy is applied locally, thus we can define volumes in a model in a simple and consistent manner. Features are preserved by freezing vertices and curves, without degrading the quality of the fairing process.

Section 2 stresses the difference between surfaces and models and briefly reviews the core concepts of conventional mesh fairing. Section 3 introduces the novel non-manifold fairing algorithm. Finally, in section 4 we address two important extensions to our basic fairing framework: volume and feature preservation.

## 2 Surfaces versus Models

In this section we briefly review some of the existing fairing methods for manifolds. All of the algorithms that we will describe can be extended for usage within our framework to smooth non-manifold models. In addition we will introduce the notions of two-manifold and non-manifold models and stress the specifics of non-manifold fairing.

## 2.1 Fairing Methods for Manifolds

We start our survey of fairing strategies by presenting a one-dimensional fairing operator. Fairing in one dimension is easier to achieve, since we are working in a regular, parametrizable setting. As [2] noted, the standard 1D Laplacian assumes the distance between the three vertices $\boldsymbol{x}_{i-1}$, $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$ as being constant. If the distance between individual vertices varies substantially, the standard Laplacian can be improved using, for instance, generalized divided differences or other concepts from numerical analysis [15], which results in

$$\Delta \boldsymbol{x}_i = \frac{2}{\delta + \Delta}\left(\frac{\boldsymbol{x}_{i-1} - \boldsymbol{x}_i}{\delta} + \frac{\boldsymbol{x}_{i+1} - \boldsymbol{x}_i}{\Delta}\right) \tag{1}$$

where $\delta = \|\boldsymbol{x}_{i-1} - \boldsymbol{x}_i\|$ and $\Delta = \|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\|$ [1].

The fairing of two-dimensional triangular meshes is inherently more complex, since in general meshes are not regular. In the following we summarize the better known fairing operators that have been constructed in recent years.

In his pioneering work [16] generalized the fairing concepts known from signal processing to smooth irregular meshes. He essentially constructed a matrix $K$ that provides a discrete approximation of the Laplacian for all the mesh vertices $\boldsymbol{x}_i$

$$\Delta \boldsymbol{x} = -K \cdot \boldsymbol{x} \tag{2}$$

and he proposed to smooth the mesh $\boldsymbol{x}$ using an iterative *Gaussian filtering* method:

$$\boldsymbol{x}' = (I - \lambda K)\boldsymbol{x} \tag{3}$$

where $I$ is the identity matrix and $0 < \lambda < 1$ is an appropriately selected scalar value. The construction of the matrix $K$ determines the properties of the fairing operators.

[9] constructed a multiresolution editing framework for meshes with arbitrary connectivity. In order to edit surfaces effectively they used a fairing method to remove high frequencies from the mesh. Their fairing algorithm combines a Gaussian smoother with an *umbrella operator* to approximate the Laplacian $\Delta \boldsymbol{x}$. A variational formulation states the fairing problem as a minimization of a *membrane energy* and a *thin plate energy* of a “mesh-function”. In order to discretize the variational formulation using divided difference operators the authors assumed that the vertices in the one-neighborhood of every vertex $\boldsymbol{x}_i$ have a regular parametrization. Under this assumption he could construct the following two operators:

$$\Delta \boldsymbol{x}_i = \frac{1}{n} \sum_{j \in N_1(i)} \boldsymbol{x}_j - \boldsymbol{x}_i \tag{4}$$

which corresponds to a discretization of the Laplacian $\Delta \boldsymbol{x}$ and

$$\Delta^2 \boldsymbol{x}_i = \frac{1}{n} \sum_{j \in N_1(i)} \Delta \boldsymbol{x}_j - \Delta \boldsymbol{x}_i \tag{5}$$

which corresponds to the discretization of $\Delta^2 \boldsymbol{x}$. In (4) and in (5) the vertices $\boldsymbol{x}_j$ lie in the one-neighborhood $N_1(i)$ of $\boldsymbol{x}_i$, and $n$ denotes the number of vertices in $N_1(i)$.

[2] proposed two new, improved operators to smooth two-manifold surfaces. The first one extends equation (4) to better handle meshes with differently sized triangles. To this end, each vertex $\boldsymbol{x}_j$ in the one-neighborhood of $\boldsymbol{x}_i$ is weighted with the length $|e_{i,j}|$ of the edge $e_{i,j}$ between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, yielding:

$$\Delta \boldsymbol{x}_i = \frac{2}{E} \sum_{j \in N_1(i)} \frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{|e_{i,j}|} \text{ with } E = \sum_{j \in N_1(i)} |e_{i,j}| \tag{6}$$

[1] Note that (1) corresponds to the standard Laplacian for $\delta = \Delta$

The second operator introduced in [2] is based on the concept of *curvature flow*. Surfaces are faired by moving the vertices $\boldsymbol{x}_i$ in the mesh along their normals $\boldsymbol{n}_i$ with a speed equal to their mean curvature $\bar{\kappa}_i$. Hence,

$$\Delta \boldsymbol{x}_i = -\bar{\kappa}_i \boldsymbol{n}_i \tag{7}$$

The right hand side of equation (7) can be computed efficiently for a triangular mesh as

$$\Delta \boldsymbol{x}_i = \frac{1}{4A} \sum_{j \in N_1(i)} (\cot\alpha_j + \cot\beta_j)(\boldsymbol{x}_j - \boldsymbol{x}_i) \tag{8}$$

where the $\alpha_j$ and $\beta_j$ are the angles opposite to the edge $e_{i,j}$ in the two triangles that share $e_{i,j}$, and $A$ is the sum of the areas of the triangles in the one-neighborhood of $\boldsymbol{x}_i$.

The authors also presented a more efficient solution of the underlying diffusion equation than (3) by using *implicit integration*, and they devised a strategy to obtain exact global volume preservation during the fairing process.

Finally, [6] described a new fairing algorithm that relies on the minimization of *second order differences* $D_e^2$ defined at every edge *e* in the mesh. The new position of a vertex $\boldsymbol{x}_i$ is chosen as to minimize the sum of the squares of the second order differences within the support of the vertex $\boldsymbol{x}_i$. From this formulation it is possible to obtain a discretized Laplacian that has the form:

$$\Delta \boldsymbol{x}_i = \sum_{j \in \tilde{N}_1(i)} w_{i,j} \boldsymbol{x}_j - \boldsymbol{x}_i \text{, with } w_{i,j} = -\frac{\sum_{j,e} c_{e,i} \cdot c_{e,j}}{\sum_e c_{e,i}^2} \tag{9}$$

where $\tilde{N}_1(i)$ stands for the extended one-neighborhood of $\boldsymbol{x}_i$ with flaps, and the $c_{e,i}$ are a set of coefficients that depend on the geometric position of the vertices in the mesh. More details can be found in [6].

## 2.2 Manifolds and Non-Manifolds

In order to better understand what follows, let us first consider a formal definition: a surface is a *two-manifold* if all of its points have an open neighborhood homeomorphic to $\Re^2$.
This definition can be extended to *two-manifold surfaces with boundaries*, where every point has an open neighborhood homeomorphic to either $\Re^2$ or $\Re^2_+$. More information on *n*-manifolds can be found in [11].

If a surface does not satisfy this criteria, it is called a *non-manifold*. Examples of non-manifold surfaces include for instance self-intersecting surfaces or T-junctions[2]. There are various data structures that describe non-manifolds. In the subsequent framework we use a *boundary representation*, such as the one presented in [13]. This representation describes a model using a graph: three-dimensional volumes are bounded by two-dimensional surfaces, which in turn are bounded by one-dimensional curves. Finally, 1D lines are bounded by zero-dimensional vertices. In general, we are going to call these primitives *n-features*, where $0 \le n \le 3$.

In this paper, we will also assume that the non-manifold models are simplicial complexes, that is, the intersection of two simplices is either empty or a simplex. An *n*-simplex is the convex hull of *n*+1 affine independent vertices: a 0-simplex is a point, a 1-simplex is a segment, a 2-simplex a triangle and so on.

In our framework, we describe a non-manifold model with the following data structure:

- A set of *geometric positions*: every vertex in the model has a unique entry in this set. Non-manifold vertices also have only one entry to guarantee that the model is smoothed correctly.

[2] Note that a T-junction, as presented in figure 3, can be considered as a special case of an intersection of two manifolds.

- A set of *two-features*: a two-feature is defined as a triangulated two-manifold. If we assume that the non-manifold model was generated from *n* two-manifold surfaces, then a two-feature corresponds to one of these two-manifolds plus all the vertices and triangles introduced in the manifold by the computation of the intersection curves.
- A set of *one-features*: one-features are piecewise linear curves defined in one or more two-features. We will distinguish in this paper between two types of one-features: *user-specified* one-features and *representation-specific* one-features. Intersection and boundary curves are examples of representation-specific one-features.
- A set of *zero-features*: zero-features are vertices in the model that must be interpolated. We distinguish between two types of zero-features: *user-specified* zero-features and *representation-specific* zero-features. Vertices shared by two or more one-features are examples of representation-specific zero-features.

The user can force any vertex to be a zero-feature, and any 1D curve to be a one-feature. This will allow us to construct point- and curve- interpolating smoothers (see section 4.2).

Furthermore, we observe that it is possible to identify two different embeddings of oriented (*n*-1)-dimensional manifolds (such as a one-feature) in an oriented *n*-dimensional manifold (such as a two-feature). Consider the intersection curve between the two two-features illustrated in figure 3: for the yellow two-feature it is an interior curve, for which the 2D Laplacian can be computed, and for the green two-feature it is an boundary curve, for which the 2D Laplacian cannot be evaluated without explicit boundary conditions. We define two types of (*n*-1)-dimensional embeddings:

- An (*n*-1)-dimensional manifold is an *l-seam* if an *n*-dimensional Laplacian will be computed for its vertices. Usually this corresponds to (*n*-1)-manifolds embedded in the interior of an *n*-manifold. They will be denoted by blue curves in our examples.
- An (*n*-1)-dimensional manifold is an *l-limit* if an *n*-dimensional Laplacian will not be computed for its vertices. Usually this corresponds to (*n*-1)-manifolds embedded on the boundary of an *n*-manifold. They will be denoted by red curves in our examples.[1]

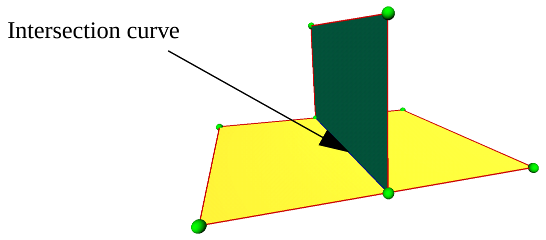

***Figure 3:*** The blue one-feature is an *l-limit* in the green two-feature and an *l-seam* in the yellow two-feature.

### 2.3 Why Manifold Fairing Operators Fail on Models

Unfortunately, a non-manifold model cannot be smoothed by straightforward application of the operators we described previously to all the two-manifold surfaces in the model. The operators from section 2.1 assume that the neighborhood of a vertex $\boldsymbol{x}_i$ is homeomorphic to $\Re^2$ and, as a consequence, they are not defined at non-manifold singularities. Of course, one could envision straightforward extensions of the manifold operators to adapt them to non-manifold models. We discuss two of them using the model illustrated in figure 2-a:

I. A simple adaptation consists in smoothing the model by fairing all the two-features separately. In this approach, each two-feature would have its own geometric information, and the position of the non-manifold vertices would be replicated in multiple two-features. Figure 2-b depicts the result of fairing the input model using this strategy. Since the two-features have been faired separately, and since the information on the l-seam has not been used, the resulting model is optimally smooth, but the l-seam has been entirely removed. In general this approach does not preserve l-seams or l-limits, and the faired models it generates are not simplicial complexes. As a consequence it is required to recompute the intersections between all the two-manifolds - a very costly computational burden that must be avoided.

II. Another strategy is to define the neighborhood of a vertex $\boldsymbol{x}_i$ as the union of all the neighborhoods of $\boldsymbol{x}_i$ in all the two-features where $\boldsymbol{x}_i$ is defined, and then use a standard manifold fairing operator to smooth the model. Figure 2-c presents the faired model obtained when using this method. The model was not smoothed at all. The problem is that the Laplacian of the vertices in the l-seam is equal to zero, since the Laplacians of these vertices in the yellow and green two-features have the same magnitude, but inverse directions.

Finally, figure 2-d shows what we consider a correct result. First of all, the l-seam has been preserved. This is very important, since we usually do not want to change the topology of the model during the fairing process. Furthermore, the two two-features in the model are smooth, also across the l-seam. Finally, all the one-features in the model are smooth as well, and the ten zero-features (the green points) have been interpolated.

In this example we used the Umbrella operator to estimate the Laplacian, and we avoided the shrinking problem by handling the boundaries of the model as l-limits.

## 3 Model-Centric Fairing

In this section we present a novel approach to smooth non-manifold models. In a model we will smooth both the two-features and the one-features simultaneously, and we will achieve cross-l-seams smoothness.

### 3.1 Overview

The manifold smoothers described in section 2.1. remove high frequencies by moving the mesh vertices so as to minimize an approximation of the curvature. Without restricting the generality of our framework, we will work with an approximation of the Laplacian for the following analysis. Other operators could be used as well.

Our goal in smoothing non-manifold models is to fair all the two-features and all the one-features while interpolating the zero-features and guaranteeing cross-l-seams smoothness. As we will explain subsequently, this can be achieved by constructing a framework on top of any conventional manifold smoother. The algorithmic flow of the resulting method is similar to the flow of conventional smoothing algorithms, and is summarized by the following pseudo-code fragment:

```
// Step 1
for n from 1 to 2 do
  for all n-features nF in the model
    for all vert in nF which are not in a l-limit of nF do
      nDLapl[n-feature][vert] = nDLaplacian (vert);

// Step 2
for n from 1 to 2 do
  for all n-features nF in the model do
    for all vert in nF do
      if vert does not belong to a m-feature with m<n do
        computeNewPos (vert, nDLapl);
```

In the first step the approximation of the Laplacian is computed for all the vertices in the model. A 2D Laplacian is computed for all vertices that do not lie in an l-limit, and a 1D Laplacian is computed for all the vertices that are not l-limits in the one-features.

[1] Note that we extended the conventional definitions of *seams* and *limits* from CAD.

Note that it might be necessary to compute multiple Laplacians for a single vertex, such as for vertices in an l-seam, where both a 1D and a 2D Laplacians are needed.

The core of our framework is the function `computeNewPos ()` in the second step, where the new position $\boldsymbol{x}_i'$ for all vertices $i$ must be computed. Immediate use of simple approaches such as (3) would not generate fair models, since cross-l-seam smoothness could not be guaranteed. Instead, the new position $\boldsymbol{x}_i'$ for the vertex $i$ is chosen as to minimize a weighted sum of the Laplacian $\Delta\boldsymbol{x}_i$ *and* the Laplacians $\Delta\boldsymbol{x}_j$ of the vertices $\boldsymbol{x}_j$ in the one-ring of $\boldsymbol{x}_i$, thereby increasing the support of the vertices in the model. The zero vertices in the model are interpolated, the vertices that belong to a one-feature are smoothed with a 1D Laplacian, and the remaining vertices are smoothed with a 2D Laplacian.

Note that depending on the choice of the underlying operator we will minimize different functionals, such as the ones reviewed in section 2.1.

## 3.2 A Fairing Functional for Non-Manifolds

The first step in the pseudo-code algorithm presented in the previous subsection can be computed using any existing fairing operator. The basic principle behind our approach for the second step is to increase the support of a vertex $\boldsymbol{x}_i$, so that during the fairing process it *will not only minimize its own Laplacian* $\Delta\boldsymbol{x}_i$*, but also the Laplacian* $\Delta\boldsymbol{x}_j$ *of its neighboring vertices* $\boldsymbol{x}_j$. Formally, the new position $\boldsymbol{x}_i'$ of a vertex $i$ is computed using (10) for vertices in the one-features and (11) for the remaining vertices.

$$\boldsymbol{x}_i' = \arg\min\,((\omega_{i,i}\cdot\Delta\boldsymbol{x}_i)^2 + (\omega_{i,l}\cdot\Delta\boldsymbol{x}_l)^2 + (\omega_{i,r}\cdot\Delta\boldsymbol{x}_r)^2) \quad (10)$$

$$\boldsymbol{x}_i' = \arg\min\left(\sum_{j\in N_1(i)}(\omega_{i,j}\cdot\Delta\boldsymbol{x}_j)^2 + (\omega_{i,i}\cdot\Delta\boldsymbol{x}_i)^2\right) \quad (11)$$

where $l$ and $r$ are the indices of the vertices to the left and right of $\boldsymbol{x}_i$ in the one-feature respectively, and $\omega_{i,j}$ represents a *weight* associated with the Laplacian $\Delta\boldsymbol{x}_j$ of the vertex $\boldsymbol{x}_j$. Note that the weights control the importance of the curvature of an individual vertex with respect to the fairing process. In the next subsection we will present a strategy to choose these weights.

Equation (10) and (11) increase the support of the vertex $\boldsymbol{x}_i$. For instance, if the original operator had a support over the one-neighborhood of $\boldsymbol{x}_i$, then (11) extends its support over the two-neighborhood of $\boldsymbol{x}_i$. This is a fundamental property, since it allows us to achieve cross-l-seam smoothness without having to move the vertices on the l-seams.

In the following we will show how to derive equation (11). A similar approach can be used to derive equation (10). The new vertex position $\boldsymbol{x}_i'$ is computed by solving an $(n+1)\times 1$ system of equations using a *least squares* method. We start with the following system of equations

$$\begin{aligned}\omega_{i,i}\Delta\boldsymbol{x}_i &= 0\\ \omega_{i,j_1}\Delta\boldsymbol{x}_{j_1} &= 0\\ &\ldots\\ \omega_{i,j_n}\Delta\boldsymbol{x}_{j_n} &= 0\end{aligned} \quad (12)$$

which depicts the ideal solution where all the Laplacians are zero. Next, we need a formulation of the Laplacian $\Delta\boldsymbol{x}_k$ for a vertex $\boldsymbol{x}_k$.

$$\Delta\boldsymbol{x}_k = \sum_{j\in N_1(k)} c_{k,j}\boldsymbol{x}_j - \boldsymbol{x}_k \quad (13)$$

This definition is general enough to represent most of the operators described in section 2.1. If more advanced operators should be devised in the future, it might be necessary to extend the definition.

We can construct a linear system of equations from equation (12) using equation (13), which yields:

$$\begin{bmatrix}\omega_{i,i}\\ -\omega_{i,j_1}c_{j_1,i}\\ \ldots\\ -\omega_{i,j_n}c_{j_n,i}\end{bmatrix}\cdot\begin{bmatrix}\boldsymbol{x}_i\end{bmatrix} = \begin{bmatrix}\omega_{i,i}(\Delta\boldsymbol{x}_i + \boldsymbol{x}_i)\\ \omega_{i,j_1}(\Delta\boldsymbol{x}_{j_1} - c_{j_1,i}\boldsymbol{x}_i)\\ \ldots\\ \omega_{i,j_n}(\Delta\boldsymbol{x}_{j_n} - c_{j_n,i}\boldsymbol{x}_i)\end{bmatrix} \quad (14)$$

Note that if we plug equation (13) in (14) $\boldsymbol{x}_i$ would disappear from the right hand side. Finally, we can compute the new position $\boldsymbol{x}_i'$ of the vertex $i$ that best solves (14) with respect to the two-norm. This can be achieved using a least squares approach that yields the normal equation (15):

$$\boldsymbol{x}_i' = \boldsymbol{x}_i + \frac{\omega_{i,i}^2\Delta\boldsymbol{x}_i - \sum_{k=1}^{n}\omega_{i,j_k}^2 c_{j_k,i}\Delta\boldsymbol{x}_{j_k}}{\omega_{i,i}^2 + \sum_{k=1}^{n}\omega_{i,j_k}^2 c_{j_k,i}} \quad (15)$$

It should be noted that we do not make any assumption about the Laplacians $\Delta\boldsymbol{x}_k$. We simply use these values as approximations of the curvature at every vertex $k$. Equation (15) essentially states the fundamental relationship used to compute the new position $\boldsymbol{x}_k'$ of the vertex $k$ analytically from the old position $\boldsymbol{x}_k$. This confirms our claim that we can use any of the manifold smoothers presented in section 2.1.

Figure 4 visualizes the ideas behind the functional described in this section: figure 4-a shows the Laplacian computed at every vertex using a manifold smoother, and figure 4-b shows the new displacements computed using (15). Finally, figure 4-d illustrates the surface generated by our fairing operator.

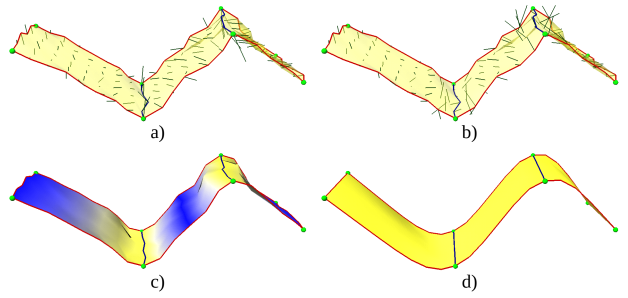

***Figure 4:*** Illustration of the fairing functional (see also CP2):
a) Magnitude and direction of the Laplacian.
b) Displacement of the vertices computed using (15).
c) Color coding of the weights, from small (blue) to high (yellow) values.
d) Smoothed surface that interpolates the l-seams (in blue).

## 3.3 Computation of Weights

In equation (15) a set of weights $\omega_{i,j}$ has been used that control the importance of the different Laplacians. In our framework we developed a simple strategy to select those weights allowing us to achieve reasonable cross-l-seam smoothness.

Our approach is simple: we want to assign larger weight values to vertices that are closer to an l-seam or a zero-feature, and smaller weights to vertices that are farther away. We accomplished this in two steps.

In the first step we compute the distance $d_i$ for each vertex $\boldsymbol{x}_i$ to the closest l-seam or zero-vertex. In our framework we used a topological measure, where the distance between two vertices $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ equals the minimum number of edges traversed to get from $\boldsymbol{x}_i$ to $\boldsymbol{x}_j$.

Next, the weight $\omega_i$ for each vertex $\boldsymbol{x}_i$ is computed as the ratio between the distance $d_i$ and the maximum distance $d_{\max}$ of any mesh vertex to the closest l-seam or zero-vertex whose path goes through $\boldsymbol{x}_i$:

$$\omega_i = W - W \cdot \frac{d_i - 1}{d_{\max} - 1} \tag{16}$$

where $W$ is a user specified maximum weight. Figure 4-c shows a colormap of the weights associated with the vertices in the model: blue corresponds to small values, yellow to high values. We have chosen a linear function to model the weights, since it was sufficient for our needs. However, note that any function can be used in its place.

During the smoothing step of a vertex $\boldsymbol{x}_i$ the weights for a vertex $\boldsymbol{x}_k$ are chosen as follows:

$$\omega_{i,k} = \begin{cases} \omega_i / m & \text{if } k \in N_1(i) \text{ and } (\omega_k > \omega_i) \\ 0 & \text{if } k \in N_1(i) \text{ and } (\omega_k \le \omega_i) \\ (1 - \omega_i) & \text{if } k = i \end{cases} \tag{17}$$

where $m$ is the number of neighbors $j$ of $\boldsymbol{x}_i$ that satisfy $\omega_j > \omega_i$. As a consequence, the new position of the vertex $\boldsymbol{x}_i$ is chosen to minimize its curvature plus the curvature of the vertices that are *closer* to an l-seam or to a zero-feature.

If the model does not contain any l-seam, our operator would educe to the standard two-manifold operator, since the weights would be defined as:

$$\omega_{i,k} = \begin{cases} 0 & \text{if } k \ne i \\ 1 & \text{if } k = i \end{cases} \tag{18}$$

## 3.4 Boundary Conditions

Real-world models usually have boundaries, i.e. they are created by computing the intersections of a set of two-manifolds with boundaries. The boundaries in our models are handled in the same way we handle any one-feature: they are smoothed using a one-dimensional fairing operator, and the zero-features are interpolated. This approach guarantees smooth boundaries, and since the boundaries are handled as one-features, we explicitly avoid having to define special operators for the case where the neighborhood of a vertex $\boldsymbol{x}_i$ is homeomorphic to $\Re^2_+$.

For the time being we are not applying additional constraints on the boundaries, but boundary constraints could be easily included into the model. This could be done for example by specifying the derivative or the curvature at the boundary vertices. Using this information we would treat the boundaries as l-seams, and we would be able to construct approximations of the 2D Laplacians for the boundary vertices.

## 3.5 Multiresolution Representation

In [9] and [6] the fairing operators were used in a multiresolution setting to construct three important applications:

- A multi-level smoother. This is a very important application since, as noted in [16] and [9], explicit solvers based on a Gauss-Seidel iteration scheme only smooth high frequencies efficiently. This effect is caused by the fact that the filtering process attenuates the eigenvectors with largest corresponding eigenvalues referring to the high mesh frequencies. The application of a fairing operator on coarse approximations of the mesh allows us to smooth lower frequencies efficiently.
- A multiresolution representation for meshes, which can be constructed using a prolongation operator $P$ and a set of local frames, as demonstrated in [6]. The operator $P$, which can be considered as a non-uniform subdivision operator, is used to approximate the position $\boldsymbol{x}_i'$ of a vertex $i$ that is being re-introduced in the mesh. The distance between the exact position $\boldsymbol{x}_i$ and $\boldsymbol{x}_i'$ is then stored in a local frame.
- A multiresolution editing tool for meshes with arbitrary connectivity can be built from a fairing operator in conjunction with a set of local frames.

In the following, we introduce the tools required to construct these three applications in our setting, namely an edge collapse operator, a prolongation operator $P$ and a set of local frames.

We constructed a multiresolution representation based on the progressive mesh algorithm presented in [7] and [8]. We extended the edge collapse strategy to meet the underlying boundary representation of our models: vertices that do not lie in a one-feature can be removed using the standard edge collapse operator. Vertices in a one-feature must be removed with special care. A vertex $\boldsymbol{x}_i$ in a one-feature can only be collapsed with its left or right neighbor in the one-feature. After the collapse, the topology of all the two-features sharing the one-feature must be updated. A collapse of a vertex in a one-feature is illustrated in figure 5.

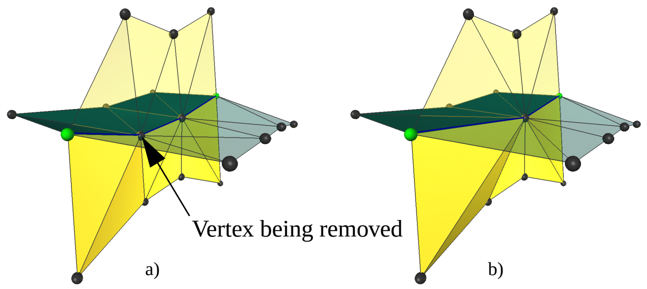

***Figure 5:*** Collapse of a vertex in a one-feature in our boundary representation: a) Before edge collapse. b) After edge collapse.

In order to construct simplified approximations of our models of good quality, we explicitly check if an edge collapse operation introduces degeneracies in the mesh, such as triangles with a very small area or folded triangles. A rigorous analysis of the problem can be found in [3]. It should be noted that the problem of *bubbling* can occur, where the removal of a vertex from a model could introduce new self-intersections in the model. The solution of this problem is outside the scope of our paper.

As a prolongation operator $P$ we can use directly the results from section 3.2 expressed by (15): $P$ inserts a vertex $i$ into the model at the position $\boldsymbol{x}_i'$ that minimizes both the Laplacian of $i$ and of the vertices $\boldsymbol{x}_j$ in its neighborhood.

We implemented the local frames proposed in [9] for our models. It should be noted that we must handle special configurations, such as vertices that lie on an l-limit or an l-seam.

We show in figure 6 how a geological model is smoothed using a multilevel approach, implemented as a full V-cycle, which can be described as

$$\boldsymbol{x}' = ((I - \lambda K) \cdot P)^n \cdot (Q \cdot (I - \lambda K))^n \cdot \boldsymbol{x} \tag{19}$$

The input mesh $\boldsymbol{x}$ (figure 6-a and b) is first smoothed using (15), denoted by $(I - \lambda K)$. Next, the model is simplified using our extension of the progressive mesh scheme, denoted by the operator $Q$. These two operations are repeated $n$ times, until the model is coarse enough (figure 6-c). The coarse model is then refined using the prolongation operator $P$, which will re-introduce vertices into the model. Since we use (15) as our operator, the resulting refinement will be smooth. After that, the model is smoothed once again using (15). These two operations must also be repeated $n$ times, in order to reconstruct a model that has the same connectivity as the input model.

Since the surfaces in the example of figure 6 are height fields, we constructed a robust global parametrization for all the two-features in the model, which allowed us to map textures to the model and avoid tearing problems caused by the drifting of vertices encountered in the fairing process.

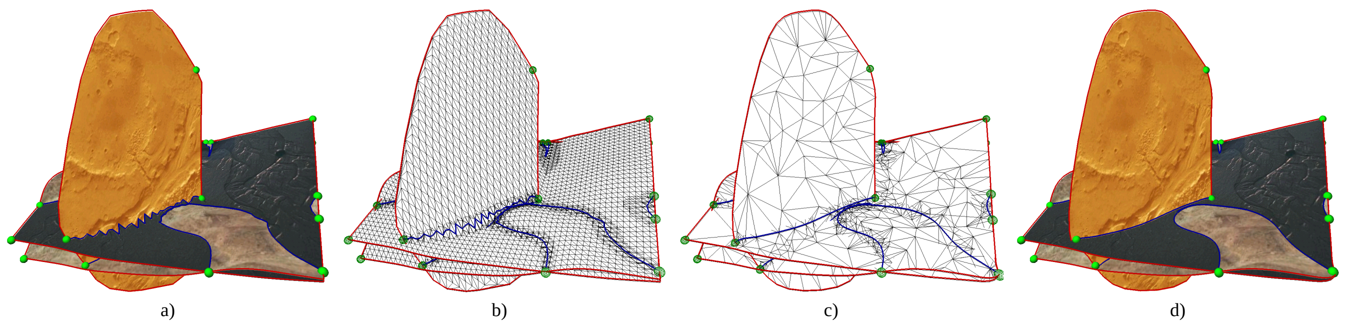

***Figure 6:*** Multilevel fairing of a geological model (see also CP3):
a) Original model. b) Wireframe of the original model. c) Wireframe of the simplified model. d) Smoothed model in full resolution

## 4 Improvements and Extensions

In this section we present two important extensions to improve the quality of the fairing procedure. It should be noted that these approaches can be applied to standard manifold smoothers as well.

### 4.1 Volume Preservation

In the literature, two volume preservation strategies have been discussed:

I. [16] applied an un-shrinking step after each fairing step. An un-shrinking step consists in applying equation (3) using a constant $\mu < -\lambda$ instead of $\lambda$. As Desbrun noted in [2] this strategy will not preserve the volume exactly leading to severe problems in certain types of modeling applications.

II. [2] proposed a new strategy to preserve the global volume of a mesh by computing its volume, and rescaling it after every fairing step to guarantee exact global volume preservation. The problem of this approach is that the local distribution of the volume in the model is not considered in the fairing process.

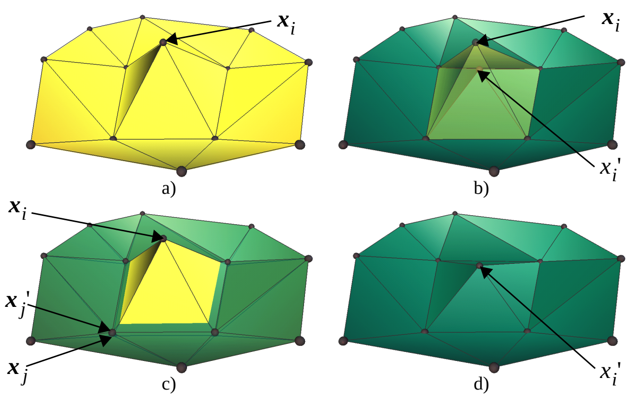

***Figure 7:*** Local volume preservation:
a) Model before fairing. b) The vertex $\boldsymbol{x}_i$ is smoothed.
c) The neighbors vertices $\boldsymbol{x}_j$ of $\boldsymbol{x}_i$ compensate for the change in the volume.
d) Final model after fairing and volume preservation.

In our framework we implemented a local volume preservation strategy derived from a simple observation: when a vertex $\boldsymbol{x}_i$ is smoothed the change of the volume $\Delta V$ can be computed locally. This is accomplished by computing the volumes of the tetrahedra defined by the triangles in the one-neighborhood of $\boldsymbol{x}_i$ and the new position $\boldsymbol{x}_i'$ of $i$. We can compensate for $\Delta V$ by moving the vertices in the one-neighborhood of $\boldsymbol{x}_i$ into the "opposite" direction that we moved $\boldsymbol{x}_i$, as shown in figure 7.

We do not allow the vertices in the one-neighborhood of $\boldsymbol{x}_i$ to move in an arbitrary direction to preserve the volume of the model. Instead we fix one vector, along which all the vertices $j \in N_i(i)$ are moved during the fairing of $\boldsymbol{x}_i$. In our implementation we chose this vector to be the Laplacian $\Delta \boldsymbol{x}_i$ of the vertex $\boldsymbol{x}_i$. This choice is motivated by being able to construct a simple formulation of the volume preservation strategy using a linear system of equations. Hence, the change of the volume $\Delta V$ can be formulated as

$$\Delta V = \sum_{j \in N_1(i)} \left| \begin{bmatrix} \boldsymbol{x}_i & \boldsymbol{x}_i' & \boldsymbol{x}_j & \boldsymbol{x}_{j+1} \\ 1 & 1 & 1 & 1 \end{bmatrix}^T \right| \tag{20}$$

where the operator $|\bullet|$ represents the determinant of the $4 \times 4$ matrices used to evaluate the volume of the associated tetrahedra. We observed reasonable numerical stability, however, it is possible to use alternative approaches, such as [5].

We then enforce each of the vertices $\boldsymbol{x}_j$ in the one-neighborhood of $\boldsymbol{x}_i$ to compensate for a part $0 \le c_j \le 1$ of $\Delta V$, where

$$\sum_{j \in N_1(i)} c_j = 1 \tag{21}$$

Since all the vertices in the one-neighborhood of $\boldsymbol{x}_i$ are moved along the same vector $\Delta \boldsymbol{x}_i$, we must solve

$$\sum_{k \in N_1(j)} \left| \begin{bmatrix} \boldsymbol{x}_j & (\boldsymbol{x}_j + t \cdot \Delta \boldsymbol{x}_i) & \boldsymbol{x}_k & \boldsymbol{x}_{k+1} \\ 1 & 1 & 1 & 1 \end{bmatrix}^T \right| = c_j \cdot \Delta V \tag{22}$$

with respect to *t*. If we expand equation (22) we can construct a linear equation with respect to the unknown $t$. The solution of this linear equation allows us to compute the new position $\boldsymbol{x}_j'$ of $j$ as

$$\boldsymbol{x}_j' = \boldsymbol{x}_j + t \cdot \Delta \boldsymbol{x}_i \tag{23}$$

Equations (22) and (23) compute the new position of the vertex $j$ correctly, since the volume changed by the movement of $\boldsymbol{x}_j$ is independent of the position of the vertices $\boldsymbol{x}_{j-1}$ and $\boldsymbol{x}_{j+1}$.

Figure 8 presents the result of the volume preserving fairing operation as applied to a simple model.

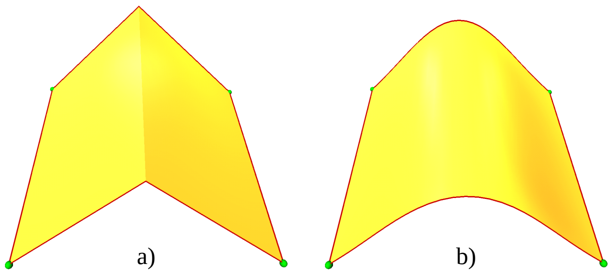

***Figure 8:*** Volume preservation: a) Input model.
b) Faired model using the local volume preservation strategy.

Since our operations are local, the overall shape of the model is not changed. For instance, if the input model is a V-shaped manifold, our algorithm will smooth it while preserving its volume *and* its shape, whereas a global volume preservation strategy would return a flat plane in the limit.

As opposed to [5], we use our local volume preservation strategy in a diffusion process, and therefore we are faced with possible convergence problems. We attacked this problem by minimizing the drifting of the vertices in the model and by choosing $\lambda$ in (3) sufficiently small. This leads to a slower convergence, however, we observed stable results.

### 4.2 Point- and Curve Constraints

One of the prominent applications of mesh fairing is the removal of noise from meshes that are acquired from real world data, such as meshes constructed from laser-scanners or from seismic data. The noise is usually introduced by imperfect acquisition systems, and it has to be removed in order to reconstruct the original shape.

The conventional fairing operators that have been constructed so far do not distinguish between noise and features, however. That is, during the fairing process special features of meshes might be removed. When constructing a simple feature preserving fairing operator, we exploited the properties of our B-rep data structure.

We allow the user to preserve two types of features in a model: *sets of zero-features* and *sets of one-features*. The zero-vertices can be interpreted as the set of vertices that contains the structural information of the model. To prevent these vertices from moving during the fairing process we set these vertices as zero vertices. The use of l-limits in the interior of a two-feature allows us to smooth curves without requiring cross-curve smoothness.

In figure 9 we depict the result of applying this simple strategy to a model. In this example we set six vertices on the top of the V-shaped manifold as zero-features. We observe that the algorithm smoothed the model while preserving its overall shape and the six interpolatory constraints.

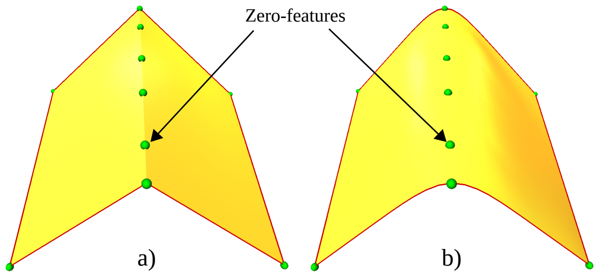

***Figure 9:*** Feature preservation of a model:
a) Input model with no l-seams and ten zero-vertices.
b) Smoothed model that interpolates the zero-vertices.

Taubin presented in [16] a similar approach to compute a smooth interpolation of vertices. The drawback of his approach, however, is that in order to achieve smooth surfaces using interpolatory constraints, it is first necessary to compute a set of smooth surfaces. In a second step, a linear system of equations must be solved having the same size as the number of vertices to be smoothly interpolated. Furthermore, if the fairing operator requires geometric information, this problem must be solved for each iteration. In our approach, the B-rep data structure gives us feature preservation at no additional cost.

## 5 Conclusion and Future Work

In this paper we presented a framework for fairing non-manifold models. We used a multi-level approach to remove both high- and lower frequencies from models by smoothing models at different levels of resolution. We also introduced important extensions of our framework that allow us to guarantee local volume and feature preservation.

We consider the described framework as a core technology enabling us to construct a multi-resolution representation of non-manifold models. Future work comprises improved multiresolution representations of non-manifold models, adaptive visualization algorithms and multi-resolution editing tools. Furthermore, we plan to develop robust texture parametrization methods that are independent of the fairing operator and the topological type of the two-features.

## 6 References

[1] J. Berta. “Integrating VR and CAD.” *IEEE Computer Graphics and Applications*, 19(6):14–19, 1999.

[2] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. “Implicit fairing of irregular meshes using diffusion and curvature flow.” In *SIGGRAPH ’99 Proceedings*, Annual Conference Series, pages 317-324. ACM SIGGRAPH, ACM Press, Aug. 1999.

[3] T. Dey, H. Edelsbrunner, S.Guha, and D.Nekhayev. “Topology preserving edge contraction.” Technical report, Raindrop Geomagic Inc., Research Triangle Park, North Carolina, 1998.

[4] M. Garland and P. S. Heckbert. “Surface simplification using quadric error metrics.” In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, Addison Wesley, Aug. 1997.

[5] A. Guéziec. “Locally toleranced surface simplification.” Technical report, IBM Watson Research Center, Yorktown Heights, N.Y., 1997.

[6] I. Guskov, W. Sweldens, and P. Schröder. “Multiresolution signal processing for meshes.” In *SIGGRAPH ’99 Proceedings*, Computer Graphics Proceedings, Annual Conference Series. ACM SIGGRAPH, ACM Press, Aug. 1999.

[7] H. Hoppe. “Progressive meshes.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, Aug. 1996.

[8] H. Hoppe. “Efficient implementation of progressive meshes.” *Computers & Graphics*, 22(1):27–36, Feb. 1998. ISSN 0097-8493.

[9] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. “Interactive multi-resolution modeling on arbitrary meshes.” In M. F. Cohen, editor, *SIGGRAPH 98 Conference proceedings*, Annual Conference Series, pages 105–114. ACM Press and Addison Wesley, July 1998.

[10] H. Q. Lu and R. Hammersley. “Adaptive visualization for interactive geometric modeling in geoscience.” The 8th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media ’2000, Feb. 2000.

[11] W. Massey. *A Basic Course in Algebraic Topology*. Springer Verlag, 1991.

[12] J. Popovic and H. Hoppe. “Progressive simplicial complexes.” In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 217–224. ACM SIGGRAPH, Addison Wesley, Aug. 1997.

[13] J. Rossignac and M. O’Connor. “A dimension-independent model for pointsets with internal structures and incomplete boundaries.” In *Geometric Modeling for Product Engineering*. M.J. Wozny, J.U. Turner and K. Preiss Eds., North Holland, 1989.

[14] W. Schröder, J. Zarge, and W. Lorensen. “Decimation of triangle meshes.” In SIGGRAPH 92 Conference Proceedings, Annual Conference Series, pages 65–70, July 1992.

[15] J. Stoer and R.Bulirsch. *Introduction to Numerical Analysis*. Springer Verlag, 1993.

[16] G. Taubin. “A signal processing approach to fair surface design.” In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 351–358. ACM SIGGRAPH, Addison Wesley, Aug. 1995.

[17] D. Zorin, P. Schröder, and W. Sweldens. “Interactive multiresolution mesh editing.” In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 259–268. ACM SIGGRAPH, Addison Wesley, Aug. 1997.