EUROGRAPHICS '99 / P. Brunet and R. Scopigno
(Guest Editors)

*Volume 18 (1999), Number 3*

# Improved Laplacian Smoothing of Noisy Surface Meshes

J. Vollmer, R. Mencl, and H. Müller

Informatik VII (Computer Graphics), University of Dortmund, Germany

**Abstract**

*This paper presents a technique for smoothing polygonal surface meshes that avoids the well-known problem of deformation and shrinkage caused by many smoothing methods, like e.g. the Laplacian algorithm. The basic idea is to push the vertices of the smoothed mesh back towards their previous locations. This technique can be also used in order to smooth unstructured point sets, by reconstructing a surface mesh to which the smoothing technique is applied. The key observation is that a surface mesh which is not necessarily topologically correct, but which can efficiently be reconstructed, is sufficient for that purpose.*

## 1. Introduction

A useful approach to acquire complex geometric models in computer graphics is digitization. From the cloud of points scanned by digitizing devices like laser scanners or tactile scanners, a surface description may be obtained by connecting the points in an appropriate manner into a surface mesh, e.g. a mesh of triangles.

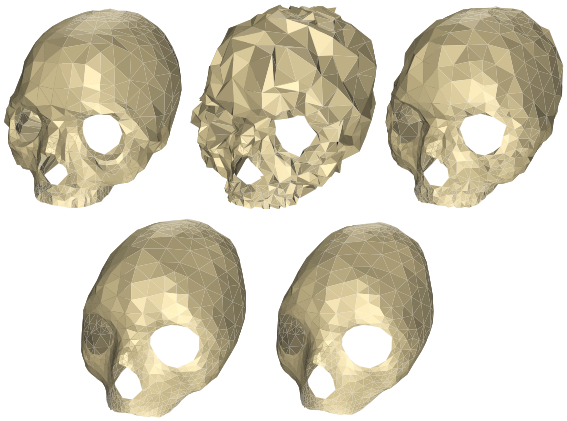


**Figure 1:** *From left to right: origin, noisy mesh, three iterations of the Laplacian algorithm.*

Unfortunately, the digitized points often do not reflect the correct location on the real surface, because of physical noise added by the technical scanning device. The effect is that the reconstructed surfaces often do not look satisfactory. The consequence is that the mesh has to be smoothed, with the goal to remove the noise.

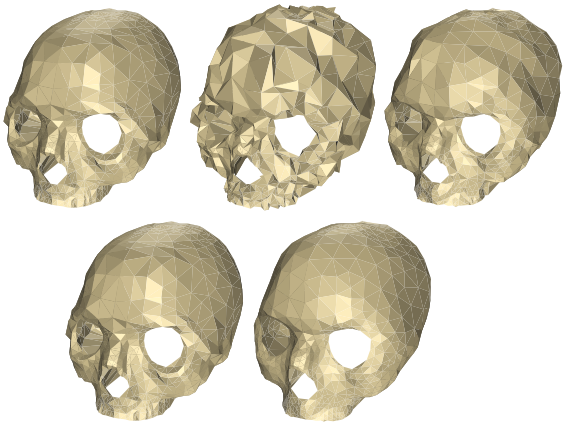


**Figure 2:** *From left to right: origin, noisy mesh, three iterations of our algorithm.*

One approach to surface smoothing is the Laplacian algorithm. Versions of Laplacian smoothing are e.g. known in image processing and finite-element-meshing.[1, 2, 3, 4] The basic idea in image processing is to replace the grey value of a pixel with an average of the grey values of pixels in some neighborhood. Similarly, the location of a vertex in a finite-element mesh is corrected by calculating a new location as the average of the locations of vertices in the neighborhood

on the mesh. This process of averaging can be applied iteratively, until the result is satisfiable.

Averaging causes a smoothing effect which is similar to that which we desire in our application of surface mesh smoothing. This observation can be transferred to surface meshes by calculating a new location of a vertex on a surface mesh by averaging the locations of the neighboring vertices. Figure 1 shows the result of an iterative application of this approach to a data set. For demonstration, the originally smooth vertices were disarranged by noise.

However, the example of figure 1 shows an undesirable effect of that approach. The essential difficulty is that the application of the Laplacian algorithm shrinks the mesh. Evidently the resulting smooth skull is significantly smaller than the original one.

In this contribution, we present a general approach, called HC-algorithm, which prevents the effect of shrinking, while preserving the effect of smoothing. The key idea is to push the vertices obtained in each step of iteration of the Laplacian algorithm back into the direction of the original vertices. Figure 2 shows the result of an application on the data set of figure 1. The smoothed mesh is quite similar to the original, also in size.

After fixing the required notation in section 2, in section 3 we introduce in some more detail the Laplacian algorithm for surface meshes, including several useful variants. Section 4 describes the HC-algorithm. In section 5, the behavior of convergence of the iteration of the original Laplacian algorithm and the HC-algorithm is given. Section 7 discusses the more general case that only a cloud of points not connected by a mesh has to be smoothed. The basic idea of our solution to that problem is to construct a surface mesh connecting the given points which, when smoothed, yields the desired smoothed point set. The interesting observation with this approach is that it is not necessary to construct a topologically correct mesh which makes the task much more easy than the rather complex surface reconstruction problem.

## 2. Formal conventions

In the following a *mesh* $M$ with vertex set $V = \{1,\dots,n\}$ is given by a tupel $(K,\mathbf{p})$, where $K \subseteq 2^V$ is a simplicial complex and $\mathbf{p} : V \to \mathbb{R}^3$ is a function, which maps every vertex $i$ to its position $\mathbf{p}_i$. In general different vertices $i \neq j$ can have the same position $\mathbf{p}_i = \mathbf{p}_j$. The "set" of points $\mathbf{p}$ will be represented in the following by the vertical *point vector* ${}^t(\mathbf{p}_1,\dots,\mathbf{p}_n)$. The set of vertices $V$ is split up into two disjoint sets of *fixed* vertices $V_{fix}$ and *movable* vertices $V_{var}$. The set of *adjacent* vertices of one vertex $i$ is denoted by $adj(i)$. Let $E := \{e \in K \mid |e| = 2\}$ be the set of *edges* and $F := \{f \in K \mid |f| = 3\}$ be the set of *faces* of $M$.

It is characteristically for *smoothing algorithms* in contrast to swapping- (see sect. 7) and subdivision algorithms (see sect. 6) that they modify the positions $\mathbf{p}_i, i \in V_{var}$ of the (moveable) vertices while keeping the topology $K$ of the mesh unchanged. We follow the convention, that $\mathbf{o}_i$ are the *original* given points, $\mathbf{q}_i$ the *current* points (before the application of the algorithm) and $\mathbf{p}_i$ the *new, modified* positions (after one iteration of the algorithm), e.g. a smoothing algorithm starts with vertex positions $\mathbf{o} =: \mathbf{q}$ and maps the current positions $\mathbf{q}$ onto $\mathbf{p}$ by one step.

Usually, the new position $\mathbf{p}_i$ of any vertex $i$ depends solely on the positions of its *adjacent* vertices $j \in adj(i)$. To avoid problematic cases, we state the convention that if $adj(i) = \emptyset$ then $i \in V_{fix}$.

## 3. Laplacian smoothing

In this section the original version and two modifications of the Laplacian algorithm will be introduced. They belong to the class of *smoothing algorithms* defined as above.

### 3.1. The original version

The Laplacian algorithm is quite simple: the position $\mathbf{p}_i$ of vertex $i$ is replaced with the average of the positions of adjacent vertices (figure 3). We have

$$\mathbf{p}_i := \begin{cases} \frac{1}{|adj(i)|} \sum_{j \in adj(i)} \mathbf{q}_j, & i \in V_{var}, \\ \mathbf{q}_i, & i \in V_{fix}. \end{cases}$$

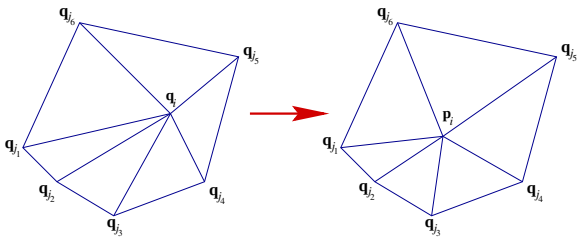



**Figure 3:** *The Laplacian algorithm.*

There are two techniques to calculate new positions $\mathbf{p}_i$. The first method is to modify *all* positions $\mathbf{q} \mapsto \mathbf{p}$ by one step. So every new position $\mathbf{p}_i$, $i = 1,\dots,n$, depends completely on the same set of positions, namely $\mathbf{q}$. This method is called the *simultaneous version*. The second variant is to update the new positions $\mathbf{p}_i$ immediately. This variant is called the *sequential version*. In this case a position $\mathbf{p}_i$ may not solely depend on the "set" of old positions $\mathbf{q}$ but can depend on a previously calculated new position $\mathbf{p}_j$, too. Hence the result of one smoothing pass through all vertices $i \in V_{var}$ will depend on the order how the vertices are considered.

The simultaneous version needs more storage space for all old positions $\mathbf{q}$ until the new are calculated completely. However, the results of this techniques are better. Surprisingly, among all other smoothing algorithms the Laplacian

algorithm solely has the property that the limits of the two techniques are the same if they exist (see 3.3 and 5).

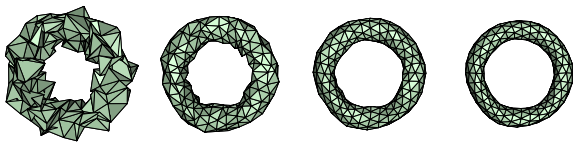


**Figure 4:** *Laplacian algorithm applied to a noisy torus.*

Figure 4 shows meshes with $V_{fix} = \emptyset$ after 1,2, and 3 iterations smoothed with the Laplacian algorithm. We can recognize a strong degree of shrinkage. If the number of iterations $k$ goes to $\infty$ the mesh will be shrinking towards a point.

### 3.2. Inclusion of the central point

It is noticeable that only the positions of the adjacent vertices and not the current position $\mathbf{q}_i$ is included in the calculation of the new position $\mathbf{p}_i$. This could be done by extending the Laplacian rule to

$$\mathbf{p}_i = \begin{cases} \alpha\mathbf{q}_i + \frac{1-\alpha}{|adj(i)|}\sum_{j\in adj(i)}\mathbf{q}_i, & i \in V_{var}, \\ \mathbf{q}_i, & i \in V_{fix}. \end{cases}$$

It turns out that this modification is not essential. We obtain neither a better smoothing quality nor better properties concerning the shrinking effect. This modification only delays the smoothing process in comparison with the original version.

### 3.3. Inclusion of the original point

In general, the Laplacian algorithm might not converge. For this, consider a mesh consisting of only two vertices $i, j$ joint by one edge $\{i, j\}$. The original simultaneous Laplacian algorithm lets the positions $\mathbf{p}_i \leftrightarrow \mathbf{p}_j$ exchange alternately. But in most cases the mesh converges towards one point.

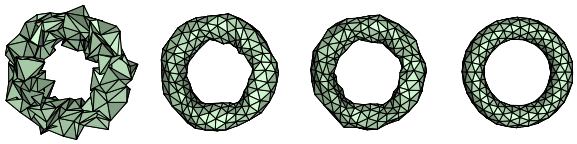


**Figure 5:** *Extended Laplacian algorithm with* $\alpha = 0.2$.

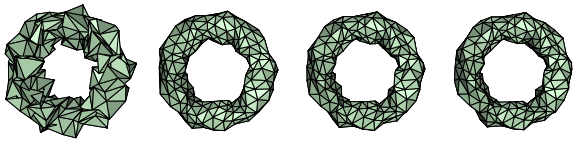


**Figure 6:** *Extended Laplacian algorithm with* $\alpha = 0.4$.

An idea to avoid these disadvantages and to force convergence against a non-trivial mesh is to include the *original* points $\mathbf{o}_i$ $\alpha$-weighted in the calculation:

$$\mathbf{p}_i := \begin{cases} \alpha\mathbf{o}_i + \frac{1-\alpha}{|adj(i)|}\sum_{j\in adj(i)}\mathbf{q}_j, & i \in V_{var}, \\ \mathbf{q}_i, & i \in V_{fix}. \end{cases}$$

Depending on the location of $\alpha$ near at 1 or near at 0, we can choose between a strong or a weak binding of $i$ to the original position $\mathbf{o}_i$. It will be shown in section 5 that non-trivial convergence is guaranteed if $\alpha > 0$. An interesting question is whether we can solve the deformation and shrinkage problem by this idea. Obviously, the last torus in figure 5 is quite better in comparison with the result produced by the original Laplacian algorithm showed in 4. However, it is not free from shrinkage. Concerning shrinkage we rather would like to obtain a mesh shown in figure 6. Unfortunately, this mesh is not smooth enough. All of our examples have shown that the results of this technique only form a bad compromise between smoothing quality and degree of deformation or shrinkage, respectively.

## 4. The HC-algorithm

The idea of the HC-algorithm (*HC* stands for *Humphrey's Classes* and has no deeper meaning) is to push the modified points $\mathbf{p}_i$ (produced by the Laplacian algorithm e.g.) back towards the previous points $\mathbf{q}_i$ and (or) the original points $\mathbf{o}_i$ by the average of the differences

$$\mathbf{b}_i := \mathbf{p}_i - (\alpha\mathbf{o}_i + (1-\alpha)\mathbf{q}_i), \text{ i.e. by}$$

$$\mathbf{d}_i := -\frac{1}{|adj(i)|}\sum_{j\in adj(i)}\mathbf{b}_j.$$

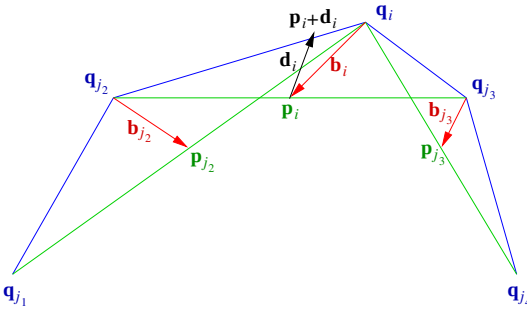



**Figure 7:** *The definition of* $\mathbf{d}_i$ *in the case of the HC-modification based on the Laplacian algorithm (for simplification with* $\alpha = 0$*, i.e. without influence of* $\mathbf{o}_i$*).*

It turns out (see sect. 5) that in this case (in contrast to sect. 3.2) the difference $\mathbf{b}_i$ at the center vertex $i$ must be included, weighted by a scalar $\beta \in [0, 1]$ so that

$$\mathbf{d}_i := \beta\mathbf{b}_i + \frac{1-\beta}{|adj(i)|}\sum_{j\in adj(i)}\mathbf{b}_j.$$

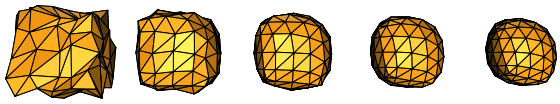


**Figure 8:** *From left to right: noisy mesh, four steps of Laplacian smoothing only. We notice a high degree of deformation and shrinkage.*

**Algorithmus 1** HC-algorithm (simultaneously)

$\mathbf{p} := \mathbf{o}$; // initialize points with original locations
**repeat**
  $\mathbf{q} := \mathbf{p}$;
  **for all** $i \in V_{var}$ **do**
    $n := |adj(i)|$;
    **if** $n \neq 0$ **then**
      $\mathbf{p}_i := \frac{1}{n}\sum_{j \in adj(i)} \mathbf{q}_j$; // Laplacian operation
    **end if**
    $\mathbf{b}_i := \mathbf{p}_i - (\alpha \mathbf{o}_i + (1-\alpha)\mathbf{q}_i)$;
  **end for**
  **for all** $i \in V_{var}$ **do**
    $n := |adj(i)|$;
    **if** $n \neq 0$ **then**
      $\mathbf{p}_i := \mathbf{p}_i - (\beta \mathbf{b}_i + \frac{1-\beta}{n}\sum_{j \in adj(i)} \mathbf{b}_j)$;
    **end if**
  **end for**
**until** <condition> // e.g. smooth enough

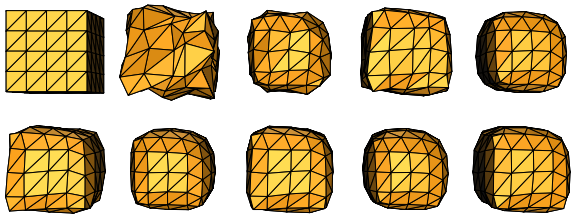
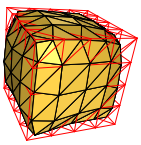


**Figure 9:** *From left to right: origin, noisy mesh, alternately a Laplacian operation and a HC-modification.*

Hence, there are two steps: the given smoothing algorithm, that maps $\mathbf{q}$ to $\mathbf{p}$, and the HC-modification, that maps $\mathbf{p}$ to $\mathbf{p}+\mathbf{d}$ as a kind of correction. It is remarkable that this calculation is similar to the Laplacian operation (from secti. 3.2) too, not related to the points $\mathbf{p}_i$ but to the differences $\mathbf{b}_i$.

Four iterations of the two steps are applied to a noisy cube in figure 9. We make out that the size of the cube de- and increases alternately. The last cube seems to be shrunk a little. The right figure shows that this is not the case. The reason for this optical illusion are the cut edges.



Obviously the cubes smoothed by the Laplacian algorithm only (figure 9) are shrinking much more. In both cases the simultaneous version of the algorithms has been performed.

## 5. Mathematical treatment

The goal of this section is to investigate the convergence behavior of the Laplacian- and of the HC-algorithm. In section 3.3 was mentioned that the version described there converges in every case. It became clear too that convergence only does not guarantee a good smoothing algorithm. However, it should be the minimum requirement we expect from an algorithm that works iteratively. While the convergence of the extended Laplacian algorithm described in 3.3 is quite evident, this is not the case for the HC-algorithm. We formulate the iteration step of the *simultaneous* Laplacian algorithm in matrix notation, since it is the more important case in practice. The treatment of the *sequential* version is more extensive. Both versions can be interpreted as two kinds of numerical algorithms for solving the same matrix equation $\mathbf{Ax} = \mathbf{b}$. Additionally, this implies that the two limits are equal as mentioned above. Further details on this are not considered here.

Let $M = (K, \mathbf{p})$ be a mesh with vertex set $V$ and set of edges $E$. Without loss of generality we require that $V_{var} = \{1, \ldots, m\}$ and $V_{fix} = \{m+1, \ldots, n\}$. So the points $\mathbf{p}_1, \ldots \mathbf{p}_m$ are the interesting moveable points. One step of the Laplacian algorithm can be represented in matrix notation:

$$\begin{pmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_m \end{pmatrix} = \mathbf{T} \begin{pmatrix} \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_m \end{pmatrix} + \begin{pmatrix} \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_m \end{pmatrix}.$$

It is easy to verify that the matrix $\mathbf{T} = (t_{ij})_{i,j=1,\ldots,m}$ is defined by

$$t_{ij} := \begin{cases} 1/|adj(i)| & \text{if } \{i,j\} \in E \\ 0 & \text{else} \end{cases},$$

and the vector $\mathbf{r} := {}^t(\mathbf{r}_1, \ldots, \mathbf{r}_m)$ by

$$\mathbf{r}_i := \frac{1}{|adj(i)|} \sum_{l=m+1}^{n} \begin{cases} \mathbf{p}_l & \text{if } \{i,l\} \in E \\ \mathbf{0} & \text{else} \end{cases}.$$

The iterations of the Laplacian algorithm provide a sequence of vectors ${}^t(\mathbf{p}_1, \ldots, \mathbf{p}_m)$ that will be denoted by $\mathbf{x}^{(k)}$ where $k$ is the index of the iteration number.

If we initialize $\mathbf{x}^{(0)} := {}^t(\mathbf{o}_1, \ldots, \mathbf{o}_m)$ the sequences of the points produced by the following equations

$$\mathbf{x}^{(k+1)} := \mathbf{T}\mathbf{x}^{(k)} + \mathbf{r} \quad (1)$$

$$\mathbf{x}^{(k+1)} := [\alpha\mathbf{E} + (1-\alpha)\mathbf{T}]\mathbf{x}^{(k)} + (1-\alpha)\mathbf{r} \quad (2)$$

$$\mathbf{x}^{(k+1)} := [(1-\alpha)\mathbf{T}]\mathbf{x}^{(k)} + \alpha\mathbf{x}^{(0)} + (1-\alpha)\mathbf{r} \quad (3)$$

represent the iterated point vectors of the three versions of the Laplacian algorithm described in sections 3.1, 3.2, and 3.3, respectively.

It follows from Banach's fixpoint theorem that those sequences converge if the matrix standing in front of the iteration variable $\mathbf{x}^{(k)}$ has only eigenvalues $\lambda$ with $|\lambda| < 1$. It can easily be seen that the sum $\sum_{j=1}^{m} t_{ij}$ of values of every matrix row is $\leq 1$. Equality happens if the corresponding vertex is joined with no fixed vertex. Since this is the general case we only can conclude $|\lambda| \leq 1$ for the eigenvalues $\lambda$ of $\mathbf{T}$. The same holds for the matrix $\alpha\mathbf{E} + (1-\alpha)\mathbf{T}$. But in the third case we can ensure convergence because $(1-\alpha)\mathbf{T}$ has eigenvalues $|\lambda| < 1$ if $\alpha > 0$. The following theorem summarizes these observations.

**Theorem 5.1** The sequence of points maintained by the three versions in sections 3.1, 3.2, 3.3 of the Laplacian algorithm are given by the three equations (1), (2), (3), resp. Let $\rho$ denote the radius of the spectrum. Convergence is ensured in all cases if $\rho(\mathbf{T}) < 1$. Since this does not hold in general, convergence can be guaranteed only in the third case if $\alpha > 0$.

A more precise analysis shows that convergence in the first two cases can be concluded if we only require that the mesh is connected and has at least one fixed point. However, in the usual case $V_{fix} = \emptyset$ for surface meshes not even this can be assumed.

Now, we will consider the HC-algorithm. Let $\mathbf{T}$ be the matrix as defined above. We can express the HC-algorithm in terms of $\mathbf{T}$, namely by

$$\begin{aligned}\mathbf{x}^{(k+1)} &= \mathbf{T}\mathbf{x}^{(k)} + \mathbf{r} - (\beta\mathbf{E} + (1-\beta)\mathbf{T}) \cdot \\ &\quad \left(\mathbf{T}\mathbf{x}^{(k)} + \mathbf{r} - \left(\alpha\mathbf{x}^{(0)} + (1-\alpha)\mathbf{x}^{(k)}\right)\right).\end{aligned}$$

**Theorem 5.2** The HC-algorithm converges if $\alpha > 0$ and $\beta > 0.5$.

**Proof** Transforming the describing equation of the HC-algorithms yields

$$\begin{aligned}\mathbf{x}^{(k+1)} &= \mathbf{T}\mathbf{x}^{(k)} + \mathbf{r} - (\beta\mathbf{E} + (1-\beta)\mathbf{T}) \cdot \\ &\quad \left(\mathbf{T}\mathbf{x}^{(k)} + \mathbf{r} - \left(\alpha\mathbf{x}^{(0)} + (1-\alpha)\mathbf{x}^{(k)}\right)\right) \\ &= \\ &\left(-(1-\beta)\mathbf{T}^2 + (2-\alpha)(1-\beta)\mathbf{T} + (1-\alpha)\beta\mathbf{E}\right)\mathbf{x}^{(k)} \\ &\quad + (\alpha\beta\mathbf{E} + \alpha(1-\beta)\mathbf{T})\mathbf{x}^{(0)} + (1-\beta)(\mathbf{E}-\mathbf{T})\mathbf{r}.\end{aligned}$$

Responsible for the convergence is the matrix in front of $\mathbf{x}^{(k)}$, i.e.

$$\mathbf{H} := -(1-\beta)\mathbf{T}^2 + (2-\alpha)(1-\beta)\mathbf{T} + (1-\alpha)\beta\mathbf{E}.$$

The eigenvalues of $\mathbf{H}$ are obtained by applying the polynomial

$$p(X) := -(1-\beta)X^2 + (2-\alpha)(1-\beta)X + (1-\alpha)\beta$$

to the eigenvalues $\lambda$ of $\mathbf{T}$ which are known to satisfy $|\lambda| \leq 1$. Calculating the amount of $p(X)$ for $X \in \mathbb{C}, |X| \leq 1$ is not easy. Fortunately, all eigenvalues of $\mathbf{T}$ are real. To see this, we consider

$$\mathbf{D} := \begin{pmatrix} |adj(1)| & 0 & \dots & 0 \\ 0 & |adj(2)| & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & |adj(m)| \end{pmatrix}$$

and notice that $\mathbf{DT}$ is symmetric. Matrices remain symmetric if they are multiplied from the left and the right with the same diagonal matrix. Now it is $|adj(i)| > 0$ for all $i \in \{1,\dots,m\} = V_{var}$ because of the convention that $adj(i) = \emptyset$ implies $i \in V_{fix}$. Therefore, it exists an inverse element and a square root. Hence, $\mathbf{D}^{-1/2}$ exists (not unambiguously) and is a diagonal matrix. Multiplying $\mathbf{DT}$ from the left and the right with $\mathbf{D}^{-1/2}$ results in $\mathbf{D}^{1/2}\mathbf{T}\mathbf{D}^{-1/2}$ which is symmetric and similar to $\mathbf{T}$. It follows that all eigenvalues of $\mathbf{T}$ are in $\mathbb{R}$.

This result makes it much more easy to show that $|p(X)| < 1$ if $X \leq 1$, $\alpha > 0$, and $\beta > 0.5$. This technical task will not be shown here in detail because of the lack of space. A proof can be found in [5]. □

The conclusion $\lambda \in \mathbb{R}$ is important since the upper bound 1 for $p(X)$ will be exceeded if $X \in \mathbb{C}$ and $|X| \leq 1$. For $\beta = 0.6$ and $\alpha = 0.1$ is $p(X) = -0.4X^2 + 0.76X + 0.54$ and so $|p(i)| = |0.94 + 0.76i| \approx 1.2$.

## 6. Discussion

We could see that the HC-algorithm yields quite satisfying results. However, a smoothing algorithm with HC-modification is not completely free from shrinkage, too. So further modifications are possible:

Smoothing algorithms map $\mathbf{q}_i$ to $\mathbf{p}_i$ for each vertex $i \in V$, according to our definition. Immediately after a local Laplacian smoothing operation at vertex $i$ it is $\mu(i) := \mathbf{p}_i - \sum_{j \in adj(i)} \mathbf{q}_j = \mathbf{0}$. Clearly, after a complete treatment of all vertices $\mu(i) = \mathbf{0}$ cannot be expected. Rather, $\mu(i)$ is a *discrete normal vector* and can be regarded as a measure of the *curvature* at vertex $i$ (it even is a discrete approximation of the Laplacian operator $\Delta$, which is for its part an approximation of the mean curvature.[6]) A mesh is *planar* if $\mu(i)$ tends towards the zero vector for all $i$. On the other hand, a mesh is *smooth* if neighboring normal vectors $\mu(j), j \in adj(i)$, are similar. This is an important difference. Hence, for mesh smoothing, the $\mu(i)$ need not necessarily tend towards the zero vector like for the Laplacian algorithm. Rather, it is appropriate to filter the high frequency parts from $\mu$. For this purpose, a spectral decomposition or Fourier analysis, respectively, of $\mu$ would be necessary to get a filtered function $\mu'$.

The HC-algorithm is a step into this direction. $\alpha = 0$ and $\beta = 0.5$ imply $\mu' = 0.5\mu \circ \mu$, i.e. the HC-algorithm filters the differences of second order. Probably, there can be found a

better choice for $\mu'$ which perhaps can avoid deformation and shrinkage completely.

The HC-modification can also be applied in combination with subdivision algorithms like the Doo-Sabin- or Catmull-Clark-algorithm.[7, 8] Most of the subdivision algorithms contain rules that calculate averages of adjacent points. Hence, they have a component of shrinkage, too. It is possible to expand such rules by the HC-modification in the same way as we did with the Laplacian algorithm.

## 7. Smoothing of noisy sample points

Mesh smoothing techniques may also be used in order to smooth an original cloud of points without surface information. The immediate idea is to reconstruct a surface mesh from the cloud of points. Smoothing of the resulting mesh yields vertices in new locations which are taken as the smoothed cloud. The difficulty with this approach is that surface reconstruction in general is not an easy problem, cf. [9] However, it will turn out in the following that, for the purpose of smoothing, it is not necessary to reconstruct a perfect mesh. The only requirement is that the neighborhoods on the reconstructed mesh are about the same as they are on the underlying real surface. Whenever a reconstructed mesh through the noisy point set is not available, the RFS-mesh (replacement for surface) can be taken in order to eliminate the noise. This RFS-mesh contains nearly the same number of faces as a valid surface mesh, but about 50 percent more edges. Faces can cross and overlap in a RFS-mesh.

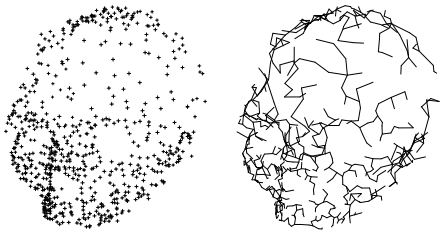


**Figure 10:** *Left: the noisy point set. Right: the EMST.*

The RFS-mesh is based on the SDG (surface description graph) of Mencl et al. [10, 11]. The kernel of the SDG, in turn, is the EMST of the given point cloud. The EMST (Euclidean minimum spanning tree) is a tree connecting all points of $P$ with line segments so that the sum of its edge lengths is minimized (see figure 10, right). The EMST (as initial surface description graph) is then extended to the surface description graph (SDG) of higher order by adding edges at appropriate regions (cf. figure 11, left for an example of the SDG). This graph consists of significantly more edges than the EMST and can serve as a base for getting neighborhood information when generating the RFS-mesh.

Independently from the degree of noise, the SDG-mesh is always defined. It contains less edges than a surface mesh but "good" edges, i.e. it contains edges, that "lie" close to the assumed (but actually unknown) surface that is described by the noisy point set.

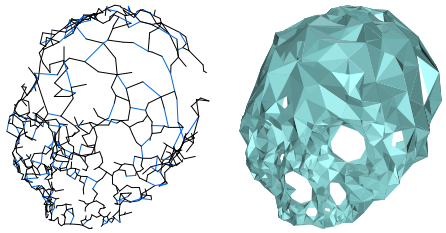


**Figure 11:** *Left: the SDG. Right: the RFS-mesh.*

In the following we outline the computation of the RFS-mesh out of the SDG.

If required, the SDG can be filled with additional nearest-neighbor-edges. To respect variations concerning the density of the noisy points we use an adaptive number of nearest neighbors (estimated by using the SDG-mesh as reference) for different vertices.

Around each vertex a corona of faces will be created on base of incident edges like a spanning umbrella. By this, for each face one new edge will be created, if it does not already exist. These edges are important for the connectivity of the new mesh.

Faces that do not fulfill a given mesh quality measure will be deleted. We use a quality measure introduced by Bank & Smith.[12]

To make edge swapping (between neighboring triangles) applicable, incident faces to an edge are deleted by comparing their enclosed dihedral angles until there remain at most two at an edge. If there are more than two faces at an edge we keep that pair of edges that form the maximum dihedral angle.

All edges that do not have at least one incident face are deleted.

The resulting mesh is called the RFS-mesh and can be taken as input for the HC-algorithm in order to eliminate the noise out of the point set. In figure 11 (right) an example for an RFS-mesh is depicted. If we would take the RFS-mesh of this figure as input for the smoothing algorithm, a point set of similar quality (with respect to the noise reduction factor) to the one of figure 2 in section 1 will be obtained.

During the smoothing process it is advantageous to apply edge swapping and one step of the HC-algorithm alternately. Edge swapping between two adjacent faces is used to increase the sum of dihedral angles at all five edges of the two faces in the triangular mesh. Freitag et al.[13] found out that the combination of edge swapping and smoothing yields better

results than one of the two techniques only. This was examined in the field of finite element methods, i.e. for meshes in $\mathbb{R}^2$ and tetrahedron meshes in $\mathbb{R}^3$. We found out that this is also valid for surface meshes. The reason is, that smoothing techniques do not change the connectivity formed by the edges and faces (see sect. 2). All examples show that we can improve the degree of smoothness if we permit edges to adjust themselves in a new manner. Note, that edge swapping does not change any vertex position and hence the mesh does not shrink thereby.

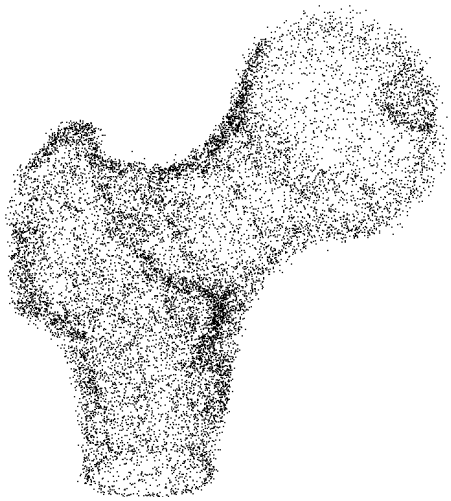


**Figure 12:** *Noisy set of points.*

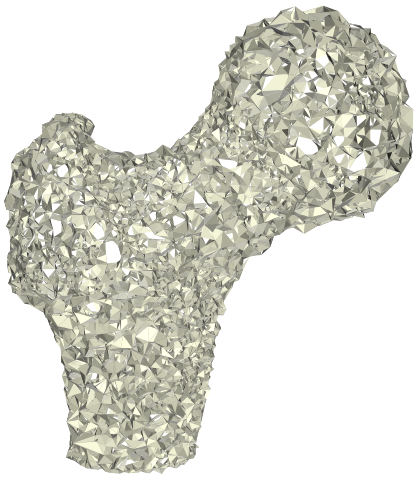


**Figure 13:** *Temporary RFS-mesh.*

An example for the smoothing process is shown with a large point set in figures 12–16. The first figure 12 shows the noisy input point set and figure 13 its corresponding RFS-mesh. In figure 14 the RFS-mesh is smoothed. The computation time for the whole smoothing process was about 10 seconds. It can be seen that the RFS-mesh is not a valid surface mesh. The smoothed point set of the RFS-mesh is depicted in figure 15. This smooth point set was taken as input for the surface reconstruction algorithm of Mencl et al. [10, 11] and the result is shown in figure 16.

**Figure 14:** *RFS-mesh smoothed by the HC-algorithm in combination with edge swapping.*

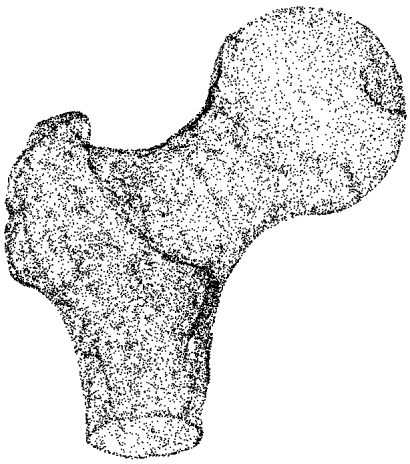


**Figure 15:** *Noise reduced set of points maintained by the smoothed RFS-mesh.*

## 8. Conclusions

A technique for the elimination of noise in sets of sample points and surface meshes has been developed and applied

**Figure 16:** *Reconstruction based on the noise reduced set of points.*

successfully. The proposed technique presents certain advantages:

- it can produce meshes with the same smoothing degree as the Laplacian algorithm,
- in contrast to the Laplacian algorithm it preserves shape and size of the sampled object far better,
- it preserves the point density of the mesh,
- the algorithm works very fast and requires only calculations of simple vector arithmetic,
- it is easy to implement.

## References


1. H.Müller and S. Amabrowski, *Geometrisches Modellieren*. Mannheim: BI Wissenschaftsverlag, (1991).
2. L. Freitag, "On combining laplacian and optimization - based mesh smoothing techniques", in *Proceedings of the 1997 Joint Summer Meeting of American Society of Mechanical Engineers (ASME) American Society of Civil Engineers (ASCE) and Society of Enginieers Science (SES)*, pp. 37–44, (1997).
3. N. Amenta, M. Bern, and D. Eppstein, "Optimal point placement for mesh smoothing", in *8th ACM-SIAM Symp. on Discrete Algorithms*, (New Orleans), (1997).
4. S. Canann, M. Stephenson, and T. Blacker, "Optismoothing: An optimization-driven approach to mesh smoothing", *Finite Elements in Analysis and Design*, **13**, pp. 185–190 (1993).
5. J. Vollmer, "Eliminierung von lokalem und globalem Rauschen in ungeordneten Punktmengen auf dreidimensionalen Oberflächen", Master's thesis, University of Dortmund, Germany, (November 1998). http://www.humphrey.de/download/vollmer98.ps.gz.
6. L. Kobbelt, *Iterative Erzeugung glatter Interpolanten*. PhD thesis, Universität Karlsruhe, (1994).
7. D. Doo and M. Sabin, "Behavior of recursive division surfaces near extraordinary points", *Computer Aided Design*, **10**(6), pp. 356 – 360 (1978).
8. E. Catmull and J. Clark, "Recursively generated b-spline surfaces on arbitrary topological meshes", *Computer Aided Design*, **10**(6), pp. 350 – 355 (1978).
9. R. Mencl and H. Müller, "Interpolation and approximation of surfaces from three–dimensional scattered data points", pp. 51–67 (1998). EUROGRAPHICS '98, STAR - State of the Art Reports, also available as Research Report No. 661, 1997, Fachbereich Informatik, Lehrstuhl VII, University of Dortmund, Germany.
10. R. Mencl, "A graph–based approach to surface reconstruction", *Computer Graphics forum*, **14**(3), pp. 445–456 (1995). Proceedings of EUROGRAPHICS '95, Maastricht, The Netherlands, August 28 - September 1, 1995.
11. R. Mencl and H. Müller, "Graph–based surface reconstruction using structures in scattered point sets", in *Proceedings of CGI '98 (Computer Graphics International), Hannover, Germany, June 22th–26th 1998.*, pp. 298–311, (1998). similar version also available as Research Report No. 661, 1997, Fachbereich Informatik, Lehrstuhl VII, University of Dortmund, Germany.
12. R. Bank and R. Smith, "Mesh smoothing using a posterior error estimation", *SIAM Journal on Numerical Analysis*, (1997).
13. L. Freitag and C. Ollovier-Gooch, "Tetrahedral mesh improvement using face swapping and smoothing", *International Journal for Numerical Methods in Engineering*, **Vol. 40**, (1997).