Feature preserving mesh smoothing algorithm based on local normal covariance

Miroslav Svub, Premysl Krsek, Michal Spanel,
Vit Stancl, Radek Barton, Jiri Vadura
{svub,krsek,spanel,stancl,barton,ivadura}@fit.vutbr.cz
PGMed@FIT
Department of Computer Graphics and Multimedia
Faculty of Information Technology
Brno University of Technology, Brno, Czech Republic

ABSTRACT

Our goal is to develop a smoothing algorithm, which would be feature preserving and simple to use without the need of extensive parameter tuning. Our method does the smoothing of vertices based on local neighbourhood character, which is modeled by a covariance matrix of neighbourhood triangle normals. The eigenvalues and eigenvectors of the covariance matrix are used for local weighting of the displacement vector of laplacian operator. This way the method is locally auto-tuned.

Keywords: smoothing, polygonal mesh, covariance, edge preserving, noise removal

1 INTRODUCTION

One of the key phases of modelling a 3D polygonal mesh is smoothing. It can substantially reduce the amount of artifacts and noise within the mesh. A rather important features of a smoothing algorithm is how it treats different mesh features. A *feature preserving algorithm* should leave **corners** and **sharp edges** untouched while smoothing and flattening the other areas of the mesh.

Different algorithms have approached this with varying success and there is no general algorithm, which works reliably in all cases. Moreover, the best algorithms often require tuning of several parameters. Our approach attempts to address these problems and offer a simple to use, yet relatively powerful smoothing method.

1.1 Related work

One of the basic approaches to smoothing is *laplacian operator* [7] which works by averaging position of the vertex with it's neighbourhood, defining a vector by the previous and average position and then moving the vertex in this direction by a fraction of the vector length. The algorithm is very simple and is easily tunable by only one parameter. The main disadvantages are volume shrinking and reducing sharp edges and corners. Therefore, An *improved laplacian smoothing* algorithm presented in [1], which improves the results of laplacian operator by pushing the smoothed vertex a bit back, thus reducing the volume shrinking effect. The performance of this algorithm can be adjusted by two parameters. Another improvement was presented in [3] and [4] which operates in alternating inward and outward diffusion of vertices in order to maintain the shape of the mesh. Again, the algorithm is controlled by two parameters. The *billateral mesh denoising* approach from [5] and similar method from [6] has been quite successful. It is essentially a bilateral filter applied on a mesh topology and works by filtering vertex positions in directions of their normals. Adjustable filter parameters can affect the output.

2 VERTEX PROPERTIES FROM NORMAL COVARIANCE MATRIX

In this section we will describe the background for our method. The main idea lies in altering the *laplacian operator* effect for each vertex type **corner**, **edge**, **flat areas**. First, let us define some basic symbolics. Let v denote the vertex that is being smoothed and n it's normal. Consequently let t_i and n_i denote i-th face(triangle) and normal of v's neighbourhood. Let N be a matrix with n_i forming it's columns.

Let us take a look at matrix C :

$$C = NN^T$$

We can see that this is a zero-mean covariance matrix (covariance matrix for average normal zero $(N-0)(N-0)^T$) of normals in the neighbourhood of v. This matrix has been used in [2] for construction of *quadric error metric* and was shown to have a couple of it's interesting properties. C presents us with a robust local vertex feature, which reflects the character of the vertex neighbourhood. The zero-mean covariance C matrix can be interpreted as a *quadratic form* defined for v and forms a quadric centered at v. Since quadric matrices are always symmetric, we can perform it's spectral decomposition into A and X, where A's columns are the eigenvectors of C and λ_i are the corresponding eigenvalues :

$$C = AXA^{-1}, where X = diag(\lambda_1, \lambda_2, \lambda_3)$$

Now let's take a look at the *eigenvalues* of C. We can distinguish three main cases :

- if $\lambda_1 \cong \lambda_2 \cong \lambda_3$, then there is a large variation of normals in every direction around the vertex and it can be assumed to be a **corner**.
- if $\lambda_1 \cong \lambda_2 < \lambda 3$, then there is a direction of maximum covariance of neighbourhood normals and the direction corresponds to an **edge**.
- if $\lambda_1 < \lambda_2 \cong \lambda_3$, then there are two directions of maximum covariance and the vertex lies on a **flat arrea**.

Let e_1, e_2, e_3 denote the *eigenvectors* of C. In the second case, the e_3 is aligned with the direction of the edge. In the third case, e_3 will be perpendicular to the ideal plane of the flat area. Let us define vertex v' which is the average of vertices from a neighbourhood of size M.

$$v' = \frac{1}{M}\sum_{i=0}^{M} v_i$$

Then $w = v - v'$ would be the vector along which the vertex would move using the *laplacian operator*.

Now the essential idea of our smoothing method is that instead of $\hat{v} = v + \alpha w$ as in laplacian smoothing, we compute the new position of v as :

$$\hat{v} = v + \alpha AYA^{-1}w, Y = diag(\lambda_1^{-1}, \lambda_2^{-1}, \lambda_3^{-1})$$

In the following chapter, we will explain the motivation for and interpretation of the formula.

3 USING VERTEX NORMAL COVARIANCE FOR SMOOTHING

Consider a vertex and it's zero mean local covariance matrix C. The eigenvectors of C form an orthogonal basis with the center at the current vertex. We have designed following procedure for smoothing of a current vertex v (see figure 1 graphical interpretation of the scheme in 2D) :

1. compute position difference vector w after applying the laplacian operator
2. compute eigenvalues and eigenvectors of zero mean covariance matrix for v's neighbourhood. We have been using the neighbourhood of 3 surrounding triangle layers, because if less layers are used, the covariance statistic is less robust.
3. express w within the basis formed by eigenvectors of C to obtain w' expressed in new basis coordinates
4. weight the coordinates of w' by inverse eigenvalues of C to obtain w'_1

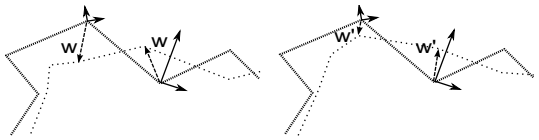

Figure 1: left : w denotes the vector to the new position of vertex, right : w' denotes the same vector after having its coordinates weightened by eigenvalues of C in the local coordinates of C's eigenvectors

5. multiply w'_1 by the smoothing factor α, which regulates the amount of smoothing
6. express w'_1 back within the canonical basis of three dimensional space to get the final displacement vector for the current vertex

These steps are to be performed on the whole mesh. Our algorithm is also iterative, so multiple runs are often required to achieve desired result.

3.1 Improving the eigenvalue weighting by heuristic

When we look at our weighting formula for a vertex, we can see, that we are using the inverse eigenvalues of C as the weights for *laplacian operator*. This works in theory, however on real models, we have to adjust the weighting so that extreme values ad therefore extreme deformations are not permitted. Let f_w denote a function mapping a three dimensional vector onto another. Let λ denote the vector of eigenvalues. Then, the smoothing formula will take the following form :

$$\hat{v} = v + \alpha AYA^{-1}w, Y = diag(f_w(\lambda))$$

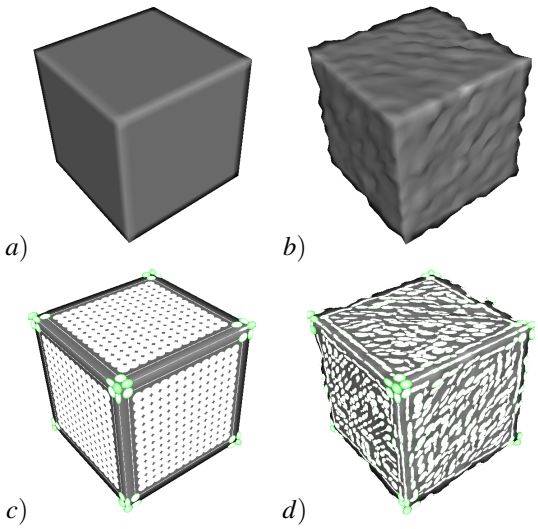

Figure 2: visualizing vertex weights a) original mesh b) covariances as degenerated elipsoids c) noisy mesh d) visualization on noisy mesh

The weighting function we have been using is a heuristic. The function is $f_w(\lambda_1, \lambda_2, \lambda_3) = normalize(\lambda)^{-\frac{2}{3}})$. The normalization of eigenvalues will ensure, the weights sum to one and the vertex displacement will not be greater than using the standard *laplacian operator*. The $-\frac{2}{3}$ power reduces the large differences between weights for a vertex.

3.2 Vizualizing vertex weights

The algorithm therefore treats directions of eigenvector differently. The smaller the smaller the weight corresponding to a direction is, the stiffer will be the movement of this vertex in that particular direction. Figure 1 shall give us better idea behind the algorithm. Suggested method of visualizing C can be found in [2]. C is the main part of *quadric* - an ellipsoid centered at the current vertex and with it's principal axes aligned with the eigenvectors of C and its principal radii proportional to the inverse eigenvalues. We can use the same principle but the ellipsoid will be slightly deformed by our weighting finction f_w.

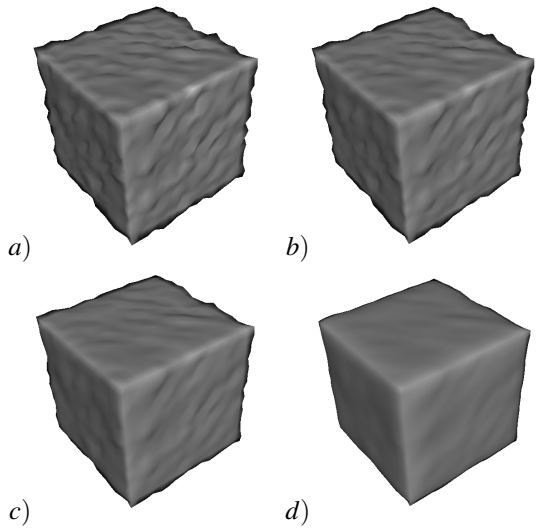

Figure 3: iterations of our algorithm on cube model a) original mesh b) iteration 1 c) iteration 3 d) iteration 7

Figure 2 shows the visualization. At the corner of the cube (green), the ellipsoid resembles a sphere. This means, that the weights are close to being equal in all directions and the movement of this point should be restricted the most. At an edge vertex, the ellipsoid is reduced to elongated line, which is aligned along the direction in which the vertex can move without damaging the edge. At a flat area vertex, the ellipsoid reduces to a disc. The left set of images show a cube that is no longer regular, but we can see, that the ellipsoid characteristics still hold. Figure 3 shows the result of smoothing the noisy cube using our algorithm.

4 EXPERIMENTAL RESULTS

We have tested our method and compared it's results to the algorithms presented in section 1 ([4], [5], [1]). We had a collection of 3D meshes, which include geometrical primitives (cube, sphere, cylinder, torus, etc.) to test the feature preservation properties and a collection of real models (Stanford bunny, Stanford dragon, skull model, etc). The rough data for the smoothing algorithm were obtained by adding a *gaussian noise* to the mesh. For the σ^2 of the noise, we have chosen values between 0.05 to 0.1 times the mesh size. The noise was added in normal direction (see fig. 3). Each model was well tessellated.

The metric chosen for algorithm performance evaluation was histogram based. Since the original, noisy and smoothed models are topologically equivalent we have chosen following metrics :

- Histogram of *angles between normals of corresponding faces*. The method, which outperforms the others should have minimal amount of corresponding normal angle differences (H_N).
- Histogram of *distance between corresponding vertices*. This metric reflects the ability of the method to preserve the volume of the model (H_V).

For the methods that require parameter tuning, we have adjusted the parameters to perform as good as possible for every particular model.

4.1 Cube model analysis

The figure 4 Shows the results obtained on the cube model. The $H_N(original)$ shows the high peak of zero angle flat vertices and a small peak of edge vertices. The slim and high peak can be seen on $H_N(our)$ is showing, that our method was able outperform the others as far as edge preservation is concerned.

The $H_N(bmds)$ shows that the bilateral mesh denoising algorithm can be viewed as the second best as far as edge preservation is concerned. The $H_N(our)$ shows, that average of normal deviations is shifted towards zero deviation. On this particular model, the best observed shape preserving was achieved by our method.

The figure 5 shows H_V histograms of the cube model. According to this metric, our method is outperformed by *hc* and *tabuin*. If we compare $H_V(our)$ with $H_V(bmds)$, we see that better volume preservation was achieved with our method, than with *bmds*. Figure 9 shows visual comparison of the results on the cube model.

4.2 Real model analysis

The cube model discussed above was an example from the set of ideal model with clearly distinguishable features. We have tested our method on Following histograms were obtained from the models of *the Stanford bunny*. Figure 6 shows that the shape similarity

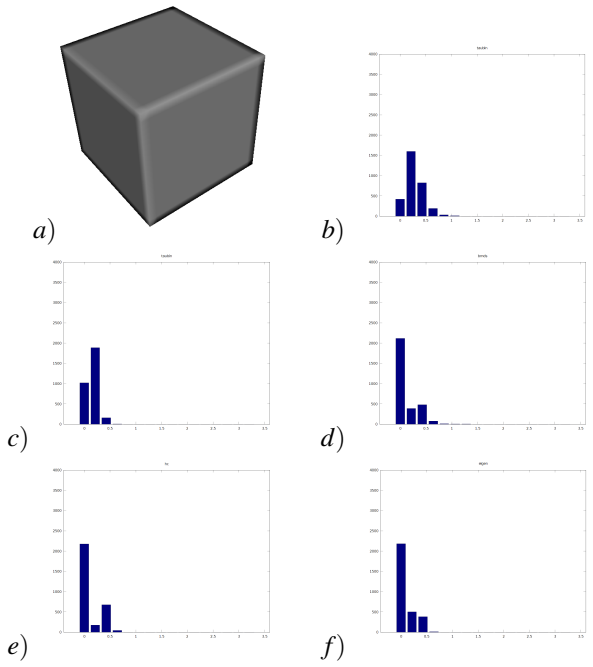

Figure 4: H_N Histogram of corresponding normal angle differences a) the model b) $H_N(noisy)$ c) $H_N(taubin)$ d) $H_N(bmds)$ e) $H_N(hc)$ f) $H_N(our)$

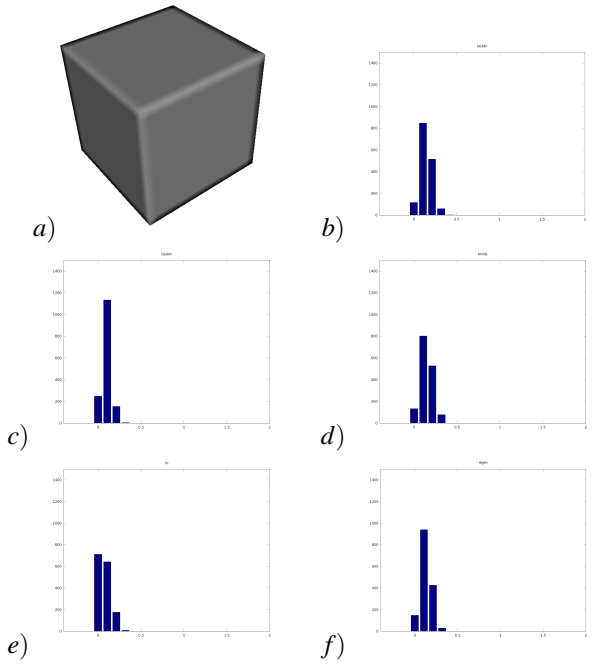

Figure 5: H_V Histogram of distances between corresponding vertices a) the model, b) $H_V(noise)$, c) $H_V(taubin)$, d) $H_V(bmds)$, e) $H_V(hc)$, e) $H_V(our)$

of $H_N(original)$ and $H_N(our)$ is good (see fig. 8 for visual comparison).

The comparison of H_V histograms (figure 7) shows that the best volume preservation was achieved by *hc* algorithm. *our* method has shown comparable results with *taubin* method and clearly outperformed *bmds*.

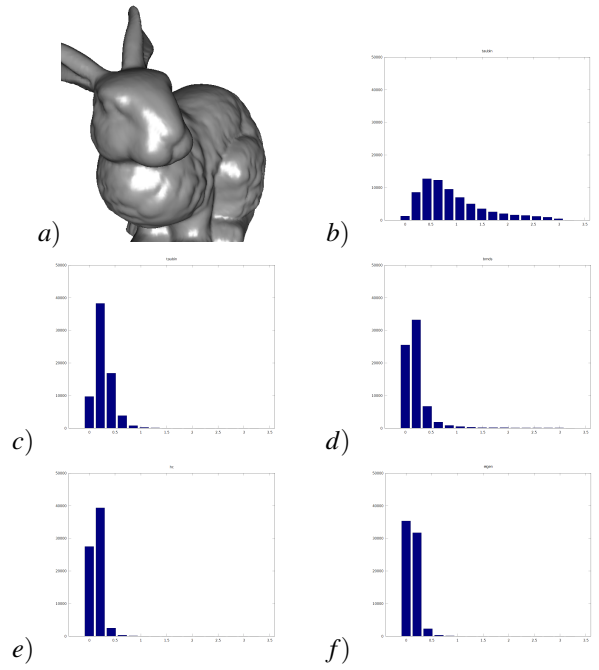

Figure 6: H_N Histogram of corresponding normal angle differences a) the model, b) $H_N(noisy)$, c) $H_N(taubin)$, d) $H_N(bmds)$, e) $H_N(hc)$, f) $H_N(our)$

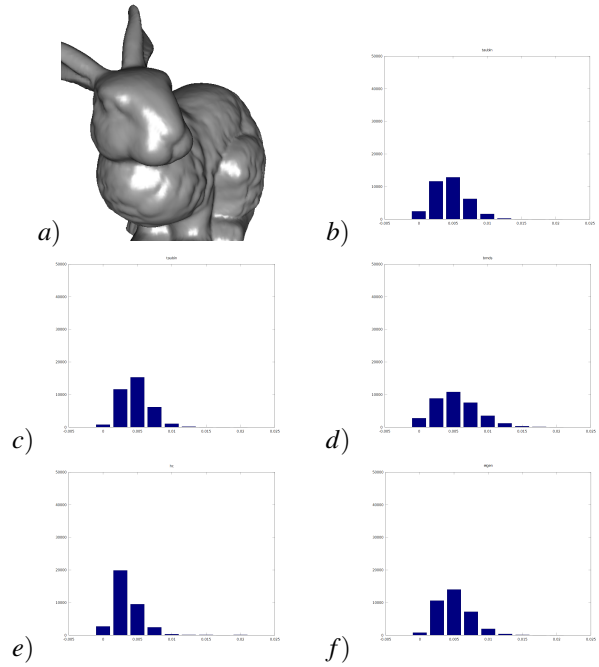

Figure 7: H_V Histogram of distances between corresponding vertices a) the model, b) $H_V(noisy)$, c) $H_V(taubin)$, d) $H_V(bmds)$, e) $H_V(hc)$, f) $H_V(our)$

4.3 Summary

Tables 1 and 2 sumarizes the results on several models we have used. In order to quantify the results, we have reduced the histogram metric to computing mean and standard deviation of the data. The key idea is, that if the original model and the smoothed one are similar, then the angles between corresponding face normals should be small and the distance between corresponding vertices should be small as well.

	taubin	bmds	hc	our
cube (μ)	0.156	0.13	0.148	0.117
cube (σ^2)	0.09	0.181	0.174	0.125
cylinder (μ)	0.106	0.076	0.084	0.084
cylinder (σ^2)	0.103	0.166	0.157	0.155
bunny (μ)	0.270	0.216	0.143	0.128
bunny (σ^2)	0.171	0.228	0.094	0.103
dragon (μ)	0.214	0.277	0.17	0.182
dragon (σ^2)	0.252	0.335	0.231	0.274
bull (μ)	0.121	0.145	0.143	0.11
bull (σ^2)	0.114	0.165	0.083	0.108
skull (μ)	0.38	0.53	0.25	0.267
skull (σ^2)	0.401	0.62	0.256	0.302

Table 1: Angles of corresponding face normals(rad)

For the models to be similar in shape, we need the mean angle difference to be small. The edge preserving ability manifests itself through the standard deviation of angles (this can be observed on ideal meshes). For the models to be similar in volume, we expect the corresponding vertices distance to be small. Therefore the average distance, the better the models correspond volume-wise.

	taubin	bmds	hc	our
cube (μ)	0.099	0.146	0.076	0.129
cube (σ^2)	0.045	0.07	0.056	0.06
cylinder (μ)	0.067	0.073	0.075	0.071
cylinder (σ^2)	0.035	0.038	0.046	0.038
bunny (μ)	0.046	0.053	0.034	0.05
bunny (σ^2)	0.021	0.032	0.022	0.023
dragon (μ)	0.031	0.044	0.033	0.031
dragon (σ^2)	0.015	0.023	0.018	0.015
bull (μ)l	0.572	0.41	0.356	0.389
bull (σ^2)	0.255	0.2	0.19	0.192
skull (μ)	0.31	0.455	0.562	0.713
skull (σ^2)	0.747	0.277	0.346	0.446

Table 2: Distance between corresponding vertices

On the ideal models (cube, cylinder), the best shape preserving algorithms are *bmds* with *our* method being only slightly less effective. On real models, the best shape preserving methods are *hc* and *our* method. With the volume preservation, our algorithm outperforms *bmds* and is comparable with *taubin* and *hc*.

5 CONCLUSION

We have been trying to develop a smoothing method, that would aim to perform well on different types of meshes and would not require extensive amount of tuning to do so. Our method is not the best performing in all cases however it's performance is rather stable and comparable to the best methods for particular case. Another advantage of our method is it's robustness, since we are using broader vertex statistics (zero mean covariance).

5.1 Future work

We would like to focus on developing a robust smoothing method based on local normal covariance. Our approach currently extends the standard *laplacian operator* by locally modifying it's results to reflect local mesh characteristics. In the future, we would like to explore following possibilities for improving the method performance :

- Explicitly clustering the vertex characteristics (eigenvalues) into categories edge, corner, flat and designing and applying custom smoothing function based on vertex type.
- Finding connectivity between vertices of the same type and using the average computed from vertices of the same type.

We would also like to explore other applications of the vertex normal covariance feature, for example using it for other mesh related tasks (matching, registration, etc.)

REFERENCES

[1] Vollmer J. , Mencel R., Muller H.: Improved laplacian smoothing of noisy surface meshes, Research report No. 711 /1999, June 1999

[2] Garland M., Heckbert, P.: Surface simplification using quadric error metrics. In: Proceedings, Siggraph 97, USA, 1997, s. 209-216

[3] Taubin G.: A signal processing approach to fair surface design. In: Proceedings of SIGGRAPH 1995

[4] Taubin G.: Geometric signal processing on polygonal meshes: Eurographics 2000 State of The Art Report(STAR), September 2000.

[5] Fleishman, S., Drori I., Cohen-Or, D. : Bilateral mesh denoising. In: Proceedings of SIGGRAPH 2003

[6] Kai-Wah, L., Wen-Ping, W. : Feature-Preserving Mesh Denoising via Bilateral Normal Filtering, In: Proceedings of 9th International Conference on Computer Aided Design and Computer Graphics 2005

[7] Field, D.A. : Laplacian Smoothing and Delaunay Triangulations, Communications in Applied Numerical Methods, Wiley, Vol 4, pp.709-712, 1988

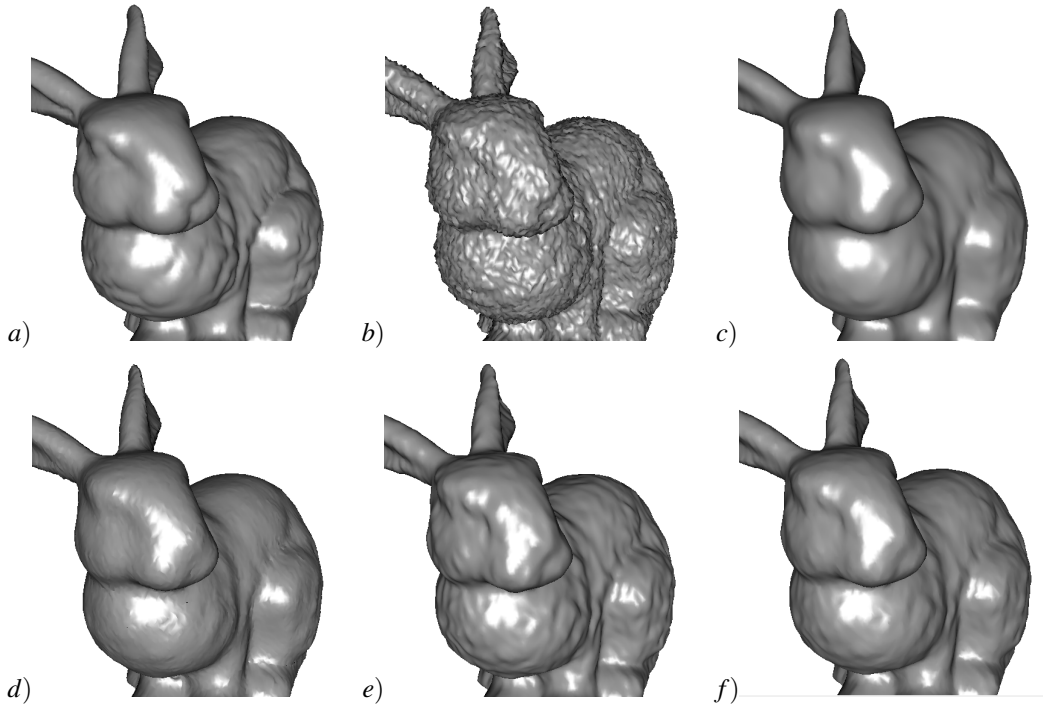

Figure 8: the stanford bunny a) *original* b) *noisy* c) *taubin* d) *bmds* e) *hc* f) *our*

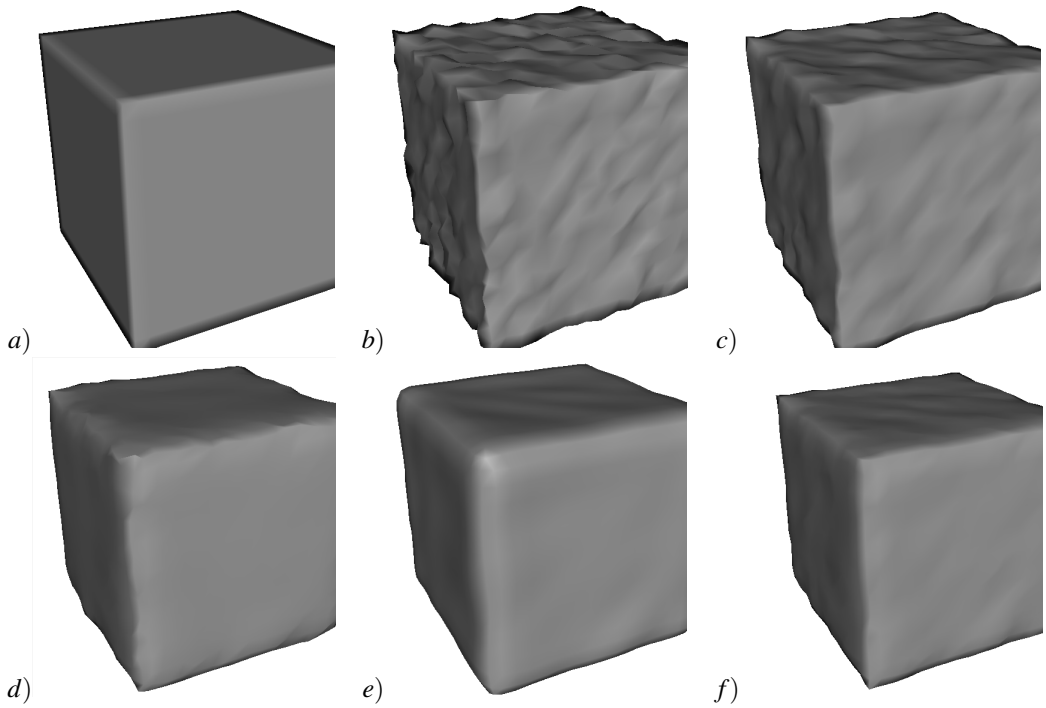

Figure 9: cube a) *original* b) *noisy* c) *taubin* d) *bmds* e) *hc* f) *our*

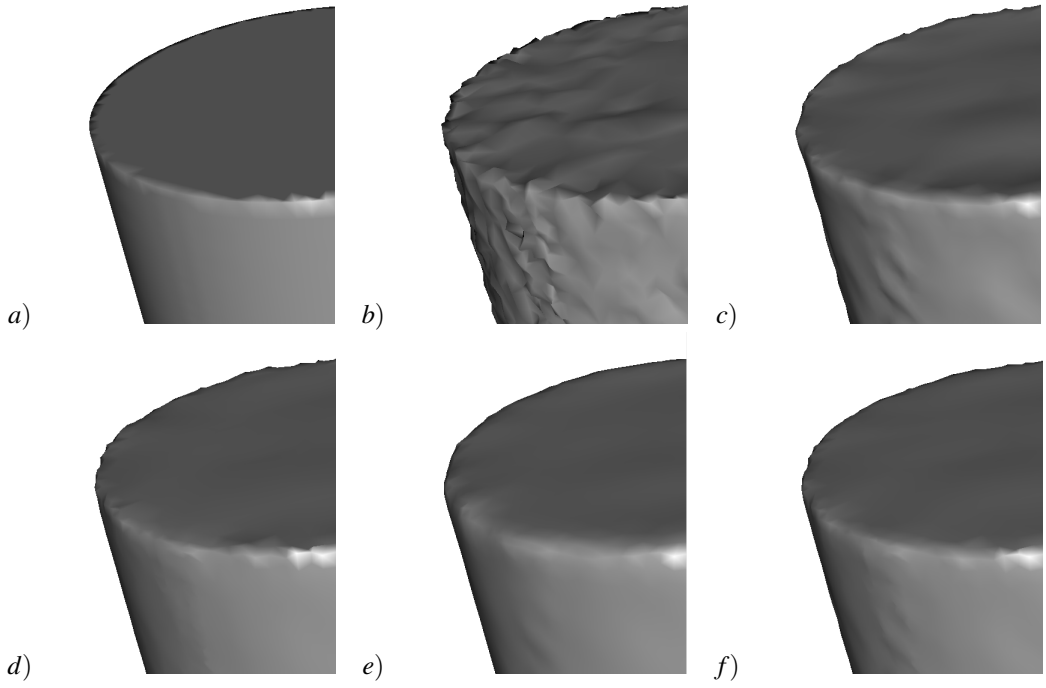

Figure 10: cylinder a) *original* b) *noisy* c) *taubin* d) *bmds* e) *hc* f) *our*

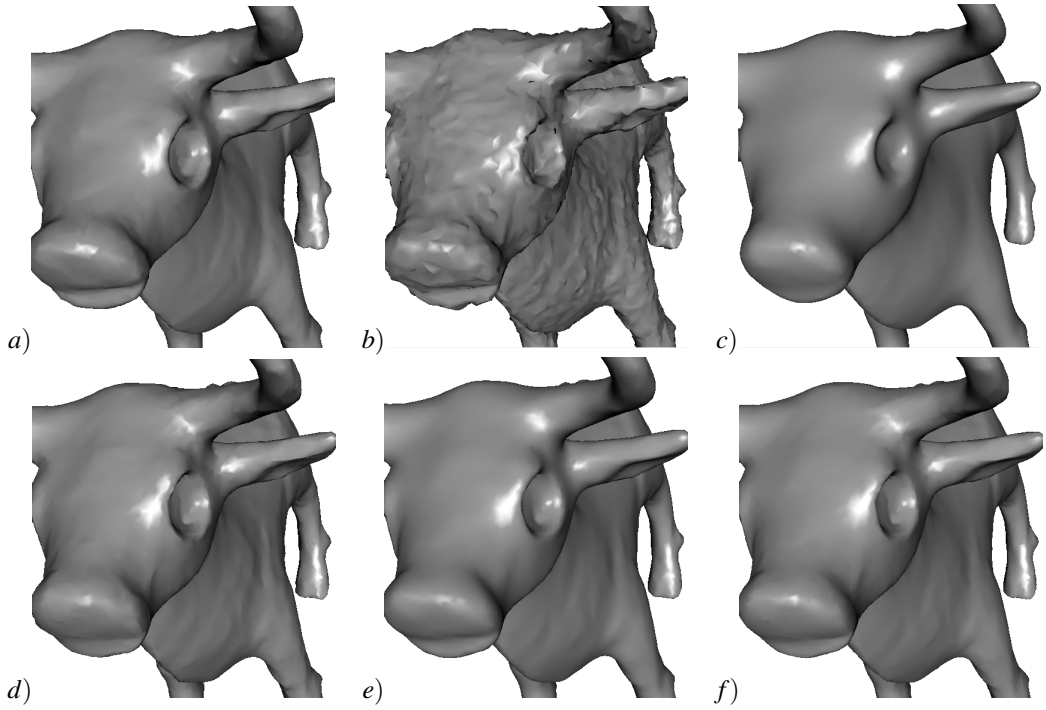

Figure 11: bull a) *original* b) *noisy* c) *taubin* d) *bmds* e) *hc* f) *our*