Multiresolution Methods for Non-Manifolds Models

Andreas Hubeli, Markus Gross, *Associate Member*, IEEE

Abstract - The concept of fairing applied to triangular meshes with irregular connectivity has become more and more important. Previous contributions proposed a variety of fairing operators for manifolds and applied them to design multiresolution representations and editing tools for meshes. In this paper, we generalize these powerful techniques to handle non-manifold models. We propose a method to construct fairing operators for non-manifolds which is based on standard operators for the manifold setting. Furthermore, we describe novel approaches to guarantee volume preservation. We introduce various multiresolution techniques that allow us to represent, smooth and edit non-manifold models efficiently. Finally, we discuss a semi-automatic feature preservation strategy to retain important model information during the fairing process.

Index Terms - 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], [16].

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 generalizing *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. This example will be discussed further in section 2.3.

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 [24]. The authors used Loop subdivision for the estimation of the high resolution mesh from the coarse representation. Another elegant system was devised by [15], who was the first to demonstrate the advantages of a discrete fairing method 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 [11] that computes a sequence of progressively refineable meshes by successive application of an edge collapse operator. In combination with appropriate data structures [12] and error metrics [7] this method provides a very powerful representation for triangle meshes. Other popular methods comprise [20] who proposes a vertex removal strategy with a local remeshing method to successively simplify an initially dense mesh.

In order to build such multiresolution mesh hierarchies efficiently, mesh fairing has often been used as a core technology to enhance the smoothness of a mesh. Unlike geometrically motivated approaches to fairing involving the costly minimization of fairing functionals, [23] 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 with 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 costs are linear in the number of vertices, this approach has become popular for mesh filtering. While

The authors are with the Department of Computer Science, ETH Zurich, ETH-Zentrum, 8092 Zurich, Switzerland.
E-Mail: {hubeli, grossm}@inf.ethz.ch

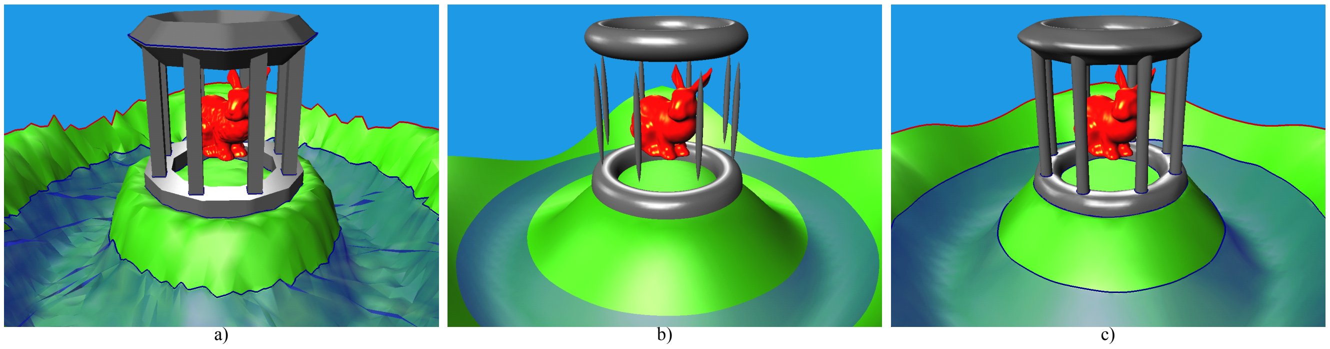


Figure 1: Fairing of an artificial scene:
a) Original input scene.
b) Scene faired in the two-manifold setting: the intersections between water and land and between the columns and the top/bottom of the shrine are lost.
c) Scene faired in the non-manifold setting: the topological type of the scene is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

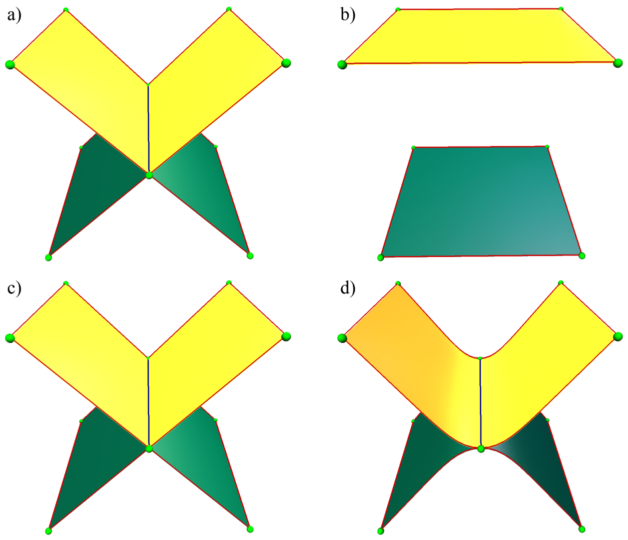


Figure 2: The problem of non-manifold fairing:
a) Initial model.
b -c) Results obtained using simple adaptations of manifold smoothers.
d) Smooth model computed with our framework.

mesh fairing can be considered as a diffusion of the perturbations over the mesh surface, [4] 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 [4] proposed a discrete curvature flow operator. A further important extension of the signal processing approach to triangle meshes has been given in [10]. 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. In fact, multiresolution meshes in principle can accommodate topological changes [7] and extend to multidimensional simplices [18]. We are, however, not aware of any fairing framework for non-manifolds.

1.3 Our Contribution

In this paper we present a framework to construct and interact with non-manifold models. We demonstrate the usefulness of our techniques by taking various examples including subsurface models from the domain of geoscience. The framework comprises the following components:

- A novel *representation for non-manifold models* which allows us to describe a large class of models and to define the semantics of the model.
- A set of *multiresolution techniques* enabling users to interact with large models. A multiresolution representation allows us to construct topologically consistent continuous level-of-detail approximations of our models. Based on existing fairing operators for two-manifolds we design a multilevel fairing algorithm to remove high frequency noise for models [13]. Finally, we construct a multiresolution editing tool for non-manifolds.
- The underlying boundary representation data structure lends itself to model *basic constraints*, such as boundary conditions, as well as *seams* and *limits*. These constraints are usually extracted automatically from our models, but they can also be specified manually by the user.
- The proposed data structure, in conjunction with the fairing algorithm, enables us to define *complex constraints,* such as volume and feature preservation, more easily. Our volume preservation strategy is applied locally. Thus, we can define volumes in the model in a simple and consistent manner. Features are preserved by freezing parts of the model 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 describes the data structure used to represent non-manifold models. Section 4 introduces our novel non-manifold fairing scheme and discusses a new approach to guarantee volume preservation. The multiresolution techniques built into our framework are discussed in section 5. Finally, we outline a feature preservation strategy in section 6.

2 Surfaces versus Models

In this section we briefly review some of the existing fairing methods for manifolds. All of the algorithms being described can be extended for usage within our smoothing framework for 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. In this setting a parameterization is always available and thus the construction of a fairing operator is simplified. As [4] 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 Laplacian can be improved by using, for instance, generalized divided differences or other concepts from numerical analysis [22] resulting 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 most popular fairing operators that have been proposed in recent years.

In his pioneering work [23] generalized the fairing concepts known from signal processing to smooth irregular meshes. He essentially constructed a matrix K providing 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 operator.

[15] 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 differences the authors assumed that the vertices in the one-neighborhood of every vertex $\boldsymbol{x}_i$ have a regular parametrization. Under this assumption we can construct the following two operators:

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

corresponding 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 $N_1(i)$.

[4] 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:

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

The second operator introduced in [4] 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 triangular meshes 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 α_j and β_j are the angles opposite to the edge $e_{i,j}$ in the two triangles sharing $e_{i,j}$, 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*. Furthermore, they devised a strategy for global volume preservation during the fairing process.

Finally, [10] 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 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 compute a discretized Laplacian of type

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

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 depending on the geometric position of the mesh vertices. More details can be found in [10].

Figure 3 depicts the results obtained by applying the four fairing techniques discussed in this section to a closed two-manifold surface. The surface has been built with triangles of different size and with noise added to portions of the mesh. The umbrella and improved umbrella operators generate the results displayed in figure 3.b-c. The high frequency noise has been removed from the mesh, but undesired tangential drifts generated triangles of similar size. By contrast, the curvature flow and second order difference operators produced the results illustrated in figure 3.d-e. In addition to removing high frequency noise, they preserved the relative size of the triangles, i.e. vertices were not allowed to drift in parameter space.

2.2 Manifolds and Non-Manifolds

In order to better understand what follows, let us first consider the definition of an *n*-manifold and a non-manifold surface [17].

Definition: an *n*-dimensional surface is an *n-manifold* if all of its points have an open neighborhood homeomorphic to $\Re^n$.

Definition: a *n*-dimensional surface is an *n-manifold with boundaries*, if every point has an open neighborhood homeomorphic to either $\Re^n$ or $\Re^n_+$.

[1] Note that for $\delta = \Delta$ (1) corresponds to the standard Laplacian using second order divided differences.

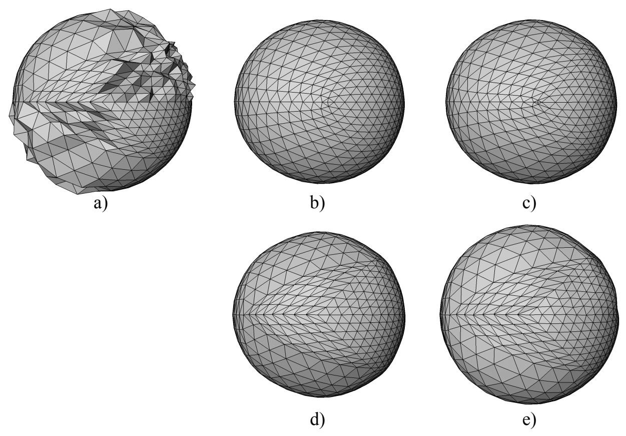


Figure 3: Fairing of manifold models:
a) Input model.
b) Fairing using the umbrella operator.
c) Fairing using the improved umbrella operator.
d) Fairing using the curvature flow operator.
e) Fairing using the second order difference operator.

Definition: a surface that does not satisfy either of the previous definitions is called a *non-manifold*.

Examples of non-manifold surfaces include self-intersecting surfaces and T-junctions[2]. Various structures can be used to represent non-manifolds. In our framework we constructed an extension of the *boundary representation* data structure [19], which describes a model using a graph: 3D volumes are bounded by 2D surfaces, which in turn are bounded by 1D curves. Finally, 1D lines are bounded by zero-dimensional vertices. The specifics of our representation will be discussed in section 3.

Throughout the 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 line segment, a 2-simplex is a triangle, and so on.

A consequence of this assumption is that models are not simply defined as a collection of manifold surfaces. Instead, when a surface is inserted into a model, the representation must be updated in order to guarantee that the new intersection curves are represented by sets of simplices. This boolean operation is performed by identifying the simplices involved in the intersections and by splitting them as needed.

2.3 Why Manifold Fairing Fail on Non-Manifolds

Unfortunately, a non-manifold model cannot be smoothed by straightforward application of the previously described operators to all its two-manifold surfaces. 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. Naively, one could envision straightforward extensions of the manifold operators to adapt them to non-manifold models. In the following we discuss two of them using the model illustrated in figure 2.a.

The first approach might consist in smoothing the model by fairing each surface separately. In this case, both the green and the yellow simplicial complexes store their own geometric information. Hence, the position of the non-manifold vertices defined on the intersection curve would be replicated in the two complexes. Figure 2.b depicts the result when fairing this representation of the input model. Since the green and yellow surfaces have been faired separately and since the intersection curve has not been utilized in the smoothing process, the resulting model is "optimally" smooth. However, the intersection has been removed from the model. That is, this approach does not preserve the topology information of the intersections. Furthermore, the smooth model is no longer a simplicial complex. Therefore, the model needs to be reconstructed explicitly - a costly computational burden that has to be avoided.

A second straightforward approach consists in re-defining the neighborhood of every non-manifold vertex $\boldsymbol{x}_i$ as being the union of all the neighborhoods of $\boldsymbol{x}_i$ in all the simplicial complexes in which $\boldsymbol{x}_i$ is defined. This definition enables us to apply simple fairing operators to the whole model. Figure 2.c presents the results obtained when using this approach. The model was not smoothed at all. The problem is that the Laplacian of the non-manifold vertices on the intersection is equal to zero, since the Laplacians of these vertices in the yellow and green simplicial complexes have the same magnitude, but opposite directions.

Figure 2.d illustrates what we consider a meaningful result. First of all, the intersection curve has to be preserved. This is a key requirement, since we do not accept to change the topology of the model during the fairing process. Furthermore, the two 2D simplicial complexes in the model are smooth, especially across the intersection curve. Finally, all the 1D simplices in the model are smooth, and the ten 0D simplices, represented by the green spheres, are interpolated.

In this example we used the Umbrella operator to estimate the Laplacian, and we avoided the shrinking problem by defining the boundaries of the model to be 1D simplices.

3 Model Representation & Data Structure

In this section we describe the data structure we chose to represent models. We start with a discussion of the requirements that our representation must satisfy. We then present the data structure implemented in our framework and conclude with a discussion of its capabilities.

3.1 Requirements

When designing complex geometric structures, different and possibly opposing requirements can be formulated. On the one hand, a representation has to be as general as possible thus allowing us to describe a large collection of non-manifold models. On the other hand, we need to implement all necessary operators efficiently. The following list summarizes the most important requirements, in order of importance:

- **Non-manifolds**: A useful representation must be able to represent non-manifold models as introduced in section 2.2. This is a strong requirement, since most of our models are non-manifolds.
- **Operators**: It must be possible to implement important operators efficiently, minimizing the storage and computational costs. In particular, we want to define a set of efficient operators to navigate through models.
- **Semantics**: The representation should allow users to describe the semantics associated with a model. By semantics we denote any information which is not inherently defined by the geometry and topology of the model. For instance, the description of the embedding of a curve into a surface should contain both the syntactical information and the meaning of the embedding. Further examples of semantics will be given in section 3.3.
- **Dimension independence**: The representation should be able to describe models of arbitrary dimension. This last requirement has been established for the following reasons: first, when extending the boundary representation data structure, dimension independence should be preserved. Second, many of our operators are dimension independent. Third, some of our models have

[2] Note that a T-junction, as presented in figure 5, can be considered as a special case of an intersection of two manifolds.

components of different dimensionality that we want to represent using the same data structure.

3.2 Data Structure

Our representation is an extension of the boundary representation introduced in [19]. The advantage of using this philosophy is that it automatically satisfies two of the requirements expressed in section 3.1: it is dimension independent and it describes non-manifold models. The data structure we implemented in our framework to represent models is summarized by the following pseudo-code fragment:

```
class Model {
  vector< NVertex >                      geometry;
  vector< NFeature >                     features;
  vector< NCell >                        cells;
  vector< NGeometricRealization >        realizations;
}

class NFeature {
  vector< NGeometricRealization >        realizations;
}

class NCell {
  vector< NSimplex >                     simplices;
  vector< NGeometricRealization >        realizations;
  vector< Embedding >                    embeddings;
}

class NGeometricRealization {
  vector< int >                          localMap;
}

class Embedding {
  short                                  embeddingType;
  NCell                                  lowDimCell;
  vector< NGeometricRealization >        lowDimRealizations;
  vector< NCell >                        highDimCells;
  vector< NGeometricRealization >        highDimRealizations;
}
```

A model is defined by four structures. The first is the geometry vector storing the spatial position of every vertex in the model. By keeping this information at the model level potential problems caused by replication, which are common during operations such as fairing, are avoided entirely. Next, a model is comprised of sets of *n-features*, *n-cells* and *geometric realizations*, its building blocks. We will describe each of them in the next few subsections.

N-Cells

The basic building block of a model is the *n-cell*. An *n*-cell stores the topological structure of the elements of a model - its *n*-dimensional simplicial complexes.

The connectivity information of the complex is encoded by the vector of *n*-dimensional simplices. The indices that define a simplex are not indices pointing to the geometry vector of the model, but rather generic local indices of the *n*-cell. This is important since it allows us to distinguish between the geometric and topological information of a complex.

The vector of embeddings is used to specify how the cell is embedded into the model. For instance, it defines how and in which higher dimensional complexes it is embedded. This data structure will be discussed in detail subsequently.

Finally, an *n*-cell contains a vector of references to geometric realizations which are used to store the geometric positions of the *n*-cell in space.

Definition: a vertex that belongs to an *n*-cell but to no other *m*-cells, with $m < n$, is called an *n-vertex*.

Geometric Realizations

The second building block of the model is the *n*-dimensional *geometric realization*. This structure is used to describe an instance of an *n*-cell in space. Since the simplices that make up a simplicial complex are already defined in the *n*-cell, the realization only needs to store a local map. This map relates the local indices of the simplices to indices in the geometry vector of the model.

As we have seen in the previous section an *n*-cell contains the reference to a vector of geometric realizations. Therefore, the same topological entity can be located at different spatial positions. We will explain in the next sections how this structure can be used to construct general models.

N-Features

The last building block of a model is the *n-feature*. An *n*-feature is defined as a collection of *n*-cells. In our case, *n*-features represent the original input surfaces that were used to build the model. During the construction process any input surface can be split into a collection of *n*-cells. This type of model semantics is stored in the *n*-features.

In cases where no model semantics is needed *n*-features could actually be left out since they do not store any geometric or topological information.

Embeddings

The *embedding* structure defines how an *n*-cell of lower dimension (which by definition is an *n*-manifold) is embedded into a set of *m*-cells of higher dimension (which are *m*-manifolds). In particular, the embedding of an *n*-cell into a set of *m*-cells, $m > n$ defines the complete neighborhood of the vertices of the *n*-cell in the higher dimensional cells.

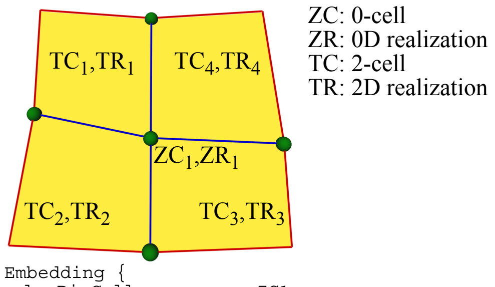
In the simple case where all the *n*-cells possess one geometric realization an embedding is described by a set of *p* pairs

```
< n-Cell,  m-Cell[i] >                               (10)
```

In the general case where an *n*-cell potentially possesses more than one geometric realization an embedding is described by a set of *p* pairs

```
< < n-cell,   n-realization[i] >,
  < m-cell[i],m-realization[i] > >                   (11)
```

Figure 4 illustrates the embedding of a zero-cell into a set of four two-cells.



```
Embedding {
  lowDimCell:             ZC1
  lowCellRealizations: {ZR1, ZR1, ZR1, ZR1}
  highDimCell:         {TC1, TC2, TC3, TC4}
  highDimRealizations: {TR1, TR2, TR3, TR4}
}
```

Figure 4: Embedding of a zero-cell into a set of two-cells.

The information stored in (10) and (11) fully represents the embedding. However, it does not specify how the lower-dimensional *n*-cell influences the higher dimensional *m*-cells in the vicinity of the embedding. The flexibility of the representation is enhanced by defining different types of embeddings:

- An *l-seam* is an embedding that connects the higher dimensional cells through the lower dimensional cell. The fairing framework guarantees cross *n*-cell smoothness if the embedding is of this type.
- An *l-limit* is an embedding that separates the higher dimensional cells bounded by the lower-dimensional cell. The fairing frame-

work does not provide cross n-cell smoothness if the embedding is of this type.

Figure 5 illustrates different types of embeddings: the zero-cells, represented by the green spheres, are embedded into the one- and two-cells as l-limits. The one-cells represented by red lines are also embedded as l-limits into the two-cells. The intersection curve shown in blue is embedded as an l-seam in the yellow two-cells, and as a limit in the green two-cell.

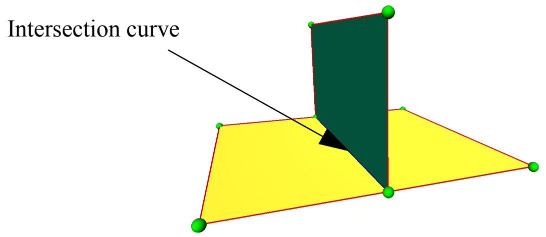


Figure 5: The intersection curve is embedded as an *l-limit* in the green two-cell and as an *l-seam* in the yellow two-cell.

3.3 Discussion

The data structure presented in section 3.2 satisfies the requirements discussed in section 3.1: it is capable to represent non-manifold models and it is dimension independent. Furthermore, the navigation in the model can be performed efficiently, both at the connectivity level and at the level of the n-cells.

The proposed representation is capable of a detailed description of models. The use of geometric realizations allows us to model topologically connected n-features that are disconnected geometrically. Consider the models illustrated in figure 2.a and figure 6.a: topologically, they are identical, except that the intersection curve has one realization in the first model and four in the second.

The use of different embedding types allows us to influence the behavior of the operators applied to a model. Consider the non-manifold model presented in figure 6.a. Different semantics can be associated with the model by changing the type of the embeddings of a single one-cell. Smoothing these syntactically identical models using the same fairing operator results in different end configurations, as illustrated by figure 6.b-d. These represent only a subset of all possible semantics that could be defined on the model shown in figure 6.a. The color coding of the images displays the norm of the Laplacian of the vertices on a logarithmic scale.

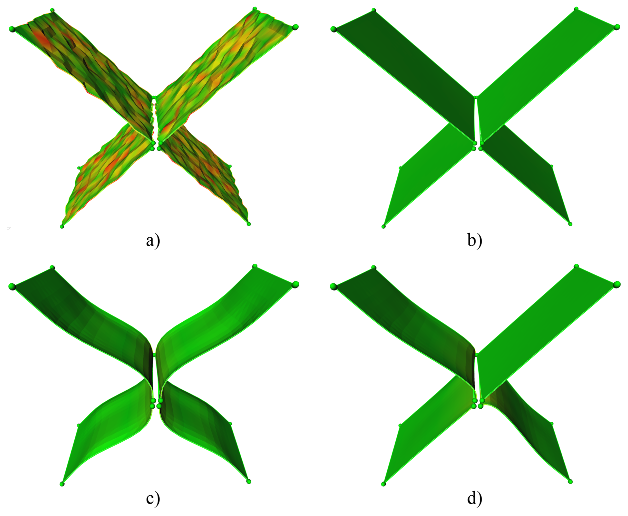


Figure 6: Fairing operator applied to syntactically identical models:
a) Input model.
b) Seams connect the two-cells diagonally.
c) Seams connect the two-cells vertically.
d) Seams connect the bottom-left two-cell to all other two-cells.

This example demonstrates the flexibility and power of a representation capable of describing the *model semantics* in addition to the *model syntax*.

4 Model-Centric Fairing

In this section we present a new set of operators to remove high-frequency information from non-manifold models. Models will be smoothed by attenuating the noise from all the n-cells, while simultaneously achieving cross-l-seams smoothness. Next, we will describe a new set of operators to preserve the volumes in a model exactly. Our operators are inherently dimension independent. However, they are based on manifold operators, which could be dimension specific.

4.1 Overview

The manifold smoothers described in section 2.1. remove high frequencies by displacing the model 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.

The goal in smoothing m-dimensional non-manifold models is to fair all the n-cells, interpolate the zero-cells and guarantee smoothness across l-seams. As we will explain subsequently, this can be achieved by constructing an operator 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 m do
  for all n-cells nC in the model do
    for all vert in nC do
      if vert is not present in an l-limit of nC do
        nDLapl[nC][vert] = nDLaplacian(vert);

// Step 2
for n from 1 to m do
  for all n-cells nC in the model do
    for all vert in nC do
      if vert is an n-vertex then
        position[vert] = computeNewPos(vert, nDLapl[nC]);
```

In the first step, the approximation of the Laplacian is computed for all the vertices in the model. An nD Laplacian is computed for all the vertices present in an n-cell, which do not lie on an l-limit embedded in the cell. By this definition, it might be necessary to compute multiple Laplacians for a vertex, such as for vertices defined in an m-cell embedded in an n-cell as an l-seam, where both an nD and an mD Laplacians need to be computed.

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 cannot be guaranteed. Instead, the new position $\boldsymbol{x}_i'$ for the vertex i is chosen 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. This allows us eventually to smooth all the n-cells in a model, interpolate the zero-cells, and obtain cross-l-seam smoothness.

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

4.2 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 equation (12):

$$\boldsymbol{x}_i' = \arg\min\left(\sum_{j \in N_1(i)} (\omega_{i,j} \cdot \Delta \boldsymbol{x}_j)^2 + (\omega_{i,i} \cdot \Delta \boldsymbol{x}_i)^2\right) \tag{12}$$

The term $\omega_{i,j}$ represents a *weight* associated with the Laplacian $\Delta \boldsymbol{x}_j$ of the vertex $\boldsymbol{x}_j$. Note that the weight $\omega_{i,j}$ controls the importance of the curvature of an individual vertex j with respect to the fairing process. In section 4.4 we will present a strategy to choose these weights.

Equation (12) increases the support of the vertex *i*. For instance, if the original operator had a support over the one-neighborhood of *i*, then (12) extends its support over the two-neighborhood of *i*. This is a fundamental property, since it allows us to achieve cross-l-seam smoothness without having to displace any of the vertices on the l-seams.

We start our investigation by assuming that the Laplacian $\Delta \boldsymbol{x}_k$ for a vertex *k* is computed as

$$\Delta \boldsymbol{x}_i = \sum_{j \in N_1(i)} c_{i,j}\boldsymbol{x}_j - \boldsymbol{x}_i \tag{13}$$

The general definition of the Laplacian given in (13) allows us to represent all the operators described in section 2.1. If new fairing operators will be designed in the future, a redefinition of (13) could be required. Both definitions specified by (12) and (13) are dimension independent. However, since not all the manifold operators are dimension independent, we will assume that the vertex i is a 2-vertex.

The minimization problem defined by (12) and (13) does not have a unique solution. In the following subsection we will present two approaches that minimize the 2-norm and the ∞-norm respectively. Furthermore, we will describe a simple approach to force any vertex i to move only along its Laplacian.

Minimization of the 2-norm

The new vertex position $\boldsymbol{x}_i'$ can be computed by solving an $(n+1) \times 1$ system of equations using a *least squares* method. We start with the following system:

$$\begin{aligned} \omega_{i,i}\Delta \boldsymbol{x}_i &= 0 \\ \omega_{i,j_1}\Delta \boldsymbol{x}_{j_1} &= 0 \\ &\dots \\ \omega_{i,j_n}\Delta \boldsymbol{x}_{j_n} &= 0 \end{aligned} \tag{14}$$

which depicts the ideal solution where all the Laplacians are zero. We can construct a linear system of equations from (14) using definition (13), which yields:

$$\begin{bmatrix} \omega_{i,i} \\ -\omega_{i,j_1}c_{j_1,i} \\ \dots \\ -\omega_{i,j_n}c_{j_n,i} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{x}_i' \end{bmatrix} = \begin{bmatrix} \omega_{i,i}(\Delta \boldsymbol{x}_i + \boldsymbol{x}_i) \\ \omega_{i,j_1}(\Delta \boldsymbol{x}_{j_1} - c_{j_1,i}\boldsymbol{x}_i) \\ \dots \\ \omega_{i,j_n}(\Delta \boldsymbol{x}_{j_n} - c_{j_n,i}\boldsymbol{x}_i) \end{bmatrix} \tag{15}$$

Note that if we plug equation (13) into (15) $\boldsymbol{x}_i$ would disappear from the right hand side of the system.

The new position $\boldsymbol{x}_i'$ of the vertex i, which best solves (15) with respect to the two-norm, can be computed using a least squares approach that yields the normal equation (16):

$$\boldsymbol{x}_i' = \boldsymbol{x}_i + \frac{\omega_{i,i}^2 \Delta \boldsymbol{x}_i - \sum_{k=1}^{n} \omega_{i,j_k}^2 c_{j_k,i} \Delta \boldsymbol{x}_{j_k}}{\omega_{i,i}^2 + \sum_{k=1}^{n} \omega_{i,j_k}^2 c_{j_k,i}} \tag{16}$$

We do not make any assumption about the Laplacians $\Delta \boldsymbol{x}_k$, except for the definition given in (13): the values $\Delta \boldsymbol{x}_k$ are used as approximations of the curvature at every vertex k. Equation (16) essentially states the fundamental relationship used to compute the new position $\boldsymbol{x}_i'$ of the vertex i *analytically* from the old position $\boldsymbol{x}_i$. This confirms our claim that we can use any of the manifold smoothers presented in section 2.1.

The ideas behind the functional described in this section are demonstrated by an example in figure 7: figure 7.a depicts the Laplacian computed at every vertex using a manifold smoother, while figure 7.b shows the new displacements computed using equation (16). Finally, figure 7.d illustrates the smooth surface generated by the fairing operator.

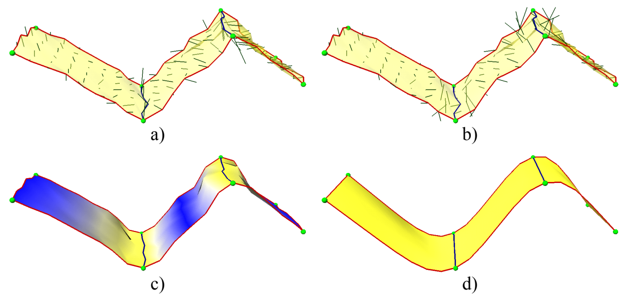


Figure 7: Illustration of the fairing functional:
a) Magnitude and direction of the Laplacian.
b) Displacement of the vertices computed using (16).
c) Color coding of the weights, from small (blue) to high (yellow) values.
d) Smoothed surface that interpolates the l-seams (in blue).

Minimization of the ∞-norm

The new position $\boldsymbol{x}_i'$ of the vertex *i* that minimizes the ∞-norm can be computed by solving an $(n+1) \times 1$ system of equations similar to (14) using the definition (13):

$$\begin{aligned} \boldsymbol{x}_i' - \boldsymbol{x}_i &= \omega_{i,i} \cdot \Delta \boldsymbol{x}_i \\ \boldsymbol{x}_i' - \boldsymbol{x}_i &= \omega_{i,j_1}/c_{j_1,i} \cdot \Delta \boldsymbol{x}_{j_1} \\ &\dots \\ \boldsymbol{x}_i' - \boldsymbol{x}_i &= \omega_{i,j_n}/c_{j_n,i} \cdot \Delta \boldsymbol{x}_{j_n} \end{aligned} \tag{17}$$

This system describes a set of displacement vectors of the vertex i that minimizes the Laplacian of i and the Laplacian of its neighbors, weighted by a factor of $\omega_{i,k}$.

The displacement vector that minimizes the residual of (17) according to the ∞-norm can be constructed by computing the bounding box of all the displacement vectors defined in (17). The new position of the vertex i is then computed from the extrema of the bounding box $\Delta \boldsymbol{x}_{i_{\min}}, \Delta \boldsymbol{x}_{i_{\max}}$ as

$$\boldsymbol{x}_i' = \boldsymbol{x}_i + \frac{1}{2} \cdot (\boldsymbol{x}_{i_{\min}} + \boldsymbol{x}_{i_{\max}}) \tag{18}$$

Displacement along the Laplacian

The two solutions presented in the previous subsection do not necessarily move vertices along their Laplacian. If this is a requirement of the user or the application, a post-processing step can be added to the fairing operator.

Given a vertex i, its old position $\boldsymbol{x}_i$ and the new position $\boldsymbol{x}_i'$ computed with either of the two methods described above, we need to find the vertex $\boldsymbol{x}_i''$ closest to $\boldsymbol{x}_i'$ along the Laplacian of i. This is achieved in two steps. First by solving

$$[\Delta \boldsymbol{x}_i] \cdot t = [\boldsymbol{x}_i' - \boldsymbol{x}_i] \tag{19}$$

using a least squares approach in (x, y, z) and then by computing the resulting vertex position $\boldsymbol{x}_i''$ as

$$\boldsymbol{x}_i'' = \boldsymbol{x}_i + t \cdot \Delta \boldsymbol{x}_i \tag{20}$$

4.3 Volume Preservation

An important extension to the standard fairing technique is volume preservation. The nature of the fairing process, similar to a diffusion process, changes the volume of the model. This side effect can cause problems in particular applications such as fluid flow simulation. Previous contributions solved this problem using two different strategies:

- The first approach, introduced in [23], consists in applying an un-shrinking step after each fairing step. This is accomplished by re-applying equation (3) using a negative constant $\mu < -\lambda$ instead of λ.
- The second approach, presented in [4], preserves the volume defined by a surface globally by first computing its volume, and then rescaling the surface after every fairing step to guarantee exact global volume preservation.

Both approaches are not adequate in the non-manifold setting. The first approach has the drawback of not preserving volumes exactly, as noted in [4]. The second approach is not applicable, since many models do not have only one volume that must be preserved, but possibly a very large number of them.

In the following we will present a novel volume preservation strategy, which is efficient, exact, and can be applied to arbitrarily complex models. Instead of preserving the volume globally, similar to the other two approaches, we will preserve volumes locally. This idea is motivated by the following observation: when a vertex $\boldsymbol{x}_i$ is smoothed the change of the volume ΔV can be computed locally. This is accomplished by calculating 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. Thus, we can compensate for ΔV 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 the example presented in figure 8.

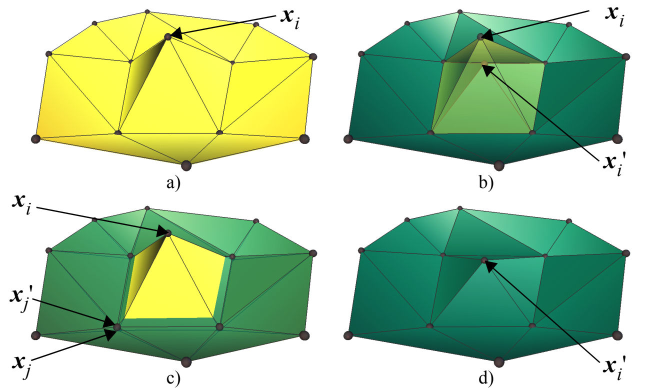


Figure 8: Local volume preservation:
a) Model before fairing.
b) The vertex $\boldsymbol{x}_i$ is smoothed.
c) The neighboring vertices $\boldsymbol{x}_j$ of $\boldsymbol{x}_i$ compensate for the change in volume.
d) Final model after fairing and volume preservation.

Figure 8 illustrates the volume preserving fairing operator step by step. The first step consists in smoothing the vertex $\boldsymbol{x}_i$. The change of the geometric position of $\boldsymbol{x}_i$ introduces a change in the volume, which can be computed efficiently using equation (21).

$$\Delta V = \sum_{j \in N_1(i)} \left| \begin{bmatrix} \boldsymbol{x}_i & \boldsymbol{x}_i' & \boldsymbol{x}_j & \boldsymbol{x}_{j+1} \\ 1 & 1 & 1 & 1 \end{bmatrix}^T \right| \tag{21}$$

where the operator $|\bullet|$ represents the determinant of the 4×4 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 [9].

The change in volume ΔV can be compensated locally by displacing the neighboring vertices of $\boldsymbol{x}_i$, as shown in figure 8.c. We propose three different strategies to perform this operation, providing a trade-off between the computational efficiency and the robustness of the algorithm.

Linear problem

The simplest strategy consists in displacing all the neighboring vertices of $\boldsymbol{x}_i$ along the same vector. In our implementation we chose this vector to be the Laplacian $\Delta \boldsymbol{x}_i$ of the vertex $\boldsymbol{x}_i$. This approach has the advantage of reducing the problem to a linear system of equations that can be solved efficiently.

Each of the vertices $\boldsymbol{x}_j$ in the one-neighborhood of $\boldsymbol{x}_i$ is forced to compensate for a part $0 \le c_j \le 1$ of ΔV, where

$$\sum_{j \in N_1(i)} c_j = 1 \tag{22}$$

Since all the neighboring vertices are moved along the same vector, it can be shown that the displacement vectors can be computed independently for all neighbors. The displacement for the neighbor $\boldsymbol{x}_j$ is computed by solving the linear equation

$$\sum_{k \in N_1(j)} \left| \begin{bmatrix} \boldsymbol{x}_j & (\boldsymbol{x}_j + t \cdot \Delta \boldsymbol{x}_i) & \boldsymbol{x}_k & \boldsymbol{x}_{k+1} \\ 1 & 1 & 1 & 1 \end{bmatrix}^T \right| = c_j \cdot \Delta V \tag{23}$$

with respect to *t*. Hence, the new position $\boldsymbol{x}_j'$ of $\boldsymbol{x}_j$ is computed as

$$\boldsymbol{x}_j' = \boldsymbol{x}_j + t \cdot \Delta \boldsymbol{x}_i \tag{24}$$

The advantage of this approach is that all the neighboring vertices can be handled separately, thus reducing the computations required. However, this technique is not always stable, since all the neighboring vertices are displaced along the same vector, potentially leading to degeneracies in the results.

Pseudo non-linear problem

The robustness of the volume preserving operator can be improved by displacing the neighboring vertices of $\boldsymbol{x}_i$ along different vectors. We decided to move them along their normals in order to minimize the displacement required to compensate for the change in volume ΔV.

This strategy, while being more robust, is less efficient than the linear approach for two reasons: the system of generated equations is non-linear, and the volume preservation problem must be solved for the entire neighborhood of $\boldsymbol{x}_i$ simultaneously. The operator presented in this subsection avoids these two potential bottlenecks by using a simple strategy similar in spirit to Gauss-Seidel: the *j*-th neighbor of $\boldsymbol{x}_i$ is forced to compensate for a part c_j of ΔV, as in the previous approach. This leads to a system similar to (23):

$$\sum_{k \in N_1(j)} \left| \begin{bmatrix} \boldsymbol{x}_j & (\boldsymbol{x}_j + t \cdot \boldsymbol{n}_j) & \boldsymbol{x}_k & \boldsymbol{x}_{k+1} \\ 1 & 1 & 1 & 1 \end{bmatrix}^T \right| = c_j \cdot \Delta V \tag{25}$$

where $\boldsymbol{n}_j$ represents the normal of the vertex $\boldsymbol{x}_j$. In order to preserve the volume exactly, the newly computed position of the first *j-1* neighbors of $\boldsymbol{x}_i$ is used to solve the system (25). We solve with respect to *t*, and compute the new position $\boldsymbol{x}_j'$ of $\boldsymbol{x}_j$ as

$$x_j' = x_j + t \cdot n_j \tag{26}$$

This second approach, while being non-linear in nature, uses the same linear system as the one described in the previous subsection. It guarantees an exact volume preservation, and it is efficient. However, the displacement of the neighbors of x_i is not symmetric, since the vertices are moved using a Gauss-Seidel-like strategy, which could lead to unsatisfactory results in certain applications.

Non-linear problem

The best quality is obtained by solving the non-linear volume preservation problem. As in the previous approach, vertices are moved along their normals to compensate for the change in volume ΔV.

The volume preservation problem can be formulated as follows: the neighboring vertices of x_i are displaced along their normals at the same speed, until the change in volume ΔV is compensated. In the simplest configuration, where the neighborhood of x_i is convex, the change in volume introduced by the displacement of the adjacent vertices of x_i is described by (27):

$$\Delta V = \sum_{j \in N_1(i)} \sum_{k \in N_1(j)} |A_{j,k}| \tag{27}$$

where the matrix $A_{j,k}$ is defined as

$$A_{j,k} = \begin{bmatrix} x_j & (x_j + t \cdot n_j) & (x_k + t \cdot n_k) & x_{k+1} \\ 1 & 1 & 1 & 1 \end{bmatrix}^T \tag{28}$$

if the k-th neighbor of x_j is also the *(j-1)*-th neighbor of x_i, and

$$A_{j,k} = \begin{bmatrix} x_j & (x_j + t \cdot n_j) & x_k & x_{k+1} \\ 1 & 1 & 1 & 1 \end{bmatrix}^T \tag{29}$$

otherwise. The determinant of the matrix defined in (28) leads to a quadratic polynomial in t, whereas the determinant of the matrix defined in (29) provides a linear polynomial in t. The resulting system can be formulated as

$$a \cdot t^2 + b \cdot t = \Delta V \tag{30}$$

In general, equation (30) has two solutions which describe how to displace the neighboring vertices to achieve volume preservation. We choose the solution with smallest absolute value, since it minimizes the norm of the displacement vectors. Their new position is then computed using (26).

Example

Since the operations of the three approaches are local, the overall shape of the model is not changed by the volume preserving fairing operator. Consider the simple V-shaped manifold depicted in figure 9: if we apply our volume preservation operators to this model they 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 [4], the local volume preservation strategies we discussed are part of a diffusion process. Thus, we are faced with potential convergence problems. We did not observe any degeneracies, especially when using the pseudo-linear and the non-linear operators. However, if the problem occurs, we recommend to attack it by choosing λ in equation (3) sufficiently small. This leads to a slower convergence, but it increases the stability of the Euler iteration.

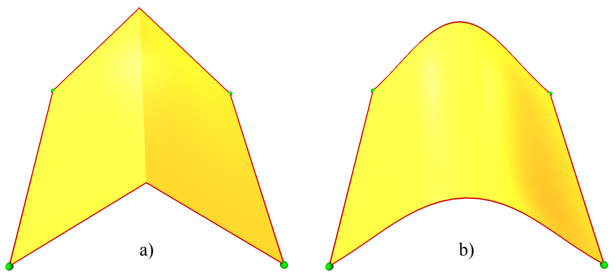


Figure 9: Volume preservation:
a) Input model.
b) Faired model using the local volume preservation strategy.

4.4 Computation of Weights

In equation (14) and (17) a set of weights $\omega_{i,j}$ has been used that controls 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 embeddings of type l-seam, and smaller weights to vertices that are farther away. We accomplish this in two steps.

We first compute the distance d_i from each vertex x_i to the closest l-seam. For the examples depicted in this paper we used a topological measure, where the distance between two vertices x_i and x_j equals the minimum number of edges traversed to get from x_i to x_j.

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

$$\omega_i = W \cdot \left(1 - \frac{d_i - 1}{d_{\max} - 1}\right) \tag{31}$$

where W is a user specified maximum weight. Figure 7.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, our framework can use any other function in its place.

During the smoothing step of a vertex i the weight associated with a neighboring vertex k is chosen as follows:

$$\omega_{i,k} = \begin{cases} \omega_i / m & \text{if } k \in N_1(i) \text{ and } (\omega_k > \omega_i) \\ 0 & \text{if } k \in N_1(i) \text{ and } (\omega_k \le \omega_i) \\ (1 - \omega_i) & \text{if } k = i \end{cases} \tag{32}$$

where m is the number of neighbors j of i that satisfy $\omega_j > \omega_i$. As a consequence, the new position of the vertex x_i is chosen to minimize its curvature plus the curvature of the vertices that are *closer* to an l-seam.

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

$$\omega_{i,k} = \begin{cases} 0 & \text{if } k \ne i \\ 1 & \text{if } k = i \end{cases} \tag{33}$$

4.5 Boundary Conditions

Real-world models usually have boundaries, i.e. they are created from a set of n-dimensional manifolds with boundaries. The boundaries in our models are handled in the same way we handle

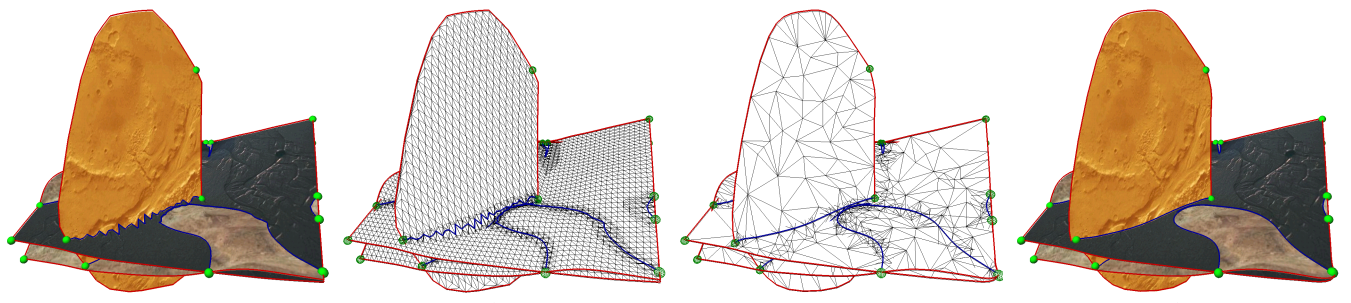


Figure 10: Multilevel fairing of a geological model:
a) Original model.
b) Wireframe of the original model.
c) Wireframe of the simplified model.
d) Smoothed model in full resolution.

any other m-cell: they are smoothed using an m-dimensional fairing operator. This approach guarantees smooth boundaries, and since they are represented as embeddings in a higher dimensional n-cell, no special operator has to be defined for the case where the neighborhood of a vertex $\boldsymbol{x}_i$ is homeomorphic to $\Re^n_+$.

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

5 Multiresolution Techniques

The models used in many applications can be arbitrarily large and complex, both topologically and in the number of vertices and simplices present in them. In order to handle very large datasets, multiresolution techniques are a key ingredient of any representation. In the following we introduce three concepts for interactive modeling: a multiresolution representation of models, a multilevel fairing technique, and a multiresolution editing tool.

5.1 Multiresolution Representation

A multiresolution representation is an important component that allows us to construct approximations of the input models, a feature required in interactive modeling and visualization systems.

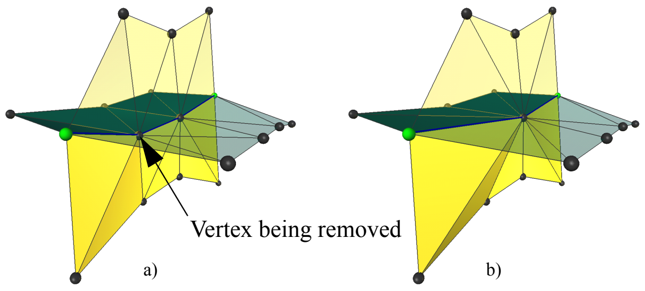


Figure 11: Collapse of a vertex of a one-feature in our boundary representation:
a) Before edge collapse.
b) After edge collapse.

In our framework, we extended the progressive mesh algorithm described in [11] and [12] to handle non-manifold models. We modified the edge collapse operator to meet the underlying boundary representation of our models: n-vertices can only be collapsed into m-vertices, $m \le n$. For example a one-vertex is to be collapsed into one of its two neighbors in a one-cell, but with no two-vertex in a two-cell. The collapse of an m-vertex does not only change the connectivity information of its m-cell, but also all the higher dimensional n-cells where the m-cell is embedded, as demonstrated by the example depicted in figure 11.

Although in theory the multiresolution representation could be defined for n-dimensional models, we confine our discussion to models that only contain n-cell, $n \le 2$.

During the construction of the multiresolution representation, we guided the simplification process in order to generate approximations of good quality. In particular, we checked for degeneracies in the mesh, such as folded or badly shaped triangles. A rigorous analysis of the problem can be found in [5]. It should be noted that *bubbling* can occur, where the removal of a vertex from a model introduces new self-intersections. The solution to this problem is outside the scope of our paper; if it has to be avoided, global tests must be performed after every edge collapse operation [21].

The geometric position $\boldsymbol{x}_i$ of the vertex i removed by an edge collapse can be encoded in different ways: by storing its absolute position, its relative position, or by using *local frames* [6], [15]. In our implementation we use the latter one and a generic prolongation operator P, which can be considered as a non-uniform subdivision operator, to approximate the position $\boldsymbol{x}_i'$ of the vertex i that is being re-introduced into the mesh. The distance between the exact position $\boldsymbol{x}_i$ and $\boldsymbol{x}_i'$ is then stored in a local frame.

5.2 Multilevel Fairing

A multilevel fairing scheme has several advantages over flat fairing approaches. First of all, models can be smoothed in fewer iterations, thus speeding up the algorithm. More importantly, explicit solvers based on a Gauss-Seidel iteration scheme only smooth high frequencies efficiently. This is caused by the fact that the filtering process attenuates the eigenvectors with largest corresponding eigenvalues referring to the high mesh frequencies [8]. Lower frequencies however can be attenuated effectively by applying the fairing operator on a coarse approximation of the input model.

We implemented a multilevel fairing operator as a full V-cycle, which can be described as

$$\boldsymbol{x}' = ((I - \lambda K) \cdot P)^n \cdot (Q \cdot (I - \lambda K))^n \cdot \boldsymbol{x} \tag{34}$$

The input model $\boldsymbol{x}$ is first smoothed using (16), denoted by the term $(I - \lambda K)$ in (34). Next, the model is simplified using our extension of the progressive mesh scheme, denoted by the operator Q. These two operations are applied n times, until we obtain a coarse approximation of the model. This approximation is then refined using the prolongation operator P, which re-introduces vertices into the model. Since we use (16) as our prolongation operator, the resulting refinement will be smooth. After that, the model is smoothed once again using (16). These two operations

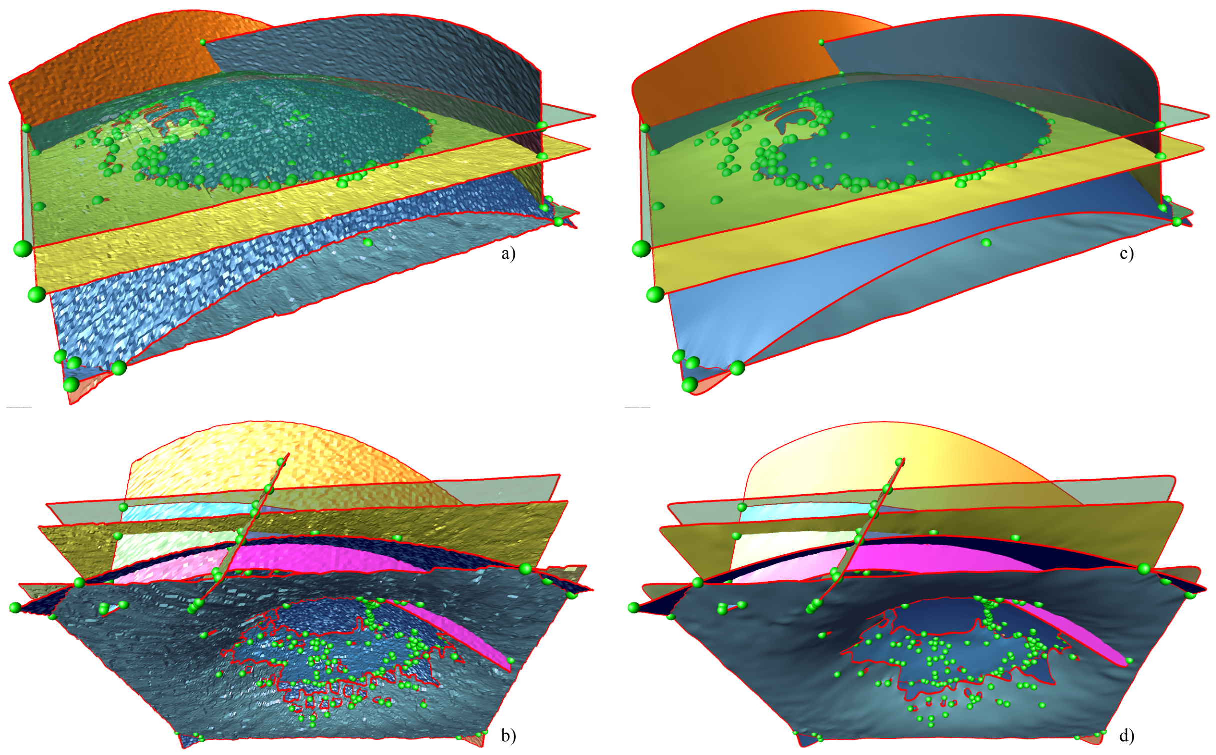


Figure 12: Fairing of a complex geological model:
a-b) Input model visualized from different viewpoints.
c-d) Faired model visualized from the same viewpoints.

must also be repeated n times to reconstruct a model that has the same connectivity as the input model.

We show in figure 10 how a geological model is smoothed using this approach: figure 10.a-b illustrate the input model. After the first half of the V-cycle has been computed, we obtain a coarse approximation of the model, depicted by figure 10.c. Finally, the second half of the V-cycle is performed, providing the full-resolution smooth model shown in figure 10.d.

Since the surfaces in the example of figure 10 are height fields, we constructed a robust global parametrization for all the two-features in the model. This allowed us to map textures to the model and to avoid tearing problems caused by the drifting of vertices encountered during the fairing process.

Figure 12.a-b present a more complex geological model that was faired using our techniques. The input model has been constructed from three real-world layer interfaces, an artificial salt dome and two artificial faults. The model generated by these six two-manifolds consists of 216'316 vertices, 253 two-cells, 407 one-cells, and 342 zero-cells. The framework automatically reconstructed six two-features, the input surfaces and 292 one-features. Figure 12.c-d depict the smoothed model from the same viewing directions than figure 12.a-b.

5.3 Multiresolution Editing

A fundamental functionality of any modeling system is an editing tool. Although extensive literature is available on editing techniques for manifold surfaces, including regular [6], semi-regular [24] and irregular [15] connectivity, editing tools for non-manifold models are not as common. The standard solution consists in pulling the two-manifold components out of a model, edit them outside the model and re-insert them into the model afterwards.

Our approach to editing is to extend standard editing techniques for two-manifolds to the non-manifold setting, and provide some of the basic functionalities. The main advantage of model centric editing is that changes can be immediately propagated into the model and are not confined to a single cell.

Ideally, edit operations should not affect the high frequency information of a model, but only change the structural low-frequencies. With this in mind, an edit operation is performed in four steps:

I. First, the user is required to define a handle which will be used in subsequent edit operations. This object is represented internally either as a zero- or one-cell embedded as an l-seam in the two-cell that is being edited.
II. The high frequency information is stored in local frames. This is accomplished by computing the smooth base domain of the model using the multilevel fairing operator defined in section 5.2. The details that describe the difference between the input model and its base domain correspond to the high frequency information that must be encoded. We refer the user to [15], where a similar technique is described for manifold meshes.
III. The model is edited by moving the handle in the model, either by translating or rotating all the vertices in the handle.
IV. The position of all the vertices in the model is recomputed each time the handle position changes. The new base domain of the model is computed using the multilevel fairing operator by freezing the user-defined position of the handle vertices, and the high frequency information is extracted from the local frames.

Figure 13 illustrates how a model is edited, step by step. The input model is depicted in figure 13.a. The user-defined handle and the base domain of the model are illustrated in figure 13.b. After the handle has been moved, a new base domain that interpolates the edited handle is computed as shown in figure 13.c. In a final step, the high frequency information encoded in local frames is reintroduced into the model, as depicted in figure 13.d. The color coding of the models in figure 13 show the magnitude of the Laplacian at every vertex in the model, using a logarithmic scale. This information is particularly useful to compare the input model and the edited model: the structural information changed, but the high frequency details have been preserved.

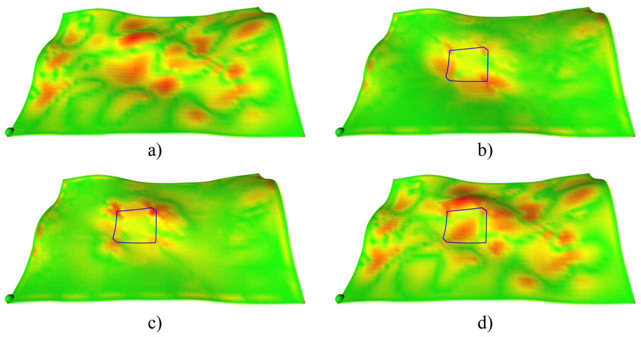


Figure 13: Multiresolution editing of a model:
a) Input model.
b) Base domain of the input model.
c) New base domain that interpolates the edited handle.
d) Modified model.

6 Feature Preservation

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. Although alternate approaches based on anisotropic diffusion exist [3], it is difficult to obtain high quality results that preserve features.

In this section we will briefly discuss two issues related to feature preservation: the *detection of features* in the model and a strategy to guarantee *feature-preservation* in the smoothing process. Feature information is either provided by the user, or it must be extracted automatically. In our framework we use the semi-automatic extraction technique presented in [14], which can be applied to two-manifold surfaces with arbitrary connectivity. Additional details on the detection step are outside the scope of this paper and can be found in the reference.

In our framework, we approach the feature preservation problem by exploiting the properties of our boundary representation data structure. Features are modeled as *n*-cells, and in particular as zero- or one-cells embedded in higher dimensional cells. The type of embedding specifies how features are to be preserved: features embedded as l-limits will not provide cross-feature smoothness, while features embedded as l-seams will provide cross-feature smoothness. Note that either type of embedding will preserve the feature: a zero-vertex will be interpolated regardless of how it is embedded into a higher dimensional cell.

In figure 14 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-vertices. We observe that the algorithm smoothed the model while preserving its overall shape and the six interpolatory constraints.

Taubin presented in [23] a different 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 itera-

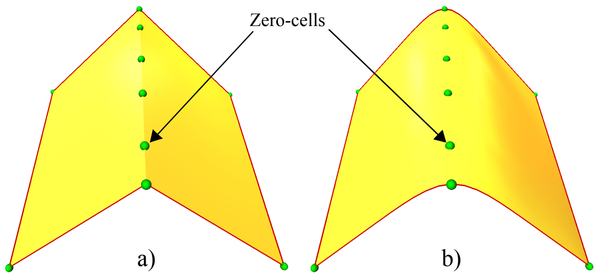


Figure 14: Feature preservation of a model:
a) Input model with no l-seams and ten zero-vertices.
b) Smoothed model that interpolates the zero-vertices.

tion. In our approach, the boundary representation data structure gives us feature preservation at no additional cost.

7 Conclusion and Future Work

In this paper we presented a framework for the representation of non-manifold models based on fairing operators. We constructed a set of fairing operators for models, and used them to define advanced modeling tools, including multi-level smoothing and editing operators. We also introduced important extensions to our framework that allow us to guarantee 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 the construction of new manifold operators, adaptive visualization techniques and a feature-rich modeling system. Furthermore, we plan to develop robust texture parametrization methods that are independent of the fairing operator and of the topological type of the two-cells.

Acknowledgments

This research was made possible by Schlumberger ATC. The authors would like to thank Richard Hammersley, Karen Lu and Christoph Ramshorn for their helpful comments on the text.

8 References

[1] J. Berta. "Integrating vr and cad." *IEEE Computer Graphics and Applications*, 19(6):14–19, 1999.

[2] H. Biermann, A. Levin, and D. Zorin. "Piecewise smooth subdivision surfaces with normal control." In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 113–120. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[3] U. Clarenz, U. Diewald, and M. Rumpf. "Anisotropic geometric diffusion in surface processing." In *Proc. of the 11th Ann. IEEE Visualization Conference (Vis) 2000*, 2000.

[4] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. "Implicit fairing of irregular meshes using diffusion and curvature flow." In *SIGGRAPH '99 Proceedings*, Computer Graphics Proceedings, Annual Conference Series. ACM SIGGRAPH, ACM Press, Aug. 1999.

[5] T. Dey, H. Edelsbrunner, S.Guha, and D.Nekhayev. "Topology preserving edge contraction." Technical report, Raindrop geomagic Inc., Research Triangle Park, North Carolina, 1998.

[6] D. R. Forsey and R. H. Bartels. "Hierarchical B-spline refinement." *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):205–212, Aug. 1988.

[7] 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. ISBN 0-89791-896-7.

[8] M. H. Gross and A. Hubeli. “Eigenmeshes.” Technical report, ETH Zurich, Mar. 2000.

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

[10] 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.

[11] H. Hoppe. “Progressive meshes.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

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

[13] A. Hubeli and M. Gross. “Fairing of non-manifolds for visualization.” In *Proc. of the 11th Ann. IEEE Visualization Conference (Vis) 2000*, 2000.

[14] A. Hubeli, K. Meyer, and M. H. Gross. “Mesh edge detection.” Technical report, ETH Zurich, 2001.

[15] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. “Interactive multi-resolution modeling on arbitrary meshes.” In M. F. Cohen, editor, *Computer graphics: proceedings: SIGGRAPH 98 Conference proceedings, July 19–24, 1998*, Computer Graphics -proceedings-1998, pages 105–114, New York, NY 10036, USA and Reading, MA, USA, 1998. ACM Press and Addison Wesley.

[16] H. 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, 2000.

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

[18] 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. ISBN 0-89791-896-7.

[19] 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.

[20] W. J. Schröder, J. A. Zarge, and W. E. Lorensen. “Decimation of triangle meshes.” In E. E. Catmull, editor, *Computer Graphics (SIGGRAPH ’92 Proceedings)*, volume 26, pages 65–70, July 1992.

[21] O. G. Staadt and M. H. Gross. “Progressive tetrahedralizations.” In *Proceedings IEEE Visualization ’98*, pages 397–402. IEEE, 1998.

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

[23] 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. held in Los Angeles, California, 06-11 August 1995.

[24] 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. ISBN 0-89791-896-7.