Mesh Smoothing with Manifold Harmonics

M. Cihan Ozer

Université de Montréal, LIGUM

Summary: In this paper, we will talk about a mesh smoothing approach depending on Manifold Harmonics [Vallet et al. 2008]. This approach smooths the mesh by decreasing number of eigenfunctions in use thus, geometric features of the surface are not preserved. This paper focuses on theoretical concept of the approach in high level, and conveys some implementation tips wrapped with theoretical information. For more details about theoretical side, you can refer [Crane 2013], [Lévy et al. 2009] and [Vallet et al. 2008]. Matlab and C++ code examples can be referred on

mcihanozer.com: Mesh Smoothing with Manifold Harmonics

1. Introduction

Fourier Transform is a classical tool for signal processing. It decomposes a function of time (a signal) into a sum of sinusoidal basis (Fourier function basis). Because of that, Fourier Transform is used to implement low-pass or more general convolution filters. The same idea can be extended to arbitrary manifolds by considering Laplace operator. Laplacian provides interesting insights in the structure and morphology of the shape. This idea is applied to geometry processing by [Taubin 1995] for the first time using Tutte Laplacian. In time, different Laplacians are used for geometry processing. For details, you can check [Zhan et al. 2010]'s survey.

For manifolds related spectral analysis, continues setting of the Laplace operator is used. However, since computers use discrete quantities, we should discretize this continues Laplace operator. For discretizing, finite element method (FEM) and discrete exterior calculus (DEC) are used. An important point at this step is keeping Laplace operator symmetric and its eigenvectors orthonormal during the discretization process. If we miss these two properties, the mesh becomes improper for spectral analysis (Figure 2).

In here, we will use DEC approach to get symmetric positive semi-definite Laplace operator based on [Vallet et al 2008]'s approach.

2. General Steps of Mesh Smoothing using MH

Mesh smoothing using manifold harmonics includes 4 main steps.

1. Compute each function basis called manifold harmonic basis (MHB) for a given triangulated mesh. This step includes 3 substeps:

    a. Assemble positive semi-definite discrete Laplacian $\bar{\Delta}$
    b. Compute eigenvectors of $\bar{\Delta}$
    c. Map each eigenvector ($\overline{H^k}$) into the canonical basis to obtain MHB (H^k):

$$H^k = \star_0^{-1/2} \overline{H^k} \quad (1)$$

    Here, $\star_0$ is called Hodge star for 0-forms, which are integration over every vertex of the mesh (dual vertices).

2. After we get MH bases, by applying manifold harmonic transform (MHT), we transform the geometry into frequency space.
3. Once our geometry transformed into frequency space, we can apply a filter like [Vallet et al. 2008] did. However, we will only decrease the number of function basis H^k in use for smoothing.
4. Finally, we apply inverse manifold harmonic transform (MHT^{-1}) to transform the geometry back into the geometric space.

3. Computing MHB

In this section, we will give details about Laplace operator, Hodge star $\star_0$, computing MH bases, and some tips about implementation.

3.1 Setting up the Laplacian

As mentioned earlier, eigenvectors of the Laplacian is used for computing MH bases. Our approach uses cotangent Laplacian (**L**), and makes **L** symmetric using DEC for getting orthonormal eigenvectors.

L is an ***n x n*** matrix where ***n*** is the number of vertices, and it is calculated as below in one-ring neighborhood:

$$L_{ij} = \begin{cases} j \in N(i), & \cot(\alpha_{ij}) + \cot(\beta_{ij}), \\ j \notin N(i), & 0, \\ i = j, & -\sum_{k \neq i} L_{ik} \end{cases}$$

where ***N(i)*** is the set of vertices adjacent (neighboring) to vertex ***i***, α_{ij} and β_{ij} are the angles opposite to edge ***ij*** (Figure 1).

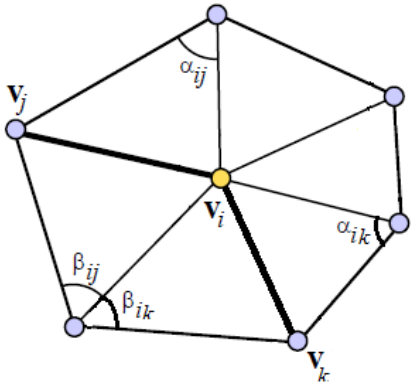

Figure 1. Opposite angles of the edge *ij* and *ik* used for computing cotangent Laplacian (Image is prepared based on [Sorkine 2005]).

Symmetry for Laplacian $\bar{\Delta}$ is quite important. If $\bar{\Delta}$ loses its symmetry ($\overline{\Delta_{ij}} \neq \overline{\Delta_{ji}}$), eigenfunction basis becomes no longer orthonormal, and this does not allow for correct reconstruction. Thus, when we apply MHT^{-1}, we get a deformed or broken mesh (Figure 2).

We use Hodge star $\star_0$ to recover the symmetry. Hodge star $\star_0$ is a diagonal matrix whose diagonal elements corresponds to dual area associated with the vertex. [Crane 2013] defines this dual area as total area of one-ring neighborhood of a vertex divided by 3 in his source code[1].

After we get Hodge star$\star_0$, we can make $\bar{\Delta}$ symmetric, and its eigenvectors orthonormal with a simple matrix multiplication (Equation 2). Alternatively, each coefficients of $\bar{\Delta}$ can be computed as in Equation 3.

[1] See "*DiscreteExteriorCalculus.inl*" file on https://github.com/dgpdec

$$\bar{\Delta} = \star_0^{-1/2} \, \Delta \, \star_0^{-1/2} \tag{2}$$

where, $\star_0$ is Hodge star, Δ is the standard discrete Laplacian except the sign. In our case, it is cotangent Laplacian **L**.

$$\overline{\Delta_{ij}} = -\frac{cot\alpha_{ij} + cot\beta_{ij}}{\sqrt{|v_i^*||v_j^*|}} \tag{3}$$

where α and β are opposite angles of the edge ***ij***, v_i^* is dual area of vertex ***i***.

Implementation starts with computing cotangent Laplacian and Hodge star$\star_0$. In our implementation, we used gptoolbox[2] for Matlab, and ligigl[3] for C++.

One of the important points in here, you should handle numerical precision issue. This issue may make eigenvectors of the Laplacian not orthonormal even if the Laplacian is symmetric. You can solve this problem by $\bar{\Delta} = (\bar{\Delta} + \bar{\Delta}^T)/2$ operation (T is transpose).

3.2 Computing the Bases

After positive semi-definite discrete Laplacian $\bar{\Delta}$ is assembled, its eigenvectors are computed. These eigenvectors are bases for manifold harmonic transformations. All Eigenvector (v^k) and eigenvalue (λ_k) pairs of the Laplacian on a manifold satisfy:

$$-\Delta v^k = \lambda_k v^k \tag{4}$$

This "−" sign is required for the eigenvalues to be positive [Lévy et al. 2009]. Thus, before starting to compute eigenfunctions of $\bar{\Delta}$, we should multiply it by -1.

Another important point is order of eigenvalues and eigenvectors. Eigenvectors are used either sorted by increasing eigenvalues or by decreasing eigenvalues. In this approach, eigenvalues are sorted in ascending order. If we denote normalized eigenvectors as $e_1, e_2, \ldots, e_n$, then corresponding eigenvalues are $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. Thus, eigenvectors of the Laplacian should be sorted by increasing eigenvalues before they map into canonical basis (Equation 1).

[2] https://github.com/alecjacobson/gptoolbox
[3] https://github.com/libigl/libigl

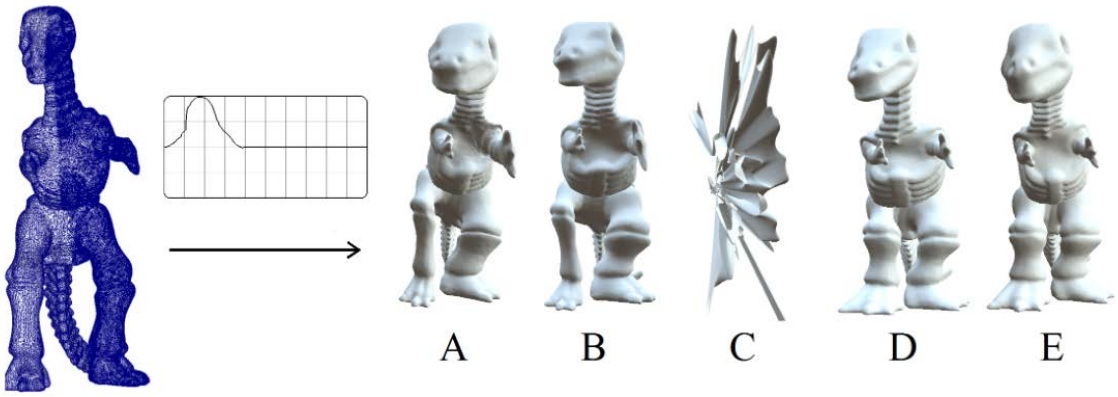

Figure 2. Filtering an irregularly sampled surface (twice denser on the right half) with different discrete Laplacians. Combinatorial Laplacian (A), unweighted cotan $\cot(\alpha_{ij}) + \cot(\beta_{ij})$ (B) and symmetrized weighted cotan $(\cot(\alpha_{ij}) + \cot(\beta_{ij})) / (A_i + A_j)$ (D) produce deformed results (Right leg is bigger). Weighted cotan $(\cot(\alpha_{ij}) + \cot(\beta_{ij})) / (A_i)$ are not symmetric which does not allow for correct reconstruction (C). Only symmetric weights $(\cot(\alpha_{ij}) + \cot(\beta_{ij})) / \sqrt{A_i + A_j}$ are fully mesh-independent (E) [Vallet et al. 2008].

[Vallet et al. 2008] proposes an efficient mechanism to compute eigenvalues for meshes with a large number of vertices. Since, we currently use meshes with relatively small vertices, this part is not included to our implementation. Instead, functions of Matlab and Eigen library are used for computation of eigenfunctions.

During our investigation, we see that neither Eigen's *EigenSolver* nor Matlab's *eig* function does not return orthogonal eigenvectors even if the Laplacian is positive semi-definite. Thus, we used singular value decomposition methods of these tools to get orthonormal eigenvectors.

4. Manifold Harmonic Transforms and Mesh Smoothing

Manifold harmonic transforms are generalization of Fourier transforms, and they are used for transforming geometries between frequency and geometry space. After we transform the mesh into frequency space, we can smooth it out and transform back to geometry space. As we mentioned earlier, we keep eigenvectors are sorted by increasing eigenvalues. Thus, lower eigenvectors contain general idea of the shape whereas higher eigenvectors contain details of the shape. By excluding these higher bases, we can smooth out the mesh.

Let us assume X is a vector containing x positions of vertices of the mesh. For transforming this vector into frequency space, we apply MHT (Equation 5). Same equation is also used for the vectors containing y and z positions of the vertices with the same MH bases.

$$\widetilde{x_k} = \sum_{i=1}^{n} X_i (\star_0)_{ii} H_i^k \quad (5)$$

where, $\widetilde{x_k}$ is the corresponding coefficient of the k^{th} basis in frequency space, *n* is number of vertices, X_i is x position of i^{th} vertex, $(\star_0)_{ii}$ is dual area of i^{th} vertex, and H_i^k eigenvector value of i^{th} vertex. Same equation can be written as a single vector-matrix multiplication as in Equation 6.

$$\widetilde{x_k} = X^T \star_0 H^k \quad (6)$$

where, $\widetilde{x_k}$ is the corresponding coefficient of the basis in frequency space, X^T transposed x position vector, $\star_0$ is Hodge star, and H^k is basis vector of k^{th} manifold harmonic basis.

Equations 5 and 6 calculate only a coefficient of a basis. For, getting all coefficients, Equation 5 (or Equation 6) is performed for all the bases.

MHT^{-1} is similar to MHT. This time, coefficients in frequency space are multiplied with corresponding MHB, and results are sum to get vertex positions in geometry space (Equation 7). Same operation is also performed for the coefficients of y and z values with the same MH bases.

$$X = \sum_{k=1}^{m} \widetilde{x_k} H^k \quad (7)$$

where, *X* is the vector of x position of the vertices in geometry space, *m* is the number of MH bases in use, $\widetilde{x_k}$ is the corresponding coefficient of the basis in frequency space, and H^k is basis vector of k^{th} manifold harmonic basis.

Smoothing is controlled by *m*. If we use a small value for *m*, only eigenvectors of smaller eigenvalues are considered, and details of the mesh are smoothed out (Figure 3). When all MH bases are used, exactly the same mesh should be obtained.

During the implementation, instead of calculating and using specific number of eigenvectors, you should use all the eigenvectors to generate the bases. Then, during the smoothing process, some of the bases should be excluded. Otherwise, you may not get correct results.

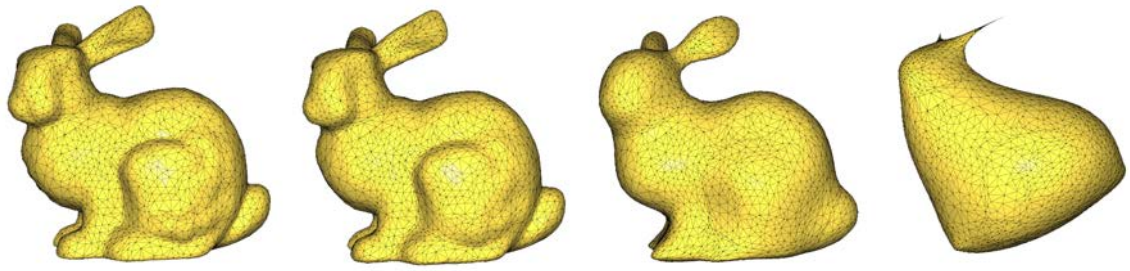

Figure 3. Smoothing process of Stanford bunny with 3485 (original), 1024, 97 and 6 MH bases respectively.

5. Conclusion

Here, we talked about mesh smoothing based on the Laplacian defined in [Vallet et al 2008]. Advantage of this method is that because of the symmetric weights, which are fully mesh-independent, you can transform the mesh back into geometry space without any deformations.

Instead of focusing only theoretical side, we gave crucial information about the approach and implementation tips. You can check the references for the details, and see some code examples at

mcihanozer.com: Mesh Smoothing with Manifold Harmonics

6. References

[Crane 2013]	Keenan Crane. Digital Geometry Processing with Discrete Exterior Calculus. In SIGGRAPH 2013 Course.
[Lévy et al. 2009]	Bruno Lévy and Hao Zhang. Spectral Mesh Processing. In SIGGRAPH Asia 2009 Course 32.
[Sorkine 2005]	Olga Sorkine. Laplacian Mesh Processing. In Eurographics 2005.
[Taubin 1995]	Gabriel Taubin. A Signal Processing Approach to Fair Surface Design. In SIGGRAPH 1995.
[Vallet et al. 2008]	Bruno Vallet and Bruno Lévy. Spectral Geometry Processing with Manifold Harmonics. In Eurographics 2008.
[Zhan et al. 2010]	H. Zang, O. van Kaick and R. Dyer. Spectral Mesh Processing. In Computer Graphics Forum Volume 29 (2010), number 6 pp. 1865-1894.