Published in Image Processing On Line on 2017–10–29.
Submitted on 2016–05–30, accepted on 2017–06–09.
ISSN 2105–1232 © 2017 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2017.179

The Bilateral Filter for Point Clouds

Julie Digne[1], Carlo de Franchis[2]

[1] LIRIS, CNRS UMR 5205, Université Lyon 1, France (julie.digne@liris.cnrs.fr)
[2] CMLA, ENS Cachan, France (carlo.de-franchis@cmla.ens-cachan.fr)

Abstract

Point sets obtained by 3D scanners are often corrupted with noise, that can have several causes, such as a tangential acquisition direction, changing environmental lights or a reflective object material. It is thus crucial to design efficient tools to remove noise from the acquired data without removing important information such as sharp edges or shape details. To do so, Fleishman et al. introduced a bilateral filter for meshes adapted from the bilateral filter for gray level images. This anisotropic filter denoises a point with respect to its neighbors by considering not only the distance from the neighbors to the point but also the distance along a normal direction. This simple fact allows for a much better preservation of sharp edges. In this paper, we analyze a parallel implementation of the bilateral filter adapted for point clouds.

Source Code

The ANSI C++ source code permitting to reproduce results from the on-line demo is available on the web page of the article[1].

Keywords: point set surface denoising; bilateral filtering

1 Introduction

Denoising 3D point clouds acquired by some acquisition device is one of the most important tasks of today's geometry processing research. Indeed, the technologies for surface acquisition have diversified: camera-based techniques use structured light, such as a laser ray, or passive light (e.g. stereoscopy), while some other techniques are purely laser-based (time-of-flight). In the meanwhile 3D acquisition devices have become more popular and lower quality devices have appeared and can be obtained at a lower cost. All these evolutions call for efficient tools to recover denoised point sets out of noisy raw data.

This topic has been widely studied since the beginning of geometry processing research. Several methods have been designed for meshes yet they seldom work for point clouds. One of the most

[1] https://doi.org/10.5201/ipol.2017.179

widely used approaches, surface fairing, moves the vertices of the mesh in a motion governed by the heat equation

$$\frac{\partial x}{\partial t} = \lambda \Delta x. \tag{1}$$

Where x is the coordinate function. However estimating the Laplacian of a surface is a difficult problem. In 1995, Taubin proposed a signal-based approach to surface fairing by introducing the umbrella operator [11]: a noisy vertex is replaced by a linear combination of the positions of its one-ring neighbors (vertices linked by an edge to the center vertex). This approach is widely used and several variations have been proposed such as a scale dependent umbrella operator [2]. In the meanwhile, the results on Laplacian discretization for surface meshes gave rise to a whole thread of work dealing with Laplacian eigenfunctions and how they can be used to smooth the surface, in the spirit of the Fourier Transform ([7], [13]).

Yet these methods remain to a certain extent isotropic: points lying on an edge are filtered no differently than the other parts of the shape, which can over-smooth features that one might want to preserve. To overcome this limitation, several methods have been proposed for an *anisotropic fairing* of surfaces ([8], [10]). Among those methods the bilateral filter for meshes proves very efficient.

The concept of bilateral filtering was introduced for images by Tomasi et al. [12], although it was already well defined under a different name in 1985 in a book by Yaroslavsky [14]. Instead of denoising a pixed by considering the color values of its euclidean neighbors, it considers the color values of the neighbors that are close both in position and in color value. This concept was adapted to the denoising of meshes by Fleishman et al. [6], and can be extended to point clouds easily, as will be explained in this paper.

The remainder of this paper is divided as follows: the rest of this introduction presents the input data. Then the bilateral filter is described in Section 2, its implementation is detailed (Section 3) as well as the proposed parallelization (Section 4). In Section 5, we comment upon the choice of parameters and give some technical details about the code (Section 6). Finally, Section 7 presents experiments that can be easily reproduced using the provided code.

1.1 Input Data

The input data of the algorithm is a set of unorganized points assumed to be sampled on a surface. For visualization reasons, the provided demo[2] requires consistently oriented normals (i.e. all normals should point either inwards or outwards), but the bilateral filter has no such limitation and works for unoriented point clouds as well. The provided code takes as input either oriented or unoriented points, possibly with additional information such as the color, or a reflectance property.

Data Structures. The data structures used in this implementation are taken from the Ball Pivoting Algorithm implementation described in [3] and only slightly modified. We refer the reader to this paper for details on methods for sorting the points in an octree and getting neighbors. The files that are specific to this paper are BilateralFilter.h, Sample.h, Sample.cpp, FileIO.h, FileIO.cpp, and main.cpp.

2 The Bilateral Filter

We start by reviewing the bilateral filter for meshes as it was introduced by Fleishman, Drori and Cohen-Or in [6]. Let us first consider a meshed surface $\mathcal{M}$ with known normals $\mathbf{n}_v$ at each vertex

[2] https://doi.org/10.5201/ipol.2017.179

position v. Let $\mathcal{N}(v)$ be the 1-ring neighborhood of vertex v (i.e. the set of vertices sharing an edge with v). Then, the filtered position of v is

$$v + \delta v \cdot \mathbf{n}_v, \tag{2}$$

where

$$\delta v = \frac{\sum_{p \in \mathcal{N}(v)} w_d(\|p - v\|) w_n(|\langle \mathbf{n}_v, p - v \rangle|) \langle \mathbf{n}_v, p - v \rangle}{\sum_{p \in \mathcal{N}(v)} w_d(\|p - v\|) w_n(|\langle \mathbf{n}_v, p - v \rangle|)}. \tag{3}$$

Where w_d and w_n are two decreasing functions.

In a nutshell, Equation (3) means that vertex v is shifted along its normal toward a weighted average of points that are both close to v in the ambient space and close to the plane passing through v with normal $\mathbf{n}_v$. To better see the link with the bilateral filter in image processing, recall that a pixel is denoised with respect to neighboring pixels that look similar, and this similarity is computed in terms of distance between gray values. For meshes, the similarity is measured in terms of distance to the tangent plane. If v belongs to a sharp edge, then the only points close to the tangent plane at v are the points on the edge: these points will be the principal contributors to the denoising of v, thus preserving the edge.

Interestingly, another version of the bilateral filter for meshes was introduced by Jones et al. [9], but its adaptation to point clouds is less simple since it takes into account local areas of the surface around each point, based, for example, on a local Voronoi diagram computation.

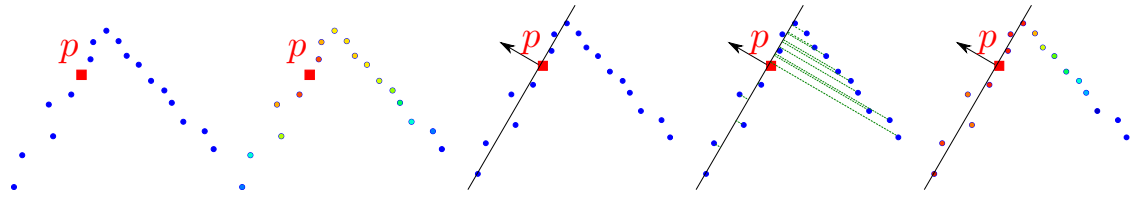
For a non oriented point cloud $\mathcal{P}$, the bilateral filter can be extended as follows:

- First a unit *unoriented* normal vector $\mathbf{n}_p$ is estimated for each point p via its neighbors $\mathcal{N}_r(p) = \{q \in \mathcal{P} | \|q - p\|_2 < r\}$.
- Then the point position is updated via $p' = p + \delta p \cdot \mathbf{n}_p$ and

$$\delta p = \frac{\sum_{q \in \mathcal{N}_r(p)} w_d(\|q - p\|) w_n(|\langle \mathbf{n}_p, q - p \rangle|) \langle \mathbf{n}_p, q - p \rangle}{\sum_{q \in \mathcal{N}_r(p)} w_d(\|q - p\|) w_n(|\langle \mathbf{n}_p, q - p \rangle|)}. \tag{4}$$

  Where w_d and w_n are two decreasing functions.

In our implementation w_d and w_n are two centered Gaussian functions whose variance (σ_d and σ_n resp.) are chosen by the user. As can be seen on Figure 1, the weights on the distances will be balanced by the weights on the normal distances, thus favoring nearby points that are close to the tangent plane: the points that lie on the same side of the edge as p.



(a) Initial point p with its neighborhood (b) Distance weights (c) Regression plane estimation (d) Normal distances $|\langle q - p, \mathbf{n}_p \rangle|$ (e) Normal distance weights

Figure 1: Contributions of neighbors to the denoising of a point p in the bilateral framework. Weights are depicted in a color scale in 1(b) and 1(e) (blue = small weights, red = large weights).

The normal $\mathbf{n}_p$ is estimated by computing the least squares regression plane of the set of neighbors $\mathcal{N}_r(p)$ and setting $\mathbf{n}_p$ to be the normal to this plane. Computing the least squares regression plane is

done trivially by computing the mean and covariance matrix of the neighbors, which yields a point on the plane (the mean) and the normal (the eigenvector corresponding to the least eigenvalues of the covariance matrix). Interestingly enough the orientation of the estimated normal $\mathbf{n}_p$ is irrelevant for the computation: inverting it will yield the same denoised point p'.

3 Implementation

The bilateral filter for a given point $p \in \mathcal{P}$ is summed up in Algorithm 1. Two distances are taken into account: the distance to p and the distance to the regression plane at p. One can see that the denoised point p' is guaranteed to stay within a r-ball centered around p since it is a positively weighted average of the projections on the line $(p, \mathbf{n}_p)$ of points lying in a r-ball centered around p.

Algorithm 1: $bilateral(p, r, \sigma_d, \sigma_n)$

Input: A point $p \in \mathcal{P}$, a radius r, two Gaussian weights σ_d and σ_n
Output: A denoised point p'

$\mathcal{N}_r(p) \leftarrow$ neighbors of p
Compute the unit normal to the regression plane $\mathbf{n}_p$ from $\mathcal{N}_r(p)$
$sum_w = 0$
$\delta_p = 0$
for $q \in \mathcal{N}_r(p)$ **do**
  $d_d \leftarrow \|q - p\|$
  $d_n \leftarrow \langle q - p, \mathbf{n}_p \rangle$
  $w = \exp -\frac{d_d^2}{2\sigma_d^2} \exp -\frac{d_n^2}{2\sigma_n^2}$
  $\delta_p = \delta_p + w d_n$
  $sum_w = sum_w + w$
$p' \leftarrow p + \frac{\delta_p}{sum_w} \mathbf{n}_p$

4 Parallelization

The bilateral filter implementation uses an octree as a space partition structure to accelerate neighborhood queries. An octree is a tree data structure in which each node (or cell) has either no children or 8 children. The root of the octree encodes the shape loose bouding box and the octree recursively subdivides the bounding box into eight octants. We call *processing depth* the depth of the nodes that will be processed in parallel (the root having depth 0). This depth should be small enough to ensure that there is no thread conflict, and large enough to ensure an efficient parallelization. We refer the reader to [3] for more details on this particular octree implementation.

The bilateral filter consisting in very local computations, the process parallelizes nicely provided some precautions are taken. An octree data structure is used to sort cells into sets, each set containing cells that can be processed independently by a different thread (see Algorithm 2), as shown on Figure 2. Thus, each thread applies the bilateral filter to the points included in a different cell. The resulting filtered points might not be located in the same cell and should be stored in the appropriate cell. The only precaution to take is to check that the denoised points obtained in two different threads will not be stored in the same cell, since that would cause conflicts between the threads. Since the filtered point p' of a point p lies inside a ball with radius r centered at p, it is enough to ensure that for two cells processed simultaneously, their dilations of radius r do not contain a common leaf cell.

The processing depth is therefore set as the minimum depth such that the size of the cell is above $d = 2.1r$ and at least 1 (Algorithm 2, line 1-2). This is easily done by computing

$$level = \max(octree.depth - \lfloor log_2 \frac{octree.size}{d} \rfloor, 1).$$

The only remaining possible conflict happens when two threads simultaneously try to add a point in a branch not yet created. Although this case is rare, it is handled by preventing the simultaneous creation of branches (critical section in method *addPoint* of class *Octree*).

The complete parallelized algorithm is summed up in Algorithm 2.

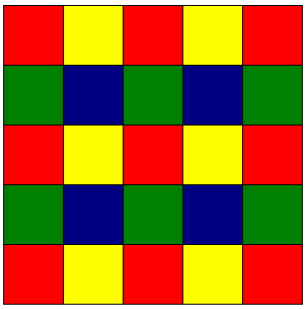


Figure 2: Parallelization principle in 2D. Cells that can be processed simultaneously are depicted in the same color.

Algorithm 2: Parallelization of the bilateral filter

Input: An unoriented input point cloud $\mathcal{P}$ sorted into an octree $\mathcal{O}$ with given depth, a radius r, two Gaussian weights σ_n, σ_d

Output: A denoised point set stored in the octree $\mathcal{O}$.

```
1 d ← 2.1r
2 l ← max(1, smallest level at which cells have size larger than d)
3 for i = 0 ··· 7 do
4     cells ← cells of the octree at level l and with child index i
5     for C ∈ cells do in parallel
6         for p ∈ C do
7             p' ← bilateral(p, r, σ_d, σ_n)
8             C' ← octree cell where p' is located
9             Store p' in the point list of C' corresponding to the next index
```

We review next the parameters of the bilateral filter and how they should be tuned.

5 Parameter Choice

The parameters for the bilateral filter are the following:

- N the number of iterations
- r the radius for the neighborhood search
- σ_d the Gaussian weight for the euclidean distance

- σ_n the Gaussian weight for the distance to the tangent plane

The problem of setting the parameters can be simplified by considering that r and σ_d are related: the contribution of the neighbors which are at distance larger than $3\sigma_d$ barely contribute to the denoising since their weight is almost 0. To avoid unnecessary computations, these points should simply be ignored. Therefore one can set both parameters by choosing a radius r and setting $\sigma_d = \frac{1}{3}r$. σ_n represents the size of the offset around the tangent plane (Figure 3). Hence, it is clear that setting σ_d and σ_n amounts to setting the size of two neighborhoods: a radius r (defining the neighborhood depicted in red on Figure 3) and a normal radius r_n (defining the neighborhood depicted in blue on Figure 3), then we have simply: $\sigma_d = \frac{1}{3}r$ and $\sigma_n = \frac{1}{3}r_n$.

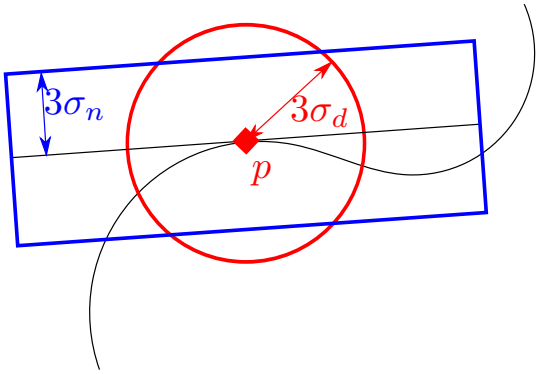


Figure 3: Schematic representation of the parameters of the bilateral filter.

In practice, a coarse heuristic for setting r consists in considering the number of points N_{points}, the size of the bounding box l and deduce the radius: $r = l\sqrt{20/N_{points}}$ (see [4] for a derivation of this formula); σ_n can be set equal to σ_d. In our implementation, this heuristic is used if no parameters are provided.

6 Code

6.1 Dependencies

The code provided is a stand-alone C++ code, available at https://doi.org/10.5201/ipol.2017.179. It uses the C++ standard template library extensively. The user can choose between the single-threaded implementation and its parallel version. The single-threaded version does not rely on any external libraries. The parallelization is done through OpenMP[3], a standard API for shared memory multiprocessing programming. The code was tested successfully on Ubuntu 16.04 and 14.04 with g++ 4.7. The compilation is performed through the CMake build system.

6.2 Integration in a Larger Project

The code is templated and the structures are kept as simple as possible in order for a better integration into different C++ projects. In particular, it should be easy to interface it with the CGAL library [5] and thus benefit from CGAL geometry kernels. Nevertheless, the goal here is to have a stand-alone code, avoiding the need to link against such a heavy library as CGAL. The next section presents the results for this implementation of the bilateral filter.

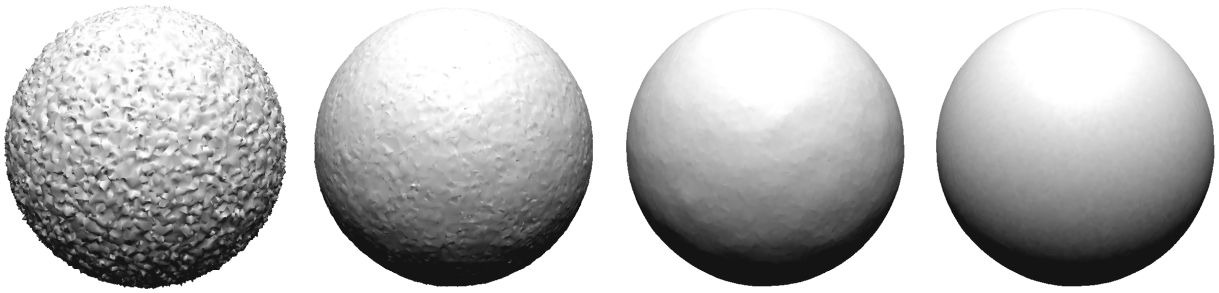
[3]Architecture Review Board, OpenMP Application Program Interface Version 3.0, May 2008, http://www.openmp.org/mp-documents/spec30.pdf

7 Experiments

For a better visualization, all point sets were first meshed using the scale space meshing algorithm with the implementation of [4]. The parameters for this reconstruction step were always: 4 iterations and the radius equal to half of r (the parameter for the bilateral filter).

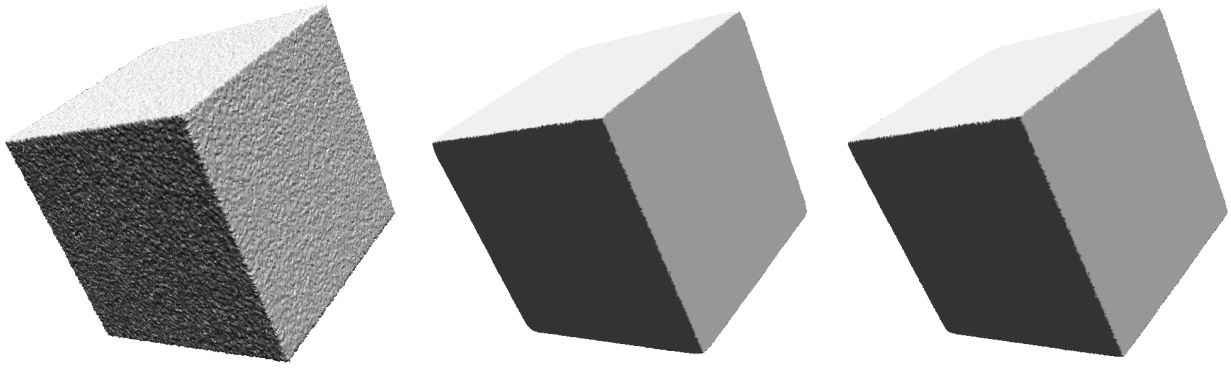
7.1 Synthetic Examples

We display the result of the bilateral filter on some synthetic shapes, a sphere (Figure 4), a cube (Figure 5) and a sharp edge (Figure 6). One can observe, that for the cube and the sharp edge, setting a larger r_n yields rounder edges. For open surfaces (as opposed to closed surfaces), this filter creates artifacts near the surface borders. This can be seen on Figure 6: the surface border near the edge appears to bend toward the concavity. Iterating the filter allows for a better preservation of sharp edges as can be seen on Figure 6.



(a) Noisy sphere (b) $r = 0.05, r_n = 0.01$ (c) $r = 0.05, r_n = 0.04$ (d) $r = 0.05, r_n = 10$

Figure 4: A noisy sphere (added noise with variance equal to 0.5% of the bounding box size) and its denoising with one iteration of the bilateral filter with various parameters

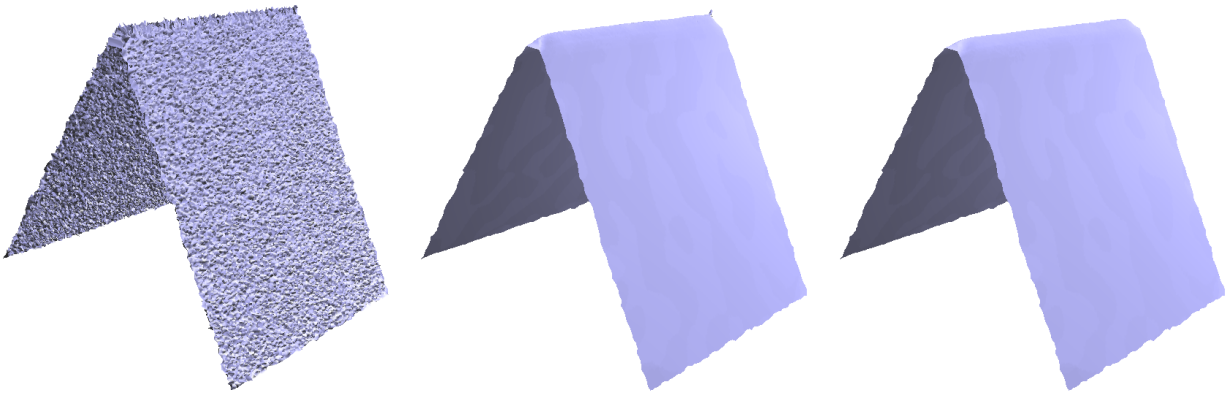


(a) Noisy cube (b) $r = 0.04, r_n = 1$ (c) $r = 0.04, r_n = 0.01$

Figure 5: A noisy cube (added noise with variance equal to 0.1% of the bounding box size) and its denoising with one iteration of the bilateral filter with various parameters

7.2 Real-world Examples

The result of the filter is then demonstrated on some real data starting with the Stanford Bunny (Figure 8) and the Brassempouy dataset (Figure 9). Computation times are given in Table 1 and



(a) Noisy sharp edge (b) $r = 0.04, r_n = 1$ (c) $r = 0.04, r_n = 100$

Figure 6: A noisy sharp edge (added noise with variance equal to 0.5% of the bounding box size) and its denoising with 4 iterations of the bilateral filter with various parameters

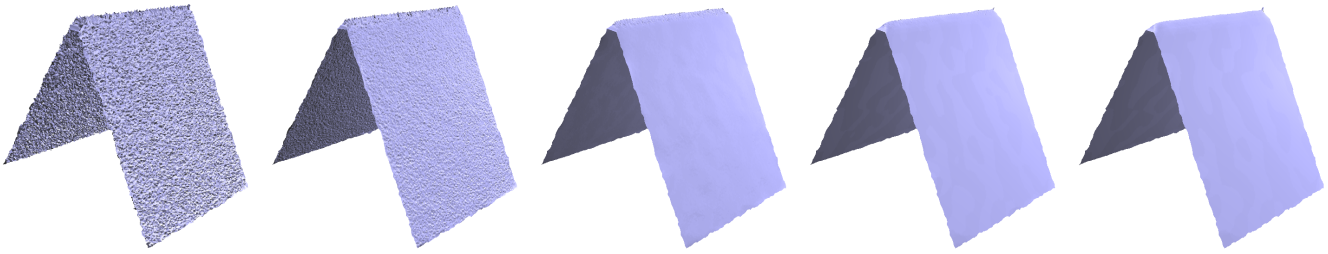


Figure 7: Iterations $1 - 4$ of the bilateral filter with parameters $r = 0.04, r_n = 0.01$. From left to right: noisy shape, iteration 1, 2 ,3 and 4.

show that for typical point sets of 300k points the filtering is fast with a low memory footprint.

Shape	$\#points$	r	r_n	N	Time	Max Memory usage
Brassempouy	312k	0.5	0.5	1	$3s$	12MB
Brassempouy	312k	0.2	0.2	1	$1s$	35MB
Cube	600k	0.03	0.05	2	$13s$	22MB
Fandisk	600k	0.1	0.05	1	$11s$	17MB

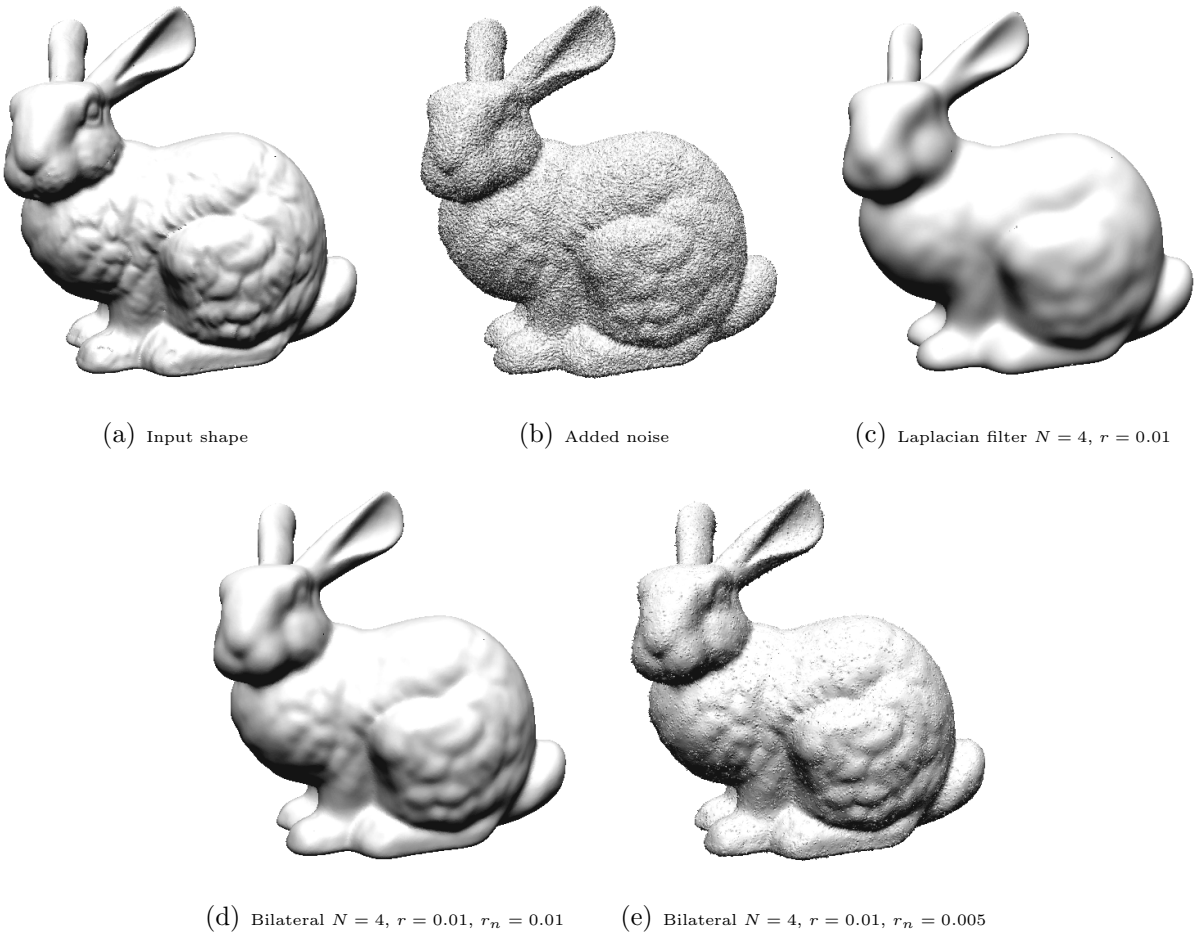
Table 1: Typical filtering times and memory usage (Ubuntu 14.04 - Intel Xeon CPU E5-2630 v3 @2.4GHz x16)

8 Conclusion

We presented a simple implementation of the bilateral filter adapted to point clouds. This filter allows to better preserve sharp edges during the denoising of a point set. It is a good trade-off between processing speed and denoising quality. Indeed, although more recent filters such as the non local means filter [1] produce much better results, they are also much slower.

The code for this implementation is available for download on the web page of the article[4].

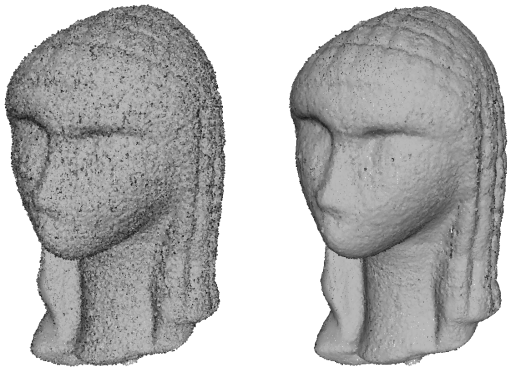
[4] https://doi.org/10.5201/ipol.2017.179



(a) Input shape (b) Added noise (c) Laplacian filter $N = 4$, $r = 0.01$

(d) Bilateral $N = 4$, $r = 0.01$, $r_n = 0.01$ (e) Bilateral $N = 4$, $r = 0.01$, $r_n = 0.005$

Figure 8: Comparison of the denoising of the bunny shape using the Laplacian filter and the bilateral filter with various parameters.



(a) Noisy shape (b) Bilateral filter

Figure 9: Denoising of the Brassempouy dataset. Parameters: $N = 1$, $r = 0.5$, $r_n = 0.5$.

Acknowledgments

This work was partially funded by Agence Nationale de la Recherche, ANR PAPS Project (ANR-14-CE27-0003).

Data Credits

The Stanford Bunny is taken from from the Stanford 3D Scanning Repository[5]. The Brassempouy dataset is taken from the Farman Institute 3D Point Sets[6].

References

[1] A. Buades, B. Coll, and J-M. Morel, *A non-local algorithm for image denoising*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005, pp. 60–65.

[2] M. Desbrun, M. Meyer, P. Schroder, and A.H. Barr, *Implicit fairing of irregular meshes using diffusion and curvature flow*, in SIGGRAPH '99, USA, 1999, pp. 317–324. http://doi.acm.org/10.1145/311535.311576.

[3] J. Digne, *An Analysis and Implementation of a Parallel Ball Pivoting Algorithm*, Image Processing On Line, 4 (2014), pp. 149–168. http://dx.doi.org/10.5201/ipol.2014.81.

[4] ——, *An implementation and parallelization of the scale space meshing algorithm*, Image Processing On Line, 5 (2015), pp. 282–295. http://dx.doi.org/10.5201/ipol.2015.102.

[5] A. Fabri, *CGAL- the computational geometry algorithm library*, in Proceedings of the 10th Annual International Meshing Roundtable, California, U.S.A., October 2001.

[6] S. Fleishman, I. Drori, and D. Cohen-Or, *Bilateral mesh denoising*, ACM Transactions on Graphics, 22 (2003), pp. 950–953. http://doi.acm.org/10.1145/882262.882368.

[7] M. Gross and A. Hubeli, *Eigenmeshes*, tech. report, ETH Zurich, 2000.

[8] K. Hildebrandt and K. Polthier, *Anisotropic filtering of non-linear surface features*, Computer Graphics Forum, 23 (2004), pp. 391–400.

[9] T.R. Jones, F. Durand, and M. Desbrun, *Non-iterative, feature-preserving mesh smoothing*, in SIGGRAPH '03: ACM SIGGRAPH 2003 Papers, USA, 2003, ACM, pp. 943–949. http://doi.acm.org/10.1145/1201775.882367.

[10] C. Lange and K. Polthier, *Anisotropic smoothing of point sets*, Computer Aided Geometric Design, 22 (2005), pp. 680–692. http://dx.doi.org/10.1016/j.cagd.2005.06.010.

[11] G. Taubin, *A signal processing approach to fair surface design*, in SIGGRAPH '95, USA, 1995, ACM Press, pp. 351–358. http://doi.acm.org/10.1145/218380.218473.

[12] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images*, in ICCV '98: Proceedings of the Sixth International Conference on Computer Vision, Washington, DC, USA, 1998, IEEE, p. 839.

[13] B. Vallet and B. Lévy, *Spectral geometry processing with manifold harmonics*, Computer Graphics Forum, 27 (2008), pp. 251–260. http://dx.doi.org/10.1111/j.1467-8659.2008.01122.x.

[14] L. P Yaroslavsky, *Digital picture processing. An introduction.*, vol. 9 of Springer Series in Information Sciences, Springer-Verlag, Berlin - Heidelberg, 1985.

[5] http://graphics.stanford.edu/data/3Dscanrep/

[6] http://www.ipol.im/pub/art/2011/dalmm_ps/