Non Superfluous Time Solutions of the Laplacian Framework

M. E.Wahed

Computer Science department, Suez Canal University, Egypt
M.Abdallah

Computer Science department, Suez Canal University, Egypt
Mohamed Soliman Elkomy

Computer Science department, Suez Canal University, Egypt

Summary
We present fast solutions to geometric mesh processing related to the Laplacian processing framework and differential representation. We first show how to divide the problem in to miniature problems to save computational time. Then we combine the Laplacian Framework with sensitivity analysis to answer the question of what if changes occurring on the model subject to solution.

1. Introduction:

We begin by some definitions of what is Geometric Processing is: Geometric processing can be defined as the field which is concerned with how geometric objects are worked upon with a computer [JJFH12].The word processing indicates that there is an algorithm involved in the processing action. The data part that the algorithm works on is the Geometry. On the other hand Geometry processing is mostly about applying algorithms to geometric models as stated by [MLMPB10].
Surface representation and processing is one key topic in computer aided design. The surface representation of a 3D object may affect the information that we can perceive about that object. For example the triangular mesh representation can be used to: display the surface, deduce some topological information about the object, in addition to knowing the differential properties of the object that the model represents.
In this paper we build on work done on mesh processing and modeling that is based on the Laplacian framework and differential representations pointed out by [Olg05] and [ATOM06]. As opposing to dealing with Cartesian coordinates, the differential representation utilized in the Laplacian framework results in detail-preserving operations.
Laplacian operator and differential surface representation and surface reconstruction:

To understand the work that this paper presents we will first talk about the Laplacian differential surface representation.
If " $\mathbb{M}$ " is a mesh representing an object with " $\mathbb{V}$ " vertices and " $\mathbb{E}$ " edges and " $\mathbb{F}$ " faces. For each vertex " v_i" we have three Cartesian coordinated associated with each vertex: x_i, y_i , and z_i.
The differential or δ-coordinates of v_i is defined to be the difference between the absolute coordinates of v_i and the center of mass of its immediate neighbors in the mesh

$$\delta_i = \left(\delta_i^{(x)} + \delta_i^{(y)} + \delta_i^{(z)}\right) = v_i - \frac{1}{d_i}\sum_{j\in N(i)} v_i$$

Where N(i) = { j |(i, j) ∈ E} and di = |N(i)| is the number of immediate neighbors of i (1-ring of the vertex). The transformation of the vector of absolute Cartesian coordinates to the vector of δ -coordinates can be represented in matrix form. Starting from the adjacency matrix A which is a square matrix that has 1 in the cell if both the row "i" and column "j" of the cell in the matrix represent an edge between the two vertices i, and j.
Also we have the diagonal matrix D that has Dii =di , hence we can write the L matrix
L= I – D-1A
And the symmetric version Ls
Ls=DL=D-A
Then we can write:

$$Lx = \delta^x$$
$$Ly = \delta^y$$
$$Lz = \delta^z$$

We cannot restore the Cartesian coordinates starting from δ –coordinates; because L is singular. In order to restore the Cartesian coordinates we need to specify the Cartesian coordinates of one vertex to resolve the translational degree of freedom. Substituting the coordinates of vertex i is equivalent to dropping both the ith row and column from L, which makes the matrix invertible [Olg05]. Usually how this is done is by placing more than one special constraint of the mesh vertices. We have therefore

Manuscript received May 5, 2014
Manuscript revised May 20, 2014

|C| additional constraints (called the positional constraints) of the form:

$$v_j = c_j \ , j \in C$$

If the vertices are ordered from 1 to m, then the linear system looks like:

$$\begin{pmatrix} L \\ \omega I_{m*m}|0 \end{pmatrix} x = \begin{pmatrix} \delta^x \\ \omega c_{1:m} \end{pmatrix} \quad (1)$$

The additional constraints make the linear system over-determined (more equations than unknowns) and in general no exact solution may exist. However, the system is full-rank and thus has a unique solution in the least-squares sense:

$$\tilde{x} = \operatorname{argmin}_x \left(\|Lx - \delta^x\|^2 + \sum_{j \in C} \omega^2 |x_j - c_j| \right)$$

Can be written on the following form

$$Ax = b \quad (2)$$

1. Laplacain Mesh Processing Division:

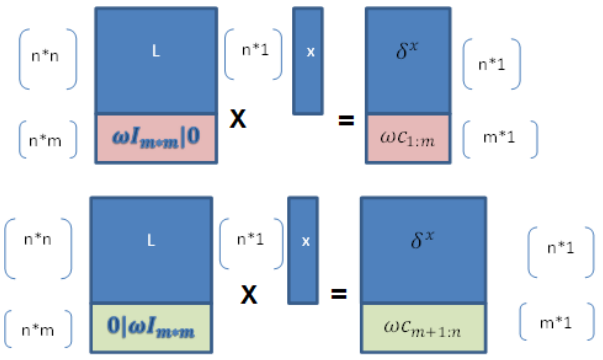


Figure [1] shows in the upper half solving the Laplacian framework with the first section of the positional constraints while the lower half is solving the Laplacian framework with the last section of positional constraints

i. Dividing the Linear problem $Ax = b$

ii. Instead of solving the least squares problem $Ax = b$ we suggest to divide it into two steps as follows (we refer to figure [1] with m=n/2):

iii. We will first solve the upper linear system with, this will give us a solution x, which is composed of two parts, the upper half is a direct result from solving both the non-positional constraints and the positional constraints, the lower half comes only from non-positional constraints.

iv. Next we solve the lower linear system, again the solution x is composed of two parts, the lower is a direct result from solving both the non-positional constraints and the positional constraints, the upper half comes only from non-positional constraints.

v. Then, the final solution will be composed of the upper half from step one combined with the lower half from step two.

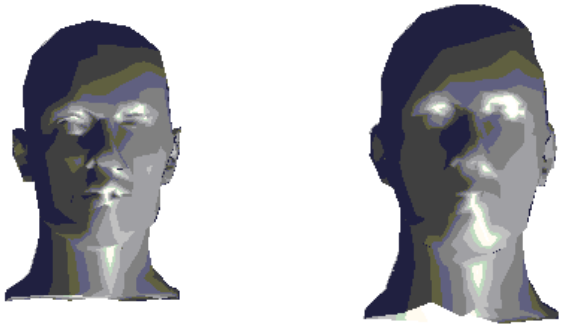


Figure [2] to the left the original object, to the right the resulting object

Figure [2] shows the combined solution object. Of course each divided half takes less time to calculate than the complete problem in which we solve all the positional and non-positional constraints as once. In deed this is evident from [Mic97] who shows that the solution is proportional to size of the problem.

Parallel processing:

In the previous section we divided the problem into two halves; we could have assigned each half to a different processor and solved them in parallel.

We could extend this idea to N processors, were in each processor we solve a linear system composed of positional constraints in addition to non-positional constraints.

In this case we will have N-2 linear systems like figure [3], in addition to the two linear systems in figure [1]. We will take from each X solution of these systems the part that corresponds to the "m" positional constraints used in it.

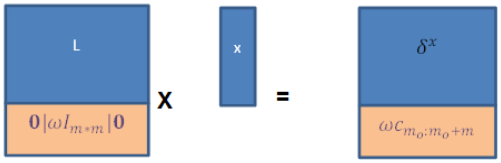


Figure [3] solving the Laplacian framework with a middle section of the positional constraints

Then we will construct the final solution as the upper part from the upper linear system in figure [1], followed by N-2 parts from the linear systems mentioned in figure [3], and finally the lower part from the lower linear system from figure [1].

In case of using heterogeneous processors with different powers the divisions that we make need not be equal, and

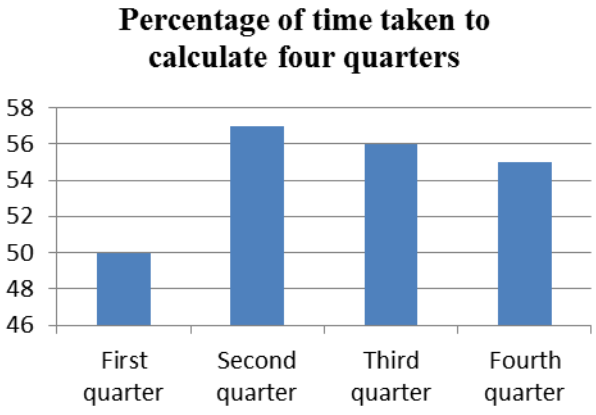
we can assign the more powerful processor a bigger size problem.

In table [1] we can see some statistical data

Metric	Dividing problem into two halves
Mean Normalized Cross-Correlation $NK = \sum_{j=1}^{M}\sum_{k=1}^{N} x_{j,k}\cdot x'_{j,k} \Big/ \sum_{j=1}^{M}\sum_{k=1}^{N} x_{j,k}^2$	0.9635
Average Difference $AD = \sum_{j=1}^{M}\sum_{k=1}^{N}\left(x_{j,k} - x'_{j,k}\right)/MN$	2.7472
Normalized Absolute Error $NAE = \sum_{j=1}^{M}\sum_{k=1}^{N}\left\|x_{j,k} - x'_{j,k}\right\| \Big/ \sum_{j=1}^{M}\sum_{k=1}^{N}$	0.0701

Table [1] statistical data

In the graph [1], we can see that to solve the problem into four parts, each time we solve the Laplacian framework with the non-positional constraints and one quarter of the positional constraints. As can be seen when we divide the problem on four processors the time is almost halved (max time=57% of complete solution).



Graph [1] time taken after dividing the positional constraints into four parts

2. Least squares Perturbation and Laplacain Mesh Processing:

vi. The Effect of delta Changes on Relative Error

vii. The Effect of Changes in b on Relative Error

In this section we consider what if questions applied to equation (2). For example what if want to make changes on the model under investigation, the question that arises do we need to solve the new problem from the beginning. [Fas13] states that the liner system $\mathbf{Ax = b}$ may be perturbed as $\mathbf{A(x + \delta x) = (b + \delta b)}$ this implies that$\mathbf{A\delta x = \delta b}$, and hence we can solve this linear equation to get $\mathbf{\delta x}$ and add to solution of the original problem and get $\mathbf{x + \delta x}$, i.e the solution of the perturbed problem.

1. The Effect of Changes in A, and b on Relative Error

If we solve (1) without including all the positional constraints we will have a deformed object of the original object. So what if we want to add more constraints to our problem latter. Again the question arises: Do we need to solve a new problem from the beginning? The answer is no, we can work on the perturbed system $\mathbf{(A + \delta A)(x + \delta x) = (b + \delta b)}$ to find:

$$\mathbf{\delta x = (A + \delta A)^{-1}(\delta b - \delta Ax)}$$

Implementation details and Results

We have used Matlab, and Graph tool box created by "Gabriel Peyre". To calculate the least squares solution "mldivide" command was used. Due to the nature of the problem the solver used is Sparse QR Factorization via "SuiteSparseQR".

The Effect of Changes in b on Relative Error

The changes in b can result from multiple factors: aging that occurred on an object, or simply because we want to make a change on the object replacing a piece by another piece.

In figure (4) we see the effect of modifying part of the model

The Implementation details go as follows:

Solve the original problem $\mathbf{Ax} = \mathbf{b}$ to get $\mathbf{x}$

Introduce a delta change on the b part in a limited number of vertices (about 10% of the total number)

Calculate $\delta\mathbf{b}$

Calculate $\delta\mathbf{x}$ from $\delta\mathbf{x} = \mathbf{A}^{-1}\delta\mathbf{b}$

Add $\mathbf{x} + \delta\mathbf{x}$

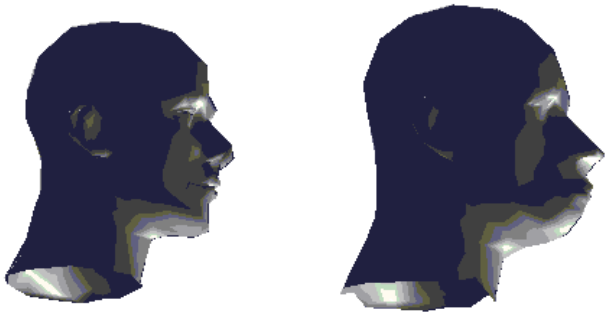


Figure (4), on the left the original object, on the right the perturbed object

3. The Effect of Changes in A, and b on Relative Error

- If we don't include all the positional constraints the solution will be deformed as can be seen in figure (5). In this figure only half of the positional constraints were used.
- If we now create $\delta\mathbf{A}$ that has the remaining positional constraints and also $\delta\mathbf{b}$.
- Note by that the size of $\delta\mathbf{A}$ is different from $\mathbf{A}$, this also applies for $\mathbf{b}$ and $\delta\mathbf{b}$, and we need to modify both the size of $\mathbf{A}$ and $\mathbf{b}$.
- [HL95] suggests a method to modify $\mathbf{A}$, and $\mathbf{b}$. What we need to do is to add rows to both $\mathbf{A}$, and $\mathbf{b}$ to represent the remaining positional constraints, so we can pretend that theses rows were actually in the original model and that they are filled with zeros.
- The Implementation details go as follows:
- Solve the original problem $\mathbf{Ax} = \mathbf{b}$ to get $\mathbf{x}$. Note by that this x is a result of only including half of the positional constraint
- Create $\delta\mathbf{b}$ and $\delta\mathbf{A}$ that account for the remaining positional constraints
- Modify $\mathbf{b}$ and $\mathbf{A}$ of the original problem to have the perturbed problem size
- Calculate $\delta\mathbf{x}$ from $\delta\mathbf{x} = (\mathbf{A} + \delta\mathbf{A})^{-1}(\delta\mathbf{b} - \delta\mathbf{A}\mathbf{x})$
- Add $\mathbf{x} + \delta\mathbf{x}$ to get the solution

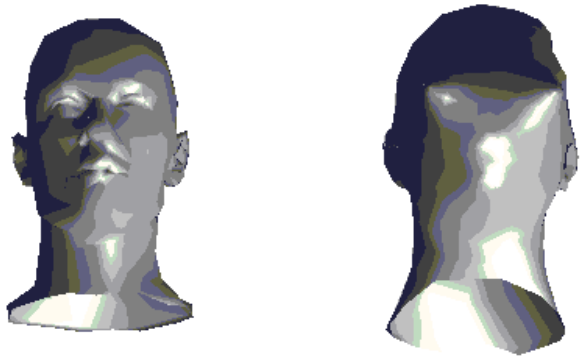


Figure (5) on the left the original object, on the right only half of the positional constraints were used

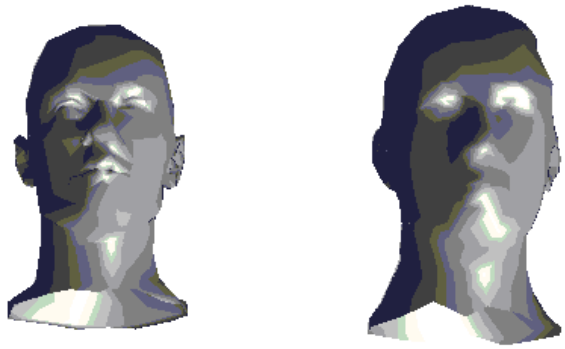


Figure (6) on the left the original object, the right image represents the solution after adding all the positional constraints and solving using the perturbation method

As can be seen from graph [2] that there is always time saving –the execution time varies from run to sun but always less from solving the complete problem from scratch- if we calculate the solution using the previous method due to changes in ($\delta\mathbf{A}$ and $\delta\mathbf{b}$) rather than solving a new problem with new $\mathbf{A}$ and $\mathbf{b}$.

This sounds reasonable, since for small $\delta\mathbf{A}$ and $\delta\mathbf{b}$ the linear system is sparser, this means that having a smaller change leads to a faster solution.

Graph (4): Percentage Time saving

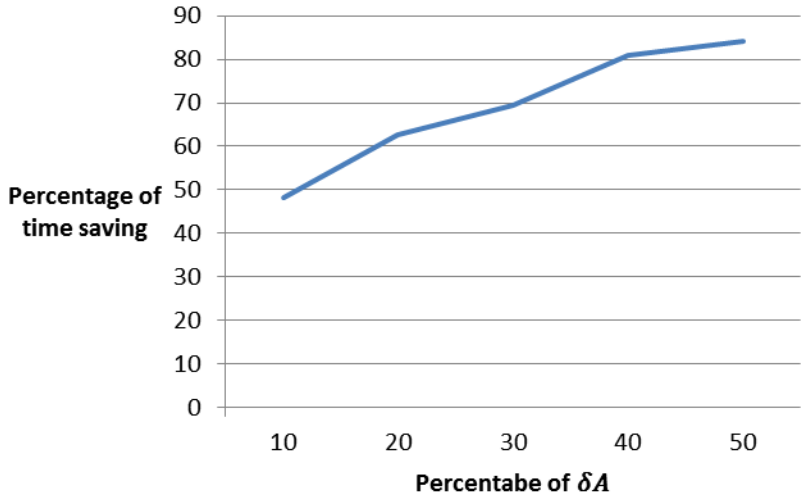


Table [2] statistical Metrics

Metric	Changes in A, and b on Relative Error	Changes in b on Relative Error
Mean Normalized Cross-Correlation $NK = \sum_{j=1}^{M}\sum_{k=1}^{N} x_{j,k} \cdot x'_{j,k} \Big/ \sum_{j=1}^{M}\sum_{k=1}^{N} x_{j,k}^2$	0.9820	0.9600
Average Difference $AD = \sum_{j=1}^{M}\sum_{k=1}^{N} \left(x_{j,k} - x'_{j,k}\right) \Big/ MN$	0.3649	3.0639
Normalized Absolute Error $NAE = \sum_{j=1}^{M}\sum_{k=1}^{N} \left\lvert x_{j,k} - x'_{j,k} \right\rvert \Big/ \sum_{j=1}^{M}\sum_{k=1}^{N} \lvert x_{j,k} \rvert$	0.0448	0.0768

We can also see in table [2] that if we compare the images of the generated objects with the image of the original object that: in case of changes in A, and b on Relative Error that Average Difference is small, and also the cross correlation and the Absolute error are very small which means that our method was successful.

While in the case of changes in b on Relative Error, the average error is large which is reasonable since we have modified the original image and made changes to it, while the cross correlation and the Absolute error are very small.

Conclusion

We have presented two time saving techniques to find the solution of the Linear equation resulting from the Laplacian Framework. It was shown that time execution drops dramatically when dividing the problem to multiple processors. Also we can find solutions to delta changes of the linear system in less time required to solve a new problem.

References:

[BGAA12] Baerentzen J., Gravesen J., Anton F., Aanaes H.: Guide to Computational Geometry Processing. Springer (2012)

[BKPAL 10] Botsch M., Kobbelt L., Pauly M., Alliez P., Levy B.: Polygon Mesh Processing. AK Peters,Wellesley (2010)

[Fas13] Fasshauer G.: online notes for course MATH 477/577 at IIT

[Hea97] Heath M.: Scientific Computing An Introductory Survey. McGraw-Hill (1997)

[HL95] Hillier F., Lieberman G.: Introduction to Operations Research. McGraw hill (1995)
[Mey00] Meyer C. Matrix Analysis and Applied Linear Algebra. SIAM (2000)
[NISA06] Nealen A., Igarashi T., Sorkine O., Alexa M.: Laplacian mesh optimization. Proceedings of ACM GRAPHITE (2006)
[Pey04] Peyre G.: Toolbox Graph Mathworks (2004)
[Sor05] Sorkine O.: Laplacian Mesh Processing. Proceedings of EUROGRAPHICS 2005, STAR Volume.
[TB97] Trefethen L., Bau D. : Numerical Linear Algebra. SIAM (1997)

Mohamed Elkomy received his B.Sc. in Electrical Engineering from Ain Shams University in 2001. After working as a teaching Assistant (from 2006 to 2007) he has received his Master in Computing from the American university in Cairo in 2009. His research interest includes Computational Geometry, Geometric Processing, and Electrical CAD design.