Bilateral Filtering for Mesh Denoising implementation

RAFAEL SENNA BENATTI, Institut Polytechnique de Paris, France

1 INTRODUCTION

This work presents an implementation of a bilateral filtering technique for mesh denoising[1]. The augmentation of use of 3D scanning tools to generate 3D meshes makes the denoising process of meshes really important, since this models may have noise from various sources. Between many techniques [1] provides a method based on moving the vertices along their normal directions to generate a smooth curvature in the surfaces. This method is derived from techniques of denoising used in image processing which use gaussian blur based on the intensity difference of neighbor pixels. Analagously, we can think in a plane generated by a point and its normal, and we can take the intensity difference as the distance of each neighbor from this plane.

2 RELATED WORKS

Some works relate different techniques of mesh smoothing. Taubin's smoothing method [2] utilizes two steps: the first step is a weighted average of vertex positions to shrink the mesh towards its center, and the second step is another weighted average that expands the mesh back out to its original size.

The algorithm has a number of adjustable parameters that allow the user to control the strength and number of iterations of the smoothing. By controlling these parameters, the user can control the balance between preserving the shape of the mesh and smoothing out noise.

Taubin's algorithm has some relation with the Fourier Descriptors method since it uses the property that the 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. This allows the extension of use of Fourier descriptors to the three-dimensional case.

Another approach developed by Peng et Al 2001 [3] uses the Wiener Filter in the images in order to smooth the surfaces which allowed great feature preservation. In this approach it's demanded to have a semi-regular mesh in order to work, also, the user needs to input the variance of the noise in the surface.

The paper "Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow" by Desbrun et al. [1999] [4] introduced a new class of mesh processing algorithms called geometric diffusion algorithms. The goal of these algorithms is to smooth or enhance the geometry of a mesh while preserving its topology and important features.

The basic idea behind these algorithms is to perform local operations on a mesh that spread information from one vertex to its neighbors. This process is similar to the diffusion of heat or fluid flow over a surface, hence the name "diffusion algorithms."

Recently, there are also methods using optimization approaches where , based on the input mesh information and a set of constraints defined by the real geometric information or the noise distribution prior, an energy equation can be formulated. Solving the optimization function is equivalent to the geometry smoothing process. Some of these papers are Cheng et al. "Feature-preserving filtering with l0 gradient minimization" 2014 [5] and Wang et al. "Decoupling noise and features via weighted l1-analysis compressed sensing" 2014 [6] which use minimization approaches in the L_0 and L_1 norms respectively.

There are also recent methods using machine learning approaches such as "Normalf-net: Normal filtering neural network for feature-preserving mesh denoising", 2020 [7].

Author's address: RAFAEL SENNA BENATTI, Institut Polytechnique de Paris, France.

3 TECHNICAL DETAILS

The bilateral mesh denoising is an anisotropic method for mesh denoising. The idea of the method is to dislocate a point along it's normal based on the distance its neighbors and the distance of the neighbors to the plane defined by a vertex and its normal. The method inheretly handle boundaries. We can see the pseudo-code of the algorithm in Fig. 1 The t is the euclidean distance between the

```
DenoisePoint(Vertex v, Normal n)
  {q_i} = neighborhood(v)
  K = |{q_i}|
  sum = 0
  normalizer = 0
  for i := 1 to K
    t = ||v − q_i||
    h = ⟨n, v − q_i⟩
    w_c = exp(−t^2/(2σ_c^2))
    w_s = exp(−h^2/(2σ_s^2))
    sum += (w_c · w_s) · h
    normalizer += w_c · w_s
  end
  return Vertex v̂ = v + n · (sum/normalizer)
```

Fig. 1. Pseudo-code of bilateral filtering

point and each neighbor. Ideally the distance used here should be geodesic, the paper states that this distance can be aproximated to a euclidean the distance. H is the distance between the neighbor and the plane formed by the vertex and its normal. Some observations here are that the neighborhood is defined by the points inside a distance $\rho = 2\sigma_c$ and the normal of the vector is the normal of the neighbor faces of the vertex weighted by their areas.

From this pseudo-code we can understand that the bigger these two distances (t and h) for a given neighbor smaller the influence of this neighbor in the displacement of the vertex v. Also, we see two parameters σ_c and σ_s. The first one the author suggests that the user should select a vertex and then σ_c would be the furthest neighbor (the author didn't specify, but I understood that it was in the 1-ring neighborhood) from the selected point. σ_s would be the standard deviation on this neighborhood.

To implement this algorithm I used the code from subdivision TP, so I could initially test with the sphere and after with the monkey mesh. Also, I could control the number of points in the mesh before

applying the filter, which was important since with a huge number of vertices I wasn't able to run it in my computer. The noise to the mesh was added artificially whenever the user presses the 'N' tile.

```
void addNoise(){
  if (_noNoiseVertexPositions.empty()){
    for(unsigned int i = 0 ; i < _vertexPositions.size() ; ++i) {
      _noNoiseVertexPositions.push_back(_vertexPositions[i]);
    }
  }
  for(unsigned int i = 0 ; i < _vertexPositions.size() ; ++i) {
    _vertexPositions[i].x += ((rand() % 100) - 50) * 0.0001;
    _vertexPositions[i].y += ((rand() % 100) - 50) * 0.0001;
    _vertexPositions[i].z += ((rand() % 100) - 50) * 0.0001;
  }
  recomputePerVertexNormals( );
  recomputePerVertexTextureCoordinates( );
}
```

Fig. 2. Function created to add noise to the mesh

The workflow in Fig. 3 is applied when the user presses the 'R' tile. The details of each function can be found in Mesh.h file.

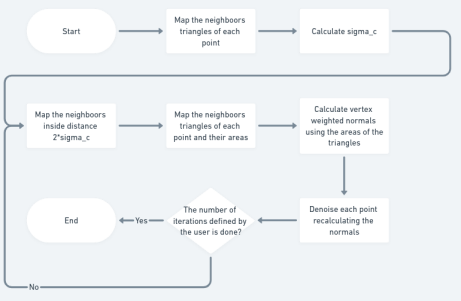

Fig. 3. Workflow of my implementation

4 MODIFICATIONS & IMPROVEMENTS

The paper proposes that the user should select a point that is supposed to be smooth and from this point the parameters σ_s and $sigma_c$ would be selected. In my opinion, choosing these parameters indirectly from the choice of a point wasn't great (mainly for $sigma_s$) since these two have direct influence in the denoising process. In my implementaation I changed the choice of $sigma_c$ was yet influenced by the neighborhoods of the mesh but there was no need of the user choose the point, since in a subdivided and well distributed mesh we would have the furthest point of the neighborhood with quite similar values, so, instead of the user choosing a point I just took a random point in the mesh after applying a subdivision loop. For the $sigma_s$ I made it an arbitrary value that the user can input in the command line when executing the program. The w_s value is a weight in function of h to the own values h, as so, this weight modifies how the function behaves. As we can see in the plot of Fig. 4 the $sigma_s$ influences in the calculations on the displacement of the plot, and the value must be selected directly by the user.

Other modification that I made in the program is about the division by 0 in the operation $sum/normalizer$. If the normalizer is 0 we would have a NaN value. This happens when for a given neighborhood all the $(w_c \cdot w_s)$ are really close to 0. My solution to this problem is approximate all the products $(w_c \cdot w_s)$ to a number x that we know that is really close to 0. As so in the weighted_sum/normalizer equation we would have:

$$\Sigma \frac{h_n \cdot x}{n \cdot x} \rightarrow \frac{1}{n} \Sigma h_n \tag{1}$$

In code, the program basically checks whether $sum/normalizer$ is a NaN value, if yes, the operation described in eq. 1

I also implemented a method called computeError in order to try to estimate the error (through the sum of the differences of each vertices between meshes) between the noisy mesh and the original one and the error between the filtered mesh and the original one.

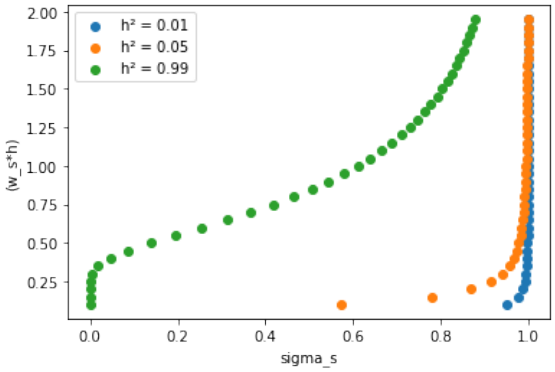

Fig. 4. w_s in function of $sigma_s$

5 OBSERVATIONS & FUTURE WORKS

The error that I calculated sometimes gave a bigger value to the denoised mesh, which is explainable by the shrink of the mesh. The anisotropic denoising methods, such as this one cause this mesh shrinkage, which is a problem. While isotropic such as Taubin's method [2] methods may not cause mesh shrinkage, they hardly preserve geometric features of the mesh which is a huge disadvantage on denoising. I also implemented a method to calculate the variance in the mesh, but for small noises it was aways really close to 0;

In order to contour this problem I suggest to perform the bilateral filtering, and after that find an scaling factor to rescale it isotropically to its original size. The scaling factor could be find by a mesh registration technique, but also we could try to measure the mean in 4 extreme regions (top, bottom, left, right) of the noisy mesh and in the denoised mesh. With these we can estimate extreme points and find an estimation for the scaling factor to rescale the denoised mesh.

6 CONCLUSION AND RESULTS

The method presents a simple denoising procedure, although the authors in the area of mesh smoothing generally do not present any numerical metric to compare with other methods, visually we can see that it worked well. My implementation was quite computationally heavy, but I used a lot of loops that I believe sometimes could

have been substituted for matricial calculations in order to optimize the software.

In the Fig. 5, 6, 7, 8 and 10 we can see the mesh of the sphere without noise, the noisy mesh and the denoised for different values of sigma and 5 iterations.

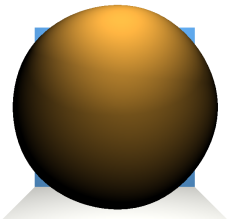

Fig. 5. Initial sphere mesh with 2562 vertices

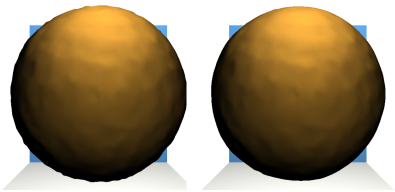

Fig. 6. Denoising for $sigma_s = 0.001$

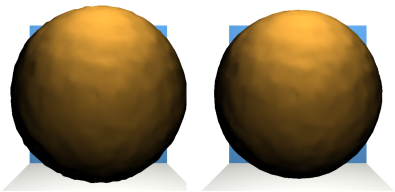

Fig. 7. Denoising for $sigma_s = 0.01$

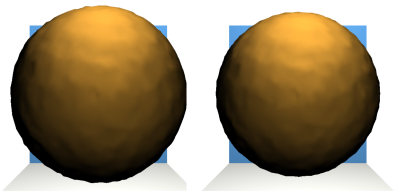

Fig. 8. Denoising for $sigma_s = 0.1$

As we can see different choices of $sigma_s$ have impact on the mesh fairing result. The increase in $sigma_s$ caused more mesh shrinkage.

The implementation was also tested with a simple mesh of a bunny with 10000 vertices. The best $sigma_s$ was 0.0001 and we can observe also some denoising in the bunny, although it's not as clear as in the sphere, but also the bunny wasn't as noisy as the sphere.

$sigma_s = 0.01$s1

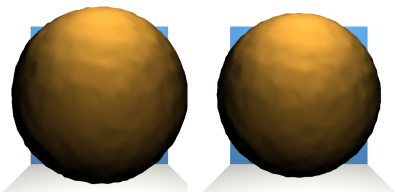

Fig. 9. Denoising for $sigma_s = 1$

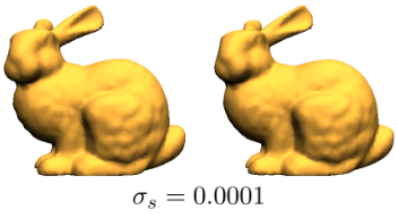

Fig. 10. Bunny denoising

REFERENCES

[1] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. 2003. Bilateral mesh denoising. In ACM SIGGRAPH 2003 Papers (SIGGRAPH '03). Association for Computing Machinery, New York, NY, USA, 950–953. https://doi.org/10.1145/1201775.882368

[2] Gabriel Taubin. 1995. A signal processing approach to fair surface design. In Proceedings of the 22nd annual conference on Computer graphics and interactive techniques (SIGGRAPH '95). Association for Computing Machinery, New York, NY, USA, 351–358. https://doi.org/10.1145/218380.218473

[3] Jianbo Peng, V. Strela and D. Zorin, "A simple algorithm for surface denoising," Proceedings Visualization, 2001. VIS '01., San Diego, CA, USA, 2001, pp. 107-548, doi: 10.1109/VISUAL.2001.964500.

[4] Desbrun, Mathieu and Meyer, Mark and Schröder, Peter and Barr, Alan. (2001). Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow. SIGGRAPH. 99.

[5] X. Cheng, M. Zeng, and X. Liu, "Feature-preserving filtering with L0 gradient minimization," Computers and Graphics, vol. 38, pp. 150–157, 2014.

[6] R. Wang, Z. Yang, L. Liu, J. Deng, and F. Chen, "Decoupling noise and features via weighted l1-analysis compressed sensing," ACM Trans. Graph., vol. 33, no. 2, pp. 18:1–18:12, 2014.

[7] Z. Li, Y. Zhang, Y. Feng, X. Xie, Q. Wang, M. Wei, and P.-A. Heng, "Normalfnet: Normal filtering neural network for feature-preserving mesh denoising," Computer-Aided Design, p. 102861, 2020.