Order-randomized Laplacian mesh smoothing

Ying Yang[1,2], Holly Rushmeier[2], and Ioannis Ivrissimtzis[3]

[1] Fujian Provincial Key Laboratory of Information Processing and Intelligent Control, Minjiang University, Fuzhou, China
[2] Yale University, USA
[3] Durham University, UK
{ying.yang.yy368, holly.rushmeier}@yale.edu
ioannis.ivrissimtzis@durham.ac.uk

Abstract. In this paper we compare three variants of the graph Laplacian smoothing. The first is the standard synchronous implementation, corresponding to multiplication by the graph Laplacian matrix. The second is a voter process inspired asynchronous implementation, assuming that every vertex is equipped with an independent exponential clock. The third is in-between the first two, with the vertices updated according to a random permutation of them. We review some well-known results on spectral graph theory and on voter processes, and we show that while the convergence of the synchronous Laplacian is graph dependent and, generally, does not converge on bipartite graphs, the asynchronous converges with high probability on all graphs. The differences in the properties of these three approaches are illustrated with examples including both regular grids and irregular meshes.

Keywords: Laplacian smoothing; graph Laplacian matrix; voter processes; triangle meshes; regular grids; Taubin smoothing

1 Introduction

Laplacian smoothing is used in a variety of applications as a simple yet effective method for data denoising. Assuming that each data point has a well-defined neighborhood, each iterative Laplacian smoothing step updates the data points by weighted averages of their neighborhoods. Since neighborhood relations are naturally described by graphs with their edges connecting neighboring points, Laplacian smoothing is often described as an operator acting on graphs, smoothing the values of a function defined over the graph's vertices.

In graphics applications, Laplacian smoothing, applied either globally or locally, is often used for smoothing discrete surfaces, especially triangle meshes. In cases of global mesh smoothing in particular, higher quality results are obtained by using more sophisticated variants of the fundamental technique, such as the HC-Laplacian smoothing [18], the curvature flow smoothing [5], or the Taubin smoothing [15]. The latter, which will be used in our experiments, is based on the alteration between a smoothing step with a positive weight w_1 and an anti-smoothing step with a negative weight w_2.

Since Laplacian smoothing is simple, general and well-understood, it is often the technique of choice when a smoothing or noise suppression algorithm needs to be incorporated into a general mesh processing framework [10], or into a more complex algorithm, as for example the machine learning algorithm for surface reconstruction in [1], or the Boundary Element Method based evolutionary structural optimization in [17].

In image processing applications, weighted neighborhood averaging, usually referred as Gaussian smoothing, is a fundamental technique at the heart of classic edge detection algorithms, or more sophisticated variants of them such as the bilateral edge preserving smoothing in [16], a technique which has been extended to triangle meshes [7]. Bilateral filtering has been studied through the spectrum of the graph Laplacian in [8].

Several curve and surface subdivision algorithms can defined as combined operators with at least one Laplacian smoothing component. The classic Lane-Riesenfield algorithms [11] apply Laplacian smoothing on a very simple graph, i.e. a graph with the connectivity of a polygon, while in some recent generalizations the smoothing step corresponds to the Laplacians of more dense graphs [3].

In this paper we compare three implementations of Laplacian smoothing, depending on the order in which vertices are updated. In the first implementation we consider, L_{sync}, each step updates all graph vertices synchronously. This implementation, which is ubiquitous in practical applications, has the major advantage that we do not have to define a specific order in which the vertices are updated, thus do not have to arbitrarily impose such an order.

In the second implementation, L_{exp}, every vertex carries an independent exponential clock and is updated when that clock rings. L_{exp} has several similarities with the discrete operators called *voter processes*, which are a major modeling and simulation tool. They have been traditionally employed on regular grid settings [4] to model physical phenomena, and recently on irregular graph settings to model social interactions, e.g. spread of influences on social media, or consumer behavior [12, 19]. In the latter cases, the basic voter processes are usually augmented with special features that allow them to capture the complexity of social interactions. As an advantage of the L_{exp} operator, we note that it is more natural than L_{sync} in the sense that it does not assume instant interaction between the parts of the system, here the vertices of the graph, and as a result it can avoid the problem of non-convergence which appears when L_{sync} is applied on bipartite graphs. Moreover, it has a trivial memory efficient implementation since, unlike L_{sync}, it does not require to retain the existing values of the function until the end of the current iteration. One potential disadvantage is that the outcome in non-deterministic, something that in several applications can be unacceptable.

In between L_{sync} and L_{exp}, we also consider an operator L_{perm}, which updates the vertices one by one as in L_{exp}, but following a random permutation of the vertex set rather than using independent exponential clocks. In L_{perm}, no vertex will be updated twice before all vertices have been updated once, allow-

ing for a clear distinction between the iterative steps of the smoothing process, similarly to L_{sync}. This is particularly convenient when Taubin smoothing is applied and thus, we alternate between two distinct Laplacian smoothing steps. Nevertheless, L_{perm} seems to be less interesting than L_{sync} and L_{perm} from a theoretical point of view and here will only be studied experimentally.

Contribution and Limitations: The main contribution of the paper is the demonstration of theoretical and practical shortcomings of the synchronous implementation of the Laplacian smoothing, which have been largely overlooked in the relevant literature. As the main limitation of the paper we note that given the theoretical and practical shortcomings of the two randomized implementations we tested, a conclusive case for their use instead of the ubiquitous synchronous implementation can only be made in the context of a specific application scenario, as for example a specific mesh processing pipeline or a specific grid simulation. The latter however is beyond the scope of this paper.

Overview: In Section 2, we describe the three implementations of Laplacian smoothing in detail, and review the theoretical properties of L_{sync} and L_{exp}. In Section 3, we test all three implementations on regular height maps and 3D triangle meshes, using Laplacian smoothing or Taubin smoothing as appropriate. We briefly conclude in Section 4.

2 Theoretical properties of L_{sync} and L_{exp}

First, we review the convergence properties of the usual synchronous Laplacian smoothing. While the convergence or not of L_{sync} is a direct consequence of well-known spectral properties of the Laplacian matrix, to the best of our knowledge, bipartite graphs have not been identified in the relevant literature as the class of graphs where L_{sync}, generally, does not converge.

Let $G = (V, E)$ be a connected graph with N vertices and let f_n be a real function defined over the vertex set V, which can also be written as a vector

$$\mathbf{f}_n = (f_n(v_0), f_n(v_1), \ldots, f_n(v_{N-1}))^T. \tag{1}$$

One iterative step of the synchronous Laplacian smoothing L_{sync} updates $\mathbf{f}_n$ by

$$\mathbf{f}_{n+1} = L \cdot \mathbf{f}_n \tag{2}$$

where the Laplacian matrix L, indexed by the vertices of G, is given by

$$L = L(i,j) = \begin{cases} 1/d_i & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where d_i is the valence of the vertex v_i. Regarding the spectral properties of L, see for example [2], its eigenvalues satisfy

$$1 = \lambda_0 > \lambda_1 \geq \cdots \geq \lambda_{N-1} \geq -1, \tag{4}$$

that is, all the eigenvalues are in the interval [1,-1] and the largest is equal to 1 and has multiplicity one. Moreover, $\lambda_{N-1} = -1$ if and only if G is bipartite,

that is, when its vertices can be split into two subsets, let say the white and the black vertices, such that only vertices in different subsets are connected with an edge. In that case, the spectrum of L is symmetric about the origin and thus λ_{N-1} also has multiplicity one [2].

Let $\mathbf{v}_i$ for $i = 0, \ldots, N-1$ be an orthonormal basis of $\mathbb{R}^N$ consisting of eigenvectors of L. The initial function $\mathbf{f}_0$ of vertex values can be written in that basis as

$$\mathbf{f}_0 = a_0\mathbf{v}_0 + \cdots + a_{N-1}\mathbf{v}_{N-1} \tag{5}$$

giving

$$\mathbf{f}_n = L^n\mathbf{f}_0 = a_0\lambda_0^n\mathbf{v}_0 + \cdots + a_{N-1}\lambda_{N-1}^n\mathbf{v}_{N-1} \tag{6}$$

which, on bipartite graphs, for large number of iterations n converges to

$$a_0\mathbf{v}_0 + a_{N-1}(-1)^n\mathbf{v}_{N-1}. \tag{7}$$

From Eq. 7 we see that L_{sync} does not converge if the graph G is bipartite and $a_{N-1} \neq 0$, i.e., the initial vertex values $\mathbf{f}_0$ contain a non-zero component of $\mathbf{v}_{N-1}$. The eigenvector $\mathbf{v}_{N-1}$ of bipartite graphs has a quite simple form, namely, it has values +1 on white vertices and -1 on black. On the other hand, for any graph G, bipartite or not, we have

$$\mathbf{v}_0 = (1, \ldots, 1)^T. \tag{8}$$

and simple computations show that if $\mathbf{f}_n$ converges to a function $\mathbf{f}$ then

$$\mathbf{f} = a_0\mathbf{v}_0 \tag{9}$$

where a_0 is the coefficient of the eigenvector $\mathbf{v}_0$ when we write $\mathbf{f}_0$ in the basis of the eigenvectors of L.

Relevance: Bipartite graphs appear often in practice. Open polygons are bipartite, as well as regular quadrilateral grids and regular cubic grids. While triangle meshes are not bipartite, regular hexagonal meshes are bipartite. Regarding general polygonal meshes, the graph with vertex set the vertices and the faces of the mesh, and edges from the vertex-vertices to the incident face-vertices and the face-vertices to the incident vertex-faces, is bipartite. Thus, if we smooth attributes defined on both vertices and faces, e.g. normals, colors or material properties, by updating the vertices by the mean of the incident faces and the faces by the mean of the incident vertices, then the process would generally not converge.

2.1 L_{exp} smoothing and voter processes

Since L_{exp} is a probabilistic process we cannot study it through the spectral properties of the Laplacian matrix. Instead, we will study it through its relation to voter processes, which can be seen as the discrete counterparts of L_{exp}. While voter processes are well-understood and have already found numerous

applications, to the best of our knowledge their connection with the L_{exp} implementation of the Laplacian smoothing has not been studied. Next we will make this connection explicit, essentially formalizing the following simple intuitive argument; under L_{exp} smoothing, the values of the function f on the graph vertices correspond to opinion expectations in a voter process.

The L_{exp} smoothing process: Following the standard terminology on voter processes, each vertex of G carries an independent exponential clock of rate 1. Each time a clock rings the value on its vertex is updated, becoming the mean of the values of its 1-ring neighborhood. In matrix notation, this is equivalent to multiplication by the matrix L_k derived from the identity matrix by substituting its k-th row with the k-th row of the Laplacian matrix. That is,

$$\mathbf{f}_{m+1} = L_k \cdot \mathbf{f}_m \tag{10}$$

where

$$L_k = L_k(i,j) = \begin{cases} L(i,j) & \text{if } i = k \\ 1 & \text{if } i = j \neq k \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

Let $t_0, t_1, ..., t_n$ be the sequence of vertex indices in the order they rang during the smoothing process. We have

$$\mathbf{f}_{n+1} = L_{t_n} \cdots L_{t_0} \mathbf{f}_0 \tag{12}$$

which, by writing $\mathbf{f}_0$ in the natural base becomes

$$\mathbf{f}_{n+1} = \sum_{i=0}^{N-1} f_0^i \cdot L_{t_n} \cdot \ \ldots \ \cdot L_{t_0} \cdot \mathbf{e}_i \tag{13}$$

where f_0^i is i-th coordinate of $\mathbf{f}_0$ and $\mathbf{e}_i$ is the vector with 0's everywhere and 1 at the i-th coordinate. From Eq. 13 we see that for large values of n the behavior of L_{exp} depends on the limit of

$$L_{t_n} \cdot \ \ldots \ \cdot L_{t_0} \cdot \mathbf{e}_i. \tag{14}$$

The corresponding voter process: In its standard form, a voter process is defined on a graph whose vertices carry a value from the set $\{0, 1\}$, commonly called the *opinion*. Each vertex carries an independent exponential clock and upon ringing that vertex will choose with uniform random probability one of its neighbors and adopt its opinion. To each vertex i we associate the binary random variable X_i with

$$X_i(0) = 0, \quad X_i(1) = 1 \tag{15}$$

and the set of vertices V is associated to the vector of binary random variables

$$\mathbf{X} = (X_0, \ldots, X_{N-1})^T. \tag{16}$$

Let the vector of the probability distributions of the opinions at step n be

$$\mathbf{Y}^n = (Y_0^n, \ldots, Y_{N-1}^n). \tag{17}$$

From Eq. 13, it suffices to study the behaviour of the process for initial opinion vector $\mathbf{e}_i$, in which case we have

$$\begin{cases} p(Y_i^0 = 0) = 0 \quad p(Y_i^0 = 1) = 1 \\ p(Y_j^0 = 0) = 1 \quad p(Y_i^j = 1) = 0 \text{ for } i \neq j \end{cases} \tag{18}$$

and the initial expectation vector is $E(\mathbf{X}) = \mathbf{e}_i$. Assuming that the clocks at the vertices of the graph ring in the order $t_0, t_1, \ldots, t_n$, we have

$$\mathbf{Y}^n = L_{t_n} \cdot \ \ldots \ \cdot L_{t_0} \cdot \mathbf{Y}^0 \tag{19}$$

and by linearity, the expectation vector at step n is

$$E(\mathbf{X}) = L_{t_n} \cdot \ \ldots \ \cdot L_{t_0} \cdot \mathbf{e}_i. \tag{20}$$

For any graph, including bipartite graphs, the above voter process reaches consensus with high probability [6]. That is, for any $\epsilon > 0$, after a sufficiently large number of steps, all the vertices will have the same opinion with probability at least $1 - \epsilon$. Moreover, the probability for the consensus being at opinion 1 is equal to the sum of the valences of the vertices with initial opinion 1 divided by the sum of all valences, which is twice the number of edges $|E|$. For the initial opinions corresponding to the basis vector $\mathbf{e}_i$, the expectation vector under the condition that consensus has been reached is

$$E_{cns}^i = \frac{d_i}{2|E|}(1, 1, \ldots, 1)^T. \tag{21}$$

Remark: [9] studies a voter process where all vertices update simultaneously, i.e., in the fashion of L_{sync} rather than L_{exp}. The main result is that consensus is reached with high probability for all non-bipartite graphs. In [13], these two voter processes are compared and the main result is that the asynchronous process reaches consensus faster than the synchronous. The latter result is reflected in our experiments regarding the speed of the smoothing processes in Section 3.

The L_{perm} smoothing process: In-between L_{sync} and L_{exp} we define a third process L_{perm} where the graph vertices are updated one by one, according to a random permutation of them. Since there is no much literature on the theoretical properties of L_{perm}, we will only study it experimentally, generally expecting a behavior similar to L_{exp}.

The motivation for including L_{perm} in our investigations is two-fold. Firstly, as L_{exp} is different from L_{exp} in that all vertices are updated the same number of times, a comparison between the two methods can reveal the affect on L_{exp} of the fact that some vertices may be updated significantly fewer times than the average. Secondly, L_{perm} is similar to L_{sync} in that a single iteration of the process is clearly defined, i.e. when all vertices have been updated once. This is particularly convenient when we alternate between different steps of Laplacian smoothing, as for example in the case of Taubin smoothing.

3 Experimental results

In our examples we use two graph types. First, graphs with the connectivity of a regular 2D grid and a real-valued function $f : V \to \mathbb{R}$ defined over its vertices. Secondly, irregular graphs with a vector valued function $\mathbf{f} : V \to \mathbb{R}^3$ defined on the vertex set. In the first case, f would typically be a spatially regular sample of measurements of a physical quantity and will often be visualized as a 2.5D height map. In the second case, $\mathbf{f}$ would typically represent the spatial coordinates of the vertices of a polygonal mesh embedded in 3D and the Laplacian smoothing process will be applied on each coordinate separately.

In Fig. 1 we applied the three Laplacian operators on a 50×50 regular grid with values

$$f(x, y) = \sin(2\pi x) + \sin(2\pi y). \tag{22}$$

Random uniform noise from the interval [-1,1] was added on the interior vertices. In all three cases the boundary of the grid is fixed and we apply the smoothing operators on the interior vertices only. After 50 iterations of the L_{sync}, the artifact related to the non-convergence of the operator on bipartite graphs is clearly visible, even though the fixed boundary means that eventually the result will be smooth as the boundary values slowly propagate towards the interior. Next, we applied the operator L_{exp} until one vertex had been updated for 50 times and as expected, the result was smooth. Notice that applying L_{exp} until one vertex has been updated for 50 times means that all other vertices are updated fewer than 50 times. However, as L_{exp} updates vertices on the fly, it mixes vertex values faster than L_{sync}. Finally, we notice that, as expected, L_{perm} yielded results similar to L_{exp}.

The next figures show results on triangle meshes. The Fandisk model was chosen for its flat areas and sharp edges, while the Eight model was chosen for the absence of sharp edges and its non-trivial topology. The subdivided Dipyramid was chosen for its rotational symmetry and its highly regular connectivity which produce artifacts when Laplacian smoothing is applied.

Fig. 2 shows the results of applying L_{exp} on noisy Fandisk and Eight models. We notice that while most of the noise has been removed from both models, the edges of the Fandisk have not been preserved and there is more residual noise compared to L_{sync} and L_{perm}, see Figs. 3 and 4. The smoothed Eight model exhibits again some residual noise, but avoids the medium frequency artifacts created by L_{sync} and L_{perm} for the given edge preserving choice of parameters w_1 and w_2, see Fig. 3. We note that the larger amount of residual noise in L_{exp} is due to the fact that some mesh vertices are smoothed a few times only and that it can be a serious limitation in certain graphics applications.

Fig. 3 compares L_{perm} and L_{sync} on Taubin smoothing. We notice that while L_{sync} and L_{perm} can produce results that visually are very similar to each other, the values of certain algorithmic variables such as the number of iterations, or the weights of the Taubin smoothing, are not directly comparable. In particular, for given Taubin smoothing weights, L_{perm} requires fewer iterations than L_{sync}. That was expected since L_{perm} updates the mesh vertices on the fly and thus,

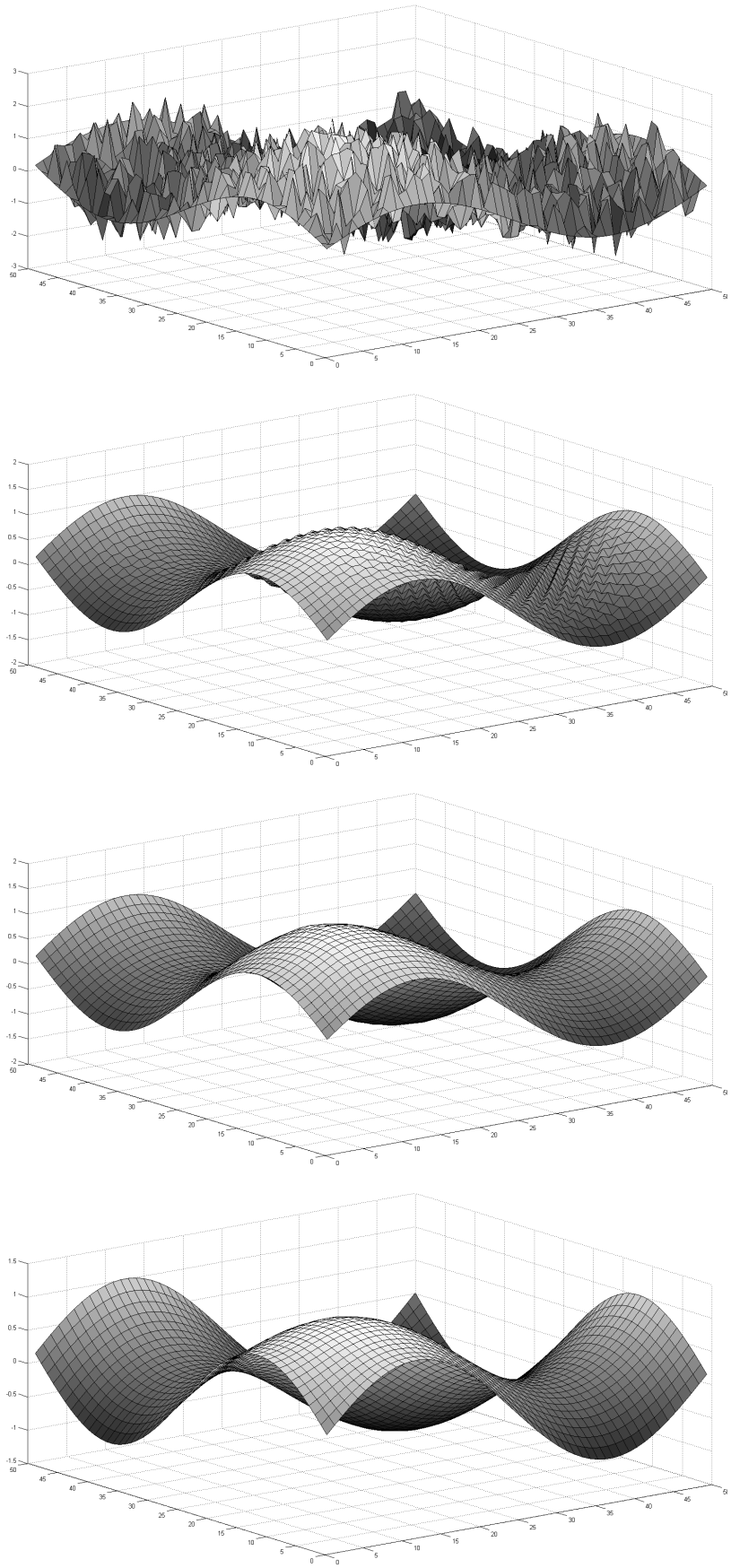

Fig. 1. Top to bottom: noisy grid data, 50 iterations of L_{sync}, L_{exp} and L_{perm} smoothing.

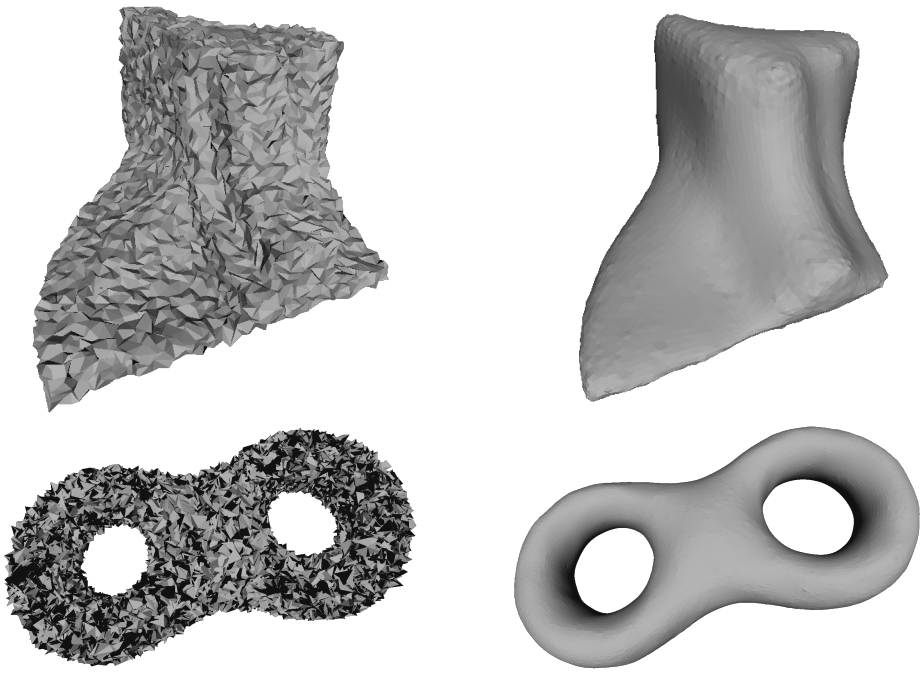

Fig. 2. The noisy Fandisk and Eight model smoothed with L_{exp}. The process terminates when one of the mesh vertices reaches 50 iterations.

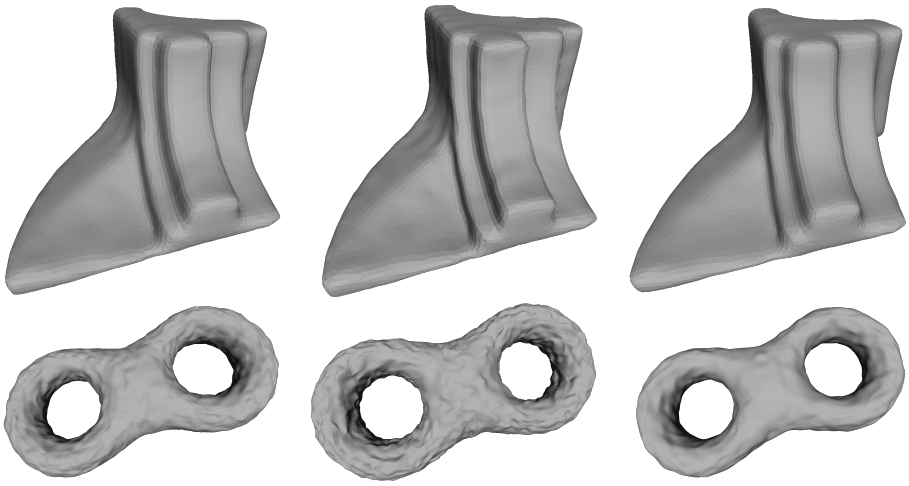

Fig. 3. Smoothing of the noisy meshes in Fig. 2 with L_{perm} and L_{sync}. **Left to right:** 50 iterations of L_{perm}, 50 iterations of L_{sync} and 100 iterations of L_{sync}. In all cases the Taubin smoothing weights are $w_1 = 0.25$ and $w_2 = -0.20$.

the new value of a vertex starts propagating immediately, rather than after the iteration is complete. We also note that in our basic Matlab implementation, a single iteration of L_{perm} runs faster than a single iteration of L_{sync}. This is because L_{sync} stores the updated vertex values in a temporary array which has to be copied at the end of each iteration.

Fig. 4 compares L_{perm} and L_{sync} on the range of acceptable Taubin smoothing weights. We notice that the L_{perm} accepts a wider range of weights than L_{sync} and in particular, the anti-smoothing weight w_2 can be significantly larger in absolute value than the smoothing weight w_1.

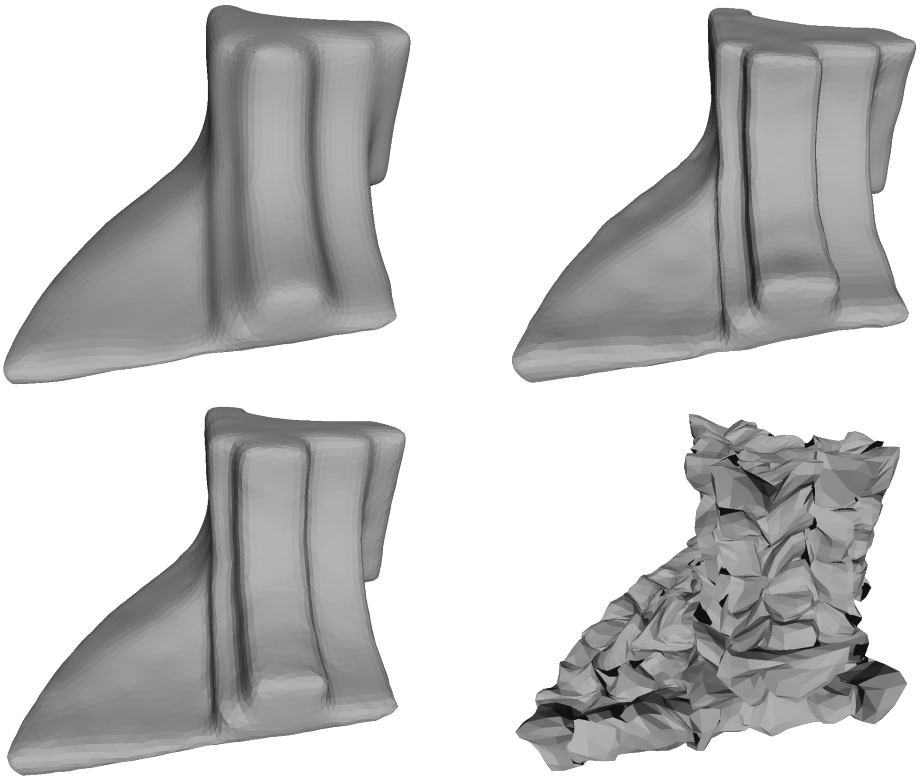

Fig. 4. Smoothing of the noisy meshes in Fig. 2 with 200 iterations of L_{perm} (left) and L_{sync} (right). **Top:** Taubin weights $w_1 = 0.25$ and $w_2 = 0.25$. **Bottom:** Taubin weights $w_1 = 0.25$ and $w_2 = 0.3$.

Finally, in Fig. 5 we apply the three processes on a linearly subdivided dipyramid. We notice that surface artifacts, as have been studied in [14], appear in all three cases and are very similar to subdivision artifacts. A possible explanation for this is that several subdivision schemes can be described as a combination a linear subdivision step followed by a weighted Laplacian smoothing step.

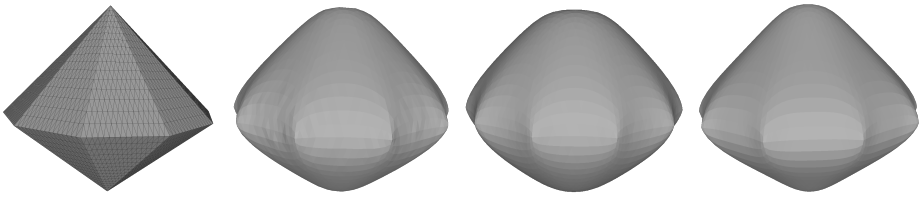

Fig. 5. From left to right: (a) The original mesh. (b) 200 iterations of L_{sync} with weight $w = 0.1$. (c) 200 iterations of L_{perm} Taubin smoothing with $w_1 = 0.25$ and $w_2 = 0.20$. (d) 200 iterations of L_{sync} Taubin smoothing with $w_1 = 0.25$ and $w_2 = 0.20$.

4 Conclusions

We compared three different implementations of Laplacian smoothing, depending on the order in which the graph vertices are updated. We reviewed their theoretical properties, focusing on their different behaviors over bipartite graphs. Our tests showed that these theoretical differences are visually significant when Laplacian smoothing is applied on regular grids.

As we noticed in the literature review in Section 1, Laplacian smoothing is most often applied in the form of several consecutive iterations of weighted Laplacian smoothing, rather than as direct application of the Laplacian matrix in Eq. 3. Moreover, it is often applied locally rather than globally and it is often just one step of a more complex mesh processing algorithm. In such settings, the results in Section 3 do not provide any compelling evidence against current practice of using L_{sync} as the default implementation of Laplacian smoothing. In the future, we plan to compare these three different implementations of Laplacian smoothing in the context of a specific 3D modeling problem, in particular, machine learning based surface reconstruction as, for example, in [1].

References

1. Hendrik Annuth and Christian Bohn. Growing surface structures: A topology focused learning scheme. In *Computational Intelligence*, pages 401–417. Springer, 2016.
2. Andries Brouwer and Willem Haemers. *Spectra of graphs*. Springer Science & Business Media, 2011.
3. Thomas J. Cashman, Kai Hormann, and Ulrich Reif. Generalized Lane–Riesenfeld algorithms. *Computer Aided Geometric Design*, 30(4):398–409, 2013.
4. Theodore Cox and David Griffeath. Diffusive clustering in the two dimensional voter model. *The Annals of Probability*, pages 347–370, 1986.
5. Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH*, pages 317–324, 1999.
6. Peter Donnelly and Dominic Welsh. Finite particle systems and infection models. *Mathematical Proceedings of the Cambridge Philosophical Society*, 94(01):167–182, 1983.

7. Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. Bilateral mesh denoising. *ACM Trans. Graph.*, 22(3):950–953, 2003.
8. Akshay Gadde, Sunil K. Narang, and Antonio Ortega. Bilateral filter: Graph spectral interpretation and extensions. In *2013 IEEE International Conference on Image Processing*, pages 1222–1226. IEEE, 2013.
9. Yehuda Hassin and David Peleg. Distributed probabilistic polling and applications to proportionate agreement. *Information and Computation*, 171(2):248–268, 2001.
10. Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH*, pages 105–114. ACM, 1998.
11. Jeffrey M. Lane and Richard F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):35, 1980.
12. Yanhua Li, Wei Chen, Yajun Wang, and Zhi-Li Zhang. Influence diffusion dynamics and influence maximization in social networks with friend and foe relationships. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 657–666. ACM, 2013.
13. Toshio Nakata, Hiroshi Imahayashi, and Masafumi Yamashita. A probabilistic local majority polling game on weighted directed graphs with an application to the distributed agreement problem. *Networks*, 35(4):266–273, 2000.
14. Malcolm A. Sabin and Loïc Barthe. Artifacts in recursive subdivision surfaces. *Curve and Surface Fitting: Saint-Malo*, pages 353–362, 2002.
15. Gabriel Taubin. A signal processing approach to fair surface design. In *SIGGRAPH*, pages 351–358, 1995.
16. Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.
17. Baseer Ullah, Jon Trevelyan, and Ioannis Ivrissimtzis. A three-dimensional implementation of the boundary element and level set based structural optimisation. *Engineering Analysis with Boundary Elements*, 58:176–194, 2015.
18. Jörg Vollmer, Robert Mencl, and Heinrich Mueller. Improved laplacian smoothing of noisy surface meshes. *Computer Graphics Forum*, 18(3):131–138, 1999.
19. Chuan Zhou, Peng Zhang, Wenyu Zang, and Li Guo. On the upper bounds of spread for greedy algorithms in social network influence maximization. *IEEE Transactions on Knowledge and Data Engineering*, 27(10):2770–2783, 2015.