GEOMETRIC SIGNAL PROCESSING ON LARGE POLYGONAL MESHES

COURSE NOTES FOR SIGGRAPH'2001

Los Angeles, California

SUNDAY, AUGUST 12, 2001

HALF DAY, 1:30 - 5 PM

Course Speakers:

LEIF KOBBELT
Rheinisch-Westfälische Technische Hochschule Aache

GABRIEL TAUBIN
IBM T.J. Watson Research Center
California Institute of Technology

ABSTRACT

Very large polyhedral models, which are used in more and more graphics applications today, are routinely generated by a variety of methods such as surface reconstruction algorithms from 3D scanned data, isosurface construction algorithms from volumetric data, and photogrametric methods from aerial photography. The course will provide an overview of several closely related methods designed to smooth, denoise, edit, compress, transmit, and animate very large polygonal models, based on signal processing techniques, constrained energy minimization, and the solution of diffusion differential equations.

SPEAKERS

Leif Kobbelt
RWTH-Aachen
Lehrstuhl für Informatik VIII
52056 Aachen
kobbelt@cs.rwth-aachen.de
http://www.informatik.rwth-aachen.de/I8/

Dr. Leif Kobbelt is Professor of Computer Science, and head of the Computer Graphics group at the University of Aachen, Germany. Previously, he was a senior researcher at the Max-Planck-Institute for computer sciences in Saarbrücken, Germany. His major research interests include Multiresolution and free-form modeling, as well as the efficient handling of large polygonal mesh data sets. He received his habilitation degree from the University of Erlangen, Germany where he worked from 1996 to 1999. In 1995/96 he spent one post-doc year at the University of Wisconsin, Madison. He received his master's (1992) and Ph.D. (1994) degrees from the University of Karlsruhe, Germany. During the last 8 years he did research in various fields of computer graphics and CAGD, and published 50 journal and reviewed conference papers.

Gabriel Taubin
IBM T.J. Watson Research Center
P.O. Box 704,
Yorktown Heights, NY 10598
Email: taubin@us.ibm.com

Gabriel Taubin is a Research Staff Member and former manager of the Visual and Geometric Computing group at the IBM Thomas J. Watson Research Center. He joined IBM Research in 1990. He is currently on sabbatical at the California Institute of Technology, in Pasadena, California, as Visiting Professor of Electrical Engineering. He holds a "Licenciado en Ciencias Matemáticas" degree (MSc. in Pure Mathematics) from the University of Buenos Aires, Argentina, and a Ph.D. in Electrical Engineering from Brown University, Providence, Rhode Island. Gabriel has published more than 35 book chapters, refereed journal or conference papers, 25 other conference papers and technical reports, and 25 patents (20 issued). He has presented the Eurographics'2000 State of the Art Report on Geometric Signal Processing on Polygonal Meshes, the Eurographics'99 State of the Art Report on 3D Geometry Compression and Progressive Transmission, and organized and taught four courses on 3D Geometry Compression at Siggraph'98, Siggraph'99, Siggraph'2000, and ACM Solid Modeling 2001. Gabriel was granted an IBM Research 1998 Computer Science Best Paper Award for his paper Geometry Compression through Topological Surgery. The IEEE Board of Directors elected Gabriel as an IEEE Fellow, effective Jan 1 2001, for his contributions to the development of three-dimensional geometry compression technology and multimedia standards. As manager of the Visual and Geometric Computing Group at IBM Research he lead three main projects. The MPEG-4 3D Mesh Coding project, where his 3D geometry compression technology was adopted

by the MPEG-4 standard; the HotMedia project, where he transfered his 3D geometry compression technology to the award winning (PC Expo'2000 Best of Show Sofware) IBM HotMedia product. And the Pieta project, where Michaelangelo's Florentine Pieta was 3D scanned and a 3D model was reconstructed to support art historian Jack Wasserman in his comprehensive study of the statue.

Course Syllabus and Timeline

1:30 INTRODUCTION (5")

1:35 TAUBIN (85")

- Representation of polygonal models
- Operations on large polygonal meshes
- Laplacian smoothing
- The shrinkage problem
- Fourier analysis on meshes
- Smoothing by partial Fourier expansion
- Smoothing as low-pass filtering
- Taubin smoothing
- FIR/IIR filter design
- Implicit Fairing / Multi-resolution modeling
- Smoothing with constraints
- Preventing tangential drift by curvature flow
- Applications to 3D geometry compression
- Optimal mesh sampling rate conversion
- Adaptive curvature-based resampling
- Filtering of normal and tensor fields
- Non-linear filtering / anisotropic diffusion

3:00 BREAK (15")

3:15 KOBBELT (85")

- Multiresolution representations
- Coarse-to-fine (refinement, remeshing)
- Fine-to-coarse (simplification)
- Detail encoding
- Fairing by constrained energy minimization
- General set-up
- Multi-level smoothing
- Fairing by solving PDEs
- Linear PDEs
- Non-linear PDEs
- Practical aspects
- Multiresolution editing
- Static connectivity
- Dynamic connectivity

4:40 QUESTIONS AND ANSWERS (15")

5:00 ADJURN

Reprints

A Signal Processing Approach to Fair Surface Design
by G. Taubin,
Siggraph'95

Optimal Surface Smoothing as Filter Design,
by G. Taubin, T. Zhang, and G. Golub,
IBM Technical Report RC-20404, March 1996; and
Fourth European Conference on Computer Vision (ECCV'96).

Interactive Multiresolution Mesh Editing
by D. Zorin, P. Schroder, and W. Sweldens
Siggraph'97

Interactive Multi-Resolution Modeling on Arbitrary Meshes,
by L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel,
Siggraph'98.

Multiresolution Hierarchies on Unstructured Triangle Meshes
by L. Kobbelt, J. Vorsatz, and H.-P. Seidel,
Computational Geometry Theory and Applications, special issue
on multi-resolution modeling and 3D geometry compression.

Multiresolution Signal Processing for Meshes,
by I. Guskov, W. Sweldens, and P. Schroder,
Siggraph'99.

Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow
by M. Desbrun, M. Meyer, P. Schroder, and A.H. Barr,
Siggraph'99.

3D Mesh Geometry Filtering Algorithms for Progressive Transmission Schemes
by R. Balan and G. Taubin, CAD Special Issue on Multiresolution
Geometric Models, 1999.

Geometric Signal Processing on Polygonal Meshes
by G. Taubin
Eurographics 2000 State of The Art Report (STAR)

Geometric Fairing and Variational Subdivision for Freeform Surface Design
by R. Schneider, and L. Kobbelt,
Computer Aided Geometric Design 2001

Geometric Signal Processing on Large Polygonal Meshes

Leif Kobbelt* Gabriel Taubin†

Since hierarchical methods in geometric modelling have emerged as a key tool to handle highly complex 3D data sets, the term *multiresolution* has been adapted from signal processing theory to emphasize the notion of decomposing a given shape into geometric frequency bands just like arbitrary (scalar or multi-modal) signals can be expressed by a superposition of different frequencies. The descriptive and functional power of these techniques is largely due to the intuitive correlation between high frequencies and fine detail on one side and low frequencies and global shape features on the other side.

Naturally, the operators to analyze or synthesize geometric shapes and their multi-resolution spectra, are very similar to the digital filters that are used in standard signal processing applications. The main difference is that the graph- or surface topology of a polygonal mesh is more complicated than the temporal or spatial domains on which conventional signals are usually defined (typically $\mathbb{R}^d$). Hence the major difficulty in applying signal processing techniques to geometric shapes is their generalization with respect to the domain topology.

The approach taken by Taubin [18] is to generalize the notion of *frequency* to triangle meshes by considering the vertex positions of a given mesh as a set of point samples from the underlying shape. The resulting piecewise linear surface is then an approximative reconstruction of the continuous shape from those samples.

In order to obtain a definition of *frequency* or *wavelength* which is independent from the specific geometric shape, we can measure the distance between mesh vertices by their topological distance, i.e., by the number of edges we have to follow to describe a path from vertex A to vertex B. For this definition, the highest frequency signal (i.e., the shortest wavelength) is achieved if the geometric location of directly neighboring vertices varies strongly and lower frequencies are characterized by vertex positions such that local minima and maxima are several edges apart.

The general task of analyzing the spectrum of a given signal is a global problem and hence computationally involved. For the efficient processing of large triangle meshes we have to find a local approximations – ideally leading to filter algorithms with linear complexity.

Consider, e.g., a triangle mesh which consists of one center vertex $\mathbf{p}$ and its surrounding neighbors $\mathbf{q}_0, \ldots, \mathbf{q}_{n-1}$. Then the smoothest possible geometric configuration (in terms of topological wavelength) is achieved if the center vertex lies at the center of gravity of its neighbors, i.e.,

$$\mathbf{p} \;=\; \frac{1}{n} \sum_{i=0}^{n-1} \mathbf{q}_i .$$

For all other configuration, the amount of higher frequency components can be measured by the difference vector

$$\mathcal{U}(\mathbf{p}) \;=\; \frac{1}{n} \sum_{i=0}^{n-1} \mathbf{q}_i \;-\; \mathbf{p}. \qquad (1)$$

An effective low-pass filter applied to the considered mesh will shift the position of the center vertex such that the length of the vector $\mathcal{U}(\mathbf{p})$ is reduced.

In a larger mesh each vertex can be understood as the center vertex to its directly adjacent neighbors. Consequently (1) gives a local estimate of the highest frequency at that location on the surface. Again, a low-pass can be implemented which moves each vertex towards the center of gravity of its neighbors and thus attenuates the highest frequency components.

After applying such a low-pass filter to a given mesh, the piecewise linear surface will become less rough. Hence to local geometry (= the local signal) could be reconstructed from fewer samples without losing significant geometric information. This is very similar to standard results from signal processing which state that the necessary sampling rate for correct reconstruction of a given signal depends on the highest frequency component.

In the context of multiresolution techniques a decomposition of a geometric data set M_0 is usually obtained by first applying a low-pass filter operation L which preserves only the low frequency components. Hence the difference of the vertex positions in the original mesh and the filtered mesh $M_0' = L(M_0)$ contains all the high frequency information. Since M_0' is *oversampled* in the sense that fewer samples would be sufficient for reliable reconstruction, we can apply a sub-sampling operator such as mesh decimation which yields a coarser mesh M_1.

Applying (1) to this mesh, again, measures the high frequency components but since the samples are distributed more sparsely, those frequencies belong to a different frequency band. This band can be isolated by computing the difference between the vertex positions in M_1 and a filtered version $M_1' = L(M_1)$.

By repeating this procedure of alternating low-pass filtering and subsampling, we generate a sequence of coarser and coarser meshes M_i with the total shape information decomposed into different frequency bands represented by the differences $M_i - M_i'$.

The basic ingredient for the above algorithm is a linear low-pass filter which is derived from the local "noise-estimator" (1). From a geometric point of view there are several interpretations for this vector $\mathcal{U}(\mathbf{p})$. Usually it is considered as some kind of discrete Laplace vector or Mean Curvature vector. An obvious generalization of (1) is

$$\mathcal{U}(\mathbf{p}) \;=\; \frac{1}{n} \sum_{i=0}^{n-1} \alpha_i (\mathbf{q}_i \;-\; \mathbf{p}) \qquad (2)$$

where the coefficients α_i can be used to adapt the operator to the local mesh distortion, i.e., to the varying edge lengths or to the angles between edges and normals. In [4] a number of possible choices is presented. From this perspective the low-pass filter operation turns out to be a diffusion operator that moves the vertices (= particles) along the direction given by (2).

A completely different interpretation emerges from the observation that a solution to the equation $\mathcal{U}(\mathbf{p}) = 0$ is in fact a discrete

*RWTH-Aachen Lehrstuhl für Informatik VIII 52056 Aachen kobbelt@cs.rwth-aachen.de

†California Institute of Technology, Department of Electrical Engineering, MS-136-93, Pasadena, CA 91125, taubin@caltech.edu. On Sabbatical from IBM T.J. Watson Research Center, Yorktown Heights, NY 10598 taubin@us.ibm.com

approximation to a continuous surface f satisfying the partial differential equation

$$\triangle f = 0. \tag{3}$$

Hence the application of the low-pass filter can be considered as one step in an iterative algorithm to solve the linear system which characterizes the solution of the PDE (3). Moreover, the PDE (3) is the Euler-Lagrange equation to the membrane optimization problem that minimizes the surface area while respecting prescribed boundary conditions.

Analoguously, if we control the low-pass filtering by higher order Laplace vectors

$$\mathcal{U}^2(\mathbf{p}) \;=\; \frac{1}{n}\sum_{i=0}^{n-1} \alpha_i(\mathcal{U}(\mathbf{q}_i) \;-\; \mathcal{U}(\mathbf{p}))$$

we iteratively solve the fourth order PDE

$$\triangle^2 f = 0$$

whose solutions minimizes the thin plate energy [9].

It turns out that the alternative interpretation of the low-pass filter as an iterative PDE solver naturally enables the inclusion of interpolation constraints to the multiresolution decomposition. For example, all meshes M_i can be forced to interpolate the same boundary curve which is useful when locally modifying the mesh model [9].

Geometrically more sophisticated generalizations of the curvature estimator (2) can be derived, if we allow the coefficients α_i to be non-linear functions of the local geometry and that they are updated in every step of the iteration [15]. By this we can generate discrete approximations to continuous solutions of non-linear PDEs whose shape is almost completely independent from the actual mesh connectivity.

References

[1] R. Balan and G. Taubin. 3d mesh geometry filtering algorithms for progressive transmission schemes. *Computer Aided Design*, 2000. Special issue on multiresolution geometric models.

[2] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 1999.

[3] U. Clarenz, U. Diewald, and M. Rumpf. Anisotropic geometric diffusion in surface processing. In *IEEE Visualization 2000, Conference Proceedings*, October 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 Conference Proceedings*, pages 317–324, August 1999.

[5] K. Fujiwara. Eigenvalues of laplacians on a closed riemannian manifold and its nets. *Proceedings of the AMS*, 123:2585–2594, 1995.

[6] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *Siggraph'99 Conference Proceedings*, pages 325–334, August 1999.

[7] S. Karbacher and G. Häusler. A new approach for modeling and smoothing of scattered 3d data. In *Three-Dimensional Image Capture and Applications, Proceedings of SPIE*, volume 3313, pages 168–177, 1998.

[8] Z. Karni and C Gotsman. Spectral compression of mesh geometry. In *Siggraph'2000 Conference Proceedings*, pages 279–286, 2000.

[9] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Siggraph'98 Conference Proceedings*, pages 105–114, July 1998.

[10] L. Kobbelt, J. Vorsatz, and H.-P. Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry Theory and Applications*, pages 5–24, 1999. special issue on multi-resolution modeling and 3D geometry compression.

[11] S. Kuriyama and K. Tachibana. Polyhedral surface modeling with a diffusion system. In *Eurographics'97 Conference Proceedings*, pages C39–C46, 1997.

[12] A.W.F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. In *Siggraph'98 Conference Proceedings*, pages 95–104, July 1998.

[13] Y. Ohtake, A.G. Belyaev, and Bogaevski I.A. Polyhedral surface smoothing with simultaneous mesh regularization. In *Geometric Modeling and Processing 2000, Conference Proceedings*, April 2000.

[14] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, July 1990.

[15] R. Schneider and L. Kobbelt. Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design*, 2001. (to appear).

[16] G. Strang. The discrete cosine transform. *SIAM Review*, 41(1):135–147, 1999.

[17] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings, International Conference on Computer Vision (ICCV)*, pages 902–907, 1995.

[18] G. Taubin. A signal processing approach to fair surface design. In *Siggraph'95 Conference Proceedings*, pages 351–358, August 1995.

[19] G. Taubin. Geometric signal processing on polygonal meshes. In *Eurographics 2000 State of The Art Reports (STAR)*, September 2000.

[20] G. Taubin, A. Guéziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *Siggraph'98 Conference Proceedings*, pages 123–132, July 1998.

[21] G. Taubin, T. Zhang, and G. Golub. Optimal surface smoothing as filter design. In *Fourth European Conference on Computer Vision (ECCV'96)*, 1996. Also as IBM Technical Report RC-20404, March 1996.

[22] H. Ugail, M.I.G. Bloor, and M.J. Wilson. Techniques for interactive design using the pde method. *ACM Transactions on Graphics*, 18(2):195–212, April 1999.

[23] A. Yamada, T. Furuhata, K. Shimada, and K. Hou. A discrete spring model for generating fair curves and surfaces. In *Proceedings of the Seventh Pacific Conference on Computer Graphics and Applications*, pages 270–279, 1998.

[24] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Siggraph'97 Conference Proceedings*, pages 259–268, August 1997.

Geometric Signal Processing on Large Polygonal Meshes
COURSE NOTES FOR SIGGRAPH'2001
LOS ANGELES, CALIFORNIA
AUGUST 12, 2001

Gabriel Taubin

IBM T.J. Watson Research Center
California Institute of Technology

Geometric Signal Processing on Polygonal Meshes

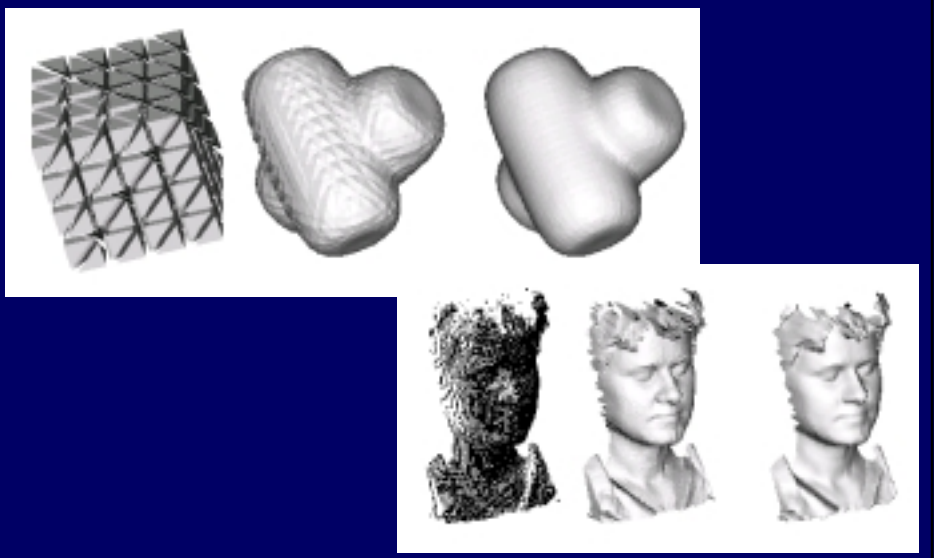
Gabriel Taubin

IBM T.J.Watson Research Center
http://www.research.ibm.com/people/t/taubin
California Institute of Technology
http://mesh.caltech.edu/taubin

Large dense polygonal meshes

- Are becoming standard representation for surface data
  - 3D Scanning (Reverse engineering, Art)
  - Isosurfaces (Scientific Visualization, Medical)
  - Subdivision Surfaces (Modeling, Animation)
- But have too many degrees of freedom (vertices)
- How to ?
  - Smooth / De-noise
  - Edit / Deform / Constrain / Animate
  - Represent / Compress / Transmit
- BUT FAST !

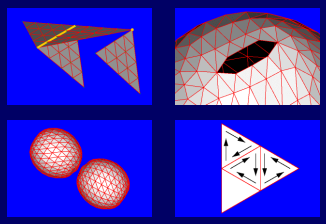
Digital Signal Processing on Meshes



ORIGINAL LAPLACIAN TAUBIN

Polygonal Meshes

- Components
  - Connectivity (Vertices, Edges, Faces)
  - Geometry (Vertex Coordinates)
  - Properties (Normals, Colors, Texture Coordinates)
- Connectivity (combinatorial algorithms)
  - Boundary / Regular / Singular Edges and Vertices
  - Connected Components
  - Manifold / Non-manifold
  - Orientable / Oriented
  - Topology



8/12/2001 Taubin / Siggraph 2001 Course 17 5

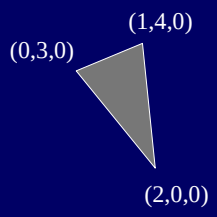
3D Representations

- Surfaces
  - Polygonal meshes
    - Disconnected triangles (STL file format)
    - IndexedFaceSet (VRML file format)
    - Half-Edge data structure (manifold meshes)
  - Boundaries of solid objects
- Volumes (solid objects)
  - Implicit surfaces
    - Defined by inside-outside function
  - Iso-surfaces : convertion to polygonal mesh

8/12/2001 Taubin / Siggraph 2001 Course 17 6

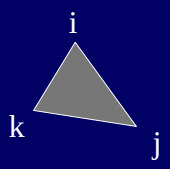
Disconnected Triangles

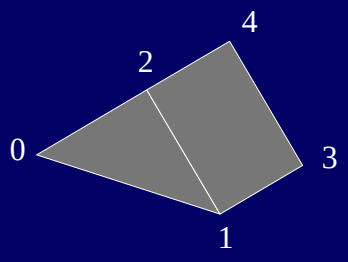
- Triangle Mesh with T triangles
  - Each triangle specified by 3 vectors = 9T floats
  - No connectivity information
  - STL file format used for Rapid Prototyping



IndexedFaceSet

- Array of vertex coordinates
- Each 3D vertex has an associated vertex index in {0,…,V-1}
- A triangle is defined by three vertex indices (i,j,k)
- A polygonal face without holes is defined by more indices
- coordIndex [0,1,2,-1,2,1,3,4,-1]
- VRML'97 file format

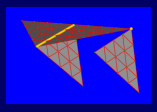




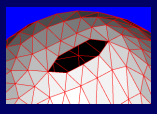
Polygonal Mesh Components

- Connectivity
  - coordIndex (faces)
- Geometry
  - coord (vertex coordinates)
- Properties
  - color/colorIndex/colorPerVertex
  - normal/normalIndex/normalPerVertex
  - texCoord/texCoordIndex

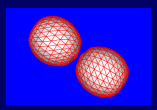
Connectivity / Classification of Elements



- Edges
  - Boundary (1 incident face)
  - Regular (2 incident faces)
  - Singular (3 or more incident faces)



- Vertices
  - Regular / Singular
- Connected components
  - Connected Components of Dual Graph

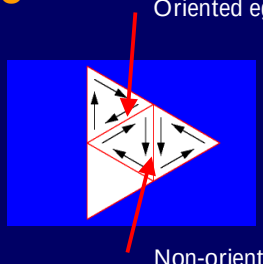


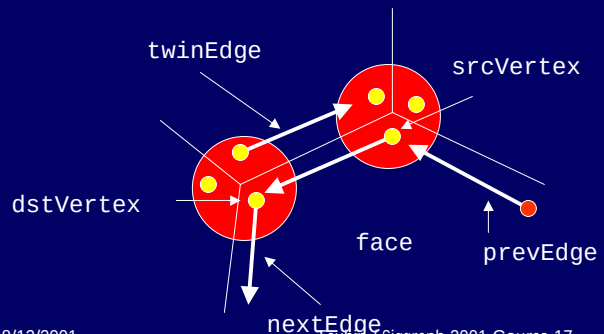
Manifold Meshes

- No singular edges
  - Boundary
    - Edge with 1 incident face
  - Regular
    - Edge with 2 incident faces
- No singular vertices
  - Boundary
    - dual graph of set of incident faces form a path
  - Regular
    - dual graph of set of incident faces form a cycle
- Data Structure to represent and operate on ?

Doubly-linked data structure

- Planar subdivisions
- Orientation
- Vertices / Faces / Half-Edges

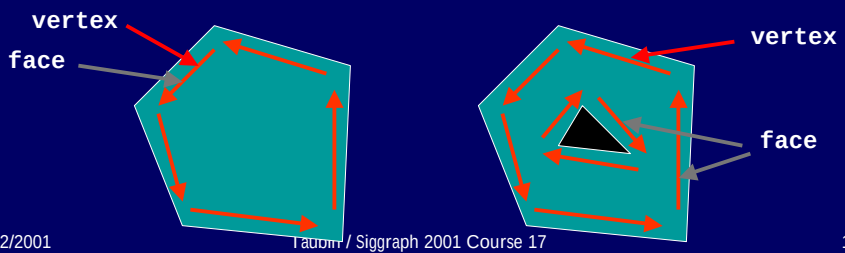





```
halfEdge {
  face
  srcVertex
  nextEdge
  prevEdge
  twinEdge
}
```

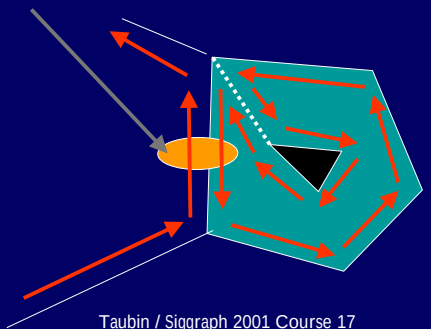
Doubly-linked data structure

- One half-edge per corner of mesh
- Simple Face
  - closed loop of half-edges
- Multiply connected face
  - 1 external loop + one or more internal loops



Orientation

- Consistent if edge is added or removed
- Counterclockwise for outer loop
- Clockwise for inner loops
- Twin edges have opposite orientations

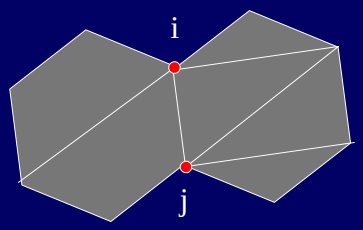


Doubly-linked data structure

- Operations
  - Traversal / triangle strips / compression
  - Surgery / Euler operations
  - Simplification / subdivision
- How to construct from IndexedFaceSet ?
  - Simply connected faces
  - Geometric intersections ignored
- Conversion to manifold
  - Removal of singular edges and vertices

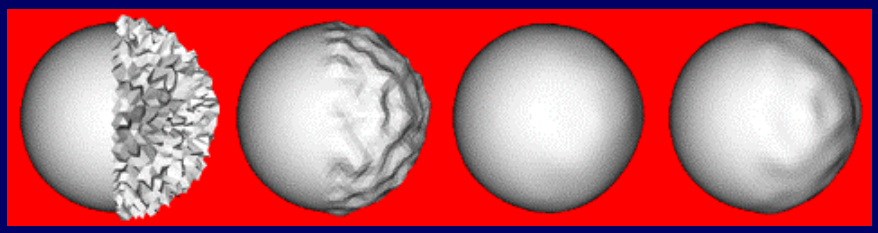
Mesh Graph

- G=(V,E)
- V set of mesh vertex indices {0,…,V-1}
- F set of faces f=[i,j,k,…]
- E set of mesh edges e={i,j}
- i* = { j : {i,j} is in E }
- Signals Defined on the vertices of a graph



Different approaches

- Signal Processing
- Physics-based / PDE Surfaces
- Variational / Regularization
- Multiresolution
- Subdivision Surfaces



Classical Digital Signal Processing

- Signals defined on regular grids
  - 1D : music / speech
  - 2D : images / video
  - 3D : medical imaging
- Shannon Sampling Theorem
- DFT/FFT Fourier Analysis
- FIR/IIR Linear Filters
- Convolution
- Multi-rate filtering / upsampling / downsampling …

The Signal Processing Approach

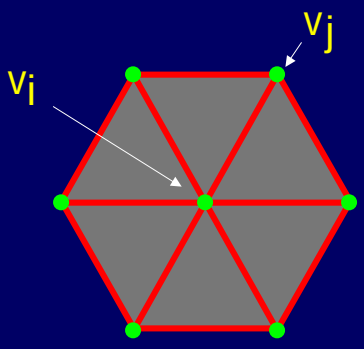
- Laplacian smoothing
  - The shrinkage problem
  - Fourier analysis on meshes
  - Smoothing by partial Fourier expansion
  - Smoothing as low-pass filtering
  - Taubin λ|μ smoothing
  - FIR/IIR filter design
  - Implicit Fairing / Multiresolution modeling
  - Weights / Hard and soft constraints
- Compression of geometry information

Main references

- Taubin λ|μ smoothing (SG'95)
- Taubin-et-al FIR filter design (ECCV'96)
- Desbrun-et-al Implicit smoothing (SG'99)
- Kobelt-et-al Multiresolution smoothing (SG'98)
- Tani-Gotsman Spectral compression (SG'00)
- Balan-Taubin prediction by filtering (CAD'00)
- Khodakovsky-Schroder-Sweldens Progressive Geometry Compression (SG'00)
- Guskov-et-al Multiresolution Signal Processing (SG'99)
- …

Laplacian smoothing in mesh generation

- Used to improve quality of 2D meshes for FE computations
- Move each vertex to the barycenter of its neighbors
- But keep boundary vertices fixed

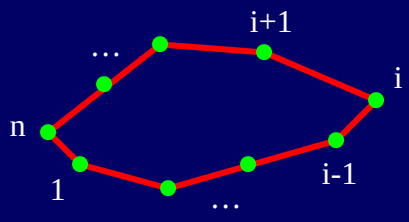


$$v_i' = \frac{1}{n_i} \sum_{j \in i^*} v_j$$

8/12/2001 Taubin / Siggraph 2001 Course 17 21

The 1D Discrete Fourier Transform

- For 1D periodic signals

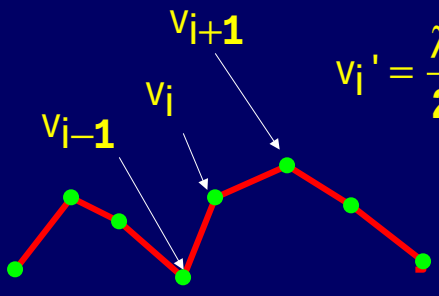


- There is a fast algorithm to compute the DFT : the FFT
- Ideal Low-Pass filter can be implemented
- Complexity is O(n log(n))

8/12/2001 Taubin / Siggraph 2001 Course 17 22

Laplacian smoothing of 1D discrete signals

- Known as Gaussian smoothing
- Convolution of 1D signal with Gaussian kernel
- Also for 2D discrete and continuous signals



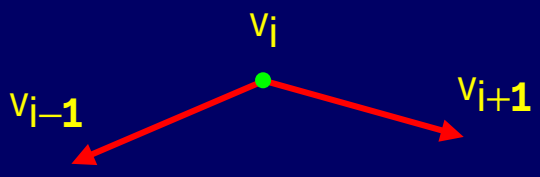
$$v_i' = \frac{\lambda}{2} v_{i-1} + (1-\lambda) v_i + \frac{\lambda}{2} v_{i+1}$$

$$0 < \lambda < 1$$

Laplacian smoothing of 1D discrete signals

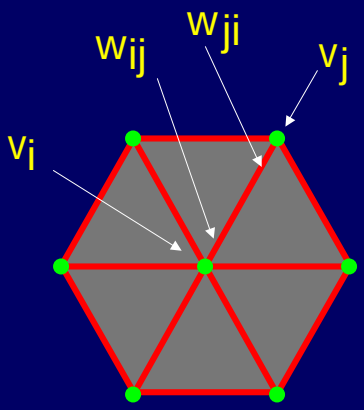
$$v_i' = \frac{\lambda}{2} v_{i-1} + (1-\lambda) v_i + \frac{\lambda}{2} v_{i+1}$$

$$v_i' = v_i + \lambda \left(\frac{1}{2}(v_{i-1} - v_i) + \frac{1}{2}(v_{i+1} - v_i) \right)$$



- Preserves DC

Laplacian smoothing with general weights



$$v_i' = v_i + \lambda \Delta v_i$$

$$\Delta v_i = \sum_j w_{ij}(v_j - v_i)$$

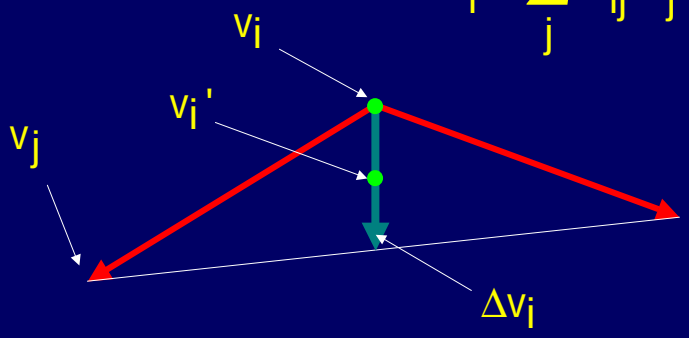
$$\mathbf{1} = \sum_j w_{ij}$$

$$\mathbf{0} \le w_{ij}$$

The Laplacian operator

$$v_i' = v_i + \lambda \Delta v_i$$

$$\Delta v_i = \sum_j w_{ij}(v_j - v_i)$$

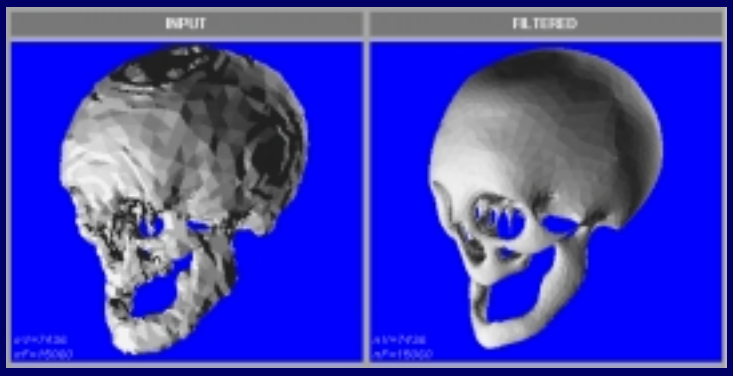


Laplacian smoothing : advantages

- Linear time
- Linear storage
- Edge length equalization (depending on the application)
- Constraints and special effects by weight control

Shrinkage of Laplacian smoothing

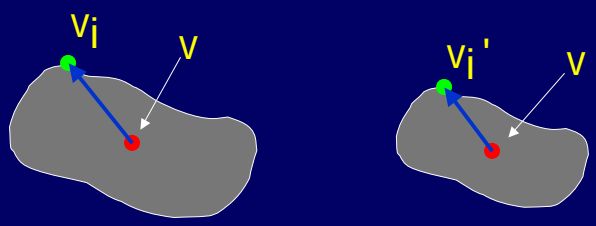
- DEMO !!!



Laplacian smoothing : disadvantages

- Shrinkage
  - Solved by scale adjustment for closed shapes ?
  - What is going on? Fourier analysis
  - Solved by Taubin's algorithm for general shapes
  - Solved by Low-Pass filtering
- Edge length equalization (depending on the application)
  - Fujiwara weights
  - Desbrun-et-al weights

Fixing shrinkage by renormalizing scale



- Adjust scale s to keep distance to barycenter v constant

$$\sum_i \|v_i - v\|^2 = \sum_i \|s(v_i' - v)\|^2$$

$$v_i'' = v + s(v_i' - v)$$

Fixing shrinkage by renormalizing scale

- It is a global solution
- Local perturbation changes shape everywhere
- Does not work !!!
- For a better solution we need to understand why shrinkage occurs
- Fourier Analysis

Fourier analysis on meshes

$$x_i' = x_i + \lambda \sum_j w_{ij}(x_j - x_i) \qquad x' = (I - \lambda K)\, x$$

- Eigenvalues of $K = I - W$ (FREQUENCIES)

$$0 = k_0 \le k_1 \le \cdots \le k_N \le 2$$

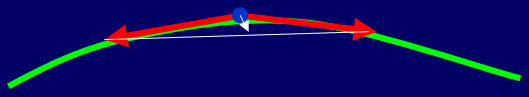
- Right eigenvectors of K (NATURAL VIBRATION MODES)

$$e_0, e_1, \cdots, e_N$$

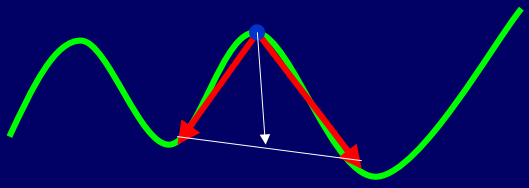
Geometry of low and high frequencies

$$k_h e_{hi} = K e_{hi}' = -\sum_j w_{ij}(e_{hj} - e_{hi})$$

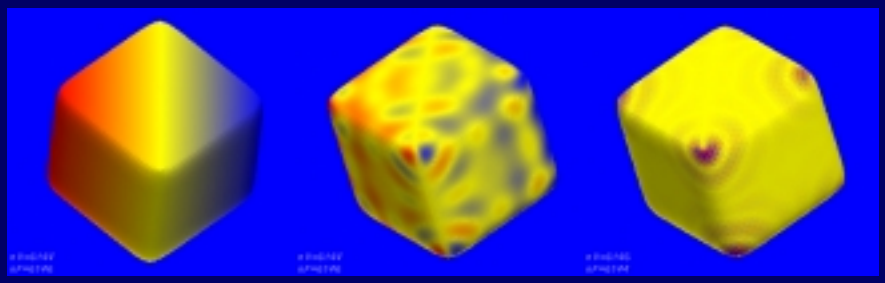
- Low frequency



- High frequency



Natural vibration modes



The Discrete Fourier Transform

- Eigenvectors form a basis of N-space
- Every signal can be written as a linear combination

$$x = \hat{x}_0 e_0 + \hat{x}_1 e_1 + \cdots + \hat{x}_N e_N$$

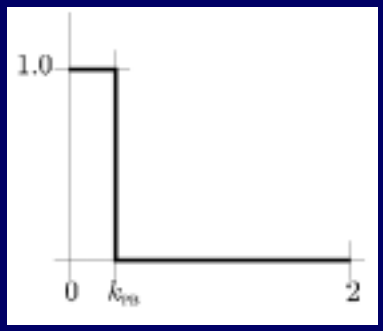
- Discrete Fourier Transform (DFT)

$$\hat{x} = (\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_N)^t$$

The Ideal Low-Pass Filter

- Truncated Fourier expansion

$$x' = \hat{x}_0 e_0 + \hat{x}_1 e_1 + \cdots + \hat{x}_L e_L$$



$$k_L \leq k_{PB}$$

The Discrete Fourier Transform

- Ideal low-pass filtering = truncated Fourier expansion

$$x' = 1\hat{x}_0 e_0 + \cdots + 1\hat{x}_L e_L + 0\hat{x}_{L+1} e_{L+1} + \cdots + 0\hat{x}_N e_N$$

- But eigenvectors cannot be computed !
- Compute an approximation instead : Linear filtering

Polynomial Functions of Matrices

- f(k) univariate polynomial
- K square matrix
- f(K) square matrix
- To evaluate f(K)x only need to know how two multiply a matrix by a vector, a number by a vector, and how to add two vectors

Analysis of Laplacian smoothing

- Laplacian smoothing transfer function

$$x^N = (I - \lambda K)^N x = f(K)\,x$$

- f(k) univariate polynomial (rational later)
- f(K) matrix
- K and f(K) have same eigenvectors
- Eigenvalues of f(K)

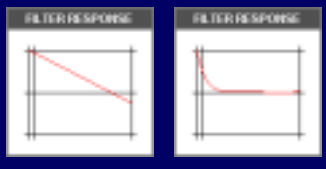
$$f(k_0)\ ,\ f(k_1)\ ,\ \dots\ ,\ f(k_N)$$

Laplacian Smoothing is a Linear Filter

- After filtering

$$f(K)\,x = f(k_0)\,\hat{x}_0\,e_0 + \cdots + f(k_N)\,\hat{x}_N\,e_N$$

- For Laplacian smoothing



$$f(k_0) = 1$$

$$f(k_j) = (1 - \lambda\, k_j)^N \to 0 \qquad j \neq 0 \qquad 0 \leq \lambda < 1$$

- Laplacian smoothing is not a low-pass filter !

Linear Filtering

- After filtering

$$f(K)\,x = f(k_0)\,\hat{x}_0\,e_0 + \cdots + f(k_N)\,\hat{x}_N\,e_N$$

- Evaluation of $f(K)\,x$ is matrix multiplication
- It does not require the computation of eigenvalues and eigenvectors (DFT)

Low-Pass Linear Filtering

- After filtering

$$f(K)\,x = f(k_0)\,\hat{x}_0\,e_0 + \cdots + f(k_N)\,\hat{x}_N\,e_N$$

- Need to find univariate polynomial $f(k)$ such that

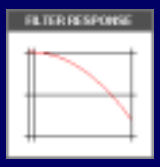
$$f(k_h) \approx 1 \qquad k_L \le k_{PB}$$
$$f(k_h) \approx 0 \qquad k_L > k_{PB}$$

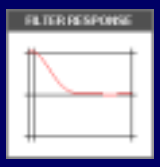
- Need to define efficient evaluation algorithm

Taubin smoothing (Siggraph'95)

- Two steps of Laplacian smoothing
- First shrinking step with positive factor
- Second unshrinking step with negative factor
- Use inverted parabola as transfer function

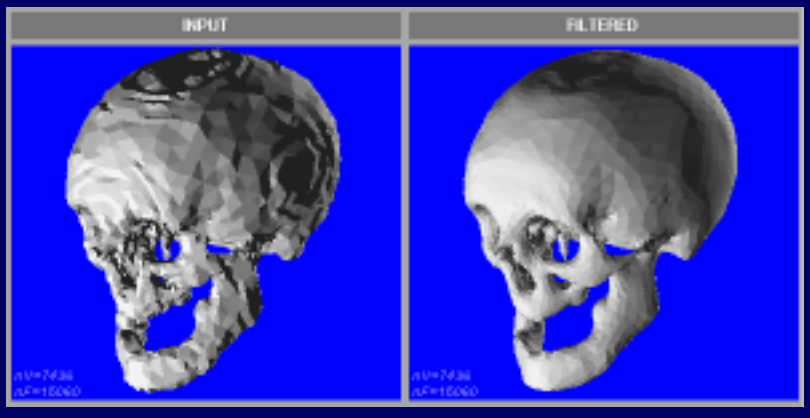
$$f(k) = ((1-\mu k)(1-\lambda k))^N \quad \text{with} \quad -\mu > \lambda > 0$$





Taubin smoothing (Siggraph'95)

- DEMO !!!



Taubin-Zhang-Golub (ECCV'96) FIR filter design

- Efficient algorithm to evaluate any polynomial transfer function
- Based on Chebyshev polynomials defined by three term recursion
- All classical Finite Impulse Response (FIR) filter design techniques can be used with no modifications
- Implemented method of "windows" based on truncated Fourier series expansion of ideal transfer function and coefficient weighting to remove Gibbs phenomenon
- DEMO !!!

FIR filters vs. IIR filters

- Sharp transitions and narrow pass-bands require very high degree polynomial transfer functions
- Infinite Inpulse Response (IIR) filters with rational transfer functions can produce good approximations using polynomials of low degree
- But require the solution of sparse linear systems
- Is it worth the effort ?

IIR filters

- If f(k)=g(k)/h(k), with h(k) non-zero in [0,2]
- Filtering a signal x requires solving the system

$$h(K)\,x' = g(K)\,x$$

- y = g(K) x is an FIR filter
- With H = h(K) solving H x' = y with the Preconditioned Biconjugate Gradients algorithm (PBCG) only requires methods to multiply a vector z by H and by H^t and a preconditioner H'

Desbrun-Meyer-Schroder-Barr (SG'99) Implicit fairing

- Corresponds to the classical Butterworth filter with transfer function

$$f(k) = \frac{1}{1 + (k/k_{PB})^N}$$

- Need to solve sparse (for small N) linear system

$$(I + (1/k_{PB})^N K^N)\,x' = x$$

- But with PDE formulation in the paper

Implicit fairing

- Laplacian smoothing corresponds to the numerical solution of

$$\frac{\partial x}{\partial t} = \lambda dt \, \Delta x$$

- using the forward Euler method

$$x' = x + \lambda dt \, \Delta x = (I + \lambda dt \, \Delta) x$$

- They use the backward Euler method instead

$$(I - \lambda dt \, \Delta) x' = x$$

- Stable for large time steps (true or false ?)
- DEMO !!!

Kobelt-et-al Multiresolution modeling (Siggraph'98)

- Minimize membrane energy $E_M = \|\Delta x\|^2$
- or thin plate energy $E_{TP} = \|\Delta^2 x\|^2$
- Requires boundary vertex position constraints
- Speed-up by multi-grid approach
- Jacobi updates similar to Laplacian and Taubin updates
- How does it compare with single-res FIR filters ?
- DEMO !!!

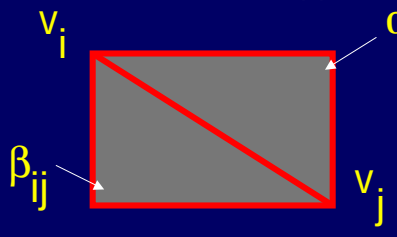
What can be done with the weights ?

$$\Delta v_i = \sum_j w_{ij}(v_j - v_i)$$

- Weights
  - Neighborhoods = non-zero weights
  - Prevention of Tangencial drift
  - Edge-length equalization
- Boundaries and creases / hierarchical smoothing
- Vertex-dependent smoothing parameters

Preventing tangencial drift

- Fujiwara (P-AMS'95)
  - Weights inversely proportional to edge length
- Desbrun-Meyer-Schroder-Barr (SG'99)
  - Based on better approximation of curvature normal



$$c_{ij} = \cot(\alpha_{ij}) + \cot(\beta_{ij})'$$

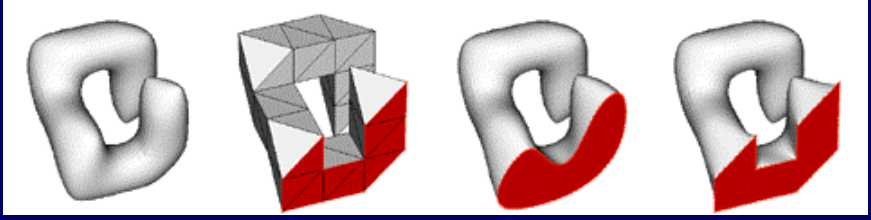
- Guskov-et-al (SG'99) based on divided differences and second order neighborhood

Smoothing with Constraints

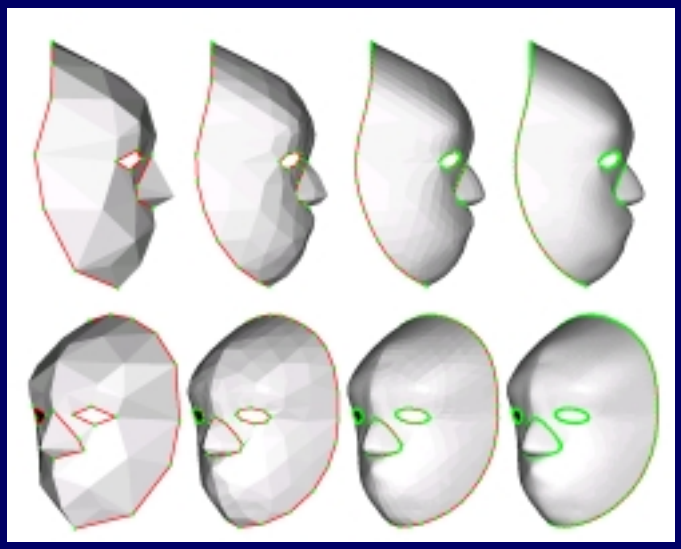
- Boundaries
- Creases
- Singular edges
- Imposing Vertex Constraints / Discrete Fairing
  - Lack of normal control
- Imposing Normal Constraints
- Detecting and Enhancing Creases
  - Anisotropic Difussion
  - Non-Linear filtering
  - Evolution of Weights

Hierarchical neighborhoods

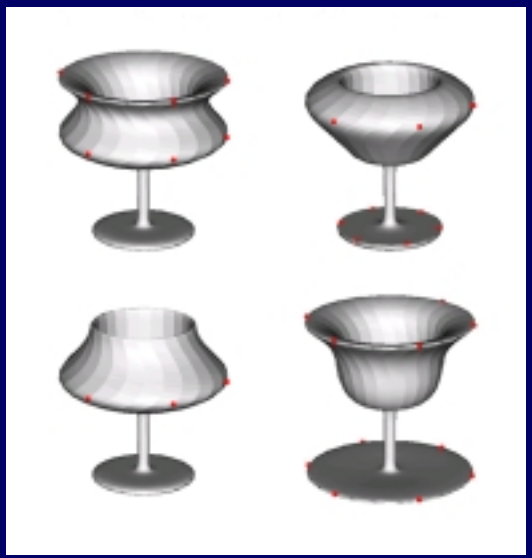
- Assign a numeric label to each vertex
- Vertex j is a neighbor of vertex i only if i and j are connected by an edge, and the label of i is less or equal than the label of j



Boundaries and creases



Vertex Constraints and Surface Design



Boundaries and creases

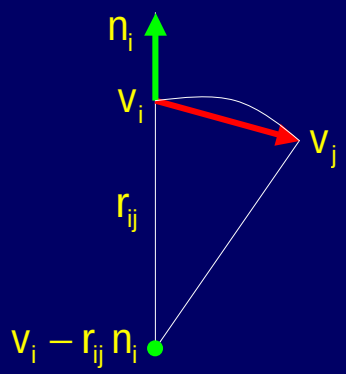
- Use hierarchical neighborhoods
- Assign label 1 to boundary and crease vertices
- Assign label 0 to all internal vertices
- The graph defined by the boundary and crease edges and vertices is smoothed independently of the rest of the mesh
- The rest of the mesh “follows” the graph defined by the boundary and crease edges and vertices

Position and normal constraints

- Hard vs. soft constraints
- Hard vertex position constraints are easy to impose
- General hard linear constraints require solving small linear systems
- Yamada-et-al Discrete Spring Model (PCCGA’98) impose soft normal constraints with a spring model that adds an extra term to the smoothing step
- Slow convergence and/or high computational cost
- Multi-resolution helps
- More work needed

Curvature-based Sampling

- Silva-Taubin Curvature-based sampling (SIAM-GD'99)
- Taubin Tensor of curvature (ICCV'95)



$$\left\| v_j - v_i + r_{ij}\, n_i \right\|^2 = r_{ij}^{\,2}$$

$$\sigma_{ij} = \frac{\left\| v_j - v_i \right\|}{r_{ij}} = \frac{2\, n_i^{\,t} (v_j - v_i)}{\left\| v_j - v_i \right\|}$$

Geometry compression

- Static or single-resolution vs. progressive
- Connectivity, geometry, and properties
- Geometry and properties cost much more than connectivity
- Commercial grade single-resolution methods available
  - Taubin-Rossignac Topological Surgery (MPEG-4/ IBM HotMedia)
  - Touma-Gotsman (Virtue Ltd.)
- Need better geometry prediction/compression schemes

Tani-Gotsman (Siggraph'00) Spectral Compression

- Based on partial DFT expansion
- Connectivity is transmitted first
- Encoder computes Eigenvalues/Eigenvectors of matrix K to evaluate Fourier coefficients
- Fourier coefficients are transmitted
- Decoder computes Eigenvalues/Eigenvectors of matrix K to reconstruct the partial sum
- Mesh partition into smaller submeshes to be able to deal with the numerical restrictions
- Need to compute lots of Eigenvalues/Eigenvectors

Balan-Taubin prediction by filtering (CAD'00)

- Based on a vertex clustering hierarchy (PM, PFS, etc.)
- Connectivity is transmitted progressively interlieved with geometry data
- Fine Geometry is predicted from coarse geometry by filtering the coarse geometry on the fine mesh
- Filter coefficients are determined by solving a LS problem
- Corrections are not transmitted

$$\min_f \| x_F - f(K)\, x_C \|^2$$

Khodakovsky-Schroder-Sweldens Progressive Geometry Compression (SG'00)

- Good for large densely sampled meshes with low topological complexity (3D scanning, etc.)
- MAPS Remeshing produces subdivision surface
- Wavelet compression
- Zero-tree encoding
- Very good results reported

Conclusion / To Do

- Fast and efficient methods to smooth with hard and soft constraints
- Relation to subdivision surfaces
- Global vs. local behavior of smoothing operators
- Goal: interactive free-form modeling based on intuitive user interface to manipulate constraints, remesh, simplify, etc.
- Goal: practical and effective methods for the compression of geometry data.
- Implementation of other popular SP operations

Geometric Signal Processing on Large Polygonal Meshes
COURSE NOTES FOR SIGGRAPH'2001
LOS ANGELES, CALIFORNIA
AUGUST 12, 2001

Leif Kobbelt

Rheinisch-Westfälische Technische Hochschule Aache

Reprints

A Signal Processing Approach to Fair Surface Design

by G. Taubin,

Siggraph'95

A Signal Processing Approach To Fair Surface Design

Gabriel Taubin[1]
IBM T.J.Watson Research Center

ABSTRACT

In this paper we describe a new tool for interactive free-form fair surface design. By generalizing classical discrete Fourier analysis to two-dimensional *discrete surface signals* – functions defined on polyhedral surfaces of arbitrary topology –, we reduce the problem of surface smoothing, or fairing, to low-pass filtering. We describe a very simple surface signal low-pass filter algorithm that applies to surfaces of arbitrary topology. As opposed to other existing optimization-based fairing methods, which are computationally more expensive, this is a linear time and space complexity algorithm. With this algorithm, fairing very large surfaces, such as those obtained from volumetric medical data, becomes affordable. By combining this algorithm with surface subdivision methods we obtain a very effective fair surface design technique. We then extend the analysis, and modify the algorithm accordingly, to accommodate different types of constraints. Some constraints can be imposed without any modification of the algorithm, while others require the solution of a small associated linear system of equations. In particular, vertex location constraints, vertex normal constraints, and surface normal discontinuities across curves embedded in the surface, can be imposed with this technique.

CR Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/image generation - *display algorithms*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling - *curve, surface, solid, and object representations*; J.6 [**Computer Applications**]: Computer-Aided Engineering - *computer-aided design*

General Terms: Algorithms, Graphics.

1 INTRODUCTION

The signal processing approach described in this paper was originally motivated by the problem of how to fair large polyhedral surfaces of arbitrary topology, such as those extracted from volumetric medical data by iso-surface construction algorithms [21, 2, 11, 15], or constructed by integration of multiple range images [36].

Since most existing algorithms based on fairness norm optimization [37, 24, 12, 38] are prohibitively expensive for very large surfaces – a million vertices is not unusual in medical images –, we decided to look for new algorithms with linear time and space complexity [31]. Unless these large surfaces are first simplified [29, 13, 11], or re-meshed using far fewer faces [35], methods based on patch technology, whether parametric [28, 22, 10, 20, 19] or implicit [1, 23], are not acceptable either. Although curvature continuous, a patch-based surface interpolant is far more complex than the original surface, more expensive to render, and worst of all, does not remove the high curvature variation present in the original mesh.

As in the fairness norm optimization methods and physics-based deformable models [16, 34, 30, 26], our approach is to move the vertices of the polyhedral surface without changing the connectivity of the faces. The faired surface has exactly the same number of vertices and faces as the original one. However, our signal processing formulation results in much less expensive computations. In these variational formulations [5, 24, 38, 12], after finite element discretization, the problem is often reduced to the solution of a large sparse linear system, or a more expensive global optimization problem. Large sparse linear systems are solved using iterative methods [9], and usually result in quadratic time complexity algorithms. In our case, the problem of surface fairing is reduced to sparse matrix multiplication instead, a linear time complexity operation.

The paper is organized as follows. In section 2 we describe how to extend signal processing to signals defined on polyhedral surfaces of arbitrary topology, reducing the problem of surface smoothing to low-pass filtering, and we describe a particularly simple linear time and space complexity surface signal low-pass filter algorithm. Then we concentrate on the applications of this algorithm to interactive free-form fair surface design. As Welch and Witkin [38], in section 3 we design more detailed fair surfaces by combining our fairing algorithm with subdivision techniques. In section 4 we modify our fairing algorithm to accommodate different kinds of constraints. Finally, in section 5 we present some closing remarks.

2 THE SIGNAL PROCESSING APPROACH

Fourier analysis is a natural tool to solve the problem of signal smoothing. The space of signals – functions defined on certain domain – is decomposed into orthogonal subspaces associated with different frequencies, with the low frequency content of a signal regarded as subjacent data, and the high frequency content as noise.

2.1 CLOSED CURVE FAIRING

To smooth a closed curve it is sufficient to remove the noise from the coordinate signals, i.e., to project the coordinate signals onto the subspace of low frequencies. This is what the method of *Fourier descriptors*, which dates back to the early 60's, does [40]. Our approach to extend Fourier analysis to signals defined on polyhedral surfaces of arbitrary topology is based on the observation that the classical Fourier transform of a signal can be seen as the decomposition of the signal into a linear combination of the eigenvectors of the Laplacian operator. To extend Fourier analysis to surfaces of arbitrary topology we only have to define a new operator that takes the place of the Laplacian.

As a motivation, let us consider the simple case of a discrete time n-periodic signal – a function defined on a regular polygon of n vertices –, which we represent as a column vector $x = (x_1, \ldots, x_n)^t$. The components of this vector are the values of the signal at the

[1]IBM T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, taubin@watson.ibm.com

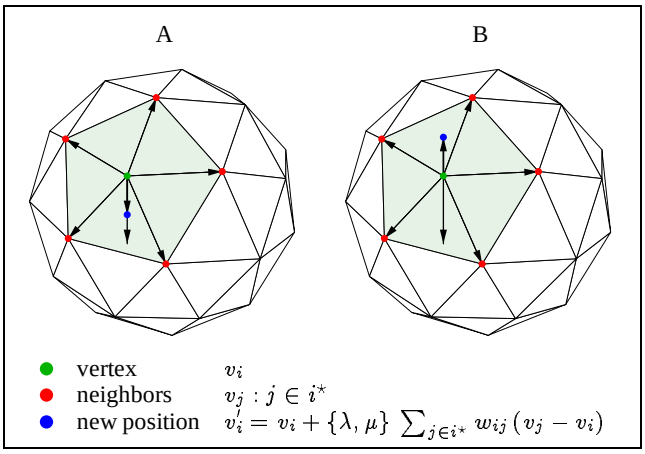


Figure 1: The two weighted averaging steps of our fairing algorithm. (A) A first step with positive scale factor λ is applied to all the vertices. (B) Then a second step with negative scale factor μ is applied to all the vertices.

vertices of the polygon. The discrete Laplacian of x is defined as

$$\Delta x_i = \frac{1}{2}(x_{i-1} - x_i) + \frac{1}{2}(x_{i+1} - x_i) \ , \tag{1}$$

where the indices are incremented and decremented modulo n. In matrix form it can be written as follows

$$\Delta x = -K x \ , \tag{2}$$

where K is the circulant matrix

$$K = \frac{1}{2} \begin{pmatrix} 2 & -1 & & & -1 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ -1 & & & -1 & 2 \end{pmatrix} .$$

Since K is symmetric, it has real eigenvalues and eigenvectors. Explicitly, the real eigenvalues $k_1, \ldots, k_n$ of K, sorted in non-decreasing order, are

$$k_j = 1 - \cos(2\pi \lfloor j/2 \rfloor / n) \ ,$$

and the corresponding unit length real eigenvectors, $u_1, \ldots, u_n$, are

$$(u_j)_h = \begin{cases} \sqrt{1/n} & \text{if } j = 1 \\ \sqrt{2/n}\, \sin(2\pi h \lfloor j/2 \rfloor / n) & \text{if } j \text{ is even} \\ \sqrt{2/n}\, \cos(2\pi h \lfloor j/2 \rfloor / n) & \text{if } j \text{ is odd} \ . \end{cases}$$

Note that $0 \leq k_1 \leq \cdots \leq k_n \leq 2$, and as the frequency k_j increases, the corresponding eigenvector u_j, as a n-periodic signal, changes more rapidly from vertex to vertex.

To decompose the signal x as a linear combination of the real eigenvectors $u_1, \ldots, u_n$

$$x = \sum_{i=1}^{n} \xi_i u_i \ , \tag{3}$$

is computationally equivalent to the evaluation of the Discrete Fourier Transform of x. To smooth the signal x with the method of Fourier descriptors, this decomposition has to be computed, and then the high frequency terms of the sum must be discarded. But

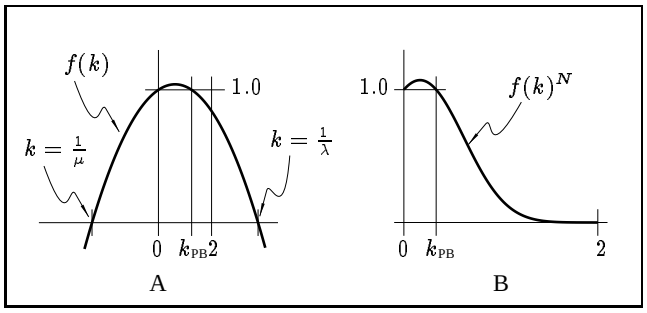


Figure 2: (A) Graph of transfer function $f(k) = (1 - \mu k)(1 - \lambda k)$ of non-shrinking smoothing algorithm.

this is computationally expensive. Even using the Fast Fourier Transform algorithm, the computational complexity is in the order of $n \log(n)$ operations.

An alternative is to do the projection onto the space of low frequencies only approximately. This is what a low-pass filter does. We will only consider here low-pass filters implemented as a convolution. A more detailed analysis of other filter methodologies is beyond the scope of this paper, and will be done elsewhere [33]. Perhaps the most popular convolution-based smoothing method for parameterized curves is the so-called *Gaussian filtering* method, associated with scale-space theory [39, 17]. In its simplest form, it can be described by the following formula

$$x_i' = x_i + \lambda \, \Delta x_i \ , \tag{4}$$

where $0 < \lambda < 1$ is a scale factor (for $\lambda < 0$ and $\lambda \geq 1$ the algorithm enhances high frequencies instead of attenuating them). This can be written in matrix form as

$$x' = (I - \lambda K)\, x \ . \tag{5}$$

It is well known though, that Gaussian filtering produces shrinkage, and this is so because the Gaussian kernel is not a low-pass filter kernel [25]. To define a low-pass filter, the matrix $I - \lambda K$ must be replaced by some other function $f(K)$ of the matrix K. Our non-shrinking fairing algorithm, described in the next section, is one particularly efficient choice.

We now extend this formulation to functions defined on surfaces of arbitrary topology.

2.2 SURFACE SIGNAL FAIRING

At this point we need a few definitions. We represent a polyhedral surface as a pair of lists $S = \{V, F\}$, a list of n vertices V, and a list of polygonal faces F. Although in our current implementation, only triangulated surfaces, and surfaces with quadrilateral faces are allowed, the algorithm is defined for any polyhedral surface.

Both for curves and for surfaces, a *neighborhood* of a vertex v_i is a set $i^\star$ of indices of vertices. If the index j belongs to the neighborhood $i^\star$, we say that v_j is a *neighbor* of v_i. The *neighborhood structure* of a polygonal curve or polyhedral surface is the family of all its neighborhoods $\{i^\star : i = 1, 2, \ldots, n\}$. A neighborhood structure is *symmetric* if every time that a vertex v_j is a neighbor of vertex v_i, also v_i is a neighbor of v_j. With non-symmetric neighborhoods certain constraints can be imposed. We discuss this issue in detail in section 4.

A particularly important neighborhood structure is the *first order* neighborhood structure, where for each pair of vertices v_i and v_j that share a face (edge for a curve), we make v_j a neighbor of v_i, and v_i a neighbor of v_j. For example, for a polygonal curve represented as a list of consecutive vertices, the first order neighborhood of a vertex v_i is $i^\star = \{i - 1, i + 1\}$. The first order neighborhood

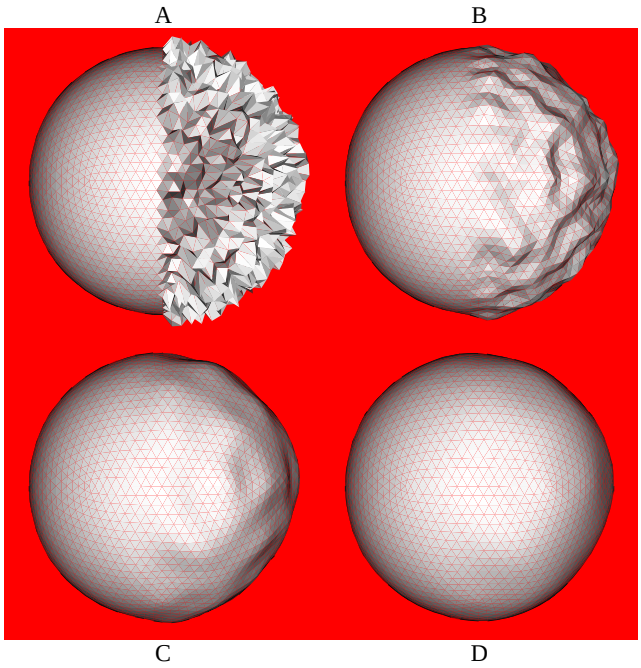


Figure 3: (A) Sphere partially corrupted by normal noise. (B) Sphere (A) after 10 non-shrinking smoothing steps. (C) Sphere (A) after 50 non-shrinking smoothing steps. (D) Sphere (A) after 200 non-shrinking smoothing steps. Surfaces are flat-shaded to enhance the faceting effect.

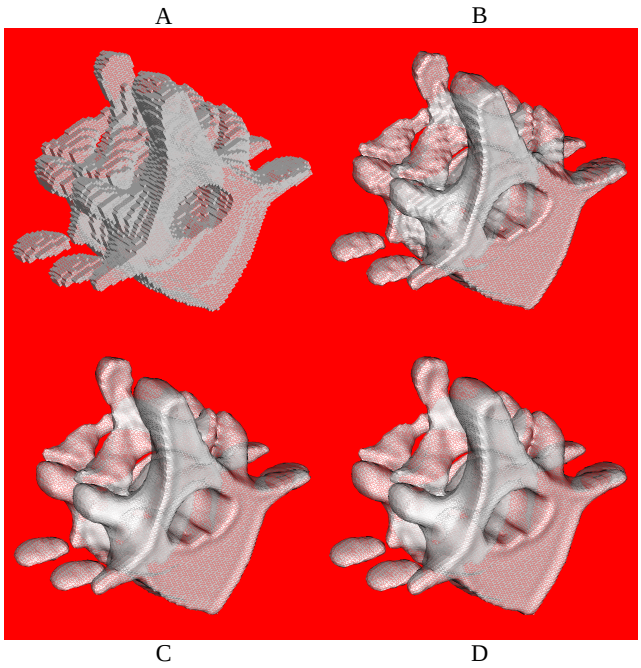


Figure 4: (A) Boundary surface of voxels from a CT scan. (B) Surface (A) after 10 non-shrinking smoothing steps. (C) Surface (A) after 50 non-shrinking smoothing steps. (D) Surface (A) after 100 non-shrinking smoothing steps. $k_{\text{PB}} = 0.1$ and $\lambda = 0.6307$ in (B), (C), and (D). Surfaces are flat-shaded to enhance the faceting effect.

structure is symmetric, and since it is implicitly given by the list of faces of the surface, no extra storage is required to represent it. This is the default neighborhood structure used in our current implementation.

A *discrete surface signal* is a function $x = (x_1, \ldots, x_n)^t$ defined on the vertices of a polyhedral surface. We define the discrete Laplacian of a discrete surface signal by weighted averages over the neighborhoods

$$\Delta x_i = \sum_{j \in i^*} w_{ij} \, (x_j - x_i) \ , \tag{6}$$

where the weights w_{ij} are positive numbers that add up to one , $\sum_{j \in i^*} w_{ij} = 1$, for each i. The weights can be chosen in many different ways taking into consideration the neighborhood structures. One particularly simple choice that produces good results is to set w_{ij} equal to the inverse of the number of neighbors $1/|i^*|$ of vertex v_i, for each element j of i^*. Note that the case of the Laplacian of a n-periodic signal (1) is a particular case of these definitions. A more general way of choosing weights for a surface with a first order neighborhood structure, is using a positive function $\phi(v_i, v_j) = \phi(v_j, v_i)$ defined on the edges of the surface

$$w_{ij} = \frac{\phi(v_i, v_j)}{\sum_{h \in i^*} \phi(v_i, v_h)} \ .$$

For example, the function can be the surface area of the two faces that share the edge, or some power of the length of the edge $\phi(v_i, v_j) = \|v_i - v_j\|^\alpha$. In our implementation the user can choose any one of these weighting schemes. They produce similar results when the surface has faces of roughly uniform size. When using a power of the length of the edges as weighting function, the exponent $\alpha = -1$ produces good results.

If $W = (w_{ij})$ is the matrix of weights, with $w_{ij} = 0$ when j is not a neighbor of i, the matrix K can now be defined as $K = I - W$. In the appendix we show that for a first order neighborhood structure, and for all the choices of weights described above, the matrix K has real eigenvalues $0 \leq k_1 \leq k_2 \leq \cdots \leq k_n \leq 2$ with corresponding linearly independent real unit length right eigenvectors $u_1, \ldots, u_n$. Seen as discrete surface signals, these eigenvectors should be considered as the *natural vibration modes* of the surface, and the corresponding eigenvalues as the associated *natural frequencies*.

The decomposition of equation (3), of the signal x into a linear combination of the eigenvectors $u_1, \ldots, u_n$, is still valid with these definitions, but there is no extension of the Fast Fourier Transform algorithm to compute it. The method of Fourier descriptors – the exact projection onto the subspace of low frequencies – is just not longer feasible, particularly for very large surfaces. On the other hand, low-pass filtering – the approximate projection – can be formulated in exactly the same way as for n-periodic signals, as the multiplication of a function $f(K)$ of the matrix K by the original signal

$$x' = f(K)\, x \ ,$$

and this process can be iterated N times

$$x^N = f(K)^N \, x \ .$$

The function of one variable $f(k)$ is the *transfer function* of the filter. Although many functions of one variable can be evaluated in matrices [9], we will only consider polynomials here. For example, in the case of Gaussian smoothing the transfer function is $f(k) = 1 - \lambda k$. Since for any polynomial transfer function we have

$$x' = f(K)\, x = \sum_{i=1}^{n} \xi_i \, f(k_i) \, u_i \ ,$$

because $K u_i = k_i u_i$, to define a low-pass filter we need to find a polynomial such that $f(k_i)^N \approx 1$ for low frequencies, and

$f(k_i)^N \approx 0$ for high frequencies in the region of interest $k \in [0, 2]$. Our choice is

$$f(k) = (1 - \lambda k)(1 - \mu k) \tag{7}$$

where $0 < \lambda$, and μ is a new negative scale factor such that $\mu < -\lambda$. That is, after we perform the Gaussian smoothing step of equation (4) with positive scale factor λ for all the vertices – the shrinking step –, we then perform another similar step

$$x_i' = x_i + \mu \, \Delta x_i \tag{8}$$

for all the vertices, but with negative scale factor μ instead of λ – the un-shrinking step –. These steps are illustrated in figure 1.

The graph of the transfer function of equation (7) is illustrated in figure 2-A. Figure 2-B shows the resulting transfer function after N iterations of the algorithm, the graph of the function $f(k)^N$. Since $f(0) = 1$ and $\mu + \lambda < 0$, there is a positive value of k, the *pass-band frequency* k_{PB}, such that $f(k_{PB}) = 1$. The value of k_{PB} is

$$k_{PB} = \frac{1}{\lambda} + \frac{1}{\mu} > 0 \ . \tag{9}$$

The graph of the transfer function $f(k)^N$ displays a typical *low-pass filter* shape in the region of interest $k \in [0, 2]$. The *pass-band region* extends from $k = 0$ to $k = k_{PB}$, where $f(k)^N \approx 1$. As k increases from $k = k_{PB}$ to $k = 2$, the transfer function decreases to zero. The faster the transfer function decreases in this region, the better. The rate of decrease is controlled by the number of iterations N.

This algorithm is fast (linear both in time and space), extremely simple to implement, and produces smoothing without shrinkage. Faster algorithms can be achieved by choosing other polynomial transfer functions, but the analysis of the filter design problem is beyond the scope of this paper, and will be treated elsewhere [33]. However, as a rule of thumb, the filter based on the second degree polynomial transfer function of equation (7) can be designed by first choosing a values of k_{PB}. Values from 0.01 to 0.1 produce good results, and all the examples shown in the paper where computed with $k_{PB} \approx 0.1$. Once k_{PB} has been chosen, we have to choose λ and N (μ comes out of equation (9) afterwards). Of course we want to minimize N, the number of iterations. To do so, λ must be chosen as large as possible, while keeping $|f(k)| < 1$ for $k_{PB} < k \leq 2$ (if $|f(k)| \geq 1$ in $[k_{PB}, 2]$, the filter will enhance high frequencies instead of attenuating them). In some of the examples, we have chosen λ so that $f(1) = -f(2)$. For $k_{PB} < 1$ this choice of λ ensures a stable and fast filter.

Figures 3 and 4 show examples of large surfaces faired with this algorithm. Figures 3 is a synthetic example, where noise has been added to one half of a polyhedral approximation of a sphere. Note that while the algorithm progresses the half without noise does not change significantly. Figure 4 was constructed from a CT scan of a spine. The boundary surface of the set of voxels with intensity value above a certain threshold is used as the input signal. Note that there is not much difference between the results after 50 and 100 iterations.

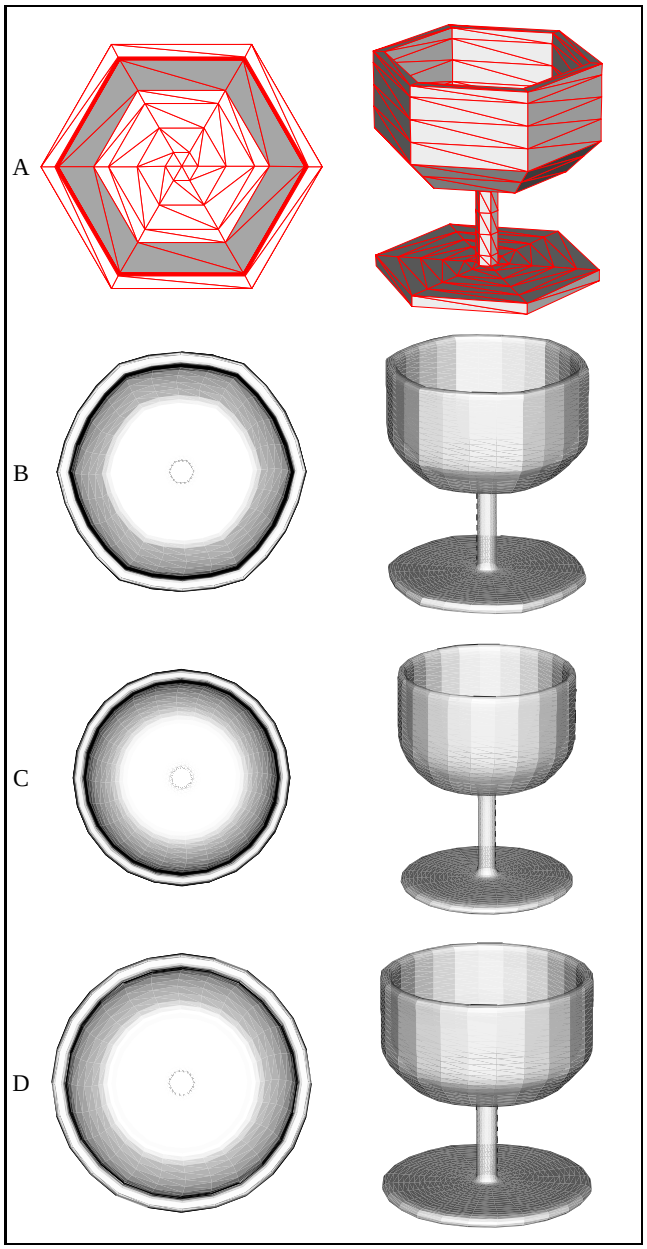


Figure 5: Surfaces created alternating subdivision and different smoothing steps. (A) Skeleton surface. (B) One Gaussian smoothing step ($\lambda = 0.5$). Note the hexagonal symmetry because of insufficient smoothing. (C) Five Gaussian smoothing steps ($\lambda = 0.5$). Note the shrinkage effect. (D) Five non-shrinking smoothing steps ($k_{PB} = 0.1$ and $\lambda = 0.6307$) of this paper. (B),(C), and (D) are the surfaces obtained after two levels of refinement and smoothing. Surfaces are flat-shaded to enhance the faceting effect.

3 SUBDIVISION

A subdivision surface is a smooth surface defined as the limit of a sequence of polyhedral surfaces, where the next surface in the sequence is constructed from the previous one by a refinement process. In practice, since the number of faces grows very fast, only a few levels of subdivision are computed. Once the faces are smaller than the resolution of the display, it is not necessary to continue. As Welch and Witkin [38], we are not interested in the limit surfaces, but rather in using subdivision and smoothing steps as tools to design fair polyhedral surfaces in an interactive environment. The classical subdivision schemes [8, 4, 12] are rigid, in the sense that they have no free parameters that influence the behavior of the algorithm as it progresses trough the subdivision process. By using our fairing algorithm in conjunction with subdivision steps, we achieve more flexibility in the design process. In this way our fairing algorithm can be seen as a complement of the existing subdivision strategies.

In the subdivision surfaces of Catmull and Clark [4, 12] and Loop [18, 6], the subdivision process involves two steps. A refinement step, where a new surface with more vertices and faces is created, and a smoothing step, where the vertices of the new sur-

face are moved. The Catmull and Clark refinement process creates polyhedral surfaces with quadrilateral faces, and Loop refinement process subdivides each triangular face into four similar triangular faces. In both cases the smoothing step can be described by equation (4). The weights are chosen to ensure tangent or curvature continuity of the limit surface.

These subdivision surfaces have the problem of shrinkage, though. The limit surface is significantly smaller overall than the initial skeleton mesh – the first surface of the sequence –. This is so because the smoothing step is essentially Gaussian smoothing, and as we have pointed out, Gaussian smoothing produces shrinkage. Because of the refinement steps, the surfaces do not collapse to the centroid of the initial skeleton, but the shrinkage effect can be quite significant.

The problem of shrinkage can be solved by a global operation. If the amount of shrinkage can be predicted in closed form, the skeleton surface can be expanded before the subdivision process is applied. This is what Halstead, Kass, and DeRose [12] do. They show how to modify the skeleton mesh so that the subdivision surface associated with the modified skeleton interpolates the vertices of the original skeleton.

The subdivision surfaces of Halstead, Kass, and DeRose interpolate the vertices of the original skeleton, and are curvature continuous. However, they show a significant high curvature content, even when the original skeleton mesh does not have such undulations. The shrinkage problem is solved, but a new problem is introduced. Their solution to this second problem is to stop the subdivision process after a certain number of steps, and fair the resulting polyhedral surface based on a variational approach. Their fairness norm minimization procedure reduces to the solution of a large sparse linear system, and they report quadratic running times. The result of this modified algorithm is no longer a curvature continuous surface that interpolates the vertices of the skeleton, but a more detailed fair polyhedral surface that usually does not interpolate the vertices of the skeleton unless the interpolatory constraints are imposed during the fairing process.

We argue that the source of the unwanted undulations in the Catmull-Clark surface generated from the modified skeleton is the smoothing step of the subdivision process. Only one Gaussian smoothing step does not produce enough smoothing, i.e., it does not produce sufficient attenuation of the high frequency components of the surfaces, and these high frequency components persist during the subdivision process. Figure 5-B shows an example of a subdivision surface created with the triangular refinement step of Loop, and one Gaussian smoothing step of equation (4). The hexagonal symmetry of the skeleton remains during the subdivision process. Figure 5-C shows the same example, but where five Gaussian smoothing steps are performed after each refinement step. The hexagonal symmetry has been removed at the expense of significant shrinkage effect. Figure 5-D shows the same example where the five non-shrinking fairing steps are performed after each refinement step. Neither hexagonal symmetry nor shrinkage can be observed.

4 CONSTRAINTS

Although surfaces created by a sequence of subdivision and smoothing steps based on our fairing algorithm do not shrink much, they usually do not interpolate the vertices of the original skeleton. In this section we show that by modifying the neighborhood structure certain kind of constraints can be imposed without any modification of the algorithm. Then we study other constraints that require minor modifications.

4.1 INTERPOLATORY CONSTRAINTS

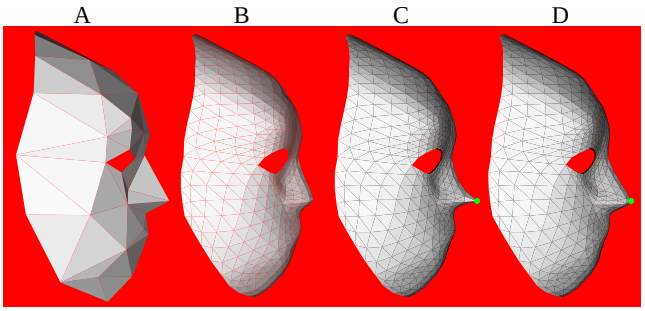


Figure 6: Example of surfaces designed using subdivision and smoothing steps with one interpolatory constraint. (A) Skeleton. (B) Surface (A) after two levels of subdivision and smoothing without constraints. (C) Same as (B) but with non-smooth interpolatory constraint. (D) Same as (B) but with smooth interpolatory constraint. Surfaces are flat-shaded to enhance the faceting effect.

As we mentioned in section 2.2, a simple way to introduce interpolatory constraints in our fairing algorithm is by using non-symmetric neighborhood structures. If no other vertex is a neighbor of a certain vertex v_1, i.e., if the neighborhood of v_1 is empty, then the value x_1 of any discrete surface signal x does not change during the fairing process, because the discrete Laplacian Δx_1 is equal to zero by definition of empty sum. Other vertices are allowed to have v_1 as a neighbor, though. Figure 6-A shows a skeleton surface. Figure 6-B shows the surface generated after two levels of refinement and smoothing using our fairing algorithm without constraints, i.e., with symmetric first-order neighborhoods. Although the surface has not shrunk overall, the nose has been flattened quite significantly. This is so because the nose is made of very few faces in the skeleton, and these faces meet at very sharp angles. Figure 6-C shows the result of applying the same steps, but defining the neighborhood of the vertex at the tip of the nose to be empty. The other neighborhoods are not modified. Now the vertex satisfies the constraint – it has not moved at all during the fairing process –, but the surface has lost its smoothness at the vertex. This might be the desired effect, but if it is not, instead of the neighborhoods, we have to modify the algorithm.

4.2 SMOOTH INTERPOLATION

We look at the desired constrained smooth signal x_C^N as a sum of the corresponding unconstrained smooth signal $x^N = F\,x$ after N steps of our fairing algorithm (i.e. $F = f(K)^N$), plus a smooth deformation d_1

$$x_C^N = x^N + (x_1 - x_1^N)\, d_1 \ .$$

The deformation d_1 is itself another discrete surface signal, and the constraint $(x_C^N)_1 = x_1$ is satisfied if $(d_1)_1 = 1$. To construct such a smooth deformation we consider the signal δ_1, where

$$(\delta_i)_j = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases} \ .$$

This is not a smooth signal, but we can apply the fairing algorithm to it. The result, let us denote it F_{n1}, the first column of the matrix F, is a smooth signal, but its value at the vertex v_1 is not equal to one. However, since the matrix F is diagonally dominated, F_{11}, the first element of its first column, must be non-zero. Therefore, we can scale the signal F_{n1} to make it satisfy the constraint, obtaining the desired smooth deformation

$$d_1 = F_{n1}\, F_{11}^{-1} \ .$$

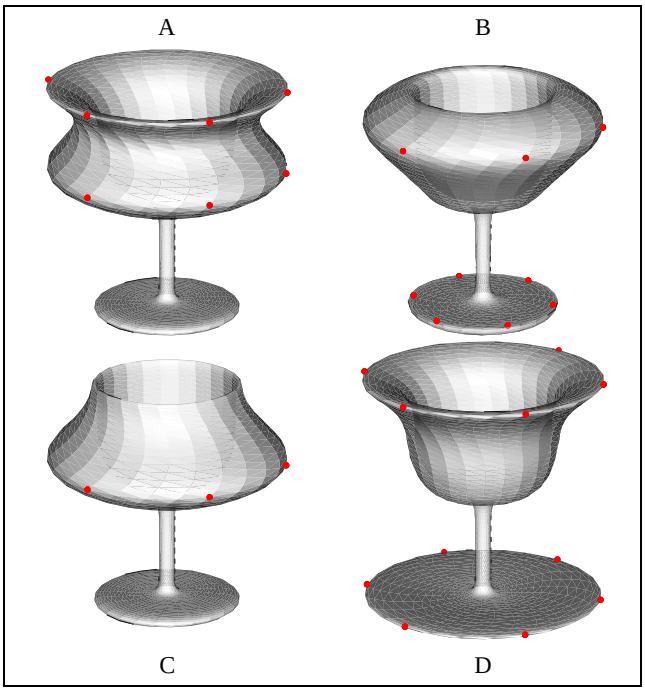


Figure 7: Examples of using subdivision and smoothing with smooth interpolatory constraints as a design tool. All the surfaces have been obtained by applying two levels of subdivision and smoothing with various parameters to the skeleton surface of figure 5-A . Constrained vertices are marked with red dots. Surfaces are flat-shaded to enhance the faceting effect.

Figure 6-D shows the result of applying this process.

When more than one interpolatory constraint must be imposed, the problem is slightly more complicated. For simplicity, we will assume that the vertices have been reordered so that the interpolatory constraints are imposed on the first m vertices, i.e., $(x_C^N)_1 = x_1, \ldots, (x_C^N)_m = x_m$. We now look at the non-smooth signals $\delta_1, \ldots, \delta_m$, and at the corresponding faired signals, the first m columns of the matrix F. These signals are smooth, and so, any linear combination of them is also a smooth signal. Furthermore, since F is non-singular and diagonally dominated, these signals are linearly independent, and there exists a linear combination of them that satisfies the m desired constraints. Explicitly, the constrained smooth signal can be computed as follows

$$x_C^N = x^N + F_{nm}\, F_{mm}^{-1} \begin{pmatrix} x_1 - x_1^N \\ \vdots \\ x_m - x_m^N \end{pmatrix} , \qquad (10)$$

where F_{rs} denotes the sub-matrix of F determined by the first r rows and the first s columns. Figure 7 shows examples of surfaces constructed using subdivision and smoothing steps and interpolating some vertices of the skeleton using this method. The parameter of the fairing algorithm have been modified to achieve different effects, including shrinkage.

To minimize storage requirements, particularly when n is large, and assuming that m is much smaller than n, the computation can be structured as follows. The fairing algorithm is applied to δ_1 obtaining the first column $F\delta_1$ of the matrix F. The first m elements of this vector are stored as the first column of the matrix F_{mm}. The remaining $m - n$ elements of $F\delta_1$ are discarded. The same process is repeated for $\delta_2, \ldots, \delta_m$, obtaining the remaining columns of F_{mm}. Then the following linear system

$$F_{mm} \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} x_1 - x_1^N \\ \vdots \\ x_m - x_m^N \end{pmatrix}$$

is solved. The matrix F_{mm} is no longer needed. Then the remaining components of the signal y are set to zero $y_{m+1} = \cdots = y_n = 0$. Now the fairing algorithm is applied to the signal y. The result is the smooth deformation that makes the unconstrained smooth signal x^N satisfy the constraints

$$x_C^N = x^N + F\, y \; .$$

4.3 SMOOTH DEFORMATIONS

Note that in the constrained fairing algorithm described above the fact that the values of the signal at the vertices of interest are constrained to remain constant can be trivially generalized to allow for arbitrary smooth deformations of a surface. To do so, the values $x_1, \ldots, x_m$ in equation (10) must be replaced by the desired final values of the faired signal at the corresponding vertices. As in in the Free-form deformation approaches of Hsu, Hughes, and Kaufman [14] and Borrel [3], instead of moving control points outside the surface, surfaces can be deformed here by pulling one or more vertices.

Also note that the scope of the deformation can be controlled by changing the number of smoothing steps applied while smoothing the signals $\delta_1, \ldots, \delta_n$. To make the resulting signal satisfy the constraint, the value of N in the definition of the matrix F must be the one used to smooth the deformations. We have observed that good results are obtained when the number of iterations used to smooth the deformations is about five times the number used to fair the original shape. The examples in figure 7 have been generated in this way.

4.4 HIERARCHICAL CONSTRAINTS

This is another application of non-symmetric neighborhoods. We start by assigning a numeric label l_i to each vertex of the surface. Then we define the neighborhood structure as follows. We make vertex v_j a neighbor of vertex v_i if v_i and v_j share an edge (or face), *and* if $l_i \leq l_j$. Note that if v_j is a neighbor of v_i and $l_i < l_j$, then v_i is not a neighbor of v_j. The symmetry applies only to vertices with the same label. For example, if we assign label $l_i = 1$ to all the boundary vertices of a surface with boundary, and label $l_i = 0$ to all the internal vertices, then the boundary is faired as a curve, independently of the interior vertices, but the interior vertices follow the boundary vertices. If we also assign label $l_i = 1$ to a closed curve composed of internal edges of the surface, then the resulting surface will be smooth *along*, and on both sides of the curve, but not necessarily *across* the curve. Figure 8-D shows examples of subdivision surface designed using this procedure. If we also assign label $l_i = 2$ to some isolated points along the curves, then those vertices will in fact not move, because they will have empty neighborhoods.

4.5 TANGENT PLANE CONSTRAINTS

Although the normal vector to a polyhedral surface is not defined at a vertex, it is customary to define it by averaging some local information, say for shading purposes. When the signal x in equation (6) is replaced by the coordinates of the vertices, the Laplacian becomes a vector

$$\Delta v_i = \sum_{j \in i^\star} w_{ij} \, (v_j - v_i) \; .$$

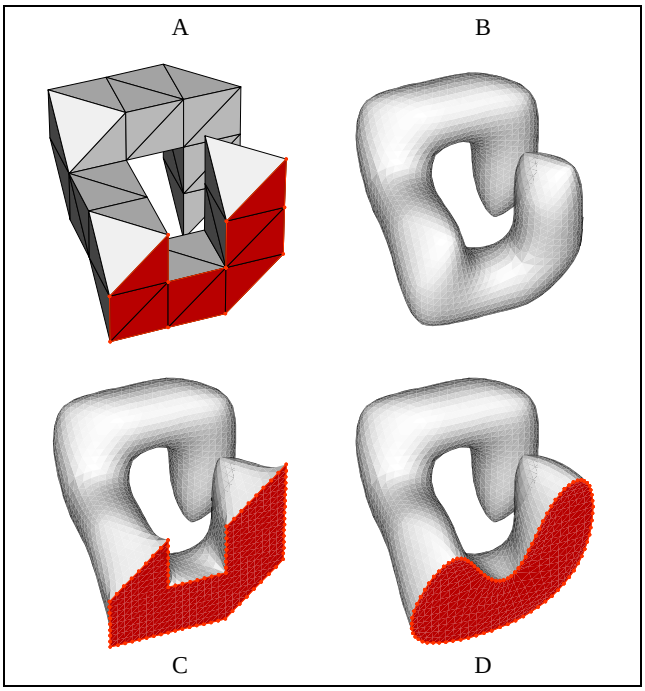


Figure 8: (A) Skeleton with marked vertices. (B) Surface (A) after three levels of subdivision and smoothing without constraints. (C) Same as (B) but with empty neighborhoods of marked vertices. (D) Same as (B) but with hierarchical neighborhoods, where marked vertices have label 1 and unmarked vertices have label 0. Surfaces are flat-shaded to enhance the faceting effect.

This vector average can be seen as a discrete approximation of the following curvilinear integral

$$\frac{1}{|\gamma|}\int_{v\in\gamma}(v-v_i)\,dl(v)\ ,$$

where γ is a closed curve embedded in the surface which encircles the vertex v_i, and $|\gamma|$ is the length of the curve. It is known that, for a curvature continuous surface, if the curve γ is let to shrink to to the point v_i, the integral converges to the mean curvature $\bar{\kappa}(v_i)$ of the surface at the point v_i times the normal vector N_i at the same point [7]

$$\lim_{\epsilon\to 0}\frac{1}{|\gamma_\epsilon|}\int_{v\in\gamma_\epsilon}(v-v_i)\,dl(v)=\bar{\kappa}(v_i)N_i\ .$$

Because of this fact, we can define the vector Δv_i as the normal vector to the polyhedral surface at v_i. If N_i is the desired normal direction at vertex v_i after the fairing process, and S_i and T_i are two linearly independent vectors tangent to N_i, The surface after N iterations of the fairing algorithm will satisfy the desired normal constraint at the vertex v_i it the following two linear constraints

$$S_i^t\Delta v_i^N=T_i^t\Delta v_i^N=0$$

are satisfied. This leads us to the problem of fairing with general linear constraints.

4.6 GENERAL LINEAR CONSTRAINTS

We consider here the problem of fairing a discrete surface signal x under general linear constraints $Cx_C^N=c$, where C is a $m\times n$ matrix of rank m (m independent constraints), and $c=(c_1,\ldots,c_m)^t$ is a vector. The method described in section 4.1 to impose smooth interpolatory constraints, is a particular case of this problem, where the matrix C is equal the upper m rows of the $m\times m$ identity matrix. Our approach is to reduce the general case to this particular case.

We start by decomposing the matrix C into two blocks. A first $m\times m$ block denoted $C_{(1)}$, composed of m columns of C, and a second block denoted $C_{(2)}$, composed of the remaining columns. The columns that constitute $C_{(1)}$ must be chosen so that $C_{(1)}$ become non-singular, and as well conditioned as possible. In practice this can be done using Gauss elimination with full pivoting [9], but for the sake of simplicity, we will assume here that $C_{(1)}$ is composed of the first m columns of C. We decompose signals in the same way. $x_{(1)}$ denotes here the first m components, and $x_{(2)}$ the last $n-m$ components, of the signal x. We now define a change of basis in the vector space of discrete surface signals as follows

$$\begin{cases} x_{(1)} &= y_{(1)}-C_{(1)}^{-1}C_{(2)}\,y_{(2)} \\ x_{(2)} &= y_{(2)} \end{cases}\ .$$

If we apply this change of basis to the constraint equation $C_{(1)}x_{(1)}+C_{(2)}x_{(2)}=c$, we obtain $C_{(1)}y_{(1)}=c$, or equivalently

$$y_{(1)}=C_{(1)}^{-1}c\ ,$$

which is the problem solved in section 4.2.

5 CONCLUSIONS

We have presented a new approach to polyhedral surface fairing based on signal processing ideas, we have demonstrated how to use it as an interactive surface design tool. In our view, this new approach represents a significant improvement over the existing fairness-norm optimization approaches, because of the linear time and space complexity of the resulting fairing algorithm.

Our current implementation of these ideas is a surface modeler that runs at interactive speeds on a IBM RS/6000 class workstation under X-Windows. In this surface modeler we have integrated all the techniques described in this paper and many other popular polyhedral surface manipulation techniques. Among other things, the user can interactively define neighborhood structures, select vertices or edges to impose constraints, subdivide the surfaces, and apply the fairing algorithm with different parameter values. All the illustrations of this paper where generated with this software.

In terms of future work, we plan to investigate how this approach can be extended to provide alternatives solutions for other important graphics and modeling problems that are usually formulated as variational problems, such as surface reconstruction or surface fitting problems solved with physics-based deformable models.

Some related papers [31, 32] can be retrieved from the IBM web server (`http://www.watson.ibm.com:8080`).

REFERENCES

[1] C.L. Bajaj and Ihm. Smoothing polyhedra using implicit algebraic splines. *Computer Graphics*, pages 79–88, July 1992. (Proceedings SIGGRAPH'92).

[2] H. Baker. Building surfaces of evolution: The weaving wall. *International Journal of Computer Vision*, 3:51–71, 1989.

[3] P. Borrel. Simple constrained deformations for geometric modeling and interactive design. *ACM Transactions on Graphics*, 13(2):137–155, April 1994.

[4] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10:350–355, 1978.

[5] G. Celniker and D. Gossard. Deformable curve and surface finite-elements for free-form shape design. *Computer Graphics*, pages 257–266, July 1991. (Proceedings SIGGRAPH'91).

[6] T.D. DeRose, M. Lounsbery, and J. Warren. Multiresolution analysis for surfaces of arbitrary topological type. Technical Report 93-10-05, Department of Computer Science and Enginnering, University of Washington, Seattle, 1993.

[7] M. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.

[8] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10:356–360, 1978.

[9] G. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins University Press, 2nd. edition, 1989.

[10] G. Greiner and H.P. Seidel. Modeling with triangular b-splines. *IEEE Computer Graphics and Applications*, 14(2):56–60, March 1994.

[11] A. Guéziec and R. Hummel. The wrapper algorithm: Surface extraction and simplification. In *IEEE Workshop on Biomedical Image Analysis*, pages 204–213, Seattle, WA, June 24–25 1994.

[12] M. Halstead, M. Kass, and T. DeRose. Efficient, fair interpolation using catmull-clark surface. *Computer Graphics*, pages 35–44, August 1993. (Proceedings SIGGRAPH'93).

[13] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *Computer Graphics*, pages 19–26, August 1993. (Proceedings SIGGRAPH'93).

[14] W.M. Hsu, J.F. Hughes, and H. Kaufman. Direct manipulation of free-form deformations. *Computer Graphics*, pages 177–184, July 1992. (Proceedings SIGGRAPH'92).

[15] A.D. Kalvin. *Segmentation and Surface-Based Modeling of Objects in Three-Dimensional Biomedical Images*. PhD thesis, New York University, New York, March 1991.

[16] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[17] T. Lindeberg. Scale-space for discrete signals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3):234–254, March 1990.

[18] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Dept. of Mathematics, University of Utah, August 1987.

[19] C. Loop. A G^1 triangular spline surface of arbitrary topological type. *Computer Aided Geometric Design*, 11:303–330, 1994.

[20] C. Loop. Smooth spline surfaces over irregular meshes. *Computer Graphics*, pages 303–310, July 1994. (Proceedings SIGGRAPH'94).

[21] W. Lorenson and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, pages 163–169, July 1987. (Proceedings SIGGRAPH).

[22] M. Lounsbery, S. Mann, and T. DeRose. Parametric surface interpolation. *IEEE Computer Graphics and Applications*, 12(5):45–52, September 1992.

[23] J. Menon. Constructive shell representations for freeform surfaces and solids. *IEEE Computer Graphics and Applications*, 14(2):24–36, March 1994.

[24] H.P. Moreton and C.H. Séquin. Functional optimization for fair surface design. *Computer Graphics*, pages 167–176, July 1992. (Proceedings SIGGRAPH'92).

[25] J. Oliensis. Local reproducible smoothing without shrinkage. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(3):307–312, March 1993.

[26] A. Pentland and S. Sclaroff. Closed-form solutions for physically based shape modeling and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):715–729, July 1991.

[27] E. Seneta. *Non-Negative Matrices, An Introduction to Theory and Applications*. John Wiley & Sons, New York, 1973.

[28] L.A. Shirman and C.H. Séquin. Local surface interpolation with bezier patches. *Computer Aided Geometric Design*, 4:279–295, 1987.

[29] W.J. Shroeder, A. Zarge, and W.E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, pages 65–70, 1992. (Proceedings SIGGRAPH'92).

[30] R.S. Szeliski, D. Tonnesen, and D. Terzopoulos. Modeling surfaces of arbitrary topology with dynamic particles. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–87, New York, NY, June 15–17 1993.

[31] G. Taubin. Curve and surface smoothing without shrinkage. Technical Report RC-19536, IBM Research, April 1994. (also in Proceedings ICCV'95).

[32] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. Technical Report RC-19860, IBM Research, December 1994. (also in Proceedings ICCV'95).

[33] G. Taubin, T. Zhang, and G. Golub. Optimal polyhedral surface smoothing as filter design. (in preparation).

[34] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4:306–311, 1988.

[35] G. Turk. Re-tiling polygonal surfaces. *Computer Graphics*, pages 55–64, July 1992. (Proceedings SIGGRAPH'92).

[36] G. Turk and M. Levoy. Zippered polygon meshes from range data. *Computer Graphics*, pages 311–318, July 1994. (Proceedings SIGGRAPH'94).

[37] W. Welch and A. Witkin. Variational surface modeling. *Computer Graphics*, pages 157–166, July 1992. (Proceedings SIGGRAPH'92).

[38] W. Welch and A. Witkin. Free-form shape design using triangulated surfaces. *Computer Graphics*, pages 247–256, July 1994. (Proceedings SIGGRAPH'94).

[39] A.P. Witkin. Scale-space filtering. In *Proceedings, 8th. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1019–1022, Karlsruhe, Germany, August 1983.

[40] C.T. Zahn and R.Z. Roskies. Fourier descriptors for plane closed curves. *IEEE Transactions on Computers*, 21(3):269–281, March 1972.

APPENDIX

We first analyze those cases where the matrix W can be factorized as a product of a symmetric matrix E times a diagonal matrix D. Such is the case for the first order neighborhood of a shape with equal weights $w_{ij} = 1/|i^*|$ in each neighborhood i^*. In this case E is the matrix whose ij-th. element is equal to 1 if vertices v_i and v_j are neighbors, and 0 otherwise, and D is the diagonal matrix whose i-th. diagonal element is $1/|i^*|$. Since in this case W is a *normal matrix* [9], because $D^{1/2} W D^{-1/2} = D^{1/2} E D^{1/2}$ is symmetric, W has all real eigenvalues, and sets of n left and right eigenvectors that form respective bases of n-dimensional space. Furthermore, by construction, W is also a *stochastic matrix*, a matrix with nonnegative elements and rows that add up to one [27]. The eigenvalues of a stochastic matrix are bounded above in magnitude by 1, which is the largest magnitude eigenvalue. It follows that the eigenvalues of the matrix K are real, bounded below by 0, and above by 2. Let $0 \leq k_1 \leq k_2 \leq \cdots \leq k_n \leq 2$ be the eigenvalues of the matrix K, and let $u_1, u_2, \ldots, u_n$ a set of linearly independent unit length right eigenvectors associated with them.

When the neighborhood structure is not symmetric, the eigenvalues and eigenvectors of W might not be real, but as long as the eigenvalues are not repeated, the decomposition of equation (3), and the analysis that follows, are still valid. However, the behavior of our fairing algorithm in this case will depend on the distribution of eigenvalues in the complex plane. The matrix W is still stochastic here, and so all the eigenvalues lie on a unit circle $|k_i - 1| < 1$. If all the eigenvalues of W are very close to the real line, the behavior of the fairing algorithm should be essentially the same as in the symmetric case. This seems to be the case when very few neighborhoods are made non-symmetric. But in general, the problem has to be analyzed on a case by case basis.

Optimal Surface Smoothing as Filter Design,
by G. Taubin, T. Zhang, and G. Golub,
IBM Technical Report RC-20404, March 1996; and
Fourth European Conference on Computer Vision
(ECCV'96).

RC-20404(#90237) 3/12/96
Computer Sciences 22 pages

Research Report

OPTIMAL SURFACE SMOOTHING AS FILTER DESIGN

Gabriel Taubin
IBM T.J.Watson Research Center
P.O.Box 704, Yorktown Heights, NY 10598
email:taubin@watson.ibm.com
.
Tong Zhang and *Gene Golub*
Computer Science Department
Stanford University
Stanford, CA 94305
email:{tzhang,golub}@cs.stanford.edu

Research Division
Yorktown Heights, New York • San Jose, California • Zurich, Switzerland

OPTIMAL SURFACE SMOOTHING AS FILTER DESIGN

Gabriel Taubin
IBM T.J.Watson Research Center
P.O.Box 704, Yorktown Heights, NY 10598
email:taubin@watson.ibm.com
.
Tong Zhang and *Gene Golub*
Computer Science Department
Stanford University
Stanford, CA 94305
email:{tzhang,golub}@cs.stanford.edu

ABSTRACT:

For a number of computational purposes, including visualization, smooth surfaces are approximated by polyhedral surfaces. An inherent problem of these approximation algorithms is that the resulting polyhedral surfaces appear faceted. A signal processing approach to smoothing polyhedral surfaces was recently introduced [10, 11]. Within this framework surface smoothing corresponds to low-pass filtering. In this paper we look at the filter design problem in more detail. We analyze the stability properties of the low-pass filter described in [10, 11], and show how to minimize its running time. Then we show that most classical techniques used to design finite impulse response (FIR) digital filters can also be used to design significantly faster smoothing filters. Finally, we describe an algorithm to estimate the power spectrum of a signal, and use it to evaluate the performance of the different filter design techniques described in the paper.

1. Introduction

The signal processing framework introduced in [10, 11], extends Fourier analysis to *discrete surface signals*, functions defined on the vertices of polyhedral surfaces. As in the method of Fourier Descriptors [12], where a closed curve is smoothed by truncating the Fourier series of its coordinate signals, a very large polyhedral surface of arbitrary topology is smoothed here by low-pass filtering its three surface coordinate signals. And although the formulation was developed mainly for signals defined on surfaces, it is in fact valid for *discrete graph signals*, functions defined on the vertices of directed graphs. Since this general formulation provides a unified treatment of polygonal curves, polyhedral surfaces, and even three-dimensional finite elements meshes, we start this paper by reviewing this formulation in its full generality.

Then we look at the filter design problem in more detail, with the main goal of minimizing the execution time of the low-pass filtering algorithm, given a desired frequency response specification. But we also take into consideration numerical issues, such as stability. We first study the tradeoffs that exists between minimizing execution time and maintaining the filter stable for the low-pass filter design of [10, 11]. Then we show that most classical finite impulse response (FIR) digital filter design techniques can be applied, with minor or no modifications in most cases, to the design of discrete graph signal filters. FIR filters, which in this framework correspond to sparse matrix multiplication, yield acceptable linear time and space complexity algorithms. Five to ten-fold speedups with respect to the low-pass filter design of [10, 11] can easily be obtained.

Then, we compare the performance of the different filter design methodologies with an algorithm to estimate the power spectrum of a discrete graph signal. This power spectrum estimator is implemented as a bank of high order band-pass filters, designed with the same techniques as the surface low-pass smoothing filters. However, the goal here is to design very sharp band-pass filters, not necessarily to minimize the order of the filter. We also use the power spectrum estimator to determine the pass-band frequency of the filter in such a way that shrinkage is prevented.

We end the paper with some experimental results and our conclusions.

2. Fourier Analysis of Discrete Graph Signals

In this section we describe the signal processing formulation of [10, 11] in its most general form, i.e., for *discrete graph signals*, functions defined on directed graphs. We represent a *directed graph* on the set $\{1, \ldots, n\}$ of n nodes as a set of *neighborhoods* $\{i^\star : i = 1, \ldots, n\}$, where $i^\star$ is a subset of nodes that we call the neighborhood of node i. If j is an element of $i^\star$ we say that j is a *neighbor* of i, and we visualize it as an arrow from i to j. In principle, except for prohibiting a node from being a neighbor of itself, we do not impose any other constraint on the neighborhoods. Note that j is allowed to be a neighbor of i without requiring i to be a neighbor of j, and neighborhoods can also be empty. We call a vector $x = (x_1, \ldots, x_n)^t$, with one component per node of the graph, a *discrete graph signal*.

We represent a polyhedral surface as a pair of arrays $S = \{V, F\}$, an array of n vertices V, and an array of faces F. A vertex is a three-dimensional vector of real coordinates, and a face is a sequence of non-repeated indices of vertices representing a closed three-dimensional polygon. *Triangulated*

surfaces are the most common, where all faces are triangles. We look at a polyhedral surface of n vertices as a directed graph, by labeling the vertices with distinct node numbers ranging from 1 to n, and defining a neighborhood for each node. We normally use *first order neighborhoods*, were node j is a neighbor of node i if i and j share an edge (or face), but other neighborhood structures can be used for other purposes, such as to impose certain types of constraints [11]. A *discrete surface signal* is a discrete graph signal defined on the associated graph. We visualize a discrete surface signal defined on a polyhedral surface as a piece-wise linear function defined on the surface. Discrete surface signals defined on polygonal curves, and on simplicial complexes of higher dimension, can be interpreted in a similar way.

The Fourier transform of a discrete graph signal cannot be defined in the traditional way because there is no notion of convolution. However, there is an alternative definition that can be generalized. Computing the Discrete Fourier Transform (DFT) of a signal defined on a closed polygon of n vertices is equivalent to decomposing the signal as a linear combination of the eigenvectors of the Laplacian operator

$$\Delta x_i = \frac{1}{2}(x_{i-1} - x_i) + \frac{1}{2}(x_{i+1} - x_i) \ , \tag{2.1}$$

were the Fourier transform is the vector of coefficients of the sum. To define the Fourier transform of a signal defined on an arbitrary directed graph we only have to define a linear operator that we will call the Laplacian operator. This is the same idea behind the method of eigenfunctions of Mathematical Physics [1].

We define the *Laplacian* of a discrete graph signal x by the formula

$$\Delta x_i = \sum_{j \in i^\star} w_{ij} \, (x_j - x_i) \tag{2.2}$$

where the weights w_{ij} are positive numbers that add up to one for each vertex

$$\sum_{j \in i^\star} w_{ij} = 1 \ .$$

These weights can be chosen in many different ways taking into consideration the neighborhoods, but in this paper we will assume that they are not functions of the signal x. Otherwise, the resulting operator is non-linear, and so, beyond the scope of this paper. One particularly simple choice that produces good results is to set w_{ij} equal to the inverse of the number of neighbors $1/|i^\star|$ of node i, for each element j of $i^\star$. Other choices of weights are discussed in [10, 11]. Note that the Laplacian of a signal defined on a closed polygon, described in equation (2.1), is a particular case of these definitions, with $w_{ij} = 1/2$, for $j \in i^\star = \{i-1, i+1\}$, for each node i.

If $W = (w_{ij})$ denotes the matrix of weights, with $w_{ij} = 0$ when j is not a neighbor of i, and $K = I - W$, the Laplacian of a discrete signal can be written in matrix form as

$$\Delta x = -K x \ . \tag{2.3}$$

Although the method applies to general neighborhood structures, in this paper we will restrict our analysis to those cases where the matrix W can be factorized as a product of a symmetric matrix times a diagonal matrix $W = ED$. In this case the matrix W is a *normal matrix* [4], because the matrix

$$D^{1/2} W D^{-1/2} = D^{1/2} E D^{1/2} \tag{2.4}$$

is symmetric. Note that such is the case for the first order neighborhoods of a surface with equal weights $w_{ij} = 1/|i^\star|$ in each neighborhood $i^\star$, where E is the *incidence matrix* of the neighborhood structure (a symmetric matrix for first order neighborhoods), the matrix whose ij-th. element is equal to 1 if the nodes i and j are neighbors, and 0 otherwise; and D is the diagonal positive definite matrix whose i-th. diagonal element is $1/|i^\star|$. When W is a normal matrix it has all real eigenvalues, and sets of n left and right eigenvectors that form dual bases of n-dimensional space. Furthermore, by construction, W is also a *stochastic matrix*, a matrix with nonnegative elements and rows that add up to one [9]. The eigenvalues of a stochastic matrix are bounded above in magnitude by 1. It follows that the eigenvalues of the matrix K are real, bounded below by 0, and above by 2. Seen as discrete graph signals, the right eigenvectors of the matrix K can be considered as the *natural vibration modes*, and the corresponding eigenvalues as the associated *natural frequencies*. In our case, a vibration mode of high natural frequency corresponds to a rapid oscillation in the space domain. For example, for any directed graph, the constant signal $(1, \ldots, 1)$ is an eigenvector of K associated with the frequency $k = 0$, and the values of a natural vibration mode associated with a low natural frequency varies slowly when we move from a vertex to a neighbor vertex.

In the simple cases of signals defined on regular polygons, or more generally on graphs with group structure [3], the eigenvectors and eigenvalues of K have analytic expressions. The Fast Fourier Transform algorithm for signals defined on closed polygons is a good example of how this structure can be exploited. However, for the typical large graphs that we are interested in processing here, there are no analytic expressions for the eigenvalues and eigenvectors of K. And although a few extremal eigenvalues and eigenvectors of K can be computed with the Lanczos method [4], it is numerically impossible to reliably compute all of them. However, and this is the most significant observation, for filtering operations it is not necessary to compute the eigenvectors explicitly.

If $0 \leq k_1 \leq \cdots \leq k_n \leq 2$ are the eigenvalues of K, $e_1, \ldots, e_n$ a set of corresponding right eigenvectors, and $\delta_1, \ldots, \delta_n$ the associated dual basis of $e_1, \ldots, e_n$, the identity matrix I, and the matrix K can be written as follows

$$I = \sum_{i=1}^{n} e_i \delta_i^t \qquad K = \sum_{i=1}^{n} k_i \, e_i \delta_i^t \, ,$$

and every discrete graph signal x has a unique decomposition as a linear combination of $e_1, \ldots, e_n$

$$x = I \, x = \sum_{i=1}^{n} \hat{x}_i \, e_i \, , \tag{2.5}$$

where $\hat{x}_i = \delta_i^t x$. We call the vector $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_n)^t$ the Discrete Fourier Transform (DFT) of x, generalizing the classical definition for signals defined on closed polygons. Note, however, that this definition does not identify a unique object yet. If a different set of right eigenvectors of K is chosen, a different DFT is obtained. To complete the definition, if $W = ED$ with E symmetric, and D diagonal, The formula $\langle x, y \rangle_D = x^t D y$ defines an inner product in our space of signals, and normalizing the right eigenvectors of K to unit length with respect to the associated norm is equivalent to orthonormalizing them with respect to the inner product, and Parseval's formula is satisfied

$$\|x\|_D^2 = \|\hat{x}\|^2 \, , \tag{2.6}$$

where the norm on the right hand side is the Euclidean norm. That is, the frequency components $\hat{x}_i e_i$ of the signal x are orthogonal with respect to the inner product defined by D. We will assume from now on that the right eigenvectors of K are normalized in this fashion. These results will be used in sections 7 and 8.

To filter the signal x is to change its frequency distribution according to a transfer function $f(k)$

$$x' = \sum_{i=1}^{n} f(k_i)\, \hat{x}_i e_i = \left(\sum_{i=1}^{n} f(k_i)\, e_i \delta_i^t \right) x \ . \tag{2.7}$$

The frequency component of x corresponding the the natural frequency k_i is enhanced or attenuated by a factor $f(k_i)$. For example, the transfer function of an ideal low-pass filter, illustrated in figure 1, is

$$f_{\text{LP}} = \begin{cases} 1 & \text{for } 0 \leq k \leq k_{\text{PB}} \\ 0 & \text{for } k_{\text{PB}} < k \leq 2 \end{cases} \ , \tag{2.8}$$

where k_{PB} is the *pass-band frequency*. In this case, all the frequencies above the pass-band frequencies

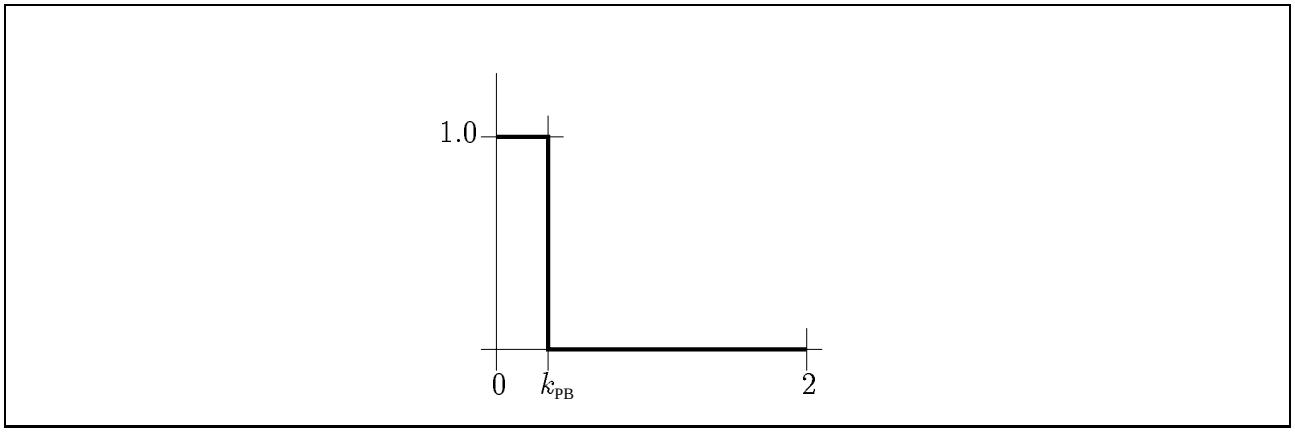


Figure 1: Graph of the ideal low-pass filter f_{LP}.

are removed, leaving only the low frequency components. The method of Fourier Descriptors [12] consists in filtering a discrete graph signal with an ideal low-pass filter transfer function. An efficient algorithm ($O(n \log(n))$) to ideal low-pass filter a signal defined on a closed polygon can be implemented using the Fast Fourier Transform algorithm. But in the general case of discrete graph signals, there is no efficient numerical method to compute its DFT, particularly when the number of nodes of the graph is very large. The computation can only be performed approximately, which is the main subject of this paper. To do this the ideal low-pass filter transfer function is replaced by an analytic approximation, usually a polynomial or rational function, for which the computation can be performed in an efficient manner. A wide range of analytic functions of one variable $f(k)$ can be evaluated in a matrix such as K [4]. The result is another matrix $f(K)$ with the same left and right eigenvectors, but with eigenvalues $f(k_1), \ldots, f(k_n)$

$$f(K) = \sum_{i=1}^{n} f(k_i)\, e_i \delta_i^t \ .$$

The main reason why the filtering operation $x' = f(K)\,x$ of equation (2.7) can be performed efficiently for a polynomial transfer function of low degree, is that when K is sparse, which is the case here, the matrix $f(K)$ is also sparse (but of wider bandwidth), and so, the filtering operation becomes the multiplication of a vector by a sparse matrix.

In Gaussian smoothing the transfer function is the polynomial $f_N(k) = (1-\lambda k)^N$, with $0 < \lambda < 1$. But this transfer function produces shrinkage

$$\lim_{N\to\infty}(1-\lambda k)^N = \begin{cases} 1 & \text{for } k = 0 \\ 0 & \text{for } 0 < k \leq 2\ . \end{cases}$$

That is, as N grows, the shape asymptotically converges to its centroid.

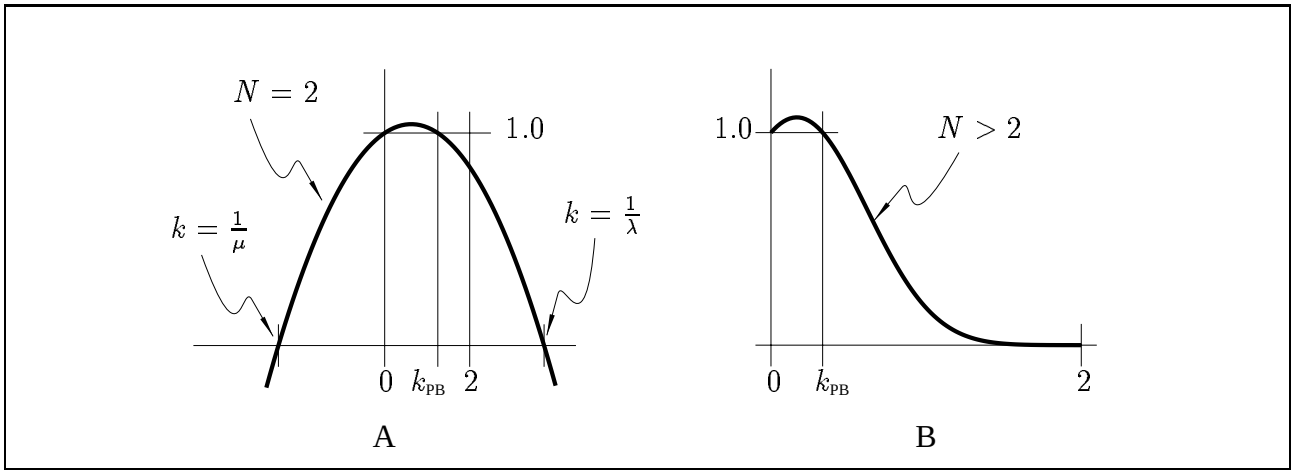


Figure 2: Graph of transfer function $f(k) = ((1-\mu k)(1-\lambda k))^{N/2}$. (A) $N = 2$. (B) $N > 2$. (out of scale)

The algorithm introduced in [10, 11] is escentially Gaussian smoothing with the difference that the scale factor λ changes from iteration to iteration, alternating between a positive value λ and a negative value μ. This simple modification still produces smoothing, but prevents shrinkage. The transfer function is the polynomial $f_N(k) = ((1-\lambda k)(1-\mu k))^{N/2}$, with $0 < \lambda < -\mu$ and N even, illustrated in figure 2-A for $N = 2$, and in 2-B for $N > 2$. This displays a typical *low-pass filter* shape in the region of interest, from $k = 0$ to $k = 2$. The *pass-band frequency* of this filter is defined as the unique value of k in the interval $(0, 2)$ such that $f_N(k) = 1$. Such a value exists when $0 < \lambda < -\mu$, and turns out to be equal to $k_{\text{PB}} = 1/\lambda + 1/\mu$. This polynomial transfer function of degree N results in a linear time and space complexity algorithm, which is very simple to implement, and produces smoothing without shrinkage. From now on we will refer to this algorithm as the $\lambda-\mu$ algorithm. However, as we will see below, faster algorithms can be achieved by choosing as transfer function a beter polynomial approximation of the same degree of the ideal low-pass filter.

3. Fast Smoothing as Filter Design

We are faced with the classical problem of digital filter design in signal processing [8, 5], but with some restrictions. Note that because of the linear complexity constraint discussed above, only

polynomial transfer functions (FIR filters) are allowed. With rational transfer functions (IIR filters) better approximations of the ideal low-pass filter could be achieved with lower degrees of polynomials, but in our context a rational transfer function $f(k) = g(k)/h(k)$ involves solving the sparse linear system $h(K)\,x' = g(K)\,x$, which is not a linear complexity operation. Because of this reason, we leave the study of rational transfer functions for the future.

Because of space restrictions, of all the traditional FIR filter design methods available in the signal processing literature, we only cover here in some detail the method of windows, which is the simplest one. With this method we can design filters which are significantly faster, or sharper, than those obtain with the $\lambda - \mu$ algorithm for the same degree.

4. Optimizing the $\lambda-\mu$ algorithm

The $\lambda-\mu$ algorithm can be described in a recursive fashion as follows

$$f_N(k) = \begin{cases} 1 & N = 0 \\ (1 - \lambda_N\, k)\, f_{N-1}(k) & N > 0 \end{cases}$$

where $\lambda_N = \lambda$, for N odd, and $\lambda_N = \mu$ for N even. Note that this algorithm requires minimum storage, only one array of dimension n to store the Laplacian of a signal if computed in place, and two arrays of dimension n in general. The algorithm is described in figure 3, where x is the input signal, and x' is the result of the filtering operation.

filter$(N, \lambda_1, \ldots, \lambda_N, K, x, x')$
  $x^0 = x$
  for $j = 1$ **to** N **step** 1 **do**
    $x^1 = Kx^0$
    $x^0 = x^0 - \lambda_j Kx^1$
  end
  $x' = x^0$
return

Figure 3: The $\lambda - \mu$ filtering algorithm.

To maintain the minimum storage property and the same simple algorithmic structure, one could try to generalize by changing the scale factors λ_N from iteration to iteration in a different way. But if we start with a given pass-band frequency $k_{\text{PB}} = 1/\lambda + 1/\mu$, as it is usually the case when one wants to *design* the filter, there are many values of λ and μ such that $0 < \lambda < -\mu$, that define a filter with the same pass-band frequency. In order for the polynomial $f(k) = (1 - \lambda k)(1 - \mu k)$ to define a low-pass filter in the interval $[0, 2]$ it is necessary that $|f(k)| < 1$ in the stop-band region, so that $f_N(k) = f(k)^N \to 0$ when N grows. Since $f(k_{\text{PB}}) = 1$ and $f(k)$ is strictly decreasing for $k > k_{\text{PB}}$,

this condition is equivalent to $f(2) > -1$, which translates into the following constraint on λ

$$\lambda < \frac{-k_{\text{PB}} + \sqrt{(2-k_{\text{PB}})^2 + 4}}{2(2-k_{\text{PB}})} \, . \tag{4.1}$$

Figure 4 shows examples of transfer functions of filters designed for the same pass-band frequency, but with different values of λ. As λ increases, the slope of the filter immediately after the pass-band frequency increases, i.e., the filter becomes sharper, but at the same time instability starts to develop at the other end of the spectrum, close to $k = 2$. If the maximum eigenvalue k_n of the matrix K is significantly less than 2 (which is not usually the case) we only need the filter to be stable in the interval $[0, k_n]$ (i.e., $1 > f(k_n) > -1$), and larger values of λ are acceptable. A good estimate of the maximum eigenvalue of K can be obtained with the Lanczos method [4]. Even if the maximum eigenvalue k_n is not known, the signal x to be smoothed might be band-limited, i.e., the coefficients $\hat{x}_i$ in equation (2.5) associated with high frequencies are all zero, or very close to zero. This condition might be difficult to determine in practice for a particular signal, but if we apply the algorithm with small λ for a certain number of iterations, the resulting signal becomes in effect band-limited. At this point λ can be increased keeping the pass-band frequency constant, maybe even making the filter unstable, and the algorithm can be applied again with the new values of λ and μ for more iterations. This process of increasing λ keeping the pass-band frequency constant can now be repeated again and again. A moderate speed-up is obtained in this way. Figure 5 show some examples of this process. All the filters in this figure produce almost the same response, but filter (F) is five times faster than filter (A).

5. Filter Design with Windows

The most straightforward approach to traditional digital filter design is to obtain a trigonometric polynomial approximation of the ideal filter transfer function by truncating its Fourier series. The resulting trigonometric polynomial minimizes the L_2 distance to the ideal filter transfer function among all the trigonometric polynomials of the same degree. Note that it is sufficient to know how to construct low-pass filters. A band-pass filter can be constructed as the difference of two low pass filters, and a high-pass filter can be constructed in a similar way. To obtain regular polynomials, not trigonometric ones, we first apply the change of variable $k = 2(1 - \cos(\theta))$. This change of variable is a $1 - 1$ mapping $[0, \pi/2] \to [0, 2]$. Then we extend the resulting function to the interval $[-\pi, \pi]$ as follows

$$h_{\text{LP}}(\theta) = \begin{cases} 0 & \pi/2 \leq \theta \leq \pi \\ f_{\text{LP}}(2(1 - \cos(\theta))) & 0 \leq \theta \leq \pi/2 \\ h(-\theta) & -\pi \leq \theta \leq 0 \, . \end{cases}$$

Note that this function, periodic of period 2π and even, is also an ideal low-pass filter as a function of θ

$$h_{\text{LP}}(\theta) = \begin{cases} 1 & \text{if } |\theta| < \theta_{\text{PB}} \\ 0 & \text{otherwise} \end{cases} \, ,$$

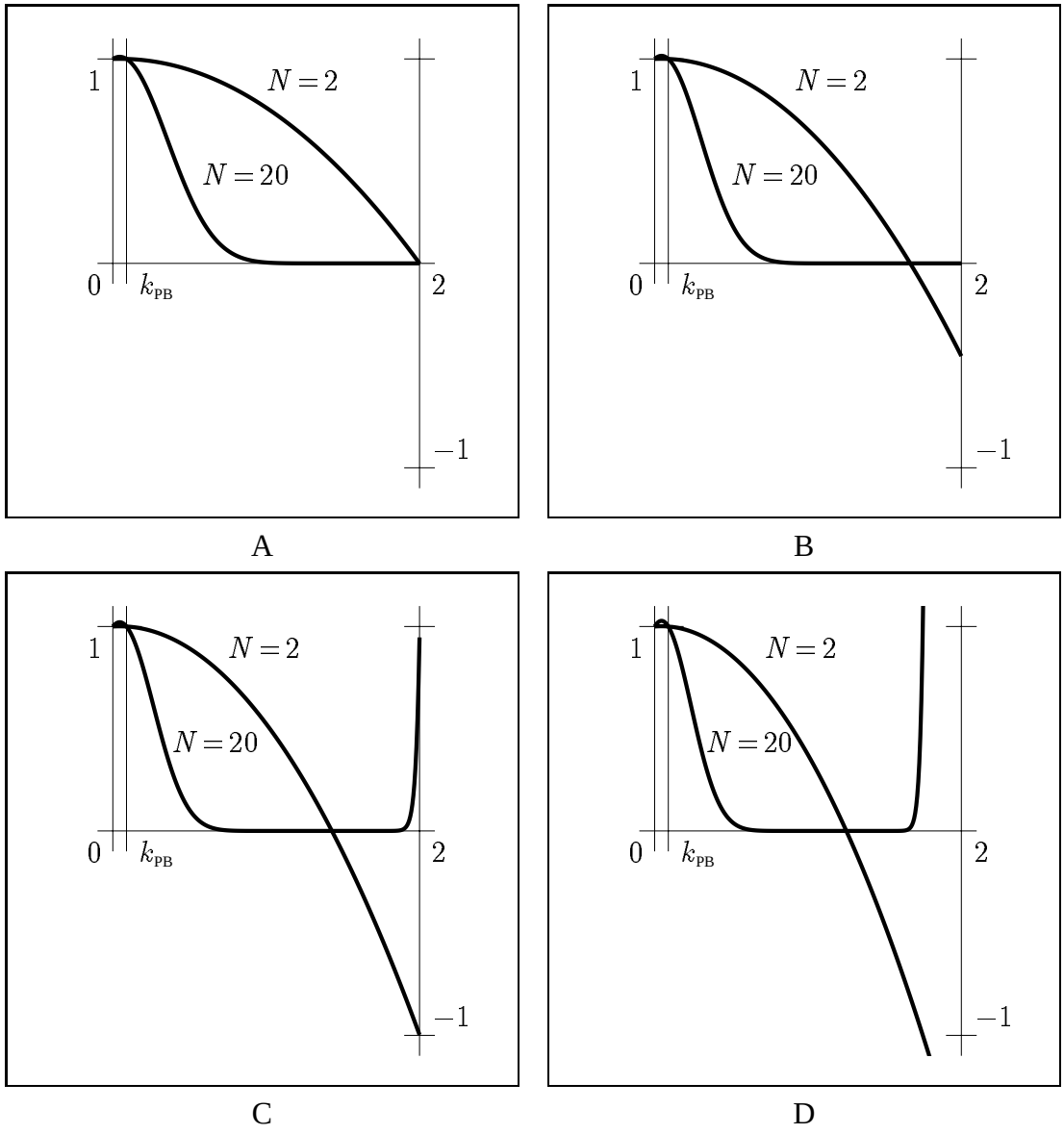


Figure 4: Graphs of transfer function $((1-\lambda k)(1-\mu k))^{N/2}$ for $N=2$ and $N=20$ and pass-band frequency $k_{\text{PB}} = 1/\lambda + 1/\mu = 0.09$. (A) $\lambda = 0.5$: stable. (B) $\lambda = 0.6$: stable. (C) $\lambda = 0.7$: limit case. (D) $\lambda = 0.8$: unstable.

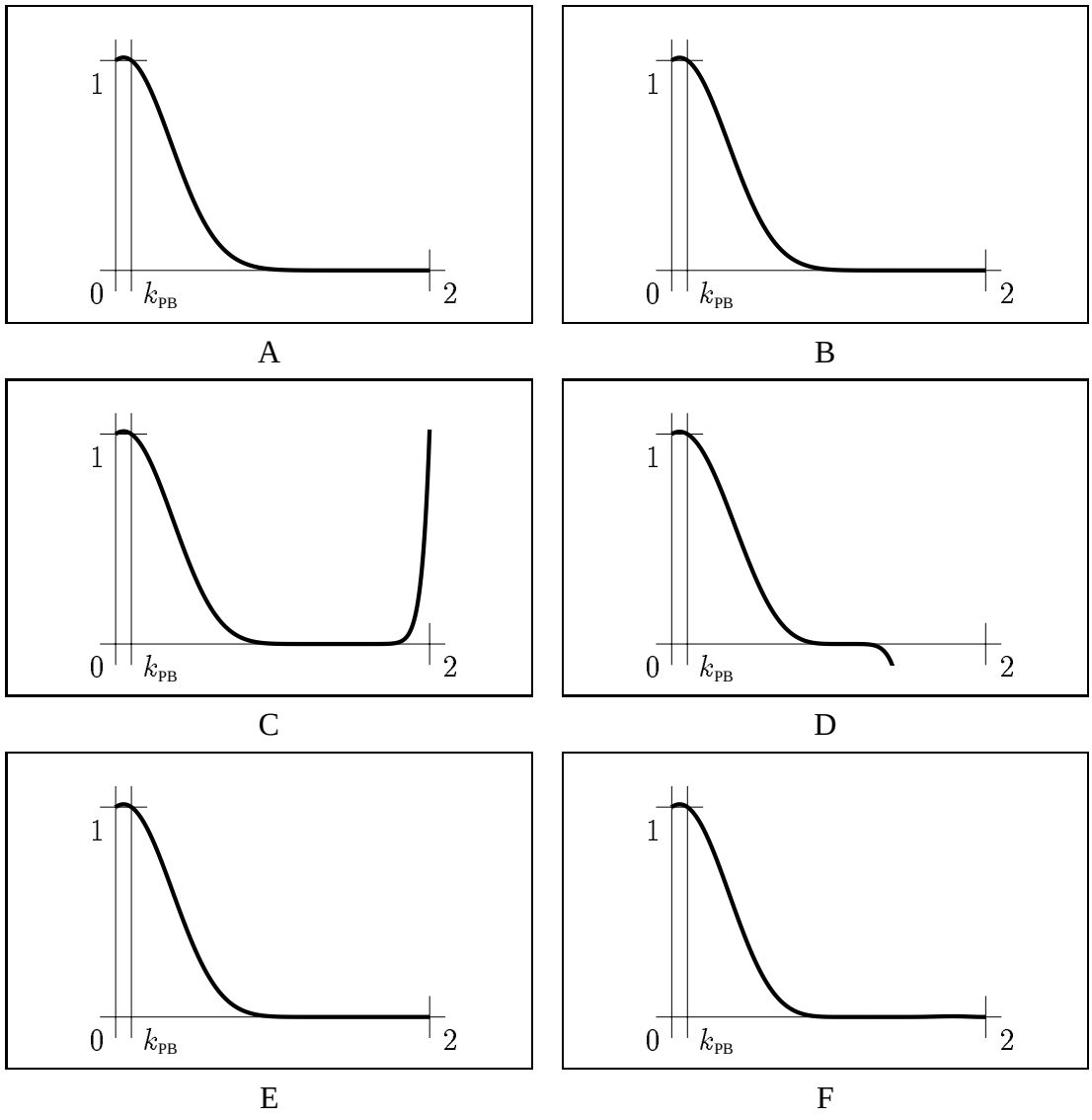


Figure 5: Different combinations of parameters in $((1-\lambda k)(1-\mu k))^{N/2}$ produce almost indistinguishable transfer functions. The pass-band frequency $k_{\text{PB}} = 1/\lambda + 1/\mu = 0.1$ is the same in the four cases. (A) $\lambda = 0.3$, $\mu = -0.3093$, $N = 120$. (B) $\lambda = 0.5$, $\mu = -0.5263$, $N = 40$. (C) $\lambda = 0.7$, $\mu = -0.7527$, $N = 20$. (D) $\lambda = 0.9$, $\mu = -0.9890$, $N = 12$. (E) $\lambda = 0.3$, $\mu = -0.3093$, $N = 12$, followed by $\lambda = 0.5$, $\mu = -0.5263$, $N = 12$, followed by $\lambda = 0.7$, $\mu = -0.7527$, $N = 12$. (F) $\lambda = 0.3$, $\mu = -0.3093$, $N = 6$, followed by $\lambda = 0.5$, $\mu = -0.5263$, $N = 6$, followed by $\lambda = 0.7$, $\mu = -0.7527$, $N = 6$, followed by $\lambda = 0.9$, $\mu = -0.9890$, $N = 6$. Note that (C) and (D) are unstable by themselves, but preceded by stable filters become stable. The degrees of the polynomials are (A) 120, (B) 40, (C) 20, (D) 12, (E) 36, and (F) 24.

where θ_{PB} is the unique solution of $k_{\text{PB}} = 2(1 - \cos(\theta_{\text{PB}}))$ in $[0, \pi/2]$. Since $h(\theta)$ is an even function, it has a Fourier series expansion in terms of cosines only

$$h_{\text{LP}}(\theta) = h_0 + 2 \sum_{n=0}^{\infty} h_n \cos(n\theta) \ ,$$

where h_n is

$$h_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} h(\theta) \cos(n\theta) \, d\theta = \begin{cases} \theta_{\text{PB}}/\pi & n = 0 \\ \sin(n\,\theta_{\text{PB}})/n\,\pi & n > 0 \end{cases} \ .$$

Now, it is well known that $\cos(n\,\theta) = T_n(\cos(\theta))$, where T_n is the n-th. Chebyshev polynomial [2], defined by the three term recursion

$$T_n(w) = \begin{cases} 1 & n = 0 \\ w & n = 1 \\ 2\,w\,T_{n-1}(w) - T_{n-2}(w) & n > 1 \end{cases}$$

The N-th. polynomial approximation of f_{LP} for $k \in [0, 2]$ is then

$$f_N(k) = \frac{\theta_{\text{PB}}}{\pi} T_0(1 - k/2) + \sum_{n=1}^{N} \frac{2 \sin(n\,\theta_{\text{PB}})}{n\,\pi} T_n(1 - k/2) \ . \tag{5.1}$$

Figure 6 shows some of these polynomials compared with the polynomials $((1 - \lambda k)(1 - \mu k))^{N/2}$ for the same pass-band frequency.

As can be easily observed in figure 6, direct truncation of the series leads to the well-known Gibbs phenomenon, i.e., a fixed percentage overshoot and ripple before and after the discontinuity. As it is shown in section 9, this is one of the problems that makes this technique unsatisfactory. The other problem is that the resulting polynomial approximation does not necesarily satify the constraint $f_N(0) = 1$, which is required to preserve the average value of the signal (DC level in classical signal processing, centroid in the case of surfaces). Our experiments show that a desirable surface smoothing filter transfer function should be as close as possible to 1 within the pass-band as possible, and then decrease to zero in the stop-band ($[k_{\text{PB}}, 2]$).

Another classical technique to control the convergence of the Fourier series is to use a weighting function to modify the Fourier coefficients. In our case the polynomial approximation of equation (5.1) is modified as follows

$$f_N(k) = w_0 \frac{\theta_{\text{PB}}}{\pi} T_0(1 - k/2) + w_n \sum_{n=1}^{N} \frac{2 \sin(n\,\theta_{\text{PB}})}{n\,\pi} T_n(1 - k/2) \ , \tag{5.2}$$

where $w_0, w_1, \ldots, w_N$ are the weights that constitute a so called *window*. Since the multiplication of Fourier coefficients by a window corresponds to convolving the original frequency response with the Fourier series defined by the window, a design criterion for windows is to find a finite window whose Fourier transform has relatively small side lobes. The polynomial approximation of equation (5.1) is a particular case of (5.2), where the weights are all equal to 1. This is called the *Rectangular window*.

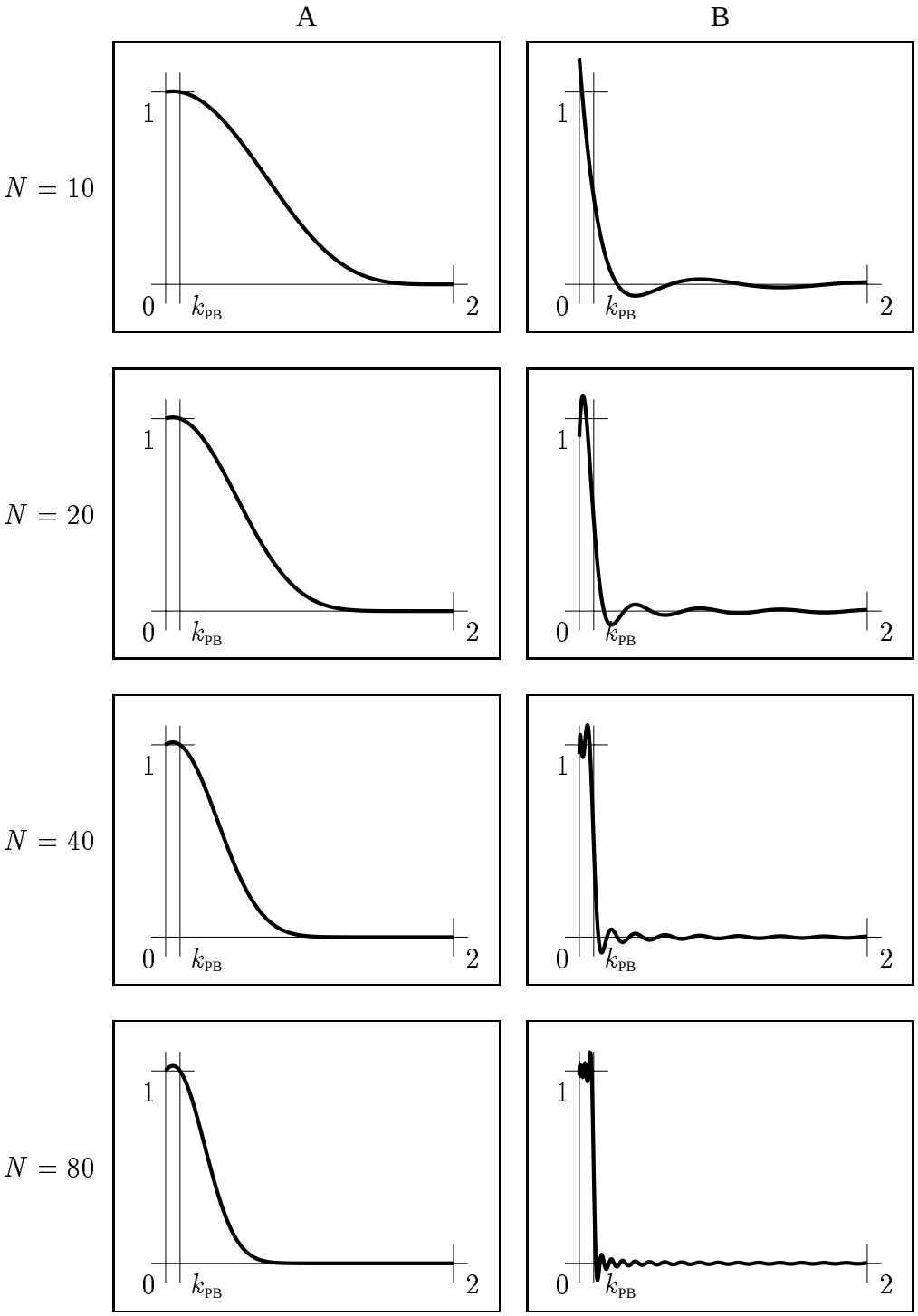


Figure 6: (A) Polynomial transfer function $f(k) = ((1 - \lambda k)(1 - \mu k))^{N/2}$ with $k_{\mathrm{PB}} = 0.1$ and $\lambda = 0.6307$. (B) Truncated Fourier series approximation of the ideal low-pass filter (Rectangular window).

Other popular windows are, the *Hanning window*, the *Hamming window*, and the *Blackman window*.

$$w_n = \begin{cases} 1.0 & \text{Rectangular} \\ 0.5 + 0.5\,\cos(n\pi/(N+1)) & \text{Hanning} \\ 0.54 + 0.46\,\cos(n\pi/(N+1)) & \text{Hamming} \\ 0.42 + 0.5\,\cos(n\pi/(N+1)) + 0.08\,\cos(2n\pi/(N+1)) & \text{Blackman .} \end{cases} \tag{5.3}$$

The Fourier series of the rectangular window has a narrow center lobe, but its side lobes contain a large part of the total energy, and decay very slowly. The Hamming window has 99.96 percent of its energy in its main lobe, but the width of the main lobe is twice the width of the rectangular window's main lobe. The Blackman window further reduces the peak side lobe ripple at the expense of a main lobe whose width is about triple the width of the rectangular window's main lobe. There are other window designs that are optimal in one way or another [7, 6, 5], but the window coefficients are in some cases difficult to compute, and as we will see below, we can design satisfactory filters with the windows described above.

If the low-pass filter must have a very narrow pass-band region, which is usually the case in the surface smoothing application, then a high degree polynomial is necessary to obtain a reasonable approximation. This is in fact a consequence of the uncertainty principle. The phenomenon can be observed even in the case of the rectangular window, illustrated in figure 6. The problem is even worse for the other windows, because they have wider main lobes. To obtain a reasonably good approximation of degree N, the pass-band must be significantly wider than the width of the main lobe of the window. If σ is the width of the main lobe of the window, the resulting filter will be approximately equal to one for $\theta \in [0, \theta_{\text{PB}} - \sigma]$, approximately equal to zero for $\theta \in [\theta_{\text{PB}} + \sigma, \pi]$, and approximately decreasing for $\theta \in [\theta_{\text{PB}} - \sigma, \theta_{\text{PB}} + \sigma]$. Our solution in this case of narrow pass-band frequency, is to design the filter for a small value of N, but with the pass-band frequency increased by σ (no longer the width of the main lobe of the window)

$$f_N(k) = w_0\,\frac{(\theta_{\text{PB}} + \sigma)}{\pi}\,T_0(1 - k/2) + w_n \sum_{n=1}^{N} \frac{2\sin(n(\theta_{\text{PB}} + \sigma))}{n\,\pi}\,T_n(1 - k/2)\ , \tag{5.4}$$

and then, eventually iterate this filter ($f(k) = f_N(k)^M$). The value of σ can be determined numerically by maximizing $f(k_{\text{PB}})$ under the constraints $|f(k)| < 1$ for $k_{\text{PB}} < k \leq 2$. In our implementation, we compute the optimal σ with a local root finding algorithm (a few Newton iterations) so that $f_N(k_{\text{PB}}) = 1$, starting from an interactively chosen initial value. Figure 7 shows some examples of filters designed in this way, compared with filters of the same degree and $\sigma = 0$, and with $\lambda - \mu$ filters of the same degree. Figure 8 shows several views of the filter design control panel of our interactive surface editing system.

6. Implementation

Figure 9 describes our algorithmic implementation of the filtering operation $x' = f_N(K)\,x$, where $f(k)$ is the transfer function

$$f(k) = \sum_{j=0}^{N} f_j\,T_j(1 - k/2)\ .$$

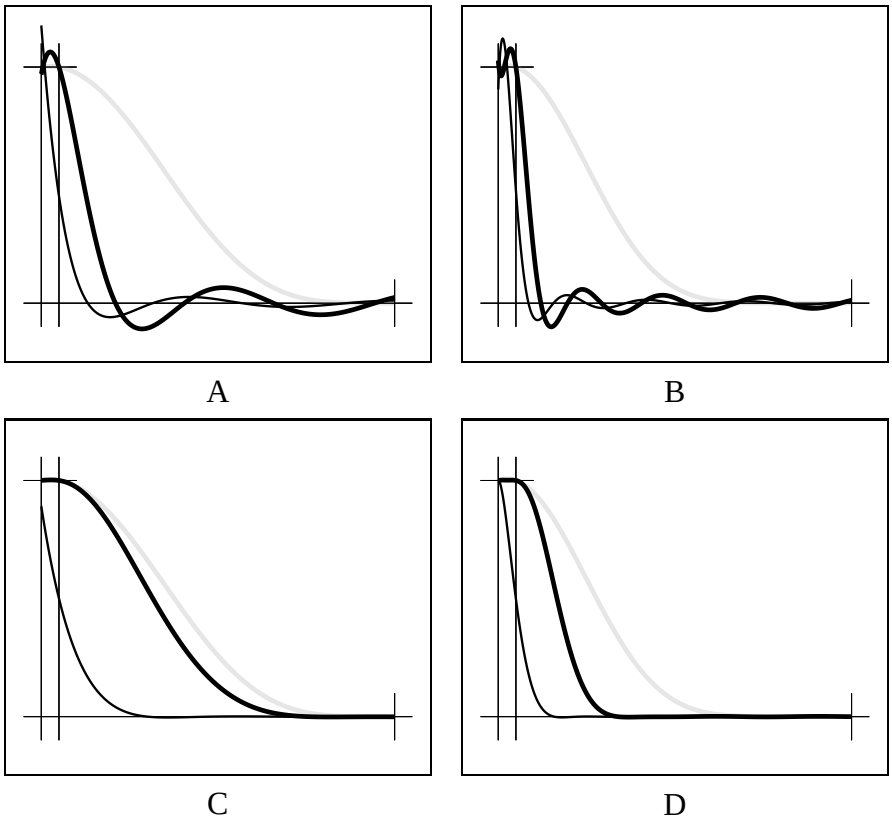


Figure 7: Filters $f_N(k)$ for $k_{\text{PB}} = 0.1$ and $\sigma > 0.0$. (A) Rectangular window, $N = 10$, $\sigma = 0.1353$. (B) Rectangular window, $N = 20$, $\sigma = 0.0637$. (C) Hamming window, $N = 10$, $\sigma = 0.5313$. (D) Hamming window, $N = 20$, $\sigma = 0.2327$. In each of the four cases the thick black line corresponds to the filter described above, the thin black line to the same filter with $\sigma = 0.0$, and the gray line is a $\lambda - \mu$ filter of the same degree and $\lambda = 0.5$.

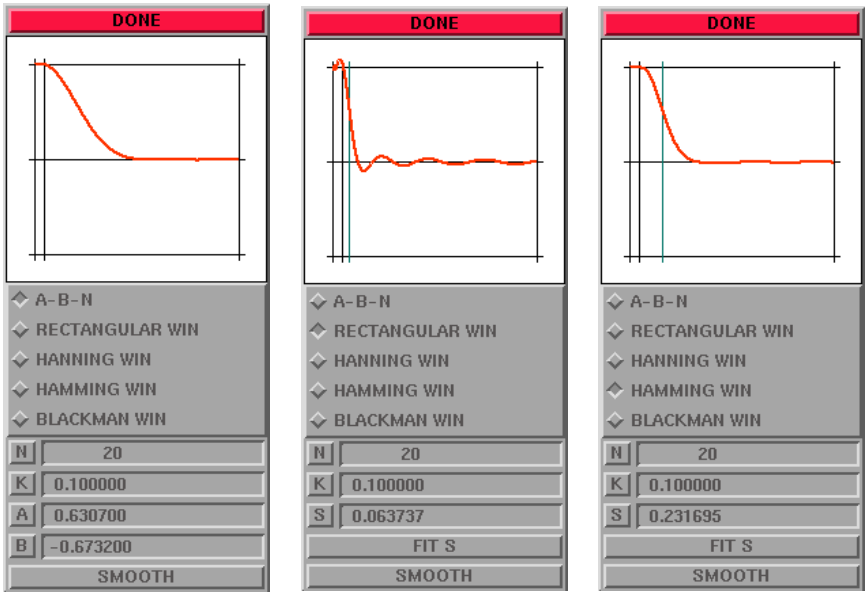


Figure 8: Interactive filter design subsystem.

This algorithm applies not only to low-pass filters, but to any polynomial transfer function expressed as linear combination of Chebyshev polynomials (every polynomial can be written in this way). From the numerical point of view, the Chebyshev polynomials constitute a better basis than the power basis because they are orthogonal in the interval $[-1, 1]$, Furthermore, the design techniques described above produce polynomial coefficients with respect to this basis. In terms of storage, the algorithm only requires four auxiliary arrays x^0, x^1, x^2, x^3 of dimension n. In terms of computation, the most expensive operation is the multiplication of a vector by the matrix K, an operation that is executed N times. This is the evaluation of the Laplacian, described in equation (2.2), which is also a linear complexity operation, because K is sparse.

filter$(N, f_0, \ldots, f_N, K, x, x')$
  $x^0 = x$
  $x^1 = Kx^0$
  $x^1 = x^0 - \frac{1}{2}x^1$
  $x^3 = f_0 x^0 + f_1 x^1$
  for $j = 2$ **to** N **step** 1 **do**
    $x^2 = Kx^1$
    $x^2 = (x^1 - x^0) + (x^1 - x^2)$
    $x^3 = x^3 + f_i x^2$
    $x^0 = x^1$
    $x^1 = x^2$
  end
  $x' = x^3$
return

Figure 9: The filtering algorithm $x' = f(K)x$.

7. How to Choose The Pass-Band Frequency

So far in our discussion of how to design low-pass filters, the pass-band frequency k_{PB} was given. In this section we are concerned with how to choose the pass-band frequency to prevent shrinkage. If the signals are the coordinates of the vertices of a closed surface, preservation of the enclosed volume is a natural criterion. But even in this case, normalizing the filtered signal to make it satisfy the criterion is an expensive global operation that requires the evaluation of a surface integral. And since the criterion does not have a natural generalization to arbitrary discrete graph signals, we will use a different criterion, more related to the signal processing formulation. As in the classical case, since the DFT $\hat{x}$ of a signal x satisfies Parseval's formula, the value of $\hat{x}_i^2$ can be interpreted as the *energy content* of x in the frequency k_i. Similarly, the sum

$$\sum_{k_i \leq k_{\text{PB}}} \hat{x}_i^2$$

measures the energy content of x in the pass-band. Our criterion is to choose the minimum pass-band frequency such that most of the energy of the signal falls in the pass-band, i.e., we choose k_{PB} such that

$$\sum_{k_i \leq k_{\text{PB}}} \hat{x}_i^2 \geq (1 - \epsilon) \|x\|_D^2 ,$$

where ϵ is a very small number. Of course, since we cannot compute the DFT of x, we cannot minimize this expression exactly. We can only get a rough estimate of the minimizer using the power spectral estimator described in the next section. What value of ϵ to use, and how accurate the estimation should be is application dependent, but in general it should be determined experimentally for a set of typical signals.

8. Power Spectrum Estimation

Ideally, to evaluate the performance of the different low-pass filter algorithms we should measure the DFT of the filter outputs, and check that the high frequency energy content is very small. Since we do not have any practical way of computing the DFT, we estimate the power spectrum, or energy distribution, of a signal as follows. We partition the interval $[0, 2]$ into a small number of non-overlapping intervals $I^1, \ldots, I^M$, and for each one of this intervals we estimate the energy content of the signal within the interval. We do so by designing a very sharp (high degree) pass-band filter $f^j(k)$ for each interval I^j. The energy content of the signal x within the interval I^j can be estimated by measuring the total energy of the output of corresponding filter applied to the signal

$$\|f^j(K)x\|_D^2 \approx \sum_{k_i \in I^j} \hat{x}_i^2 .$$

By designing all these FIR filters of the same degree, a filter-bank, we can evaluate all of them simultaneously at a greatly reduced computational cost. The only disadvantage is that we need M arrays of the same dimension as the input signal x to accumulate the filter outputs before their norms are evaluated. If the pass-band filters were ideal, Parseval's formula implies that the sum of the total energies of the filter outputs must be equal to the total energy of the input signal. Since the transfer functions of the filters overlap, this condition is only approximately satisfied. But the error can be made arbitrarily small by increasing the degree of the polynomials.

Figure 8 shows several views of the spectrum estimation control panel of our interactive surface editing system. In this figure N is the degree of the filters in the filter bank, M is the number of bands, and $K0$ is the width of each band. We recommend using filters designed with the Hanning or Hamming windows of a degree at lest ten times the number of spectrum bands.

9. Experimental Results

We have integrated all the methods described above within our surface editing and visualization system, illustrated in figure 11. Figure 12 shows the result of applying the filters of figure 7 to the same input surface. The spectrum estimate for the input surface yields the 99.88% of the energy in the band $[0, 0.1]$. This is a typical result for relatively large surfaces, and we have found that a default

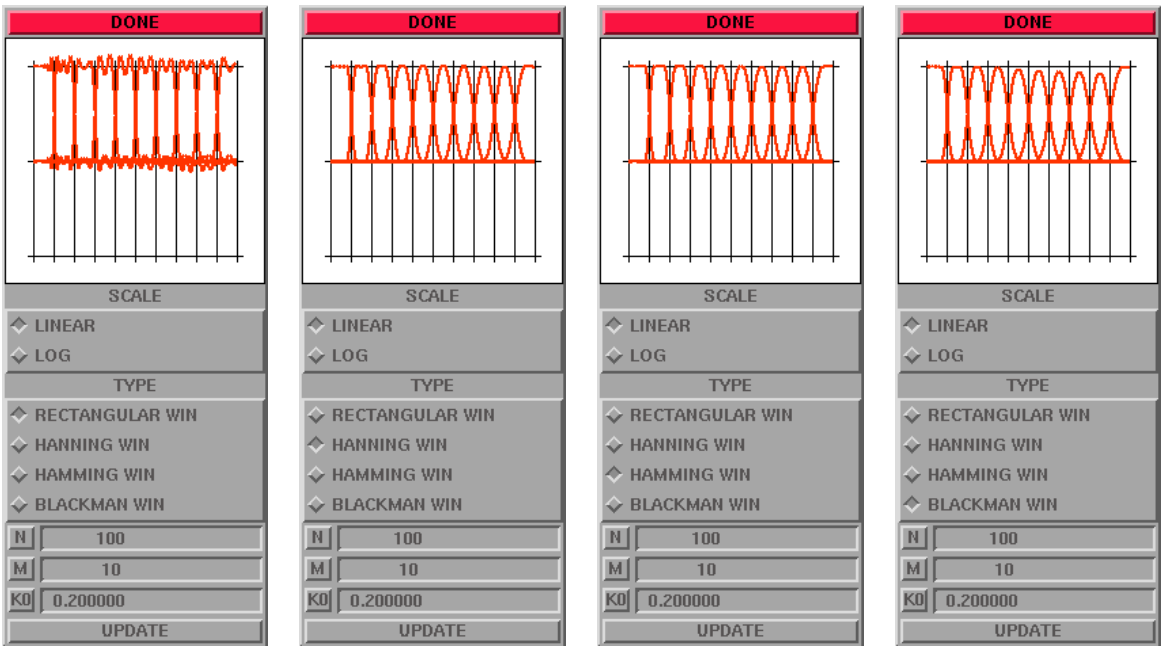


Figure 10: Interactive power spectrum estimation subsystem.

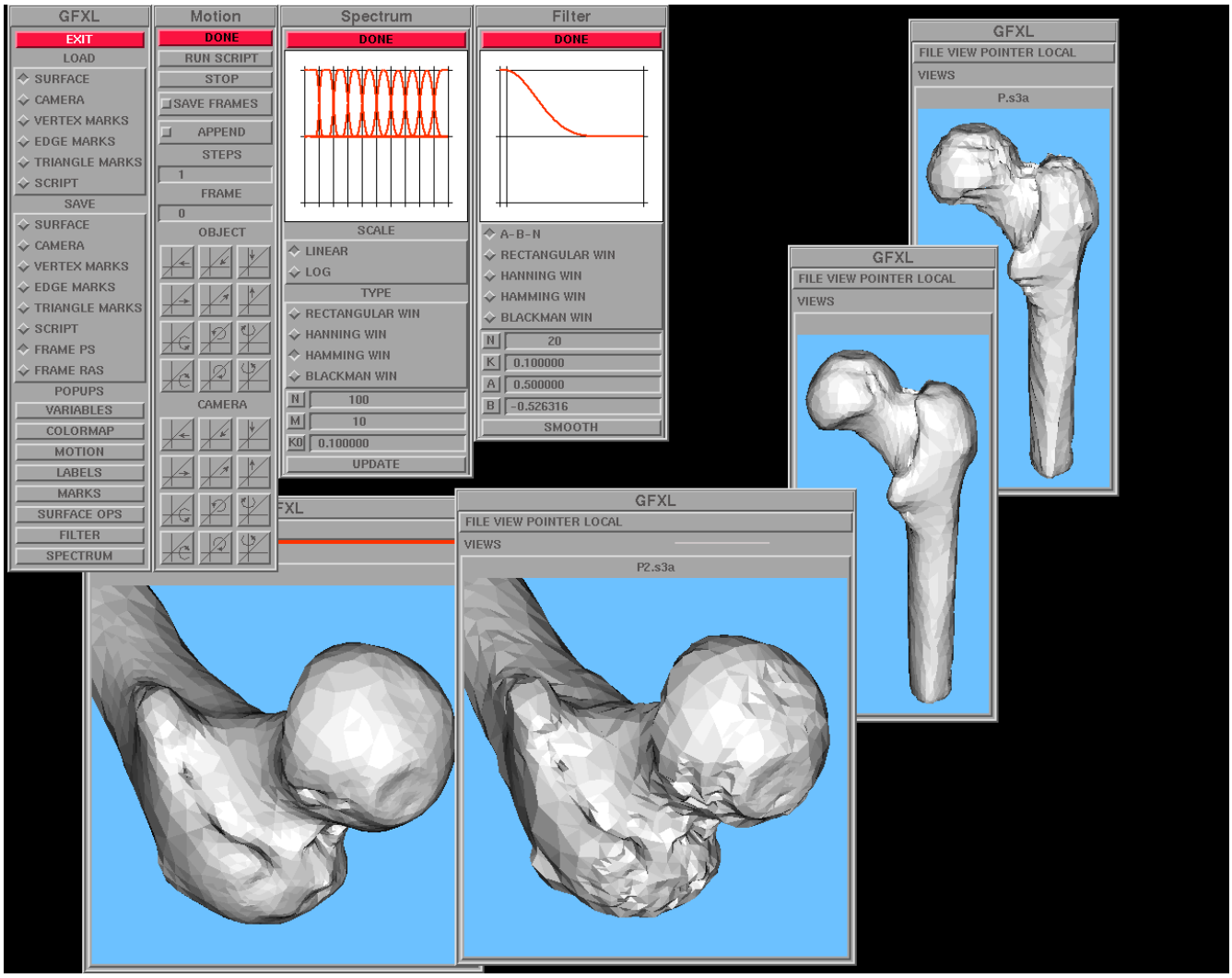


Figure 11: Interactive surface editing system.

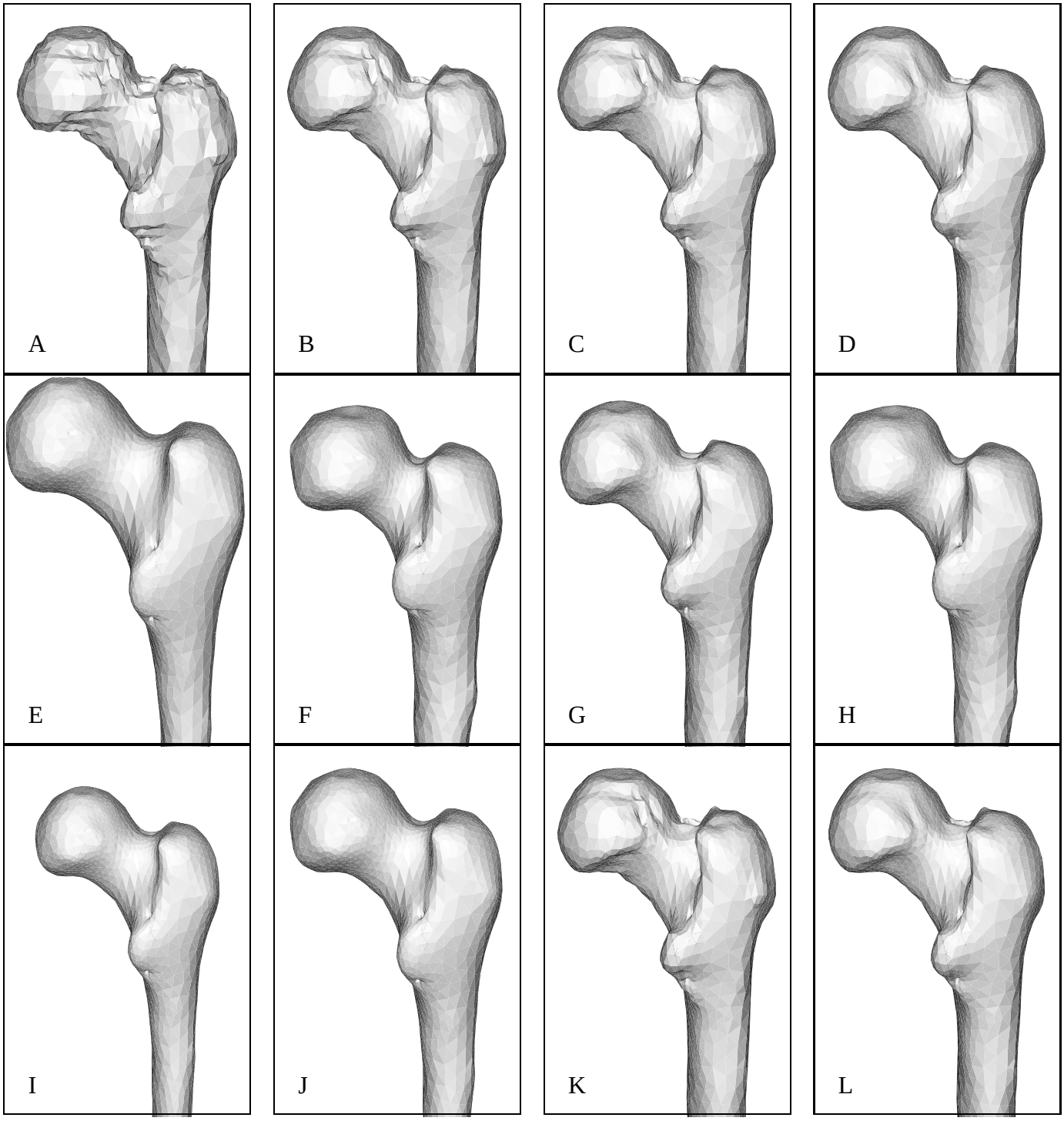


Figure 12: Filters of Figure 7 applied to the same surface. In all these examples $k_{\text{PB}} = 0.1$. (A) Input surface (2565 vertices, 5138 triangles). (B) $\lambda - \mu$ filter $\lambda = 0.5$ $n = 10$. (C) $\lambda - \mu$ filter $\lambda = 0.5$ $n = 20$. (D) $\lambda - \mu$ filter $\lambda = 0.5$ $n = 60$. (E) Rectangular window $\sigma = 0.0$ $n = 10$. (F) Rectangular window $\sigma = 0.0$ $n = 20$. (G) Rectangular window $\sigma = 0.01353$ $n = 10$. (H) Rectangular window $\sigma = 0.06374$ $n = 20$. (I) Hamming window $\sigma = 0.0$ $n = 10$. (J) Hamming window $\sigma = 0.0$ $n = 20$. (K) Hamming window $\sigma = 0.5313$ $n = 10$. (L) Hamming window $\sigma = 0.2327$ $n = 20$.

value $k_{\text{PB}} = 0.1$ produces very good results. But as we pointed out before, the appropriate value for a family of similar signals must be determined experimentally by estimating the spectrum of a typical sample.

The $\lambda - \mu$ algorithm produces very good results, but to significantly reduce high frequencies, a relatively large number of iterations might be necessary. The results obtained with rectangular filters are unsatisfactory. They are somehow better when we increase the value of σ, as described in section 5, but although they are faster, they change the low frequencies components too much, altering the shape quite significantly.

The ideal transfer function should be as flat as possible in the pass-band region ($f(k) \approx 1$ for $k \in [0, k_{\text{PB}}]$), and then decrease as fast as possible in the stop-band region ($k \in [k_{\text{PB}}, 2]$). The transfer function of the $\lambda - \mu$ algorithm has this shape, but does not decrease fast enough in the stop-band. The filters designed with the other three windows (Hanning, Hamming, and Blackman), and with increased σ produce transfer functions of similar shape. The Blackman window produces transfer functions that are much flatter in the pass-band, but at the expense of a slower rate of decrease in the stop-band. Hanning and Hamming windows produce similar results, but the Hamming window produces transfer functions with less oscillations. As figure 12 shows, filters designed with the Hamming window produce filters of similar quality as the $\lambda - \mu$ algorithm, but much faster.

10. Conclusions

Generalizing the signal processing formulation of [10, 11], in this paper we formulated the most significant concepts of Fourier analysis for signals defined on oriented graphs, and showed that linear filters with polynomial transfer function can be implemented in an efficient manner, and designed with classical digital filter design methods. In particular, we have shown how to design surface smoothing filters that produce almost the same effect as the filter described in [10, 11], but in a fraction of the time. We have also described a method to estimate the power spectrum of a signal, and used this power spectrum estimate to determine the pass-band frequency for a surface smoothing filter, and as a tool to evaluate the performance of different filter designs. We have also given a formal definition of the shrinkage problem, which is valid not only for closed surfaces, but for any signal.

References.

[1] R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume 1. Interscience, 1953.

[2] P.J. Davis. *Interpolation and Approximation*. Dover Publications, Inc., 1975.

[3] H. Dym and H.P. McKean. *Fourier Series and Integrals*, volume 14 of *Probability and Mathematical Statistics*. Academic Press, 1972.

[4] G. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins University Press, 2nd. edition, 1989.

[5] R.W. Hamming. *Digital Filters*. Prentice Hall, 1989.

[6] H.D. Helms. Nonrecursive digital filters: Design methods for achieving specifications on frequency response. *IEEE Transactions on Audio and Electroacustics*, AU-16:336–342, September 1968.

[7] J.F. Kaiser. Digital filters. In F.F. Kuo and J.F. Kaiser, editors, *System Analysis by Digital Computer*. Wiley, New York, 1966.

[8] A.V. Oppenheim and R.W. Schafer. *Digital Signal Processing*. Prentice Hall, Englewood Cliffs, NJ, 1975.

[9] E. Seneta. *Non-Negative Matrices, An Introduction to Theory and Applications*. John Wiley & Sons, New York, 1973.

[10] G. Taubin. Curve and surface smoothing without shrinkage. In *Proceedings, Fifth International Conference on Computer Vision*, pages 852–857, June 1995.

[11] G. Taubin. A signal processing approach to fair surface design. *Computer Graphics*, pages 351–358, August 1995. (Proceedings SIGGRAPH'95).

[12] C.T. Zahn and R.Z. Roskies. Fourier descriptors for plane closed curves. *IEEE Transactions on Computers*, 21(3):269–281, March 1972.

Interactive Multiresolution Mesh Editing

by D. Zorin, P. Schroder, and W. Sweldens

Siggraph'97

Interactive Multiresolution Mesh Editing

Denis Zorin* Caltech
Peter Schröder† Caltech
Wim Sweldens‡ Bell Laboratories

Abstract

We describe a multiresolution representation for meshes based on subdivision, which is a natural extension of the existing patch-based surface representations. Combining subdivision and the smoothing algorithms of Taubin [26] allows us to construct a set of algorithms for interactive multiresolution editing of complex hierarchical meshes of arbitrary topology. The simplicity of the underlying algorithms for refinement and coarsification enables us to make them local and adaptive, thereby considerably improving their efficiency. We have built a scalable interactive multiresolution editing system based on such algorithms.

1 Introduction

Applications such as special effects and animation require creation and manipulation of complex geometric models of arbitrary topology. Like real world geometry, these models often carry detail at many scales (cf. Fig. 1). The model might be constructed from scratch (ab initio design) in an interactive modeling environment or be scanned-in either by hand or with automatic digitizing methods. The latter is a common source of data particularly in the entertainment industry. When using laser range scanners, for example, individual models are often composed of high resolution meshes with hundreds of thousands to millions of triangles.

Manipulating such fine meshes can be difficult, especially when they are to be edited or animated. Interactivity, which is crucial in these cases, is challenging to achieve. Even without accounting for any computation on the mesh itself, available rendering resources alone, may not be able to cope with the sheer size of the data. Possible approaches include mesh optimization [15, 13] to reduce the size of the meshes.

Aside from considerations of economy, the choice of representation is also guided by the need for multiresolution editing semantics. The representation of the mesh needs to provide control at a large scale, so that one can change the mesh in a broad, smooth manner, for example. Additionally designers will typically also want control over the minute features of the model (cf. Fig. 1). Smoother approximations can be built through the use of patches [14], though at the cost of loosing the high frequency details. Such detail can be reintroduced by combining patches with displacement maps [17]. However, this is difficult to manage in the arbitrary topology setting and across a continuous range of scales and hardware resources.

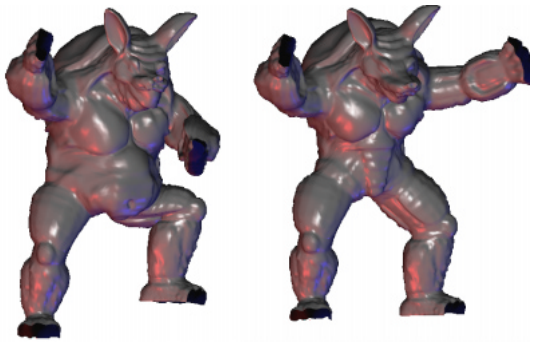


Figure 1: Before the Armadillo started working out he was flabby, complete with a double chin. Now he exercises regularly. The original is on the right (courtesy Venkat Krischnamurthy). The edited version on the left illustrates large scale edits, such as his belly, and smaller scale edits such as his double chin; all edits were performed at about 5 frames per second on an Indigo R10000 Solid Impact.

For reasons of efficiency the algorithms should be highly adaptive and dynamically adjust to available resources. Our goal is to have a single, simple, uniform representation with scalable algorithms. The system should be capable of delivering multiple frames per second update rates even on small workstations taking advantage of lower resolution representations.

In this paper we present a system which possesses these properties

- **Multiresolution control:** Both broad and general handles, as well as small knobs to tweak minute detail are available.
- **Speed/fidelity tradeoff:** All algorithms dynamically adapt to available resources to maintain interactivity.
- **Simplicity/uniformity:** A single primitive, triangular mesh, is used to represent the surface across all levels of resolution.

Our system is inspired by a number of earlier approaches. We mention multiresolution editing [11, 9, 12], arbitrary topology subdivision [6, 2, 19, 7, 28, 16], wavelet representations [21, 24, 8, 3], and mesh simplification [13, 17]. Independently an approach similar to ours was developed by Pulli and Lounsbery [23].

It should be noted that our methods rely on the finest level mesh having subdivision connectivity. This requires a remeshing step before external high resolution geometry can be imported into the editor. Eck et al. [8] have described a possible approach to remeshing arbitrary finest level input meshes fully automatically. A method that relies on a user's expertise was developed by Krishnamurthy and Levoy [17].

1.1 Earlier Editing Approaches

H-splines were presented in pioneering work on hierarchical editing by Forsey and Bartels [11]. Briefly, H-splines are obtained by adding finer resolution B-splines onto an existing coarser resolution B-spline patch relative to the coordinate frame induced by the

*dzorin@gg.caltech.edu
†ps@cs.caltech.edu
‡wim@bell-labs.com

coarser patch. Repeating this process, one can build very complicated shapes which are entirely parameterized over the unit square. Forsey and Bartels observed that the hierarchy induced coordinate frame for the offsets is essential to achieve correct editing semantics.

H-splines provide a uniform framework for representing both the coarse and fine level details. Note however, that as more detail is added to such a model the internal control mesh data structures more and more resemble a fine polyhedral mesh.

While their original implementation allowed only for regular topologies their approach could be extended to the general setting by using surface splines or one of the spline derived general topology subdivision schemes [18]. However, these schemes have not yet been made to work adaptively.

Forsey and Bartels' original work focused on the ab initio design setting. There the user's help is enlisted in defining what is meant by different levels of resolution. The user decides where to add detail and manipulates the corresponding controls. This way the levels of the hierarchy are hand built by a human user and the representation of the final object is a function of its editing history.

To edit an a priori given model it is crucial to have a general procedure to define coarser levels and compute details between levels. We refer to this as the *analysis* algorithm. An H-spline analysis algorithm based on weighted least squares was introduced [10], but is too expensive to run interactively. Note that even in an ab initio design setting online analysis is needed, since after a long sequence of editing steps the H-spline is likely to be overly refined and needs to be consolidated.

Wavelets provide a framework in which to rigorously define multiresolution approximations and fast analysis algorithms. Finkelstein and Salesin [9], for example, used B-spline wavelets to describe multiresolution editing of curves. As in H-splines, parameterization of details with respect to a coordinate frame induced by the coarser level approximation is required to get correct editing semantics. Gortler and Cohen [12], pointed out that wavelet representations of detail tend to behave in undesirable ways during editing and returned to a pure B-spline representation as used in H-splines.

Carrying these constructions over into the arbitrary topology surface framework is not straightforward. In the work by Lounsbery et al. [21] the connection between wavelets and subdivision was used to define the different levels of resolution. The original constructions were limited to piecewise linear subdivision, but smoother constructions are possible [24, 28].

An approach to surface modeling based on variational methods was proposed by Welch and Witkin [27]. An attractive characteristic of their method is flexibility in the choice of control points. However, they use a global optimization procedure to compute the surface which is not suitable for interactive manipulation of complex surfaces.

Before we proceed to a more detailed discussion of editing we first discuss different surface representations to motivate our choice of synthesis (refinement) algorithm.

1.2 Surface Representations

There are many possible choices for surface representations. Among the most popular are polynomial patches and polygons.

Patches are a powerful primitive for the construction of coarse grain, smooth models using a small number of control parameters. Combined with hardware support relatively fast implementations are possible. However, when building complex models with many patches the preservation of smoothness across patch boundaries can be quite cumbersome and expensive. These difficulties are compounded in the arbitrary topology setting when polynomial parameterizations cease to exist everywhere. Surface splines [4, 20, 22] provide one way to address the arbitrary topology challenge.

As more fine level detail is needed the proliferation of control points and patches can quickly overwhelm both the user and the most powerful hardware. With detail at finer levels, patches become less suited and polygonal meshes are more appropriate.

Polygonal Meshes can represent arbitrary topology and resolve fine detail as found in laser scanned models, for example. Given that most hardware rendering ultimately resolves to triangle scan-conversion even for patches, polygonal meshes are a very basic primitive. Because of sheer size, polygonal meshes are difficult to manipulate interactively. Mesh simplification algorithms [13] provide one possible answer. However, we need a mesh simplification approach, that is hierarchical and gives us shape handles for smooth changes over larger regions while maintaining high frequency details.

Patches and fine polygonal meshes represent two ends of a spectrum. Patches efficiently describe large smooth sections of a surface but cannot model fine detail very well. Polygonal meshes are good at describing very fine detail accurately using dense meshes, but do not provide coarser manipulation semantics.

Subdivision connects and unifies these two extremes.

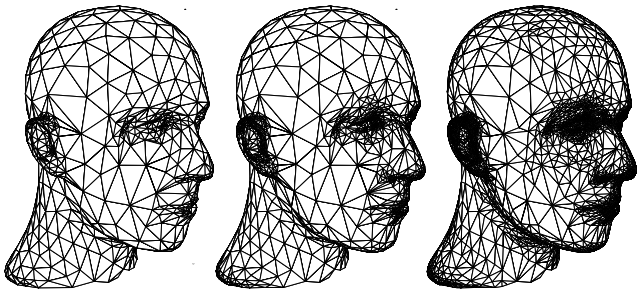


Figure 2: Subdivision describes a smooth surface as the limit of a sequence of refined polyhedra. The meshes show several levels of an adaptive Loop surface generated by our system (dataset courtesy Hugues Hoppe, University of Washington).

Subdivision defines a smooth surface as the limit of a sequence of successively refined polyhedral meshes (cf. Fig. 2). In the regular patch based setting, for example, this sequence can be defined through well known knot insertion algorithms [5]. Some subdivision methods generalize spline based knot insertion to irregular topology control meshes [2, 6, 19] while other subdivision schemes are independent of splines and include a number of interpolating schemes [7, 28, 16].

Since subdivision provides a path from patches to meshes, it can serve as a good foundation for the unified infrastructure that we seek. A single representation (hierarchical polyhedral meshes) supports the patch-type semantics of manipulation *and* finest level detail polyhedral edits equally well. The main challenge is to make the basic algorithms fast enough to escape the exponential time and space growth of naive subdivision. This is the core of our contribution.

We summarize the main features of subdivision important in our context

- **Topological Generality:** Vertices in a triangular (resp. quadrilateral) mesh need not have valence 6 (resp. 4). Generated surfaces are smooth everywhere, and efficient algorithms exist for computing normals and limit positions of points on the surface.
- **Multiresolution:** because they are the limit of successive refinement, subdivision surfaces support multiresolution algorithms, such as level-of-detail rendering, multiresolution editing, compression, wavelets, and numerical multigrid.

- **Simplicity:** subdivision algorithms are simple: the finer mesh is built through insertion of new vertices followed by *local* smoothing.
- **Uniformity of Representation:** subdivision provides a single representation of a surface at all resolution levels. Boundaries and features such as creases can be resolved through modified rules [14, 25], reducing the need for trim curves, for example.

1.3 Our Contribution

Aside from our perspective, which unifies the earlier approaches, our major contribution—and the main challenge in this program—is the design of highly adaptive and dynamic data structures and algorithms, which allow the system to function across a range of computational resources from PCs to workstations, delivering as much interactive fidelity as possible with a given polygon rendering performance. Our algorithms work for the class of 1-ring subdivision schemes (definition see below) and we demonstrate their performance for the concrete case of Loop's subdivision scheme.

The particulars of those algorithms will be given later, but Fig. 3 already gives a preview of how the different algorithms make up the editing system. In the next sections we first talk in more detail about subdivision, smoothing, and multiresolution transforms.

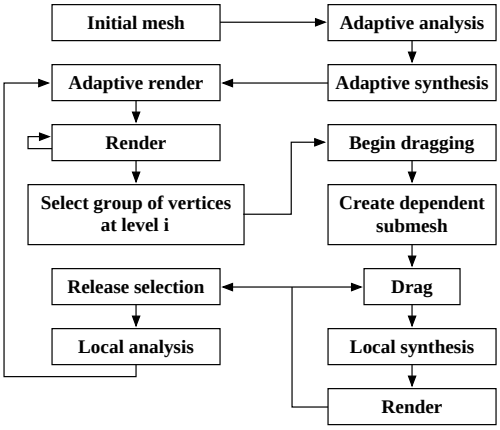


Figure 3: The relationship between various procedures as the user moves a set of vertices.

2 Subdivision

We begin by defining subdivision and fixing our notation. There are 2 points of view that we must distinguish. On the one hand we are dealing with an abstract *graph* and perform topological operations on it. On the other hand we have a *mesh* which is the geometric object in 3-space. The mesh is the image of a map defined on the graph: it associates a *point* in 3D with every *vertex* in the graph (cf. Fig. 4). A *triangle* denotes a face in the graph or the associated polygon in 3-space.

Initially we have a triangular graph T^0 with vertices V^0. By recursively *refining* each triangle into 4 subtriangles we can build a sequence of finer triangulations T^i with vertices V^i, $i > 0$ (cf. Fig. 4). The superscript i indicates the *level* of triangles and vertices respectively. A triangle $t \in T^i$ is a triple of indices $t = \{v_a, v_b, v_c\} \subset V^i$.

The vertex sets are nested as $V^j \subset V^i$ if $j < i$. We define *odd* vertices on level i as $M^i = V^{i+1} \setminus V^i$. V^{i+1} consists of two disjoint sets: *even* vertices (V^i) and *odd* vertices (M^i). We define the *level* of a vertex v as the smallest i for which $v \in V^i$. The level of v is $i+1$ if and only if $v \in M^i$.

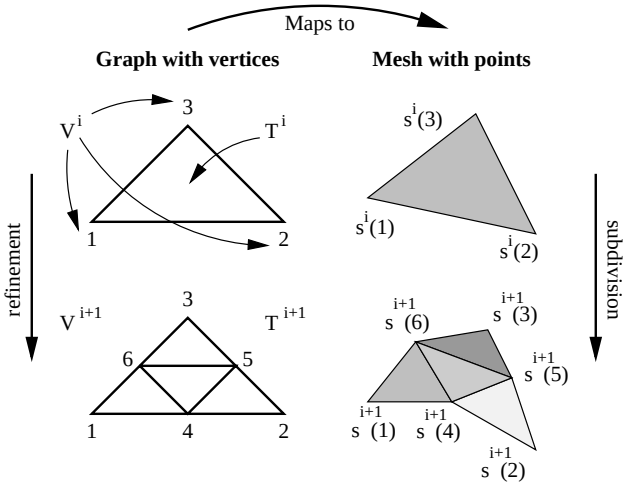


Figure 4: Left: the abstract graph. Vertices and triangles are members of sets V^i and T^i respectively. Their index indicates the level of refinement when they first appeared. Right: the mapping to the mesh and its subdivision in 3-space.

With each set V^i we associate a map, i.e., for each vertex v and each level i we have a 3D point $s^i(v) \in \mathbf{R}^3$. The set s^i contains all points on level i, $s^i = \{s^i(v) \mid v \in V^i\}$. Finally, a *subdivision scheme* is a linear operator S which takes the points from level i to points on the *finer* level $i+1$: $s^{i+1} = S\, s^i$

Assuming that the subdivision converges, we can define a limit surface σ as

$$\sigma = \lim_{k \to \infty} S^k\, s^0.$$

$\sigma(v) \in \mathbf{R}^3$ denotes the point on the limit surface associated with vertex v.

In order to define our offsets with respect to a local frame we also need tangent vectors and a normal. For the subdivision schemes that we use, such vectors can be defined through the application of linear operators Q and R acting on s^i so that $q^i(v) = (Qs^i)(v)$ and $r^i(v) = (Rs^i)(v)$ are linearly independent tangent vectors at $\sigma(v)$. Together with an orientation they define a local orthonormal frame $F^i(v) = (n^i(v), q^i(v), r^i(v))$. It is important to note that in general it is not necessary to use precise normals and tangents during editing; as long as the frame vectors are affinely related to the positions of vertices of the mesh, we can expect intuitive editing behavior.

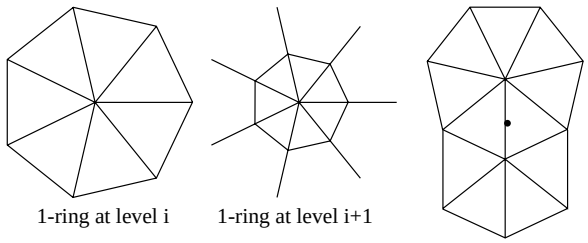


Figure 5: An even vertex has a 1-ring of neighbors at each level of refinement (left/middle). Odd vertices—in the middle of edges—have 1-rings around each of the vertices at either end of their edge (right).

Next we discuss two common subdivision schemes, both of which belong to the class of *1-ring schemes*. In these schemes points at level $i+1$ depend only on 1-ring neighborhoods of points

at level i. Let $v \in V^i$ (v even) then the point $s^{i+1}(v)$ is a function of only those $s^i(v_n)$, $v_n \in V^i$, which are immediate neighbors of v (cf. Fig. 5 left/middle). If $m \in M^i$ (m odd), it is the vertex inserted when splitting an edge of the graph; we call such vertices *middle vertices* of edges. In this case the point $s^{i+1}(m)$ is a function of the 1-rings around the vertices at the ends of the edge (cf. Fig. 5 right).

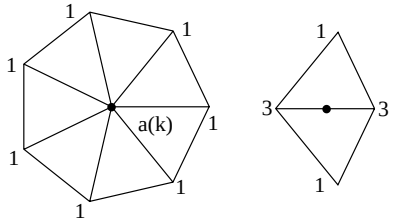


Figure 6: Stencils for Loop subdivision with unnormalized weights for even and odd vertices.

Loop is a non-interpolating subdivision scheme based on a generalization of quartic triangular box splines [19]. For a given even vertex $v \in V^i$, let $v_k \in V^i$ with $1 \le k \le K$ be its K 1-ring neighbors. The new point $s^{i+1}(v)$ is defined as $s^{i+1}(v) = (a(K)+K)^{-1}(a(K)\, s^i(v) + \sum_{k=1}^{K} s^i(v_k))$ (cf. Fig. 6), $a(K) = K(1-\alpha(K))/\alpha(K)$, and $\alpha(K) = 5/8-(3+2\cos(2\pi/K))^2/64$. For odd v the weights shown in Fig. 6 are used. Two independent tangent vectors $t_1(v)$ and $t_2(v)$ are given by $t_p(v) = \sum_{k=1}^{K} \cos(2\pi(k+p)/K)\, s^i(v_k)$.

Features such as boundaries and cusps can be accommodated through simple modifications of the stencil weights [14, 25, 29].

Butterfly is an interpolating scheme, first proposed by Dyn et al. [7] in the topologically regular setting and recently generalized to arbitrary topologies [28]. Since it is interpolating we have $s^i(v) = \sigma(v)$ for $v \in V^i$ even. The exact expressions for odd vertices depend on the valence K and the reader is referred to the original paper for the exact values [28].

For our implementation we have chosen the Loop scheme, since more performance optimizations are possible in it. However, the algorithms we discuss later work for any 1-ring scheme.

3 Multiresolution Transforms

So far we only discussed subdivision, i.e., how to go from coarse to fine meshes. In this section we describe analysis which goes from fine to coarse.

We first need *smoothing*, i.e., a linear operation H to build a smooth coarse mesh at level $i-1$ from a fine mesh at level i:

$$s^{i-1} = H\, s^i.$$

Several options are available here:

- **Least squares:** One could define analysis to be optimal in the least squares sense,

$$\min_{s^{i-1}} \|s^i - S\, s^{i-1}\|^2.$$

  The solution may have unwanted undulations and is too expensive to compute interactively [10].

- **Fairing:** A coarse surface could be obtained as the solution to a global variational problem. This is too expensive as well. An alternative is presented by Taubin [26], who uses a *local* non-shrinking smoothing approach.

Because of its computational simplicity we decided to use a version of Taubin smoothing. As before let $v \in V^i$ have K neighbors $v_k \in V^i$. Use the average, $\overline{s}^i(v) = K^{-1} \sum_{k=1}^{K} s^i(v_k)$, to define the discrete Laplacian $\mathcal{L}(v) = \overline{s}^i(v) - s^i(v)$. On this basis Taubin gives a Gaussian-like smoother which does not exhibit shrinkage

$$H := (I + \mu\, \mathcal{L})\, (I + \lambda\, \mathcal{L}).$$

With subdivision and smoothing in place, we can describe the transform needed to support multiresolution editing. Recall that for multiresolution editing we want the difference between successive levels expressed with respect to a frame induced by the coarser level, i.e., the offsets are relative to the smoother level.

With each vertex v and each level $i > 0$ we associate a *detail vector*, $d^i(v) \in \mathbf{R}^3$. The set d^i contains all detail vectors on level i, $d^i = \{d^i(v) \mid v \in V^i\}$. As indicated in Fig. 7 the detail vectors are defined as

$$d^i = (F^i)^t\, (s^i - S\, s^{i-1}) = (F^i)^t\, (I - S\, H)\, s^i,$$

i.e., the detail vectors at level i record how much the points at level i differ from the result of subdividing the points at level $i-1$. This difference is then represented with respect to the local frame F^i to obtain coordinate independence.

Since detail vectors are sampled on the fine level mesh V^i, this transformation yields an overrepresentation in the spirit of the Burt-Adelson Laplacian pyramid [1]. The only difference is that the smoothing filters (Taubin) are not the dual of the subdivision filter (Loop). Theoretically it would be possible to subsample the detail vectors and only record a detail per odd vertex of M^{i-1}. This is what happens in the wavelet transform. However, subsampling the details severely restricts the family of smoothing operators that can be used.

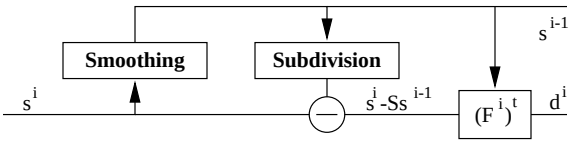


Figure 7: Wiring diagram of the multiresolution transform.

4 Algorithms and Implementation

Before we describe the algorithms in detail let us recall the overall structure of the mesh editor (cf. Fig 3). The analysis stage builds a succession of coarser approximations to the surface, each with fewer control parameters. Details or offsets between successive levels are also computed. In general, the coarser approximations are not visible; only their control points are rendered. These control points give rise to a *virtual surface* with respect to which the remaining details are given. Figure 8 shows wireframe representations of virtual surfaces corresponding to control points on levels 0, 1, and 2.

When an edit level is selected, the surface is represented internally as an approximation at this level, plus the set of all finer level details. The user can freely manipulate degrees of freedom at the edit level, while the finer level details remain unchanged relative to the coarser level. Meanwhile, the system will use the synthesis algorithm to render the modified edit level with all the finer details added in. In between edits, analysis enforces consistency on the internal representation of coarser levels and details (cf. Fig. 9).

The basic algorithms `Analysis` and `Synthesis` are very simple and we begin with their description.

Let $i = 0$ be the coarsest and $i = n$ the finest level with N vertices. For each vertex v and all levels i finer than the first level

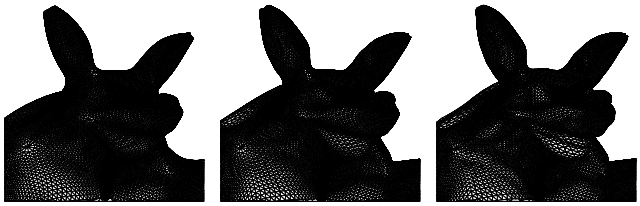


Figure 8: Wireframe renderings of virtual surfaces representing the first three levels of control points.

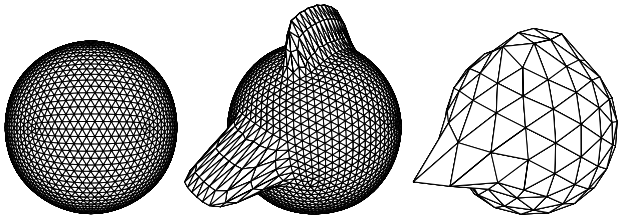


Figure 9: Analysis propagates the changes on finer levels to coarser levels, keeping the magnitude of details under control. Left: The initial mesh. Center: A simple edit on level 3. Right: The effect of the edit on level 2. A significant part of the change was absorbed by higher level details.

where the vertex v appears, there are storage locations $v.s[i]$ and $v.d[i]$, each with 3 floats. With this the total storage adds to $2 * 3 * (4N/3)$ floats. In general, $v.s[i]$ holds $s^i(v)$ and $v.d[i]$ holds $d^i(v)$; temporarily, these locations can be used to store other quantities. The local frame is computed by calling $v.F(i)$.

Global analysis and synthesis are performed level wise:

```
Analysis
  for i = n downto 1
    Analysis(i)
```

```
Synthesis
  for i = 1 to n
    Synthesis(i)
```

With the action at each level described by

```
Analysis(i)
  ∀v ∈ V^{i-1} : v.s[i-1] := smooth(v, i)
  ∀v ∈ V^i     : v.d[i] := v.F(i)^t * (v.s[i] - subd(v, i-1))
```

and

```
Synthesis(i)
  ∀v ∈ V^i : s.v[i] := v.F(i) * v.d[i] + subd(v, i-1)
```

Analysis computes points on the coarser level $i-1$ using smoothing (`smooth`), subdivides s^{i-1} (`subd`), and computes the detail vectors d^i (cf. Fig. 7). Synthesis reconstructs level i by subdividing level $i-1$ and adding the details.

So far we have assumed that all levels are uniformly refined, i.e., all neighbors at all levels exist. Since time and storage costs grow exponentially with the number of levels, this approach is unsuitable for an interactive implementation. In the next sections we explain how these basic algorithms can be made memory and time efficient.

Adaptive and *local* versions of these generic algorithms (cf. Fig. 3 for an overview of their use) are the key to these savings. The underlying idea is to use lazy evaluation and pruning based on thresholds. Three thresholds control this pruning: ϵ_A for adaptive analysis, ϵ_S for adaptive synthesis, and ϵ_R for adaptive rendering. To make lazy evaluation fast enough several caches are maintained explicitly and the order of computations is carefully staged to avoid recomputation.

4.1 Adaptive Analysis

The generic version of analysis traverses entire levels of the hierarchy starting at some finest level. Recall that the purpose of analysis is to compute coarser approximations and detail offsets. In many regions of a mesh, for example, if it is flat, no significant details will be found. *Adaptive analysis* avoids the storage cost associated with detail vectors below some threshold ϵ_A by observing that small detail vectors imply that the finer level almost coincides with the subdivided coarser level. The storage savings are realized through *tree pruning*.

For this purpose we need an integer $v.finest := \max_i\{\|v.d[i]\| \geq \epsilon_A\}$. Initially $v.finest = n$ and the following precondition holds before calling `Analysis(i)`:

- The surface is uniformly subdivided to level i,
- $\forall v \in V^i : v.s[i] = s^i(v)$,
- $\forall v \in V^i \mid i < j \leq v.finest : v.d[j] = d^j(v)$.

Now `Analysis(i)` becomes:

```
Analysis(i)
  ∀v ∈ V^{i-1} : v.s[i-1] := smooth(v, i)
  ∀v ∈ V^i :
    v.d[i] := v.s[i] - subd(v, i-1)
    if v.finest > i or ||v.d[i]|| ≥ ε_A then
      v.d[i] := v.F(i)^t * v.d[i]
    else
      v.finest := i - 1
  Prune(i - 1)
```

Triangles that do not contain details above the threshold are unrefined:

```
Prune(i)
  ∀t ∈ T^i : If all middle vertices m have m.finest = i - 1
             and all children are leaves, delete children.
```

This results in an adaptive mesh structure for the surface with $v.d[i] = d^i(v)$ for all $v \in V^i$, $i \leq v.finest$. Note that the resulting mesh is not restricted, i.e., two triangles that share a vertex can differ in more than one level. Initial analysis has to be followed by a synthesis pass which enforces restriction.

4.2 Adaptive Synthesis

The main purpose of the general synthesis algorithm is to rebuild the finest level of a mesh from its hierarchical representation. Just as in the case of analysis we can get savings from noticing that in flat regions, for example, little is gained from synthesis and one might as well save the time and storage associated with synthesis. This is the basic idea behind *adaptive synthesis*, which has two main purposes. First, ensure the mesh is restricted on each level, (cf. Fig. 10). Second, refine triangles and recompute points until the mesh has reached a certain measure of local flatness compared against the threshold ϵ_S.

The algorithm recomputes the points $s^i(v)$ starting from the coarsest level. Not all neighbors needed in the subdivision stencil of a given point necessarily exist. Consequently adaptive synthesis

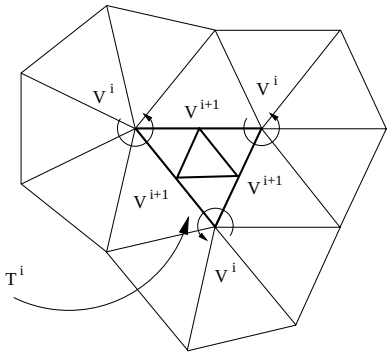


Figure 10: A restricted mesh: the center triangle is in T^i and its vertices in V^i. To subdivide it we need the 1-rings indicated by the circular arrows. If these are present the graph is restricted and we can compute s^{i+1} for all vertices and middle vertices of the center triangle.

lazily creates all triangles needed for subdivision by temporarily refining their parents, then computes subdivision, and finally deletes the newly created triangles unless they are needed to satisfy the restriction criterion. The following precondition holds before entering `AdaptiveSynthesis`:

- $\forall t \in T^j \mid 0 \leq j \leq i$: t is restricted
- $\forall v \in V^j \mid 0 \leq j \leq v.depth$: $v.s[j] = s^j(v)$

where $v.depth := \max_i\{s^i(v)\text{has been recomputed}\}$.

```
AdaptiveSynthesis

  ∀v ∈ V^0 : v.depth := 0
  for i = 0 to n − 1
    temptri := {}
    ∀t ∈ T^i :
      current := {}
      Refine(t, i, true)
    ∀t ∈ temptri : if not t.restrict then
      Delete children of t
```

The list $temptri$ serves as a cache holding triangles from levels $j < i$ which are temporarily refined. A triangle is appended to the list if it was refined to compute a value at a vertex. After processing level i these triangles are unrefined unless their $t.restrict$ flag is set, indicating that a temporarily created triangle was later found to be needed permanently to ensure restriction. Since triangles are appended to $temptri$, parents precede children. Deallocating the list tail first guarantees that all unnecessary triangles are erased.

The function `Refine`(t, i, dir) (see below) creates children of $t \in T^i$ and computes the values $Ss^i(v)$ for the vertices and middle vertices of t. The results are stored in $v.s[i+1]$. The boolean argument dir indicates whether the call was made directly or recursively.

```
Refine(t, i, dir)

  if t.leaf then  Create children for t
  ∀v ∈ t : if v.depth < i + 1 then
    GetRing(v, i)
    Update(v, i)
    ∀m ∈ N(v, i + 1, 1) :
      Update(m, i)
      if m.finest ≥ i + 1 then
        forced := true
  if dir and Flat(t) < ε_S and not forced then
    Delete children of t
  else
    ∀t ∈ current : t.restrict := true

Update(v, i)
  v.s[i + 1] := subd(v, i)
  v.depth := i + 1
  if v.finest ≥ i + 1 then
    v.s[i + 1] += v.F(i + 1) ∗ v.d[i + 1]
```

The condition $v.depth = i + 1$ indicates whether an earlier call to `Refine` already recomputed $s^{i+1}(v)$. If not, call `GetRing`(v, i) and `Update`(v, i) to do so. In case a detail vector lives at v at level i ($v.finest \geq i + 1$) add it in. Next compute $s^{i+1}(m)$ for middle vertices on level $i + 1$ around v ($m \in N(v, i + 1, 1)$, where $N(v, i, l)$ is the l-ring neighborhood of vertex v at level i). If m has to be calculated, compute `subd`(m, i) and add in the detail if it exists and record this fact in the flag $forced$ which will prevent unrefinement later. At this point, all s^{i+1} have been recomputed for the vertices and middle vertices of t. Unrefine t and delete its children if `Refine` was called directly, the triangle is sufficiently flat, and none of the middle vertices contain details (i.e., $forced =$ `false`). The list $current$ functions as a cache holding triangles from level $i - 1$ which are temporarily refined to build a 1-ring around the vertices of t. If after processing all vertices and middle vertices of t it is decided that t will remain refined, none of the coarser-level triangles from $current$ can be unrefined without violating restriction. Thus $t.restrict$ is set for all of them. The function `Flat`(t) measures how close to planar the corners and edge middle vertices of t are.

Finally, `GetRing`(v, i) ensures that a complete ring of triangles on level i adjacent to the vertex v exists. Because triangles on level i are restricted triangles all triangles on level $i - 1$ that contain v exist (precondition). At least one of them is refined, since otherwise there would be no reason to call `GetRing`(v, i). All other triangles could be leaves or temporarily refined. Any triangle that was already temporarily refined may become permanently refined to enforce restriction. Record such candidates in the $current$ cache for fast access later.

```
GetRing(v, i)

  ∀t ∈ T^{i−1} with v ∈ t :
    if t.leaf then
      Refine(t, i − 1, false); temptri.append(t)
      t.restrict := false; t.temp := true
    if t.temp then
      current.append(t)
```

4.3 Local Synthesis

Even though the above algorithms are adaptive, they are still run everywhere. During an edit, however, not all of the surface changes. The most significant economy can be gained from performing analysis and synthesis only over submeshes which require it.

Assume the user edits level l and modifies the points $s^l(v)$ for $v \in V^{*l} \subset V^l$. This invalidates coarser level values s^i and d^i for certain subsets $V^{*i} \subset V^i$, $i \leq l$, and finer level points s^i for subsets $V^{*i} \subset V^i$ for $i > l$. Finer level detail vectors d^i for $i > l$ remain correct by definition. Recomputing the coarser levels is done by *local incremental analysis* described in Section 4.4, recomputing the finer level is done by *local synthesis* described in this section.

The set of vertices V^{*i} which are affected depends on the support of the subdivision scheme. If the support fits into an m-ring around the computed vertex, then all modified vertices on level $i+1$ can be found recursively as

$$V^{*i+1} = \bigcup_{v \in V^{*i}} N(v, i+1, m).$$

We assume that $m = 2$ (Loop-like schemes) or $m = 3$ (Butterfly type schemes). We define the *subtriangulation* T^{*i} to be the subset of triangles of T^i with vertices in V^{*i}.

`LocalSynthesis` is only slightly modified from `AdaptiveSynthesis`: iteration starts at level l and iterates only over the submesh T^{*i}.

4.4 Local Incremental Analysis

After an edit on level l *local incremental analysis* will recompute $s^i(v)$ and $d^i(v)$ locally for coarser level vertices ($i \leq l$) which are affected by the edit. As in the previous section, we assume that the user edited a set of vertices v on level l and call V^{*i} the set of vertices affected on level i. For a given vertex $v \in V^{*i}$ we define

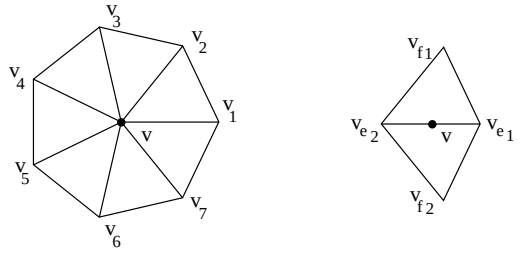


Figure 11: Sets of even vertices affected through smoothing by either an even v or odd m vertex.

$R^{i-1}(v) \subset V^{i-1}$ to be the set of vertices on level $i-1$ affected by v through the smoothing operator H. The sets V^{*i} can now be defined recursively starting from level $i = l$ to $i = 0$:

$$V^{*i-1} = \bigcup_{v \in V^{*i}} R^{i-1}(v).$$

The set $R^{i-1}(v)$ depends on the size of the smoothing stencil and whether v is even or odd (cf. Fig. 11). If the smoothing filter is 1-ring, e.g., Gaussian, then $R^{i-1}(v) = \{v\}$ if v is even and $R^{i-1}(m) = \{v_{e1}, v_{e2}\}$ if m is odd. If the smoothing filter is 2-ring, e.g., Taubin, then $R^{i-1}(v) = \{v\} \cup \{v_k \mid 1 \leq k \leq K\}$ if v is even and $R^{i-1}(m) = \{v_{e1}, v_{e2}, v_{f1}, v_{f2}\}$ if v is odd. Because of restriction, these vertices always exist. For $v \in V^i$ and $v' \in R^{i-1}(v)$ we let $c(v, v')$ be the coefficient in the analysis stencil. Thus

$$(H\, s^i)(v') = \sum_{v \,|\, v' \in R^{i-1}(v)} c(v, v') s^i(v).$$

This could be implemented by running over the v' and each time computing the above sum. Instead we use the dual implementation, iterate over all v, accumulating (`+=`) the right amount to $s^i(v')$ for $v' \in R^{i-1}(v)$. In case of a 2-ring Taubin smoother the coefficients are given by

$$\begin{aligned} c(v,v) &= (1-\mu)(1-\lambda) + \mu\lambda/6 \\ c(v,v_k) &= \mu\lambda/6K \\ c(m,v_{e1}) &= ((1-\mu)\lambda + (1-\lambda)\mu + \mu\lambda/3)/K \\ c(m,v_{f1}) &= \mu\lambda/3K, \end{aligned}$$

where for each $c(v, v')$, K is the outdegree of v'.

The algorithm first copies the old points $s^i(v)$ for $v \in V^{*i}$ and $i \leq l$ into the storage location for the detail. If then propagates the incremental changes of the modified points from level l to the coarser levels and adds them to the old points (saved in the detail locations) to find the new points. Then it recomputes the detail vectors that depend on the modified points.

We assume that before the edit, the old points $s^l(v)$ for $v \in V^{*l}$ were saved in the detail locations. The algorithm starts out by building V^{*i-1} and saving the points $s^{i-1}(v)$ for $v \in V^{*i-1}$ in the detail locations. Then the changes resulting from the edit are propagated to level $i-1$. Finally $S\, s^{i-1}$ is computed and used to update the detail vectors on level i.

```
LocalAnalysis(i)

  ∀v ∈ V^{*i} : ∀v' ∈ R^{i-1}(v) :
    V^{*i-1} ∪= {v'}
    v'.d[i-1] := v'.s[i-1]
  ∀v ∈ V^{*i} : ∀v' ∈ R^{i-1}(v) :
    v'.s[i-1] += c(v,v') * (v.s[i] - v.d[i])
  ∀v ∈ V^{*i-1} :
    v.d[i] = v.F(i)^t * (v.s[i] - subd(v, i-1))
    ∀m ∈ N(v,i,1) :
      m.d[i] = m.F(i)^t * (m.s[i] - subd(m, i-1))
```

Note that the odd points are actually computed twice. For the Loop scheme this is less expensive than trying to compute a predicate to avoid this. For Butterfly type schemes this is not true and one can avoid double computation by imposing an ordering on the triangles. The top level code is straightforward:

```
LocalAnalysis

  ∀v ∈ V^{*l} : v.d[l] := v.s[l]
  for i := l downto 0
    LocalAnalysis(i)
```

It is difficult to make incremental local analysis adaptive, as it is formulated purely in terms of vertices. It is, however, possible to adaptively clean up the triangles affected by the edit and (un)refine them if needed.

4.5 Adaptive Rendering

The *adaptive rendering* algorithm decides which triangles will be drawn depending on the rendering performance available and level of detail needed.

The algorithm uses a flag $t.draw$ which is initialized to `false`, but set to `true` as soon as the area corresponding to t is drawn. This can happen either when t itself gets drawn, or when a set of its descendents, which cover t, is drawn. The top level algorithm loops through the triangles starting from the level $n-1$. A triangle

is always responsible for drawing its children, never itself, unless it is a coarsest-level triangle.

```
AdaptiveRender

  for i = n − 1 downto 0
    ∀t ∈ T^i : if not t.leaf then
      Render(t)
  ∀t ∈ T^0 : if not t.draw then
    displaylist.append(t)
```

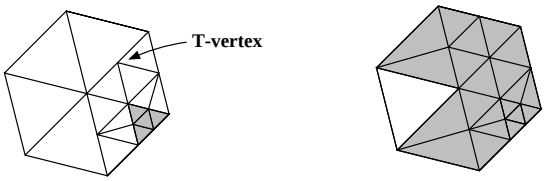


Figure 12: Adaptive rendering: On the left 6 triangles from level i, one has a covered child from level $i+1$, and one has a T-vertex. On the right the result from applying `Render` to all six.

The `Render`(t) routine decides whether the children of t have to be drawn or not (cf. Fig.12). It uses a function `edist`(m) which measures the distance between the point corresponding to the edge's middle vertex m, and the edge itself. In the when case any of the children of t are already drawn or any of its middle vertices are far enough from the plane of the triangle, the routine will draw the rest of the children and set the draw flag for all their vertices and t. It also might be necessary to draw a triangle if some of its middle vertices are drawn because the triangle on the other side decided to draw its children. To avoid cracks, the routine `cut`(t) will cut t into 2, 3, or 4, triangles depending on how many middle vertices are drawn.

```
Render(t)

  if (∃ c ∈ t.child | c.draw = true
   or ∃ m ∈ t.mid_vertex | edist(m) > ε_D) then
    ∀c ∈ t.child :
      if not c.draw then
        displaylist.append(c)
        ∀v ∈ c : v.draw := true
    t.draw := true
  else if ∃ m ∈ t.mid_vertex | m.draw = true
    ∀t' ∈ cut(t) : displaylist.append(t')
    t.draw := true
```

4.6 Data Structures and Code

The main data structure in our implementation is a forest of triangular quadtrees. Neighborhood relations within a single quadtree can be resolved in the standard way by ascending the tree to the least common parent when attempting to find the neighbor across a given edge. Neighbor relations between adjacent trees are resolved explicitly at the level of a collection of roots, i.e., triangles of a coarsest level graph. This structure also maintains an explicit representation of the boundary (if any). Submeshes rooted at any level can be created on the fly by assembling a new graph with some set of triangles as roots of their child quadtrees. It is here that the explicit representation of the boundary comes in, since the actual trees are never copied, and a boundary is needed to delineate the actual submesh.

The algorithms we have described above make heavy use of container classes. Efficient support for sets is essential for a fast implementation and we have used the C++ Standard Template Library. The mesh editor was implemented using OpenInventor and OpenGL and currently runs on both SGI and Intel PentiumPro workstations.

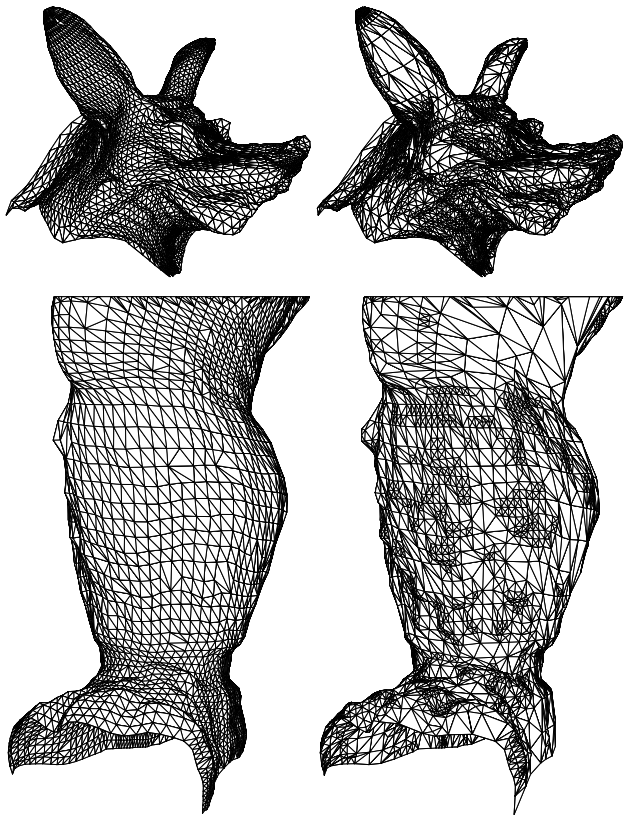


Figure 13: On the left are two meshes which are uniformly subdivided and consist of 11k (upper) and 9k (lower) triangles. On the right another pair of meshes mesh with approximately the same numbers of triangles. Upper and lower pairs of meshes are generated from the same original data but the right meshes were optimized through suitable choice of ϵ_S. See the color plates for a comparison between the two under shading.

5 Results

In this section we show some example images to demonstrate various features of our system and give performance measures.

Figure 13 shows two triangle mesh approximations of the Armadillo head and leg. Approximately the same number of triangles are used for both adaptive and uniform meshes. The meshes on the left were rendered uniformly, the meshes on the right were rendered adaptively. (See also color plate 15.)

Locally changing threshold parameters can be used to resolve an area of interest particularly well, while leaving the rest of the mesh at a coarse level. An example of this “lens” effect is demonstrated in Figure 14 around the right eye of the Mannequin head. (See also color plate 16.)

We have measured the performance of our code on two platforms: an Indigo R10000@175MHz with Solid Impact graphics, and a PentiumPro@200MHz with an Intergraph Intense 3D board.

We used the Armadillo head as a test case. It has approximately 172000 triangles on 6 levels of subdivision. Display list creation took 2 seconds on the SGI and 3 seconds on the PC for the full model. We adjusted ϵ_R so that both machines rendered models at 5 frames per second. In the case of the SGI approximately 113,000 triangles were rendered at that rate. On the PC we achieved 5 frames per second when the rendering threshold had been raised enough so that an approximation consisting of 35000 polygons was used.

The other important performance number is the time it takes to recompute and re-render the region of the mesh which is changing as the user moves a set of control points. This submesh is rendered in immediate mode, while the rest of the surface continues to be rendered as a display list. Grabbing a submesh of 20-30 faces (a typical case) at level 0 added 250 mS of time per redraw, at level 1 it added 110 mS and at level 2 it added 30 mS in case of the SGI. The corresponding timings for the PC were 500 mS, 200 mS and 60 mS respectively.

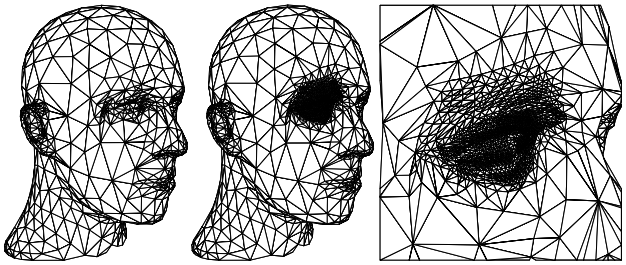


Figure 14: It is easy to change ϵ_S locally. Here a "lens" was applied to the right eye of the Mannequin head with decreasing ϵ_S to force very fine resolution of the mesh around the eye.

6 Conclusion and Future Research

We have built a scalable system for interactive multiresolution editing of arbitrary topology meshes. The user can either start from scratch or from a given fine detail mesh *with subdivision connectivity*. We use smooth subdivision combined with details at each level as a uniform surface representation across scales and argue that this forms a natural connection between fine polygonal meshes and patches. Interactivity is obtained by building both local and adaptive variants of the basic analysis, synthesis, and rendering algorithms, which rely on fast lazy evaluation and tree pruning. The system allows interactive manipulation of meshes according to the polygon performance of the workstation or PC used.

There are several avenues for future research:

- Multiresolution transforms readily connect with compression. We want to be able to store the models in a compressed format and use progressive transmission.
- Features such as creases, corners, and tension controls can easily be added into our system and expand the users' editing toolbox.
- Presently no real time fairing techniques, which lead to more intuitive coarse levels, exist.
- In our system coarse level edits can only be made by dragging coarse level vertices. Which vertices live on coarse levels is currently fixed because of subdivision connectivity. Ideally the user should be able to dynamically adjust this to make coarse level edits centered at arbitrary locations.
- The system allows topological edits on the coarsest level. Algorithms that allow topological edits on all levels are needed.
- An important area of research relevant for this work is generation of meshes with subdivision connectivity from scanned data or from existing models in other representations.

Acknowledgments

We would like to thank Venkat Krishnamurthy for providing the Armadillo dataset. Andrei Khodakovsky and Gary Wu helped beyond the call of duty to bring the system up. The research was supported in part through grants from the Intel Corporation, Microsoft, the Charles Lee Powell Foundation, the Sloan Foundation, an NSF CAREER award (ASC-9624957), and under a MURI (AFOSR F49620-96-1-0471). Other support was provided by the NSF STC for Computer Graphics and Scientific Visualization.

References

[1] BURT, P. J., AND ADELSON, E. H. Laplacian Pyramid as a Compact Image Code. *IEEE Trans. Commun. 31*, 4 (1983), 532–540.

[2] CATMULL, E., AND CLARK, J. Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design 10*, 6 (1978), 350–355.

[3] CERTAIN, A., POPOVIĆ, J., DEROSE, T., DUCHAMP, T., SALESIN, D., AND STUETZLE, W. Interactive Multiresolution Surface Viewing. In *SIGGRAPH 96 Conference Proceedings*, H. Rushmeier, Ed., Annual Conference Series, 91–98, Aug. 1996.

[4] DAHMEN, W., MICCHELLI, C. A., AND SEIDEL, H.-P. Blossoming Begets B-Splines Bases Built Better by B-Patches. *Mathematics of Computation 59*, 199 (July 1992), 97–115.

[5] DE BOOR, C. *A Practical Guide to Splines*. Springer, 1978.

[6] DOO, D., AND SABIN, M. Analysis of the Behaviour of Recursive Division Surfaces near Extraordinary Points. *Computer Aided Design 10*, 6 (1978), 356–360.

[7] DYN, N., LEVIN, D., AND GREGORY, J. A. A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. *ACM Trans. Gr. 9*, 2 (April 1990), 160–169.

[8] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. In *Computer Graphics Proceedings*, Annual Conference Series, 173–182, 1995.

[9] FINKELSTEIN, A., AND SALESIN, D. H. Multiresolution Curves. *Computer Graphics* Proceedings, Annual Conference Series, 261–268, July 1994.

[10] FORSEY, D., AND WONG, D. Multiresolution Surface Reconstruction for Hierarchical B-splines. Tech. rep., University of British Columbia, 1995.

[11] FORSEY, D. R., AND BARTELS, R. H. Hierarchical B-Spline Refinement. *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, No. 4, pp. 205–212, August 1988.

[12] GORTLER, S. J., AND COHEN, M. F. Hierarchical and Variational Geometric Modeling with Wavelets. In *Proceedings Symposium on Interactive 3D Graphics*, May 1995.

[13] HOPPE, H. Progressive Meshes. In *SIGGRAPH 96 Conference Proceedings*, H. Rushmeier, Ed., Annual Conference Series, 99–108, August 1996.

[14] HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J., AND STUETZLE, W. Piecewise Smooth Surface Reconstruction. In *Computer Graphics Proceedings*, Annual Conference Series, 295–302, 1994.

[15] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Mesh Optimization. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, J. T. Kajiya, Ed., vol. 27, 19–26, August 1993.

[16] KOBBELT, L. Interpolatory Subdivision on Open Quadrilateral Nets with Arbitrary Topology. In *Proceedings of Eurographics 96*, Computer Graphics Forum, 409–420, 1996.

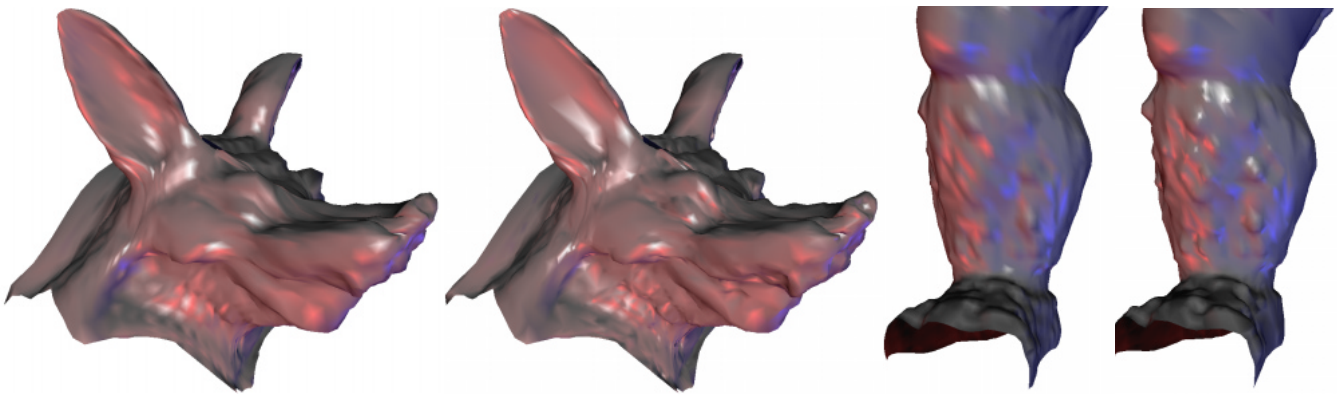


Figure 15: Shaded rendering (OpenGL) of the meshes in Figure 13.

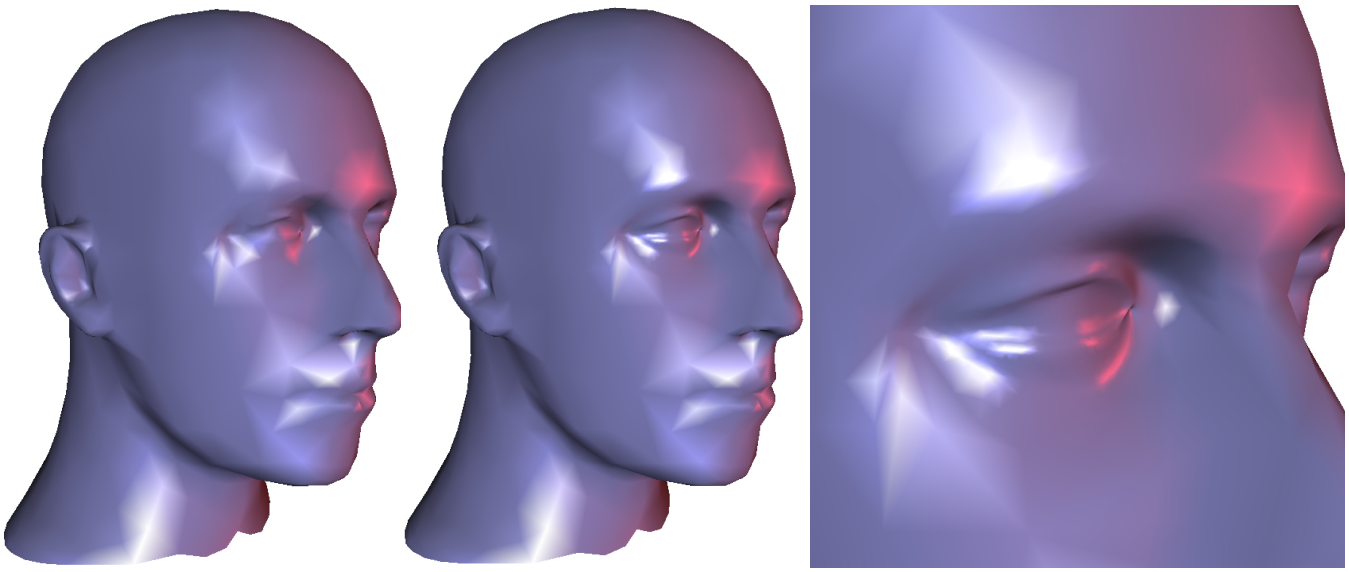


Figure 16: Shaded rendering (OpenGL) of the meshes in Figure 14.

[17] KRISHNAMURTHY, V., AND LEVOY, M. Fitting Smooth Surfaces to Dense Polygon Meshes. In *SIGGRAPH 96 Conference Proceedings*, H. Rushmeier, Ed., Annual Conference Series, 313–324, August 1996.

[18] KURIHARA, T. Interactive Surface Design Using Recursive Subdivision. In *Proceedings of Communicating with Virtual Worlds*. Springer Verlag, June 1993.

[19] LOOP, C. Smooth Subdivision Surfaces Based on Triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[20] LOOP, C. Smooth Spline Surfaces over Irregular Meshes. In *Computer Graphics Proceedings*, Annual Conference Series, 303–310, 1994.

[21] LOUNSBERY, M., DEROSE, T., AND WARREN, J. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *Transactions on Graphics 16*, 1 (January 1997), 34–73.

[22] PETERS, J. C^1 Surface Splines. *SIAM J. Numer. Anal. 32*, 2 (1995), 645–666.

[23] PULLI, K., AND LOUNSBERY, M. Hierarchical Editing and Rendering of Subdivision Surfaces. Tech. Rep. UW-CSE-97-04-07, Dept. of CS&E, University of Washington, Seattle, WA, 1997.

[24] SCHRÖDER, P., AND SWELDENS, W. Spherical wavelets: Efficiently representing functions on the sphere. *Computer Graphics Proceedings, (SIGGRAPH 95)* (1995), 161–172.

[25] SCHWEITZER, J. E. *Analysis and Application of Subdivision Surfaces*. PhD thesis, University of Washington, 1996.

[26] TAUBIN, G. A Signal Processing Approach to Fair Surface Design. In *SIGGRAPH 95 Conference Proceedings*, R. Cook, Ed., Annual Conference Series, 351–358, August 1995.

[27] WELCH, W., AND WITKIN, A. Variational surface modeling. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, E. E. Catmull, Ed., vol. 26, 157–166, July 1992.

[28] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interpolating Subdivision for Meshes with Arbitrary Topology. *Computer Graphics Proceedings (SIGGRAPH 96)* (1996), 189–192.

[29] ZORIN, D. N. *Subdivision and Multiresolution Surface Representations*. PhD thesis, Caltech, Pasadena, California, 1997.

Interactive Multi-Resolution Modeling on Arbitrary Meshes,

by L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel, Siggraph'98.

Interactive Multi-Resolution Modeling on Arbitrary Meshes

Leif Kobbelt* Swen Campagna Jens Vorsatz Hans-Peter Seidel

University of Erlangen–Nürnberg

Abstract

During the last years the concept of multi-resolution modeling has gained special attention in many fields of computer graphics and geometric modeling. In this paper we generalize powerful multi-resolution techniques to arbitrary triangle meshes without requiring subdivision connectivity. Our major observation is that the hierarchy of nested spaces which is the structural core element of most multi-resolution algorithms can be replaced by the sequence of intermediate meshes emerging from the application of incremental mesh decimation. Performing such schemes with local frame coding of the detail coefficients already provides effective and efficient algorithms to extract multi-resolution information from unstructured meshes. In combination with discrete fairing techniques, i.e., the constrained minimization of discrete energy functionals, we obtain very fast mesh smoothing algorithms which are able to reduce noise from a geometrically specified frequency band in a multi-resolution decomposition. Putting mesh hierarchies, local frame coding and multi-level smoothing together allows us to propose a flexible and intuitive paradigm for interactive detail-preserving mesh modification. We show examples generated by our mesh modeling tool implementation to demonstrate its functionality.

1 Introduction

Traditionally, geometric modeling is based on piecewise polynomial surface representations [8, 16]. However, while special polynomial basis functions are well suited for describing and modifying smooth triangular or quadrilateral *patches*, it turns out to be rather difficult to smoothly join several pieces of a composite surface along common (possibly trimmed) boundary curves. As flexible patch layout is crucial for the construction of non-trivial geometric shapes, spline-based modeling tools do spend much effort to maintain the global smoothness of a surface.

Subdivision schemes can be considered as an algorithmic generalization of classical spline techniques enabling control meshes with arbitrary topology [2, 5, 6, 18, 22, 39]. They provide easy access to globally smooth surfaces of arbitrary shape by iteratively applying simple refinement rules to the given control mesh. A sequence of meshes generated by this process quickly converges to a smooth limit surface. For most practical applications, the refined meshes are already sufficiently close to the smooth limit after only a few refinement steps.

Within a multi-resolution framework, subdivision schemes provide a set of basis functions $\phi_{i,j} = \phi(2^i \cdot - j)$ which are suitable to build a cascade of nested spaces $V_i = \mathrm{span}([\phi_{i,j}]_j)$ [4, 33]. Since the functions $\phi_{i,j}$ are defined by uniform refinement of a given control mesh $\mathcal{M}_0 \cong V_0$, the spaces V_i have to be isomorphic to meshes $\mathcal{M}_i$ with *subdivision connectivity*.

While being much more flexible than classical (tensor-product) spline techniques, the multi-resolution representation based on the uniform refinement of a polygonal base mesh is still rather rigid. When analyzing a given mesh $\mathcal{M}_k$, i.e., when decomposing the mesh into disjoint frequency bands $W_i = V_{i+1} \setminus V_i$, we have to *invert* the uniform refinement operation $V_i \to V_{i+1}$. Hence, the input mesh always has to be topologically isomorphic to an iteratively refined base grid. In general this requires a global remeshing/resampling of the input data prior to the multi-resolution analysis [7]. Moreover, if we want to fuse several separately generated subdivision meshes (e.g. laser range scans) into one model, restrictive compatibility conditions have to be satisfied. Hence, subdivision schemes are able to deal with arbitrary *topology* but not with arbitrary *connectivity*!

The *scales* of subdivision based multi-resolution mesh representations are defined in terms of topological distances. Since every vertex $\mathbf{p}_{i,j}$ on each level of subdivision $\mathcal{M}_i$ represents the weight coefficient of a particular basis function $\phi_{i,j}$ with fixed support, its region of influence is determined by topological neighborhood in the mesh instead of geometric proximity. Being derived from the regular functional setting, the refinement rules of stationary subdivision schemes only depend on the valences of the vertices but not on the length of the adjacent edges. Hence, surface artifacts can occur when the given base mesh is locally strongly distorted.

Assume we have a subdivision connectivity mesh and want to apply modifications on a specific scale V_i. The usual way to implement this operation is to run a decomposition scheme several steps until the desired resolution level is reached. On this level the mesh $\mathcal{M}_i$ is modified and the reconstruction starting with $\mathcal{M}_i'$ yields the final result. The major draw-back of this procedure is the fact that coarse basis functions exist for the coarse-mesh vertices only and hence all low-frequency modifications have to be *aligned* to the grid imposed by the subdivision connectivity. Shifted low-frequency modifications can be faked by moving a *group* of vertices from a finer scale simultaneously but this annihilates the mathematical elegance of multi-resolution representations.

A standard demo example for multi-resolution modeling is pulling the nose tip of a human head model. Depending on the chosen scale either the whole face is affected or just the nose is elongated. On uniformly refined meshes this operation only works if a coarse-scale control vertex happens to be located right at the nose tip. However, for an *automatic* remeshing algorithm it is very difficult, if not impossible, to place the coarse-scale vertices at the semantically relevant features of an object.

In this paper we present an alternative approach to multi-resolution modeling which avoids these three major difficulties, i.e. the restriction to subdivision connectivity meshes, the restriction to basis functions with fixed support and the alignment of potential coarser-scale modifications.

*Computer Sciences Department (IMMD9), University of Erlangen-Nürnberg, Am Weichselgarten 9, 91058 Erlangen, Germany, Leif.Kobbelt@informatik.uni-erlangen.de

The first problem is solved by using mesh hierarchies which emerge from the application of a mesh decimation scheme. In Section 2 we derive the necessary equipment to extract multi-resolution information from arbitrary meshes and geometrically encode detail information with respect to local frames which adapt to the local geometry of the coarser approximation of the object.

To overcome the problems arising from the fixed support and aligned distribution of subdivision basis functions, we drop the structural concept of considering a surface in space to be a linear combination of scalar-valued basis functions. On each level of detail, the lower-frequency components of the geometric shape are simply characterized by energy minimization (*fairing*). In Section 3 we overview the discrete fairing technique [19, 38] and show how a combination with the non-uniform mesh hierarchy leads to highly efficient mesh optimization algorithms. Due to the local smoothing properties of the fairing operators, we are able to define a *geometric* threshold for the wavelength up to which a low-pass filter should remove noise.

With an efficient hierarchical mesh smoothing scheme available, we propose a flexible mesh modification paradigm in Section 4. The basic idea is to let the designer freely define the region of influence and the characteristics of the modification which both can be adapted to the surface geometry instead of being determined by the connectivity. The selected region defines the "frequency" of the modification since it provides the boundary conditions for a constrained energy minimization. Nevertheless the detail information within the selected region is preserved and does change according to the global modification. Exploiting the efficient schemes from Section 3 leads to interactive response times for moderately complex models.

Throughout the paper, we consider a modeling scenario where a triangle mesh $\mathcal{M}$ with arbitrary connectivity is *given* (no from-scratch design). All modifications just alter the position of the vertices but not their adjacency. In particular, we do not consider ad infinitum subdivision to establish infinitesimal smoothness. The given mesh $\mathcal{M} = \mathcal{M}_k$ represents per definition the finest level of detail.

2 Multi-resolution representations

Most schemes for the multi-resolution representation and modification of triangle meshes emerge from generalizing harmonic analysis techniques like the wavelet transform [1, 23, 30, 33]. Since the fundamentals have been derived in the scalar-valued functional setting $\mathbb{R}^d \to \mathbb{R}$, difficulties emerge from the fact that manifolds in space are in general not topologically equivalent to simply connected regions in $\mathbb{R}^d$.

The philosophy behind multi-resolution modeling on surfaces is hence to mimic the algorithmic structure of the related functional transforms and preserve some of the important properties like locality, smoothness, stability or polynomial precision which have related meaning in both settings [9, 12, 40]. Accordingly, the nested sequence of spaces underlying the decomposition into disjoint frequency bands is thought of being generated bottom-up from a coarse base mesh up to finer and finer resolutions. This implies that subdivision connectivity is mandatory on higher levels of detail. Not only the mesh has to consist of large regular regions with isolated extra-ordinary vertices in between. Additionally, we have to make sure that the topological distance between the singularities is the same for every pair of neighboring singularities and this topological distance has to be a power of 2.

Such special topological requirements prevent the schemes from being applicable to arbitrary input meshes. Global remeshing and resampling is necessary to obtain a proper hierarchy which gives rise to alias-errors and requires involved computations [7].

Luckily, the restricted topology is not necessary to define different levels of resolution or approximation for a triangle mesh. In the literature on mesh decimation we find many examples for hierarchies built on arbitrary meshes [11, 15, 20, 24, 27, 31, 35]. The key is always to build the hierarchy top-down by eliminating vertices from the current mesh (*incremental reduction, progressive meshes*). Running a mesh decimation algorithm, we can stop, e.g., every time a certain percentage of the vertices is removed. The intermediate meshes can be used as a level-of-detail representation [15, 23].

In both cases, i.e., the bottom-up or the top-down generation of nested (vertex-) grids, the multi-resolution concept is rigidly attached to topological entities. This makes sense if hierarchies are merely used to reduce the complexity of the representation. In the context of multi-resolution modeling, however, we want the hierarchy not necessarily to rate meshes according to their *coarseness* but rather according to their *smoothness* (cf. Fig 1).

We will use multi-resolution hierarchies for two purposes. First we want to derive highly efficient algorithms for mesh optimization. In Section 3 we will see that topologically reduced meshes are the key to significantly increase the performance (levels of coarseness). On the other hand, we want to avoid any restrictions that are imposed by topological peculiarities. In particular, when interactively modifying a triangle mesh, we do not want any alignment. The *support* of a modification should have no influence on *where* this modification can be applied (levels of smoothness).

To describe the different set-ups for multi-resolution representation uniformly, we define a generic decomposition scheme $\mathbf{A} = (\mathbf{A}_\Phi | \mathbf{A}_\Psi)^T$ (*analysis*) as a general procedure that transforms a given mesh $\mathcal{M}_i$ into a coarser/smoother one $\mathcal{M}_{i-1} = \mathbf{A}_\Phi \mathcal{M}_i$ plus detail coefficients $\mathcal{D}_{i-1} = \mathbf{A}_\Psi \mathcal{M}_i$. In the standard wavelet setting the cardinalities satisfy $\#\mathcal{D}_{i-1} + \#\mathcal{M}_{i-1} = \#\mathcal{M}_i$ since decomposition is a proper basis transform.

If a (bi-orthogonal) wavelet basis is not known, we have to store more detail information ($\#\mathcal{D}_{i-1} + \#\mathcal{M}_{i-1} > \#\mathcal{M}_i$) since the reconstruction operator $\mathbf{A}^{-1}$ might be computationally expensive or not even uniquely defined. Well-known examples for this kind of decomposition with extra detail coefficients are the Laplacian-pyramid type of representation in [40] and the progressive mesh representation [15].

When $\mathbf{A}_\Phi$ is merely a smoothing operator which does not change the topological mesh structure of $\mathcal{M}_i$ we have $\mathbf{A}_\Psi = \mathbf{I}d - \mathbf{A}_\Phi$ and $\#\mathcal{D}_{i-1} = \#\mathcal{M}_{i-1} = \#\mathcal{M}_i$.

2.1 Local Frames

In a multi-resolution representation of a geometric object $\mathcal{M} = \mathcal{M}_k$, the detail coefficients $\mathcal{D}_{i-1}$ describe the difference between two approximations $\mathcal{M}_{i-1}$ and $\mathcal{M}_i$ having different levels of detail. For parametric surfaces, the detail coefficients, i.e., the spatial location of the vertices in $\mathcal{M}_i$ have to be encoded relative to the local geometry of the coarser approximation $\mathcal{M}_{i-1}$. This is necessary since modifications on the coarser level should have an intuitive effect on the geometric features from finer scales.

First proposed by [10] it has become standard to derive local coordinate frames from the partial derivative information of the coarse representation $\mathcal{M}_{i-1}$. Since we do not assume the existence of any global structure or auxiliary information in the sequence of meshes $\mathcal{M}_i$, we have to rely on intrinsic geometric properties of the triangles themselves. Depending on the intended application we assign the local frames to the triangles or the vertices of $\mathcal{M}_{i-1}$. A detail vector is then defined by three coordinate values $[u, v, n]$ plus an index i identifying the affine frame $F_i = [\mathbf{p}_i, U_i, V_i, N_i]$ with respect to which the coordinates are given.

2.1.1 Vertex-based frames

We can use any heuristic to estimate the normal vector N_i at a vertex $\mathbf{p}_i$ in a polygonal mesh, e.g., taking the average of the adjacent triangle normals. The vector $U_i = E - (E^T N_i) N_i$ is obtained by pro-

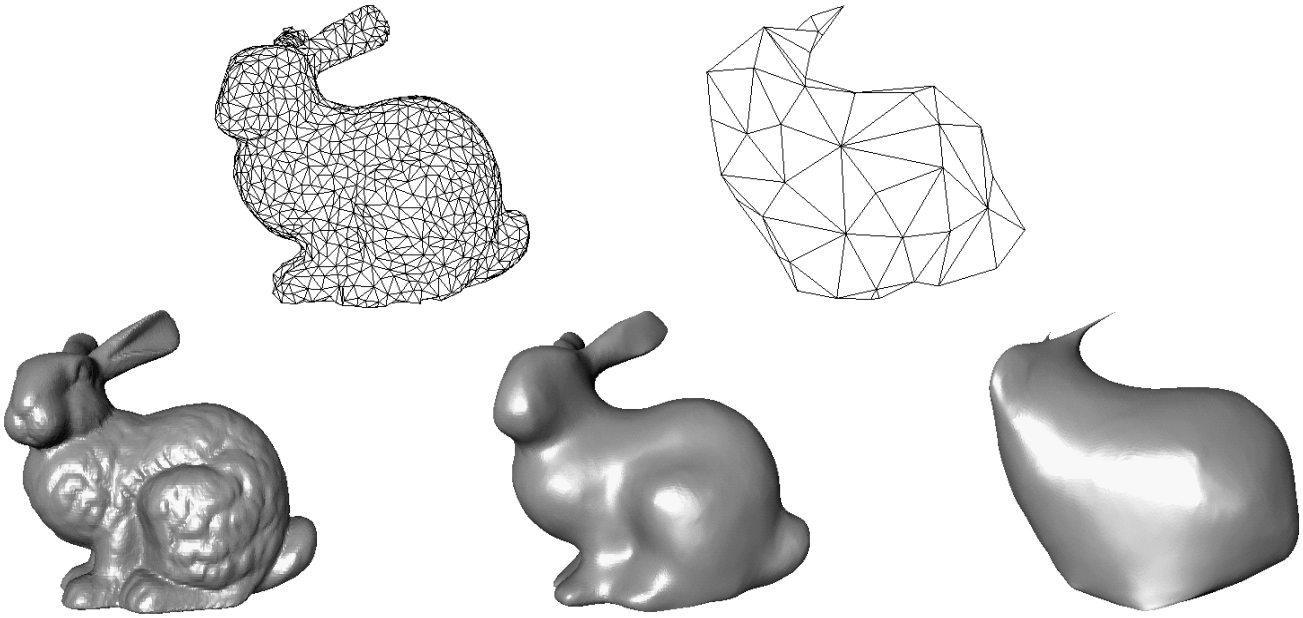


Figure 1: The well-known Stanford-Bunny. Although the original mesh does not have subdivision connectivity, mesh decimation algorithms easily generate a hierarchy of topologically simplified meshes. On the other hand, multi-resolution modeling also requires hierarchies of differently *smooth* approximations. Notice that the meshes in the lower row have identical connectivity.

jecting any adjacent edge E into the tangent plane and $V_i := N_i \times U_i$. The data structure for storing the mesh $\mathcal{M}_{i-1}$ has to make sure that E is uniquely defined, e.g. as the first member in a list of neighbors.

2.1.2 Face-based frames

It is tempting to simply use the local frame which is given by two triangle edges and their cross product. However, this will not lead to convincing detail reconstruction after modifying the coarser level. The reason for this is that the local frames would be rigidly attached to one coarse triangle. In fact, tracing the dependency over several levels of detail shows that the original mesh is implicitly partitioned into sub-meshes being assigned to the same coarse triangle T. Applying a transformation to T implies the same transformation for all vertices being defined relative to T. This obviously leads to artifacts between neighboring sub-meshes in the fine mesh.

A better choice is to use local low order polynomial interpolants or approximants that depend on more than one single triangle. Let $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$ be the vertices of a triangle $T \in \mathcal{M}_{i-1}$ and $\mathbf{p}_3$, $\mathbf{p}_4$, and $\mathbf{p}_5$ be the opposite vertices of the triangles adjacent to T (cf. Fig. 2). To construct a quadratic polynomial

$$\mathbf{F}(u,v) \;=\; \mathbf{f} + u\mathbf{f}_u + v\mathbf{f}_v + \frac{u^2}{2}\mathbf{f}_{uu} + uv\mathbf{f}_{uv} + \frac{v^2}{2}\mathbf{f}_{vv}$$

approximating the $\mathbf{p}_i$ we have to define a parameterization first. Note that the particular choice of this parameterization controls the quality of the approximant. Since we want to take the geometric constellation of the $\mathbf{p}_i$ into account, we define a parameterization by projecting the vertices into the supporting plane of T.

Exploiting the invariance of the polynomial interpolant with respect to affine re-parameterizations, we can require $\mathbf{F}(0,0) := \mathbf{p}_0$, $\mathbf{F}(1,0) := \mathbf{p}_1$, and $\mathbf{F}(0,1) := \mathbf{p}_2$ which implies

$$\begin{aligned} \mathbf{f} &= \mathbf{p}_0 \\ \mathbf{f}_u &= \mathbf{p}_1 - \mathbf{p}_0 - \tfrac{1}{2}\mathbf{f}_{uu} \\ \mathbf{f}_v &= \mathbf{p}_2 - \mathbf{p}_0 - \tfrac{1}{2}\mathbf{f}_{vv}. \end{aligned} \qquad (1)$$

Let the vertices $\mathbf{p}_3$, $\mathbf{p}_4$, and $\mathbf{p}_5$ be projected to (u_3,v_3), (u_4,v_4), and (u_5,v_5) according to the frame $[\mathbf{p}_0,\mathbf{p}_1,\mathbf{p}_2]$. To additionally stabilize the interpolation scheme, we introduce a tension parameter $\tau \in [0,1]$ which trades approximation error at $\mathbf{p}_3$, $\mathbf{p}_4$, and $\mathbf{p}_5$ for minimizing the bending energy $\mathbf{f}_{uu}^2 + 2\mathbf{f}_{uv}^2 + \mathbf{f}_{vv}^2$. Using (1) we obtain

$$\begin{pmatrix} \frac{1}{2}u_3(u_3-1) & u_3v_3 & \frac{1}{2}v_3(v_3-1) \\ \frac{1}{2}u_4(u_4-1) & u_4v_4 & \frac{1}{2}v_4(v_4-1) \\ \frac{1}{2}u_5(u_5-1) & u_5v_5 & \frac{1}{2}v_5(v_5-1) \\ \tau & 0 & 0 \\ 0 & 2\tau & 0 \\ 0 & 0 & \tau \end{pmatrix} \begin{pmatrix} \mathbf{f}_{uu} \\ \mathbf{f}_{uv} \\ \mathbf{f}_{vv} \end{pmatrix} = \begin{pmatrix} (\mathbf{p}_3-\mathbf{p}_0)+u_3(\mathbf{p}_0-\mathbf{p}_1)+v_3(\mathbf{p}_0-\mathbf{p}_2) \\ (\mathbf{p}_4-\mathbf{p}_0)+u_4(\mathbf{p}_0-\mathbf{p}_1)+v_4(\mathbf{p}_0-\mathbf{p}_2) \\ (\mathbf{p}_5-\mathbf{p}_0)+u_5(\mathbf{p}_0-\mathbf{p}_1)+v_5(\mathbf{p}_0-\mathbf{p}_2) \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

which has to be solved in a least squares sense.

To compute the detail coefficients $[\hat{u},\hat{v},h]$ for a point $\mathbf{q}$ with respect to T, we start from the center $(u,v) = (\frac{1}{3},\frac{1}{3})$ and simple Newton iteration steps $(u,v) \leftarrow (u,v) + (\triangle u, \triangle v)$ with $\mathbf{d} = \mathbf{q} - \mathbf{F}(u,v)$ and

$$\begin{pmatrix} \mathbf{F}_u^T\mathbf{F}_u & \mathbf{F}_u^T\mathbf{F}_v \\ \mathbf{F}_u^T\mathbf{F}_v & \mathbf{F}_v^T\mathbf{F}_v \end{pmatrix} \begin{pmatrix} \triangle u \\ \triangle v \end{pmatrix} = \begin{pmatrix} \mathbf{F}_u^T\mathbf{d} \\ \mathbf{F}_u^T\mathbf{d} \end{pmatrix}$$

quickly converge to the point $\mathbf{F}(\hat{u},\hat{v})$ with the detail vector $\mathbf{d}$ perpendicular to the surface $\mathbf{F}(u,v)$. The third coefficient is then $h = \text{sign}(\mathbf{d}^T(\mathbf{F}_u \times \mathbf{F}_v))\,\|\mathbf{d}\|$.

Although the parameter values $(\hat{u},\hat{v})$ can lie outside the unit triangle (which occasionally occurs for extremely distorted configurations) the detail coefficient $[\hat{u},\hat{v},h]$ is still well-defined and reconstruction works. Notice that the scheme might produce counter-intuitive results if the maximum dihedral angle between T and one of its neighbors becomes larger than $\frac{\pi}{2}$. In this case the parameter-

ization for $\mathbf{p}_3$, $\mathbf{p}_4$, and $\mathbf{p}_5$ could be derived by rotation about T's edges instead of projection.

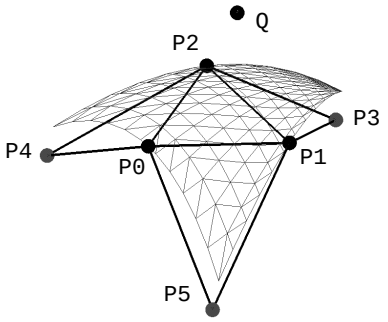


Figure 2: Vertex labeling for the construction of a local frame.

Obviously, the detail coefficient $[\hat{u}, \hat{v}, h]$ is not coded with respect to a local frame in the narrow sense. However, it has a similar semantics. Recovering the vertex position $\mathbf{q}'$ requires to construct the approximating polynomial $\mathbf{F}'(u,v)$ for the possibly modified vertices $\mathbf{p}'_i$, evaluate at $(\hat{u}, \hat{v})$ and move in normal direction by h. The distance h is a measure for the "size" of the detail.

In our current implementation on a SGI R10000/195 MHz workstation the analysis $\mathbf{q} \to [\hat{u}, \hat{v}, h]$ takes about $20\mu S$ while the reconstruction $[\hat{u}, \hat{v}, h] \to \mathbf{q}$ takes approximately $8\mu S$. Since a progressive mesh representation introduces two triangles per vertex split, this means that for the reconstruction of a mesh with 10^5 triangles, the computational overhead due to the local frame representation is less than half a second.

2.2 Decomposition and reconstruction

To complete our equipment for the multi-resolution set-up we have to define the decomposition and reconstruction operations which separate the high-frequency detail from the low-frequency shape and eventually recombine the two to recover the original mesh. We apply different strategies depending on whether decomposition generates a coarser approximation of the original geometry or a smoother approximation.

In either case the decomposition operator $\mathbf{A}$ is applied to the original mesh $\mathcal{M}_i$ and the details $\mathcal{D}_{i-1}$ are coded in local frame coordinates with respect to $\mathcal{M}_{i-1}$. Since the reconstruction is an extrapolation process, it is numerically unstable. To stabilize the operation we have to make the details as small as possible, i.e., when encoding the spatial position of a point $\mathbf{q} \in \mathbb{R}^3$ we pick that local frame on $\mathcal{M}_{i-1}$ which is closest to $\mathbf{q}$.

Usually the computational complexity of generating the detail coefficients is higher than the complexity of the evaluation during reconstruction. This is an important feature since for interactive modeling the (dynamic) reconstruction has to be done in real-time while the requirements for the (static) decomposition are not as demanding.

2.2.1 Mesh decimation based decomposition

When performing an incremental mesh decimation algorithm, each reduction step removes one vertex and retriangulates the remaining hole [15, 31]. We use a simplified version of the algorithm described in [20] that controls the reduction process in order to optimize the fairness of the coarse mesh while keeping the global approximation error below a prescribed tolerance.

The basic topological operation is the *half edge collapse* which shifts one vertex $\mathbf{p}$ into an adjacent vertex $\mathbf{q}$ and removes the two degenerate triangles. In [20] a fast algorithm is presented to determine that triangle T in the neighborhood of the collapse which lies closest to the removed vertex $\mathbf{p}$. The position of $\mathbf{p}$ is then coded with respect to the local frame associated with this triangle.

The inverse operation of an edge collapse is the *vertex split* [15]. Since during reconstruction the vertices are inserted in the reverse order of their removal, it is guaranteed that, when $\mathbf{p}$ is inserted, the topological neighborhood looks the same as when it was deleted and hence the local frame to transform the detail coefficient for $\mathbf{p}$ back into world coordinates is well-defined.

During the iterative decimation, each intermediate mesh could be considered as one individual level of detail approximation. However, if we want to define disjoint frequency bands, it is reasonable to consider a whole sub-sequence of edge collapses as one atomic transformation from one level $\mathcal{M}_i$ to $\mathcal{M}_{i-1}$.

There are several criteria to determine which levels mark the boundaries between successive frequency bands. One possibility is to simply define $\mathcal{M}_i$ to be the coarsest mesh that still keeps a maximum tolerance of less than some ε_i to the original data. Alternatively we can require the number of vertices in $\mathcal{M}_{i-1}$ to be a fixed percentage of the number of vertices in $\mathcal{M}_i$. This simulates the geometric decrease of complexity known from classical multi-resolution schemes. We can also let the human user decide when a significant level of detail is reached by allowing her to browse through the sequence of incrementally reduced meshes.

In order to achieve optimal performance with the multi-level smoothing algorithm described in the next section, we decided in our implementation to distribute the collapses evenly over the mesh: When a collapse $\mathbf{p} \to \mathbf{q}$ is performed, all vertices adjacent to $\mathbf{q}$ are locked for further collapsing (independent set of collapses). If no more collapses are possible, the current mesh defines the next level of detail and all vertices are un-locked. One pass of this reduction scheme removes about 25% of the vertices in average.

2.2.2 Mesh smoothing based decomposition

For multi-resolution *modeling* we have to separate high frequency features from the global shape in order to modify both individually. Reducing the mesh complexity cannot help in this case since coarser meshes do no longer have enough degrees of freedom to be smooth, i.e., to have small angles between adjacent triangles. Hence, the decomposition operator $\mathbf{A}_\Phi$ becomes a mere smoothing operator that reduces the discrete bending energy in the mesh without changing the topology (cf. Section 3).

A natural way to define the detail coefficients would be to store the difference vectors between the original vertex position $\mathbf{q}$ and the shifted position $\mathbf{q}'$ with respect to the local frame defined at $\mathbf{q}'$. However, in view of numerical stability this choice is not optimal. Depending on the special type of smoothing operator $\mathbf{A}_\Phi$ the vertices can move "within" the surface such that another vertex $\mathbf{p}' \in \mathcal{M}_{i-1} = \mathbf{A}_\Phi \mathcal{M}_i$ could lie closer to $\mathbf{q}$ than $\mathbf{q}'$ (cf. Fig. 3).

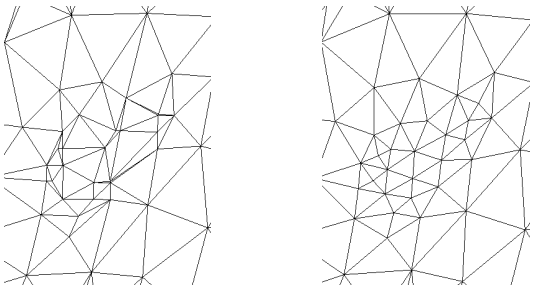


Figure 3: Although the bending energy minimizing smoothing operator $\mathbf{A}_\Phi$ is applied to a *plane* triangulation, the vertices are moved within the plane since linear operators always do the fairing with respect to a specific parameterization (cf. Section 3).

To stabilize the reconstruction, i.e., to minimize the length of the detail vectors, we apply a simple local search procedure to find the

nearest vertex $\mathbf{p}' \in \mathcal{M}_{i-1}$ to $\mathbf{q}$ and express the detail vector with respect to the local frame at $\mathbf{p}'$ or one of its adjacent triangles. This searching step does not noticeably increase the computation time (which is usually dominated by the smoothing operation $\mathbf{A}_\Phi$) but leads to much shorter detail vectors (cf. Fig 4).

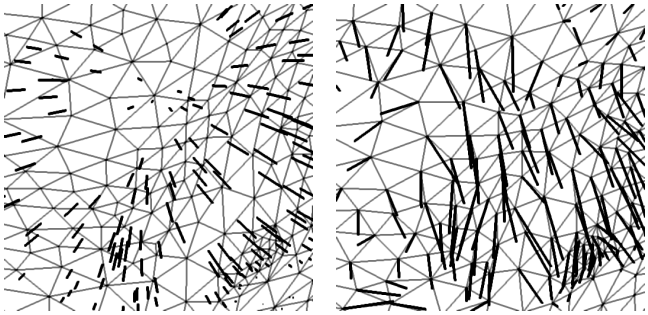


Figure 4: The shortest detail vectors are obtained by representing the detail coefficients with respect to the nearest local frame (left) instead of attaching the detail vectors to the topologically corresponding original vertices.

3 Discrete fairing

From CAGD it is well-known that constrained energy minimization is a very powerful technique to generate high quality surfaces [3, 13, 25, 28, 37]. For efficiency, one usually defines a simple quadratic energy functional $\mathcal{E}(f)$ and searches among the set of functions satisfying prescribed interpolation constraints for that function f which minimizes $\mathcal{E}$.

Transferring the continuous concept of energy minimization to the discrete setting of triangle mesh optimization leads to the discrete fairing approach [19, 38]. Local polynomial interpolants are used to estimate derivative information at each vertex by divided difference operators. Hence, the differential equation characterizing the functions with minimum energy is discretized into a linear system for the vertex positions.

Since this system is global and sparse, we apply iterative solving algorithms like the Gauß-Seidel-scheme. For such algorithms one iteration step merely consists in the application of a simple local averaging operator. This makes discrete fairing an easy accessible technique for mesh optimization.

3.1 The umbrella-algorithm

The most prominent energy functionals that are used in the theory and practice of surface design are the membrane energy

$$\mathcal{E}_M(f) := \int f_u^2 + f_v^2 \tag{2}$$

which prefers functions with smaller surface area and the thin plate energy

$$\mathcal{E}_{TP}(f) := \int f_{uu}^2 + 2\, f_{uv}^2 + f_{vv}^2 \tag{3}$$

which punishes strong bending. The variational calculus leads to simple characterizations of the corresponding minimum energy surfaces

$$\triangle f \;=\; f_{uu} + f_{vv} \;=\; 0 \tag{4}$$

or

$$\triangle^2 f \;=\; f_{uuuu} + 2\, f_{uuvv} + f_{vvvv} \;=\; 0 \tag{5}$$

respectively. Obviously, low degree polynomials satisfy both differential equations and hence appropriate (Dirichlet-) boundary conditions have to be imposed which make the semi-definite functionals $\mathcal{E}_M$ and $\mathcal{E}_{TP}$ positive-definite.

The discrete fairing approach discretizes either the energy functionals (2–3) [19, 38] or the corresponding Euler-Lagrange equations (4–5) [17, 36] by replacing the differential operators with divided difference operators. To construct these operators, we have to choose an appropriate parameterization in the vicinity of each vertex. In [38] for example a discrete analogon to the exponential map is chosen. In [17] the *umbrella-algorithm* is derived by choosing a symmetric parameterization

$$(u_i, v_i) \;:=\; \Big(\cos(2\pi\frac{i}{n}), \sin(2\pi\frac{i}{n})\Big), \quad i = 0, \ldots, n-1 \tag{6}$$

with n being the valence of the center vertex $\mathbf{p}$ (cf. Fig 5). This parameterization does not adapt to the local geometric constellation but it simplifies the construction of the corresponding difference operators which are otherwise obtained by solving a Vandermonde system for local polynomial interpolation. With the special parameterization (6) the discrete analogon of the Laplacian $\triangle f$ turns out to be the umbrella-operator

$$\mathcal{U}(\mathbf{p}) \;=\; \frac{1}{n}\sum_{i=0}^{n-1} \mathbf{p}_i \;-\; \mathbf{p}$$

with $\mathbf{p}_i$ being the direct neighbors of $\mathbf{p}$ (cf. Fig. 5). The umbrella-operator can be applied recursively leading to

$$\mathcal{U}^2(\mathbf{p}) \;=\; \frac{1}{n}\sum_{i=0}^{n-1} \mathcal{U}(\mathbf{p}_i) \;-\; \mathcal{U}(\mathbf{p})$$

as a discretization of $\triangle^2 f$.

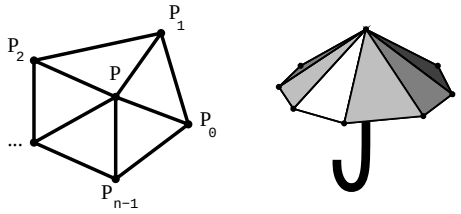


Figure 5: To compute the discrete Laplacian, we need the 1-neighborhood of a vertex $\mathbf{p}$ ($\rightarrow$ umbrella-operator).

The boundary conditions are imposed to the discrete problem by freezing certain vertices. When minimizing the discrete version of $\mathcal{E}_M$ we hold a closed boundary polygon fixed and compute the membrane that is spanned in between. For the minimization of $\mathcal{E}_{TP}$ we need two rings of boundary vertices, i.e., we keep a closed strip of triangles fixed. This imposes a (discrete) C^1 boundary condition to the optimization problem (cf. Fig 6). All internal vertices can be moved freely to minimize the global energy. The properly chosen boundary conditions imply positive-definiteness of the energy functional and guarantee the convergence of the iterative solving algorithm [29].

The characteristic (linear) system for the corresponding unconstrained minimization problem has rows $\mathcal{U}(\mathbf{p}_i) = 0$ or $\mathcal{U}^2(\mathbf{p}_i) = 0$ respectively for the free vertices $\mathbf{p}_i$. An iterative solving scheme approaches the optimal solution by solving each row of the system separately and cycling through the list of free vertices until a stable solution is reached. In case of the membrane energy $\mathcal{E}_M$ this leads to the local update rule

$$\mathbf{p}_i \;\leftarrow\; \mathbf{p}_i + \mathcal{U}(\mathbf{p}_i) \tag{7}$$

and for the thin plate energy $\mathcal{E}_{TP}$, we obtain

$$\mathbf{p}_i \leftarrow \mathbf{p}_i - \frac{1}{\nu}\,\mathcal{U}^2(\mathbf{p}_i) \tag{8}$$

with the "diagonal element"

$$\nu = 1 + \frac{1}{n_i}\sum_j \frac{1}{n_{i,j}}$$

where n_i and $n_{i,j}$ are the valences of the center vertex $\mathbf{p}_i$ and its jth neighbor respectively.

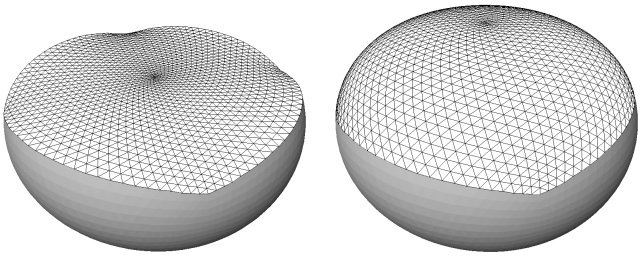


Figure 6: A closed polygon or a closed triangle strip provide C^0 or C^1 boundary conditions for the discrete fairing. On the left the triangle mesh minimizes $\mathcal{E}_M$ on the right it minimizes $\mathcal{E}_{TP}$.

Although the rule (8) can be implemented recursively, the performance is optimized if we use a two step process where all umbrella vectors $\mathcal{U}(\mathbf{p}_i)$ are computed in a first pass and $\mathcal{U}^2(\mathbf{p}_i)$ in the second. This avoids double computation but it also forces us to use in fact a plain Jacobi-solver since no intermediate updates from neighboring vertices can be used. However the $(n+2):2$ speed-up for a vertex with valence n amortizes the slower convergence of Jacobi compared to Gauß-Seidel.

3.2 Connection to Taubin's signal processing approach

The local update operator (7) in the iterative solving scheme for constrained energy minimization is exactly the Laplace smoothing operator proposed by Taubin in [34] where he derived it (also in the context of mesh smoothing) from a filter formalism based on generalized Fourier analysis for arbitrary polygonal meshes. In his paper, Taubin starts with a matrix version of the scaled update rule (7)

$$[\mathbf{p}_i'] := (I + \lambda\,\mathcal{U})\,[\mathbf{p}_i]$$

where λ is a damping factor and formally rewrites it by using a transfer function notation

$$f(k) := 1 - \lambda k$$

with respect to the eigenbasis of the (negative) Laplace operator. Since no proper boundary conditions are imposed, the continued filtering by $f(k)$ leads to severe shrinking and hence he proposes combined filters

$$f(k) := (1-\lambda k)\,(1-\mu k) \tag{9}$$

where λ and μ are set in order to minimize the shrinking. A feasible heuristic is to choose a *pass-band frequency*

$$k_{PB} = \frac{1}{\lambda} + \frac{1}{\mu} \in [0.01 \ldots 0.1]$$

and set λ and μ while observing the stability of the filter.

Obviously, the update rule for the difference equation $\mathcal{U}(\mathbf{p}_i) = 0$ which characterizes meshes with minimum membrane energy corresponds to a special low-pass filter with transfer function $f_{\mathcal{U}}(k) = (1-k)$. For the minimization of the total curvature, characterized by $\mathcal{U}^2(\mathbf{p}_i) = 0$, the iteration rule (8) can be re-written in transfer function notation as

$$f_{\mathcal{U}^2}(k) = (1 - \frac{1}{\nu}k^2) = (1 + \frac{1}{\sqrt{\nu}}k)\,(1 - \frac{1}{\sqrt{\nu}}k)$$

which corresponds to a combined Laplace filter of the form (9) with pass-band frequency $k_{PB} = 0$. Although this is not optimal for reducing the shrinking effect, we observe that the transfer function happens to have a vanishing derivative at $k = 0$. From signal processing theory it is known that this characterizes maximal smoothness [26], i.e., among the two step Laplace filters, the $\mathcal{U}^2$-filter achieves optimal smoothing properties. To stabilize the filter we might want to introduce a damping factor $0 < \sigma < \frac{1}{2}\nu$ into the update-rule

$$\mathbf{p}_i \leftarrow \mathbf{p}_i - \frac{\sigma}{\nu}\,\mathcal{U}^2(\mathbf{p}_i)$$

3.3 Multi-level smoothing

A well-known negative result from numerical analysis is that straight forward iterative solvers like the Gauß-Seidel scheme are not appropriate for large sparse problems [32]. More sophisticated solvers exploit knowledge about the *structure* of the problem. The important class of multi-grid solvers achieve linear running times in the number of degrees of freedom by solving the same problem on grids with different step sizes and combining the approximate solutions [14].

For difference (= discrete differential) equations of elliptic type the Gauß-Seidel iteration matrices have a special eigenstructure that causes high frequencies in the error to be attenuated very quickly while for lower frequencies no practically useful rate of convergence can be observed. Multi-level schemes hence solve a given problem on a very coarse scale first. This solution is used to predict initial values for a solution of the same problem on the next refinement level. If these predicted values have only small deviations from the true solution in low-frequency sub-spaces, then Gauß-Seidel performs well in reducing the high-frequency error. The alternating refinement and smoothing leads to highly efficient variational subdivision schemes [19] which generate fair high-resolution meshes with a rate of several thousand triangles per second (linear complexity!).

As we saw in Section 2, the bottom-up way to build multi-resolution hierarchies is just one of two possibilities. To get rid of the restriction that the uniform multi-level approach to fairing cannot be applied to arbitrary meshes, we generate the hierarchy top-down by incremental mesh decimation.

A complete V-cycle multi-grid solver recursively applies operators $\Phi_i = \Psi P \Phi_{i-1} R \Psi$ where the first (right) Ψ is a generic (pre-)smoothing operator — a Gauß-Seidel scheme in our case. R is a restriction operator to go one level coarser. This is where the mesh decimation comes in. On the coarser level, the same scheme is applied recursively, Φ_{i-1}, until on the coarsest level the number of degrees of freedom is small enough to solve the system directly (or any other stopping criterion is met). On the way back-up, the prolongation operator P inserts the previously removed vertices to go one level finer again. P can be considered as a non-regular subdivision operator which has to predict the positions of the vertices in the next level's solution. The re-subdivided mesh is an approximative solution with mostly high frequency error. (Post-)smoothing by some more iterations Ψ removes the noise and yields the final solution.

Fig 7 shows the effect of multi-level smoothing. On the left you see the original bunny with about 70K triangles. In the center left,

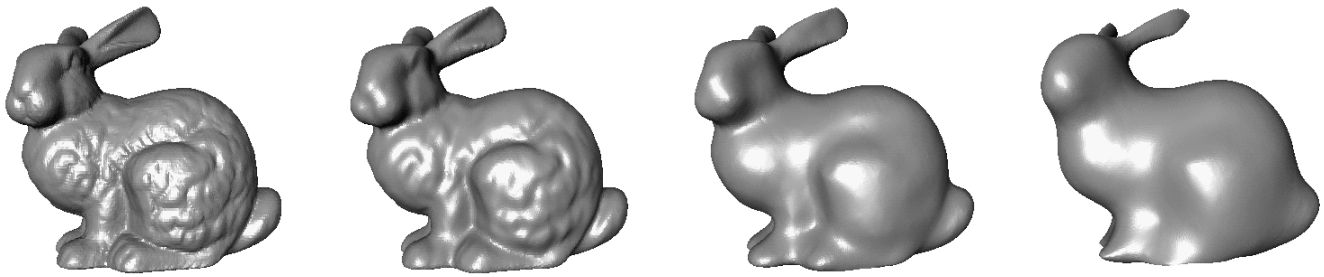


Figure 7: Four versions of the Stanford bunny. On the left the original data set. In the center left the same object after 200 iterations of the non-shrinking Laplace-filter. On the center right and far right the original data set after applying the multi-level umbrella filter with three or six levels respectively.

we applied the Laplace-filter proposed in [34] with $\lambda = 0.6307$ and $\mu = -0.6732$. The iterative application of the local smoothing operator

$$\mathbf{p}_i \leftarrow \mathbf{p}_i + [\lambda|\mu]\ \mathcal{U}(\mathbf{p}_i) \tag{10}$$

removes the highest frequency noise after a few iterations but does not have enough impact to flatten out the fur even after several hundred iterations. On the right you see the meshes after applying a multi-level smoothing with the following schedule: Hierarchy levels are generated by incremental mesh decimation where each level has about 50% of the next finer level's vertices. The pre-smoothing rule (8) is applied twice on every level before going downwards and five times after coming back up. On the center right model we computed a three level V-cycle and on the far right model a six level V-cycle. Notice that the computation time of the multi-level filters (excluding restriction and prolongation) corresponds to about $(2+5)(1+0.5+0.5^2+\ldots) < 14$ double-steps of the one-level Laplace-Filter (10).

3.4 Geometric filtering

The bunny example in Fig. 7 is well suited for demonstrating the effect of multi-level smoothing but we did not impose any boundary conditions and thus we applied the smoothing as a mere filter and not as a solving scheme for a well-posed optimization problem. This is the reason why we could use the number of levels to control the impact of the smoothing scheme on the final result. For constrained optimization, it does not make any sense to stop the recursion after a fixed number of decimation levels: we always reduce the mesh down to a small fixed number of triangles. Properly chosen boundary condition will ensure the convergence to the true solution and prevent the mesh from shrinking.

Nevertheless, we can exploit the effect observed in Fig. 7 to define more sophisticated geometric low-pass filters. Since the support of the Laplace-filters is controlled by the neighborhood relation in the underlying mesh, the smoothing characteristics are defined relative to a "topological wavelength". Noise which affects every other vertex is removed very quickly independent from the length of the edges while global distortions affecting a larger sub-mesh are hardly reduced. For *geometric* filters however we would like to set the pass-band frequency in terms of Euclidian distances by postulating that all geometric features being smaller than some threshold ε are considered as noise and should be removed.

Such filters can be implemented by using an appropriate mesh reduction scheme that tends to generate intermediate meshes with strong coherence in the length of the edges. In [20] we propose a mesh decimation scheme that rates the possible edge collapses according to some generic fairness functional. A suitable objective function for our application is to maximize the *roundness* of triangles, i.e., the ratio of its inner circle radius to its longest edge. If the mesh decimation scheme prefers those collapses that improve the global roundness, the resulting meshes tend to have only little variance in the lengths of the edges. For the bunny example, we can keep the standard deviation from the average edge length below one percent for incremental decimation down to about 5K triangles.

By selecting the lowest level $\mathcal{M}_0$ down to which the V-cycle multi-level filtering iterates, we set the threshold $\varepsilon = \varepsilon(\mathcal{M}_0)$ for detail being removed by the multi-level smoothing scheme. The thresholding works very well due to the strong local and poor global convergence of the iterative update rule (8). Fig. 8 shows the base meshes for the multi-level smoothing during the computation of the two right bunnies of Fig. 7.

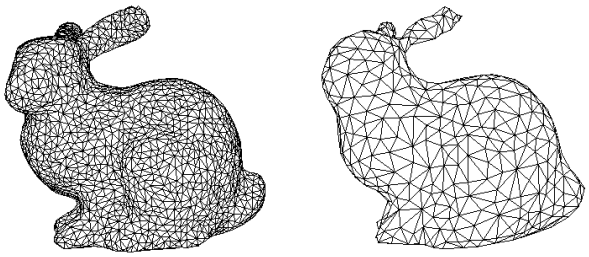


Figure 8: Base meshes where the V-cycle recursion stopped when generating the right models in Fig. 7. The final meshes do not loose significant detail (watch the silhouette). Notice how in the left example the fur is removed but the bunny's body preserved while in the right example the leg and the neck start to disappear.

4 Multi-resolution modeling on triangle meshes

In this section we describe a flexible and intuitive multi-resolution mesh modeling metaphor which exploits the techniques presented in the last two sections. As we discussed in the introduction, our goal is to get rid of topological restrictions for the mesh but also to get rid of difficulties emerging from the alignment of the basis functions in a hierarchical representation and the rigid shape of the basis function's support.

From a designer's point of view, we have to distinguish three *semantic* levels of detail. These levels are defined relative to a specific modeling operation since, of course, in a multi-resolution environment the features that are detail in a (global) modification become the global shape for a minute adjustment.

- The *global shape* is that part of the geometry that is the subject of the current modification. Intuitively, the designer selects a piece of the global shape and applies a transformation to it.
- The *structural detail* are features that are too small to be modified by hand but still represent actual geometry. This detail should follow the modification applied to the global shape in a

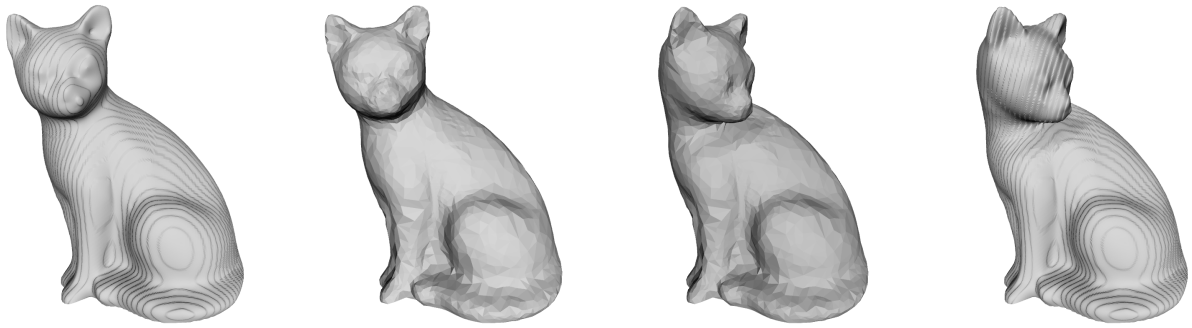


Figure 9: The wooden cat model $\mathcal{M}_k$ (178K triangles, left) is in progressive mesh representation. The high resolution is necessary to avoid alias errors in the displacement texture. The center left model $\mathcal{M}_i$ (23K triangles) is extracted by stopping the mesh reduction when a prescribed complexity is reached. On this level interactive mesh modification is done which yields the model $\mathcal{M}_i'$ (center right). The final result $\mathcal{M}_k'$ (right) is obtained by running the reconstruction on the modified mesh.

geometrically intuitive manner. The preservation of structural detail during interactive modeling is crucial for plausible visual feed-back (cf. the eyes and ears of the wooden cat model in Fig. 9).

- The *textural detail* does not really describe geometric shape. It is necessary to let the surface appear more realistic and is often represented by displacement maps [21]. In high quality mesh models it is the source for the explosive increase in complexity (cf. the wood texture in Fig. 9).

Having identified these three semantic levels of detail, we suggest a mesh modeling environment which provides flexible mesh modification functionality and allows the user to adapt the mesh complexity to the available hardware resources.

In an off-line preprocessing step, an incremental mesh decimation algorithm is applied and the detail coefficients are stored with respect to local frames as explained in Section 2.2.1. This transforms the highly complex input model into a progressive-mesh type multi-resolution representation without any remeshing or resampling. The representation allows the user to choose an appropriate number of triangles for generating a mesh model that is fine enough to contain at least all the structural detail but which is also coarse enough to be modified in realtime. This pre-process removes the textural detail prior to the actual interactive mesh modification.

Suppose the original mesh model $\mathcal{M}_k$ is transformed into the progressive mesh sequence $[\mathcal{M}_k, \ldots, \mathcal{M}_0]$ with $\mathcal{M}_0$ being the coarsest base mesh. If the user picks the mesh $\mathcal{M}_i$ and applies modifications then this invalidates the subsequence $[\mathcal{M}_{i-1}, \ldots, \mathcal{M}_0]$. If the working resolution is to be reduced afterwards to $\mathcal{M}_j$ $(j < i)$ then the intermediate meshes have to be recomputed by online mesh decimation. The textural detail encoded in the subsequence $[\mathcal{M}_k, \ldots, \mathcal{M}_{i+1}]$ however remains unchanged since it is stored with respect to local frames such that reconstruction starting from a modified mesh $\mathcal{M}_i'$ leads to the intended result $\mathcal{M}_k'$. Fig. 9 shows an example of this procedure.

4.1 Interactive mesh modeling by discrete fairing

The most important feature in the proposed multi-resolution mesh modeling environment is the modification functionality itself (*modeling metaphor*) which hides the mesh connectivity to the designer.

The designer starts by marking an arbitrary region on the mesh $\mathcal{M}_i$. In fact, she picks a sequence of surface points (not necessarily vertices) on the triangle mesh and these points are connected either by geodesics or by projected lines. The strip of triangles $\mathcal{S}$ which are intersected by the geodesic (projected) boundary polygon separates an interior region $\mathcal{M}_*$ and an exterior region $\mathcal{M}_i \setminus (\mathcal{M}_* \cup \mathcal{S})$. The interior region $\mathcal{M}_*$ is to be affected by the following modification.

A second polygon (not necessarily closed) is marked within the first one to define the *handle*. The semantics of this arbitrarily shaped handle is quite similar to the handle metaphor in [37]: when the designer moves or scales the virtual tool, the same geometric transformation is applied to the rigid handle and the surrounding mesh $\mathcal{M}_*$ follows according to a constrained energy minimization principle.

The freedom to define the boundary strip $\mathcal{S}$ and the handle geometry allows the designer to build "custom tailored" basis functions for the intended modification. Particularly interesting is the definition of a *closed* handle polygon which allows to control the characteristics of a bell-shaped dent: For the same region $\mathcal{M}_*$, a tiny ring-shaped handle in the middle causes a rather sharp peak while a bigger ring causes a wider bubble (cf. Fig 10). Notice that the mesh vertices in the interior of the handle polygon move according to the energy minimization.

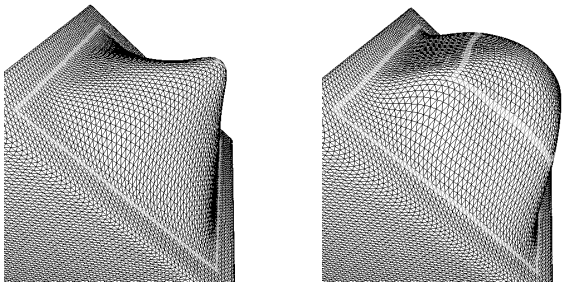


Figure 10: Controlling the characteristics of the modification by the size of a closed handle polygon.

Since we are working on triangle meshes, the energy minimization on $\mathcal{M}_*$ is done by discrete fairing techniques (cf. Section 3). The boundary triangles $\mathcal{S}$ provide the correct C^1 boundary conditions for minimizing the thin plate energy functional (3).

The handle imposes additional interpolatory constraints on the location only — derivatives should not be affect by the handle. Hence, we cannot have triangles being part of the handle geometry. We implemented the handle constraint in the following way: like the boundary polygon, the handle polygon defines a strip of triangles being intersected by it. Whether the handle polygon is open or closed, we find two polygons of mesh edges on either side of that strip. We take any one of the two polygons and collect every other mesh vertex in a set of *handle vertices*. Keeping these handle vertices fixed during the mesh optimization is the additional interpolatory constraint.

The reason for freezing only every other handle vertex is that three fixed vertices directly connected by two edges build a rigid constellation leaving no freedom to adjust the *angle* between them. During discrete optimization this would be the source of undesired artifacts in the smooth mesh.

With the boundary conditions properly set we perform the thin plate energy minimization by using the umbrella algorithm described in Section 3.1. To obtain interactive response times, we exploit the multi-level technique: a mesh decimation algorithm is applied to the mesh $\mathcal{M}_* \cup \mathcal{S}$ to build up a hierarchy. Then starting from the coarsest level, we apply the $\mathcal{U}^2$ smoothing filter alternating with mesh refinement. This process is fast enough to obtain several frames per second when modeling with meshes of $\#\mathcal{M}_* \approx 5K$ triangles (SGI R10000/195MHz). Typically, we set the ratio of the complexities between successive meshes in the hierarchy to $1:2$ or $1:4$ and iterate the smoothing filter 3 to 5 times on each level.

During the interactive mesh smoothing we do not compute the full V-cycle algorithm of Sect. 3.3. In fact, we omit the pre-smoothing and always start from the coarsest level. When a vertex is inserted during a mesh refinement step we place it initially at its neighbor's center of gravity unless the vertex is a handle vertex. Handle vertices are placed at the location prescribed by the designer's interaction (*handle interpolation constraint*). Hence the mesh is computed from scratch in every iteration instead of just updating the last position. This is very important for the modeling dialog since only the current position, orientation and scale of the handle determines the smoothed mesh and not the whole history of movements.

For the fast convergence of the optimization procedure it turns out to be important that the interpolation constraints imposed by the handle vertices show up already on rather coarse levels in the mesh hierarchy. Otherwise their impact cannot propagate far enough through the mesh such that cusps remain in the smoothed mesh which can only be removed by an excessive number of smoothing iterations. This additional requirement can easily be included into the mesh reduction scheme by lowering the priority ranking of collapses involving handle vertices.

4.2 Detail preservation

If the modified mesh $\mathcal{M}_*'$ is merely defined by constrained energy minimization, we obviously loose all the detail of the originally selected submesh $\mathcal{M}_*$. Since only the boundary and the handle vertices put constraints on the mesh, all other geometric features are smoothed out.

To preserve the detail, we use the multi-resolution representation explained in Section 2.2.2. After the boundary $\mathcal{S}$ and the handle polygon are defined but before the handle is moved by the designer, we apply the multi-level smoothing scheme once. Although the original mesh $\mathcal{M}_*$ and the smoothed mesh $\widetilde{\mathcal{M}_*}$ are topologically equivalent, they do have different levels of (geometric) resolution and hence constitute a two-scale decomposition based on varying levels of smoothness. We encode the difference $\mathcal{D}_*$ between the two meshes, i.e., the detail coefficients for the vertices $\mathbf{p}_i \in \mathcal{M}_*$ by storing the displacement vectors with respect to the local frame associated with the nearest triangle in $\widetilde{\mathcal{M}_*}$.

When the designer moves the handle, the bottom-up mesh smoothing is performed to re-adjust the mesh to the new interpolation conditions. On the resulting smooth mesh $\widetilde{\mathcal{M}_*'}$, the detail $\mathcal{D}_*$ is added and the final mesh $\mathcal{M}_*'$ is rendered on the screen. Due to the geometric coding of the detail information, this leads to intuitive changes in the surface shape (cf. Figs. 11, 12). The "frequency" of the modification is determined by the size of the area, i.e., by the boundary conditions and the fact that the *supporting mesh* is optimal with respect to the thin-plate functional.

5 Conclusions and future work

We presented a new approach to multi-resolution mesh representation and modeling which does not require the underlying triangle mesh to have subdivision connectivity. By adapting multi-level techniques known from numerical analysis to the non-regular setting of arbitrary mesh hierarchies, we are able to approximately solve constrained mesh optimization in realtime. Combining the two results allows us to present a flexible metaphor for interactive mesh modeling where the shape of the modification is controlled by energy minimization while the geometric detail is preserved and updated according to the change of the global shape.

Our current implementation of an experimental mesh modeling tool already provides sufficient functionality to apply sophisticated realtime modifications to arbitrary input meshes with up to 100K triangles. However, all changes do affect the *geometry* of the meshes only. So far we did not consider *topological* modifications of triangle meshes. In the future, when modifying a given mesh, we would like new vertices to be inserted where the mesh is locally stretched too much and, on the other hand, we would like vertices to be removed when strong global modification causes local self-intersection of the reconstructed detail.

References

[1] BONNEAU, G., HAHMANN, S., AND NIELSON, G. Blac-wavelets: a multiresolution analysis with non-nested spaces. In *Visualization Proceedings* (1996), pp. 43–48.

[2] CATMULL, E., AND CLARK, J. Rcursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design 10*, 6 (Nov. 1978), 239–248.

[3] CELNIKER, G., AND GOSSARD, D. Deformable curve and surface finite elements for free-form shape design. In *Computer Graphics (SIGGRAPH 91 Proceedings)* (July 1991), pp. 257–265.

[4] DAUBECHIES, I. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conf. Series in Appl. Math., Vol. 61. SIAM, Philadelphia, PA, 1992.

[5] DOO, D., AND SABIN, M. Behaviour of recursive division surfaces near extraordinary points. *CAD* (1978).

[6] DYN, N., LEVIN, D., AND GREGORY, J. A. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics 9*, 2 (April 1990), 160–169.

[7] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. In *Computer Graphics (SIGGRAPH 95 Proceedings)* (1995), pp. 173–182.

[8] FARIN, G. *Curves and Surfaces for CAGD*, 3rd ed. Academic Press, 1993.

[9] FINKELSTEIN, A., AND SALESIN, D. H. Multiresolution Curves. In *Computer Graphics (SIGGRAPH 94 Proceedings)* (July 1994), pp. 261–268.

[10] FORSEY, D. R., AND BARTELS, R. H. Hierarchical B-spline refinement. In *Computer Graphics (SIGGRAPH 88 Proceedings)* (1988), pp. 205–212.

[11] GARLAND, M., AND HECKBERT, P. S. Surface Simplification Using Quadric Error Metrics. In *Computer Graphics (SIGGRAPH 97 Proceedings)* (1997), pp. 209–218.

[12] GORTLER, S. J., AND COHEN, M. F. Hierarchical and Variational Geometric Modeling with Wavelets. In *Proceedings Symposium on Interactive 3D Graphics* (May 1995).

[13] GREINER, G. Variational design and fairing of spline surfaces. *Computer Graphics Forum 13* (1994), 143–154.

[14] HACKBUSCH, W. *Multi-Grid Methods and Applications*. Springer Verlag, Berlin, 1985.

[15] HOPPE, H. Progressive Meshes. In *Computer Graphics (SIGGRAPH 96 Proceedings)* (1996), pp. 99–108.

[16] HOSCHEK, J., AND LASSER, D. *Fundamentals of Computer Aided Geometric Design*. AK Peters, 1993.

[17] KOBBELT, L. *Iterative Erzeugung glatter Interpolanten*. Shaker Verlag, ISBN 3-8265-0540-9, 1995.

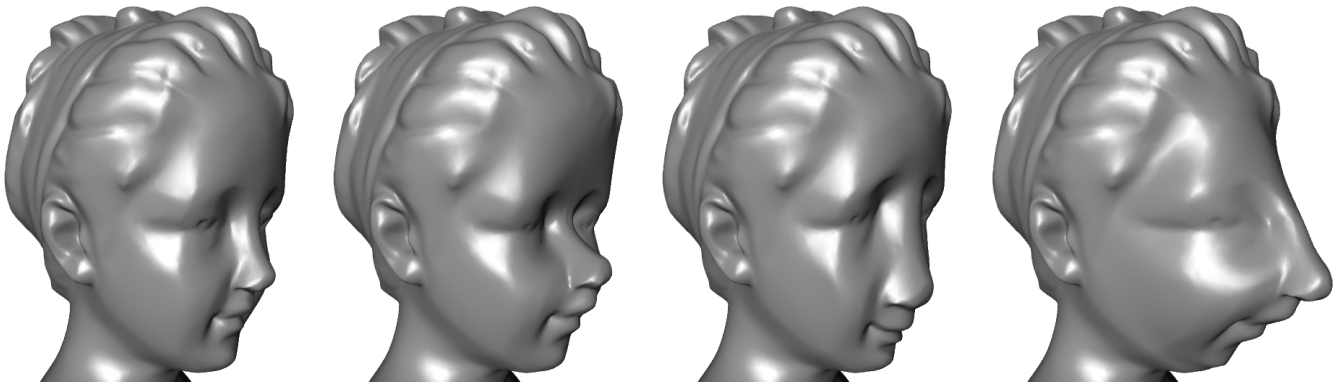


Figure 11: The mesh model of a bust (62K triangles, left, courtesy Stefan Karbacher) is modified by multi-resolution edits. The modified area $\mathcal{M}_*$ is the bust's face while the handle polygon lies around the nose. From left to right, we apply rotation, scaling and translation.

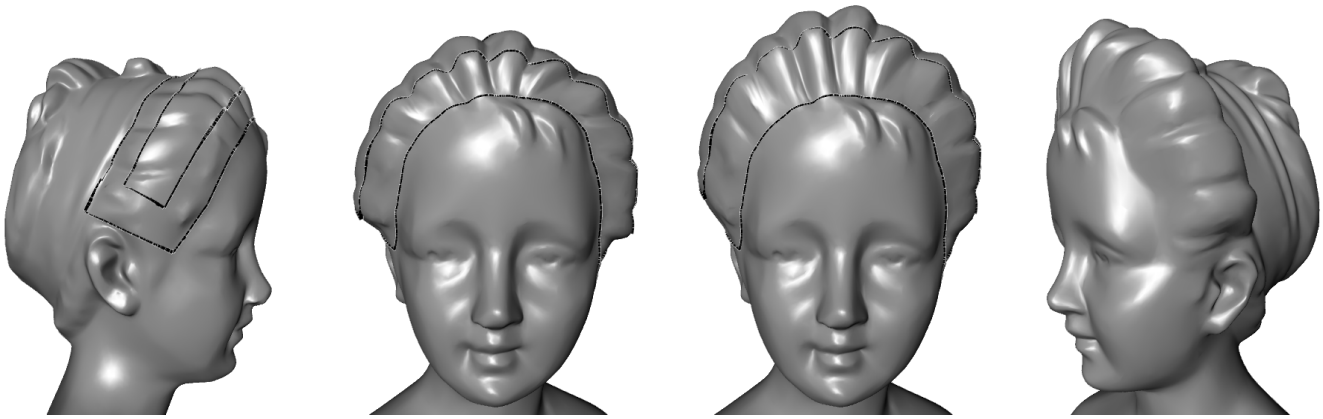


Figure 12: Some more modifications on the bust model. The support of the modification and the handle geometry adapt to the intended design operation. The detail is preserved while the global modification is controlled by discrete fairing.

[18] KOBBELT, L. Interpolatory Subdivision on Open Quadrilateral Nets with Arbitrary Topology. In *Computer Graphics Forum, Proceedings of Eurographics '96* (1996), pp. C407–C420.

[19] KOBBELT, L. Discrete fairing. In *Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces* (1997), pp. 101–131.

[20] KOBBELT, L., CAMPAGNA, S., AND SEIDEL, H.-P. A general framework for mesh decimation. In *Proceedings of the Graphics Interface conference '98* (1998).

[21] KRISHNAMURTHY, V., AND LEVOY, M. Fitting smooth surfaces to dense polygon meshes. In *Computer Graphics (SIGGRAPH 96 Proceedings)* (1996), pp. 313–324.

[22] LOOP, C. Smooth spline surfaces over irregular meshes. In *Computer Graphics Proceedings* (1994), Annual Conference Series, ACM Siggraph, pp. 303–310.

[23] LOUNSBERY, M., DEROSE, T., AND WARREN, J. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM Transactions on Graphics 16*, 1 (January 1997), 34–73.

[24] LUEBKE, D., AND ERIKSON, C. View-Dependent Simplification of Arbitrary Polygonal Environments. In *Computer Graphics (SIGGRAPH 97 Proceedings)* (1997), pp. 199–208.

[25] MORETON, H., AND SÉQUIN, C. Functional optimization for fair surface design. In *Computer Graphics (SIGGRAPH 92 Proceedings)* (1992), pp. 167–176.

[26] OPPENHEIM, A., AND SCHAFER, R. *Discrete-Time Signal Processing*. Prentice Hall, 1989.

[27] ROSSIGNAC, J. Simplification and Compression of 3D Scenes, 1997. Tutorial Eurographics '97.

[28] SAPIDIS, N. E. *Designing Fair Curves and Surfaces*. SIAM, 1994.

[29] SCHABAK, R., AND WERNER, H. *Numerische Mathematik*. Springer Verlag, 1993.

[30] SCHRÖDER, P., AND SWELDENS, W. Spherical wavelets: Efficiently representing functions on the sphere. In *Computer Graphics (SIGGRAPH 95 Proceedings)* (1995), pp. 161–172.

[31] SCHROEDER, W. J., ZARGE, J. A., AND LORENSEN, W. E. Decimation of Triangle Meshes. In *Computer Graphics (SIGGRAPH 92 Proceedings)* (1992), pp. 65–70.

[32] STOER, J. *Einführung in die Numerische Mathematik I.* Springer Verlag, 1983.

[33] STOLLNITZ, E., DEROSE, T., AND SALESIN, D. *Wavelets for Computer Graphics*. Morgan Kaufmann Publishers, 1996.

[34] TAUBIN, G. A signal processing approach to fair surface design. In *Computer Graphics (SIGGRAPH 95 Proceedings)* (1995), pp. 351–358.

[35] TURK, G. Re-Tiling Polygonal Surfaces. In *Computer Graphics (SIGGRAPH 92 Proceedings)* (1992), pp. 55–64.

[36] WARREN, J. Subdivision schemes for variational splines, 1997. Preprint.

[37] WELCH, W., AND WITKIN, A. Variational surface modeling. In *Computer Graphics (SIGGRAPH 92 Proceedings)* (1992), pp. 157–166.

[38] WELCH, W., AND WITKIN, A. Free–Form shape design using triangulated surfaces. In *Computer Graphics (SIGGRAPH 94 Proceedings)* (1994), A. Glassner, Ed., pp. 247–256.

[39] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interpolating subdivision for meshes with arbitrary topology. In *Computer Graphics (SIGGRAPH 96 Proceedings)* (1996), pp. 189–192.

[40] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interactive multiresolution mesh editing. In *Computer Graphics (SIGGRAPH 97 Proceedings)* (1997), pp. 259–268.

Multiresolution Hierarchies on Unstructured Triangle Meshes

by L. Kobbelt, J. Vorsatz, and H.-P. Seidel, Computational Geometry Theory and Applications, special issue on multi-resolution modeling and 3D geometry compression.

Multiresolution Hierarchies on Unstructured Triangle Meshes

Leif Kobbelt* Jens Vorsatz Hans-Peter Seidel

Max-Planck-Institut für Informatik

Abstract

The use of polygonal meshes for the representation of highly complex geometric objects has become the de facto standard in most computer graphics applications. Especially triangle meshes are preferred due to their algorithmic simplicity, numerical robustness, and efficient display. The possibility to decompose a given triangle mesh into a hierarchy of differently detailed approximations enables sophisticated modeling operations like the modification of the global shape under preservation of the detail features.

So far, multiresolution hierarchies have been proposed mainly for meshes with subdivision connectivity. This type of connectivity results from iteratively applying a uniform split operator to an initially given coarse base mesh. In this paper we demonstrate how a similar hierarchical structure can be derived for arbitrary meshes with no restrictions on the connectivity. Since smooth (subdivision) basis functions are no longer available in this generalized context, we use constrained energy minimization to associate *smooth* geometry with *coarse* levels of detail. As the energy minimization requires one to solve a global sparse system, we investigate the effect of various parameters and boundary conditions in order to optimize the performance of iterative solving algorithms.

Another crucial ingredient for an effective multiresolution decomposition of unstructured meshes is the flexible representation of detail information. We discuss several approaches.

1 Introduction

Subdivision techniques provide very efficient and flexible algorithms for the generation of free form surface geometry [2, 5, 6, 18, 25, 39]. Starting with an arbitrary control mesh $\mathcal{M}_0$ we can apply the subdivision rules to compute finer and finer meshes $\mathcal{M}_m$ with control vertices $\mathbf{p}_i^m$ becoming more and more dense until the desired approximation tolerance required for a given application is reached. The result is a smooth surface having the same topology as the initial control mesh.

The distinct subdivision levels $\mathcal{M}_m$ give rise to powerful multiresolution semantics since we can consider a subdivision scheme as the low pass reconstruction operator in the filter bank algorithm for a wavelet-type decomposition of the geometric shape. The subdivision basis functions which are associated with the control vertices generalize the concept of dyadic scaling functions to polyhedral parameter domains [26, 31, 40].

*Computer Graphics Group, Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany, kobbelt@mpi-sb.mpg.de

However, subdivision techniques are genuinely based on the *coarse-to-fine* generation of hierarchical geometry representations: a coarse base mesh with only few faces is iteratively refined by introducing an exponentially increasing number of degrees of freedom for capturing finer and finer detail information. As a consequence, the control meshes must have so-called *subdivision connectivity* which means that sub-regions of the refined mesh $\mathcal{M}_m$ which correspond to one single face of the original base mesh $\mathcal{M}_0$, have the connectivity of regular grids (cf. Fig. 1).

It turns out that this restriction is not suitable for several standard application scenarios. In practice one is often given some *existing* geometric model which is to be modified by making local or global adjustments. Since such triangular meshes usually do not come with the rather special subdivision connectivity, we cannot apply subdivision techniques without preprocessing.

This preprocessing has to perform a global remeshing of the data. Although several flexible and robust algorithms have been proposed for this problem [7, 24, 22] there are still difficulties with automatically finding a suitable layout for the base mesh. Semi-automatic approaches like [23, 24] with constraints set by the user only partially solve this problem. Moreover, the remeshing is *always* a resampling process and hence even an optimal remeshing algorithm cannot recover the original shape exactly. High frequency artifacts due to alias errors are rather likely to appear.

The rigidity of subdivision connectivity meshes emerges from the fact that the classification of the detail coefficients into predefined refinement levels is done topologically. The actual *size* or *geometric frequency* associated with a detail coefficient hence strongly depends on the size of the corresponding base triangle in the unrefined control mesh. As it is usually not possible to have all triangles in the base mesh of unit size, detail features on the same refinement level and their corresponding support can vary by one or more orders of magnitude. Avoiding this problem by using adaptive refinement strategies is not appropriate in some applications.

Another problem which is inherent to the multiresolution representation of free form geometry based on subdivision surfaces is the *fixed support* of the modifications. If control vertices are used as handles to modify the surface geometry on a certain level of detail then the region of the mesh which actually changes, is determined by the support of the associated basis function. We could simulate more flexibility in the definition of the support by moving several control vertices from some finer level simultaneously but this would diminish the advantages of a multiresolution representation.

Moreover, the coarse scale control vertices in a subdivision representation are *aligned* to the coarse scale grid. This means that we lose spatial resolution if we modify a surface on a low frequency band. Consequently, we can apply modifications of the global shape only at a very limited number of locations. In fact, as every control vertex $\mathbf{c}$ in a subdivision connectivity mesh is introduced on a certain refinement level $l(\mathbf{c})$ the support of the modification when moving $\mathbf{c}$ is bounded by the size of the basis functions on that level.

For example, if we move a control vertex $\mathbf{c}_0$ which topologically corresponds to a vertex in the base mesh then we can choose the basis function controlling the edit from any refinement level. However, moving a directly adjacent vertex $\mathbf{c}_m$ on refinement level m can only affect the finest scale since $\mathbf{c}_m$ does not have a representation on any coarser level. Hence, a coarse scale modification can be centered at $\mathbf{c}_0$ but not at $\mathbf{c}_m$ which lies only ε away. This is not intuitive for the designer to whom the actual surface representation

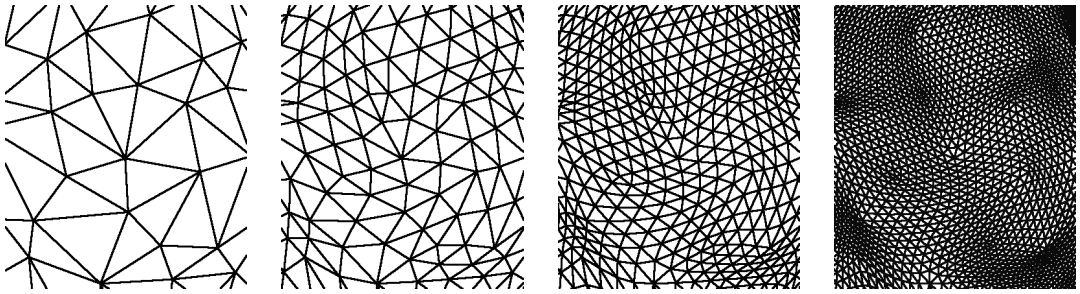


Figure 1: Subdivision connectivity meshes result from iteratively applying a uniform split operation to the faces of an initial control mesh. Only a fixed number of isolated extraordinary vertices with valence $\neq 6$ remain in the mesh.

Figure 2: In a multiresolution modeling environment, the support of the modification and its characteristic shape should adapt to the given geometry (here: the bust's hair). The low-frequency modification affects exactly the region defined by the designer. The high frequency detail is preserved in a natural way.

should be opaque.

With all these difficulties enumerated, we understand that coarse-to-fine hierarchies emerging from subdivision techniques might certainly be the best way to effectively represent smooth free form geometry in applications like surface reconstruction, scattered data interpolation, or *ab initio* design where the face layout for the base mesh is defined by the designer. However, it does not appear to be the optimal solution for flexibly modifying *existing* models like the ones obtained from capturing real object geometry by laser scanning devices.

Our goal is to enable true free form multiresolution edits where the support and the characteristics of a modification can adapt to the surface geometry (cf. Fig. 2). In [21] we generalized the concept of multiresolution decomposition and modeling to meshes with arbitrary topology and connectivity. The key observation is that we can no longer stick to the notion of surface geometry being represented by the superposition of smooth scalar valued basis functions over a nested sequence of grids. The reason for this is that we cannot make any assumptions on the actual distribution of the mesh vertices a priori. Hence, imposing any kind of vector space structure would require us to construct explicitly a custom tailored basis function for each vertex.

Leaving the classical set-up, it turns out that for mere polygonal meshes (not control meshes), *coarseness* and *smoothness* are no longer synonyms. While in the subdivision framework the basis functions on the coarse scales are also smoother in the sense that they have less curvature, we find that for plain polygonal meshes the effect of shifting a control vertex on a coarse scale still causes a sharp feature. To speak about *smooth* polygonal meshes we need more degrees of freedom since smooth meshes are typically rather fine tesselations.

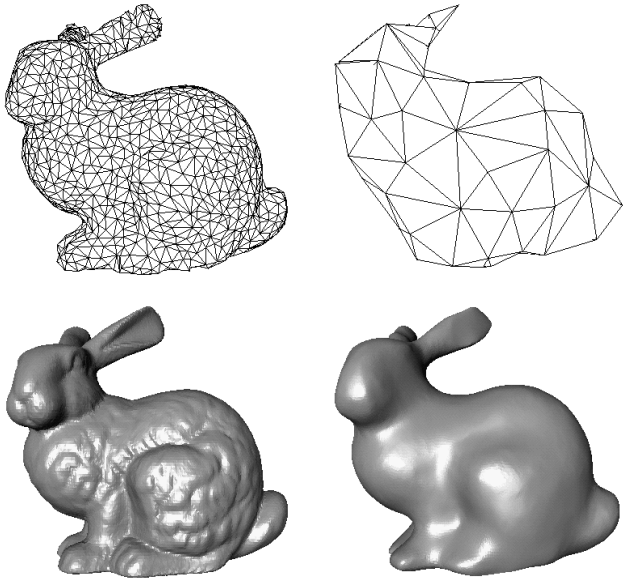


Figure 3: For plain triangle meshes we have to distinguish coarse and smooth approximations (upper and lower row). If meshes are considered as control meshes with respect to scalar valued basis functions then the connection between the upper an the lower row is provided by evaluating the weighted superposition of the control vertices' influence.

We have to solve two central problems in order to develop effective multiresolution algorithms for arbitrary meshes. First we have to construct a *topological hierarchy* of different resolutions with the finest resolution being the original mesh. This hierarchy must not rely on any assumptions about the connectivity of the given mesh.

Besides the topological levels of detail we need a *geometric hierarchy*, i.e., we need a proper characterization of *smooth* coarse-scale geometry. In the subdivision based multiresolution setting,

we have the associated scaling functions which fill in the smooth geometry between the coarse scale control vertices. In the generalized setting we have to find an alternative definition since a priori defined scaling functions are no longer available. A possible solution to this problem is to use discrete energy minimization techniques to obtain smooth low-detail approximations to the original model.

While the basic principles of this approach have been presented in [21], we discuss more technical details in this paper. After explaining the generation of coarse-to-fine hierarchies and fine-to-coarse hierarchies, we compare different ways to represent the detail information between the resolution levels. The crucial issues are here how to define the local frames with respect to which the detail is encoded and how to chose the number of hierarchy levels. In the context of discrete energy minimization we investigate the effect of various parameters in the multi-level solving algorithm, namely the number of hierarchy levels and the number of Gauß-Seidel iterations on each level. We demonstrate that imposing interpolation constraints at the centers of the triangular faces accelerates the global convergence of the iterative solver compared to imposing the constraints at the vertices.

2 Multiresolution representations

Most schemes for the multiresolution representation and modification of triangle meshes emerge from generalizing harmonic analysis techniques like the wavelet transform [1, 26, 31, 34]. Since the fundamentals are derived in the scalar-valued functional setting $\mathbb{R}^d \to \mathbb{R}$, difficulties emerge from the fact that manifolds in space are in general not topologically equivalent to simply connected regions in $\mathbb{R}^d$.

The philosophy behind multiresolution modeling on surfaces is hence to mimic the algorithmic structure of the related functional transforms and preserve some of the important properties like locality, smoothness, stability or polynomial precision which have related meaning in both settings [8, 13, 40]. Accordingly, the nested sequence of spaces underlying the decomposition into disjoint frequency bands is thought of being generated bottom-up from a coarse base mesh up to finer and finer resolutions. This implies that subdivision connectivity is mandatory on higher levels of detail, i.e., the mesh has to consist of large regular regions with isolated extra-ordinary vertices. Additionally, we have to make sure that the topological distance between the singularities is the same for every pair of neighboring singularities and this topological distance has to be a power of 2. Obviously, sophisticated modeling operations like *boolean operations* necessarily require a complete restructuring of the resulting mesh to re-establish subdivision connectivity.

These special topological requirements prevent such techniques from being applicable to arbitrary input meshes. To obtain a proper hierarchy, global remeshing and resampling is necessary which gives rise to alias-errors and requires involved computations [7, 24].

Luckily, the restricted connectivity is not necessary to define different levels of resolution or approximation for a triangle mesh. In the literature on mesh decimation we find many examples for hierarchies built on arbitrary meshes [12, 17, 20, 27, 29, 32, 36]. The key is always to build the hierarchy top-down by eliminating vertices from the current mesh (*incremental reduction*, cf. Fig. 4). Running a mesh decimation algorithm, we can stop, e.g., every time a certain percentage of the vertices is removed. The intermediate meshes can be used as a level-of-detail representation [17, 26].

In both cases, i.e., the coarse-to-fine or the fine-to-coarse generation of nested (vertex-) grids, the multiresolution concept is rigidly attached to topological entities. This makes sense if hierarchies are merely used to adjust the complexity of the representation. We will exploit the sequence of nested grids emerging from this topological hierarchy to generalize the concept of multi-grid solvers for large sparse systems.

In the context of multiresolution *modeling*, however, we want the hierarchy not necessarily to rate meshes according to their *coarseness* but rather according to their *smoothness*. For this we need a geometric hierarchy accompanying the topological one. To complete our basic equipment for the multiresolution set-up on unstructured meshes we hence need (besides the static levels of detail) to define the *decomposition* and *reconstruction* operations which separate the high-frequency detail from the low-frequency shape and eventually recombine the two to recover the original mesh. Here, the reconstruction operator has to generate the smooth low-frequency shape if the detail information is suppressed during reconstruction. This is where discrete fairing techniques come in. Further, we have to encode the detail information relative to the low-frequency shape in order to guarantee intuitive detail preservation after a global modification (*local frames*).

2.1 Coarse-to-fine Hierarchies

For subdivision based multiresolution representation the reconstruction operator is given by the underlying subdivision scheme. We transform a given mesh $\mathcal{M}_m$ to the next refinement level $\mathcal{M}'_{m+1} = \mathbf{S}\,\mathcal{M}_m$ by applying the stationary subdivision operator $\mathbf{S}$ and move the obtained control vertices by adding the associated detail vectors: $\mathcal{M}_{m+1} = \mathcal{M}'_{m+1} + \mathcal{D}_m$. The support of the subdivision mask implies that each control vertex $\mathbf{p}_i^m$ in $\mathcal{M}_m$ has influence on several control vertices in $\mathcal{M}'_{m+1}$. Consequently, the modification of $\mathbf{p}_i^m$'s position eventually causes a smooth bump on the resulting surface. The actual shape of this bump can be computed by applying the subdivision operator $\mathbf{S}$ *without* detail reconstruction, i.e. $\mathcal{D}_m := 0$. Obviously, the support of the bump depends on the refinement level m on which the modification is applied.

The decomposition operator has to be an inverse of the subdivision operator, i.e., given a fine mesh $\mathcal{M}_{m+1}$ we have to find a mesh $\mathcal{M}_m$ such that $\mathcal{M}_{m+1} \approx \mathbf{S}\,\mathcal{M}_m$. In this case the detail vectors $\mathcal{D}_m := \mathcal{M}_{m+1} - \mathbf{S}\,\mathcal{M}_m$ become as small as possible [40]. Due to the uniform split which is part of the subdivision operator $\mathbf{S}$, it is obvious that this technique applies only if $\mathcal{M}_{m+1}$ has subdivision connectivity.

2.2 Fine-to-coarse Hierarchies

If we build the hierarchy by using an incremental mesh decimation scheme, the decomposition operator $\mathbf{D}$ applies to arbitrary meshes. Given a fine mesh $\mathcal{M}_{m+1}$ we find $\mathcal{M}_m = \mathbf{D}\,\mathcal{M}_{m+1}$, e.g., by applying a number of edge collapse operations. However, it is not clear how to define the detail coefficients since inverse mesh decimation (*progressive meshes*) always reconstructs the original mesh and there is no canonical way to generate smooth low-frequency geometry by suppressing the detail information during reconstruction.

To solve this problem we split each step of the progressive mesh refinement into a topological operation (vertex insertion) and a geometric operation which places the re-inserted vertices at their original position. In analogy to the plain subdivision without detail reconstruction, we have to figure out a heuristic which places the new vertices such that they lie on a smooth surface (instead of their original position). The difference vector between this predicted position and the original location of the vertex can then be used as the associated detail vector.

Since we operate on unstructured meshes, we cannot use fixed (stationary) rules for the placement of the re-inserted vertices. Instead we use discrete energy minimization which means that the re-inserted vertices are placed such that some global bending energy is minimized. In Section 3 we review a simple technique for the effective generation of meshes with minimum bending energy without specific requirements on the connectivity.

2.3 Detail encoding

In order to guarantee intuitive detail preservation under modification of the global shape, we cannot simply store the detail vectors

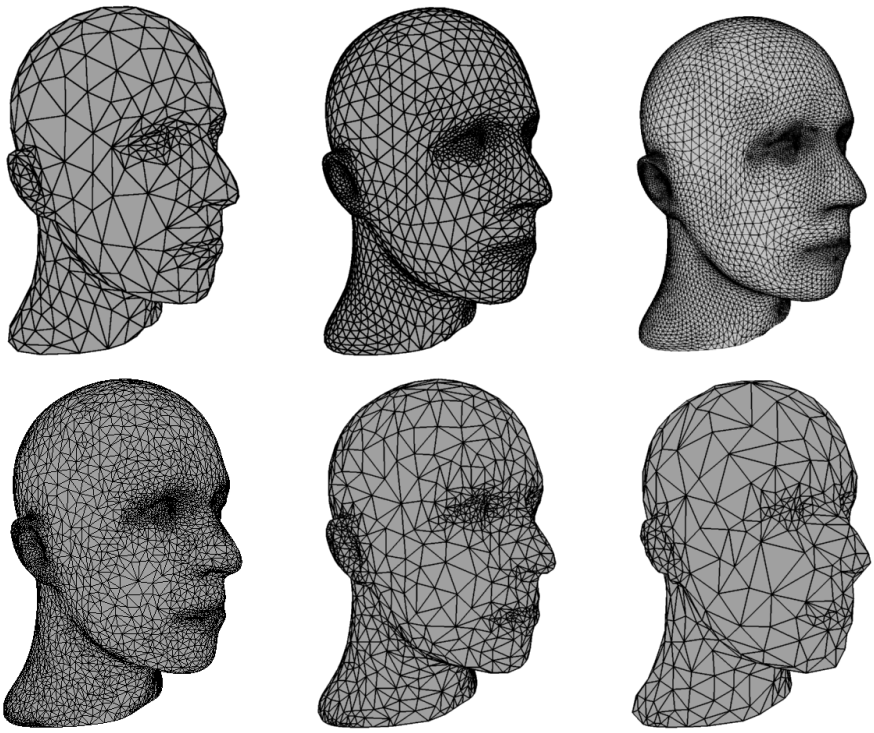


Figure 4: For multiresolution representations based on subdivision techniques, the hierarchies are built from coarse to fine by applying a uniform subdivision operator (top row, left to right) while incremental mesh decimation generates hierarchies from fine to coarse by iteratively removing vertices (bottom row, left to right).

with respect to a global coordinate system but have to define them with respect to local frames which are aligned to the low-frequency geometry [10, 11]. Usually, the associated local frame for each vertex has its origin at the location predicted by the reconstruction operator with suppressed detail. This is in analogy to decompositions based on a global parameterization of the surfaces.

However, in many cases this can lead to rather long detail vectors with a significant component within the local tangent plane (cf. Fig. 5). Since we prefer short detail vectors for stability reasons, it makes sense to use a different origin for the local frame. In fact, the optimal choice is to find that point on the low-frequency surface whose normal vector points directly to the original vertex. In this case, the detail is not given by a three dimensional vector $(\triangle x, \triangle y, \triangle z)^T$ but rather by a base point $\mathbf{p} = \mathbf{p}(u,v)$ on the low-frequency geometry plus a scalar value h for the displacement in normal direction. If a local parameterization of the surface is available then the base point $\mathbf{p}$ can be specified by a two-dimensional parameter value (u,v).

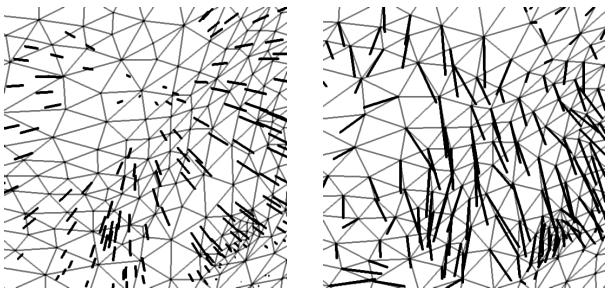


Figure 5: The shortest detail vectors are obtained by representing the detail coefficients with respect to the nearest local frame (left) instead of attaching the detail vectors to the topologically corresponding original vertices.

The general setting for detail computation is that we have given two meshes $\mathcal{M}_{m+1}$ and $\mathcal{M}'_{m+1}$ where $\mathcal{M}_{m+1}$ is the original data while $\mathcal{M}'_{m+1}$ is reconstructed from the low-frequency approximation $\mathcal{M}_m$ with suppressed detail, i.e. for coarse-to-fine hierachies, the mesh $\mathcal{M}'_{m+1}$ is generated by applying a stationary subdivision scheme and for fine-to-coarse hierarchies $\mathcal{M}'_{m+1}$ is optimal with respect to some global bending energy functional. Encoding the geometric difference between both meshes requires us to associate each vertex $\mathbf{p}$ of $\mathcal{M}_{m+1}$ with a corresponding base point $\mathbf{q}$ on the continuous (piecewise linear) surface $\mathcal{M}'_{m+1}$ such that the difference vector between the original point and the base point is parallel to the normal vector at the base point. Any point $\mathbf{q}$ on $\mathcal{M}'_{m+1}$ can be specified by a triangle index i and barycentric coordinates within the referred triangle.

To actually compute the detail coefficients, we have to define a normal field on the mesh $\mathcal{M}'_{m+1}$. The most simple way to do this is to use the normal vectors of the triangular faces for the definition of a piecewise constant normal field. However, since the orthogonal prisms spanned by a triangle mesh do not completely cover the vicinity of the mesh, we have to accept negative barycentric coordinates for the base points if an original vertex lies close to an edge of $\mathcal{M}'_{m+1}$ or if $\mathcal{M}'_{m+1}$ is not smooth enough (cf. Fig 6). This leads to non-intuitive detail reconstruction if the low-frequency geometry is modified (cf. Fig. 7).

A technique used in [21] is based on the construction of a local quadratic interpolant to the low-frequency geometry. The base point is found by Newton-iteration. Although this technique reduces the number of pathological configurations with negative barycentric coordinates for the base point, we still observe artifacts in the reconstructed high-frequency surface which are caused by the fact that the resulting global normal field of the combined local patches is not continuous.

We therefore propose a different approach which adapts the basic

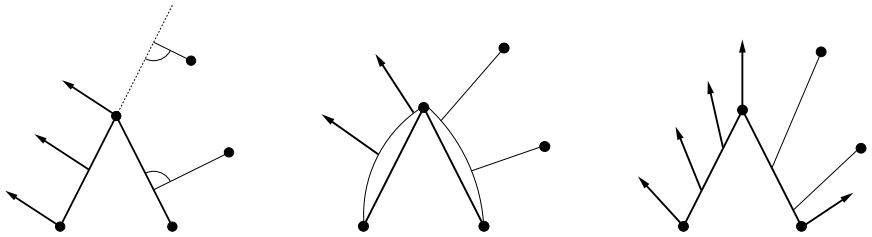


Figure 6: The position of a vertex in the original mesh (high-frequency geometry) is given by a base point on the low-frequency geometry plus a displacement in normal direction. There are many ways to define a normal field on a triangle mesh. With piecewise constant normals (left) we do not cover the whole space and hence we sometimes have to use virtual base points with negative barycentric coordinates. The use of local quadratic patches and their normal fields (center) somewhat improves the situation but problems still occur since the overall normal field is not globally continuous. Such difficulties are completely avoided if we generate a Phong-type normal field by blending estimated vertex normals (right).

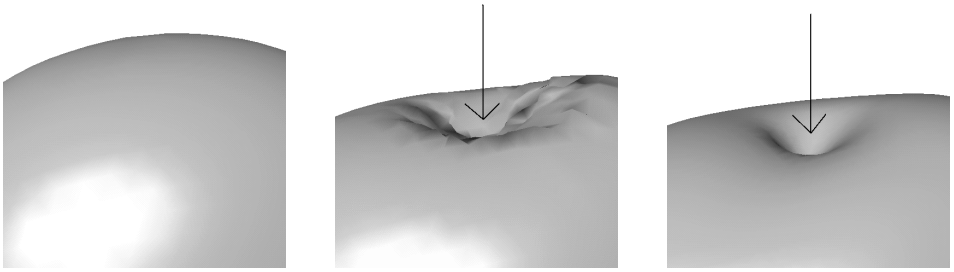


Figure 7: We modified the original surface (left) by using a two-band multiresolution decomposition. Since in this particular experiment the low-frequency geometry was chosen not sufficiently smooth, many detail vectors have base points with negative barycentric coordinates when we use a piecewise constant normal field. Consequently, no proper detail reconstruction is possible after the modification (center). Representing the detail vectors with respect to the Phong normal field on the low-frequency mesh leads to the expected result (right).

idea of Phong-shading [9] where normal vectors are estimated at the vertices of a triangle mesh and a continuous normal field for the interior of the triangular faces is computed by linearly blending the normal vectors at the corners.

Suppose we are given a triangle $\triangle(\mathbf{a},\mathbf{b},\mathbf{c})$ with the associated normal vectors $N_\mathbf{a}$, $N_\mathbf{b}$, and $N_\mathbf{c}$. For each interior point

$$\mathbf{q} \;=\; \alpha\mathbf{a}+\beta\mathbf{b}+\gamma\mathbf{c}$$

with $\alpha+\beta+\gamma=1$ we find the associated normal vector $N_\mathbf{q}$ by

$$N_\mathbf{q} \;=\; \alpha N_\mathbf{a}+\beta N_\mathbf{b}+\gamma N_\mathbf{c}.$$

When computing the detail coefficients for a given point $\mathbf{p}$ we have to find the base point $\mathbf{q}$ such that

$$\Big(\mathbf{p}-\mathbf{q}\Big)\times N_\mathbf{q}$$

has all three coordinates vanishing. By plugging in the definition of $\mathbf{q}$ and $N_\mathbf{q}$ and eliminating $\gamma=1-\alpha-\beta$ we obtain a bivariate quadratic function

$$F \;:\; (u,v) \;\to\; \mathbb{R}^3$$

and we have to find the parameter value (α,β) such that $F(\alpha,\beta)=(0,0,0)^T$. This can be accomplished by performing several steps of Newton-iteration. Notice that F can be interpreted as a quadratic surface patch in $\mathbb{R}^3$ which passes through the origin. The Taylor-coefficients of F can explicitly be given by

$$\begin{array}{lcl} F & = & W+WW \\ F_u & = & U+UW-W-2WW \\ F_v & = & V+VW-W-2WW \\ F_{uu} & = & UU-UW+WW \\ F_{uv} & = & UV-UW-VW+2WW \\ F_{vv} & = & VV-VW+WW \end{array}$$

where

$$\begin{array}{lcl} U & = & \mathbf{p}\times N_\mathbf{a} \\ V & = & \mathbf{p}\times N_\mathbf{b} \\ W & = & \mathbf{p}\times N_\mathbf{c} \\ UU & = & N_\mathbf{a}\times\mathbf{a} \\ VV & = & N_\mathbf{b}\times\mathbf{b} \\ WW & = & N_\mathbf{c}\times\mathbf{c} \\ UV & = & (N_\mathbf{b}\times\mathbf{a})+(N_\mathbf{a}\times\mathbf{b}) \\ UW & = & (N_\mathbf{c}\times\mathbf{a})+(N_\mathbf{a}\times\mathbf{c}) \\ VW & = & (N_\mathbf{c}\times\mathbf{b})+(N_\mathbf{b}\times\mathbf{c}) \end{array}$$

In case one of the barycentric coordinates of the resulting point $\mathbf{q}$ is negative, we continue the search for a base point in the corresponding neighboring triangle. Since the Phong normal field is globally continuous we always find a base point with positive barycentric coordinates. Fig. 6 depicts the situation schematically and Fig. 7 shows an example edit where the piecewise constant normal field causes mesh artifacts which do not occur if the Phong normal field is used.

2.4 Hierarchy levels

For coarse-to-fine hierarchies the levels of detail are determined by the uniform refinement operator. Starting with the base mesh $\mathcal{M}_0$, the mth refinement level is reached after applying the refinement operator m times. For fine-to-coarse hierarchies there is no such canonical choice for the levels of resolution. Hence we have to figure out some heuristics to define such levels.

In [21] a simple two-band decomposition has been proposed for the modeling, i.e. the high frequency geometry is given by the original mesh and the low-frequency geometry is the solution of some constrained optimization problem. This simple decomposition performs well if the original geometry can be projected onto the low-frequency geometry without self-intersections. Fig. 8 schematically

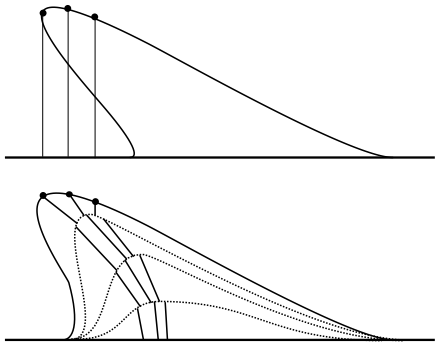


Figure 8: When the difference between two geometric levels of detail is too big, the high-frequency geometry cannot be projected directly onto the low-frequency geometry without self-intersections. In order to guarantee correct detail reconstruction, we have to generate intermediate levels such that the mapping between two successive levels is one-to-one.

shows a configuration where this requirement is not satisfied and consequently the detail feature does not deform intuitively with the change of the global shape.

This effect can be avoided by introducing several intermediate levels of detail, i.e., by using a true multi-band decomposition. The number of hierarchy levels has to be chosen such that the $(i+1)$st level can be projected onto level i without self-intersection. Detail information has to be computed for every intermediate level.

The intermediate levels can be generated by the following algorithm. We start with the original mesh and apply an incremental mesh decimation algorithm which performs a sequence of edge collapse operations. When a certain mesh complexity is reached, we perform the reverse sequence of vertex split operations which reconstructs the original mesh connectivity. The position of the reinserted vertices is found by solving a global bending energy minimization problem (discrete fairing). The mesh that results from this procedure is a smoothed version of the original mesh where the degree by which detail information has been removed depends on the target complexity of the decimation algorithm (cf. Fig 10)

Suppose the original mesh has n_m vertices, where m is the number of intermediate levels that we want to generate. We can compute the meshes $\mathcal{M}_m, \ldots, \mathcal{M}_0$ with fewer detail by applying the above procedure where the decimation algorithm stops at a target resolution of $n_m, \ldots, n_0$ remaining vertices respectively. The resulting meshes yield a multi-band decomposition of the orignal data. When a modeling operation changes the shape of $\mathcal{M}_0$ we first reconstruct the next level $\mathcal{M}_1'$ by adding the stored detail vectors and then proceed by successively reconstructing $\mathcal{M}_{i+1}'$ from $\mathcal{M}_i'$.

The remaining question is how to determine the numbers n_i. A simple way to do this is to build a geometric sequence with $n_{i+1}/n_i = \text{const}$. This mimics the exponential complexity growth of the coarse-to-fine hierarchies. Another approach is to stop the decimation every time a certain average edge length $\bar{l}_i$ in the remaining mesh is reached.

A more complicated heuristic tries to equalize the sizes of the differences between levels, i.e., the sizes of the detail vectors. We first compute a multi-band decomposition with, say, 100 levels of detail where we choose $\sqrt[i]{n_i} = \text{const}$. For every pair of successive levels we can compute the average length of the detail vectors (displacement values). From this information we can easily choose appropriate values $n_j = \bar{n}_{i_j}$ such that the geometric difference is distributed evenly among the detail levels.

In practice it turned out that about five intermediate levels is usually enough to guarantee correct detail reconstruction. Fig. 9 compares the results of a modeling operation based on a two-band and a multi-band decomposition.

3 Constrained discrete fairing

In the previous section we explained how to generate topological hierarchies for meshes with arbitrary connectivity by incremental mesh decimation. An associated geometric hierarchy can be obtained by re-inserting the removed vertices and moving them to a new position such that a global bending energy functional is minimized. The idea is to compute a mesh which is as smooth as possible while still containing a controllable amount of geometric detail. Fig. 10 shows an example.

From CAGD it is well-known that constrained energy minimization is a very powerful technique to generate high quality surfaces [3, 14, 28, 30, 37]. For efficiency, one usually defines a simple quadratic energy functional $\mathcal{E}(f)$ and searches among the set of functions satisfying prescribed interpolation constraints for that function f which minimizes $\mathcal{E}$.

Transferring the continuous concept of energy minimization to the discrete setting of triangle mesh optimization leads to the discrete fairing approach [19, 38]. Local polynomial interpolants are used to estimate derivative information at each vertex by divided difference operators. Hence, the differential equation characterizing the functions with minimum energy is discretized into a linear system for the vertex positions.

Since this system is global and sparse, we apply iterative solving algorithms like the Gauß-Seidel-scheme. For such algorithms one iteration step merely consists in the application of a simple local averaging operator. This makes discrete fairing an easy accessible technique for mesh optimization.

For the most popular fairing functional, the *thin-plate energy*, this approach leads to a simple update-rule [21]

$$\mathbf{p} \leftarrow \mathbf{p} - \frac{1}{\nu}\,\mathcal{U}^2(\mathbf{p}) \tag{1}$$

which has to be applied to all vertices of the mesh. Here, the umbrella-operator $\mathcal{U}$ is a discretization of the Laplace-operator [35]

$$\mathcal{U}(\mathbf{p}) \;=\; \frac{1}{n}\sum_{j=0}^{n-1}\mathbf{p}_j \;-\; \mathbf{p}$$

with $\mathbf{p}_j$ being the directly adjacent neighbor vertices of $\mathbf{p}$ (cf. Fig. 11). The umbrella-operator can be applied recursively leading to

$$\mathcal{U}^2(\mathbf{p}) \;=\; \frac{1}{n}\sum_{j=0}^{n-1}\mathcal{U}(\mathbf{p}_j) \;-\; \mathcal{U}(\mathbf{p})$$

as a discretization of the squared Laplacian. The coefficient ν in (1) is given by

$$\nu \;=\; 1+\frac{1}{n}\sum_{j}\frac{1}{n_j}$$

where n and n_j are the valences of the center vertex $\mathbf{p}$ and its jth neighbor $\mathbf{p}_j$ respectively.

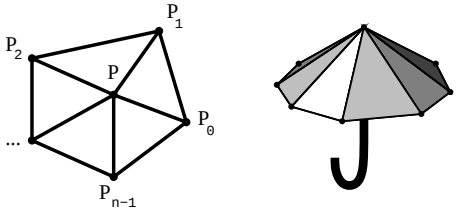


Figure 11: To compute the discrete Laplacian, we need the 1-neighborhood of a vertex $\mathbf{p}$ ($\rightarrow$ *umbrella-operator*).

In the context of discrete energy minimization, the iterative application of the update-rule (1) implements a Gauß-Seidel solver for the underlying linear system. From a more abstract point of view, the rule can also be considered as a mere relaxation operator that effectively filters out high frequency noise from the mesh [35].

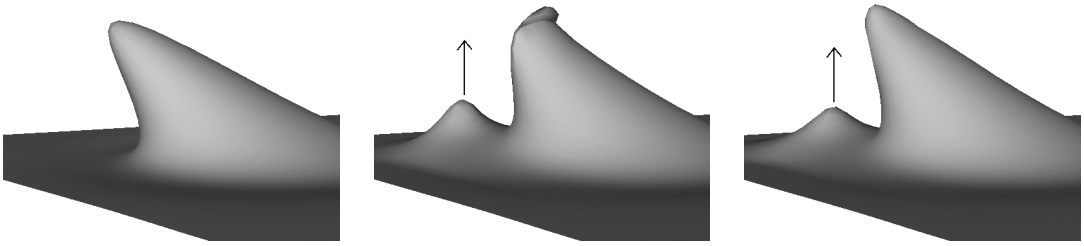


Figure 9: Non-projectable detail features are not reconstructed correctly. The original geometry (left) is modified by using a two-band decomposition in the center and a multi-band decomposition with five intermediate levels on the right.

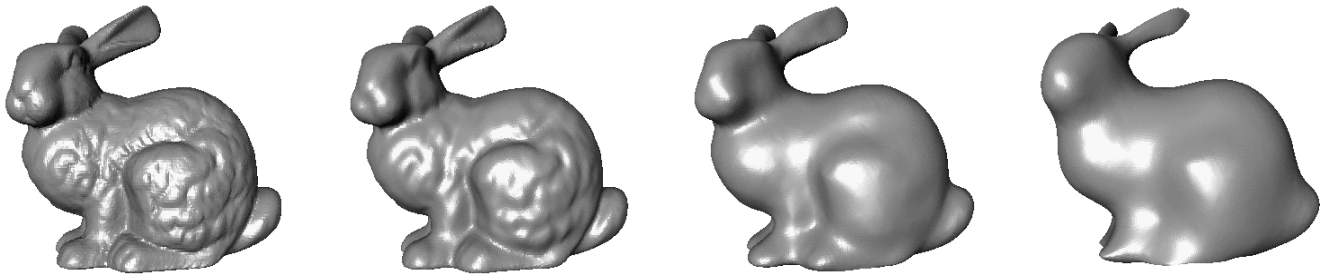


Figure 10: Four versions of the Stanford bunny. The smoother versions are generated by applying mesh decimation down to a certain target complexity and then re-inserting the vertices under minimization of some discrete fairness functional. The degree by which geometric detail is removed depends on the coarseness of the base mesh. Notice that all shown meshes have exactly the same connectivity.

3.1 Multi-level smoothing

A well-known negative result from numerical analysis is that straight forward iterative solvers like the Gauß-Seidel scheme are not appropriate for large sparse problems [33]. More sophisticated solvers exploit knowledge about the *structure* of the problem. The important class of multi-grid solvers achieve linear running times in the number of degrees of freedom by solving the same problem on grids with different step sizes and combining the approximate solutions [16].

For difference (= discrete differential) equations of elliptic type the Gauß-Seidel iteration matrices have a special eigenstructure that causes high frequencies in the error to be attenuated very quickly while for lower frequencies no practically useful rate of convergence can be observed. Multi-level schemes hence solve a given problem on a very coarse scale first. This solution is used to predict initial values for a solution of the same problem on the next refinement level. If these predicted values have only small deviations from the true solution in low-frequency sub-spaces, then Gauß-Seidel performs well in reducing the remaining high-frequency error. The alternating refinement and smoothing leads to highly efficient *variational subdivision schemes* [19] which generate fair high-resolution meshes with a rate of several thousand triangles per second (linear complexity!).

We can apply the same principle to hierarchical mesh structures which are generated from fine-to-coarse. Instead of iteratively solving the discretized optimization problem on the finest level, we solve it on coarser intermediate levels first and then use the coarse solutions to estimate better starting values for the iterative solver on the finer levels.

A complete V-cycle multi-grid solver recursively applies operators $\Phi_i = \Psi P \Phi_{i-1} R \Psi$ where the first (right) Ψ is a generic (pre-) smoothing operator — a Gauß-Seidel scheme in our case. R is a restriction operator to go one level coarser. This is where the mesh decimation comes in. On the coarser level, the same scheme is applied recursively, Φ_{i-1}, until on the coarsest level the number of degrees of freedom is small enough to solve the system directly (or any other stopping criterion is met). On the way back-up, the prolongation operator P inserts the previously removed vertices to go one level finer again. P can be considered as a non-regular subdivision operator which has to predict the positions of the vertices in the next level's solution. The re-subdivided mesh is an approximative solution with mostly high frequency error. (Post-)smoothing by some more iterations Ψ removes the noise and yields the final solution.

In our particular setting of thin-plate optimization on fine-to-coarse hierarchies, the Ψ-operator is simply the update-rule (1) and the restriction operator is a sequence of edge-collapse or vertex removal steps which are performed by the mesh decimation algorithm. The prolongation operator re-inserts the vertices. Since the prolongation operator can be designed to insert the new vertices to a locally optimal position, i.e., the center of gravity of its direct neigbors such that $\mathcal{U}(\mathbf{p}) = 0$, there is no need to actually perform any pre-smoothing. In fact, the whole multi-level smoothing algorithm reduces to mesh decimation down to a certain resolution and then alternating the re-inserting and Gauß-Seidel smoothing. Another consequence is that more sophisticated W-cycle schedules are very unlikely to improve the convergence of the algorithm.

The are several algorithmic parameters in this generic multi-level scheme. First, we have to choose the number of Gauß-Seidel steps which are performed on every level. As this is the most time consuming step of the algorithm and since our goal is to run the optimization in real-time with a prescribed number of frames per second, we cannot allow the iteration to proceed until the residuum drops below some given threshold. We rather perform a fixed number of iterations on each level. By adjusting that number we directly trade the quality of the resulting mesh for the speed of the algorithm.

Another algorithmic parameter is the number of hierarchy levels. The two extreme positions are either to re-insert all vertices and then perform Gauß-Seidel on the finest level only or to apply (1) after the insertion of every single vertex. From a practical point of view, the upper bound for the granulatity of hierarchy levels is reached if the vertices which are inserted when going from level $\mathcal{M}_i$ to $\mathcal{M}_{i+1}$, are *independent* from each other, i.e., their topological distance is larger than some threshold. This is because the local update operation (1) propagates geometric changes very slowly. An alternative to combining a sequence of independent vertex splits

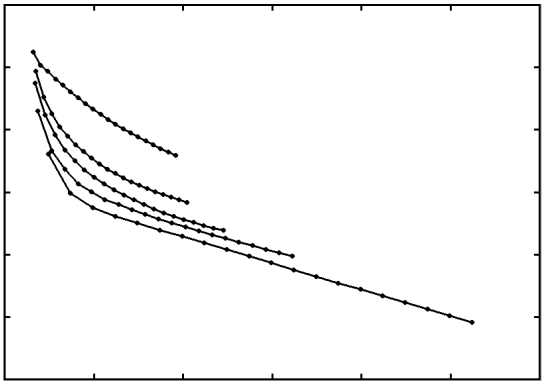


Figure 12: This diagram shows the logarithm of the approximation error (vertical axis) vs. the computation time (horizontical axis). The knots on each polygon mark the measurements for a different number of Gauß-Seidel iterations $(1,\ldots,20)$. The different polygons connect the measurements for the same number of hierarchy levels (from bottom to top: $27, 14, 9, 7, 6$ levels). The monotony of the curves shows that for a fixed amount of computation time (vertical line) or a prescribed approximation error (horizontal line) the multi-level smoothing schedule with the higher number of levels always outperforms the others.

(or edge collapses) is proposed in [15] where the local smoothing operator is applied only in the vicinity of the newly inserted vertex.

Since the eigenstructure of the Gauß-Seidel iteration matrix and hence the convergence behavior of the generalized multi-level scheme strongly depends on the actual connectivity of the mesh, we cannot derive general estimates for the convergence rates. Nevertheless we can analyze the typical behavior of the multi-level smoothing on fine-to-coarse hierarchies by numerical experiments. We made some experiments where we performed the multi-level smoothing with a varying number of hierarchy levels and Gauß-Seidel iterations per level. The results are shown in Fig. 12.

Obviously the approximation error decreases with increasing number of Gauß-Seidel steps and with increasing number of levels but also the computational costs become higher. When using the multi-level smoothing in practical applications we typically prescribe the maximum time or the maximum approximation error, i.e., we want to find the best approximation within a given period of time or we want to find a solution with a prescribed approximation error as fast as possible. In Fig. 12 these constraints correspond to vertical or horizontical lines respectively.

As a general rule of thumb it turned out that more Gauß-Seidel iterations per level only marginally improve the final result. This is due to the bad convergence on each individual level. Better results can be achieved if more hierarchy levels are used but with fewer iterations per level.

Notice that the number of *topological* hierarchy levels as one algorithmic parameter in the multi-level smoothing scheme has nothing to do with the number of *geometric* hierarchy levels in the geometric multi-band decomposition (topological vs. geometric hierarchy). One is used to make the detail reconstruction more robust while the other is used to accelerate the global optimization procedure.

3.2 Boundary constraints

In order to enable intuitive modeling functionality we have to implement a simple and effective interaction metaphor. As the shape of the mesh is controlled by discrete curvature minimization, the most simple way to influence the result is by imposing appropriate boundary constraints. These constraints determine the support and the shape of the modification.

In [21] we proposed a simple metaphor where the designer starts by marking an arbitrary region on the mesh $\mathcal{M}_m$. In fact, she picks a sequence of surface points (not necessarily vertices) on the triangle mesh and these points are connected either by geodesics or by projected lines. The strip of triangles $\mathcal{S}$ which are intersected by the geodesic (projected) boundary polygon separates an interior region $\mathcal{M}_*$ and an exterior region $\mathcal{M}_i \setminus (\mathcal{M}_* \cup \mathcal{S})$. The interior region $\mathcal{M}_*$ is to be affected by the following modification.

A second polygon (not necessarily closed) is marked within the first one to define the *handle*. The semantics of this arbitrarily shaped handle is quite similar to the handle metaphor in [37]: when the designer moves or scales the virtual tool, the same geometric transformation is applied to the rigid handle and the surrounding mesh $\mathcal{M}_*$ follows according to a constrained energy minimization principle.

The freedom to define the boundary strip $\mathcal{S}$ and the handle geometry allows the designer to build "custom tailored" basis functions for the intended modification. Particularly interesting is the definition of a *closed* handle polygon which allows to control the characteristics of a bell-shaped dent: For the same region $\mathcal{M}_*$, a tiny ring-shaped handle in the middle causes a rather sharp peak while a bigger ring causes a wider bubble (cf. Fig 13). Notice that the mesh vertices in the interior of the handle polygon also move according to the energy minimization.

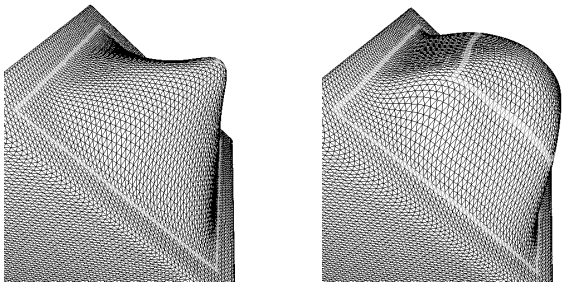


Figure 13: Controlling the characteristics of the modification by the size of a closed handle polygon.

Since we are working on triangle meshes, the energy minimization on $\mathcal{M}_*$ is done by discrete fairing techniques. To enable realtime editing we use the multi-level smoothing approach (cf. Fig. 14). While Fig. 15 depicts the general modeling set-up for a geometric two-band decomposition, more intermediate levels can be used for the detail reconstruction if the original geometry cannot be projected onto the optimized mesh without self-intersections. The boundary triangles $\mathcal{S}$ provide the correct C^1 boundary conditions for minimizing the thin plate energy functional. The handle imposes additional interpolatory constraints on the location only — derivatives should not be affect by the handle.

In [21] we proposed to impose the handle interpolation constraints to the optimization problem by simply freezing every other vertex of the handle polygon. On one hand this is a simple way to implement interpolation constraints, one the other hand it prevents any influence on the tangent plane.

Another way to impose interpolation constraints is to prescribe them for *centers* of triangles. Such constraints can easily be embedded into the iterative energy minimization by allowing Gauß-Seidel updates for *all* vertices and re-enforcing the constraints after each iteration. This means that we shift the constrained triangles such that their centers coincide with the interpolation points after every smoothing cycle. By *shifting* the triangles without rotation we allow the tangent at the interpolation point to be controlled by the optimization process (and hence we do *not* impose a C^1 constraint). Fig 16 demonstrates that the convergence behavior is much better for this kind of interpolation constraint compared to freezing vertices.

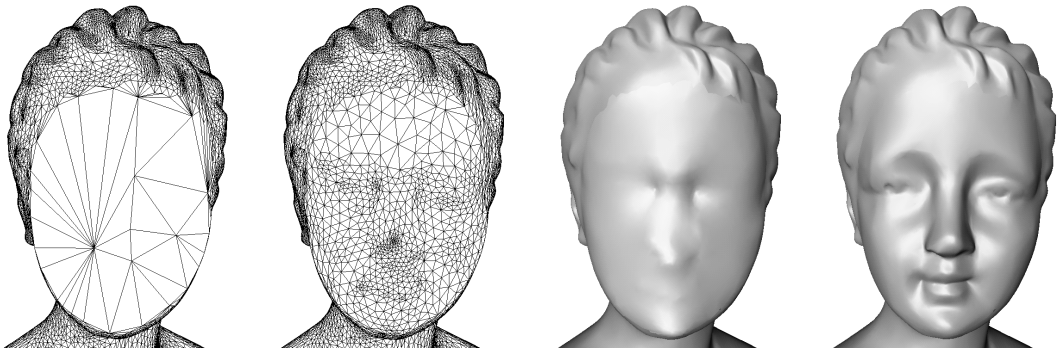


Figure 14: During the real-time modeling, the multi-level smoothing always starts on the coarses level down to which $\mathcal{M}_*$ is reduced (left). We alternate vertex re-insertion and Gauß-Seidel smoothing (center left) until the mesh with minimum thin plate energy with respect to the current interpolation constraints is found (center right). To this smooth mesh, we add the detail coefficients to reconstruct the modified surface (right).

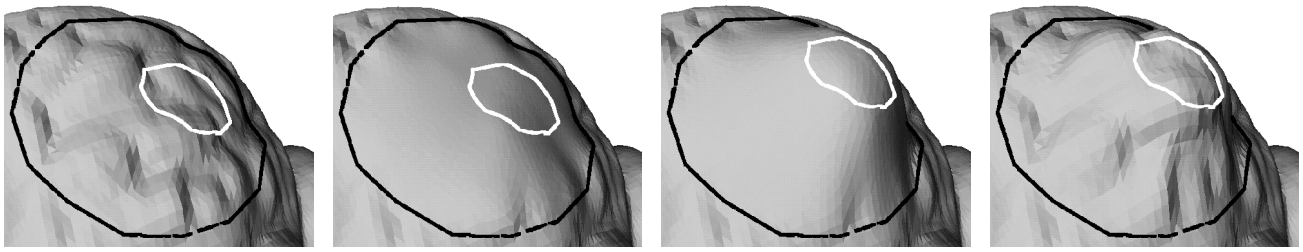


Figure 15: A flexible metaphor for multiresolution edits. On the left, the original mesh is shown. The black line defines the region of the mesh which is subject to the modification. The white line defines the handle geometry which can be moved by the designer. Both boundaries can have an arbitrary shape and hence they can, e.g., be aligned to geometric features in the mesh. The boundary and the handle impose C^1 and C^0 boundary conditions to the mesh and the smooth version of the original mesh is found by applying discrete fairing while observing these boundary constraints. The center left shows the result of the curvature minimization (the boundary and the handle are interpolated). The geometric difference between the two left meshes is stored as detail information with respect to loacal frames. Now the designer can move the handle polygon and this changes the boundary constraints for the curvature minimization. Hence the discrete fairing generates a modified smooth mesh (center right). Adding the previously stored detail information yields the final result on the right. Since we can apply fast multi-level smoothing when solving the optimization problem, the modified mesh can be updated with several frames per second during the modeling operation. Notice that all four meshes have the same connectivity.

4 Conclusions and future work

We explained how to address various technical problems when using the fine-to-coarse multiresolution mesh representation which has been proposed in [21]. We presented a new way to encode the geometric detail information by using a continuous normal field on the low-frequency geometry. This makes the detail reconstruction more robust than other local frame based techniques. We also showed how the use of several intermediate levels of detail enables the handling of geometric configurations which cannot be processed correctly with a plain two-band decomposition. We further investigated the influence of various algorithmic parameters onto the overall performance of multi-level smoothing schemes when applied to a fine-to-coarse hierarchy on arbitrary meshes.

In our current implementation of the multiresolution mesh modeling technique, the supporting mesh which is controlled by constrained optimization during the interactive modeling has the same connectivity as the original mesh. In the future it might be promising to drop this restriction. We could improve the stability and convergence speed of the multi-level scheme by using regularly connected meshes instead. Moreover, this could provide the possibility to use ”better” local parameterizations for the discrete Laplace operator with reasonable computational effort [4, 15]. However, imposing the boundary conditions into the optimization would become more involved.

References

[1] BONNEAU, G., HAHMANN, S., AND NIELSON, G. Blac-wavelets: a multiresolution analysis with non-nested spaces. In *Visualization Proceedings* (1996), pp. 43–48.

[2] CATMULL, E., AND CLARK, J. Rcursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design 10*, 6 (Nov. 1978), 239–248.

[3] CELNIKER, G., AND GOSSARD, D. Deformable curve and surface finite elements for free-form shape design. In *Computer Graphics (SIGGRAPH 91 Proceedings)* (July 1991), pp. 257–265.

[4] DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Computer Graphics (SIGGRAPH 99 Proceedings)* (1999), pp. 317–324.

[5] DOO, D., AND SABIN, M. Behaviour of recursive division surfaces near extraordinary points. *CAD* (1978).

[6] DYN, N., LEVIN, D., AND GREGORY, J. A. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics 9*, 2 (April 1990), 160–169.

[7] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. In *Computer Graphics (SIGGRAPH 95 Proceedings)* (1995), pp. 173–182.

[8] FINKELSTEIN, A., AND SALESIN, D. H. Multiresolution Curves. In *Computer Graphics (SIGGRAPH 94 Proceedings)* (July 1994), pp. 261–268.

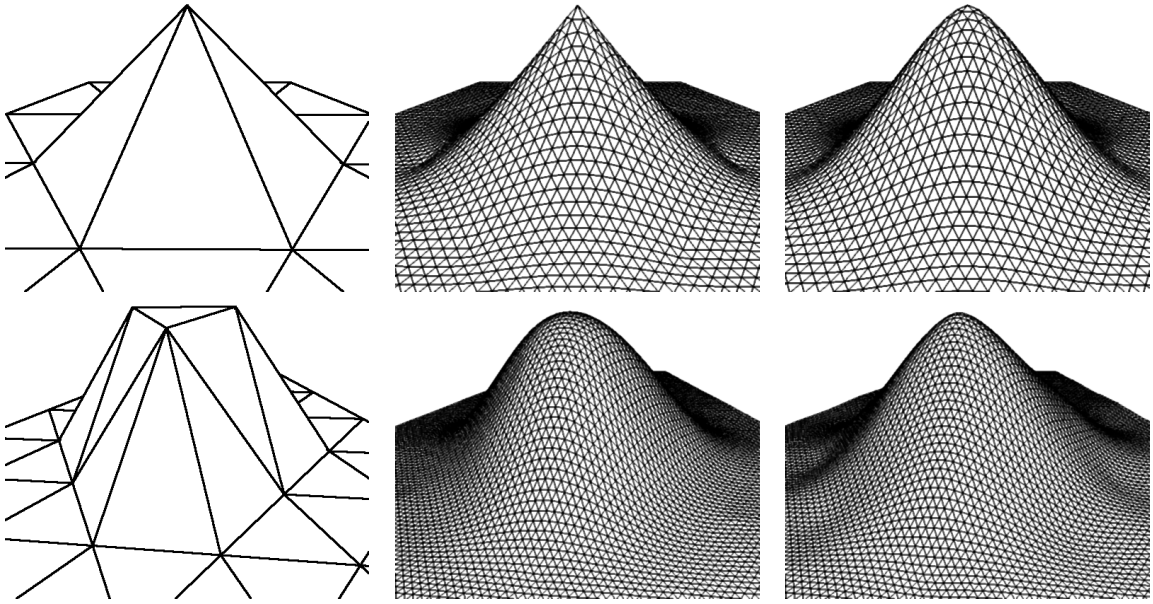


Figure 16: In the top row discrete thin plate energy is minimized while imposing point interpolation constraint at the vertices of the original mesh (left: original, center: after 5 multi-level Gauß-Seidel iterations, right: exact result). In the bottom row the interpolation constraints are imposed at the centers of the triangles (left: original, center: after 5 multi-level Gauß-Seidel iterations, right: exact result).

[9] FOLEY, VAN DAM, FEINER, AND HUGHES. *Computer Graphics*. Addison Wesley, 1990.

[10] FORSEY, D. R., AND BARTELS, R. H. Hierarchical B-spline refinement. In *Computer Graphics (SIGGRAPH 88 Proceedings)* (1988), pp. 205–212.

[11] FORSEY, D. R., AND BARTELS, R. H. Surface Fitting with Hierarchical Splines. *ACM Transactions on Graphics 14*, 2 (April 1995), 134–161.

[12] GARLAND, M., AND HECKBERT, P. S. Surface Simplification Using Quadric Error Metrics. In *Computer Graphics (SIGGRAPH 97 Proceedings)* (1997), pp. 209–218.

[13] GORTLER, S. J., AND COHEN, M. F. Hierarchical and Variational Geometric Modeling with Wavelets. In *Proceedings Symposium on Interactive 3D Graphics* (May 1995).

[14] GREINER, G. Variational design and fairing of spline surfaces. *Computer Graphics Forum 13* (1994), 143–154.

[15] GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. Multiresolution signal processing for meshes. In *Computer Graphics (SIGGRAPH 99 Proceedings)* (1999), pp. 325–334.

[16] HACKBUSCH, W. *Multi-Grid Methods and Applications*. Springer Verlag, Berlin, 1985.

[17] HOPPE, H. Progressive Meshes. In *Computer Graphics (SIGGRAPH 96 Proceedings)* (1996), pp. 99–108.

[18] KOBBELT, L. Interpolatory Subdivision on Open Quadrilateral Nets with Arbitrary Topology. In *Computer Graphics Forum, Proceedings of Eurographics '96* (1996), pp. C407–C420.

[19] KOBBELT, L. Discrete fairing. In *Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces* (1997), pp. 101–131.

[20] KOBBELT, L., CAMPAGNA, S., AND SEIDEL, H.-P. A general framework for mesh decimation. In *Proceedings of the Graphics Interface conference '98* (1998).

[21] KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. Interactive multi-resolution modeling on arbitrary meshes. In *Computer Graphics (SIGGRAPH 98 Proceedings)* (1998), pp. 105–114.

[22] KOBBELT, L., VORSATZ, J., LABSIK, U., AND SEIDEL, H.-P. A shrink wrapping approach to remeshing polygonal surfaces. In *Computer Graphics Forum, Proceedings of Eurographics '99* (1999).

[23] KRISHNAMURTHY, V., AND LEVOY, M. Fitting smooth surfaces to dense polygon meshes. In *Computer Graphics (SIGGRAPH 96 Proceedings)* (1996), pp. 313–324.

[24] LEE, A., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. Multiresolution adaptive parameterization of surfaces. In *Computer Graphics (SIGGRAPH 98 Proceedings)* (1998), pp. 95–104.

[25] LOOP, C. Smooth spline surfaces over irregular meshes. In *Computer Graphics Proceedings* (1994), Annual Conference Series, ACM Siggraph, pp. 303–310.

[26] LOUNSBERY, M., DEROSE, T., AND WARREN, J. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM Transactions on Graphics 16*, 1 (January 1997), 34–73.

[27] LUEBKE, D., AND ERIKSON, C. View-Dependent Simplification of Arbitrary Polygonal Environments. In *Computer Graphics (SIGGRAPH 97 Proceedings)* (1997), pp. 199–208.

[28] MORETON, H., AND SÉQUIN, C. Functional optimization for fair surface design. In *Computer Graphics (SIGGRAPH 92 Proceedings)* (1992), pp. 167–176.

[29] ROSSIGNAC, J. Simplification and Compression of 3D Scenes, 1997. Tutorial Eurographics '97.

[30] SAPIDIS, N. E. *Designing Fair Curves and Surfaces*. SIAM, 1994.

[31] SCHRÖDER, P., AND SWELDENS, W. Spherical wavelets: Efficiently representing functions on the sphere. In *Computer Graphics (SIGGRAPH 95 Proceedings)* (1995), pp. 161–172.

[32] SCHROEDER, W. J., ZARGE, J. A., AND LORENSEN, W. E. Decimation of Triangle Meshes. In *Computer Graphics (SIGGRAPH 92 Proceedings)* (1992), pp. 65–70.

[33] STOER, J. *Einführung in die Numerische Mathematik I.* Springer Verlag, 1983.

[34] STOLLNITZ, E., DEROSE, T., AND SALESIN, D. *Wavelets for Computer Graphics*. Morgan Kaufmann Publishers, 1996.

[35] TAUBIN, G. A signal processing approach to fair surface design. In *Computer Graphics (SIGGRAPH 95 Proceedings)* (1995), pp. 351–358.

[36] TURK, G. Re-Tiling Polygonal Surfaces. In *Computer Graphics (SIGGRAPH 92 Proceedings)* (1992), pp. 55–64.

[37] WELCH, W., AND WITKIN, A. Variational surface modeling. In *Computer Graphics (SIGGRAPH 92 Proceedings)* (1992), pp. 157–166.

[38] WELCH, W., AND WITKIN, A. Free–Form shape design using triangulated surfaces. In *Computer Graphics (SIGGRAPH 94 Proceedings)* (1994), A. Glassner, Ed., pp. 247–256.

[39] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interpolating subdivision for meshes with arbitrary topology. In *Computer Graphics (SIGGRAPH 96 Proceedings)* (1996), pp. 189–192.

[40] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interactive multiresolution mesh editing. In *Computer Graphics (SIGGRAPH 97 Proceedings)* (1997), pp. 259–268.

Multiresolution Signal Processing for Meshes,
by I. Guskov, W. Sweldens, and P. Schroder,
Siggraph'99.

Multiresolution Signal Processing for Meshes

Igor Guskov
Princeton University

Wim Sweldens
Bell Laboratories

Peter Schröder
Caltech

Figure 1: *Mount Meshmore (Design Khrysaundt Koenig).*

Abstract

We generalize basic signal processing tools such as downsampling, upsampling, and filters to irregular connectivity triangle meshes. This is accomplished through the design of a non-uniform relaxation procedure whose weights depend on the geometry and we show its superiority over existing schemes whose weights depend only on connectivity. This is combined with known mesh simplification methods to build subdivision and pyramid algorithms. We demonstrate the power of these algorithms through a number of application examples including smoothing, enhancement, editing, and texture mapping.

CR Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling - *hierarchy and geometric transformations, object hierarchies*; I.4.3 [**Image Processing and Computer Vision**]: Enhancement - *filtering, geometric correction, sharpening and deblurring, smoothing*; G.1.2 [**Numerical Analysis**]: Approximation - *approximation of surfaces and contours, wavelets and fractals*

Additional Keywords: Meshes, subdivision, irregular connectivity, surface parameterization, multiresolution, wavelets, Laplacian Pyramid.

1 Introduction

3D range sensing is capable of producing detailed and densely sampled triangular meshes of high quality. Increasing deployment of this technology in the automotive and entertainment industries, as well as many other areas, has fueled the need for algorithms to process such datasets. Examples include editing, simplification, denoising, compression, and finite element simulation.

In the case of regularly sampled data, for example images, basic signal processing tools such as filtering, subsampling, and upsampling exist. These can be used to build subdivision and pyramid algorithms, which are useful in many applications. Our goal is the construction of signal processing style analyses and algorithms for triangle meshes.

Building the elements of a signal processing toolbox for meshes is not immediately straightforward since there are essential differences between images, for example, and meshes. Images are functions defined on Euclidean ("flat") geometry and are almost always sampled on a regular grid. Consequently, algorithms such as subsampling and upsampling are straightforward to define, and uniform filtering methods are appropriate. This makes Fourier analysis an elegant and efficient tool for the construction and analysis of signal processing algorithms.

In contrast, triangle meshes of arbitrary connectivity form an inherently irregular sampling setting. Additionally we are dealing with general 2-manifolds as opposed to a Euclidean space. Consequently new algorithms need to be developed which incorporate the fundamental differences between images and meshes.

A crucial first observation concerns the difference between geometric and parametric smoothness. *Geometric* smoothness measures how much triangle normals vary over the mesh. Geometric

Terminology

In order to describe our contribution and its relationship to existing work we need to set some terminology. Among triangulations we distinguish three types

- **Regular:** every vertex has degree six;
- **Irregular:** vertices can have any degree;
- **Semi-regular:** formed by starting with a coarse irregular triangulation and performing repeated quadrisection on all triangles. Coarse vertices have arbitrary degree while all other vertices have degree six.

In all cases we assume that any triangulation is a proper 2-manifold with boundary. On the boundary regular vertices have degree four. Each of these triangulations call for different filtering and subdivision algorithms:

- **Uniform:** fixed coefficient stencils everywhere; typically used only on regular triangulations;
- **Non-uniform:** filter coefficients depend on the connectivity *and* geometry of the triangulation;
- **Semi-uniform:** coefficients of filters depend only on the (local) connectivity of the triangulation; typically used on semi-regular triangulations.

Using our terminology, for example, traditional subdivision [22] uses semi-uniform filters on semi-regular meshes.

smoothness implies that there exists *some* smooth (differentiable) parameterization of the mesh. However, any particular parameterization may well be non-smooth. The smoothness of the parameterizations is important in most numerical algorithms, which work only with the coordinate functions the user provides. The algorithms' behavior, such as convergence rates or the quality of the results, generally depends strongly on the smoothness of the coordinate functions.

In the regular setting of an image, or the knots of a uniform tensor product spline, we may simply use a uniform parameterization and will get parametric smoothness wherever there is geometric smoothness. In the irregular triangle mesh setting there is a priori no such "obvious" parameterization. In this case using a uniformity assumption leads to parametric non-smoothness with undesirable consequences for further processing. One approach to remedy this situation is the use of remeshing [8, 19], which maintains the original geometric smoothness, but improves the sampling to vary smoothly. This enables subsequent treatment with a uniform parameter assumption without detrimental effects. Here we wish to build tools which work on the original meshes directly.

To understand the role of the parameterization further, consider traditional subdivision [22], such as Loop or Catmull-Clark. In the signal processing context, subdivision can be seen as upsampling followed by filtering. One starts with an arbitrary connectivity mesh and uses regular upsampling techniques such as triangle quadrisection to obtain a semi-regular triangulation. The subdivision weights depend only on connectivity, not geometry. Such stencils can be designed with existing Fourier or spectral techniques. These schemes result in geometrically smooth limit surfaces with smooth semi-uniform parameterizations. Because traditional subdivision is only concerned with *refinement* one has the freedom to choose regular upsampling, and semi-uniform schemes suffice.

The picture changes entirely if we wish to compute a mesh pyramid, i.e., we want to be able to *coarsify* a given fine irregular mesh and later *refine* it again. We then need to filter, downsample, upsample and filter again. The downsampling typically involves a standard mesh simplification hierarchy. When subdividing back, we want to build a mesh with the *same* connectivity as the original mesh and a smooth geometry. This time the upsampling procedure is determined by reversing the previously computed simplification hierarchy. We no longer have a choice as in the classical subdivision setting. Consequently the filters used before downsampling and after upsampling should use non-uniform weights, which depend on the local parameterization. The challenge is to ensure that these local parameterizations are smooth so that subsequent algorithms act on the geometry and not some potentially bad parameterization.

1.1 Contributions

In this paper we present a series of non-uniform signal processing algorithms designed for irregular triangulations and show their usefulness in several application areas. Specifically, we make the following contributions:

- We show how the non-uniform subdivision algorithm of Guskov [12] can be used for geometric smoothing of triangle meshes. Our scheme is fast, local, and straightforward to implement.
- We use the smoothing algorithm combined with existing hierarchy methods to build subdivision, pyramid, and wavelet algorithms for irregular connectivity meshes.
- We show how these signal processing algorithms can be used in applications such as smoothing, enhancement, editing, animation, and texture mapping.

1.2 Related Work

In our approach we draw on observations made by researchers in several different areas. These include classical subdivision [22], which we generalize to the irregular setting with the help of mesh simplification [13] and careful attention to the role of smooth parameterizations. Parameterizations were examined in the context of remeshing [19, 8, 9], texture mapping (e.g., [20]), and variational modeling [16, 28, 21]. One area which employs these elements is hierarchical editing for semi-regular [29] and irregular meshes [18].

Signal processing as an approach to surface fairing in the irregular setting was first considered by Taubin [26, 27]. He defines frequencies as the eigenvectors of a discrete Laplacian generalized to irregular triangulations. The resulting smoothing schemes were used to denoise meshes, apply smooth deformations, and build semi-uniform subdivision over irregular meshes. Our approach is related to Taubin's and can be seen as a generalization to the non-uniform setting. In particular we build a smoothing method by minimizing multivariate finite differences. Together with progressive mesh simplification [14] we use these to define a non-uniform subdivision scheme and pyramid algorithm on top of an irregular mesh hierarchy.

Progressive meshes and a semi-uniform discrete Laplacian were used by Kobbelt et al. [18] to perform multiresolution editing on irregular meshes. Given some region of the mesh, discrete fairing is used to compute a smoothed version with the same connectivity. This smoothed region is deformed and offsets to the original mesh are added back in. Kobbelt discusses the issue of geometric vs. parametric smoothing. Smoothing of irregular meshes based on uniform approximations of the Laplacian results in vertex motion "within" the surface, even in a perfectly planar triangulation. It is geometrically smooth, yet the parameter functions appear non-smooth due to a non-uniform parameterization. This has undesirable effects in a hierarchical setting in which fine levels are defined as offsets ("details") from a coarse level: using the difference between topologically corresponding vertices in the original and smoothed mesh can lead to large detail vectors [18, Figure 4]. To minimize the size of detail vectors they employed a search procedure to find the nearest vertex on the smoothed mesh to a given vertex on the original mesh. This diminishes the advantage of having a smoothed version with the exact same connectivity. In contrast, our non-uniform smoothing scheme affects only geometric smoothness and so does not need a search procedure. We will present two ways

in which our scheme can be used for editing: one is based on multiresolution and combines the work of Kobbelt et al. [18] with the ideas of Zorin et al. [29]. The other method relies on defining vector displacement fields with controllable decay similar to the ideas presented in the work of Singh and Fiume [23].

We construct our subdivision scheme by designing a non-uniform relaxation operator which minimizes second differences. This is motivated by the smoothness analysis of the 1D irregular setting [2]. This analysis relies on the decay of divided differences, carefully designed to respect the underlying parameterization. These ideas were extended to the multi-variate setting in [12] and we employ them here. While the schemes we present have many nice properties and work very well in practice, we note that their analytic smoothness is currently unknown.

2 Signal Processing Algorithms

Before describing the actual numerical algorithms we begin with some remarks regarding different settings and establish our notation for triangulations and difference operators defined on them.

Coordinate Functions To describe our algorithms we must distinguish between two settings: the functional and the surface setting. The *functional setting* deals with a function $g(u, v)$ of two independent variables in the plane. The dependent variable g can be visualized as height above the (u, v) parameter plane. In practice we only have discrete data $g_i = g(u_i, v_i)$. The sample points (u_i, v_i) are triangulated in the plane and this connectivity can be transferred to the corresponding points (u_i, v_i, g_i) in $\mathbf{R}^3$. The canonical example of this is a terrain model.

The *surface setting* deals with a triangle mesh of arbitrary topology and connectivity embedded in $\mathbf{R}^3$ with vertices $p_i = (x_i, y_i, z_i)$. It is important to treat all three coordinates x, y, and z as *dependent* variables with independent parameters u and v, giving us three functional settings. The independent parameters are typically unknown and must be estimated. Algorithms to estimate *global* smooth parameterizations are described in [19, 8, 9, 20]. We require only *local* parameterizations which are consistent over the support of a small filter stencil.

Triangulations To talk about local neighborhoods of vertices within the mesh it is convenient to describe the topological and geometric aspects of a mesh separately. We use notation inspired by [24]. A triangle mesh is denoted as a pair $(\mathcal{P}, \mathcal{K})$, where $\mathcal{P}$ is a set of N point positions $\mathcal{P} = \{p_i \in \mathbf{R}^3 \mid 1 \le i \le N\}$ (either $p_i = (u_i, v_i, f_i)$ in a functional setting or $p_i = (x_i, y_i, z_i)$ in the surface setting), and $\mathcal{K}$ is an *abstract simplicial complex* which contains all the topological, i.e., adjacency information. The complex $\mathcal{K}$ is a set of subsets of $\{1, \ldots, N\}$. These subsets are called simplices and come in three types: vertices $v = \{i\} \in \mathcal{V}$, edges $e = \{i, j\} \in \mathcal{E}$, and faces $f = \{i, j, k\} \in \mathcal{F}$, so that $\mathcal{K} = \mathcal{V} \cup \mathcal{E} \cup \mathcal{F}$. Two vertices i and j are *neighbors* if $\{i, j\} \in \mathcal{E}$. The 1-ring neighbors of a vertex i form a set $\mathcal{V}_1(i) = \{j \mid \{i, j\} \in \mathcal{E}\}$ (see Figure 2, left). $K_i = \#\mathcal{V}_1(i)$ is the *degree* of i. The edges from i to its neighbors form a set $\mathcal{E}_1(i) = \{\{i, j\} \mid j \in \mathcal{V}_1(i)\}$. A 1-ring neighborhood with flaps is shown in Figure 2 (middle). Its vertices except the center vertex form a set $\mathcal{V}_2(i)$ and its interior edges form a set $\mathcal{E}_2(i)$. Finally, the neighborhood $\omega(e)$ of an edge (see Figure 2) is formed by the 4 vertices of its incident triangles.

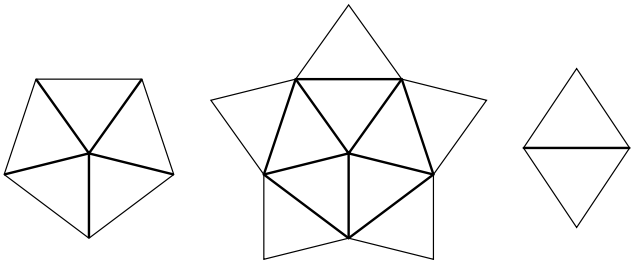


Figure 2: *Left: 1-ring neighborhood. The vertices except the center one form $\mathcal{V}_1(i)$ and the bold edges form $\mathcal{E}_1(i)$. Middle: 1-ring with flaps. The vertices except the center one form $\mathcal{V}_2(i)$ and the bold edges form $\mathcal{E}_2(i)$. Right: Edge neighborhood. The four vertices of the incident triangles form $\omega(e)$.*

The *geometric realization* $\varphi(s)$ of a simplex s is defined as the strictly convex hull of the points p_i with $i \in s$. The polyhedron $\varphi(\mathcal{K})$ is defined as $\cup_{s \in \mathcal{K}} \varphi(s)$ and consists of points, segments, and triangles in $\mathbf{R}^3$.

2.1 Divided Differences in the Functional Setting

Our relaxation algorithm relies on minimizing divided differences. In the one dimensional setting divided differences are straightforward to define, but for multivariate settings many approaches are possible (see for example [10, 4, 3]). An approach that was developed specifically with subdivision in mind is described in [12] and we use it here for our purposes.

Consider a face $f = \{i, j, k\}$ and the triangle $t = \varphi(f)$ where $p_i = (u_i, v_i, g_i)$. Then the first order divided difference of g at f is simply the gradient of the piecewise linear spline interpolating g denoted by $\nabla_f g = (\partial g/\partial u, \partial g/\partial v)$. Note that the gradient depends on the parameter locations (u_i, v_i) and converges in the limit to the first partial derivatives. If we create a three vector by adding a third component equal to 1, we obtain the normal $n_f = (-\partial g/\partial u, -\partial g/\partial v, 1)$ to the triangle t. Conversely, the gradient is the projection of the normal in the parameter plane. Consequently the gradient is zero only if the triangle t is horizontal ($g_i = g_j = g_k$).

Second order differences are defined as the difference between two normals on neighboring triangles and can be associated with the common edge (see Figure 3, left). Consider an edge $e = \{j, k\}$ with its two incident faces $f_1 = \{j, k, l_1\}$ and $f_2 = \{j, k, l_2\}$ (see Figure 2, right). Compute the difference between the two normals $m_e = n_{f_2} - n_{f_1}$. Given that the two normals are orthogonal to $\varphi(e)$ so is their difference m_e (see Figure 3, right). But the third component of m_e is zero, hence m_e itself lies in the parameter plane, which also contains the segment between $(u_j, v_j, 0)$ and $(u_k, v_k, 0)$. This implies that m_e is orthogonal to the segment and hence only its signed magnitude matters (see Figure 3).

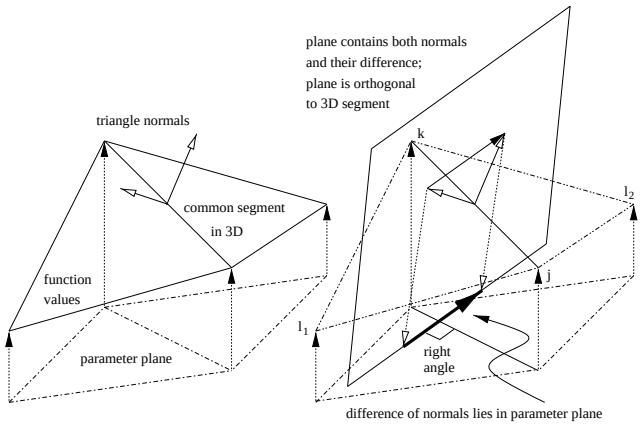


Figure 3: *In the functional setting triangles are erected over the parameter plane. Their normals generate a plane orthogonal to the edge in 3-space. Any vector in that plane which is also in the parameter plane must be at right angles with the parameter plane segment. Hence D_e^2 is orthogonal to $(u_j, v_j) - (u_k, v_k)$.*

This argument justifies defining the second order difference $D_e^2 g$ as the component of m_e orthogonal to the segment in the parameter plane. $D_e^2 g$ depends on four function values at vertices $\omega(e) = \{j, k, l_1, l_2\}$. Since all operations to compute $D_e^2 g$ are linear (gradient, difference, and projection) so is the entire expression

$$D_e^2 g = \sum_{l \in \omega(e)} c_{e,l} g_l.$$

The coefficients are given by

$$c_{e,l_1} = \frac{L_e}{A_{[l_1,k,j]}}, \quad c_{e,l_2} = \frac{L_e}{A_{[l_2,j,k]}},$$

$$c_{e,j} = -\frac{L_e A_{[k,l_2,l_1]}}{A_{[l_1,k,j]}\, A_{[l_2,j,k]}}, \quad c_{e,k} = -\frac{L_e A_{[j,l_1,l_2]}}{A_{[l_1,k,j]}\, A_{[l_2,j,k]}}, \quad (1)$$

where $A_{[k_1,k_2,k_3]}$ is the signed area of the triangle formed by (u_{k_1}, v_{k_1}), (u_{k_2}, v_{k_2}), (u_{k_3}, v_{k_3}); and L_e is the length of the segment between (u_j, v_j) and (u_k, v_k) [12]. All the parameterization information is captured in the edge length and signed triangle areas. Given that we later only use squares of D_e^2 the actual sign of the areas is not important as long as the orientations prescribed by (1) are consistent. Also, note that the second order difference operator is zero only if the two triangles lie in the same plane.

2.2 Relaxation in the Functional Setting

The central ingredient in our signal processing toolbox is a non-uniform relaxation operator. It generalizes the usual notion of a low pass filter. We begin by discussing the construction of such a relaxation operator in the functional setting.

The purpose of the relaxation operation is the minimization of second order differences. To this end we define a quadratic energy, which is an instance of a discrete fairing functional [16]

$$\mathbf{E} = \textstyle\sum_{e \in \mathcal{E}} (D_e^2 g)^2.$$

The relaxation is computed locally, i.e., for a given vertex i we compute a relaxed function value Rg_i based on neighboring function values g_j. Treating $\mathbf{E}$ as a function of a given g_i the relaxed value Rg_i is defined as the minimizer of $\mathbf{E}(g_i)$. Given that the stencil for D_e^2 consists of two triangles, all edges which affect $\mathbf{E}(g_i)$ belong to $\mathcal{E}_2(i)$ (see Figure 2, middle)

$$Rg_i = \arg\min \mathbf{E}(g_i) = \arg\min \textstyle\sum_{e \in \mathcal{E}_2(i)} (D_e^2 g)^2. \quad (2)$$

Since the functional is quadratic the relaxation operator is linear in the function values. To find the expression, write each of the $D_e^2 g$ with $e \in \mathcal{E}_2(i)$, i.e., all second differences depending on g_i, as a linear function of g_i

$$D_e^2 g = c_{e,i} g_i + \alpha_e \;\text{ with }\; \alpha_e = \textstyle\sum_{l \in \omega(e) \setminus \{i\}} c_{e,l} g_l.$$

Setting the partial derivative of $\mathbf{E}$ with respect to g_i equal to zero yields

$$Rg_i = -\left(\textstyle\sum_{e \in \mathcal{E}_2(i)} c_{e,i}\, \alpha_e\right) / \left(\textstyle\sum_{e \in \mathcal{E}_2(i)} c_{e,i}^2\right), \quad (3)$$

which can be rewritten as

$$Rg_i = \textstyle\sum_{j \in \mathcal{V}_2(i)} w_{i,j} g_j, \;\; w_{i,j} = -\frac{\sum_{\{e \in \mathcal{E}_2(i) | j \in \omega(e)\}} c_{e,i} c_{e,j}}{\sum_{e \in \mathcal{E}_2(i)} c_{e,i}^2}.$$

There are two ways to implement R which trade off speed versus memory. One can either precompute and store the $w_{i,j}$ and use the above expression or one can use (3) and compute R on the fly.

Note that if g is a linear function, i.e., all triangles lie in one plane, the fairing functional $\mathbf{E}$ is zero. Consequently linear functions are invariant under R. In particular R preserves constants from which we deduce that the $w_{i,j}$ sum to one.

To summarize, given an arbitrary but fixed triangulation in the parameter plane and function values g_i with the associated (u_i, v_i) coordinates, simple linear expressions describe first and second differences. The coefficients of these expressions depend on the parameterization. The relaxation operator R acts on individual function values to minimize the discrete second difference energy over the $\mathcal{E}_2(i)$ neighborhood of a given $p_i = (u_i, v_i, g_i)$, leaving linears invariant.

2.3 Relaxation in the Surface Setting

To apply the above relaxation in the surface setting we need to have parameter values (u, v) associated with every point in our mesh. Typically such parameter values are not available and we must compute them. One possible solution is to compute a global parameterization to a coarse base domain using approaches such as those described in [8, 19]. However, specifying parameter values for an entire region is equivalent to flattening that region and thus invariably introduces distortion. Therefore we wish to keep the parameter regions as small as possible. Typically one computes parameter values for a certain local neighborhood like a 1-ring. We propose an even more local scheme in which parameter values are specified *separately* for each of the D_e^2 stencils. The two triangles of the D_e^2 stencil get flattened with the so-called *hinge map*: using the common edge as a hinge, rotate one triangle until it lies in the plane defined by the other triangle and compute the needed edge lengths and areas from (1). Note that the hinge map leaves the areas of the triangles $\varphi(f_1)$ and $\varphi(f_2)$ unchanged and only affects the faces $\{j, k, l_1\}$ and $\{j, k, l_2\}$. The surface relaxation operator is defined as before, but acts on points in $\mathbf{R}^3$

$$Rp_i = \textstyle\sum_{j \in \mathcal{V}_2(i)} w_{i,j} p_j.$$

Our minimization is similar to minimizing dihedral angles [21]. However, minimizing exact dihedral angles is difficult as the expressions depend non-linearly on the points. Instead one can think of the D_e^2 as a linear expression which behaves like the dihedral angle.

Features With our scheme it is particularly easy to deal with features in the mesh. Examples include sharp edges across which one does not wish to smooth. In that case the D_e^2 associated with those edges are simply removed from the functional.

One may worry what happens with the equations in (1) in case one of the triangles is degenerate, i.e., two of its points coincide and its area is zero. Then the D_e^2 that use this triangle are not defined and simply can be left out from the optimization. This is similar to coinciding knots in the case of splines.

Comparison with Existing Schemes The approach followed in [18] is to assume that the 1-ring neighborhood of a vertex i is parameterized over a regular K_i-gon. Using this approximation a discrete Laplacian, dubbed umbrella, is computed as

$$Lp_i = K_i^{-1} \textstyle\sum_{j \in \mathcal{V}_1(i)} p_j - p_i.$$

This discrete Laplacian was used in a relaxation operator $R = I + L$ which replaces a vertex with the average of its 1-ring neighbors.

In our setting, we can build a 1-ring relaxation scheme by only taking the minimum in (2) over $\mathcal{E}_1(i)$. The relaxation operator is then computed as in (3) with summations over $\mathcal{E}_1(i)$ rather than $\mathcal{E}_2(i)$. Our 1-ring scheme parameterized on a *regular* K_i-gon leads to the same relaxation operator as used by Kobbelt. Our scheme can thus be seen as a natural non-uniform generalization of the umbrella

which is still linear. In general we use the $\mathcal{E}_2(i)$ (1-ring with flaps) scheme as it yields visually smoother surfaces.

Taubin [26] presents a two step relaxation operator $R = (I + \mu L)(I + \lambda L)$, with μ and λ tuned to minimize shrinkage of the mesh.

Both of these schemes are semi-uniform filters since the weights only depend on K_i and not the geometry. Consequently they affect both geometry and parameterization. Consider again an irregular triangulation of a plane. Semi-uniform schemes try to make each 1-ring look as much as possible like a regular K-gon. Thus the triangulation may change globally while the plane remains the same. As we will see, this will lead to unwanted effects in applications such as editing and texture mapping. On the other hand our non-uniform scheme is linearly invariant, leaves the triangles unchanged, and does not suffer from the problems concerning movement "inside" the surface observed in [18, Figure 4].

Figure 4 shows the effect on a non-planar triangulation like the eye of the mannequin head. Our non-uniform scheme (right) smoothes the geometry without affecting the triangle shapes much. The semi-uniform scheme (middle) tries to make edge lengths as uniform as possible which can only be done by effectively destroying the delicate mesh structure around the eye. This effect also applies to any other attributes that vertices may carry such as detail vectors for editing or texture map coordinates causing distortion during smoothing (see Figure 8).

Taubin [26] also uses a non-uniform discrete Laplacian in which the weights vary as the powers of the respective edge lengths. While such an operator can greatly reduce the triangle distortions, it can be shown that such a scheme can never be linearly invariant.

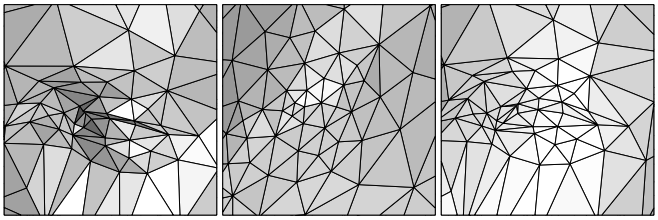


Figure 4: *Smoothing of the eye (left) with our non-uniform (right) and a semi-uniform scheme (middle). The semi-uniform scheme tries to make edge lengths as uniform as possible and severely distorts the geometry, while the non-uniform scheme only smoothes the geometry and does not affect the triangle shapes much.*

3 Multiresolution Signal Processing

Up to this point we have only considered operators which act on a scale comparable to their small finite support. To build more powerful signal processing tools we now consider a multiresolution setting.

Multiresolution algorithms such as subdivision, pyramids, and wavelets require decimation and upsampling procedures. For images decimation comes down to removing every other row or column. The situation for meshes is more complex, but a considerable body of work is available [13].

We employ Hoppe's Progressive Mesh (PM) approach [14]. In the PM setting, an edge collapse provides the atomic decimation step, while a vertex split becomes the atomic upsampling step. For simplicity we only employ half-edge collapses in our implementation. As a priority criterion we use a combination of the Garland-Heckbert quadric error metric [11] and edge length to favor removal of long edges (see also [17]).

Each half edge collapse removes one vertex and we number them in reverse so that the one with highest index gets removed first. This gives a sequence of N meshes $(\mathcal{P}^n, \mathcal{K}^n)$, $1 \le n \le N$, and $\mathcal{P}^n = \{p_i \mid 1 \le i \le n\}$. Later we will consider mesh sequences $(\mathcal{Q}^{(n)}, \mathcal{K}^{(n)})$ where the points on coarser meshes do move from their finest mesh position. These are denoted $q_i^{(n)}$, $i \le n$.

In traditional signal processing, downsampling creates a coarser level through the removal of a constant fraction of samples. This leads to a logarithmic number of levels. A PM does not have such a notion of levels. However, one may think of each removed vertex as living on its own level, and the number of levels being linear.

3.1 Subdivision

Subdivision starts from a coarse mesh and successively builds finer and smoother versions [22]. In signal processing terms it consists of upsampling followed by relaxation. So far the word subdivision has been associated in the literature with either regular or semi-regular meshes with corresponding uniform or semi-uniform operators. If one only has an original, coarse mesh and cares about building a smooth version, then semi-regular is the correct approach.

Our setting is different. The coarse mesh comes from a PM started at the original, finest level. Hence the connectivity of the finer levels is fixed and determined by the reverse PM. Our goal is to use non-uniform subdivision to build a *geometrically* smooth mesh with the *same* connectivity as the original mesh and with as little triangle shape distortion as possible. Such smoothed meshes can subsequently be used to build pyramid algorithms.

Subdivision is computed one level at a time starting from level n_0 in the progressive mesh $\mathcal{Q}^{(n_0)} = \mathcal{P}^{(n_0)}$. Since the reverse PM adds one vertex per level, our non-uniform subdivision is computed one vertex at a time. We denote the vertex positions as $\mathcal{Q}^{(n)} = \{q_i^{(n)} \mid 1 \le i \le n\}$ $(n \ge n_0)$ and use meshes $(\mathcal{Q}^{(n)}, \mathcal{K}^{(n)})$ with the same connectivity as the PM meshes.

Going from $\mathcal{Q}^{(n-1)}$ to $\mathcal{Q}^{(n)}$ involves three groups of vertices. (I) the new vertex n, which is introduced together with a point position $q_n^{(n)}$ to be computed. (II) certain points from the $\mathcal{Q}^{(n-1)}$ mesh change position; these correspond to *even* vertices. There is only a small number of them. (III) the remainder of the points of $\mathcal{Q}^{(n-1)}$, typically the majority, remains unchanged. Specifically:

- The new position $q_n^{(n)}$ is computed after upsampling from $\mathcal{K}^{n-1}$ to $\mathcal{K}^n$:
$$q_n^{(n)} = \textstyle\sum_{j \in \mathcal{V}_2^n(j)} w_{n,j}^{(n)} q_j^{(n-1)}.$$
The position of the new vertex is computed to satisfy the relaxation operator using points from $\mathcal{Q}^{n-1}$ with weights using areas and lengths of mesh $(\mathcal{P}^n, \mathcal{K}^{(n)})$.
- The even points of $\mathcal{Q}^{n-1}$ form a 1-ring neighborhood of n. Their respective $\mathcal{V}_2^n$ neighborhoods contain n, which has just received an updated position $q_n^{(n)}$
$$\forall j \in \mathcal{V}_1^n(n): \quad q_j^{(n)} = \textstyle\sum_{k \in \mathcal{V}_2^n(j) \setminus \{n\}} w_{j,k}^{(n)} q_k^{(n-1)} + w_{j,n}^{(n)} q_n^{(n)}.$$
The even vertices are relaxed using the point positions from $\mathcal{Q}^{(n-1)}$ (except for $q_n^{(n)}$), using weights coming from $(\mathcal{P}^n, \mathcal{K}^{(n)})$.
- Finally, the remainder of the positions do not change
$$\forall j \in \mathcal{V}^{n-1} \setminus \mathcal{V}_1^n(n): \quad q_j^{(n)} = q_j^{(n-1)}.$$

A central ingredient in our construction is the fact that the weights $w_{i,j}^{(n)}$ depend on parameter information from the mesh $\mathcal{P}^{(n)}$. No globally or even locally consistent parameterization is required. For each D_e^2 stencil we use the hinge map as described above. In effect the original mesh provides the parameterizations and in this way enters into the subdivision procedure. The actual *areas* and *lengths*, which make up the expressions for $w_{i,j}^{(n)}$ are assembled based on the connectivity $\mathcal{K}^{(n)}$ of level n, and hence induce the level dependence

of the weights. As a result all $w_{i,j}^{(n)}$ may be precomputed during the PM construction and can be stored if desired for later use during repeated subdivision. It is easy to see that the storage is linear in the total degree, $\sum_i K_i$, of the mesh.

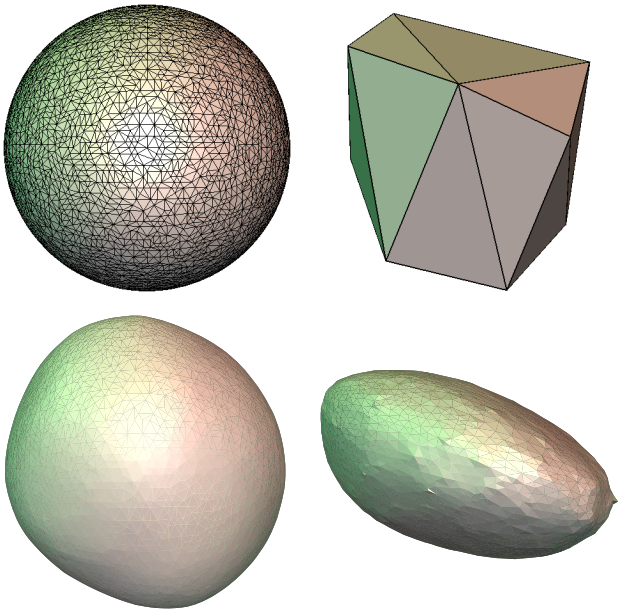


Figure 5: *Starting with the irregular triangulation of a sphere (upper left) we compute a PM down to 16 triangles (upper right). We then compute our non-uniform subdivision scheme back to the finest level (lower left) and obtain a smooth mesh which approximates the original. For comparison the lower right shows the limit surface of a semi-uniform subdivision scheme.*

To illustrate the behavior of uniform functional subdivision schemes one considers the so called *scaling function* or fundamental solution obtained from starting with a Kronecker sequence on the coarsest level. For surface subdivision, there is no equivalent to this. To illustrate the behavior of the surface scheme we perform the following experiment (see Figure 5). We start with an irregular triangulation of a sphere with 12000 triangles (upper left) and compute a PM down to 16 triangles (upper right). Next the non-uniform surface subdivision scheme starting from the 16 triangles back to the original mesh is computed (lower left). We clearly get a smooth mesh. For comparison the lower right shows the limit function using a semi-uniform scheme. It is important to understand that the non-uniform scheme has access to the parameterization information of the original finest mesh whereas the semi-uniform scheme does not use this additional information.

While for uniform and semi-uniform subdivision, extensive literature on regularity of limit functions exists, few results are known for non-uniform subdivision [2, 12]. The goal of our strategy of minimizing D_e^2 is to obtain C^1 smoothness. However, there is currently no regularity result for our scheme in either the functional or surface setting.

3.2 Burt-Adelson Pyramid

The pyramid proposed by Burt and Adelson [1] (BA) is another important signal processing tool. We show how to generalize it to a mesh pyramid. We start from the finest level points $\mathcal{S}^N = \mathcal{P}$ and compute a sequence of meshes $(\mathcal{S}^n, \mathcal{K}^n)$ $(1 \leq n \leq N)$ as well as oversampled differences $d_i^{(n)}$ between levels.

To go from $\mathcal{S}^n$ to $\mathcal{S}^{n-1}$, i.e., to remove vertex n, we follow the diagram in Figure 6. The top wire represents the points of $\mathcal{S}^{n-1}$ while the bottom wire represent the points of $\mathcal{S}^n$. There are four

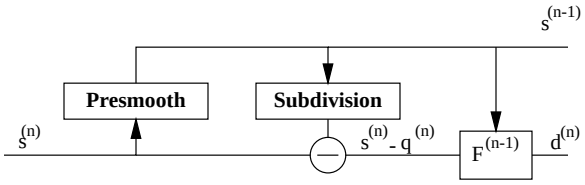


Figure 6: *Burt-Adelson style pyramid scheme.*

stages: presmoothing, downsampling, subdivision, and computation of details.

- **Presmoothing:** Presmoothing in the original BA pyramid is important to avoid aliasing. We have found that in a PM the presmoothing step can often be omitted because the downsampling steps (edge collapses) are chosen carefully, depending heavily on the data. In essence vertices are removed mostly in smooth regions, where presmoothing does not make a big difference. Thus, no presmoothing was used in our implementation.
- **Downsampling:** n is removed in a half-edge collapse.
- **Subdivision:** Using the points from $\mathcal{S}^{n-1}$ we compute subdivided points $q_j^{(n)}$ for the vertex just removed and the surrounding even vertices exactly as described in Section 3.1
- **Detail Computation:** Finally, detail values are computed for all even vertices as well as the vertex n. These detail vectors are expressed in a local frame $F_j^{(n-1)}$ which depends on the coarser level:

$$\forall j \in \mathcal{V}_1^n(n) \cup \{n\} : \quad d_j^{(n)} = F_j^{(n-1)}(s_j^{(n)} - q_j^{(n)}).$$

  We refer to the entire group of $d_j^{(n)}$ as an array $d^{(n)}$. In the implementation this array is stored with n.

One of the features of the BA pyramid is that the above procedure can always be inverted independent of which presmoothing operator or subdivision scheme is used. For reconstruction, we start with the points of $\mathcal{S}^{n-1}$, subdivide values $q_j^{(n)}$ for both the new and even vertices and add in the details to recover the original values $s_j^{(n)}$.

To see the potential of a mesh pyramid in applications it is important to understand that the details $d^{(n)}$ can be seen as an approximate frequency spectrum of the mesh. The details $d^{(n)}$ with large n come from edge collapses on the finer levels and thus correspond to small scales and high frequencies, while the details $d^{(n)}$ with small n come from edge collapses on the coarser levels and thus correspond to large scales and low frequencies.

Oversampling factor A standard image pyramid has an oversampling factor of 4/3, while we have an expected oversampling factor of 7. The advantage of oversampling is that the details are quite small and lead to natural editing behavior [29]. If needed, a technique exists to reduce the oversampling factor. The idea is to use levels with more than one vertex. Say, we divide the N vertices of $\mathcal{V}$ into M levels with $M \ll N$:

$$\mathcal{V} = \mathcal{V}_0 \cup \bigcup_{1 \leq m \leq M} \mathcal{W}_m \quad \text{and} \quad \mathcal{V}_m = \mathcal{V}_{m-1} \cup \mathcal{W}_m.$$

This can be done, for example, so that the sizes of the $\mathcal{V}_m$ grow with a constant factor [7]. The BA pyramid then goes from $\mathcal{V}_m$ to $\mathcal{V}_{m-1}$. First presmooth all even vertices in $\mathcal{V}_m$, then compute subdivided values for all vertices in $\mathcal{W}_m$ and their 1-ring neighbors in $\mathcal{V}_m$. For the subdivided points, which need not be all vertices of $\mathcal{V}_m$, compute the details as differences with the original values from $\mathcal{V}_m$. One can see that the above algorithm with oversampling factor 7 is a special case when $\mathcal{W}_m = \{m\}$. The other extreme is the case with only one level containing all vertices. In that case

there is no multiresolution as all details live on the same level. The oversampling factor is 1. By choosing the levels appropriately one can obtain any oversampling between 1 and 7. It is theoretically possible to build a wavelet-like, i.e, critically sampled multiresolution transform based on the Lifting scheme [25]. However, at this point it is not clear how to design filters that make the transform stable.

Caveat Often in this paper we use signal processing terminology such as frequency, low pass filter, aliasing, to describe operations on 2-manifolds. One has to be extremely careful with this and keep in mind that unlike in the Euclidean setting, there is no formal definition of these terms in the manifold setting. For example in a mesh the notion of a DC component strictly does not exist. Also in connection with the pyramid we often talk about frequency bands. Again one has to be careful as even in the Euclidean setting the coefficients in a a pyramid do not represent exact frequencies due to the Heisenberg uncertainty principle.

4 Applications

The algorithms we described above provide a powerful signal processing toolbox. In this section we demonstrate this claim by considering a variety of applications that use them. These include smoothing and filtering, enhancement, texture coordinate generation, vector displacement field editing, and multiresolution editing.

4.1 Smoothing and Filtering

One way to smooth a mesh is through repeated application of the relaxation operator R. Numerically this behaves similarly to traditional Jacobi iterations for an elliptic PDE solver. The relaxation rapidly attenuates the highest frequencies in the mesh, but has little impact on low frequencies. Even though each iteration of the operator is linear in the number of vertices, the number of iterations to attenuate a fixed frequency band grows linearly with the mesh size. This results in quadratically increasing run times as the sample density increases relative to a fixed geometric scale. One way to combat this behavior is through the use of appropriate preconditioners, as was done in [18], or through the use of implicit solvers [6].

Using a mesh pyramid we can build much more direct and flexible filtering operations. Recall that the details in a pyramid measure the local deviation from smoothness at different scales. In that sense they capture the local frequency content of the mesh. This spectrum can be shaped arbitrarily by scaling particular details. Multiresolution filtering operators are built by setting certain ranges of detail coefficients in the pyramid to zero. A low pass filter sets all detail arrays $d^{(n)}$ with $n > n_l$ to zero, while a high pass filter annihilates $d^{(n)}$ for $n < n_h$. However, for meshes it makes little sense to put the coarsest details to zero as this would collapse the mesh. More natural for meshes are stopband filters which zero out detail arrays $d^{(n)}$ in some intermediate range, $n_l < n < n_h$.

Figure 7 shows these procedures applied to the venus head ($N = 50000$). On the upper left the original mesh. The upper right shows the result of applying the non-uniform relaxation operator 20 times at the finest level. High frequency ripples quickly diffuse, but no attenuation is noticeable at larger length scales. The bottom left shows the result of a low pass filter which sets all details above $n_l = 1000$ to zero. Finally the bottom right shows the result of a stopband filter, annihilating all details $1000 < n < 15000$. Note how the last mesh keeps its fine level details, while intermediate frequencies were attenuated. If desired all these filtering operations can be performed in a spatially varying manner due to the space-frequency localization of the mesh pyramid. Figure 8 shows the difference between non-uniform (left) and semi-uniform smoothing (right) on the actual vertex positions. By keeping the original finest level texture coordinates for the vertices of both meshes we can

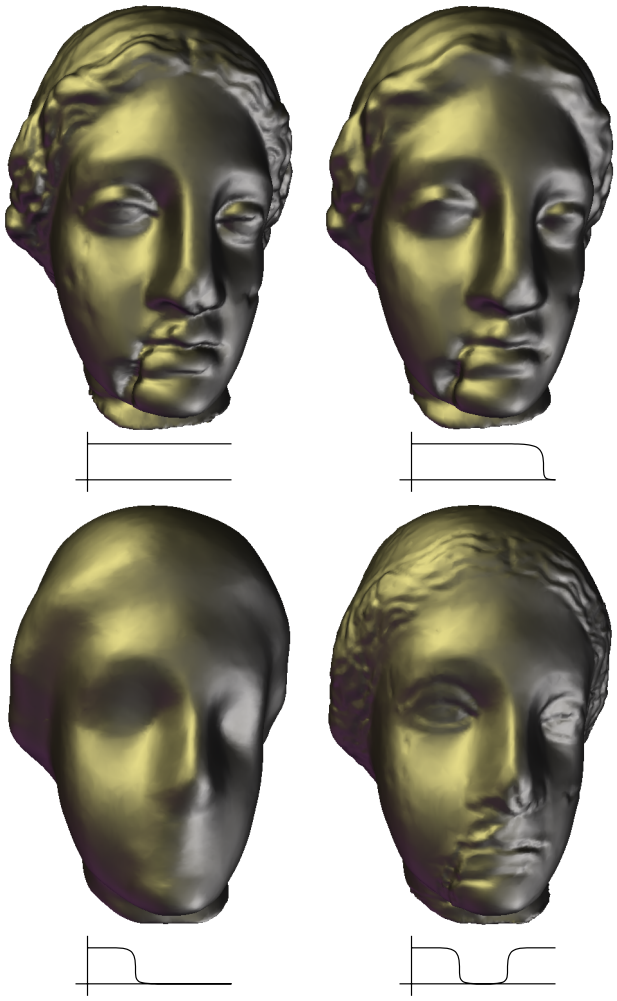


Figure 7: *Smoothing and filtering of the venus head. Original on the top left; 20 finest level relaxation steps on the top right; low pass filter on the bottom left; stopband filter on the bottom right.*

visualize the effect of movement "within" the surface after smoothing. This hints at another application: if one has a scanned mesh with color (r,g,b) attributes per vertex then non-uniform *geometry* smoothing will not distort those colors.

4.2 Enhancement

Enhancement provides the opposite operation to smoothing in that it emphasizes certain frequency ranges. As before this can be done in a single resolution manner as well as in the more flexible multiresolution setup.

The single resolution scheme is easy to compute and typically works best for fairly small meshes, such as those used as control polyhedra for splines or semi-regular subdivision surfaces. The main idea is to extrapolate the difference between the original mesh and a single resolution relaxed mesh. The enhanced points are given by

$$Ep_i = p_i + \xi(R^k p_i - p_i),$$

where $\xi > 1$. Figure 9 illustrates the procedure. On the left the original mannequin head, in the middle the result after 20 relaxation steps, and on the right the enhanced version with $\xi = 2$. The first and last models of Figure 1 show the Loop subdivided meshes of the original and enhanced head. By using combinations of the different algorithms peculiar effects can be obtained. The second

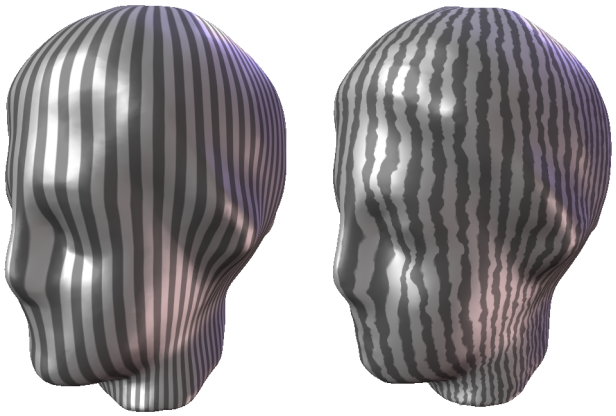


Figure 8: *Movement "within" the surface due to smoothing visualized by letting the vertices keep their original finest level texture coordinates. Left non-uniform smoothing and right semi-uniform smoothing.*

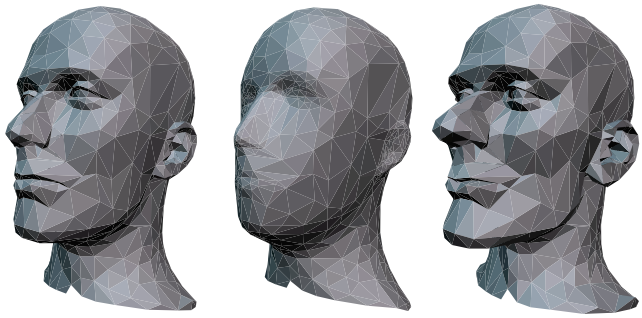


Figure 9: *Enhancement of control mesh. On the left the original, in the middle the smoothed mesh, and on the right the enhanced mesh (see also Figure 1 for the resulting subdivision surfaces).*

model in Figure 1 is obtained by extrapolating from a base model built by 5 semi-uniform relaxation steps followed by 5 non-uniform relaxation steps (needed to recover the parameterization and "pull" features back in place). The third model in Figure 1 is extrapolated from a base built by first simplifying to level 100, then applying 1 relaxation step (which made the chin collapse and ears shrink), and reconstructing.

The single level scheme is simple and easy to compute, but limited in its use. For example, it does not compute offsets with respect to local frames. If the mesh contains fine level detail self intersections quickly appear. As in image enhancement one must be careful not to amplify high frequency noise. For these reasons we need the more flexible setup of multiresolution enhancement. The approach is simple, we compute a mesh pyramid, scale the desired details and then reconstruct. As in the filtering application, the user has control over the different frequency bands. Additionally, the local frames across the many levels of the mesh pyramid tend to stabilize the procedure and lead to a more natural behavior. As a result the multiresolution enhancement scheme deals better with large scanned meshes which usually contain high frequency noise.

Figure 10 shows Loop subdivided versions of the original cow head and an enhanced version obtained by multiplying the details $d^{(n)}$ with $257 < n \leq 2904 = N$ by two (see also Figure 15, right column for an edit of the enhanced model). Finally, Figure 11 shows enhancement on the Stanford bunny ($N = 34835$). Here details with indices $1000 < n < 7000$ were multiplied by 2, and details with indices $7000 < n < 13000$ were multiplied by 1.5.

Figure 10: *Enhancement of cow head (original on the left).*

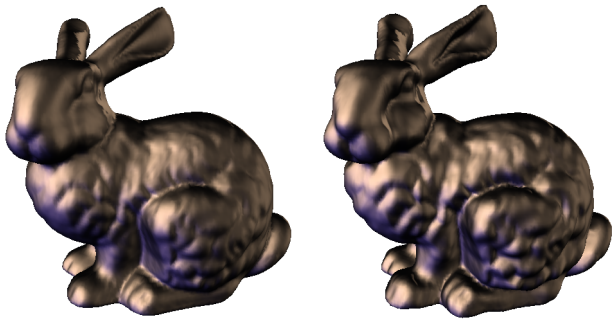


Figure 11: *Enhancement on the bunny. The original is on the left and the frequency enhanced version on the right.*

4.3 Subdivision of Scalar Functions on Manifolds

We can use subdivision to quickly build smooth scalar functions defined *on* a manifold. Simply start with scalar values on a coarse level and use non-uniform subdivision to build a smooth function defined on the finest level.

We present two applications. The first creates smoothly varying texture coordinate assignments for the finest level mesh from some user supplied texture coordinate assignments at a coarse level. The second creates a smoothly varying function over a limited region of an irregular mesh and then uses this function to generate a smooth vector displacement field for shape editing purposes.

Texture Coordinate Generation DeRose et al. [5] discuss this problem in the context of classical, semi-uniform subdivision. Their goal was the construction of smooth texture coordinates for Catmull-Clark surfaces. Beginning with user supplied texture coordinates at some coarse level they subdivide these parameter assignments to the finest subdivision level using the same subdivision operator for texture coordinates as for the vertices.

Figure 12 shows the application of this idea to our setting. Initial texture coordinate assignments were made using a cylindrical projection of all vertices in $\mathcal{P}^{1000}$. The left image shows a test texture on the coarse polygonal mesh. We then reconstruct the original finest level mesh and concurrently subdivide the texture coordinates to the finest level. The resulting mapping is shown on the right. Even though the geometry has much geometric detail and uneven triangle sizes the final texture coordinates vary smoothly over the entire surface.

Displacement Vector Field Editing Singh and Fiume [23] present an algorithm for deformation edits based on vector displacement fields. These fields are defined through a smooth falloff function around a "wire" which drags the surface along. The region of influence is a function of distance in $\mathbf{R}^3$. Controlling this behavior in regions of high curvature or in the vicinity of multiple close objects can be tricky. In our setting we have the opportunity to define the falloff function *only* on the surface itself. A similar idea was used in [15] for feature editing.

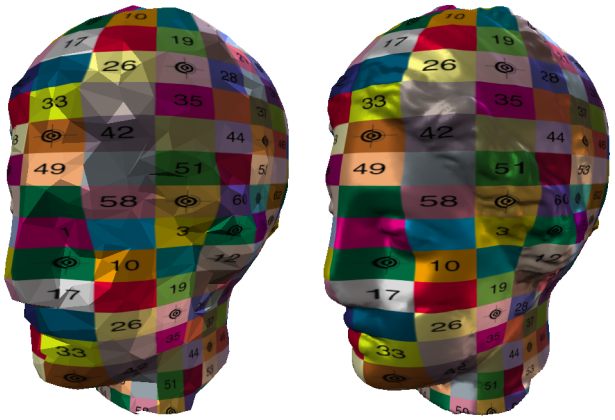


Figure 12: *A test texture is mapped to a coarse level of the mesh pyramid under user control. The resulting texture coordinates are then subdivided to the finest level and the result shown on the right.*

We illustrate this idea with an example. Consider the horse to "giraffe" edit in Figure 13. The user first outlines three regions by drawing closed curves on the mesh. A region that remains unchanged (A); a region that will be gradually stretched (B); and a region that will undergo a translation (C). In our example, region (A) is the back body and the four legs; (B) are the neck and torso; and (C) is the head. The boundary between (A) and (B) consists of three closed curves. Next we define a scalar parameter θ, which is 0 on the boundary between (A) and (B), and 1 on the boundary between (B) and (C). The algorithm computes values for θ that vary smoothly between 0 and 1 in region (B).

This is accomplished by running a PM on the interior of region (B) to a maximally coarse level. Then the initial value $\theta = 1/2$ is assigned to all interior vertices of the coarse region (B). Next we apply relaxation to θ on the coarsest level within (B). This converges quickly because there are few triangles; three steps suffice. These θ values are then used as the starting values for subdivision from the coarsest level back to the original region (B) while keeping the θ values on the boundary fixed. The resulting θ values on the finest region (B) vary smoothly between 0 and 1. The only problem is that at the boundary they meet in a C^0 and not a C^1 fashion. This is because we only imposed Dirichlet like conditions and no Neumann condition. We address this with the following smoothing transformation, $\theta := 1/2 - 1/2\cos(\pi\theta)$.

On the left of Figure 13 the red lines are specified by the user and the black lines show the θ isolines, visualizing how θ varies smoothly. The edit is now done by letting the user drag the head. Every vertex in region B is subjected to θ times the displacement vector of the head. This requires very little computation. The right side of Figure 13 shows the result.

4.4 Multiresolution Editing

The displacement vector editing is simple and fast, but has limited use. We next discuss full fledged multiresolution editing for irregular meshes. Our algorithm combines ideas of Zorin et al. [29] and Kobbelt et al. [18]. The former used multiresolution details and semi-regular meshes, while the latter used single resolution details and irregular meshes. We combine the best of both approaches by using multiresolution details with the irregular mesh setting.

The algorithm is straightforward. The user can manipulate a group of points $s_i^{(n)}$ in the mesh pyramid and the system adds the finer level details back in. This is exactly the same use of the pyramid as Zorin et al. only now for irregular meshes. Kobbelt et al. used a *multiresolution/multigrid* approach to define a smoothed mesh over a user selected region, but then compute *single resolution* details between the original and smoothed mesh.

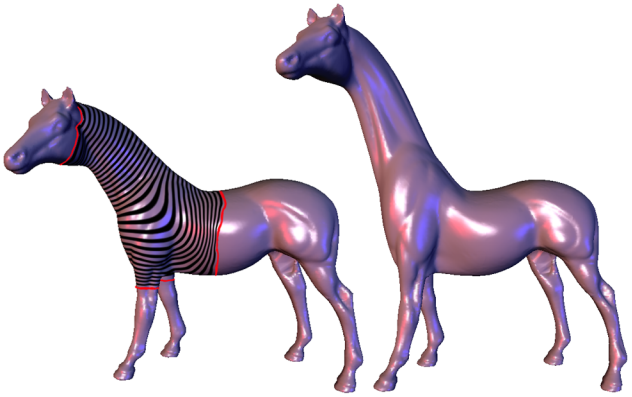


Figure 13: *Horse to giraffe edit using a surface based smooth displacement vector field.*

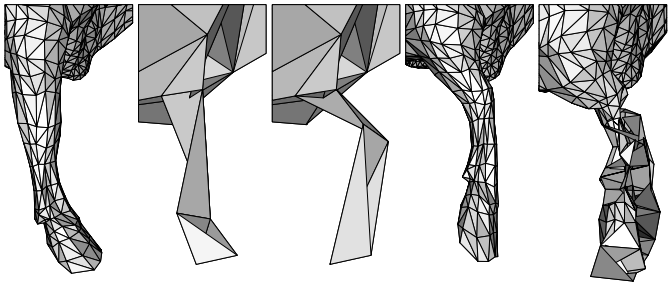


Figure 14: *Cow leg editing sequence: original, coarsest scale, edit, reconstruction with multiresolution details, reconstruction with single resolution details.*

The use of multiresolution details is important when the user wishes to make large scale edits in regions with complicated fine scale geometry. Because the multiresolution details are all described in local frames, they have more flexibility to adjust themselves to a coarse scale edit.

We illustrate this with an edit on the leg of the cow (Figure 14). The sequence shows the original leg, the coarse leg, a coarse edit, and two reconstructions. The first used multiresolution details while the second used single resolution details.

Finally, Figure 15 shows some additional edits. The horse was edited at a level containing only 34 vertices (compare to the original shape shown in Figure 13). The cow edit on the right column involves both manipulation at coarse levels (snout, horns, leg, tail) and overall enhancement.

Dataset	Venus	Horse	Bunny	Cow	Mann.
Size (fine)	50000	48485	34835	2904	689
Size (coarse)	4	34	19	57	5
Timings (s)					
Simpl. & Anal.	79	75	55	3.6	0.8
Reconstruction	9	8	5.8	.37	0.1
Analysis	9	8	5.8	.37	0.1

Table 1: *Timings for mesh pyramid computation assuming storage rather then recomputation of all areas and length needed in stencil weight computations. The size field counts the total vertices (N). Face counts are generally twice as large. All times are given in seconds on an SGI R10k O2 @175Mhz.*

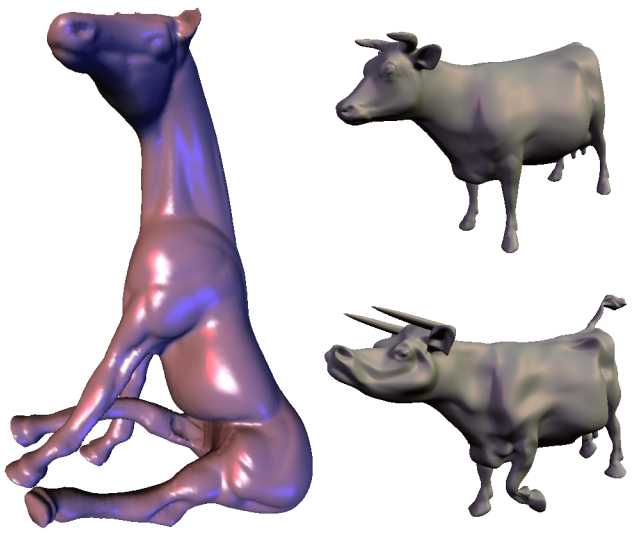


Figure 15: *Multiresolution edits.*

5 Conclusions and Future Work

We have shown how basic signal processing tools such as up and down sampling and filtering can be extended to irregular meshes. These tools can be built into powerful algorithms such as subdivision and mesh pyramids. We have demonstrated their use in texturing, editing, smoothing and enhancement.

Further research can be pursued in several directions. On the algorithms side there is incorporation of various boundary conditions, construction of positive weight schemes, and extensions to tetrahedralizations. On the applications side there is adaptive gridding for time dependent PDE's, computing globally smooth parameterizations, extracting texture maps from scanned textures, and space-frequency morphing.

Compression Another potential future application is compression. However, one needs to be extremely careful: our subdivision weights depend on the parameterization which in turn depends on the geometry of the original mesh. Thus one cannot use the subdivision scheme as a predictor in a compression framework unless sender and receiver share parameter information, i.e., the needed areas and lengths to compute the subdivision. Only a setting where one repeatedly has to communicate functions or attributes defined over a fixed triangulation would justify this overhead.

This touches upon a deeper issue. In some sense for a geometrically smooth irregular mesh only one dimension can effectively be predicted by a subdivision scheme. Even for a geometrically smooth mesh, no subdivision scheme can compress the information implicitly present in the parameterization. Ideally for smooth surfaces one would like to use meshes with as little parametric information as possible.

A typical example are semi-uniform meshes. This argument strongly makes the case for resampling onto semi-regular meshes using smooth parameterizations [8, 19] before compression.

Acknowledgments Igor Guskov was partially supported by a Harold W. Dodds Fellowship and a Summer Internship at Bell Laboratories, Lucent Technologies. Other support was provided by NSF (ACI-9624957, ACI-9721349, DMS-9874082), Alias|wavefront and through a Packard Fellowship. Special thanks to Ingrid Daubechies, Aaron Lee, Adam Finkelstein, Zoë Wood, and Khrysaundt Koenig. Our implementation uses the triangle facet data structure and code of Ernst Mücke, and the priority queue implementation by Michael Garland.

References

[1] BURT, P. J., AND ADELSON, E. H. Laplacian Pyramid as a Compact Image Code. *IEEE Trans. Commun. 31*, 4 (1983), 532–540.

[2] DAUBECHIES, I., GUSKOV, I., AND SWELDENS, W. Regularity of irregular subdivision. *Const. Approx.* (1999), to appear.

[3] DE BOOR, C. A multivariate divided differences. *Approximation Theory VIII 1* (1995), 87–96.

[4] DE BOOR, C., AND RON, A. On multivariate polynomial interpolation. *Constr. Approx. 6* (1990), 287–302.

[5] DEROSE, T., KASS, M., AND TRUONG, T. Subdivision Surfaces in Character Animation. *Computer Graphics (SIGGRAPH '98 Proceedings)* (1998), 85–94.

[6] DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, Aug. 1999.

[7] DOBKIN, D., AND KIRKPATRICK, D. A Linear Algorithm for Determining the Separation of Convex Polyhedra. *Journal of Algorithms 6* (1985), 381–392.

[8] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, 173–182, 1995.

[9] FLOATER, M. S. Parameterization and Smooth Approximation of Surface Triangulations. *Computer Aided Geometric Design 14* (1997), 231–250.

[10] FORNBERG, B. Generation of finite difference formulas on arbitrarily spaced grids. *Math. Comput. 51* (1988), 699–706.

[11] GARLAND, M., AND HECKBERT, P. S. Surface Simplification Using Quadric Error Metrics. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, 209–216, 1996.

[12] GUSKOV, I. Multivariate Subdivision Schemes and Divided Differences. Tech. rep., Department of Mathematics, Princeton University, 1998.

[13] HECKBERT, P. S., AND GARLAND, M. Survey of Polygonal Surface Simplification Algorithms. Tech. rep., Carnegie Mellon University, 1997.

[14] HOPPE, H. Progressive Meshes. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, 99–108, 1996.

[15] KHODAKOVSKY, A., AND SCHRÖDER, P. Fine Level Feature Editing for Subdivision Surfaces. In *ACM Solid Modeling Symposium*, 1999.

[16] KOBBELT, L. Discrete Fairing. In *Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces*, 101–131, 1997.

[17] KOBBELT, L., CAMPAGNA, S., AND SEIDEL, H.-P. A General Framework for Mesh Decimation. In *Proceedings of Graphics Interface*, 1998.

[18] KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. Interactive Multi-Resolution Modeling on Arbitrary Meshes. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, 105–114, 1998.

[19] LEE, A., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. MAPS: Multiresolution Adaptive Parametrization of Surfaces. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, 95–104, 1998.

[20] LÉVY, B., AND MALLET, J. Non-Distorted Texture Mapping for Sheared Triangulated Meshes. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, 343–352, July 1998.

[21] MORETON, H. P., AND SÉQUIN, C. H. Functional optimization for fair surface design. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, vol. 26, 167–176, July 1992.

[22] SCHRÖDER, P., AND ZORIN, D., Eds. *Course Notes: Subdivision for Modeling and Animation.* ACM SIGGRAPH, 1998.

[23] SINGH, K., AND FIUME, E. Wires: A Geometric Deformation Technique. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, 405–414, 1998.

[24] SPANIER, E. H. *Algebraic Topology*. McGraw-Hill, New York, 1966.

[25] SWELDENS, W. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal. 29*, 2 (1997), 511–546.

[26] TAUBIN, G. A Signal Processing Approach to Fair Surface Design. In *Computer Graphics (SIGGRAPH' 95 Proceedings)*, 351–358, 1995.

[27] TAUBIN, G., ZHANG, T., AND GOLUB, G. Optimal Surface Smoothing as Filter Design. Tech. Rep. 90237, IBM T.J. Watson Research, March 1996.

[28] WELCH, W., AND WITKIN, A. Free–Form Shape Design Using Triangulated Surfaces. In *Computer Graphics (SIGGRAPH '94 Proceedings)*, 247–256, July 1994.

[29] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interactive Multiresolution Mesh Editing. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, 259–268, 1997.

Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow

by M. Desbrun, M. Meyer, P. Schroder, and A.H. Barr, Siggraph'99.

Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow

Mathieu Desbrun Mark Meyer Peter Schröder Alan H. Barr

Caltech*

Abstract

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

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

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

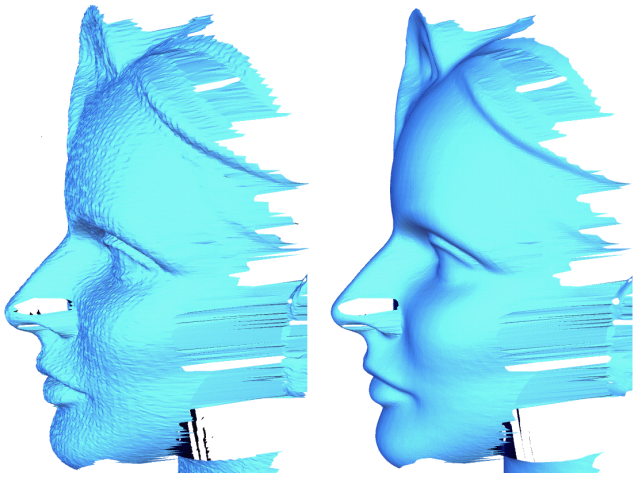
1 Introduction

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

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

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

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



(a) (b)

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

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

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

2 Implicit fairing

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

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

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

2.1 Notation and definitions

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

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

$$E(S) = \int_S \kappa_1^2 + \kappa_2^2 \, dS. \tag{1}$$

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

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

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

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

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

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

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

2.2 Diffusion equation for mesh fairing

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

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

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

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

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

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

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

2.3 Time-shifted evaluation

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

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

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

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

2.4 Solving the sparse linear system

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

2.5 Interpretation of the implicit integration

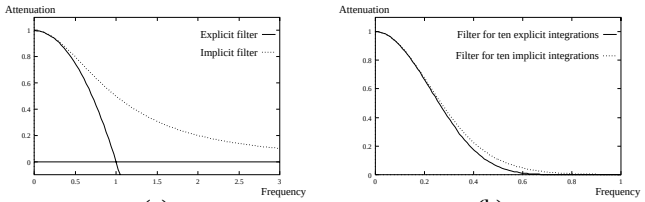
Although this implicit integration for diffusion is sound as is, there are useful connections with other prior work. We review the analogies with signal processing approaches and physical simulation.

2.5.1 Signal processing

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

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

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



(a) (b)

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

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

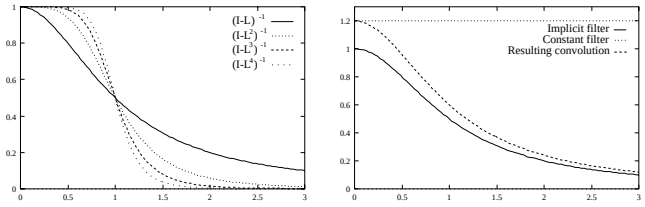
2.5.2 Mass-spring network

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

2.6 Filter improvement

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

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



(a) (b)

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

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

2.7 Constraints

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

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

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

2.8 Discussion

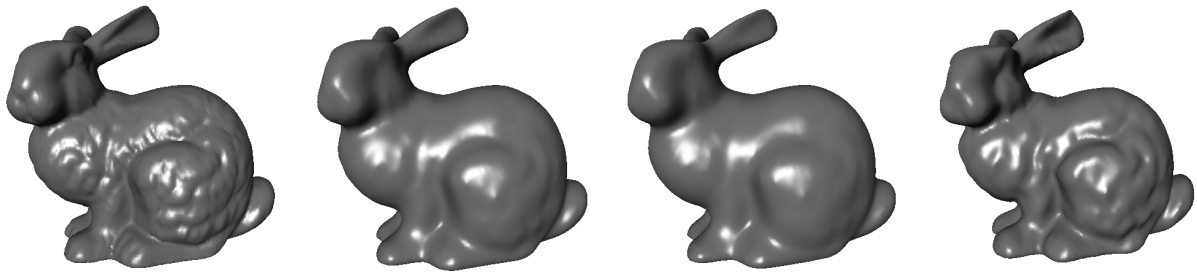
Even if adding a linear solver step to the integration of the diffusion equation seems to slow down the problem at first glance, it turns out that we gain significantly by doing so. For instance, the implicit integration can be performed with an arbitrary time step. Since the matrix of the system is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

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

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

3 Automatic anti-shrinking fairing

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



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

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

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

3.1 Volume computation

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

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

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

3.2 Exact volume preservation

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

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

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

3.3 Discussion

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

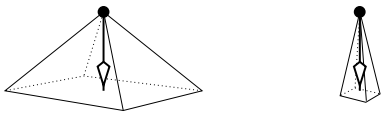
4 An accurate diffusion process

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

4.1 Inadequacy of the umbrella operator

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

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



(a) (b)

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

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

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

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

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

4.2 Simulation of the 1D heat equation

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

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

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

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

4.3 Extension to 3D

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

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

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

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

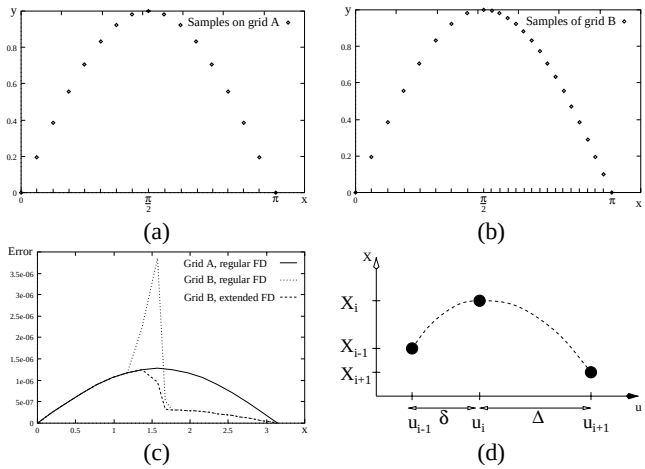


Figure 6: *Test on the heat equation: (a) regular sampling vs. (b) irregular sampling. Numerical errors in one step of integration (c): using the usual FD weight on an irregular grid to approximate second derivatives creates noise, and gives a worse solution than on the coarse grid, whereas extended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

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

5 Curvature flow for noise removal

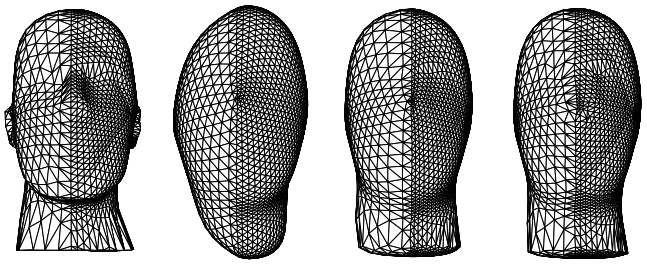
In terms of differential equations, diffusion is a close relative of curvature flow. In this section, we first explore the advantages of using curvature flow over diffusion, and then propose an efficient algorithm for noise removal using curvature flow.

5.1 Diffusion vs. curvature flow

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

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

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



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

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

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

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

5.2 Curvature normal calculation

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

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

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

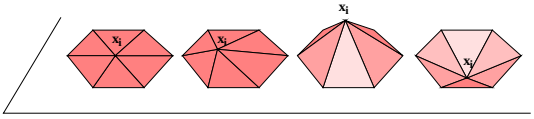
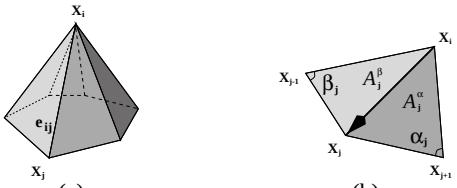


Figure 8: *The area around a vertex x_i lying in the same plane as its 1-ring neighbors does not change if the vertex moves within the plane, and can only increase otherwise. Being a local minimum, it thus proves that the derivative of the area with respect to the position of x_i is zero for flat regions.*

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

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

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



(a) (b)

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

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

5.3 Boundaries

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

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

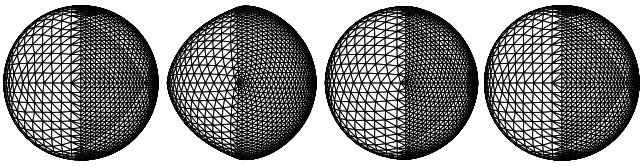
5.4 Implementation

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

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

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

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



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

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

5.5 Normalized version of the curvature operator

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

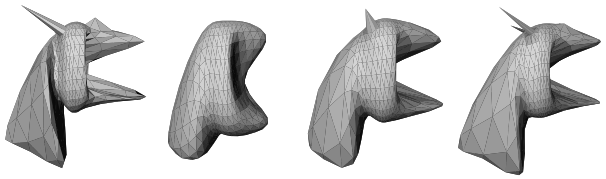
$$(\overline{\kappa}\,\mathbf{n})_{normalized} = \frac{1}{\sum_j (cot\ \alpha_j^l + cot\ \alpha_j^r)} \sum_j (cot\ \alpha_j^l + cot\ \alpha_j^r)(X_i - X_j)$$

5.6 Comparison of results

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

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

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



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

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

6 Discussion and conclusion

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

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

Acknowledgements

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

References

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

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

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

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

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

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

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

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

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

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

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

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

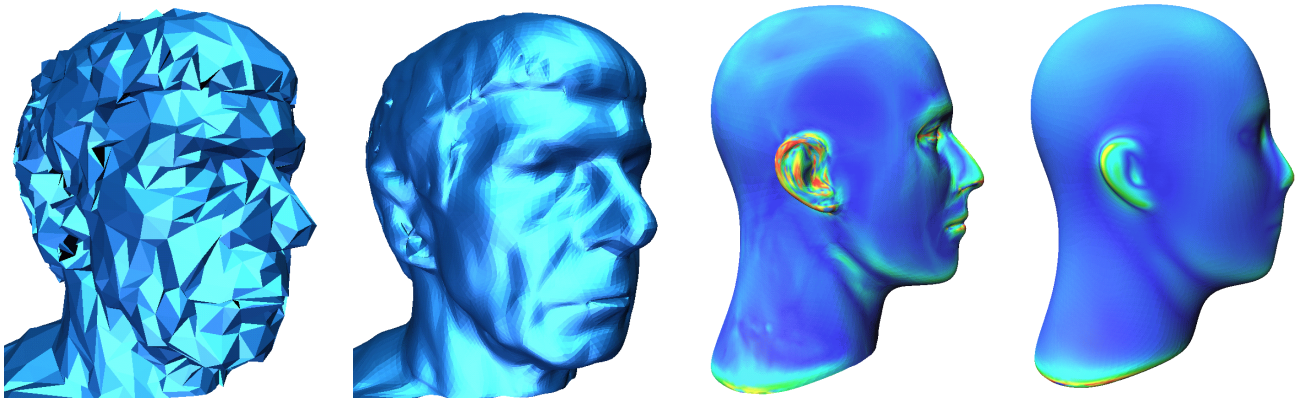
[PP93] Ulrich Pinkall and Konrad Polthier. Computing Discrete Minimal Surfaces and Their Conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

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

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

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

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



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

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

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

Appendix

A Preconditioned Bi-Conjugate Gradient

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

A.1 Preconditioning

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

A.2 Convergence criterion

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

A.3 Memory requirements

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

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

B Curvature normal approximation

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

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

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

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

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

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

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

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

Consequently:

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

Using Equ. (13), we find:

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

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

3D Mesh Geometry Filtering Algorithms for Progressive Transmission Schemes

by R. Balan and G. Taubin,

CAD Special Issue on Multiresolution
Geometric Models, 1999.

RC 21707 (Log 97779) 03/28/2000
Subject Area : Computer Science / Mathematics

IBM Research Report

3D Mesh Geometry Filtering Algorithms For Progressive Transmission Schemes

Radu Balan[1]
Simens Corporate Research
755 College Road East, Princeton, NJ 08540
rvbalan@scr.siemens.com

Gabriel Taubin
IBM T.J.Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598
taubin@us.ibm.com

To appear in CAD, Special Issue on Multi-resolution Geometric Models, 2000.

[1] This paper was written while the author was a postdoctoral associate at IBM and the IMA, University of Minnesota.

3D Mesh Geometry Filtering Algorithms for Progressive Transmission Schemes

RaduBalan * Gabriel Taubin †

Keywords: Compression , Geometry, Multi Resolution , Filter

ABSTRACT

A number of static and multi-resolution methods have been introduced in recent years to compress 3D meshes. In most of these methods the connectivity information is encoded without loss of information, but user-controllable loss of information is tolerated while compressing the geometry and property data. All these methods are very efficient at compressing the connectivity information, in some cases to a fraction of a bit per vertex, but the geometry and property data typically occupies much more room in the compressed bitstream than the compressed connectivity data. In this paper we investigate the use of polynomial linear filtering as studied in [Taubin95, TaZhGo96], as a global predictor for the geometry data of a 3D mesh in multi-resolution 3D geometry compression schemes. Rather than introducing a new method to encode the multi-resolution connectivity information, we choose one of the efficient existing schemes depending on the structure of the multi-resolution data. After encoding the geometry of the lowest level of detail with an existing scheme, the geometry of each subsequent level of detail is predicted by applying a polynomial filter to the geometry of its predecesor lifted to the connectivity of the current level. The polynomial filter is designed to minimize the l^2-norm of the approximation error but other norms can be used as well. Three properties of the filtered mesh are studied next: accuracy, robustness and compression ratio. The Zeroth Order Filter (unit polynomial) is found to have the best compression ratio. But higher order filters achieve better accuracy and robustness properties at the price of a slight decrease of the compression ratio.

1 Introduction

Polygonal models are the primary 3D representations for the manufacturing, architectural, and entertainment industries. They are also central to multimedia standards such as VRML and MPEG-4. In these standards, a polygonal model is defined by the position of its vertices (geometry); by the association between each face and its sustaining vertices (connectivity); and optional colors, normals and texture coordinates (properties).

Several single-resolution [TaRo98, LiKuo98] and multi-resolution methods [Hoppe96, PoHo97, TaGuHoLa98, Ross99] have been introduced in recent years to represent 3D meshes in compressed form for compact storage and transmission over networks and other communication channels. In most of these methods the connectivity information is encoded without loss of information, and user-controllable loss is tolerated while compressing the geometry and property data. In fact, some of these methods only addressed the encoding of the connectivity data [GuSt98]. Multi-resolution schemes reduce the burden of generating hierarchies of levels on the fly, which may be computationally expensive, and time consuming. In some of the multi-resolution schemes the levels of detail are organized in the compressed data in progressive fashion, from low to high resolution. This is a desirable property for applications which require transmission of large 3D data sets over low bandwith communication channels. Progressive schemes are more complex and typically not as efficient as single-resolution methods, but reduce quite significantly the latency in the decoder process.

In this paper we investigate the use of polynomial linear filtering [Taubin95, TaZhGo96], as a global predictor for the geometry data of a 3D mesh in multi-resolution 3D geometry compression schemes. As in other multi-resolution geometry compression schemes, the geometry of a certain level of detail is predicted as a function of the geometry of the next coarser level of detail. However, other 3D geometry compression schemes use simpler and more localized prediction schemes.

* Siemens Corporate Research, 755 College Road East, Princeton, NJ 08540, USA, rvbalan@scr.siemens.com. This paper was written while the first author was a postdoctoral associate at IBM (T.J.Watson Research Center) and IMA (University of Minnesota)

† IBM T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, taubin@us.ibm.com

Although we concentrate on the compression of geometry data, property data may be treated similarly. The methods introduced in this paper apply to the large family of multi-resolution connectivity encoding schemes. These linear filters are defined by the connectivity of each level of detail and a few parameters, which in this paper are obtained by minimizing a global criterion related to certain desirable properties. Our simulations present the least square filters compared with some other standard filters.

In [PaRo99] the Butterfly subdivision scheme is used for the predictor. In [GuSwSc99], a special second order local criterion is minimized to refine the coarser resolution mesh. The latter class of algorithms have concentrated on mesh simplification procedures and efficient connectivity encoding schemes. For instance in the Progressive Forest Split scheme [TaGuHoLa98], the authors have used a technique where the sequence of splits is determined based on the local volume conservation criterion. Next, the connectivity can be efficiently compressed as presented in the aforementioned paper or as in [Ross99].

Mesh simplification has been also studied in a different context. Several works address the remeshing problem, usually for editing purposes. For instance in [Eck&all95] the harmonic mapping is used to resample the mesh. Thus the remashing is obtained by minimizing a global curvature-based energy criterion. A conformal map is used in [Lee&all98] for similar purposes, whereas in [MaYaVe93] again a global length based energy criterion is used to remesh.

The organization of the paper is the following: in section 2 we review the mesh topology based filtering and introduce the basic notions; in section 3 we present two geometry encoding algorithms; in section 4 we analyze three desirable properties, accuracy, robustness and compression ratio; in section 5 we present numerical and graphical results; finally, the conclusions are contained in section 6 and are followed by the bibliography.

2 Mesh Topology Based Filtering

Consider a mesh $(\mathcal{V}, \mathcal{F})$ given by a list of vertex coordinates $\mathcal{V}$ (the *mesh geometry*) of the nV vertices, and a list of polygonal faces $\mathcal{F}$ (the *mesh connectivity*). The mesh geometry can be thought of as a collection of three vectors (x, y, z) of length nV containing, respectively, the three coordinates of each vertex; alternatively we can see $\mathcal{V}$ as representing a collection of nV vectors $(r_0, r_1, \ldots, r_{nV})$ of length 3, each of them being the position vector of some mesh vertex. To the list $\mathcal{F}$ we associate the symmetric $nV \times nV$ vertex to vertex incidence matrix M, and the $nV \times nV$ matrix K defined by:

$$K = I - DM \tag{1}$$

where D is the $nV \times nV$ diagonal matrix whose (i, i) element is the inverse of the number of first order neighbors the vertex i has. As shown in [Taubin95], K has nV real eigenvalues all in the interval $[0, 2]$.

Consider now a collection $P = (P_x(X), P_y(X), P_z(X))$ of three polynomials each of degree d, for some positive integer d.

Definition We call P a *polynomial filter of length* $d + 1$ (and *degree* or *order* d), where its action on the mesh $(\mathcal{V}, \mathcal{F})$ is defined by a new mesh $(\mathcal{V}', \mathcal{F})$ of identical connectivity but of geometry $\mathcal{V}' = (x', y', z')$ given by:

$$x' = P_x(K)x \ , \ \ y' = P_y(K)y \ , \ \ z' = P_z(K)z \tag{2}$$

A *rational filter* (Q, P) of orders (m, n) is defined by two collections of polynomials (Q_x, Q_y, Q_z) and (P_x, P_y, P_z) of degrees m, respectively n, whose action on the mesh $(\mathcal{V}, \mathcal{F})$ is defined by the new mesh $(\mathcal{V}', \mathcal{F})$ through:

$$Q_x(K)x' = P_x(K)x \ , \ \ Q_y(K)y' = P_y(K)y \ , \ \ Q_z(K)z' = P_z(K)z \tag{3}$$

To avoid possible confusions, we assume $Q_x(K), Q_y(K)$ and $Q_z(K)$ invertible. We point out the filtered mesh has the same connectivity as the original mesh; only the geometry changes. Note also the filter works for non-manifold connectivity as well.

In this report we consider only polynomial filters, i.e. rational filters of the form $(1, P)$. In [DeMeScBa99], the authors considered the case $(Q, 1)$. Note the distinction between polynomial and rational filters is artificial. Indeed, any rational filter is equivalent to a polynomial filter of length nV, in general, and in fact, any polynomial filter of degree larger than nV is equivalent to a polynomial filter of degree at most $nV - 1$. These facts are results of the Cayley-Hamilton theorem (see [FrInSp79], for instance) that says the characteristic polynomial of K vanishes when applied on K. Therefore:

$$Q(K)^{-1}P(K) = P_0(K) \tag{4}$$

for some polynomial P_0 of degree at most $nV - 1$. Hence the notion of IIR (Infinite Impulse Response) filter does not have any correspondence in the mesh topology based filtering, because any polynomial of higher order or rational filter is equivalent to a polynomial filter of degree at most $nV - 1$, thus a FIR (Finite Impulse Response) filter. However, the difference between

polynomial and rational filters lays in their implementation. The polynomial filter is easily implemented by a forward iteration scheme. The rational filter can be implemented by a forward-backward iteration scheme:

$$\begin{aligned} w &= P_x(K)x \\ Q_x(K)x' &= w \end{aligned} \tag{5}$$

involving the solution of a linear system of size nV. For small degrees m, n compared to nV (when the rational form has an advantage), the backward iteration turns into a sparse linear system, and thus efficient methods can be applied to implement it.

Two particular filtering schemes are of special importance to us and are studied next. The first scheme is called the *Zeroth Order Filter* and is simply defined by the constant polynomial 1:

$$P_Z(X) = (1, 1, 1) \tag{6}$$

Technically speaking, with the order definition given before, this is *a* zero order filter, but the most general form of zero order filters would be constant polynomials, not necessary 1. However, throughout this paper we keep this convention to call the constant polynomial 1, the Zeroth Order Filter. Note its action is trivial: it does not change anything.

The second distinguished filtering scheme is called *Gaussian Smoothing* and it is a first order filter defined by:

$$P_G(X) = (1 - X, 1 - X, 1 - X) \tag{7}$$

Using the definition of K and the filter action on the mesh geometry, the geometry of the Gaussian filtered mesh is given by:

$$x' = DMx \ , \ y' = DMy \ , \ z' = DMz \tag{8}$$

which turns into the following explicit form (using the position vectors r_i and the first order neighborhood i^* of vertex i):

$$r_i^{'} = \frac{1}{|i^*|} \sum_{v \in i^*} r_v \tag{9}$$

In other words, the new mesh geometry is obtained by taking the average of the first order neighbors positions on the original mesh.

3 The Progressive Approximation Algorithms

In Progressive Transmission schemes, the original mesh is represented as a sequence of succesively simplified meshes obtained by edge collapsing and vertex removal. Many simplification techniques have been proposed in the literature. For instance in [TaGuHoLa98] the Progressive Forest Split method is used. It consists of partitioning the mesh into disjoint patches and in each patch a connected sequence of edge collapsing is performed.

The meshes we are using here have been simplified by a clustering procedure. First all the coordinates are normalized so that the mesh is included in a 3D unit cube. Next the cube is divided along each coordinate axis into 2^B segments (B is the *quantizing rate*, representing the number of bits per vertex and coordinate needed to encode the geometry), thus obtaining 2^{3B} smaller cubes. In each smaller cube all the edges are collapsed to one vertex placed in the center of the corresponding cube. The mesh such obtained represents the quantized mesh at the finest resolution level. The coarsening process proceeds now as follows: 2^{3K} smaller cubes are replaced by one of edge size 2^K times bigger, and all the vertices inside are removed and replaced by one placed in the middle of the bigger cube. Next, the procedure is repeated until we obtain a sufficiently small number of vertices (i.e. a sufficient coarse resolution).

At each level of resolution, the *collapsing ratio* (i.e. the number of vertices of the finer resolution, divided by the number of vertices of the coarser resolution) is not bigger than 2^{3K}. In practice, however, this number could be much smaller than this bound, in which case some levels may be skipped. After l steps, the number of bits needed to encode one coordinate of any such vertex is $B - lK$. Thus, if we consider all the levels of detail and a constant collapsing ratio R, the total number of bits per coordinate needed to encode the geometry becomes:

$$M_b = NB + \frac{N}{R}(B - K) + \frac{N}{R^2}(B - 2K) + \cdots + \frac{N}{R^L}(B - LK)$$

where N is the initial number of vertices and L the numbers of levels. Assuming $\frac{1}{R^L} \ll 1$ we obtain $M_b = NB\frac{R}{R-1} + NK\frac{R}{(R-1)^2}$. Thus, the number of bits per vertex (of initial mesh) and coordinate turns into:

$$N_{bits} \simeq \frac{R}{R-1}(B + \frac{K}{R-1}) \ \ [bits/vertex \cdot coordinate] \tag{10}$$

Thus, if we quantize the unit cube using $B = 10$ bits and we resample at each level with a coarsening factor of $2^K = 2$ and a collapsing ratio $R = 2$, we obtain the sequence has $B/K = 10$ levels of details encoded using an average of 22 $bits/vertex \cdot coordinate$, or 66 $bits/vertex$ (including all three coordinates). Thus a single resolution encoding would require only B (10, in this example) bits per vertex and coordinate in an uncompressed encoding. Using the clustering decomposition algorithm, the encoding of all levels of details would require N_{bits} (given by (10)), about 22 in this example, which is more than twice the single resolution rate.

In this scenario no information about the coarser resolution mesh has been used to encode the finer resolution mesh. In a progressive transmission, the coarser approximation may be used to predict the finer approximation mesh and thus only the differences should be encoded and transmitted. Moreover, the previous computations did not take into account the internal redundancy of the bit stream. An entropic encoder would perform much better than (10). In this paper we do not discuss the connectivity encoding problem, since we are interested in the geometry encoding only. Yet, we assume at each level of detail the decoder knows the connectivity of that level mesh.

Suppose that

$$(Mesh_{nL-1}, map_{nL-2,nL-1}, Mesh_{nL-2}, map_{nL-3,nL-2}, \ldots, map_{1,2}, Mesh_1, map_{0,1}, Mesh_0)$$

is the sequence of meshes obtained by the coarsening algorithm, where $Mesh_{nL-1}$ is the coarsest resolution mesh, $Mesh_0$ the finest resolution mesh, and $map_{l-1,l} : \{0, 1, \ldots, nV_{l-1} - 1\} \to \{0, 1, \ldots, nV_l - 1\}$ is the *collapsing map* that associates to the nV_{l-1} vertices of the finer resolution mesh the nV_l vertices of the coarser resolution mesh where they collapse. Each mesh $Mesh_l$ has two components $(Geom_l, Conn_l)$, the geometry and connectivity respectively, as explained earlier. We are concerned with the encoding of the sequence of geometries

$$(Geom_{nL-1}, Geom_{nL-2}, \ldots, Geom_1, Geom_0).$$

Our basic encoding algorithm is the following:

The Basic Encoding Algorithm

Step 1. Encode $Geom_{nL-1}$ using an entropic or arithmetic encoder;

Step 2. For $l = nL - 1$ down to 1 repeat:

Step 2.1 Based on mesh $Mesh_l$ and connectivity $Conn_{l-1}$ find a set of parameters $Param_{l-1}$ and construct a predictor of the geometry $Geom_{l-1}$:

$$\hat{Geom}_{l-1} = Predictor(Mesh_l, Conn_{l-1}, map_{l-1,l}; Param_{l-1})$$

Step 2.2 Encode the parameters $Param_{l-1}$;

Step 2.3 Compute the approximation error $Diff_{l-1} = Geom_{l-1} - \hat{Geom}_{l-1}$ and encode the differences. $\diamond$

The decoder will reconstruct the geometry at each level by simply adding up the difference to his prediction:

$$Geom_{l-1} = Predictor(Mesh_l, Conn_{l-1}, map_{l-1,l}; Param_{l-1}) + Diff_{l-1}$$

It is clear that different predictors yield different performance results. In the next section we present several desirable properties of the encoding scheme.

The data packet structure is represented in Table 1.

$Mesh_{nL-1}$	$Map_{nL-2,nL-1}\&Conn_{nL-2}$	$Param_{nL-2}$	$Diff_{nL-2}$	$\cdots$
$\cdots$	$Map_{0,1}\&Conn_0$	$Param_0$	$Diff_0$	

Table 1: The data packet structure for the basic encoding algorithm

The *Predictor* consists of applying the sequence of operators *extension*, where the geometry of level l is extended to level $l - 1$, and *update*, where the geometry is updated using a polynomial filter whose coefficients are called *parameters* of the predictor and whose matrix is the finer resolution incidence matrix K_{l-1}.

The extension step is straightforwardly realized using the collapsing maps:

$$r_i^{l-1;ext} = r^l_{map_{l-1,l}(i)} \tag{11}$$

Thus the first "prediction" of the new vertex i is on the same point where it collapses, i.e. the position of the vertex $map_{l-1,l}(i)$ in mesh l. Next, the updating step is performed by polynomial filtering as in (2). The filter coefficients are the predictor

parameters and have to be found and encoded. On each coordinate we use a separate filter. In the next section we introduce different criteria to measure the prediction error associated to a specific property. In [TaGuHoLa98] Taubin filters (i.e. of the form $P(X) = (1 - \lambda X)(1 + \mu X)$) have been used as predictors, but no optimization of the parameters has been done. Here we use more general linear filters taking into account several performance criteria as well.

More specific, let us denote by $x_i^{l-1;ext}$ the nV_{l-1}-vector of x-coordinates obtained by extension (11), and by $x^{l-1;updt}$ the filtered vector with the polynomial $P_x(X) = \sum_{k=0}^{d} c_k X^k$ of degree d,

$$x^{l-1;updt} = P_x(K_{l-1})x^{l-1;ext} \tag{12}$$

Let r^{l-1} denote the $nV_{l-1} \times 3$ matrix containing all the coordinates in the natural order, $r^{l-1} = [x^{l-1}|y^{l-1}|z^{l-1}]$. Similar for $r^{l-1;updt}$. The update is our prediction for the geometry $Geom_{l-1}$. Then the coefficients are chosen to minimize some l^p-norm of the prediction error:

$$\min_{Filters\ Coefficients} J^{l-1} = \|r^{l-1} - r^{l-1;updt}\|_{l^p} \tag{13}$$

Note the optimization problem decouples into 3 independent optimization problems, because we allow different filters on each coordinate. The polynomial $P_x(X)$ can be represented either in the power basis, i.e. $P_x(X) = \sum_{k=0}^{d} c_k X^k$, or in another basis. We tried the Chebyshev basis as well, in which case $P_x(X) = \sum_{k=0}^{d} c_k T_k(X)$ with T_k the k^{th} Chebyshev polynomial. On each coordinate, the criterion J^{l-1} decouples as follows:

$$J^{l-1} = \|(J_x^{l-1}, J_y^{l-1}, J_z^{l-1})\|_{l^p} \ , \ J_x^{l-1} = \|A_x c_x^{l-1} - x^{l-1}\|_{l^p} \ , \ J_y^{l-1} = \|A_y c_y^{l-1} - y^{l-1}\|_{l^p} \ , \ J_z^{l-1} = \|A_z c_z^{l-1} - z^{l-1}\|_{l^p} \ , \tag{14}$$

where the $nV \times d + 1$ matrix A_x is either

$$A_x = [x^{l-1;updt}|K_{l-1}x^{l-1;updt}|\cdots|K_{l-1}^d x^{l-1;updt}] \tag{15}$$

in the power basis case, or

$$A_x = [x^{l-1;updt}|T_1(K_{l-1})x^{l-1;updt}|\cdots|T_d(K_{l-1})x^{l-1;updt}] \tag{16}$$

in the Chebyshev basis case. c_x^{l-1} is the $d + 1$ - vector of the x-coordinate filter coefficients and x^{l-1} the nV_{l-1}-vector of the actual x-coordinates all computed at level $l - 1$. Similar for A_y, A_z, c_y, c_z, and y^{l-1}, z^{l-1}.

The Basic Encoding Algorithm can be modified to a more general context. The user may select the levels for which the differences are sent. Then, for those levels the differences are not sent, the extension step to the next level has to use the predicted values instead of the actual values of the current level. In particular we may want to send the differences starting with level $nL - 1$ and going down to some level $S + 1$; then, from level S down to level 0 we do not send any difference but just the parameters, excepted for the level 0 when we send the differences as well. The algorithm just described is presented next:

The Variable Length Encoding Algorithm

Step 1. Encode $Mesh_{nL-1}$;

Step 2. For $l = nL - 1$ down to S repeat:

Step 2.1 Estimate the parameters $Param_{l-1}$ by minimizing J^{l-1}, where the predictor uses the true geometry of level l, $Geom_l$:

$$\hat{Geom}_{l-1} = f(Conn_l, Conn_{l-1}, map_{l-1,l}, Geom_l; Param_{l-1})$$

Step 2.2 Encode the parameters $Param_{l-1}$;

Step 2.3 If $l \neq S$, encode the differences $Diff_{l-1} = Geom_{l-1} - \hat{Geom}_{l-1}$;

Step 3. For $l = S - 1$ down to 1

Step 3.1 Estimate the parameters $Param_{l-1}$ by minimizing J^{l-1} where the predictor uses the estimated geometry of level l:

$$\hat{Geom}_{l-1} = f(Conn_l, Conn_{l-1}, map_{l-1,l}, \hat{Geom}_l; Param_{l-1})$$

Step 3.2 Encode the parameters $Param_{l-1}$;

Step 4. Using the last prediction of level 0, encode the differences, $Diff_0 = Geom_0 - \hat{Geom}_0$. $\diamond$

In this case the data packet structure is the one represented in Table 2. In particular, for $S = nL$ only the last set of differences is encoded. This represents an alternative to the single-resolution encoding scheme.

$Mesh_{nL-1}$	$Map_{nL-2,nL-1}\&Conn_{nL-2}$	$Param_{nL-2}$	$Diff_{nL-2}$	$\cdots$	$Map_{s+2,2+1}\&Conn_{s+1}$

$Param_{s+1}$	$Diff_{s+1}$	$Map_{s+1,s}\&Conn_s$	$Param_s$	$Map_{s,s-1}\&Conn_{s-1}$	$Param_{s-1}$	$\cdots$

$\cdots$	$Map_{2,1}\&Conn_1$	$Param_1$	$Map_{0,1}\&Conn_0$	$Param_0$	$Diff_0$

Table 2: The data packet structure for the variable encoding algorithm

4 Desired Properties

In this section we discuss three properties we may want the encoding scheme to possess. The three peoperties, accuracy, robustness, compression ratio, yield different optimization problems all of the type mentioned before. The l^p-norm to be minimized is different in each case. For accuracy the predictor has to minimize the l^∞ norm, for robustness the l^2 norm should be used, whereas the compression ratio is optimized for $p \in [1,2]$ in general. Thus a sensible criterion should be a trade-off between these various norms. Taking the computational complexity into account, we have chosen the l^2-norm as our criterion and in the following section of examples we show several results we have obtained.

4.1 Accuracy

Consider the following scenario: Suppose we choose $S = nL$, the number of levels, in the Variable Length Encoding Algorithm. Suppose also the data block containing the level zero differences is lost (note this is the only data block containing differences because $S = nL$) In this case we would like to predict the finest resolution mesh as accurately as possible based on the available information. Equivalently, we would like to minimize the distance between $Mesh_0$ and the prediction $\hat{Mesh}_0$, under the previous hypotheses. There are many ways of measuring mesh distances. One such measure is the Haussdorf distance. Although it describes very well the closeness of two meshes, the Haussdorf distance yields a computational expensive optimization problem. Instead of Haussdorf distance one can consider the maximum distance between vertices (i.e. the l^∞-norm, see [Al&all88]):

$$\varepsilon_a = \max_{0 \le i \le nV_0 - 1} \|r_i^0 - \hat{r}_i^0\| =: \|r^0 - \hat{r}^0\|_{l^\infty}$$

Note the l^∞-norm is an upper bound for the Haussdorf distance. Consequently ε_a controls the meshes closeness as well. As mentioned in the previous section, the optimization problem (13) decouples in three independent optimization problems. For $p = \infty$, these have the following form:

$$\inf_c \|Ac - b\|_{l^\infty} \tag{17}$$

where A was introduced by (15) and (16), depending on the basis choice, c is the nf-vector of unknown fiter coefficients, and b is one of the three vectors x, y or z. For $0 \le i \le nV - 1, 0 \le j \le fL - 1$, $A = [a_{ij}]$, $b = (b_i)$ and writing $c_j = f_j - g_j$ with $f_j \ge 0$, the positive part, and $g_j \ge 0$, the negative part of c_j (thus at least one of them is always zero), the optimization problem (17) turns into the following linear programming problem:

$$\begin{aligned} \max_{w,f_j,g_j,u_i,v_i} \quad & [-w - \varepsilon \textstyle\sum_{j=0}^{d}(f_j + g_j)] \\ subject\ to: \quad & w, f_j, g_j, u_i, v_i \ge 0 \\ & b_i = u_i + \textstyle\sum_{j=0}^{d} a_{ij}(f_j - g_j) - w \\ & b_i = -v_i + \textstyle\sum_{j=0}^{d} a_{ij}(f_j - g_j) + w \end{aligned} \tag{18}$$

with ε a small number to enforce at least one of f_j or g_j to be zero (for instance $\varepsilon = 10^{-6}$). With the standard simplex algorithm, this problem requires the storage of a $(2nV + 2) \times (2nV + 2d + 2)$-matrix (the so called *tableaux*) which is prohibitive for large number of vertices (nV of order 10^5, for instance). In any case, the moral of this subsection is to point out that the more accurate predictor is the one that achieves a lower l^∞-norm error.

4.2 Robustness

Consider now the following scenario: the differences associated to the prediction algorithm are not set to zero but perturbed by some random quantities. This may be due to several causes. We can either imagine irretrievable transmission errors or even a resampling process at the transmitter to reduce the code length of the entire object. In any case we assume the true difference d_i is perturbed by some stochastic process ν_i. Thus the reconstructed geometry has the form $x_i^{l-1;reconstr} =$

$x_i^{l-1;updt} + diff_i + \nu_i$. We assume the perturbations are about of the same size as the prediction differences. Next suppose we want to minimize *in average* the effect of these perturbations. Then one such criterion is the noise variance $E[\nu_i^2]$. Assuming the stochastic process is ergodic, it follows the noise variance can be estimated by the average of all the coordinate perturbations: $E[\nu_i^2] = \frac{1}{N}\sum_{i=0}^{N-1}\nu_i^2$. Next, since the perturbation is of the same order as the prediction error, the later term can be replaced by the average of the differences. Hence we want to minimize:

$$E[\nu_i^2] \simeq \frac{1}{N}\sum_{i=0}^{N-1} d_i^2$$

This shows that a criterion of robustness is the total energy of the differences. In this case our goal to increase the robustness of the algorithm is achieved by decreasing the l^2-norm of the prediction errors. Thus the filters are the solvers of the optimization problem (13) for $p = 2$. The solution in terms of filter coefficients is very easily obtained by using the pseudoinverse matrix. Thus the solution of:

$$\inf_c \|Ac - b\|_{l^2}$$

is given by:

$$c = (A^T A)^{-1} A^T b \tag{19}$$

4.3 Compression Ratio

The third property we discuss now is the compression ratio the algorithm achieves. In fact, if no error or further quantization is assumed, the compression ratio is perhaps the most important criterion in judging and selecting an algorithm. In general estimating compression ratios is a tough problem due to several reasons. First of all one should assume a stochatic model of the data to be encoded. In our case we encode the vectors of prediction errors, which in turn depend on the mesh geometry and the way we choose the filters coefficients. Next one should have an exact characterization of the encoder's compression ratio. The best compression ratio, assuming a purely stochastic data, is given by Shannon's entropic formula and consequently by the entropic encoder which strives to achieve this bound (Shannon-Fano and Huffman codings - see [ZiTr90] or [DaGr76]). However the entropic encoder requires some a priori information about the data to be sent, as well as overhead information that may affect the global compression ratio. Alternatively one can use adaptive encoders like the adaptive arithmetic encoder as in the JPEG/MPEG standards (see [PeMi93]). This encoder may perform better in practice than *blind* entropic or arithmetic encoders, however it has the important shortcoming that its compression ratio is not characterized by a closed formula. In any case, for purely stochastic data the best compression ratio is bounded by Shannon's formula which we discuss next. We thus assume our bit sequence encoding scheme achieves this optimal bound. Suppose the quantized differences x_i, $0 \le i \le N-1$, are independently distributed and have a known probability distribution, say $p(n)$, $-2^{B-1} \le n \le 2^B$. Thus $p(n)$ is the probability that a difference is n. In this case the average (i.e. expected value) of the number of bits needed to encode one such difference is not less than:

$$R_{Shannon} = -\sum_{n=-2^{B-1}}^{2^{B-1}} p(n) log_2 p(n)$$

where 2^B is the number of quantization levels (see [ZiTr90]). Assuming now the ergodic hypothesis holds true, $p(n)$ can be replaced by the repetition frequency $p(n) = \frac{f(n)}{N}$, where $f(n)$ is the repeatition number of the value n and N is the total number of values (presumably $N = 3nV$). Thus, if we replace the first $p(n)$ in the above formula by this frequency, the sum turns into

$$R = -\frac{1}{N}\sum_{i=0}^{N-1} log_2 p(n = x_i)$$

Note the summation index has changed. At this point we have to asssume a stochastic model for the prediction errors. We consider the power-type distribution that generalizes both the Gaussian and Laplace distributions, that are frequently used in computer graphics models (see [PaRo99], for instance):

$$p(x) = \frac{a^{\frac{1}{\alpha}}\alpha}{2\Gamma(\frac{1}{\alpha})} exp(-a|x|^\alpha) \tag{20}$$

where $\Gamma(x)$ is the Euler's Gamma function (to normalize the expression) and a is a parameter. For $\alpha = 1$ it becomes the Laplace distribution, whereas for $\alpha = 2$ it turns into the Gauss distribution. Then, the previous rate formula turns into:

$$R = R_0 + \frac{a log_2 e}{N}\sum_{i=0}^{N-1} |x_i|^\alpha \ , \ R_0 = 1 + log_2\Gamma(\frac{1}{\alpha}) - log_2\alpha - \frac{1}{\alpha}\log_2 a$$

Now we replace the parameter a by an estimate of it. An easy computation shows the expected value of $|x|^{\alpha}$ for the α-power p.d.f. (20) is $E[|x|^{\alpha}] = \frac{1}{\alpha a}$. Thus we get the following estimator for the parameter a:

$$\hat{a} = \frac{1}{\alpha} \frac{N}{\sum_{i=0}^{N-1} |x_i|^{\alpha}}$$

and the above formula of the rate becomes:

$$R = \rho_0(\alpha) + \frac{1}{\alpha} \log_2 [\sum_{i=0}^{N-1} |x_i|^{\alpha}] \ , \ \ \rho_0(\alpha) = 1 + \log_2 \frac{\Gamma(\frac{1}{\alpha})}{\alpha} + \frac{1}{\alpha} \log_2 \frac{e\alpha}{N} \tag{21}$$

Consider now two linear predictors associated to two different linear filters. Each of them will have different prediction errors. If we assume the prediction errors are independent in each case and distributed by the same power law with exponent α but maybe different parameters a_1, respectively a_2, then the prediction scheme that yields the sequence of differences with smaller l^{α}-norm has a better (entropic) compression bound and therefore is more likely to achieve a better compression ratio. Equivalently, the p.d.f. that has a larger parameter a, or is narrower, would be encoded using fewer bits.

The argument we presented here suggests that a better compression ratio is achieved by the prediction scheme that minimizes the l^{α}-norm of the prediction error, where α is the p.d.f.'s characteristic exponent (when it is a power-type law), usually between 1 (the Laplace case) and 2 (the Gaussian case). For $p = 2$ the optimizing filter is found by using the pseudoinverse of A as in (19). For $p = 1$, the optimizer solves the linear programming problem:

$$\begin{array}{rc} \max\limits_{f_j, g_j, u_i, v_i} & \sum_{i=0}^{N-1} [-u_i - v_i - \varepsilon \sum_{j=0}^{d} (f_j + g_j)] \\ subject\ to: & f_j, g_j, u_i, v_i \geq 0 \\ & b_i = u_i - v_i + \sum_{j=0}^{d} a_{ij} (f_j - g_j) \end{array} \tag{22}$$

with ε as in (18), which involves (in the simplex algorithm) a $(N+2) \times (2N + 2d + 1)$ matrix and the same computational problems as (18).

5 Examples

In this section we present a number of examples of our filtering algorithm. For several meshes we study the accuracy the fine resolution mesh is approximated, and also the compression ratio obtained for different filter lengths.

First we analyze the Basic Encoding Algorithm presented in section 3. The filters coefficients are obtained by solving the optimal problem (13) for $p = 2$, i.e. we use the least squares solution.

The car mesh represented in Figure 1 (left) having $nV_0 = 12784$ vertices and 24863 faces is decomposed into a sequence of 8 levels of details. The coarsest resolution mesh of $nV_7 = 219$ vertices is rendered in Figure 1 (right). We used several filter lengths to compress the meshes. In particular we study four types of filters, namely the Zeroth Order Filter, the Gaussian Smoother and filters of order $d = 1$ and $d = 3$ (decomposed in power basis). The last two filters will be termed as "higher order filters", although their order is relatively low. To check the accuracy of the approximation we used the prediction algorithm assuming the differences are zero at all levels. The four meshes corresponding to the four filters are represented in Figure 2.

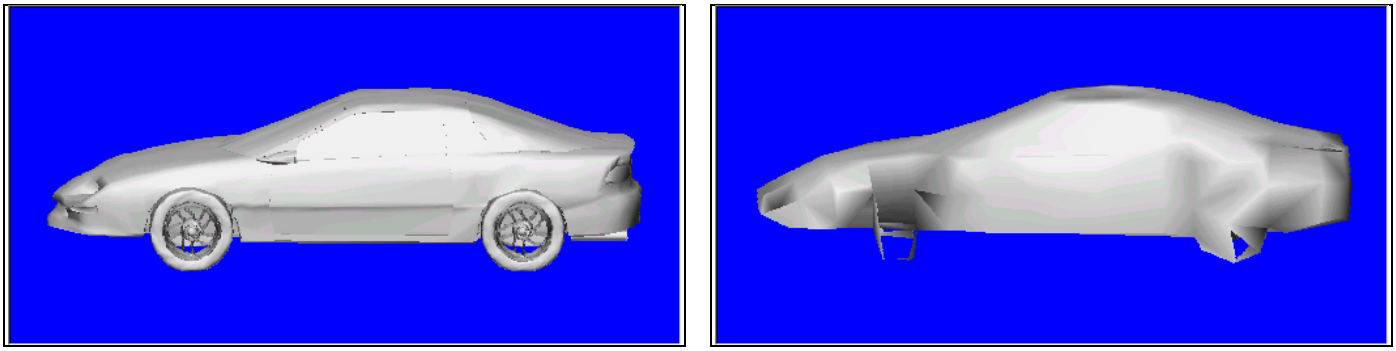


Figure 1: The car mesh at: the finest resolution (left) and the coarsest resolution (right) after 7 levels of reduction.

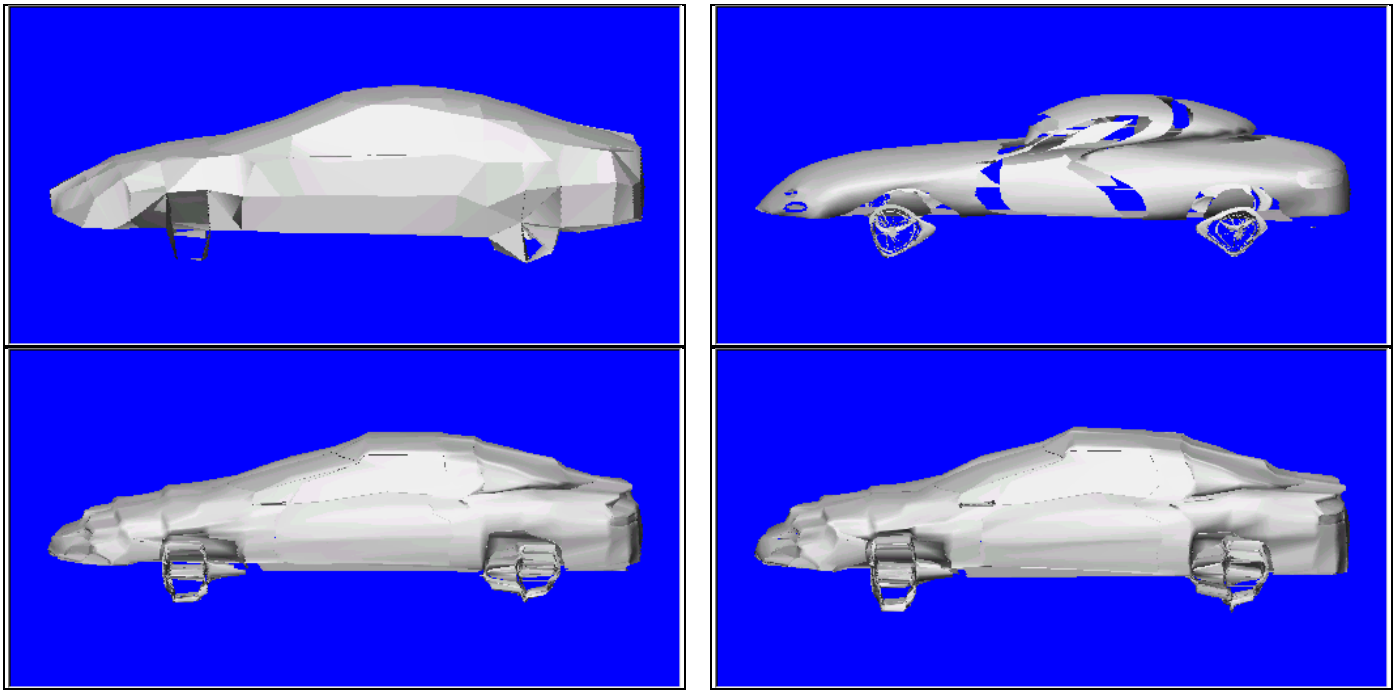


Figure 2: The car mesh at the finest resolution level when no difference is used and the filtering is performed by: the Zeroth Order Filter (top left), the Gaussian Smoother (top right), the least squares filter of order 1 (bottom left) and the least squares filter of order 3 (bottom right).

Note the Zeroth Order Filter does not change the geometry at all (because of its pure extension nature). It gives the worst approximation of the mesh, yet it has the best compression ratio (see below). The Gaussian filter smoothes out all the edges, a natural consequence since it really corresponds to a discrete diffusion process. The higher order filters (i.e. first order and third order) trade-off between smoothing and compression.

In terms of the compression ratio, the four filters have performed as shown in Table 3. Varying the filter length we found the compression ratios indicated in Table 4. All the results apply to the geometry component only. The connectivity is not presented here.

Filter	Zeroth Order	Gaussian Smoothing	LS of Degree 1	LS of Degree 3
Coarsest mesh	795	795	795	795
Coefficients	0	0	168	336
Differences	21840	64611	22520	22574
Total (bytes)	22654	65425	23503	23725
Rate (bits/vertex)	14.17	40.94	14.71	14.82

Table 3: Compression ratio results for several filtering schemes applied to the car mesh rendered in Figure 1.

Filter's Degree	1	2	3	4	5	6	7
bits/vertex	14.71	14.82	14.85	14.89	14.96	15.04	15.11

Table 4: Compression ratios for different filter lengths in power basis.

Next we study the Variable Length Encoding Algorithm for the four particular filters mentioned before with the parameter $S = nL$ (i.e. in the Single Resolution case). Thus the mesh geometry is obtained by successively filtering the extensions of the coarser mesh and, at the last level, the true differences are encoded. In terms of accuracy we obtained very similar meshes. More significantly are the compression ratios, shown in Table 5. To analyse the compression ratios of these four filters, we have also plotted the histogram of the errors on a semilogarithmic scale in Figure 3. Note the power-type p.d.f. hypothesis is well satisfied by the Zeroth, LS 1^{st} and LS 3^{rd} order filters, and less by the Gaussian smoother. Also as smaller the l^2-norm error gets, as narrower the p.d.f. and as smaller the rate becomes, in accordance with the conclusions of Section 4.3.

Equally important is how these errors are distributed on the mesh. In Figure 4 we convert the actual differences into a scale of colors and set this color as an attribute for each vertex. Darker colors (blue,green) represent a smaller error, whereas lighter colors (yellow, red) represent a larger prediction error. The darker the color the better the prediction and also the accuracy. All the errors are normalized with respect to the average l^2-norm error per vertex for that particular filter. The average l^2-norm error is given on the last row in Table 5.

Filter	Zeroth Order	Gaussian Smoothing	LS of Degree 1	LS of Degree 3
Coarsest mesh	795	795	795	795
Coefficients	0	0	168	336
Differences	27489	27339	25867	25532
Total (bytes)	28306	28156	26859	26690
Rate (bits/vertex)	17.71	17.62	16.81	16.68
$l^2\ \mathrm{error/vertex}(\cdot 10^{-4})$	11.96	18.64	9.33	8.44

Table 5: Compression ratios in the Single Resolution implementation of the Variable Length Encoding Algorithm applied to the car mesh rendered in Figure 1.

Note in the Single Resolution case there is no much difference among the filtering schemes considered. In particular the higher order filters perform better than the Zeroth Order Filter, and the Gaussian filter behaves similarly to the other filters. This is different to the Multi Resolution case in Table 3. There, the Gaussian filter behaves very poorly, and the Zeroth Order Filter gives the best compression ratio. In fact it is better to the Single Resolution case. On the other hand, with respect to the accuracy, the higher order filters give a more accurate approximation than the Zeroth Order Filter.

About the same conclusions hold for three other meshes we used: the round table, the skateboard and the piping construction.

The round table rendered in Figures 5, left, has $nV_0 = 11868$ vertices and 20594 faces. The coarsest resolution mesh (at level 8, pictured on the right side) has $nV_7 = 112$ vertices. The predicted mesh after 8 levels of decomposition when no difference is used, is rendered in Figure 6.

The skateboard mesh at the finest resolution (left, in Figure 9) has $nV_0 = 12947$ vertices and 16290 faces. At the coarsest resolution (right, in the same figure) it has $nV_7 = 125$ vertices.

The piping construction has $nV_0 = 18138$ vertices, and after 7 levels of details it is reduced to $nV_0 = 147$ vertices. The first simplification step achieves almost the theoretical bound: from 18138 vertices, the mesh is simplified to 2520 vertices. The original mesh and its approximations are rendered in Figures 13-14.

Filter	Zeroth Order	Gaussian Smoothing	LS of Degree 1	LS of Degree 3
Coarsest mesh	438	438	438	438
Coefficients	0	0	168	336
Differences	22739	51575	23645	23497
Total (bytes)	23196	52032	24274	24295
Rate (bits/vertex)	15.63	35.07	16.36	16.37

Table 6: Compression ratio results for several filtering schemes applied to the round table mesh rendered in Figure 5.

Filter's Degree	1	2	3	4	5	6	7
bits/vertex	16.36	16.34	16.37	16.50	16.62	16.73	16.88

Table 7: Compression ratios for different filter lengths, in power basis, for the round table.

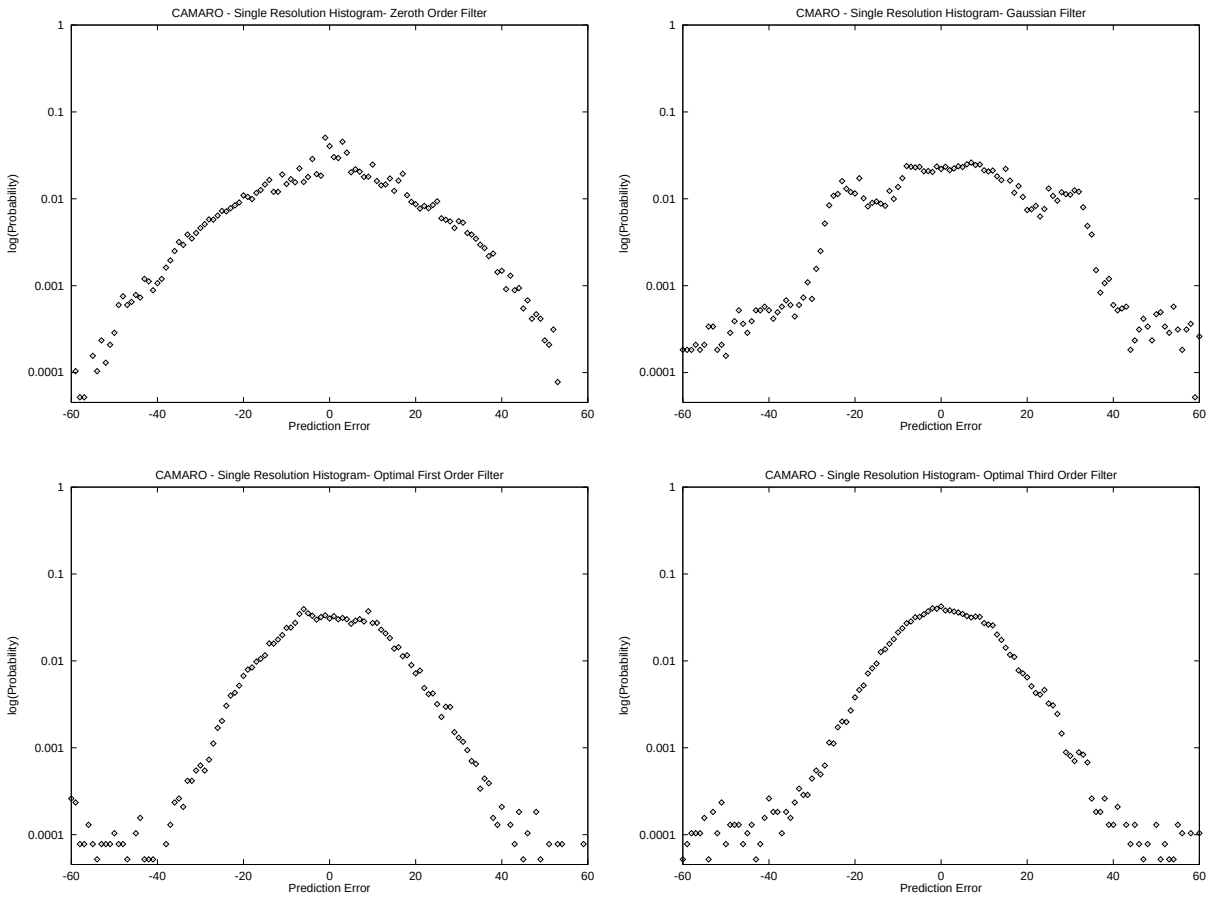


Figure 3: Semilog histograms of the prediction errors associated to the four filters for the Single Resolution scheme applied to the car mesh rendered in Figure 1.

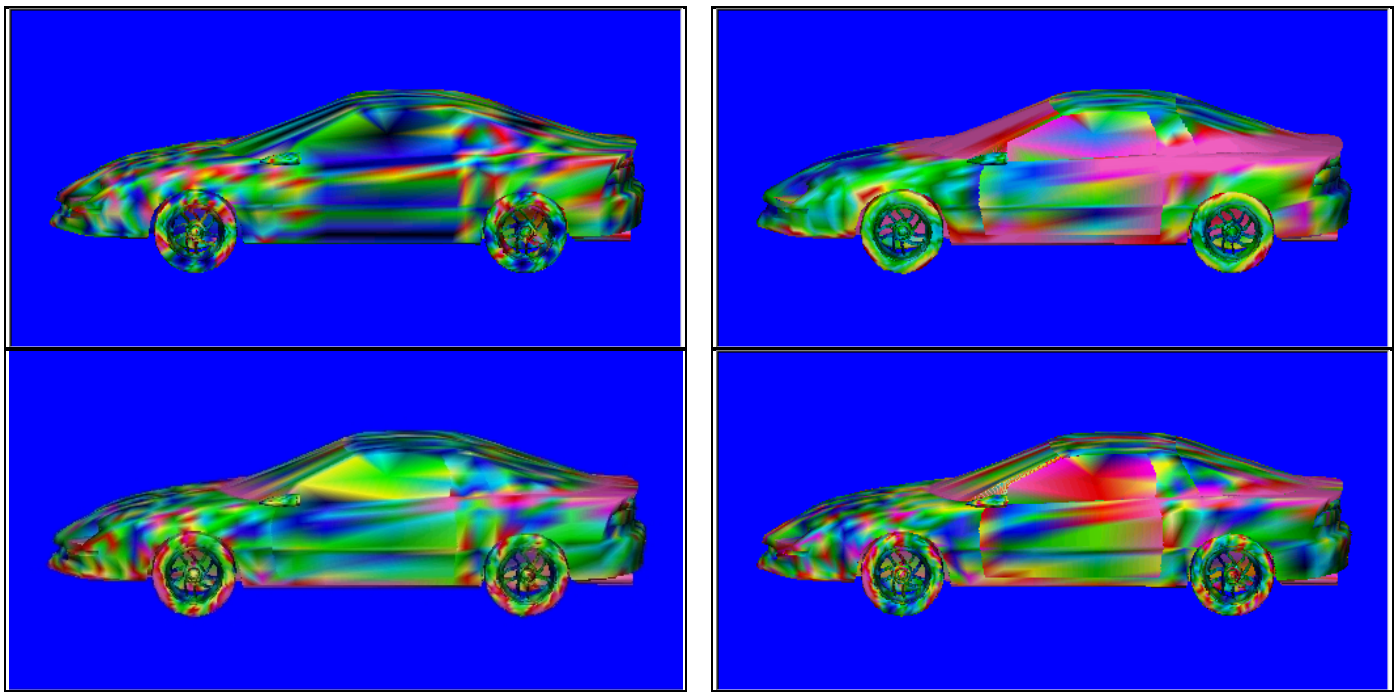


Figure 4: The color plot of the single resolution approximation errors for the four different filters: Zeroth Order Filter (upper left), Gaussian Smoothing (upper right), LS first order (lower left) and LS third order (lower right).

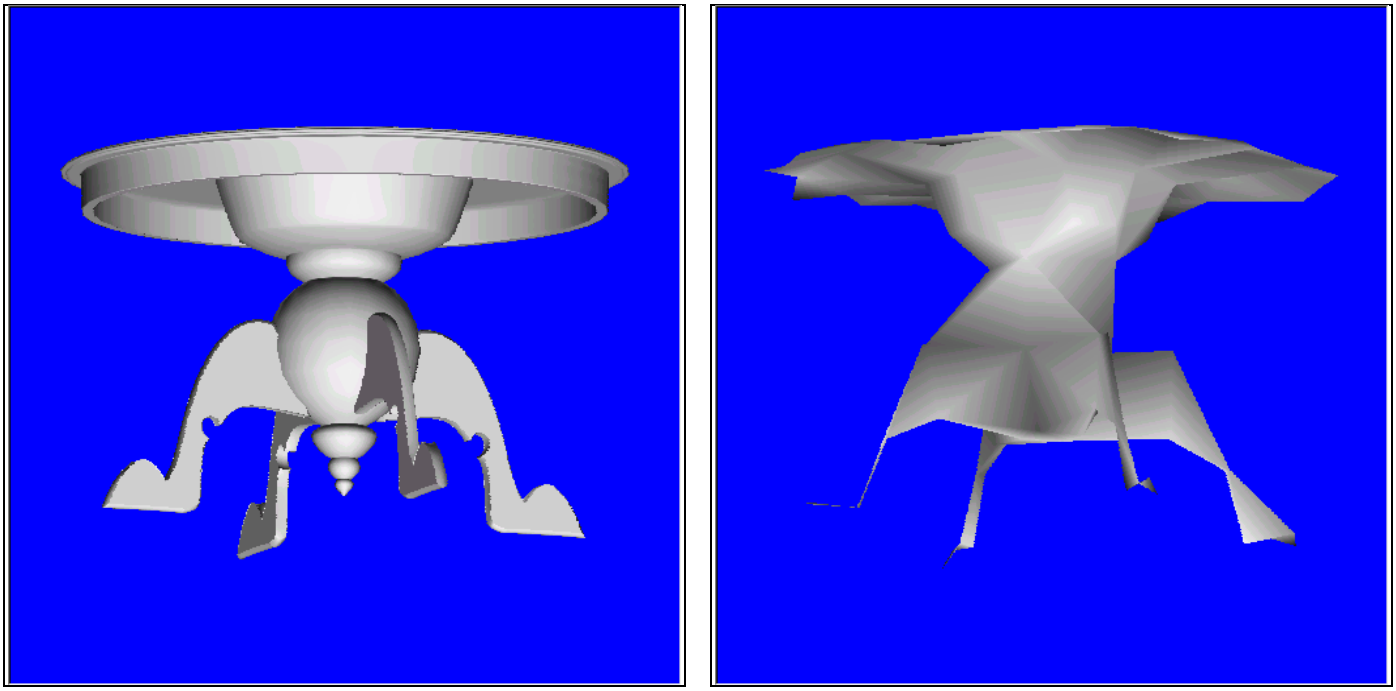


Figure 5: The round table mesh at: the finest resolution (left) and the coarsest resolution (right) after 7 levels of reduction.

Filter	Zeroth Order	Gaussian Smoothing	LS of Degree 1	LS of Degree 3
Coarsest mesh	438	438	438	438
Coefficients	0	0	168	336
Differences	30386	29138	27972	27377
Total (bytes)	30847	29599	28609	28146
Rate (bits/vertex)	20.79	19.95	19.28	18.97
l^2 error/vertex$(\cdot 10^{-2})$	15.85	17.38	11.17	9.97

Table 8: Compression ratios in the Single Resolution implementation of the Variable Length Encoding Algorithm applied to the round table mesh rendered in Figure 5.

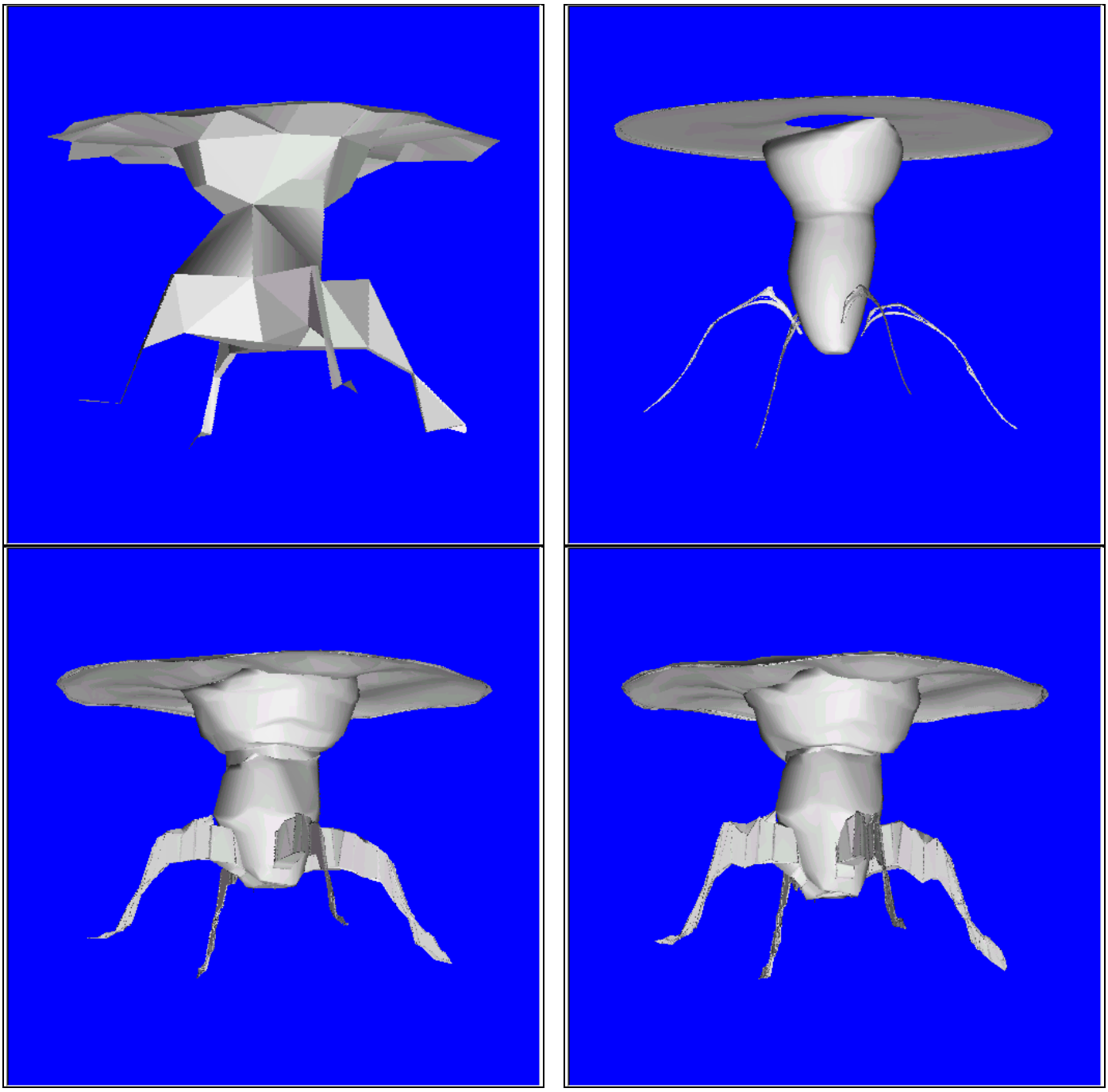


Figure 6: The round table mesh at the finest resolution level when no difference is used and the filtering is performed by: the Zeroth Order Filter (top left), the Gaussian Smoother (top right), the least squares filter of order 1 (bottom left) and the least squares filter of order 3 (bottom right).

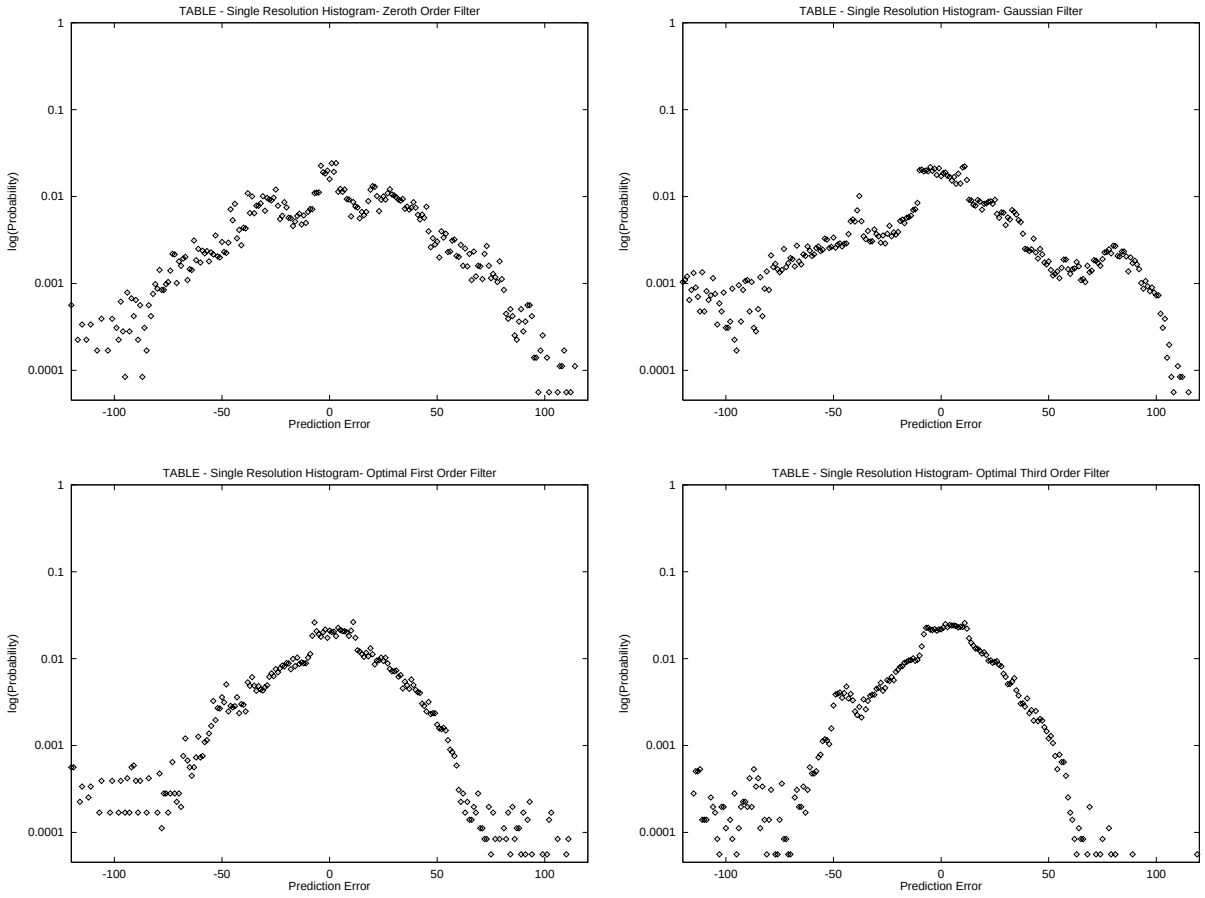


Figure 7: Semilog histograms of the prediction errors associated to the four filters for the Single Resolution scheme applied to the round table rendered in Figure 5.

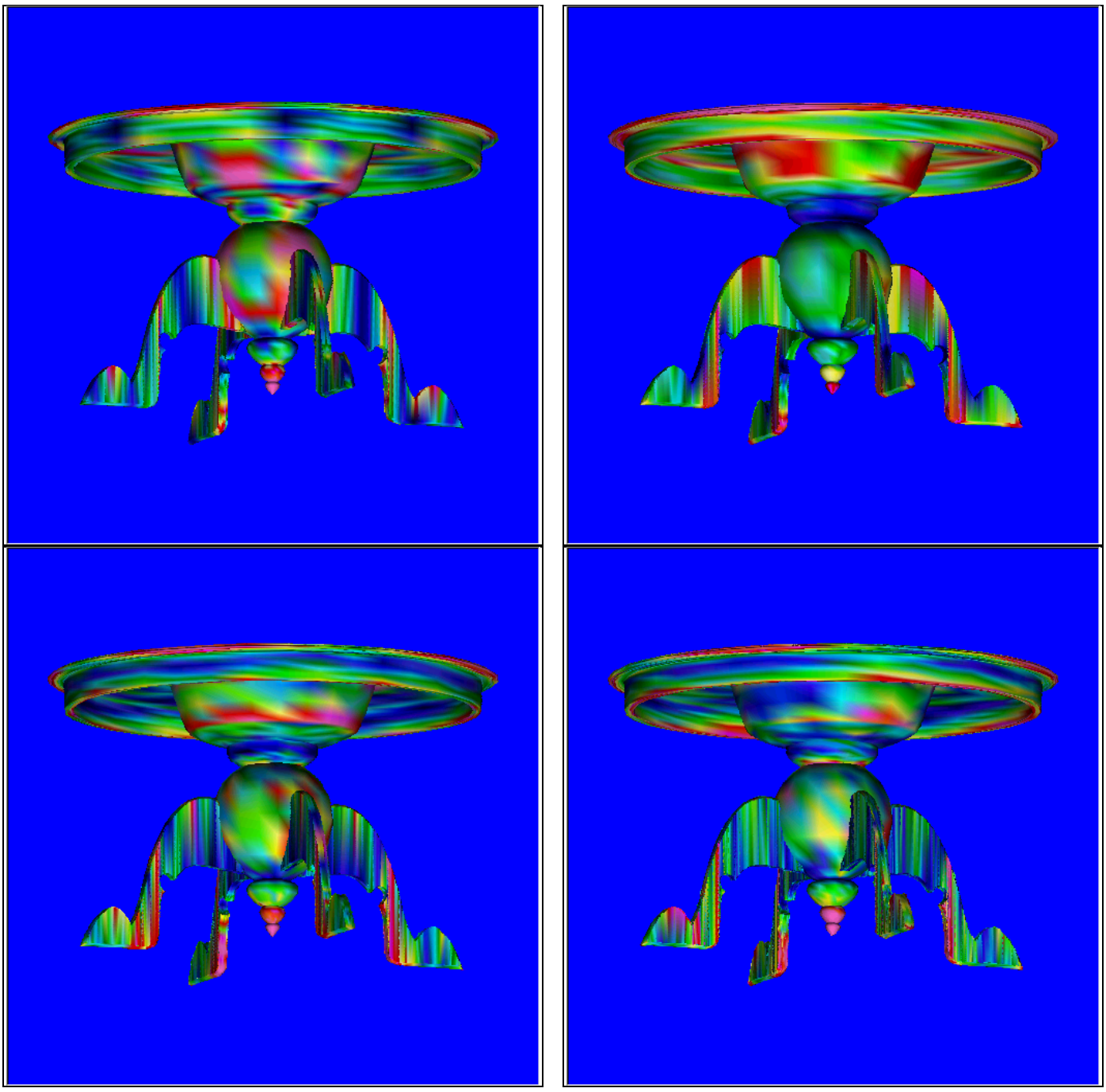


Figure 8: The color plot of the single resolution approximation errors for the four different filters: Zeroth Order Filter (upper left), Gaussian Smoothing (upper right), LS first order (lower left) and LS third order (lower right).

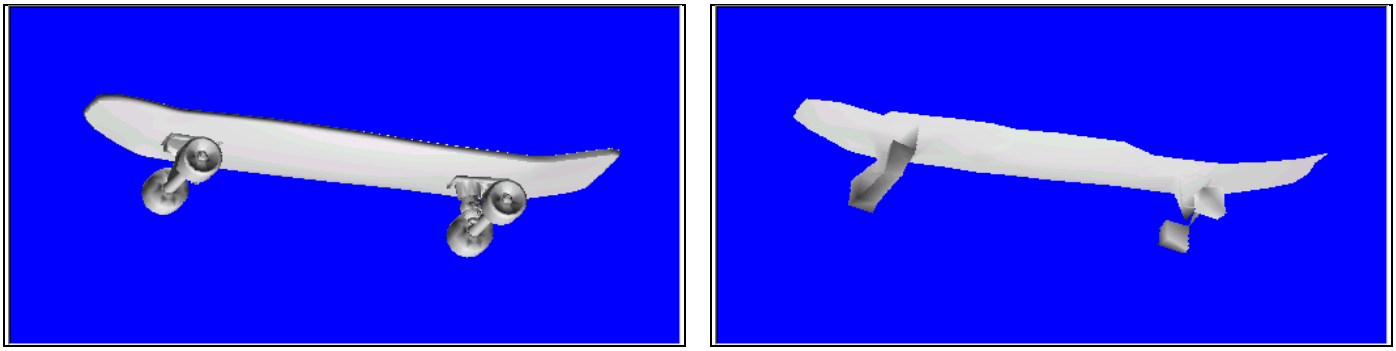


Figure 9: The skateboard mesh at: the finest resolution (left) and the coarsest resolution (right) after 8 levels of reduction.

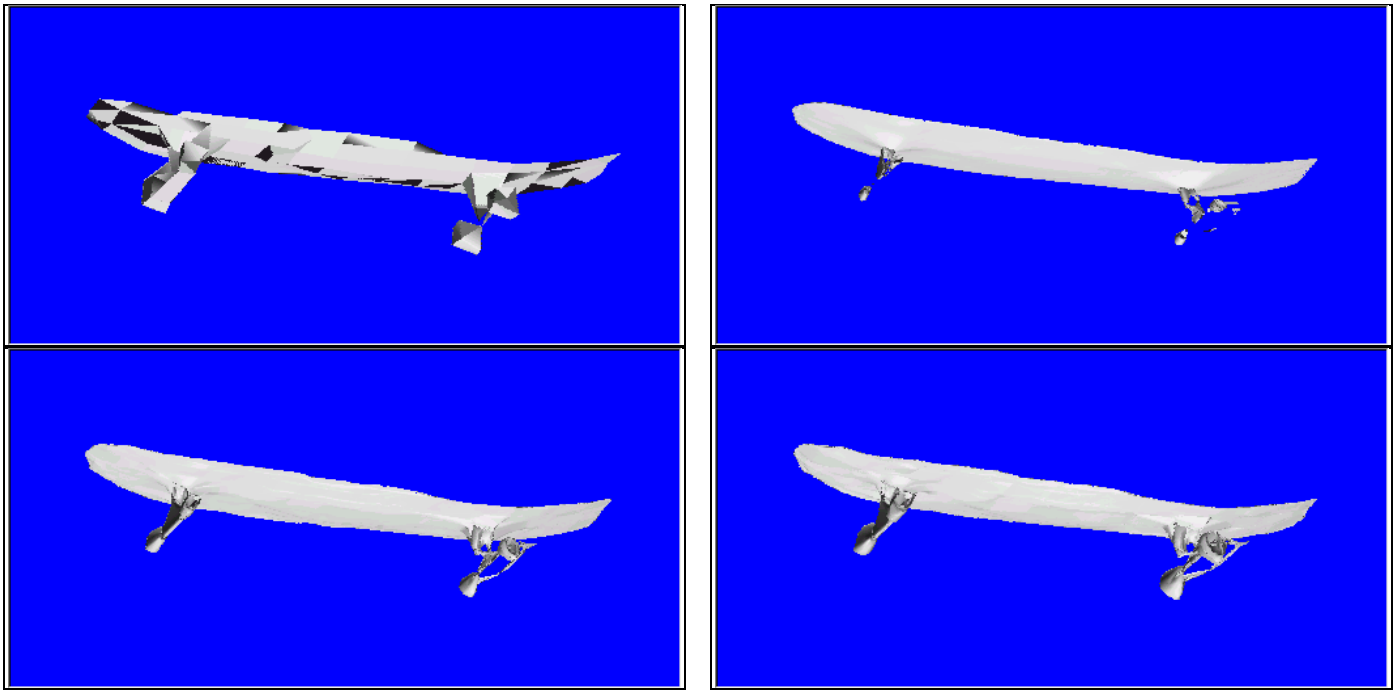


Figure 10: The skateboard mesh at the finest resolution level when no difference is used and the filtering is performed by: the Zeroth Order Filter (top left), the Gaussian Smoother (top right), the least squares filter of order 1 (bottom left) and the least squares filter of order 3 (bottom right).

Filter	Zeroth Order	Gaussian Smoothing	LS of Degree 1	LS of Degree 3
Coarsest mesh	444	444	444	444
Coefficients	0	0	168	336
Differences	22735	46444	22627	22443
Total (bytes)	23199	46908	23259	23247
Rate (bits/vertex)	14.33	28.98	14.37	14.36

Table 9: Compression ratio results for several filtering schemes applied to the skateboard rendered in Figure 9.

Filter's Degree	1	2	3	4	5	6	7
bits/vertex	14.37	14.32	14.36	14.45	14.48	14.54	16.01

Table 10: Compression ratios for different filter lengths, in power basis, for the round table.

	Zeroth Order Filter	Gaussian Smoothing	LS Filter of Degree 1	LS Filter of Degree 3
Coarsest mesh	444	444	444	444
Coefficients	0	0	168	336
Differences	28931	27082	26542	26436
Total (bytes)	29397	27549	27184	27250
Rate (bits/vertex)	18.16	17.02	16.80	16.84
$l^2\ \mathrm{error}/\mathrm{vertex}(\cdot 10^{-4})$	43.87	51.83	38.82	36.12

Table 11: Compression ratios in the Single Resolution implementation of the Variable Length Encoding Algorithm applied to the skateboard rendered in Figure 9.

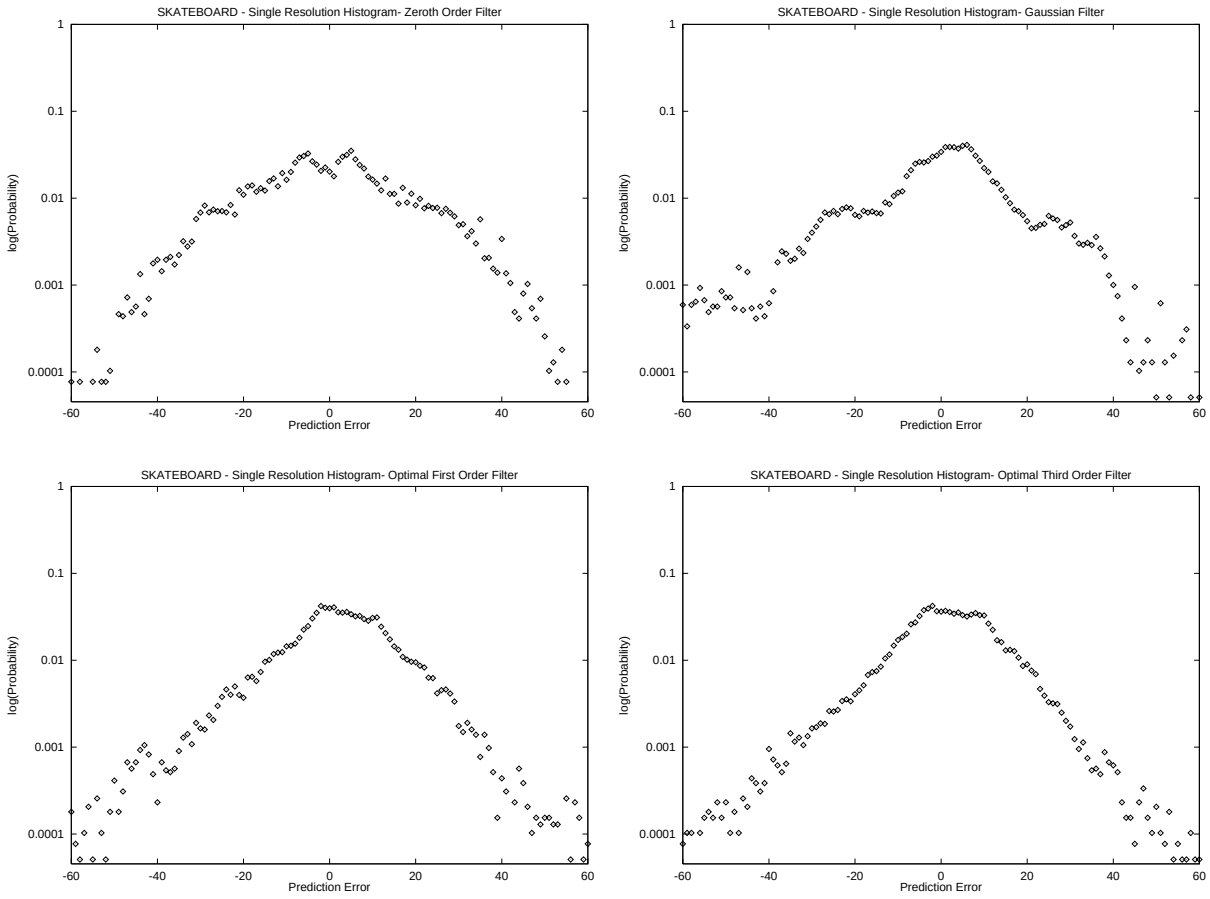


Figure 11: Semilog histograms of the prediction errors associated to the four filters for the Single Resolution scheme applied to the mesh rendered in Figure 9.

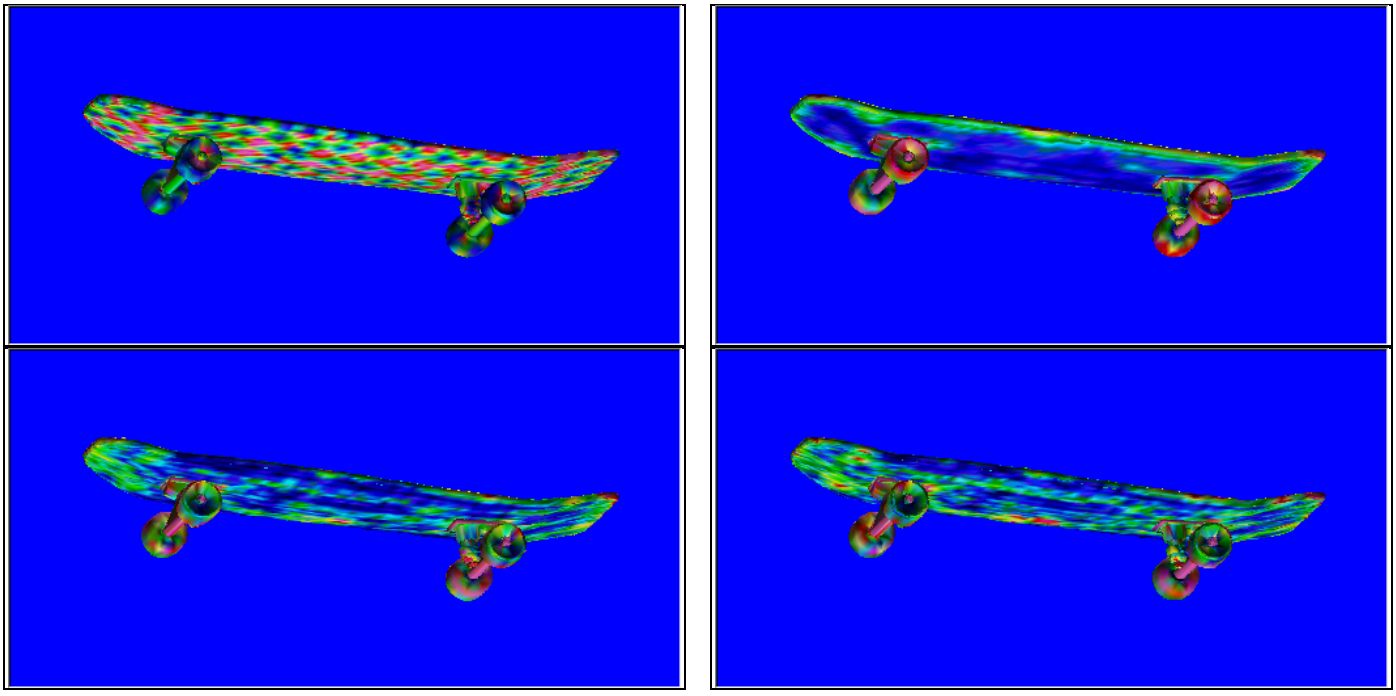


Figure 12: The color plot of the single resolution approximation errors for the four different filters: Zeroth Order Filter (upper left), Gaussian Smoothing (upper right), LS first order (lower left) and LS third order (lower right).

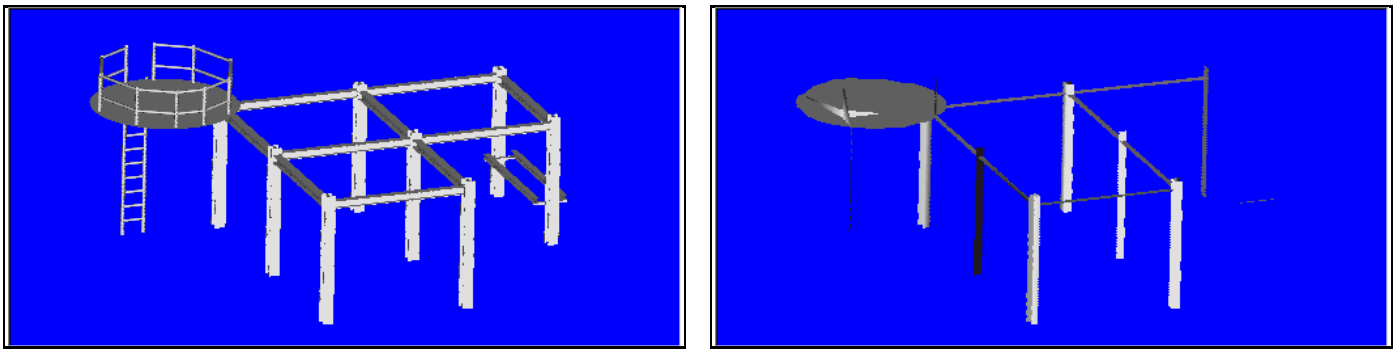


Figure 13: The piping mesh at: the finest resolution (left) and the coarsest resolution (right) after 7 levels of reduction.

Filter	Zeroth Order	Gaussian Smoothing	LS of Degree 1	LS of Degree 3
Coarsest mesh	522	522	522	522
Coefficients	0	0	144	288
Differences	2605	31595	2625	2673
Total (bytes)	3146	32136	3311	3503
Rate (bits/vertex)	1.38	14.17	1.46	1.54

Table 12: Compression ratio results for several filtering schemes applied to the piping construction mesh rendered in Figure 13.

Filter's Degree	1	2	3	4	5	6	7
bits/vertex	1.46	1.50	1.54	1.58	1.61	1.65	1.69

Table 13: Compression ratios for different filter lengths, in power basis, for the piping construction.

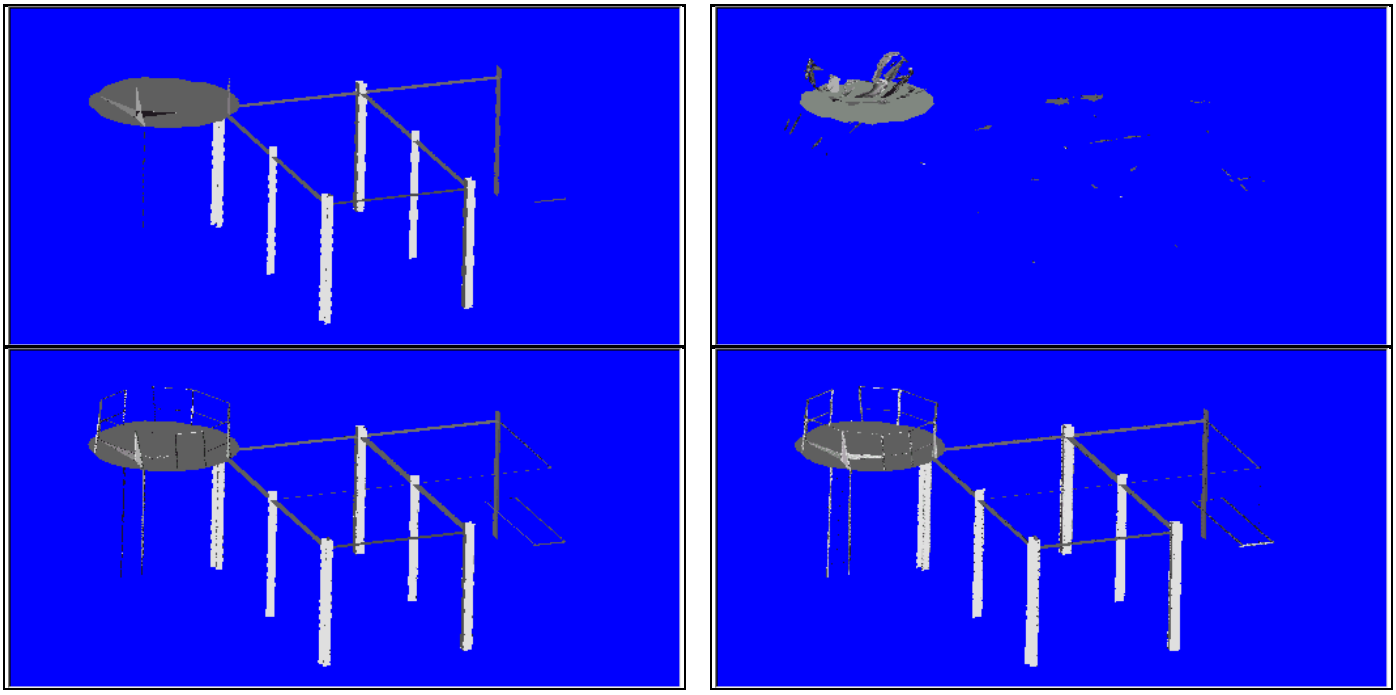


Figure 14: The piping mesh at the finest resolution level when no difference is used and the filtering is performed by: the Zeroth Order Filter (top left), the Gaussian Smoother (top right), the least squares filter of order 1 (bottom left) and the least squares filter of order 3 (bottom right).

Filter	Zeroth Order	Gaussian Smoothing	LS of Degree 1	LS of Degree 3
Coarsest mesh	522	522	522	522
Coefficients	0	0	168	336
Differences	19160	41278	21009	21573
Total (bytes)	19704	41823	21701	22458
Rate (bits/vertex)	8.69	18.44	9.57	9.90
$l^2\ \mathrm{error/vertex}(\cdot 10^{-2})$	1.21	22.96	1.20	1.20

Table 14: Compression ratios in the Single Resolution implementation of the Variable Length Encoding Algorithm applied to the piping construction rendered in Figure 13.

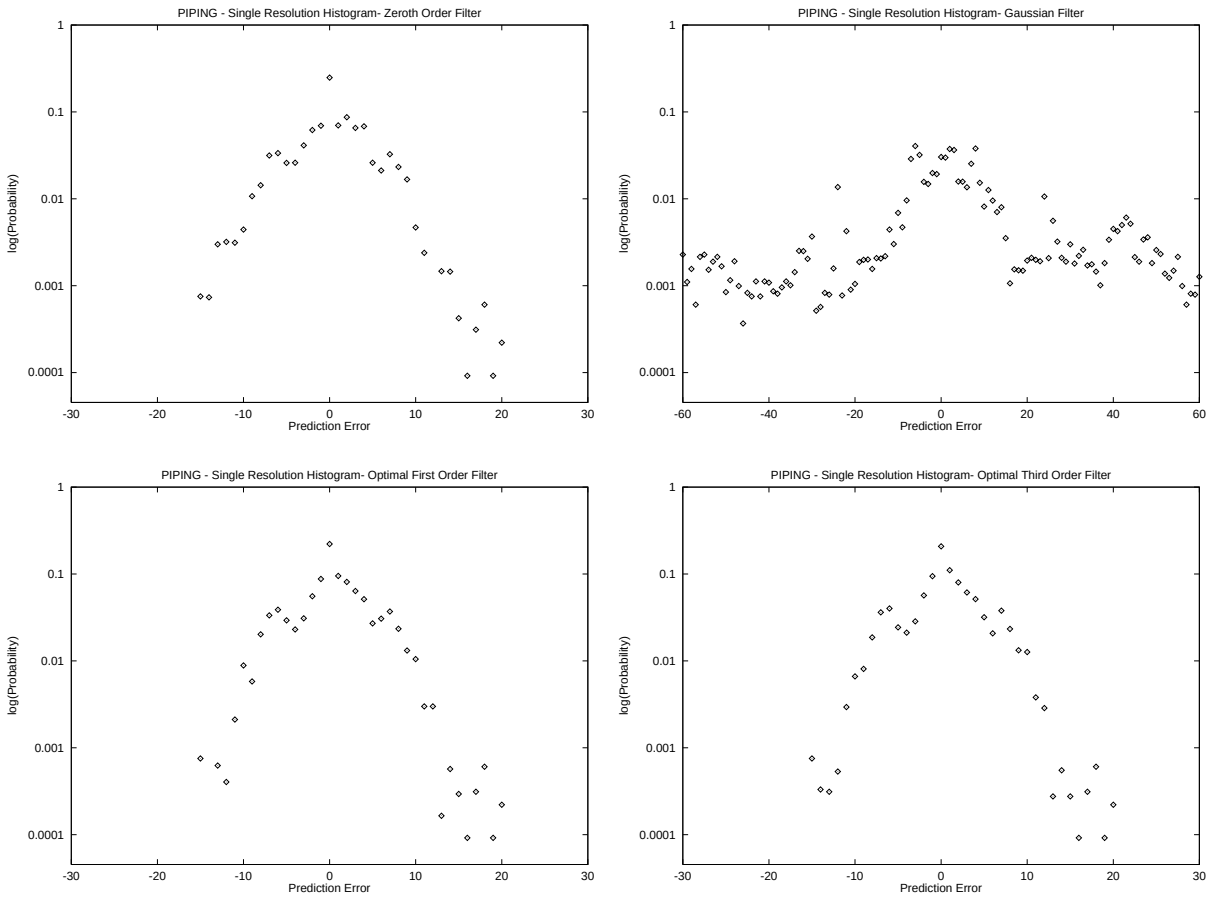


Figure 15: Semilog histograms of the prediction errors associated to the four filters for the Single Resolution scheme applied to the mesh rendered in Figure 13.

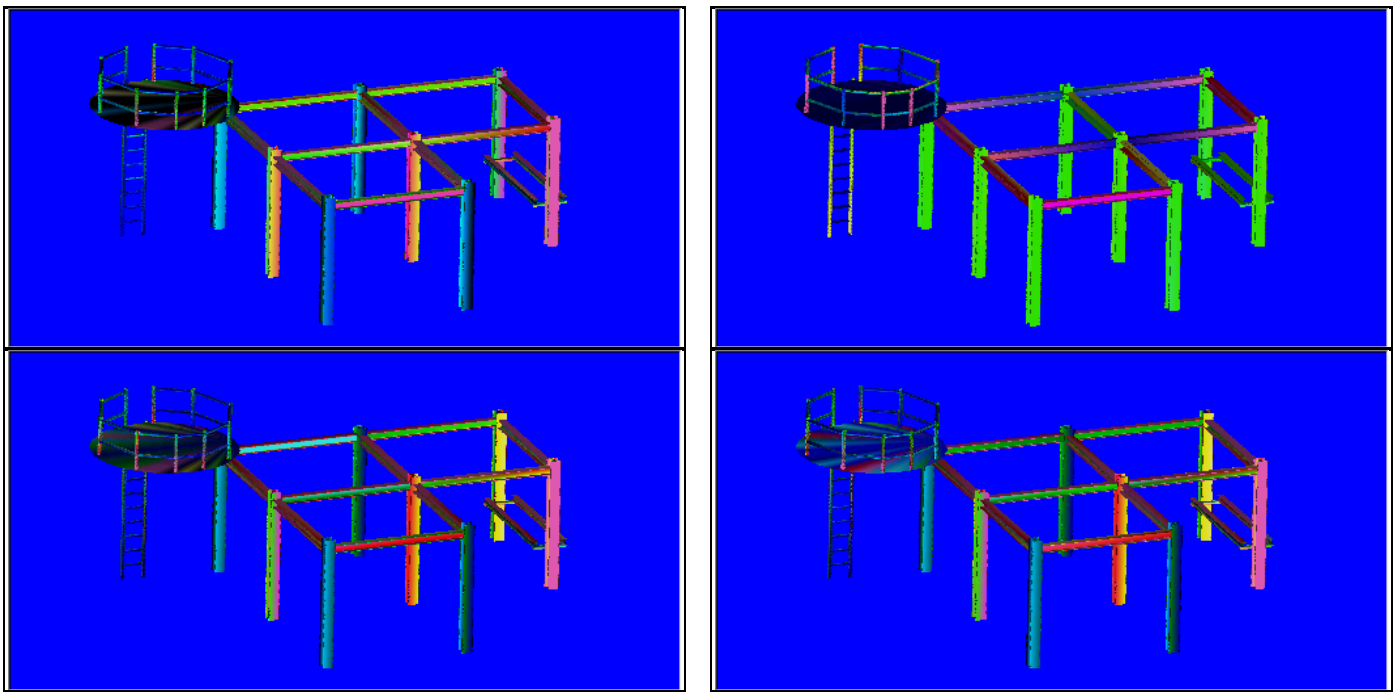


Figure 16: The color plot of the single resolution approximation errors for the four different filters: Zeroth Order Filter (upper left), Gaussian Smoothing (upper right), LS first order (lower left) and LS third order (lower right).

The last mesh we discuss is somewhat different to the other. It is a sphere of $nV_0 = 10242$ vertices and 20480 faces that reduces after 4 levels to $nV_3 = 224$ vertices. The striking difference is the compression ratio of the Zeroth Order filter: it is the worst of all the filters we checked. Even the Gaussian filter fares better than this filter. Snapshots of the approximated meshes are pictured in Figures 17-18. The mesh used is non-manifold but this is not a problem for the geometry encoder. The histograms shown in Figure 19 are in accordance with the rate results presented in Table 17: the narrower the distribution the better the rate. Note also how well a power-type low fits the 3^{rd} order filtered distribution.

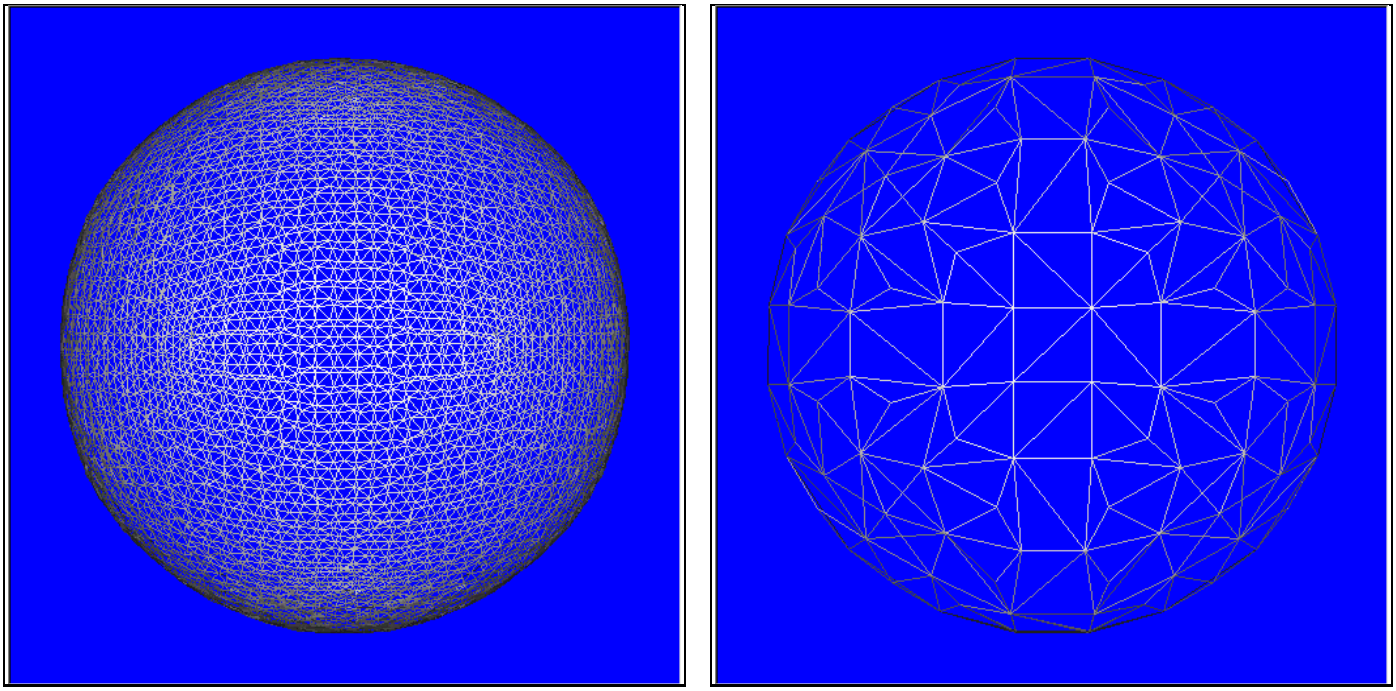


Figure 17: The sphere at: the finest resolution (left) and the coarsest resolution (right) after 4 levels of reduction.

Filter	Zeroth Order	Gaussian Smoothing	LS of Degree 1	LS of Degree 3
Coarsest mesh	881	881	881	881
Coefficients	0	0	72	144
Differences	29770	22614	19762	13395
Total (bytes)	30673	23518	20738	14440
Rate (bits/vertex)	23.96	18.37	16.20	11.28

Table 15: Compression ratio results for several filtering schemes applied to the sphere rendered in Figure 17.

Filter's Degree	1	2	3	4	5	6	7
bits/vertex	16.20	12.90	11.28	10.59	10.36	10.42	10.69

Table 16: Compression ratios for different filter lengths, in power basis, for sphere.

These examples show that in terms of compression ratio, the Zeroth Order Filter compresses best the irregular and less smooth meshes, whereas higher order filter are better for smoother and more regular meshes. However, in terms of accuracy and robustness, the higher order filters perform much better than its main "competitor", the Zeroth Order Filter. Note, except for highly regular meshes (like sphere, for instance), relatively low order filters are optimal. The range $[1..5]$ seems enough for most of the encoding schemes.

6 Conclusions

In this paper we study the 3D geometry filtering using the discrete Laplace operator. We next apply the filtering technique to Multi Resolution Analysis where the original mesh is converted into a sequence of successive refinements. Based on the

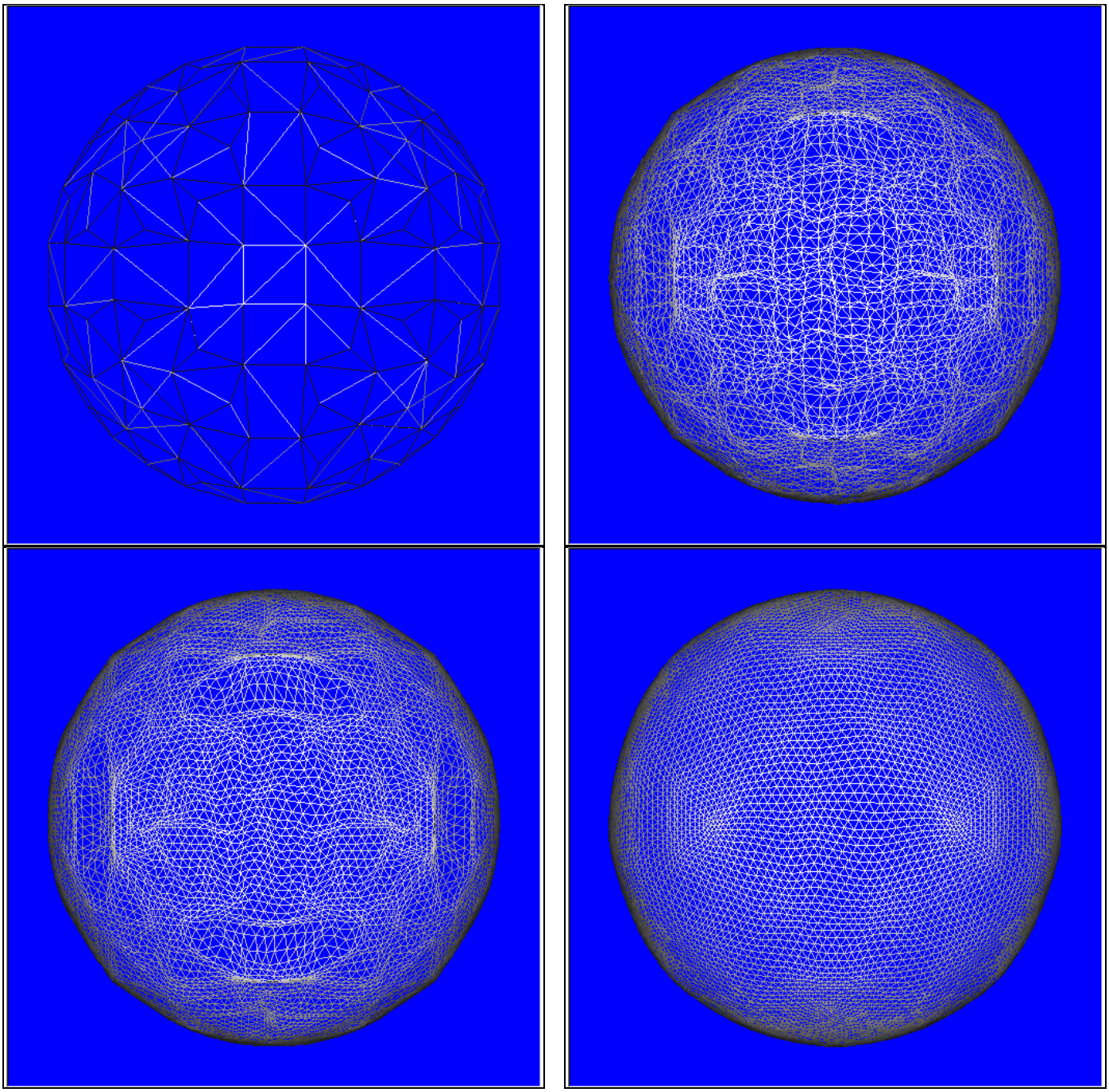


Figure 18: The sphere at the finest resolution level when no difference is used and the filtering is performed by: the Zeroth Order Filter (top left), the Gaussian Smoother (top right), the least squares filter of order 1 (bottom left) and the least squares filter of order 3 (bottom right).

Filter	Zeroth Order	Gaussian Smoothing	LS of Degree 1	LS of Degree 3
Coarsest mesh	881	881	881	881
Coefficients	0	0	168	336
Differences	28904	22152	20904	17920
Total (bytes)	29806	23055	21885	18971
Rate (bits/vertex)	23.28	18.00	17.09	14.82
$l^2\ \mathrm{error}/\mathrm{vertex}(\cdot 10^{-4})$	6.85	2.36	1.81	1.11

Table 17: Compression ratios in the Single Resolution implementation of the Variable Length Encoding Algorithm applied to the sphere rendered in Figure 17.

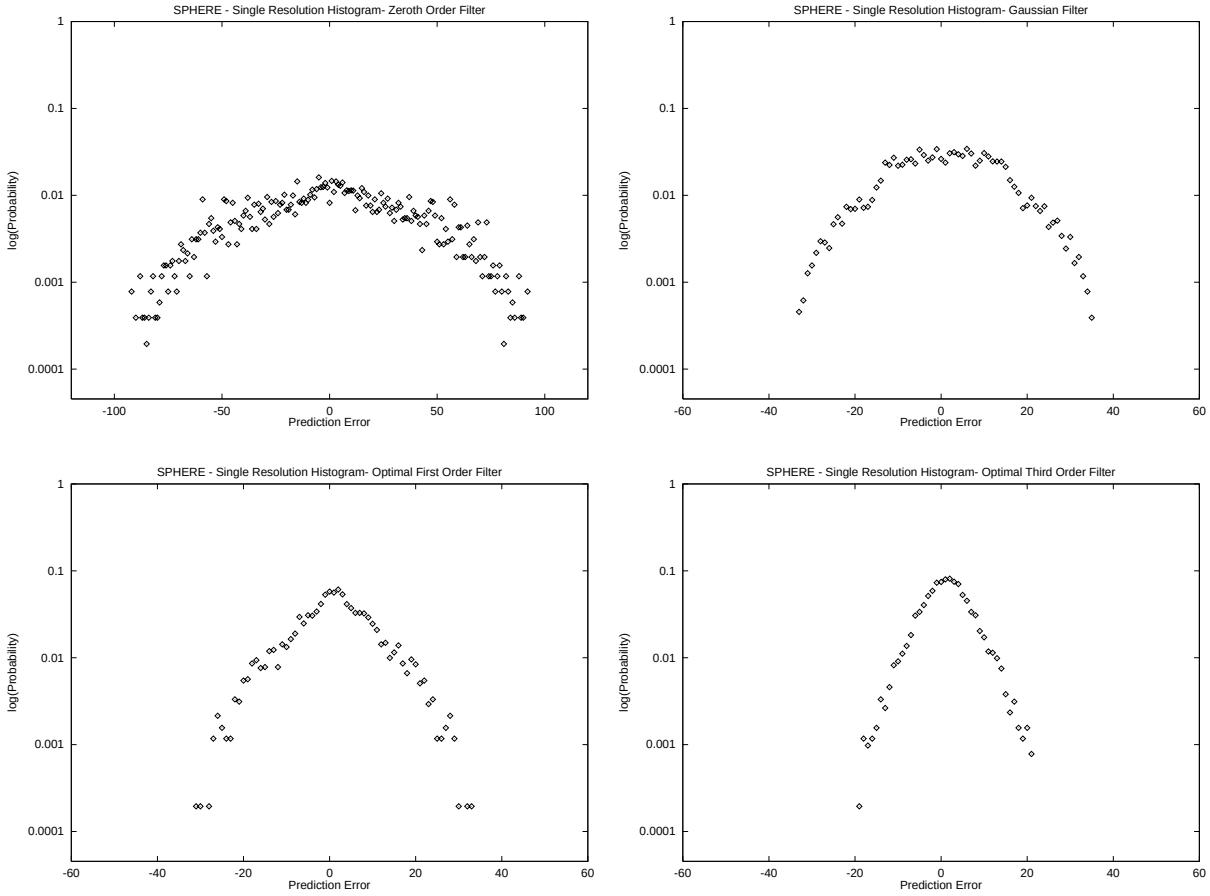


Figure 19: Semilog histograms of the prediction errors associated to the four filters for the Single Resolution scheme applied to the sphere mesh rendered in Figure 17.

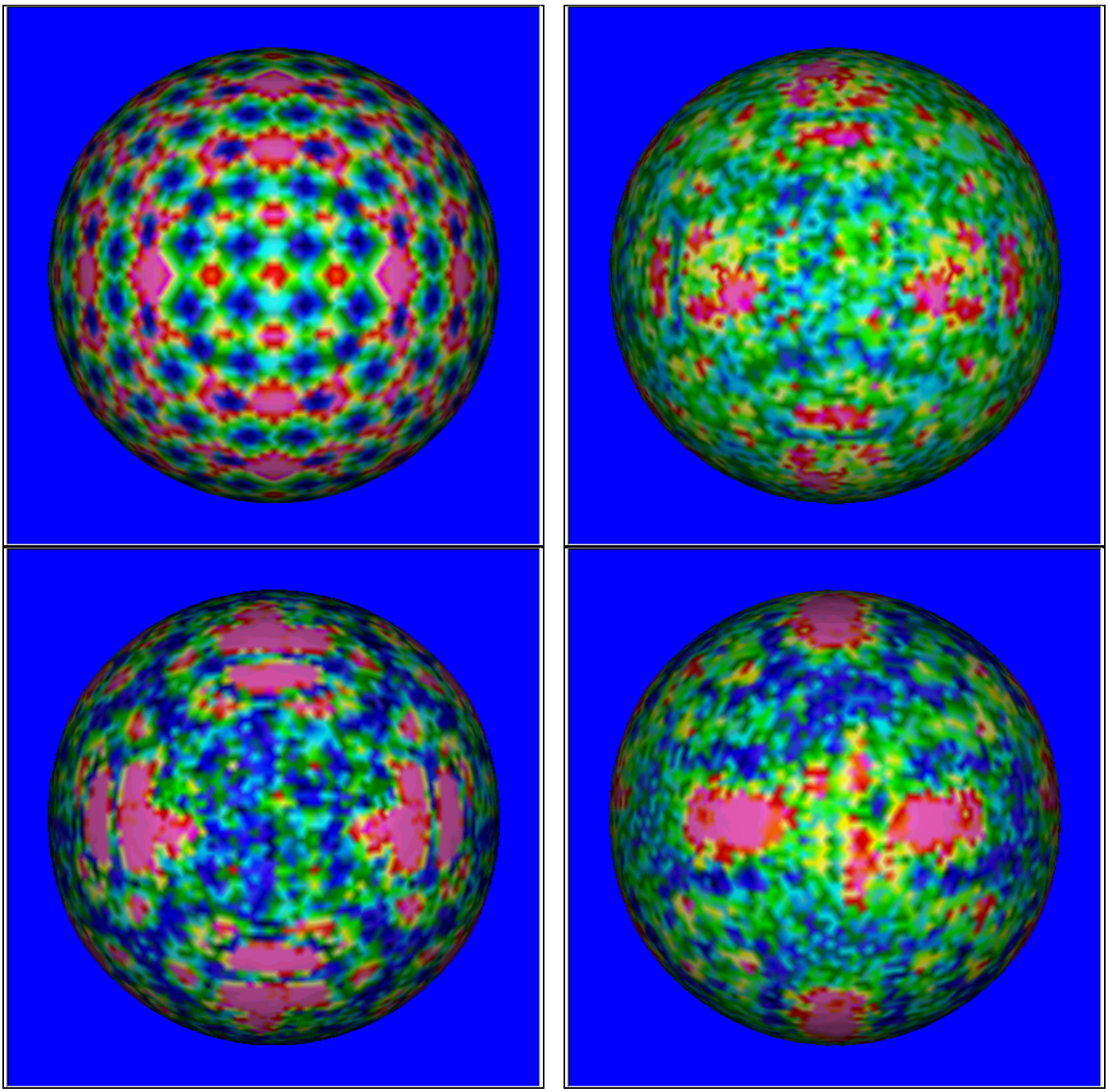


Figure 20: The color plot of the single resolution approximation errors for the four different filters: Zeroth Order Filter (upper left), Gaussian Smoothing (upper right), LS first order (lower left) and LS third order (lower right).

coarser resolution mesh, the finer resolution mesh is predicted using an extension map followed by filtering. At each level, the coordinate vectors are filtered separately using different filters. These filters are optimizers of some prediction error norm. Thus the geometry of a sequence of successively refined meshes is encoded in the following format: first the coarsest resolution mesh geometry and next for each successive level, the filters coefficients and prediction errors. The connectivity information is supposed known at each level separately.

Next we study several desirable properties of any encoding scheme, finding for each one the appropriate criterion to be optimized. Thus for a better accuracy of the predicted mesh when no difference is available, the filter coefficients should minimize the l^∞-norm of the prediction errors. For robustness, as understood in signal processing theory, the filters should minimize the l^2-norm of the differences. The third property, the compression rate, is maximized when the l^∞-norm is replaced by a l^α-norm with α usually between 1 and 2, depending on the prediction error's p.d.f. Thus, if the differences are Laplace distributed, the l^1-norm should be minimized, whereas if they are Gaussian, then the l^2-norm should be used. In any case, each of the three extreme cases (l^∞, l^2 or l^1) can be solved exactly. The l^2-norm case is the simplest and reltively computational inexpensive, and is solved by a linear system. The other two cases turn into linear programming problems which are very computational expensive to solve.

These theoretical results are next applied to concrete examples. In general for large, non-smooth and irregular meshes the Zeroth Order Filtering scheme yields the best compression ratio, but the poorest accuracy or, for the same reason, robustness. Instead, by paying a small price in the compression ratio, a least square filter give a better rendering accuracy and superior robustness. At the other end of the scale, for very smooth and regular meshes, the Gaussian filter (which in general behaves very poorly) gives a better compression ratio than the Zeroth Order filter.

The Basic Encoding Algorithm can be modified to allow a variable structure. The user can choose for what levels the differences are encoded and, by choosing a limit case, only the highest resolution level errors are encoded. Thus the MRA scheme becomes a Single Resolution encoding scheme. Examples in terms of accuracy and compression ratio are shown in the Examples section.

The novelty of this study consists in using linear filter in Multi Resolution encoding schemes and finding appropriate optimization criteria for specific compression or rendering properties. We hope this compression scheme will prove effective in Progressive Transmission protocols as MPEG4.

References

[Al&all88] H.Alt, K.Mehlhorn, H.Wagener, E.Welzl, *Congruence, similarity and symmetries of geometric objects*, Discrete Comp.Geom. **3**, 237–256 (1988)

[DaGr76] *Data Compression*, Ed. by L.D.Davisson and R.M.Gray, Dowden, Hutchinson & Ross Inc. (1976)

[DeMeScBa99] M.Desbrun, M.Meyer, P.Schröder, A.H.Barr, *Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow*, SIGGRAPH 1999 (to appear)

[Eck&all95] M.Eck, T.DeRose, T.Duchamp, H.Hope, M.Lounsbery, W.Stuetzle, *Multiresolution Analysis of Arbitrary Meshes*, SIGGRAPH Conf.Proc. 1995, 173–182

[FrInSp79] S.H.Friedberg, A.J.Insel, L.E.Spencer, *Linear Algebra*, Prentice-Hall 1979

[GuSt98] S.Gumhold, W.Strasser, *Real Time Compression of Triangle Mesh Connectivity*, SIGGRAPH 1998, 133–140

[GuSwSc99] I.Guskov, W.Sweldens, P.Schröder, *Multiresolution Signal Processing for Meshes*, SIGGRAPH 1999,

[Hart98] E.Hartmann, *A marching method for the triangulation of surfaces*, Visual Computer **14**, 95–108 (1998)

[Hoppe96] H.Hoppe, *Progressive Meshes*, SIGGRAPH 1996, 99–108

[LiKuo98] J.Li, C.-C.J.Kuo, *Progressive Coding of 3-D Graphics Models*, Proceedings of IEEE (Special Issue on Multimedia Signal Processing) **86**, no.6 June 1998, 1052–1063

[Lee&all98] A.W.F.Lee, W.Sweldens, P.Schröder, L.Cowsar, D.Dobkin, *MAPS: Multiresolution Adaptive Parametrization of Surfaces*, SIGGRAPH Proceedings, 1998, 95–104

[MaYaVe93] J.Maillot, H.Yahia, A.Verroust, *Interactive Texture Mapping*, SIGGRAPH Proceedings, 1993, 27–34

[PaRo99] P.Pajarola, J.Rossignac, *Compressed Progressive Meshes*, Technical Report GIT-GVU-99-05, GVU Center, Georgia Institute of Technology, 1999

[PeMi93] B.P.Pennebaker, J.L.Mitchell, *JPEG, Still Image Compression Standard*, Von Nostrand Reinhold, 1993

[PoHo97] J.Popocić, H.Hoppe, *Progressive Simplicial Complexes*, SIGGRAPH 1997, 217–224

[Ross99] J.Rossignac, *Edgebreaker: Connectivity Compression for Triangle Meshes*, IEEE Trans.on Vis.Comp.Graph. **5**, no.1 (1999), 47–61

[Taubin95] G.Taubin, *A Signal Processing Approach to Fair Surface Design*, Computer Graphics Proc., Annual Conference Series 1995, 351–358

[TaZhGo96] G.Taubin, T.Zhang, G.Golub, *Optimal Surface Smoothing as Filter Design*, IBM Research report RC-20404 1996

[TaGuHoLa98] G.Taubin, A.Guézier, W.Horn, F.Lazarus, *Progressive Forest Split Compression*, SIGGRAPH Proceedings , 1998,

[TaHoBo99] G.Taubin, W.Horn, P.Borrel, *Compression and Transmission of Multi-Resolution Clustered Meshes*, IBM Research report RC-21398, February 1999

[TaRo98] G.Taubin, J.Rossignac, *Geometry Compression through Topological Surgery*, ACM Transaction on Graphics **17**, no.2 April 1998, 84–115

[ZiTr90] R.E.Ziemer, W.H.Tranter, *Principles of Communications - System, Modulation, and Noise*, Houghton Mifflin Comp. 1990

Geometric Signal Processing on Polygonal Meshes

by G. Taubin

Eurographics 2000 State of The Art Report (STAR)

Geometric Signal Processing on Polygonal Meshes

G. Taubin†

IBM T.J. Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598
http://www.research.ibm.com/people/t/taubin

Abstract

Very large polygonal models, which are used in more and more graphics applications today, are routinely generated by a variety of methods such as surface reconstruction algorithms from 3D scanned data, isosurface construction algorithms from volumetric data, and photogrametric methods from aerial photography. In this report we provide an overview of several closely related methods developed during the last few yers, to smooth, denoise, edit, compress, transmit, and animate very large polygonal models.

1. Introduction

The geometric signal processing approach was originally motivated by the problem of smoothing large irregular polygonal meshes of arbitrary topology [36], such as those extracted from volumetric medical data by iso-surface construction algorithms, or constructed by integration of multiple range images, and the related problem of fair surface design. Because of the size of the typical data sets, only linear time and space algorithms can be considered, particularly for applications such as surface design and mesh editing, where interactive rates are a primary concern. This constraint on the complexity of the algorithms discards most early algorithms based on fairness norm optimization [42, 28, 13, 43], parametric [31, 26, 11, 25, 24] and implicit [1, 27] patch technology, physics-based deformable models [20, 41, 33, 30], and variational formulations [5, 28, 43, 13]. In these approaches, the problem is often reduced to the solution of a large sparse linear system, or a more expensive global optimization problem. Large sparse linear systems are solved using iterative methods [10], and usually result in quadratic time complexity algorithms. However, more recent work formulations have shown efficient solutions to the variational formulation based on multigrid algorithms [21, 22], and stable implicit sparse solvers that are competitive when agressive smoothing is required [7].

Most smoothing algorithms move the vertices of the polygonal mesh without changing the connectivity of the faces. The smoothed mesh has exactly the same number of vertices and faces as the original one. The simplest smoothing algorithm that satisfies the linear complexity requirement is Laplacian smoothing, described in detail in section 2. Laplacian smoothing is an iterative process, where in each step every vertex of the mesh is moved to the barycenter of its neighbors.

The only problem with Laplacian smoothing is *shrinkage*. When a large number of Laplacian smoothing steps are iteratively performed, the shape undergoes significant deformations, eventually converging to the centroid of the original data. The algorithm introduced by Taubin [36] solves this problem and introduced the signal processing machinery necessary to analyze the behavior of these smoothing processes. This work was followed by a number of extensions [40, 7] and applications to interactive shape design [46, 23, 21, 44, 22, 12], 3D geometry compression [37, 2, 19], and shape reconstruction from multiple 3D scans [3].

Within the context of interactive shape design, Zorin [46] defines a multi-resolution subdivision structure over an irregular mesh, using the signal processing smoothing algorithms as the basis of his analysis process.

Guskov [12] follows a different signal processing approach over the Progressive Meshes [16] structure, where *frequency* has a completely different meaning. He is able to perform similar filtering operations, as with the methods described in this paper.

† On sabbatical from 08/01/2000 to 07/31/2001, Dept. of Electrical Engineering, California Institute of Technology Mail Code 136-93, Pasadena, CA 91125

In 3D geometry compression[38, 39], Taubin et.al. [37] use these signal processing smoothing algorithms to predict the position of high resolution vertices from their low resolution counterparts in their progressive transmission scheme. Balan and Taubin [2], study the problem of constructing optimal filters in this context. Karni and Gotsman [19] use the partial Fourier expansion applied to the vertices of a mesh partition to define a JPEG-like compression scheme for meshes.

In the area of shpe reconstruction from multiple 3D scans, Bernardini et.al. [3] define a *conforming* process to estimate the average shape of several overlaping meshes by allowing them to deform at very low frequency, while preserving the details. This process is based on applying a very aggressive smoothing filter to the deformation field that would make each vertex of each overlapping mesh move to the average position of vertices of other meshes in a neighborhood.

The paper is organized as follows. In section 2 we introduce Laplacian smoothing withing the context of meshes. In section 3 we show how Fourier Analysis can be performed on signals defined on meshes and graphs. In section 4 we discuss methods to smooth or denoise signals defined on meshes and graphs as low-pass filtering. In section 5 we describe Taubin's $\lambda|\mu$ algorithm. In section 6 we discuss how edge weights can be manipulated to compensate for irregular edge lengths and face angles. In section 7 we show that classic filter design methods can be used to construct faster smoothing algorithms, and other feature enhancing filters. In section 8 we discuss how different constraints can be imposed to the smoothing algorithms and their relation to interactive shape design. Finally, in section 9 we present our conclusions.

2. Laplacian Smoothing

Laplacian smoothing is a well established technique to improve the geometric irregularity of a 2D mesh in the field of finite-elements meshing [15]. In this context, boundary vertices of the mesh are constrained not to move, but internal vertices are simultaneously moved to the barycenter of its neighboring vertices. And then the process is iterated a number of times.

When Laplacian smoothing is applied to a noisy 3D polygonal mesh without constraints, noise is removed, but significant shape distortion may be introduced. The main problem is that Laplacian smoothing produces *shrinkage*, because in the limit, all the vertices of the mesh converge to their barycenter.

To understand why the Laplacian smoothing algorithm removes high frequency noise, why it produces shrinkage, and how to solve the shrinkage problem, we need to develop the basic concepts of signal processing on meshes, or more generally, on graphs.

3. Fourier Analysis on Meshes and Graphs

A graph $G = (V, E)$, composed of a set of n vertices V, and a set of edges E can be directed or undirected. The undirected graph of a Mesh M is composed of the set of mesh vertices and the set of mesh edges as unordered pairs. In the directed case, where the edges of G are ordered pairs of vertices, every edge of M corresponds to two oriented edges of G.

We look at the vertices of M as a three-dimensional *graph signal* $v = (v_1, \ldots, v_n)^t$ defined on G. In general, a d-dimensional graph signal on a graph G is a $d \times n$ matrix $x = (x_1, \ldots, x_n)^t$, where each row of x is regarded as the signal value at the i-th. vertex of the graph.

A *neighborhood* or *star* of a vertex index i in the graph G is the set $i^\star$ of vertex indices j connected to i by an edge (i, j).

$$i^\star = \{ j : (i, j) \in E \} .$$

If the index j belongs to the neighborhood $i^\star$, we say that j is a *neighbor* of i. The neighborhood structure of an undirected graph, such as the graph of a mesh defined above, are symmetric. That is, every time that a vertex j is a neighbor of vertex i, also i is a neighbor of j. With non-symmetric neighborhoods, which are associated with directed graphs, certain constraints can be imposed. We discuss this issue in detail in section 8.

The set of displacements Δv_i produced by the Laplacian smoothing step that moves each vertex to the barycenter of its neighbors can be described as the result of applying the Laplacian operator to the vertices of the mesh.

The Laplacian operator is defined on a graph signal x by weighted averages over the neighborhoods

$$\Delta x_i = \sum_{j \in i^\star} w_{ij} (x_j - x_i) , \qquad (1)$$

where the weights w_{ij} are non-negative numbers that add up to one for each vertex star

$$\sum_{j \in i^\star} w_{ij} = 1 . \qquad (2)$$

Since the Laplacian operator $x \to \Delta x$ is linear on the space of graph signals defined on G, and operates on the coordinates of x independently, it is sufficient to consider the case of one-dimensional graph signals.

In section 6 we discuss in detail different ways of choosing weights. For the time being, lets assume that the edge weights are determined by first choosing an edge cost $c_{ij} = c_{ji} \geq 0$ for each graph edge, and then setting $w_{ij} = c_{ij}/c_i$, where c_i is the average cost of edges incident to i

$$c_i = \sum_{j \in i^\star} c_{ij} > 0 .$$

For example, if all the edges have unit cost $c_{ij} = 1$, then for each neighbor j of i, the weight w_{ij} is equal to the inverse of the number of neighbors $1/|i^\star|$ of v. We organize the edge

costs and weights as matrices $C = (c_{ij})$, $W = (w_{ij})$, with elements equal to zero if j is not a neighbor of i. We also assume that once set, the weights are kept constant during the iterative smoothing process. We will relax this asumption in section 6.

This choice of weights is independent of the vertex positions, or *geometry*, of the mesh, and only function of the structure of the graph G, i.e. the *connectivity* of the mesh. Note that as a result of the neighborhood normalization constraint of equation 2, although the $n \times n$ matrix of edge costs C is symmetric, in general the matrix of edge weigths W is not. We consider edge weights that are function of the geometry in section 6.

If we define the matrix $K = I - W$, with I the identity matrix, the Laplacian operator applied to a graph signal x can be written in matrix form as follows

$$\Delta x = -K x \,. \tag{3}$$

For undirected graphs and the choice of weights described above, the matrix K has real eigenvalues $0 \le k_1 \le k_2 \le \cdots \le k_n \le 2$ with corresponding linearly independent real unit length right eigenvectors $e^1, \ldots, e^n$ [36]. In matrix form

$$K\mathbf{E} = \mathbf{E}\,\mathrm{diag}(k) \,, \tag{4}$$

with $\mathbf{E} = (e^1, \ldots, e^n)$, $k = (k_1, \ldots, k_n)^t$, and $\mathrm{diag}(k)$ the diagonal matrix with k_i in its i-th. diagonal position. Seen as one-dimensional graph signals, these eigenvectors can be considered as the *natural vibration modes* of the graph, and the corresponding eigenvalues as the associated *natural frequencies*.

Since $e^1, \ldots, e^n$ form a basis of n-dimensional space, every graph signal x can be written as a linear combination

$$x = \sum_{j=1}^{n} \hat{x}_j \, e^j = \mathbf{E}\,\hat{x} \,. \tag{5}$$

The vector of coefficients $\hat{x}$ is the Discrete Fourier Transform (DFT) of x, and $\mathbf{E}$ is the Fourier Matrix.

If instead of being derived from the vertices and edges of a mesh, the graph G is a closed polygonal curve with n vertices and edges, i.e., a cycle, we are in the classical case of discrete-time n-periodic signals.

Fourier analysis is a natural tool to solve the problem of signal smoothing. The space of signals is decomposed into orthogonal subspaces associated with different frequencies, with the low frequency content of a signal regarded as subjacent data, and the high frequency content as noise. To denoise a signal it is sufficient to compute its DFT, discard its high frequency coefficients, and compute the linear combination of remaning terms as the result. This is exatly what the method of *Fourier descriptors*[45] does to smooth a closed curve.

In the case of closed polygonal curves the DFT of a signal x can be computed very efficiently using the Fast Fourier Transform (FFT) algorithm[32], and the eigenvalues and eigenvectors of K can be computed analytically. In general, the matrix K is large, and although sparse, it is almost impossible to reliably compute its eigenvalues and eigenvectors. This makes it impractical to smooth vertex positions of large meshes with the Fourier descriptors method.

Note that even using the FFT algorithm in the closed polygonal curve case, the computational complexity is $O(n\log(n))$, i.e., not linear.

```
Laplacian(G,W,x)
  new Δx = 0;
  for(e = (i, j) ∈ E)
    Δx_i = x_i + w_ij(x_i − x_j);
  end;
return Δx;
```

Figure 1: *Algorithm to evaluate the Laplacian operator. $G = (V,E)$ directed graph, W matrix of weights defined on the edges of G, x input signal on G, Δx output signal.*

```
LaplacianSmoothing(G,W,N,λ,x)
  new Δx
  for(i = 0; i < N; i = i + 1)
    Δx = Laplacian(G,W,x);
    x = x + λΔx;
  end;
return;
```

Figure 2: *The Laplacian Smoothing Algorithm. G graph, W matrix of weights defined on the edges of G, N number of iterations, λ scaling factor, x signal on G to be smoothd.*

4. Smoothing as Low Pass Filtering

Figure 4 describes the algorithm to evaluate the Laplacian operator on a signal x defined on a directed graph G, with given weight matrix W. And figure 4 describes the Laplacian smoothing algorithm, with a scaling factor $0 < \lambda < 1$ which is used to control the speed of the diffusion process. With this parameter, one step of the Laplacian smoothing algorithm can be described in matrix form as follows

$$x^1 = x + \lambda\Delta x = (I - \lambda K)\,x = f(K)\,x \,, \tag{6}$$

where $f(K)$ is a matrix obtained by evaluating the univariate polynomial $f(k) = 1 - \lambda k$ in the matrix K. If the process is iterated N times, the output can still be expressed as $x^N = f(K)\,x$, but with a different univariate polynomial $f(k) = (1 - \lambda k)^N$.

A *Linear Filter* is defined by a univariate function $f(k)$ that can be evaluated on the square matrix K to produce another matrix of the same size. Although many functions of one variable can be evaluated in matrices [10], in this section

```
TaubinSmoothing(G, W, N, λ, μ, x)
  new Δx
  for(i = 0; i < N; i = i + 1)
    Δx = Laplacian(G, W, x);
    if i is even
      x = x + λΔx;
    else
      x = x + μΔx;
  end;
return;
```

Figure 3: *The Taubin Smoothing Algorithm. G graph, W matrix of weights defined on the edges of G, N number of iterations, λ and μ scaling factors, x signal on G to be smoothd.*

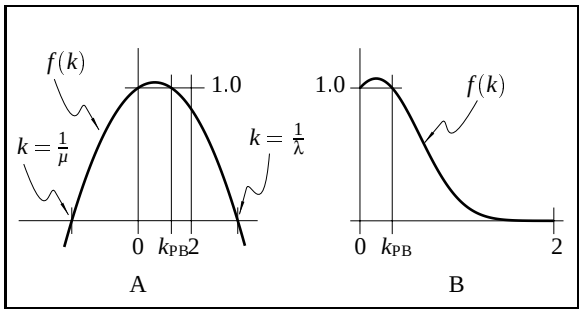


Figure 4: *Graph of transfer functions for the λ|μ algorithm. (A) $f(k) = (1-\mu k)(1-\lambda k)$. (B) $f(k) = ((1-\mu k)(1-\lambda k))^{N/2}$ with $N > 1$.*

we only consider polynomials. In section 7 we also consider rational functions. The function $f(k)$ is the *transfer function* of the filter. It is well known that for any of these functions, the matrix $f(K)$ has as eigenvectors the eigenvectors $e^1, \ldots, e^n$ of the matrix K, and as eigenvalues the result $f(k_1), \ldots, f(k_n)$ of evaluating the function on the eigenvalues of K. Since for any polynomial transfer function

$$x' = f(K)\,x = \sum_{i=1}^{n} f(k_i)\hat{x}_i\, e^i \,,$$

because $Ke^i = k_i e^i$, to define a low-pass filter we need to find a polynomial such that $f(k_i) \approx 1$ for low frequencies, and $f(k_i) \approx 0$ for high frequencies in the region of interest $k \in [0,2]$.

In the case of Laplacian smoothing, where the transfer function is $f(k) = (1-\lambda k)^N$, with $0 < \lambda < 1$, we see that for every $k \in (0,2]$, we have $(1-\lambda k)^N \to 0$when $N \to \infty$ because $|1-\lambda k| < 1$. This means that all the frequency components, other than the zero frequency component (the barycenter of all the vertices), are atenuated for large N. On the other hand, the neighborhood normalization constraint of equation 2 implies that the matrix K always has 0 as its first eigenvalue with associated eigenvector $(1, \ldots, 1)^t$, and the zero frequency component is preserved without changes because $f(0) = 1$ independently of the values of λ and N. In conclusion Laplacian smoothing filters out too many frequencies.

5. The λ|μ Algorithm

Taubin [36] proposed the following second degree transfer function to solve the problem of shrinkage

$$f(k) = (1-\lambda k)(1-\mu k)\,, \tag{7}$$

which can be implemented as two consecutive steps of Laplacian smoothing with different scaling factors; the first one with $\lambda > 0$, and the second one with $\mu < -\lambda < 0$. That is, after the Laplacian smoothing step with positive scale factor λ is performed (shrinking step), a second Laplacian smoothing step with negative scale factor μ is performed (unshrinking step). Figure 5 describes the algorithm.

The graph of the transfer function of equation (7) is illustrated in figure 4-A. Figure 4-B shows the resulting transfer function after N iterations of the algorithm. Since $f(0) = 1$ and $\mu + \lambda < 0$, there is a positive value of k, let us denote it k_{PB} (the *pass-band frequency*), such that $f(k_{PB}) = 1$. The value of k_{PB} is

$$k_{PB} = \frac{1}{\lambda} + \frac{1}{\mu} > 0\,. \tag{8}$$

The graph of the transfer function $f(k)$ shown in Figure 4-B displays a typical *low-pass filter* shape in the region of interest $k \in [0,2]$. The *pass-band region* extends from $k = 0$ to $k = k_{PB}$, where $f(k) \approx 1$. As k increases from $k = k_{PB}$ to $k = 2$, the transfer function decreases to zero. The faster the transfer function decreases in this region, the better. The rate of decrease is controlled by the number of iterations N.

For example, choosing λ so that $f(1) = -f(2)$, i.e.,

$$0 = f(1) + f(2) = 1 - 3(\lambda + \mu) + 5\lambda\mu\,, \tag{9}$$

ensures a stable and fast filter [40]. A typical value for k_{PB} is 0.1. The corresponding typical scaling factor values are then computed from equations 8 and 9.

Figures 5 and 6 show examples of large surfaces smoothed with this algorithm. Figures 5 is a synthetic example, where noise has been added to one half of a polyhedral approximation of a sphere. Note that while the algorithm progresses the half without noise does not change. Figure 6 was constructed from a CT scan of a spine. The boundary surface of the set of voxels with intensity value above a certain threshold is used as the input signal. Note that there is not much difference between the results after 50 and 100 iterations.

6. Weights

With *Equal weights*, determined by unit edge costs, very satisfactory results are obtained on meshes which display very

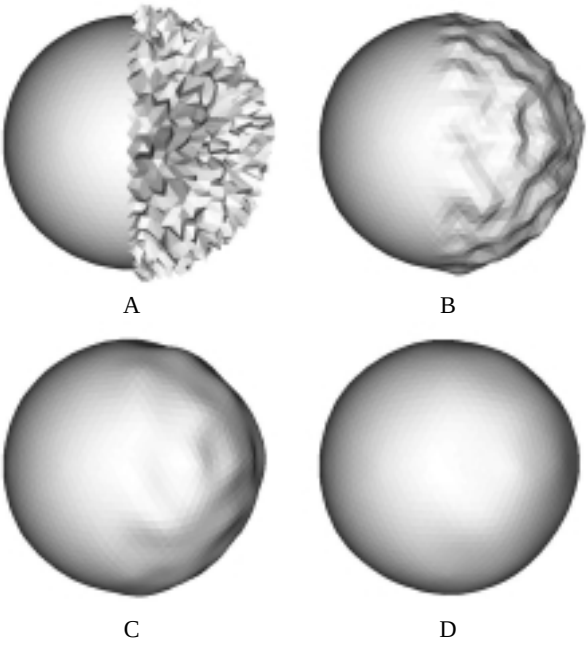


Figure 5: *(A) Sphere partially corrupted by normal noise. (B) Sphere (A) after 10 non-shrinking smoothing steps. (C) Sphere (A) after 50 non-shrinking smoothing steps. (D) Sphere (A) after 200 non-shrinking smoothing steps. Surfaces are flat-shaded to enhance the faceting effect.*

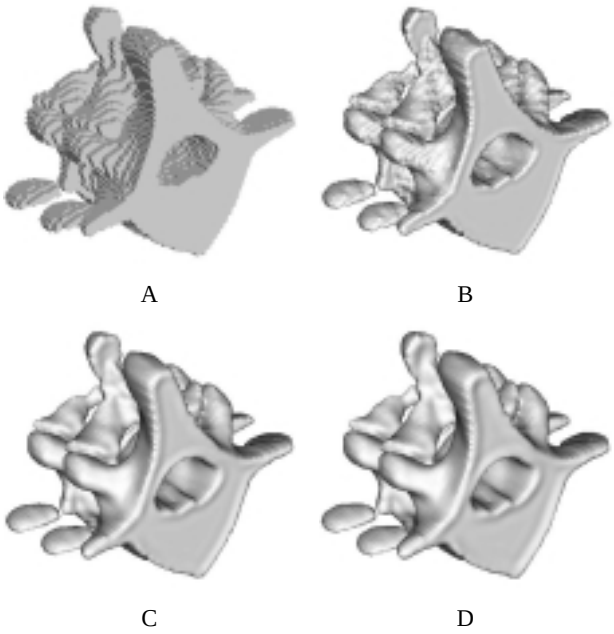


Figure 6: *(A) Boundary surface of voxels from a CT scan. (B) Surface (A) after 10 non-shrinking smoothing steps. (C) Surface (A) after 50 non-shrinking smoothing steps. (D) Surface (A) after 100 non-shrinking smoothing steps.* $k_{PB} = 0.1$ *and* $\lambda = 0.6307$ *in (B), (C), and (D). Surfaces are flat-shaded to enhance the faceting effect.*

small variation in edge length and face angles across the whole mesh, such as those shown in figures 5 and 6. When these assumptions are not met, local distortions are introduced. The edge weights can be used to compensate for the irregularities of the teselation, and produce results which are function of the local geometry of the signal, rather than the local parameterization.

Fujiwara weights try to compensate for irregular edge lengths by determining the edge costs as a function of the edge length $c_{ij} = \phi(\|v_j - v_i\|)$. For example, both Taubin [36] and Fujiwara [9] propose choosing the inverse of the edge length $\phi(t) = 1/t$ as the function, which makes the Laplacian operator independent of the edge lengths, and only dependent on the directions of the vectors pointing to the neighboring vertices. This weighting scheme does not solve the problems arising from unequal face angles.

Desbrun weights compensate not only for unequal edge lengths, but also for unequal face angles. Laplacian smoothing with equal edge costs tends to equalize the lengths of the edges, and so, tends to make the triangular faces equilateral. The vertex displacements produced by the Laplacian operator can be decomposed into a normal and a tangencial component. In some cases the edge equalization may be the desired effect. This is the case when mesh smoothing is used to improve the quality of finite-elements mesh. But in other cases, such as when a texture is mapped onto the mesh, having a non-zero tangencial component is undesirable. Based on a better approximation to the curvature normal, Desbrun[7] proposes the following choice of edge costs

$$c_{ij} = \cot\alpha_i j + \cot\beta_i j \,, \tag{10}$$

where $\alpha_i j$ and $\beta_i j$ are the two angles opposite to the edge $e = (i, j)$ in the two triangles having e in common. This choice of weights produces no tangencial drift when all the faces incident to the vertex are coplanar.

The three weighting schemes described in this section can be applied to both Laplacian smoothing and Taubin smoothing, but bot Fujiwara weights and Desbrun weights must be recomputed after each iteration, or after a small number of iterations. This makes the whole smoothing process a non-linear operation, and computationally more expensive.

An interactive implementation of these techniques is available as a Java applet[34]. Figure 7 shows a screen shot of this applet.

Guskov [12] proposed another weighting scheme based on divided differences, but applies to a smoothing process based on a second order neighborhood.

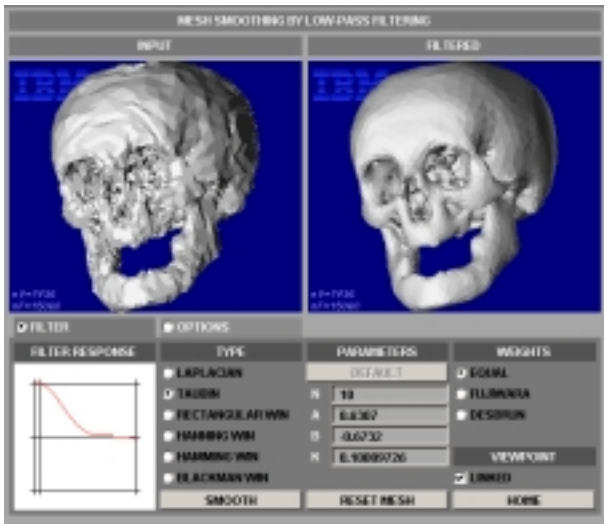


Figure 7: *Implementation of some of the techniques described in this paper as a Java applet* [34].

```
FirFilter(G,W,N, f,x)
  new x^0 = x
  new x^1 = Laplacian(G,W,x^0);
  new x^2 = x^0 − 0.5x^1
  new x = f_0 x^0 + f_1 x^1
  for(i = 2; i < N; i = i + 1)
    x^2 = Laplacian(G,W,x^1);
    x = x + f_i x^2;
    x^0 = x^1;
    x^1 = x^2;
  end;
return;
```

Figure 8: *The FIR Filter Algorithm of Taubin et.al.* [40]. *G graph, W matrix of weights defined on the edges of G, N number of iterations, $f = (f_0, \ldots, f_{N-1})$ polynomial coefficients in Chebyshev basis, x signal on G to be filtered.*

7. Fast Smoothing as Filter Design

In the $\lambda|\mu$ algorithm different combinations of the parameters λ, μ, and N produce almost identical transfer functions $f(k)$. For example if the scaling factors λ is reduced in magnitude, and then μ is recomputed to keep the pass-band frequency unchanged using equation 8, an equivalent result can be achieved with more iterations [40].

Taubin et.al. [40] showed how to efficiently implement any polynomial transfer function expressed as a linear combination of Chebyshev polynomials[6]. Figure 7 describes the algorithm. Chebyshev polynomials are numerically more stable than the power basis, and are defined by a three term recursion that results in an algorithm with low storage use

```
IirFilter(G,W,N_g,g,N_h,h,x)
  FirFilter(G,W,N_g,g,x)
  new x^1 = x;
  new H = h(K);
  solve Hx = x^1;
return;
```

Figure 9: *The IIR Filter Algorithm Taubin et.al.* [40]. *G graph, W matrix of weights defined on the edges of G, N number of iterations, $g = (g_0, \ldots, g_{N_g-1})$ and $h = (h_0, \ldots, h_{N_h-1})$ polynomial coefficients in Chebyshev basis, x signal on G to be filtered.*

and linear complexity

$$\begin{cases} T_0(w) &= 1 \\ T_1(w) &= w \\ T_j(w) &= 2wT_{j-1}(w) - T_{j-2}(w) \end{cases} \tag{11}$$

Since the domain of Chebyshev polynomials is $w \in [0,1]$, the following change of variable is necessary $w = 1 - k/2$.

The ability to efficiently implement any polynomial transfer function, reduces the problem of minimizing the number of iterations to a univariate polynomial approximation problem, i.e., to the classical problem of Finite Impulse Response (FIR) filter desgin in signal processing [29]. As an example, Taubin et.al. [40] showed how to design filters based on the classical Window-based method [14], but other polynomial approximation technique can be used to design stable FIR filters. For example, The Parks-McClellan algorithm [18] uses the Remez exchange algorithm and Chebyshev approximation theory to design filters with an optimal fit between the desired and actual frequency responses. The filters are optimal in the sense that the maximum error between the desired frequency response and the actual frequency response is minimized. Filters designed this way exhibit an equiripple behavior in their frequency responses and are sometimes called equiripple filters.

The only problem with FIR filters is that high degrees are usually needed to obtain a good approximations of ideal frequency responses with sharp transitions, such as low-pass filters with a narrow pass-band. Infinite Impulse Response filters (IIR), with rational transfer functions with polynomials of low degree, solve this problem. In our case, if the transfer function is a ratio of two polynomials $f(k) = g(k)/h(k)$, with $h(k) \neq 0$ for $k \in [0,2]$, filtering a signal x corresponds to solving the following system of equations

$$h(K)\,x' = g(K)\,x\,. \tag{12}$$

Evaluation of this filter can be performed in three steps. First, if $g(k)$ is not constant, the right hand side of this equation is evaluated with the FIR algorithm of Taubin et.al. $x^1 = g(K)x$. Then the the matrix $H = h(K)$ has to be constructed, and finally the linear system of equations $Hx = x^1$ is solved. Figure 7 describes this algorithm. In this context, IIR filters only

makes sense if the polynomial $h(k)$ is of very low degree, i.e., if the matrix H is sparse. Some sparse linear solvers only need the to evaluate the product of the matrix H by a vector. In that case the matrix H does not need to be constructed explicitly, and the FIR algorithm of Taubin et.al. can be used again to evaluate this filter as many times as necessary by the linear solver.

The Implicit Fairing method of Desbrun et.al. [7] is a particular case this type of filter. It corresponds to the classical Butterworth filter with transfer function

$$f(k) = \frac{1}{1+(k/k_{PB})^N} \cdot \qquad (13)$$

Desbrun et.al. development is based on a PDE formulation. They show that the Laplacian smoothing algorithm corresponds the solution of the diffusion process

$$\frac{\partial x}{\partial t} = \lambda dt \Delta x \,,$$

using the *forward Euler method*

$$x' = x + \lambda dt \Delta x = (I + \lambda dt \Delta) x \,,$$

with unit time step $dt = 1$. They use the *backward Euler method* instead, which requires the solution of the linear system

$$(I - \lambda dt \Delta) x' = x \,,$$

but is stable for arbitrary large time steps, as opposd to the explicit scheme which is stable only for $|\lambda dt| < 1$. Although having to solve a sparse linear system per step, as apposed to multiplying by a sparse matrix, seems to slow down the computation, they report computational time similar or better than the explicit method.

8. Constraints

The ability to impose constraints to the smoothing process, such as specifying the positions of some vertices, or normal vectors, specifying ridge curves, or the behavior of the smoothing process along the boundaries of the mesh, is needed in the context of free-form interactive shape design.

All the methods described so far allows the signals to freely evolve without imposing any constraint. For example, although shrinkage prevention minimizes the problem in the $\lambda|\mu$ algorithm, all the smooth signal values are different from the original ones.

Taubin [36] shows that by modifying the neighborhood structure certain kind of constraints can be imposed without any modification of the algorithm, while other constraints that require minor modifications and the solution of small linear systems.

Kobbelt [21, 22] formulates the problem as an energy minimization problem, and solves it efficiently with a multiresolution approach on levels of detail hierachies generated by decimation.

Kuriyama [23] and Yamada [44] impose hard constraints on vertex positions, but modify the displacement produced by the Laplacian operator to impose soft normal constraints.

We will only discuss here some of these methods.

8.1. Interpolatory Constraints

A simple way to introduce interpolatory constraints in the smoothing algorithm is by using non-symmetric neighborhood structures. If no other vertex is a neighbor of a certain vertex v_1, i.e., if the neighborhood of v_1 is empty, then the value x_1 of any signal x does not change during the smoothing process, because the Laplacian operator Δx_1 is equal to zero by definition of empty sum. Other vertices are allowed to have v_1 as a neighbor, though.

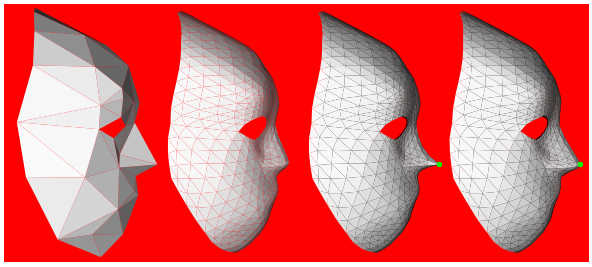


Figure 10: *Example of surfaces designed using subdivision and smoothing steps with one interpolatory constraint. (A) Skeleton. (B) Surface (A) after two levels of subdivision and smoothing without constraints. (C) Same as (B) but with non-smooth interpolatory constraint. (D) Same as (B) but with smooth interpolatory constraint. Surfaces are flat-shaded to enhance the faceting effect.*

Figure 10-A shows a skeleton surface. Figure 10-B shows the surface generated after two levels of refinement and smoothing using our smoothing algorithm without constraints, i.e., with symmetric first-order neighborhoods. Although the surface has not shrunk overall, the nose has been flattened quite significantly. This is so because the nose is made of very few faces in the skeleton, and these faces meet at very sharp angles. Figure 10-C shows the result of applying the same steps, but defining the neighborhood of the vertex at the tip of the nose to be empty. The other neighborhoods are not modified. Now the vertex satisfies the constraint – it has not moved at all during the smoothing process –, but the surface has lost its smoothness at the vertex. This might be the desired effect, but if it is not, instead of the neighborhoods, we have to modify the algorithm.

8.2. Smooth Interpolation

We look at the desired constrained smooth signal x_C^N as a sum of the corresponding unconstrained smooth signal $x^N = F\,x$

after N steps of our smoothing algorithm (i.e. $F = f(K)^N$), plus a smooth deformation d_1

$$x_C^N = x^N + (x_1 - x_1^N)\, d_1 \ .$$

The deformation d_1 is itself another discrete surface signal, and the constraint $(x_C^N)_1 = x_1$ is satisfied if $(d_1)_1 = 1$. To construct such a smooth deformation we consider the signal δ_1, where

$$(\delta_i)_j = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases} \ .$$

This is not a smooth signal, but we can apply the smoothing algorithm to it. The result, let us denote it F_{n1}, the first column of the matrix F, is a smooth signal, but its value at the vertex v_1 is not equal to one. However, since the matrix F is diagonally dominated, F_{11}, the first element of its first column, must be non-zero. Therefore, we can scale the signal F_{n1} to make it satisfy the constraint, obtaining the desired smooth deformation

$$d_1 = F_{n1} F_{11}^{-1} \ .$$

Figure 10-D shows the result of applying this process.

When more than one interpolatory constraint must be imposed, the problem is slightly more complicated. For simplicity, we will assume that the vertices have been reordered so that the interpolatory constraints are imposed on the first m vertices, i.e., $(x_C^N)_1 = x_1, \ldots, (x_C^N)_m = x_m$. We now look at the non-smooth signals $\delta_1, \ldots, \delta_m$, and at the corresponding faired signals, the first m columns of the matrix $F = f(K)^N$. These signals are smooth, and so, any linear combination of them is also a smooth signal. Furthermore, since F is non-singular and diagonally dominated, these signals are linearly independent, and there exists a linear combination of them that satisfies the m desired constraints. Explicitly, the constrained smooth signal can be computed as follows

$$x_C^N = x^N + F_{nm} F_{mm}^{-1} \begin{pmatrix} x_1 - x_1^N \\ \vdots \\ x_m - x_m^N \end{pmatrix} \ , \qquad (14)$$

where F_{rs} denotes the sub-matrix of F determined by the first r rows and the first s columns.

To minimize storage requirements, particularly when n is large, and assuming that m is much smaller than n, the computation can be structured as follows. The smoothing algorithm is applied to δ_1 obtaining the first column $F\delta_1$ of the matrix F. The first m elements of this vector are stored as the first column of the matrix F_{mm}. The remaining $m-n$ elements of $F\delta_1$ are discarded. The same process is repeated for $\delta_2, \ldots, \delta_m$, obtaining the remaining columns of F_{mm}. Then the following linear system

$$F_{mm} \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} x_1 - x_1^N \\ \vdots \\ x_m - x_m^N \end{pmatrix}$$

is solved. The matrix F_{mm} is no longer needed. Then the remaining components of the signal y are set to zero $y_{m+1} = \cdots = y_n = 0$. Now the smoothing algorithm is applied to the signal y. The result is the smooth deformation that makes the unconstrained smooth signal x^N satisfy the constraints

$$x_C^N = x^N + F\, y \ .$$

8.3. Smooth Deformations

Note that in the constrained smoothing algorithm described above the fact that the values of the signal at the vertices of interest is constraint to remain constant can be trivially generalized to allow for arbitrary smooth deformations of a surface. To do so, if in equation (14), the values $x_1, \ldots, x_m$ must be replaced by the desired final values of the faired signal at the corresponding vertices. As in in the Free-form deformation approaches of Hsu, Hughes, and Kaufman [17] and Borrel [4], instead of moving control points outside the surface, surfaces can be deformed here by pulling one or more vertices.

Also note that the scope of the deformation can be controlled by changing the number of smoothing steps applied while smoothing the signals $\delta_1, \ldots, \delta_n$. To make the resulting signal satisfy the constraint, the value of N in the definition of the matrix F must be the one used to smooth the deformations. We have observed that good results are obtained when the number of iterations used to smooth the deformations is about five times the number used to fair the original shape.

8.4. Hierarchical Constraints

This is another application of non-symmetric neighborhoods. We start by assigning a numeric label l_i to each vertex of the surface. Then we define the neighborhood structure as follows. We make vertex v_j a neighbor of vertex v_i if v_i and v_j share an edge (or face), *and* if $l_i \leq l_j$. Note that if v_j is a neighbor of v_i and $l_i < l_j$, then v_i is not a neighbor of v_j. The symmetry applies only to vertices with the same label. For example, if we assign label $l_i = 1$ to all the boundary vertices of a surface with boundary, and label $l_i = 0$ to all the internal vertices, then the boundary is faired as a curve, independently of the interior vertices, but the interior vertices follow the boundary vertices. If we also assign label $l_i = 1$ to a closed curve composed of internal edges of the surface, then the resulting surface will be smooth *along*, and on both sides of the curve, but not necessarily *across* the curve. Figure 11-D shows examples of subdivision surface designed using this procedure. If we also assign label $l_i = 2$ to some isolated points along the curves, then those vertices will in fact not move, because they will have empty neighborhoods.

8.5. Tangent Plane Constraints

Although the normal vector to a polyhedral surface is not defined at a vertex, it is customary to define it by averaging some local information, say for shading purposes. When the

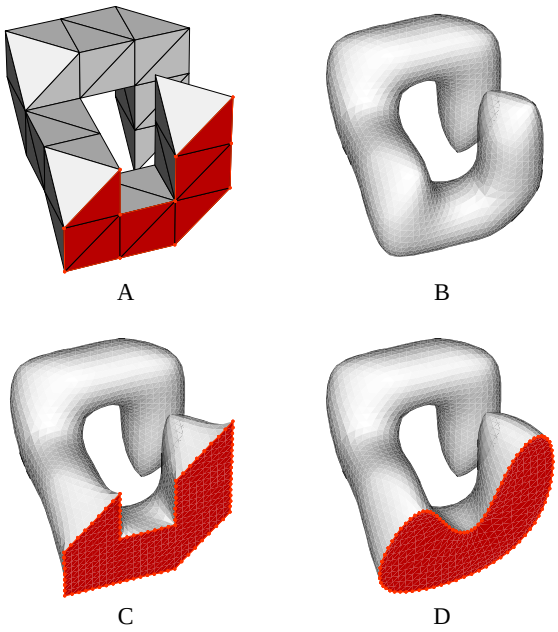


Figure 11: *(A) Skeleton with marked vertices. (B) Surface (A) after three levels of subdivision and smoothing without constraints. (C) Same as (B) but with empty neighborhoods of marked vertices. (D) Same as (B) but with hierarchical neighborhoods, where marked vertices have label 1 and unmarked vertices have label 0. Surfaces are flat-shaded to enhance the faceting effect.*

signal x in equation (1) is replaced by the coordinates of the vertices, the Laplacian becomes a vector

$$\Delta v_i = \sum_{j \in i^*} w_{ij} \, (v_j - v_i) \; .$$

This vector average can be seen as a discrete approximation of the following curvilinear integral

$$\frac{1}{|\gamma|} \int_{v \in \gamma} (v - v_i) \, dl(v) \; ,$$

where γ is a closed curve embedded in the surface which encircles the vertex v_i, and $|\gamma|$ is the length of the curve. It is known that, for a curvature continuous surface, if the curve γ is let to shrink to to the point v_i, the integral converges to the mean curvature $\bar{\kappa}(v_i)$ of the surface at the point v_i times the normal vector N_i at the same point [8]

$$\lim_{\varepsilon \to 0} \frac{1}{|\gamma_\varepsilon|} \int_{v \in \gamma_\varepsilon} (v - v_i) \, dl(v) = \bar{\kappa}(v_i) N_i \; .$$

The expression on the right hand side is the *curvature normal*, where $\bar{\kappa}(v_i)$ is the mean curvature of the surface at v_i and N_i is the surface normal at v_i. It follows that the length of the laplacian vector is equal to the product of the average edge length times the mean curvature

$$\Delta v_i = \left(\sum_{j \in i^*} w_{ij} \, \| (v_j - v_i) \| \right) \bar{\kappa}(v_i) N_i \; ,$$

which can be used as a definition of discrete mean curvature [35].

It follows that imposing normal constraints at v_i is achieved by imposing linear constraints on Δv_i. If N_i is the desired normal direction at vertex v_i after the smoothing process, and S_i and T_i are two linearly independent vectors tangent to N_i, the surface after N iterations of the smoothing algorithm will satisfy the normal desired constraint at the vertex v_i it the following two linear constraints

$$S_i^t \Delta v_i^N = T_i^t \Delta v_i^N = 0$$

are satisfied. This leads us to the problem of smoothing with general linear constraints.

8.6. General Linear Constraints

We consider here the problem of smoothing a discrete surface signal x under general linear constraints $C x_C^N = c$, where C is a $m \times n$ matrix of rank m (m independent constraints), and $c = (c_1, \ldots, c_m)^t$ is a vector. The method described in section 8.1 to impose smooth interpolatory constraints, is a particular case of this problem, where the matrix C is equal the upper m rows of the $m \times m$ identity matrix. Our approach is to reduce the general case to this particular case.

We start by decomposing the matrix C into two blocks. A first $m \times m$ block denoted $C_{(1)}$, composed of m columns of C, and a second block denoted $C_{(2)}$, composed of the remaining columns. The columns that constitute $C_{(1)}$ must be chosen so that $C_{(1)}$ become non-singular, and as well conditioned as possible. In practice this can be done using Gauss elimination with full pivoting [10], but for the sake of simplicity, we will assume here that $C_{(1)}$ is composed of the first m columns of C. We decompose signals in the same way. $x_{(1)}$ denotes here the first m components, and $x_{(2)}$ the last $n - m$ components, of the signal x. We now define a change of basis in the vector space of discrete surface signals as follows

$$\begin{cases} x_{(1)} &= y_{(1)} - C_{(1)}^{-1} C_{(2)} y_{(2)} \\ x_{(2)} &= y_{(2)} \end{cases} \; .$$

If we apply this change of basis to the constraint equation $C_{(1)} x_{(1)} + C_{(2)} x_{(2)} = c$, we obtain $C_{(1)} y_{(1)} = c$, or equivalently

$$y_{(1)} = C_{(1)}^{-1} c \; ,$$

which is the problem solved in section 8.2.

9. Conclusions

In this paper I described the basic elements of the signal processing approach on meshes. It started as a solution to the

shrinkage problem of Laplacian smoothing, and has evolved quite significantly during the last five years, with many important contributions and extensions by many authors, and applications to other areas. In my opinion, the main reason for this interest has been the simplicity of the algorithms and the good qulity of the results produced. I believe that this area will continue evolving in the near future, with theoretical advances, new efficient algorithms, and important applications. Many concepts of classical signal processing may see usefull applications in computer graphics and geometric design, if efficient implementations become available. I look forward to continue contributing to this field myself.

References

1. C.L. Bajaj and Ihm. Smoothing polyhedra using implicit algebraic splines. *Computer Graphics*, pages 79–88, July 1992. (Proceedings SIGGRAPH'92).
2. R. Balan and G. Taubin. 3d mesh geometry filtering algorithms for progressive transmission schemes. *Computer Aided Design*, 2000. Special issue on multiresolution geometric models.
3. F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 1999.
4. P. Borrel. Simple constrained deformations for geometric modeling and interactive design. *ACM Transactions on Graphics*, 13(2):137–155, April 1994.
5. G. Celniker and D. Gossard. Deformable curve and surface finite-elements for free-form shape design. *Computer Graphics*, pages 257–266, July 1991. (Proceedings SIGGRAPH'91).
6. P.J. Davis. *Interpolation and Approximation*. Dover Publications, Inc., 1975.
7. M. Desbrun, M. Meyer, P. Schröder, and A.H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Siggraph'99 Conference Proceedings*, pages 317–324, August 1999.
8. M. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.
9. K. Fujiwara. Eigenvalues of laplacians on a closed riemannian manifold and its nets. *Proceedings of the AMS*, 123:2585–2594, 1995.
10. G. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins University Press, 2nd. edition, 1989.
11. G. Greiner and H.P. Seidel. Modeling with triangular b-splines. *IEEE Computer Graphics and Applications*, 14(2):56–60, March 1994.
12. I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *Siggraph'99 Conference Proceedings*, pages 325–334, August 1999.
13. M. Halstead, M. Kass, and T. DeRose. Efficient, fair interpolation using catmull-clark surface. *Computer Graphics*, pages 35–44, August 1993. (Proceedings SIGGRAPH'93).
14. R.W. Hamming. *Digital Filters*. Prentice Hall, 1989.
15. K. Ho-Le. Finite element mesh generation methods: A review and classification. *Computer Aided Design*, 20(1):27–38, 1988.
16. H. Hoppe. Progressive meshes. In *Siggraph'96 Conference Proceedings*, pages 99–108, August 1996.
17. W.M. Hsu, J.F. Hughes, and H. Kaufman. Direct manipulation of free-form deformations. *Computer Graphics*, pages 177–184, July 1992. (Proceedings SIGGRAPH'92).
18. IEEE Press, New York. *Programs for Digital Signal Processing*, 1979. Algorithm 5.1.
19. Z. Karni and C Gotsman. Spectral compression of mesh geometry. In *Siggraph'2000 Conference Proceedings*, 2000.
20. M. Kass, A. Witkin, and D. Terzopoulos. Snakes: active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.
21. L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Siggraph'98 Conference Proceedings*, pages 105–114, July 1998.
22. L. Kobbelt, J. Vorsatz, and H.-P. Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry Theory and Applications*, 1999. special issue on multiresolution modeling and 3D geometry compression.
23. S. Kuriyama and K. Tachibana. Polyhedral surface modeling with a diffusion system. In *Eurographics'97 Conference Proceedings*, pages C39–C46, 1997.
24. C. Loop. A G^1 triangular spline surface of arbitrary topological type. *Computer Aided Geometric Design*, 11:303–330, 1994.
25. C. Loop. Smooth spline surfaces over irregular meshes. *Computer Graphics*, pages 303–310, July 1994. (Proceedings SIGGRAPH'94).
26. M. Lounsbery, S. Mann, and T. DeRose. Parametric surface interpolation. *IEEE Computer Graphics and Applications*, 12(5):45–52, September 1992.
27. J. Menon. Constructive shell representations for freeform surfaces and solids. *IEEE Computer Graphics and Applications*, 14(2):24–36, March 1994.
28. H.P. Moreton and C.H. Séquin. Functional optimization for fair surface design. *Computer Graphics*, pages 167–176, July 1992. (Proceedings SIGGRAPH'92).
29. A.V. Oppenheim and R.W. Schafer. *Digital Signal Processing*. Prentice Hall, Englewood Cliffs, NJ, 1975.
30. A. Pentland and S. Sclaroff. Closed-form solutions for physically based shape modeling and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):715–729, July 1991.
31. L.A. Shirman and C.H. Séquin. Local surface interpolation with bezier patches. *Computer Aided Geometric Design*, 4:279–295, 1987.
32. G. Strang. The discrete cosine transform. *SIAM Review*, 41(1):135–147, 1999.

33. R.S. Szeliski, D. Tonnesen, and D. Terzopoulos. Modeling surfaces of arbitrary topology with dynamic particles. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–87, New York, NY, June 15–17 1993.

34. G. Taubin. Mesh smoothing applet. http://www.research.ibm.com/people/t/taubin/smooth.

35. G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings, International Conference on Computer Vision (ICCV)*, 1995.

36. G. Taubin. A signal processing approach to fair surface design. In *Siggraph'95 Conference Proceedings*, pages 351–358, August 1995.

37. G. Taubin, A. Guéziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *Siggraph'98 Conference Proceedings*, pages 123–132, July 1998.

38. G. Taubin and J. Rossignac, editors. *3D Geometry Compression*, Siggraph'98 Course Notes 21, July 1998.

39. G. Taubin and J. Rossignac, editors. *3D Geometry Compression*, Siggraph'99 Course Notes 22, August 1999.

40. G. Taubin, T. Zhang, and G. Golub. Optimal surface smoothing as filter design. In *Fourth European Conference on Computer Vision (ECCV'96)*, 1996. Also as IBM Technical Report RC-20404, March 1996.

41. D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4:306–311, 1988.

42. W. Welch and A. Witkin. Variational surface modeling. *Computer Graphics*, pages 157–166, July 1992. (Proceedings SIGGRAPH'92).

43. W. Welch and A. Witkin. Free-form shape design using triangulated surfaces. *Computer Graphics*, pages 247–256, July 1994. (Proceedings SIGGRAPH'94).

44. A. Yamada, T. Furuhata, K. Shimada, and K. Hou. A discrete spring model for generating fair curves and surfaces. In *Proceedings of the Seventh Pacific Conference on Computer Graphics and Applications*, pages 270–279, 1998.

45. C.T. Zahn and R.Z. Roskies. Fourier descriptors for plane closed curves. *IEEE Transactions on Computers*, 21(3):269–281, March 1972.

46. D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Siggraph'97 Conference Proceedings*, pages 259–268, August 1997.

Geometric Fairing and Variational Subdivision for Freeform Surface Design

by R. Schneider, and L. Kobbelt,

Computer Aided Geometric Design 2001

Geometric Fairing of Irregular Meshes for Free-Form Surface Design

Robert Schneider *, Leif Kobbelt [1]

Max-Planck Institute for Computer Sciences, Stuhlsatzenhausweg 85, D-66123 Saarbrücken, Germany

Abstract

In this paper we present a new algorithm for smoothing arbitrary triangle meshes while satisfying G^1 boundary conditions. The algorithm is based on solving a non-linear fourth order partial differential equation (PDE) that only depends on intrinsic surface properties instead of being derived from a particular surface parameterization. This continuous PDE has a (representation-independent) well-defined solution which we approximate by our triangle mesh. Hence, changing the mesh complexity (refinement) or the mesh connectivity (remeshing) leads to just another discretization of the same smooth surface and doesn't affect the resulting geometric shape beyond this. This is typically not true for filter-based mesh smoothing algorithms. To simplify the computation we factorize the fourth order PDE into a set of two nested second order problems thus avoiding the estimation of higher order derivatives. Further acceleration is achieved by applying multigrid techniques on a fine-to-coarse hierarchical mesh representation.

Key words: Discrete fairing, Free-Form modeling, PDE method, Parameterization independence

1 Introduction

Although piecewise polynomial patches are still the dominating free-form surface representation in engineering applications, the use of triangle meshes has become increasingly important – especially with the rising complexity of 3D models. A general approach to generate free-form surfaces that satisfy aesthetic requirements is *surface fairing* where auxiliary degrees of freedom are

* Corresponding author. E-mail: schneider@mpi-sb.mpg.de
[1] E-mail: kobbelt@mpi-sb.mpg.de

used to improve the global distribution of curvature (or optimize any other quality criterion). For triangle meshes this type of optimization has two different aspects. First, the triangle mesh should have *outer fairness*, i.e. the imaginary surface that is approximated by the mesh should be optimal with respect to curvature distribution. Additionally, the mesh should have *inner fairness* which means that the distribution of mesh vertices *within* the surface and the shape of the individual faces should be good according to application dependent requirements.

The result of recent mesh fairing algorithms usually depends on the underlying mesh connectivity, thus the inner fairness influences the resulting outer fairness in some way and even for the most sophisticated schemes there is no guarantee that the surfaces are free of parameterization artifacts (Fig. 1). A solution to this shortcoming is to use a fairing algorithm that is able to separate the two fairness types (Fig. 2). This is achieved by using an outer fairness measure that is based on intrinsic surface properties only, i.e. properties that depend on the geometry alone. Unfortunately, intrinsic fairing is a nonlinear problem, and while the linear fairing operators are highly efficient and in general mathematically well understood, the analysis in the non-linear case is much more difficult. To the authors knowledge, even for the rather simple fairing functional

$$\int_A \kappa_1^2 + \kappa_2^2 \, dA, \tag{1}$$

leading to a minimal energy surface (MES), it is still unknown if a solution always exists within specific smoothness classes.

In this paper the separation of outer and inner mesh fairness and parameter independence is achieved by using an outer fairness concept that is based on a discrete solution of an intrinsic PDE. The PDE we choose is of fourth order and leads to surfaces of high quality. To speed up the construction scheme, we factorize the fourth order PDE into two second order problems and use a hierarchic mesh representation to enable multigrid techniques.

The paper is organized as follows: Section 2 reviews related work on mesh fairing. In section 3 we present the concept of our intrinsic fairing approach. Section 4 defines the notation we use throughout the paper. In section 5 and 6 we show how to discretize the necessary intrinsics. Finally, in section 7 we present the details of our algorithm.

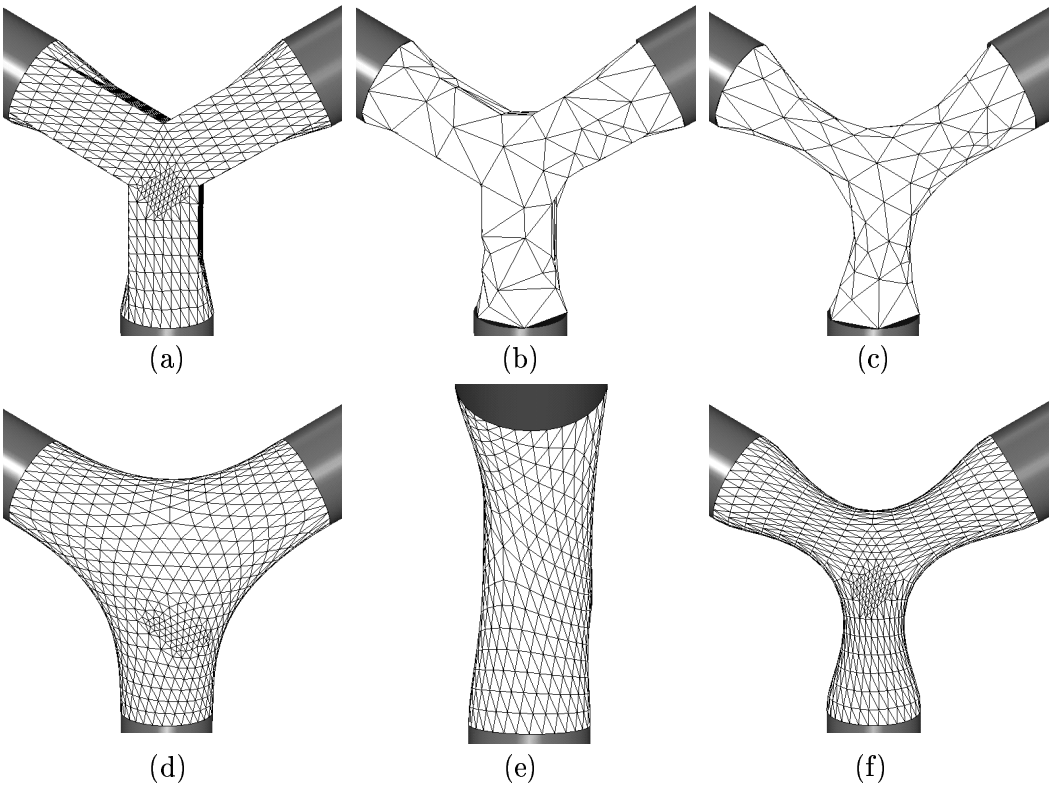


Fig. 1. Mesh fairing based on discretizing the equation $\Delta^2 f = 0$ using two different local parameterization strategies. The boundary condition is determined by 3 cylinders that are arranged symmetrically. a) shows the original mesh (920 vertices) and b) a reduced version with 118 vertices. In c), d) and e) we see the results if local uniform parameterization is assumed. In f) we used a discretization of a laplacian that was derived from a discrete harmonic map to fair the original mesh. As we can see, the mesh size and the local parameterization strategy heavily influence the resulting surfaces. The rectangular patch introduced in the original mesh leads to local as well as global shape distortions and prevents a symmetric solution.

2 Mesh fairing - previous work

Most mesh fairing schemes are based on linear operators. While this leads to simple and fast algorithms, there is a serious consequence: The outer fairness and therefore the shape of the resulting mesh highly depends on the chosen parameterization strategy. As a consequence it is not possible to separate the outer and inner fairness concept in this case. If we change the inner fairness strategy we have to change the parameterization and will therefore also change the outer fairness (see Fig. 1 (d) and (f)).

There are two types of fairing algorithms, depending on whether the fairing is based on the calculation of a well defined surface or whether it is based on filtering operations.

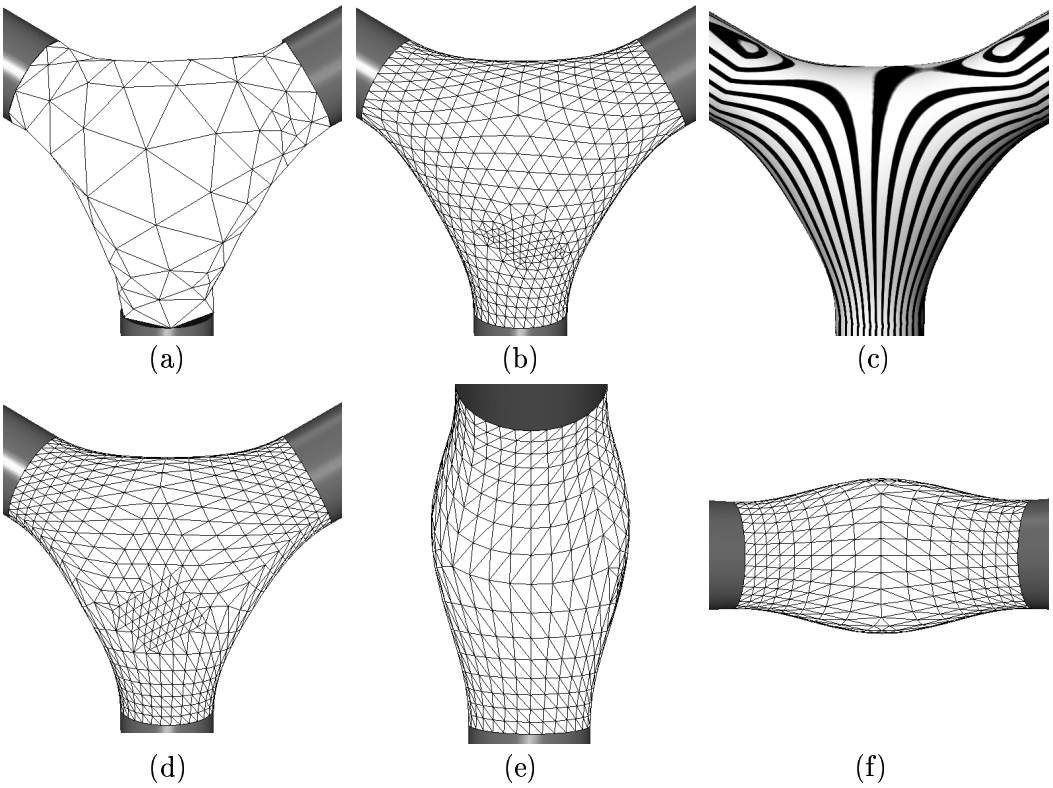


Fig. 2. Our new fairing approach applied on the example shown in Fig 1. In a), b) and c) we optimized the inner fairness with respect to a local uniform parameterization. In d), e) and f) the mesh is discrete conformal to the original mesh in Fig. 1 a). We can see that the chosen inner fairness strategy and the mesh structure have only marginal influence on the shape. The influence of the mesh size is also much smaller than in the linear setting. The continuity of the refection lines in c) indicates G^1 continuity at the boundary.

The standard approach for fair surface construction is based on the idea to minimize a fairness metric, punishing features that are inconsistent with the fairness principle of the simplest shape (Burchard et al., 1994). Applied to meshes this leads to the discrete fairing approach proposed by Kobbelt (1997). Besides of energy minimization, during the last years various other linear mesh fairing schemes have been developed.

A very effective method for smoothing polyhedral surfaces is the discrete diffusion flow (Taubin, 1995a). Here the idea is to iteratively update each vertex

$$q_i^{'} = q_i + \lambda \Delta q_i \tag{2}$$

by adding a displacement vector that is a scaled discrete laplacian Δq_i. For stability reasons the scale factor λ has to satisfy $0 < \lambda < 1$. A diffusion flow that is unconditionally stable and hence enables larger scale factors was presented by Desbrun et al. (1999).

The main purpose of the diffusion flow is to remove the high frequencies in noisy meshes. Since the equilibrium surface of the flow only enables C^0 boundary conditions, (2) is of only limited use in surface design. To enable smooth boundary conditions one has to consider diffusion equations of higher order. Taubin (1995a) proposed to combine two such smoothing steps with positive and negative scale factors and developed an algorithm that enables various interpolation constraints. Another idea that enables smooth boundary conditions is to use higher powers of the laplacian in the diffusion flow. As a good trade-off between efficiency and quality one can chose the bilaplacian flow, enabling C^1 boundary conditions.

Another mesh fairing approach is based on the idea to discretize the PDE approach of Bloor and Wilson (1990). In (Kobbelt et al., 1998b) it was proposed to discretize

$$\Delta^2 f = 0, \tag{3}$$

where Δ is the laplacian operator, to create surfaces satisfying prescribed C^1 boundary conditions. This equation results if we apply variational calculus to the thin plate energy

$$\iint f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2 \, dxdy$$

to describe the minimum of the functional.

Fairing based on some kind of diffusion flow and fairing based on the discretization of the PDE (3) that describes the smooth solution are tightly connected problems, since (3) is the equilibrium of the bilaplacian flow. In both cases the discretization of the laplacian plays the central role. During the last years various linear discretizations of the laplacian have been developed (Taubin, 1995a; Kobbelt et al., 1998b; Desbrun et al., 1999; Guskov et al., 1999), differing in how the geometry of the mesh affects the discretization. The chosen discretization determines the inner mesh fairness, but it also greatly influences the shape of the resulting mesh (Fig. 1).

To make the outer fairness independent of the mesh parameterization, other approaches are based on intrinsic surface properties. These approaches lead to nonlinear fairing schemes.

In (Welch and Witkin, 1994) a mesh fairing algorithm was presented that enables G^1 boundary conditions based on the idea to minimize the total curvature (1) thus punishing large curvature values. For an isometric parameterization this approach coincides with minimizing the thin plate energy (3). The necessary intrinsic curvature values were estimated using local quadratic ap-

proximations over local planar parameterizations. However, in constellations where the local quadratic approximations are not uniquely defined, their approach may lead to stability problems.

An intrinsic diffusion operator using a discrete mean curvature flow was presented by Desbrun et al. (1999), leading to an excellent noise reduction algorithm. While this fairing algorithm is mainly designed for outer fairness without tangential shift, in (Ohtake et al., 2000) this algorithm was combined with an inner fairness criterion that leads to more regular meshes. Since these algorithms converge to a discrete minimal surface satisfying $H = 0$, they only enable C^0 boundary conditions and are therefore, as their linear counterparts, of only limited use in surface design. Again, the solution is to use curvature flows of higher order (Brakke, 1992; Hsu et al., 1992; Chopp and Sethian, 1999).

In (Schneider and Kobbelt, 2000a) it was proposed to approximate a PDE based on intrinsics to create fair meshes with G^1 boundary conditions in the special case where the meshes have subdivision connectivity. As we will see in the next section, our method shares the principal idea to solve an intrinsic PDE, but enables much greater flexibility.

3 Our fairing concept

We present a fairing algorithm for arbitrary triangle meshes that enables G^1 boundary conditions (prescribed vertices and unit normals) and allows us to completely separate outer and inner fairness. To achieve this, the outer fairness strategy is based on the idea to discretize an intrinsic PDE. Given G^1 information at the boundary, the PDE defines a smooth surface satisfying the constraints, altering the mesh size, the connectivity or the inner fairness condition only produces another discretization of the same smooth surface and hence leads to geometries that will be close to each other (Fig. 2).

In practice this allows us to choose our inner fairness criterion freely. In this paper we restricted ourselves to two especially important cases. One leads to a regular mesh parameterization, the other produces meshes that are conformally parameterized to a given initial polyhedron. The latter inner fairness method plays a fundamental role for fairing of textured meshes and in mesh editing, since it minimizes local distortions. An other consequence is that a coarse mesh already approximates the shape of the smooth surface that is implicitly defined by the PDE, so increasing the mesh size mainly improves the quality of the approximation not the quality of the underlying shape. We exploit this property to improve the efficiency of our construction algorithm by using multigrid methods for arbitrary meshes. The necessary mesh hierar-

chies are created using the progressive mesh representation as introduced by Hoppe (1996).

Following the set-up presented in (Schneider and Kobbelt, 2000a) to define fair surfaces satisfying G^1 boundary constraints, the PDE that determines our outer fairness concept in this paper is defined as

$$\Delta_B H = 0, \tag{4}$$

which can be interpreted as surface analogon to the planar equation $\kappa'' = 0$ (the derivative of the curvature κ is with respect to arc length) leading to clothoid splines. Here Δ_B is the Laplace-Beltrami operator and H the mean curvature. The PDE only depends on geometric intrinsics and is comparatively simple for a forth order equation. Because of the mean value property of the laplacian, it is guaranteed that the extremal mean curvature values of a solution of (4) will be reached at the border and that there are no local extrema in the interior. Since constant mean curvature surfaces satisfy this equation, important basic shapes as spheres, cylinders and minimal surfaces with $H = 0$ can be reconstructed. Although the PDE we propose can be derived as a simplification of the Euler-Lagrange equation resulting from MESs (1), this is a fairing technique in its own right, which therefore doesn't have to be inferior to the MES approach. In fact, this approach even has some advantages over MESs, e.g. it reproduces cylinders, while it follows from the Euler-Lagrange equation of (1) that MESs do not reproduce that surface class. This again is completely analogous to the planar case. It is well known, that minimal energy curves do not reproduce circles, while a clothoid spline obviously does.

In (Schneider and Kobbelt, 2000a) solutions of (4) were approximated by meshes in the special case where the meshes have subdivision connectivity and the boundary vertices could be regularly sampled on a smooth curve. The construction scheme was based on the idea to design an inner fairness criterion that partitions the surface in regular regions. Exploiting this regularity knowledge in advance, it was possible to assign a local planar parameterization to each vertex. Using these domains, the intrinsic values could be approximated by local quadratic approximations, thus the quality of the discretization depends on the quality of the estimated local planar parameterizations.

The algorithm presented in this paper doesn't have such limitations. There are also no restrictions concerning the mesh structure and the boundary vertices and we are free to choose an inner fairness criteria. Nevertheless, the resulting construction algorithm is fast and can be implemented compactly. Instead of trying to simulate the continuous case using local quadratic approximations, we use the discrete data of our minimization process directly.

4 Notation

We partition the vertices of a mesh M into two classes, denoting the set of all border vertices with $V_B(M)$ and the set of all vertices in the interior of M with $V_I(M)$. For each vertex q_i of M let $N(q_i)$ be the set of vertices q_j that are adjacent to q_i and let $D(q_i) = N(q_i) \cup \{q_i\}$ be the according 1-disk. Let $H_i = H(q_i)$ denote the discrete mean curvature and $\vec{n}_i = \vec{n}(q_i)$ the discrete unit normal vector at the vertex q_i. When it is clear which mean curvature value or normal vector is meant, in some cases we omit the index to increase the readability.

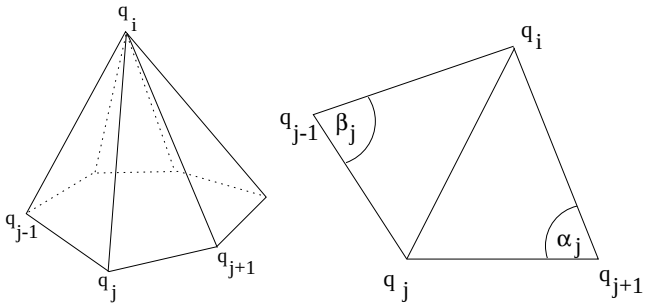


Fig. 3. The Laplace-Beltrami operator at the vertex q_i can be discretized using $D(q_i)$.

5 Discretization of the Laplace-Beltrami operator

The discretization relies on the fact that there is a tight connection between the Laplace-Beltrami operator Δ_B and the mean curvature normal of a surface. In (Desbrun et al., 2000) also an improved discretization can be found that uses a more detailed area calculation based on voronoi regions, but the following discretization is sufficient for our needs. Let $f : \mathbb{R}^2 \to \mathbb{R}^3$ be a parameterization of a surface, then it is well known that the following equation holds

$$\Delta_B f \;=\; 2H\vec{n}.$$

Exploiting this relation, a discretization of Δ_B follows directly from the mean curvature flow approach for arbitrary meshes that was presented by Desbrun et al. (1999). They showed that the mean curvature normal at a vertex q_i of a triangular mesh M can be discretized using its 1-neighborhood by

$$H\vec{n} \;=\; \frac{3}{4A} \sum_{q_j \in N(q_i)} (\cot\, \alpha_j + \cot\, \beta_j)(q_j - q_i), \tag{5}$$

where A is the sum of the triangle areas of the 1-disk at q_i and α_j and β_j are the triangle angles as shown in figure 3. Since the vertices can be interpreted as sample points of a smooth surface parameterized by f and exploiting the fact that a scaling factor does not influence the result, we can discretize the equation $\Delta_B H = 0$ at a vertex q_i as

$$\sum_{q_j \in N(q_i)} (\cot\, \alpha_j + \cot\, \beta_j)(H_i - H_j) \;=\; 0.$$

If this equation is satisfied at all inner vertices $q_i \in V_I(M)$ and if we further know all mean curvature values for the boundary vertices $V_B(M)$, this leads us to a sparse linear system in the unknowns $H_i \in V_I(M)$, whose matrix S has the coefficients

$$S_{ii} = \sum_{q_j \in N(q_i)} (\cot\, \alpha_j + \cot\, \beta_j), \tag{6}$$

$$S_{ij} = \begin{cases} -(\cot\, \alpha_j + \cot\, \beta_j) & : \quad q_j \in N(q_i) \cap V_I(M) \\ 0 & : \quad \text{otherwise.} \end{cases} \tag{7}$$

The matrix S is symmetric and – as long as no triangle areas of the mesh M vanish – positive definite. To see this, we note that S also appears in a paper by Pinkall and Polthier (1993), where a stable construction algorithm for discrete minimal surfaces based on the idea to minimize the discrete Dirichlet energy of a mesh was presented. Hence, for an elegant proof of the mathematical structure of this matrix we can refer to this paper. It should be mentioned that S further appears in a paper about piecewise linear harmonic functions (Duchamp et al., 1997).

6 Discretization of the mean curvature

In this section we present a discretization of the mean curvature H_i at a vertex q_i that depends on the vertices in a local neighborhood. There are various techniques to discretize surface curvatures, but to be applicable for our construction algorithm, it is important that – for a given mesh connectivity – the discretization of H_i is a continuous function of those vertices.

Not all curvature discretization schemes satisfy this property. In (Welch and Witkin, 1994) it was proposed to discretize curvature information using local quadratic least square approximation over local planar parameterizations. However, it is well known that least square approximation fails, if the points in

the parameterization plane lie on a curve of degree 2. To be able to detect such cases, the authors estimated the condition number of the least square problem in the Frobenius Norm and reduced the number of the basis functions if the problem was ill-conditioned. This approach is not only costly, but makes the discretization process discontinuous and thus leads to a potential instability in the mesh fairing algorithm.

To avoid analogous stability problems, in the following we propose a mean curvature discretization technique that satisfies the continuity criterion. Moreover, as we will show in section 7.2, the presented scheme has other favorable properties that simplify the construction process.

At first glance, it seems tempting to use equation (5) to discretize the mean curvature, but this would only be applicable for inner vertices. However, in our construction algorithm presented later in section 7, we have to be able to discretize the mean curvature not only for inner vertices, but also for boundary vertices, where we have a completely different situation. Due to the boundary constraints, here we already know the tangent plane of the final surface, but we don't have a complete 1-disk. To avoid having mean curvature discretizations of different accuracy, we choose one technique that is able to handle both cases, so our method expects that a tangent plane is known at every vertex. At interior vertices where no normal vector is known in advance, we define it to be the normalized sum of the vector crossproducts of the incident triangle faces, in order to minimize square root operations.

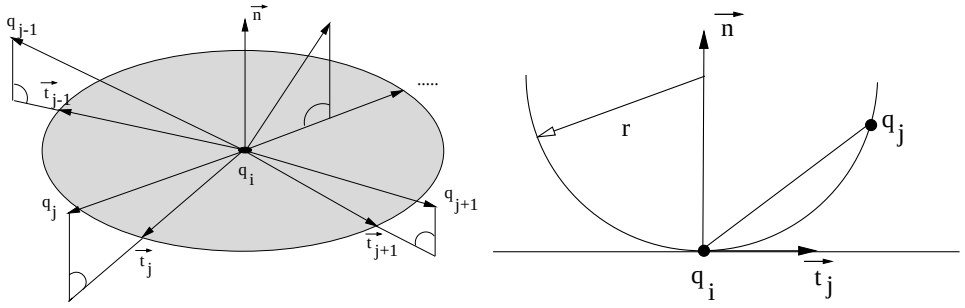


Fig. 4. Left: Projecting the neighborhood of q_i onto the plane defined by $\vec{n}$ and normalizing the results we get the normal curvature directions $\vec{t_j}$. Right: The normal curvature κ_j along the direction $\vec{t_j}$ is discretized by interpolating q_i and q_j with a circle and using the inverse of the circle radius r as normal curvature. The center of the circle lies on the line defined by q_i and $\vec{n}$.

6.1 Moreton and Séquin's curvature discretization algorithm

A curvature discretization algorithm that seems ideal for our needs was presented by Moreton and Séquin (1992). The idea of their approach is to use the fact that the normal curvature distribution cannot be arbitrary, but is

determined by Euler's theorem (doCarmo, 1993).

Let $\vec{b}_x$ and $\vec{b}_y$ be an arbitrary orthonormal basis of the plane defined by the normal $\vec{n}$. To each vertex $q_j \in N(q_i)$ we can assign a unit direction vector $\vec{t}_j$ by projecting q_j into the plane and scaling this projection to unit length. For each q_j we can now estimate a normal curvature $\tilde{\kappa}_j$ as the inverse of the circle radius defined by q_i, q_j and t_j (Fig. 4)

$$\tilde{\kappa}_j = 2\frac{\left\langle q_j - q_i \middle| \vec{n} \right\rangle}{\left\langle q_j - q_i \middle| q_j - q_i \right\rangle}. \tag{8}$$

Using Euler's theorem, we can express the normal curvature κ_n for a direction $\vec{t}$ by the principal curvatures κ_1 and κ_2 and the principal curvature directions $\vec{e}_1$ and $\vec{e}_2$. Let t_x and t_y be the coordinates of $\vec{t}$ in the basis $\vec{b}_x, \vec{b}_y$ and let e_x and e_y be the coordinates of $\vec{e}_1$, then the normal curvature can be expressed as

$$\kappa_n = \begin{pmatrix} t_x \\ t_y \end{pmatrix}^t \cdot K \cdot \begin{pmatrix} t_x \\ t_y \end{pmatrix}, \tag{9}$$

with

$$K = \begin{bmatrix} e_x & e_y \\ -e_y & e_x \end{bmatrix} \cdot \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} \cdot \begin{bmatrix} e_x & e_y \\ -e_y & e_x \end{bmatrix}^{-1}.$$

The idea of Moreton and Séquin is to use the normal curvatures $\tilde{\kappa}_j$ to create a linear system and find estimates for the unknown principal curvature values by determining the least square solution. Let $t_{j,x}$ and $t_{j,y}$ denote the coordinates of $\vec{t}_j$ and let m be the valence of q_i, then we get by evaluating (9)

$$A\vec{x} = \vec{b}$$

where

$$A = \begin{bmatrix} t_{1,x}^2 & t_{1,x}t_{1,y} & t_{1,y}^2 \\ t_{2,x}^2 & t_{2,x}t_{2,y} & t_{2,y}^2 \\ \cdot & \cdot & \cdot \\ t_{m,x}^2 & t_{m,x}t_{m,y} & t_{m,y}^2 \end{bmatrix}, \ \vec{b} = \begin{bmatrix} \tilde{\kappa}_1 \\ \tilde{\kappa}_2 \\ \cdot \\ \tilde{\kappa}_m \end{bmatrix}$$

and

$$\vec{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} e_x^2\kappa_1 + e_y^2\kappa_2 \\ 2e_xe_y(\kappa_1 - \kappa_2) \\ e_x^2\kappa_2 + e_y^2\kappa_1 \end{pmatrix}.$$

Since $x_0 + x_2 = \kappa_1 + \kappa_2$ this means the mean curvature is determined by

$$H = \frac{1}{2}(x_0 + x_2). \tag{10}$$

In our case the most efficient method to solve the least square problem is to use the normal equations approach (Golub and Van Loan, 1989), since this mainly involves to calculate the inverse of a symmetric 3×3 matrix. Using this approach, the least square solution $\vec{x}$ can be expressed as

$$\vec{x} = (A^tA)^{-1}A^t\vec{b}. \tag{11}$$

The cases when the matrix A^tA becomes singular can be detected by a simple criterion:

Lemma 1 *The Matrix A^tA is singular if and only if all points $(t_{i,x}, t_{i,y})$ are intersection points of the unit circle with two straight lines through the origin.*

Proof: Since singularity of A^tA is equivalent with $\mathbf{rank}\, A < 3$, singularity occurs iff any 3 row vectors of A are linear dependent. Therefore, the proof is complete if we show that a matrix of type

$$\begin{bmatrix} t_{1,x}^2 & t_{1,x}t_{1,y} & t_{1,y}^2 \\ t_{2,x}^2 & t_{2,x}t_{2,y} & t_{2,y}^2 \\ t_{3,x}^2 & t_{3,x}t_{3,y} & t_{3,y}^2 \end{bmatrix}$$

with $t_{i,x}^2 + t_{i,y}^2 = 1$ is singular if and only if the tree points lie on two lines through the origin. This matrix is singular, if there are coefficients a_i which are not all zero such that

$$a_1 \begin{pmatrix} t_{1,x}^2 \\ t_{2,x}^2 \\ t_{3,x}^2 \end{pmatrix} + a_2 \begin{pmatrix} t_{1,x}t_{1,y} \\ t_{2,x}t_{2,y} \\ t_{3,x}t_{3,y} \end{pmatrix} + a_3 \begin{pmatrix} t_{1,y}^2 \\ t_{2,y}^2 \\ t_{3,y}^2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix},$$

but this means each point $(t_{i,x}, t_{i,y})$ lies on a curve that satisfies $a_1x^2 + a_2xy + a_3y^2 = 0$. Depending on the coefficients a_i this equation characterizes a single point (the origin) or two lines through the origin. $\square$

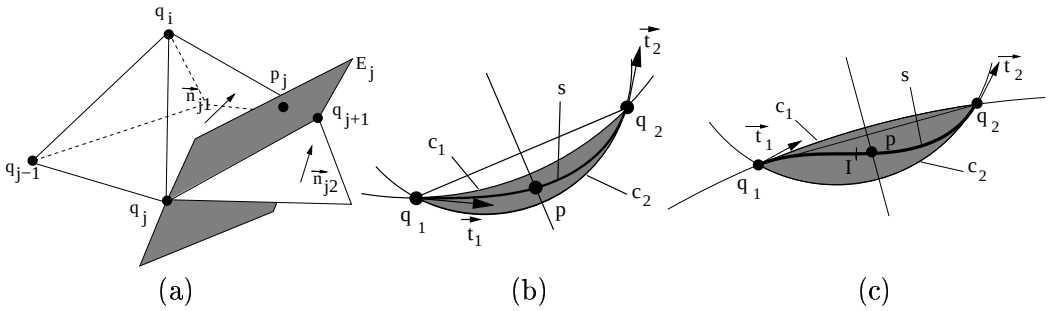


Fig. 5. a) At vertices q_i with valence 3 or 4, between all $q_j, q_{j+1} \in N(q_i)$ that share a common edge, new vertices p_j are introduced. The p_j lie on a plane E_j determined by q_j, q_{j+1} and the vector $\vec{n}_{j1} + \vec{n}_{j2}$, where $\vec{n}_{j1}$ and $\vec{n}_{j2}$ are the triangle normals of the faces adjacent to the edge q_jq_{j+1}. b) Approximation of a spiral (planar curve with monotone nonzero curvature) by the area enclosed by two arcs. c) This approximation also produces reasonable results for planar curves with monotone curvature and an inflection point (denoted as I).

6.2 Singularity handling

Perhaps the most obvious strategy to avoid singularities of A^tA is to simply check whether the criterion presented in Lemma 1 is satisfied and to apply a special method only to such cases near singularity. Such an approach is straightforward, but it can lead to instabilities. A small perturbation of one vertex can trigger a special case handling and thus it could switch the mean curvature discretization method, making the process discontinuous.

An elegant solution to this discontinuity problem is to exploit the connection between possible singularity of A^tA and the vertex valence. Assuming that all points $(t_{i,x}, t_{i,y})$ are distinct, a simple consequence of Lemma 1 is that the matrix A^tA can only become singular, if the valence of q_i is 3 or 4.

For all vertices of valence 3 or 4 we increase the data quantity that serves as input for our algorithm. Instead of enlarging the vertex neighborhood – which lacks symmetry if the valence of the neighbor vertices varies largely – we increase the local input data by estimating new vertices p_j between adjacent $q_j \in N(q_i)$. The p_j and q_j then serve as input for the mean curvature discretization as described in section 6.1, making the problem well posed.

Simply setting $p_j = (q_j + q_{j+1})/2$ would be fast and convenient, but such an approach distorts the resulting mean curvature discretization considerably and should not be used if high quality results have to be generated. A scheme

that has proven to be adequate during our numerical experiments is based on the idea to determine p_j by sampling a planar curve with monotone curvature, that interpolates the vertices and normals at q_j and q_{j+1} (Fig. 5 a). Since it is not obvious what type of spiral (or pair of spirals, if the curve has an inflection point) is most promising and the computation of such an interpolating curve can be expensive, we exploit a nice approximation property of spirals (Marciniak and Putz, 1984):
Given two planar points q_1 and q_2 with tangent vectors $\vec{t}_1$ and $\vec{t}_2$, let s be an arbitrary spiral that satisfies this G^1 interpolation problem. Let c_1 be the circle defined by q_1, q_2 and $\vec{t}_1$ and c_2 be the circle defined by q_1, q_2 and $\vec{t}_2$ (such that $\vec{t}_1$ and $\vec{t}_2$ are circle tangents). Then the spiral s can be approximated by the area enclosed by the circular arcs of c_1 and c_2 between q_1 and q_2, if the tangent angle between $\vec{t}_1$ and $\vec{t}_2$ does not change exceedingly (Fig. 5 b). This argument is also reasonable for planar curves with monotone curvature and an inflection point (Fig. 5 c).

The tangent vectors $\vec{t}_1$ and $\vec{t}_2$ are computed by intersecting the tangent planes at the vertices q_j and q_{j+1} (defined by their normal vectors) with the plane E_j (Fig. 5 a), where we choose in each case the direction that has the smaller angle to the vector $q_{j+1} - q_j$. After intersecting the enclosed area with the perpendicular bisector of q_1 and q_2, we set $p_j = p$ to be the center point of the intersection interval. Each circle has two intersection points with the perpendicular bisector, here we choose only that intersection point that is closer to the line defined by q_1 and q_2.

7 Construction algorithm

In this section we finally present the construction algorithm for a mesh M_S that is a discrete solution of equation (4), i.e. it satisfies

$$\Delta_B H(q_i) = 0 \qquad \forall\, q_i \in V_I(M_S) \tag{12}$$

plus an additional inner fairness criterion. The input data for our algorithm consists of vertices and unit normals that form the G^1 boundary condition and an initial mesh M^0 that interpolates the boundary vertices. The idea of the construction algorithm is to create a mesh sequence $M^k, k = 0, 1, 2, \ldots$ by iteratively updating the vertices, until the outer and inner fairness conditions are sufficiently satisfied. Although in our implementation the mesh connectivity varies during the construction algorithm (see section 7.4), let us first assume that only the position of the vertices changes, while the connectivity remains constant.

One possible strategy to transform the initial mesh M^0 into the solution M_S would be to use motion by intrinsic laplacian of curvature (Chopp and Sethian, 1999), exploiting the fact that (4) describes the equilibrium of that flow. Here, for fine meshes very small time steps are needed to achieve stability and it is hard to decide what concrete time step values should be used in an application.

To avoid such problems, in this paper we do not use curvature flow by the laplacian of curvature directly, instead we adapt the ideas presented in (Schneider and Kobbelt, 1999) to our case. In that paper an algorithm for the fast construction of discrete planar clothoid splines – the planar analogon to our problem – was presented. The key idea of that algorithm was to factorize the fourth order problem into two problems of second order. In combination with a multigrid scheme and an iteration step that alternates between local and global update strategies, an efficient and reliable algorithm for the discretization of planar clothoid splines was presented.

7.1 Factorization

Instead of solving a fourth order problem directly, we factorize it into two second order problems which are solved sequentially. The factorization idea is inspired by the following observation: Given a fixed Laplace-Beltrami operator and fixed mean curvature values at the boundary vertices $V_B(M^k)$ of a mesh M^k, (12) can be interpreted as a Dirichlet problem for the H_i, where the unknown scalar mean curvature values at the inner vertices are determined by a nonsingular linear system with a symmetric and positive definite matrix S whose coefficients are defined in (6) and (7). Solving the resulting nonsingular linear system yields scalar values $\tilde{H}_i$ at all inner vertices $q_i \in V_I(M^k)$, that represent a discrete harmonic function. The idea is now to use this calculated scalar values $\tilde{H}_i$ to update each inner vertex q_i such that $H(q_i^{k+1}) = \tilde{H}_i$, which is again a second order problem. Expressed in two formulas, this factorization of $M^k \to M^{k+1}$ becomes

$$\left.\begin{array}{rcl} I. \quad \Delta_B \tilde{H}_i & = & 0 \\ II.\ H(q_i^{k+1}) & = & \tilde{H}_i \end{array}\right\} \quad \forall\, q_i^k \in V_I(M^k).$$

We determine the Laplace-Beltrami operator and the boundary mean curvature values by calculating the according values of the current mesh M^k. In practice, it is not necessary to solve the Dirichlet problem exactly. When we have determined the linear system, we apply some iteration steps of an iterative linear solver, using the current mean curvature values as starting values. So one step to update a mesh $M^k \to M^{k+1}$ becomes:

- All mean curvature values $H(q_i^k)$ at all vertices of the current mesh M^k are calculated, the mean curvature discretization technique depends on the vertex valence as described in section 6. Also the discrete Laplace-Beltrami operator is determined, that means we calculate the cotangent weights (6) and (7) for every inner vertex.
- The mean curvature values at the boundary and the discretized Laplace-Beltrami operator determine a Dirichlet problem that can be formulated as a linear system for the interior mean curvature values. Use the mean curvature values at the interior vertices as initial values and iterate the linear system n times using an iterative linear system solver.
- The second step results in improved scalar values $\tilde{H}_i$ for the interior vertices and this scalar values are used to update all $q_i^k \in V_I(M^k)$. The update is done in a Gauss-Seidel like manner, that means adjacent vertices that have already been updated are used with their new position.

While discrete harmonic functions do not always have to share all the mathematical properties of their continuous counterparts, e.g. the convex hull property (Pinkall and Polthier, 1993), they will nevertheless approximate continuous harmonic functions. This means our scalar values $\tilde{H}_i$ will approximate a function that does not have local extrema and whose maximal values occur at the boundary, so the $\tilde{H}_i$ will behave well and can be approximately bounded by the current mean curvature values at the boundary vertices.

As in (Schneider and Kobbelt, 1999), we noticed that we can improve the convergence rate, if we mix high and low frequency smoothing steps. For that purpose, in our implementation we used Gauss-Seidel and conjugate gradient (Golub and Van Loan, 1989) iterations. Both schemes enable especially efficient coding if applied on meshes and both would converge for $n \to \infty$, since the matrix S is positive definite. We alternate between a mesh update $M^k \to M^{k+1}$ based on Gauss-Seidel with $n = 1$ and a conjugate gradient update where n is chosen larger. As a non interactive criterion to terminate the fairing algorithm, we can iterate until $||\Delta_B H|| < \epsilon$ at every vertex for a prescribed ϵ.

7.2 Vertex updates

The inner vertices are updated $q_i^k \to q_i^{k+1}$, using the scalar mean curvature values $\tilde{H}_i$ resulting from the iterative solver described above. The aim of the update step is to produce a new mesh M^{k+1} whose mean curvature values $H(q_i^{k+1})$ at the vertices $q_i \in V_I(M^{k+1})$ are closer to the calculated $\tilde{H}_i$ values than those of the previous mesh M^k. In order to be able to separate between inner and outer fairness, we only allow the vertex to move along the surface

normal vector. This means we search a scalar value t such that

$$H(q_i^{k+1}) = \tilde{H}_i \;\text{ with }\; q_i^{k+1} = q_i^k + t\vec{n}. \tag{13}$$

If we take a look at the mean curvature discretization algorithm presented in section 6.1, we find that the matrix A does not change if we move q_i along the normal vector. So only the right side of equation (11) is influenced by such a motion and for the normal curvatures (8) we obtain

$$\tilde{\kappa}_j = 2\frac{\left\langle q_j - q_i - t\vec{n} \middle| \vec{n} \right\rangle}{\left\langle q_j - q_i - t\vec{n} \middle| q_j - q_i - t\vec{n} \right\rangle}.$$

This normal curvature discretization is nonlinear in t, but since the position of q_i will not change much during the update step, we assume the distance between q_i and q_j to remain constant. With this linearization technique we get

$$\tilde{\kappa}_j \approx 2\frac{\left\langle q_j - q_i \middle| \vec{n} \right\rangle}{\left\langle q_j - q_i \middle| q_j - q_i \right\rangle} - 2t\frac{1}{\left\langle q_j - q_i \middle| q_j - q_i \right\rangle}.$$

Using this assumption, the discretization of $H(q_i^{k+1})$ determined by (10) and (11), becomes a linear function in t. Solving this linear equation for t, we finally update $q_i^{k+1} = q_i^k + t\vec{n}$ which approximately solves (13).

7.3 Inner fairness condition

To control how the vertices are distributed on the surface of the solution, for every inner vertex $q_i \in V_I$ we assign a generalized discrete laplacian Δ that determines the local parameterization of the surface. Assigning a scalar weight λ_{ij} to every $q_j \in N(q_i)$ with the constraint $\sum_j \lambda_{ij} = 1$, the discrete laplacian Δ is defined as

$$\Delta(q_i) = -q_i + \sum_{q_j \in N(q_i)} \lambda_{ij} q_j.$$

A mesh M is said to satisfy the inner fairness condition, if all $\Delta(q_i)$ have vanishing tangential components, so that there are scalar values t_i such that $\Delta(q_i) = t_i\vec{n}_i$ for all $q_i \in V_I(M)$.

We integrate the inner fairness condition into the construction algorithm by including it into the update step described in section 7.2. Instead of updating $q_i^k \to q_i^{k+1}$ along the line $q_i^k + t\vec{n}_i$, we update along the line $\tilde{q}_i^k + t\vec{n}_i$, where $\tilde{q}_i^k$ is the projection of $\sum \lambda_{ij} q_j$ onto the plane defined by q_i^k and $\vec{n}_i$

$$\tilde{q}_i^k = \sum_{q_j \in N(q_i)} \lambda_{ij} q_j + \left\langle \Delta(q_i) \middle| \vec{n}_i \right\rangle \vec{n}_i.$$

In our examples we used two different generalized laplacians Δ. If we assume local uniform parameterization, we arrive at a laplacian with weights $\lambda_{ij} = \frac{1}{m}$, where m is the valence of the vertex q_i. This laplacian leads to meshes that have a regular vertex distribution on the surface. However, in some important cases such a kind of mesh parameterization is not wanted. For example in multiresolution mesh modeling (Kobbelt et al., 1998b; Guskov et al., 1999), one would like to have a faired mesh that minimizes local distortions when compared to the original mesh M^0.

For meshes it is usually not possible to get a map that is conformal in the sense that angles are preserved, but we can use the fact that the discretization of $\Delta_B f = 0$ can be interpreted as a discrete harmonic map (Pinkall and Polthier, 1993; Duchamp et al., 1997) and thus approximates a continuous conformal map. So to minimize local distortions, we use the weights resulting from the discretization of the Laplace-Beltrami operator of the original mesh M^0. The weights λ_{ij} at the vertex q_i are then given by

$$\lambda_{ij} = \frac{\cot\, \alpha_j + \cot\, \beta_j}{\sum_{q_j \in N(q_i)} (\cot\, \alpha_j + \cot\, \beta_j)},$$

where α_j and β_j are the triangle angles as shown in figure 3.

Other promising parameterizations are e. g. the *weighted least square* and the *shape-preserving* parameterizations presented by Floater (1997). Compared to the discrete conformal parameterization they have the advantage to be based on convex combinations only.

7.4 Multigrid approach

It is well known, that the convergence of mesh fairing algorithms can be dramatically accelerated, if multigrid techniques are integrated into the construction process (Kobbelt, 1997; Guskov et al., 1999). In our implementation we followed this idea. An overview of the algorithm is shown in figure 6. The necessary hierarchy levels are constructed using the progressive mesh approach

- Create a progressive mesh representation of the given mesh and subdivide the vertex splits into hierarchies
- Solve the fairing problem on the base mesh
  - Create a smooth initial mesh using an algorithm that can handle noisy meshes
  - Solve intrinsic fairing problem on the base mesh
- For each hierarchy level
  - Add vertices of the progressive mesh representation until the next hierarchy level is reached as described in section 7.4
  - Smooth the complete mesh on the current hierarchy level. When determining the estimated new mean curvature distribution, alternate between local and global iterative schemes
    - local: Determine the $\tilde{H}_i$ values using one Gauss-Seidel iteration and update the vertices
    - global: Determine the $\tilde{H}_i$ values using n conjugate gradient iterations (with diagonal preconditioner) and update the vertices

Fig. 6. The multigrid fairing algorithm

(Hoppe, 1996) with half-edge collapses (Kobbelt et al., 1998a). However, instead of reducing a mesh while trying to keep the details, we are more interested in creating a mesh whose smallest edge length is maximal while avoiding distorted triangles (long triangles with small inner circle). The number of hierarchy levels can be specified by the user. A reasonable strategy to fix the number of vertices per hierarchy is exponential growth.

Our multigrid algorithm exploits the fact that a coarse mesh already approximates the shape of the smooth surface that is implicitly defined by the PDE, increasing the mesh size mainly improves the smoothness of the approximation (Fig. 2 and 8). Therefore, we start with the construction of a discrete solution on the coarsest level of the progressive mesh representation and then each solution on a coarse level serves as starting point for the iteration algorithm on the next finer hierarchy level. Between two hierarchy levels we need a prolongation operator that introduces new vertices using the vertex split information of the progressive mesh. When adding a new vertex q_i, we have to take care that the outer fairness is not destroyed at that position. This is achieved in three steps, where the first two steps are similar to the prolongation operator used in (Guskov et al., 1999):

- First we update the mesh topology and introduce q_i at the position given

by its inner fairness criterion

$$q_i = \sum_{q_j \in N(q_i)} \lambda_{ij} q_j$$

- In some cases the first step is not enough to avoid triangle distortions, therefore in the second step we further update the complete 1-ring of q_i. This means we solve the local linear problem $\Delta q_l = 0$ for all $q_l \in D(q_i)$.
- Since the second step disturbs the outer fairness, we finally solve $\Delta_B H_l = 0$ for all $q_l \in D(q_i)$ by applying the construction algorithm locally on the 1-disk $D(q_i)$.

Our mean curvature discretization (section 6) as well as the update step (section 7.2) assume that the mesh is not a noisy surface. Therefore, before starting the multigrid algorithm we first construct at the coarsest level the linear solution of the problem $\Delta^2 f = 0$ using the laplacian defined by the chosen inner fairness criterion (see Fig. 1 c). We further developed an improved bilaplacian mesh fairing algorithm (Schneider et al., 2000b) that produces an initial mesh with superior quality than those resulting from linear schemes. For sparse meshes with very irregular structure, this scheme is much better suited to create an initial mesh (Fig. 7). Later at each hierarchy level our mesh is already presmoothed by the multigrid fairing concept.

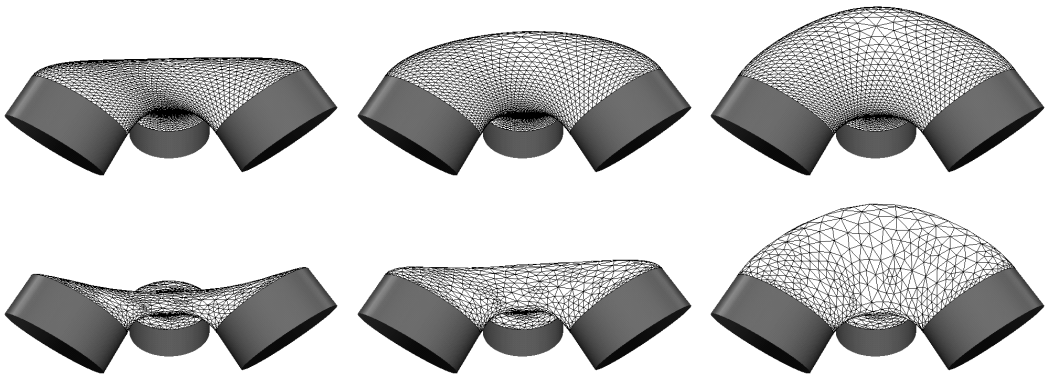


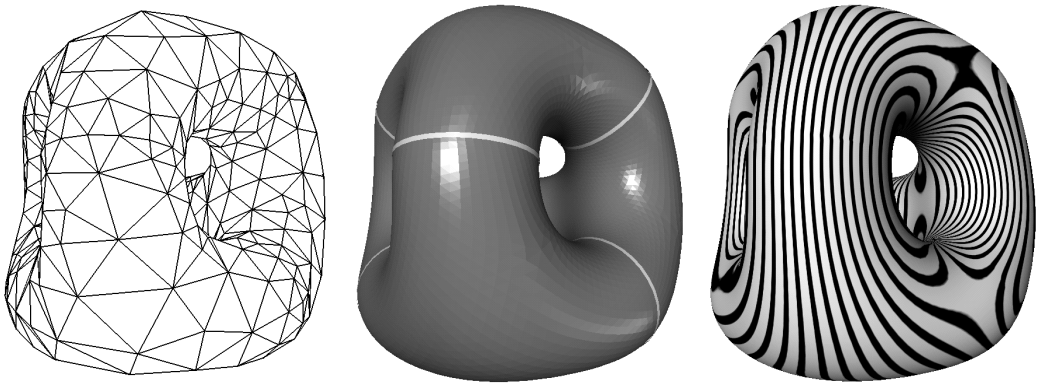
Fig. 7. Comparison of the results from a linear bilaplacian (left), our improved bilaplacian (middle) and our intrinsic method. In the upper row the mesh has a regular subdivision connectivity structure and in the lower row it is sparse and has a very irregular connectivity

7.5 Examples and remarks

Because of the importance of the discrete mean curvature at the boundary vertices for the construction algorithm and the fact that at such points we only have one-sided vertex information, in our examples we calculated the mean curvature at every boundary vertex using the special case discretization

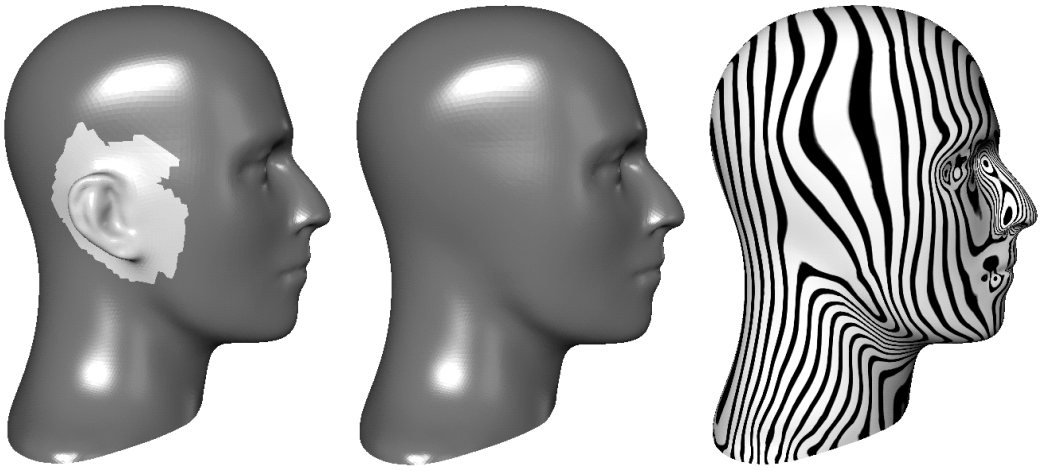
Table 1
Running times of the intrinsic fairing algorithm

Dataset	Vertices complete mesh	Triangles complete mesh	Vertices base mesh	Time sec.
cylinder blend	920	1746	140	1.9
bust face	2926	5720	256	12
human ear	4665	9220	248	21
tetra thing	13308	26624	308	57



(a) (b) (c)

Fig. 8. 6 circles are used to define a tetra thing. a) and b) show the solution of a mesh with 500 resp. 13308 vertices. c) shows the reflection lines of the mesh b). Due to the symmetric constellation the final smooth solution would be G^2 continuous.



(a) (b) (c)

Fig. 9. Feature removal of a mesh with user defined boundary curve.

as for inner vertices of valence 3 or 4. For coarse meshes, we also used the special case handling for interior vertices of higher valence.

Instead of updating $q_i^{k+1} = q_i^k + st\vec{n}$ with $s = 1.0$ as described in section 7.2, in practice s has to lie in the range $0 < s < 1.0$. While $s = 1.0$ usually will work well, we noticed that in some rare constellations it is possible that vertices can alternate between two positions. All of our examples presented in this paper were constructed using $s = 0.9$.

The conformal inner fairness is only hardly more time consuming to construct than the uniform parameterization, but requires much more memory. Here the original mesh has to be reconstructed from the progressive mesh representation parallel to the faired mesh to be able to update the necessary coefficients of the laplacian – which also have to be stored in memory – at every hierarchy level.

The lines of reflection in our example pictures indicate G^1 continuity at the boundary (continuous lines) and at least G^2 continuity in the interior (smooth lines). As can be seen in examples 9 and 10 the boundary does not have to be smooth. In such cases the fixed unit normal information at the boundary is derived by averaging the adjacent triangle normal vectors before the construction.

The concept behind our construction algorithm works best for $\Delta_B H = 0$, but it is not restricted to that. We also constructed MESs by solving their Euler-Lagrange equation $\Delta_B H = -2H(H^2 - K)$ and surfaces satisfying $\Delta_B H = const$. However, these inhomogen problems are more costly to solve and a stable construction algorithm is more involved.

In this paper we assumed that the final mesh should have a prescribed vertex connectivity. If this is not the case, flipping of edges should be enabled to avoid long triangles, new vertices may be introduced where necessary or removed where the vertex density becomes too high (see Hsu et al., 1992; Welch and Witkin, 1994).

All shaded pictures are flat shaded to enhance the faceting effect. The construction algorithm was implemented in Java 1.3 and Java 3D for Windows NT running on a Pentium III with 500MHz. We noticed that our fairing algorithm is very fast for a fairing approach based on intrinsics. The running times for the examples shown in this paper are presented in table 1, they do not include the time needed to build the progressive mesh representation. An optimized C implementation would improve these timings without doubt.

8 Conclusion and future work

The technique we presented in this paper shows that it is possible to use higher order intrinsic PDEs in modeling based on irregular meshes. Since the surface is completely defined by G^1 boundary conditions and a PDE, the designer does not have to take surface parameterization or the actual mesh representation into account and can freely distribute the vertices on the surface.

One of our future plans is to integrate the presented technique in a multiresolution mesh modeling tool as those described in (Kobbelt et al., 1998b). Although the new construction algorithm is obviously slower than the simple linear fairing algorithm used there, based on the results we got from our current implementation, it seems possible to achieve interactive frame rates with a decent mesh complexity. In combination with an inner fairness criterion that enables local shape preservation, this would lead to a much simpler detail coding. Because of the intrinsic surface definition, this would also simplify a mesh modeling that does not fix the connectivity or the mesh size during the modifications.

We will also explore other curvature and update algorithms and study different intrinsic PDEs to introduce new design freedoms. Furthermore, it will be necessary to integrate surface constraints in the interior of the surface such as interpolation of points, which might require PDEs of higher order.

Acknowledgements

We thank Jens Vorsatz and Mario Botsch for their substantial contributions to the creation of our examples and also Kolja Kähler for providing the progressive mesh algorithm.

References

Bloor, M. I. G., and Wilson, M. J. (1990), Using partial differential equations to generate free-form surfaces, Computer-Aided Design 22, 202–212.

Brakke, K. A. (1992), The Surface Evolver, Experimental Mathematics 1 (2), 141–165.

Burchard, H. G., Ayers, J. A., Frey, W. H. and Sapidis, N. S. (1994), Approximation with Aesthetic Constraints, in: Sapidis, N. S., ed., *Designing Fair Curves and Surfaces*, SIAM, Philadelphia.

do Carmo, M. P. (1993), *Differential Geometry of Curves and Surfaces*, Prentice-Hall, Inc Englewood Cliffs, New Jersey.

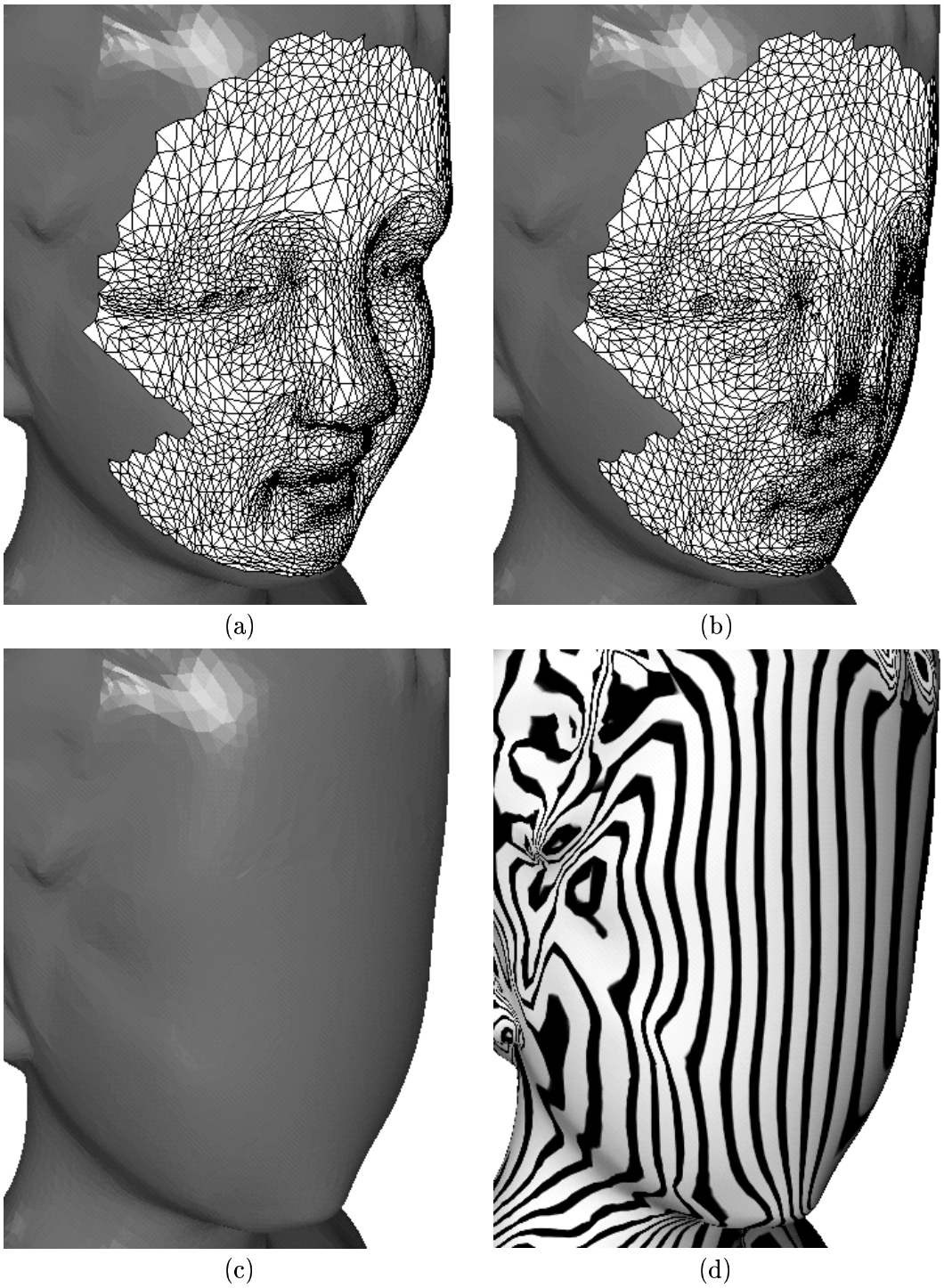


Fig. 10. Here we smoothed the face of a bust model shown in a). The triangulation of the solution in this example is discrete conformal to the initial mesh.

Chopp, D. L., and Sethian, J. A. (1999), Motion by Intrinsic Laplacian of Curvature, Interfaces and Free Boundaries 1, 1–18.

Desbrun, M., Meyer, M., Schröder, P. and Barr, A. H. (1999), Implicit fairing of irregular meshes using diffusion and curvature flow, SIGGRAPH '99

Conference Proceedings, 317–324.
Desbrun, M., Meyer, M., Schröder, P. and Barr, A. H. (2000), Discrete Differential-Geometry Operators in nD, submitted for publication.
Duchamp, T., Certain, A., DeRose, T. and Stuetzle, W. (1997), Hierarchical computation of PL harmonic embeddings. Technical report, University of Washington.
Floater, M. S. (1997), Parametrization and smooth approximation of surface triangulations, Computer Aided Geometric Design 14, 231–250.
Golub, G. H. and Van Loan, C. F. (1989), *Matrix Computations*, Johns Hopkins University Press, Baltimore.
Greiner, G. (1994), Variational design and fairing of spline surfaces. Computer Graphics Forum 13(3), 143–154.
Guskov, I., Sweldens, W. and Schröder, P. (1999), Multiresolution Signal Processing for Meshes, SIGGRAPH '99 Conference Proceedings, 325–334.
Hoppe, H. (1996), Progressive meshes, SIGGRAPH '96 Conference Proceedings, 99–108.
Hsu, L., Kusner, R., and Sullivan, J. (1992), Minimizing the Squared Mean Curvature Integral for Surfaces in Space Forms, Experimental Mathematics 1 (3), 191–207.
Kobbelt, L. (1997), Discrete fairing, Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces, 101–131.
Kobbelt, L., Campagna, S. and Seidel, H.-P. (1998a), A General Framework for Mesh Decimation, Proceedings of the Graphics Interface conference, 43–50.
Kobbelt, L., Campagna, S., Vorsatz, J. and Seidel, H.-P. (1998b), Interactive Multi-Resolution Modeling on Arbitrary Meshes, SIGGRAPH '98 Conference Proceedings, 105–114.
Marciniak, K., and Putz, B. (1984), Approximation of spirals by piecewise curves of fewest circular arc segments, Computer-Aided Design 16, 87–90.
Meek, D. S. and Walton, D. J. (2000), On surface normal and Gaussian curvature approximations given data sampled from a smooth surface, Computer Aided Geometric Design 17, 521–543.
Moreton, H. P. and Séquin, C. H. (1992), Functional optimization for fair surface design, SIGGRAPH '92 Conference Proceedings, 167–176.
Ohtake, Y., Belyaev, A. G. and Bogaevski, I. A. (2000), Polyhedral Surface Smoothing with Simultaneous Mesh Regularization, Geometric Modeling and Processing 2000 Proceedings, 229–237.
Pinkall, U. and Polthier, K. (1993), Computing discrete minimal surfaces and their conjugates, Experimental Mathematics 2 (1), 15–36.
Schneider, R. and Kobbelt, L. (1999), Discrete Fairing of Curves and Surfaces based on linear Curvature Distribution, Curve and Surface Design - Saint-Malo '99 Proceedings, 371–380.
Schneider, R. and Kobbelt, L. (2000a), Generating Fair Meshes with G^1 Boundary Conditions, Geometric Modeling and Processing 2000 Proceedings, 251–261.

Schneider, R., Kobbelt, L. and Seidel, H.-P. (2000b), Improved Bilaplacian Mesh Fairing, submitted for publication.

Taubin, G. (1995a), A signal processing approach to fair surface design, SIGGRAPH '95 Conference Proceedings, 351–358.

Taubin, G. (1995b), Estimating the tensor of Curvature of a surface from a polyhedral approximation, ICCV '95 Proceedings, 902–907.

Weimer, H. and Warren, J. (1998), Subdivision Schemes for Thin Plate Splines, Computer Graphics Forum 17, 303–314.

Welch, W. and Witkin, A. (1994), Free-Form shape design using triangulated surfaces, SIGGRAPH '94 Conference Proceedings, 247–256.