Parallelization of Mesh Contraction and Fairing using OpenCL

Martin Madaras[*]
Comenius University Bratislava

Roman Ďurikovič[†]
Comenius University Bratislava

Abstract

We propose a parallel method for computing local Laplacian curvature flows for triangular meshes. Laplace operator is widely used in mesh processing for mesh fairing, noise removal or curvature estimation. If the Laplacian flow is used in global sense constraining a whole mesh with an iterative weighted linear system, it can be used even for mesh contraction. However, numerical solution of such a global linear system is computationally expensive. Therefore, we have developed a method to compute such an iterative linear system using only local neighbourhoods of each vertex in parallel. Parallel computation of local linear systems is performed on GPU using OpenCL. We have evaluated speedups of the parallelization using both local and global Laplacian flows. We show test cases, where the parallel local method can be used for mesh fairing. In contrary, we also investigate and outline a fail case, where the local Laplacian flow cannot be used. When the local Laplacian flow has problems with global convergence, we offer a global parallelization of the linear system solving as an alternative.

CR Categories: Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

Keywords: mesh fairing, mesh contraction, Laplace operator, parallelization

1 Introduction

A polygonal mesh smoothing process called mesh fairing lowers the variation of maximum principal curvature across the surface. Mesh fairing is usually used for improving the quality of final meshes obtained by geometric modeling or just to smooth noisy input data. This can be either achieved by mesh subdivision and smoothing, or by moving mesh vertices of the original mesh, while the topology and connectivity of the mesh remains. New vertex positions are generated in order to minimize a surface energy [Desbrun et al. 1999]. Suitable energy function would be based on principal curvatures or thin-shell energies that measures stretching and bending. However, minimizing of such a non-linear energies becomes very computationally expensive and numerically unstable. Furthermore, evaluation of curvature on mesh data is an undefined operator in the geometric sense. This leads to the first and second order Laplacians. Using the Laplace operator, the mesh fairing process can be performed as diffusion process in several time steps. When working with polygonal meshes, the Laplace operator has to be approximated by discrete function defined on the mesh vertices. Therefore, the differences are replaced by partial derivatives of a displacement function [Botsch et al. 2010]. During the smoothing process using Laplace operator, the volume of the mesh is reduced as well. If this is not desired, the volume has to be restored. However, this effect can be used to contract the mesh into a zero volume [Au et al. 2008]. The smoothing process has to be iteratively applied on the mesh with well-defined contraction weights that drive the contraction process into final state. This iterative process can be computationally expensive with complex meshes. Therefore, we have developed a parallel method to apply Laplacian smoothing on the mesh. The mesh is decomposed into local patches constructed of local neighbourhoods of each vertex. This results into solving of N small systems instead of one big global system. We provide OpenCL [OpenCL and Munshi 2013] implementation using interoperability with OpenGL. When geometry is already loaded on the GPU in Vertex Buffer Object (VBO), OpenCL reads the buffer data on GPU, fair the mesh using local Laplacian flow and writes to modified geometry into the buffer. We show the case of mesh fairing and minor mesh contractions, where this local method can be used. We also analyze the case, why this method cannot be used for contraction of the mesh into zero volume, because the convergence is not guaranteed.

[*]e-mail: madaras@sccg.sk
[†]e-mail: durikovic@fmph.uniba.sk

2 Related Work

A linear interpolation at each vertex can be used to approximate Laplace operator. In this case, the approximation is called Umbrella operator [Kobbelt et al. 1998]. Umbrella operator uses uniform interpolation and therefore is inappropriate for non-uniform meshes.Thus, an operator based on curvature flow is more suitable. In addition, Laplacian of the surface at a vertex has both normal and tangent components, so the Laplacian is rarely being zero even if the surface is locally flat. Curvature flow smooths the surface by moving along the surface normal and can be calculated using cotangent scheme [Desbrun et al. 1999]. Curvature normal using cotangent scheme can be computed from vertex positions, polygon areas and mesh angles as:

$$\Delta_{f(v_i)} = \frac{1}{2A_i} \sum_{v_j \in N_1(v_i)} (\cot \alpha_{i,j} + \cot \beta_{i,j})(f_j - f_i), \qquad (1)$$

where A_i is an area of the one-ring neighborhood of vertex v_i and angles in the cotangent formula are computed as shown in Figure 1.

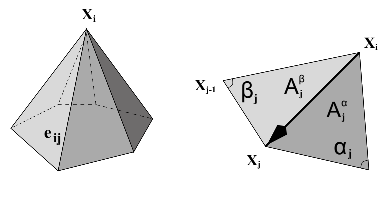

Figure 1: Opposite angles corresponding to the edge used in curvature flow calculation [Desbrun et al. 1999].

The Laplace operator can be used to approximate mesh curvature at given vertex. However, this method requires the input mesh to be closed connected 2D-manifold. When resulting vector given by the operator is globally minimized as proposed in [Au et al. 2008], underlying mesh tends to contract into zero volume. Contracted zero volume mesh can be used for extraction of a skeleton encoding topology of the input mesh. The same minimization method was used by [Cao et al. 2010] to extract skeleton from point clouds. Authors used weighted Gaussian interpolation of vertex positions of the point cloud to approximate the Laplacian.

Mesh processing algorithms with local behaviour are ideal candidates for parallel processing. Cotangent scheme in Equation 1 can be evaluated for each vertex in parallel. Values in the equations depend only on local neighbourhood of the vertex. There are few possibilities how to implement such a parallel evaluation, for example CUDA, OpenCL or Compute Shaders. For us the OpenCL is the most suitable environment, because it runs on all processors and all types of GPU.

3 Mesh Contraction Algorithm

Here we offer a quick overview of an iterative mesh contraction algorithm without parallelization. In the preprocessing stage before each iteration a matrix describing Laplacian curvature flow is constructed as in Equation 2. The matrix looks like connectivity matrix of a graph with cotangent weighting. In each column, the diagonal value is equal to a negative sum of values in each row.

In each iteration the mesh graph vertices are contracted by Laplacian smoothing. To compute new vertex position, the linear system (Equation 3) is solved in least-squares sense by QR decomposition [Gentle 1998]. Some examples of visualized contraction steps can be seen in Figure 2.

Vertex positions are smoothly contracted along their normals by solving the Equation 3. Laplacian smoothing operator L is $N \times N$ square matrix:

$$L_{ij} = \begin{cases} w_{ij} = \cot\alpha_{ij} + \cot\beta_{ij} & \text{if } (i,j) \in E \\ \sum_{(i,k)\in E}^{k} -w_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where E is the set of edges in the input triangular mesh and α_{ij}, β_{ij} are the opposite angles corresponding to the edge (i, j).

This operator is applied on N vertices in vector V as a filter. Term LV approximates curvature flow normals, so solving $LV' = 0$ removes normal components of vertices and contracts the geometry, resulting into a new set of vertices V'. The Laplacian coordinates $LV = [\delta_1^T, \delta_2^T, \delta_3^T, \ldots, \delta_n^T]^T$ can be referred to as contraction constrains, because they provide the forces to contract the mesh. The term δ_i equals $-4A_i k_i n_i$, where A_i, k_i, n_i is the area of adjacent faces surrounding the vertex i, the approximate local mean curvature and the approximate outward normal of vertex i, respectively. Thus, for vertices with the same shape of the surrounding area, the scale of the Laplace operator coordinate is proportional to the length of the edges adjacent to the vertex.

3.1 Linear System

As mentioned in [Au et al. 2008], an unconstrained solving of this equation contracts the mesh into a single point, so the equation is solved in more iterations with carefully chosen weights which control the contractions. Weights W_L and W_H are diagonal matrices which control the contraction process. Weighting matrices have to be updated after each iteration to drive the iteration process into a desired state. By increasing $W_{L,i}$ we can increase the collapsing speed for vertex i and by increasing $W_{H,i}$ we increase the attraction weight to attract vertex i to its current position. All the $W_{L,i}$ are in the next step multiplied by a predefined constant s_L and $W_{H,i}$ are updated in such a way that the attraction weight is multiplied by a ratio of the change of the area of faces adjacent to the vertex i. If the area of adjacent faces surrounding vertex i is smaller, the multiplicative term is higher. Thus, if the vertex i is more attracted into its current position, then the geometry is less contracted in the vertex i in the next iteration. Final global linear system can be written as:

$$\begin{bmatrix} \mathrm{W}_L L \\ \mathrm{W}_H \end{bmatrix} V' = \begin{bmatrix} 0 \\ \mathrm{W}_H V \end{bmatrix}. \tag{3}$$

Each step of iterative contraction process works as follows (t denotes the iteration number):

1. Solve $\begin{bmatrix} \mathrm{W}_L^t L^t \\ \mathrm{W}_H^t \end{bmatrix} V^{t+1} = \begin{bmatrix} 0 \\ \mathrm{W}_H^t V^t \end{bmatrix}$
2. Update $W_L^{t+1} = s_L W_L^t$ and $W_{H,i}^{t+1} = W_{H,i}^0 \sqrt{A_i^0 / A_i^t}$, where A_i^0 and A_i^t are the original and current areas of adjacent faces for vertex i, respectively.
3. Compute the new Laplace operator L^{t+1} with the vertex positions computed from previous iteration V^{t+1} using Equation 3.

In this iterative process, L is sparse squared $N \times N$ matrix. Matrices W_L and W_H are diagonal $N \times N$ squared weighting matrices. Terms V' and V are $N \times 3$ matrices with vertex positions in three dimensions. Thus, the system can be solved in each dimension independently. In this case, 0 upper matrix is a zero vector of size N.

This results into an overdetermined linear system $Ax = b$, where matrix A is $2N \times N$ sparse matrix. Such a system with rectangular matrix A can be solved in a linear least squares manner using normal equations [Weisstein 2013] or QR decomposition.

3.2 Parallelization

There are few options how to parallelize the mesh contraction process. Here we analyze all three possibilities of parallelization and discuss, which one is suitable for mesh fairing and mesh contraction to zero volume.

The first option is the parallelization of QR decomposition process. The QR algorithm decomposes matrix A into a product $A = QR$ of an orthogonal matrix Q and upper triangular matrix R. Matrix Q is orthogonal and therefore $Q^T Q = I$ and solving linear system $Ax = b$ is equal to solving linear system $Rx = Q^T b$. An Householder transformation [Hou 1953] is used to compute matrix Q and R. For this we have used a free open-source scientific computing library ViennaCL [Rupp 2013]. ViennaCL offers parallel QR decomposition using OpenCL backend. When matrices Q and R are obtained, the linear system $Rx = Q^T b$ can be evaluated very fast even on CPU, because R is a triangular matrix.

The second option is to multiply the linear system with A^T and create normal equation $A^T Ax = A^T b$. This leads into a new linear

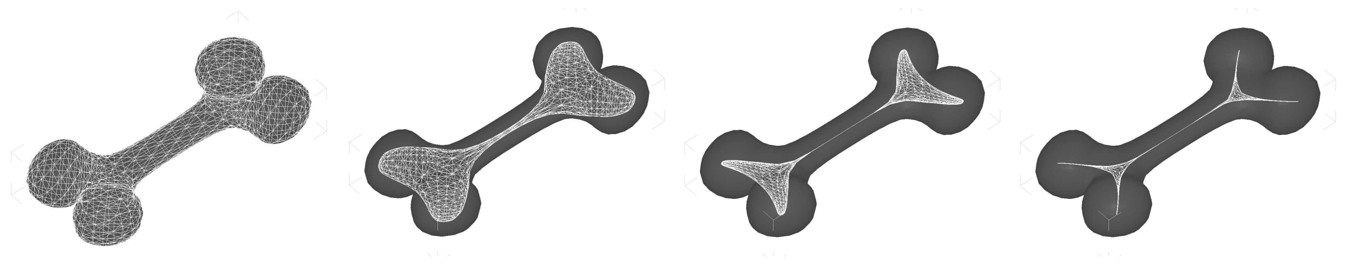

Figure 2: Examples of mesh contraction. From left to right: converted mesh graph, contraction after 1 iteration, 2 iterations and after 3 iterations, where the volume is close to zero.

system $A^{'}x = b^{'}$ which can be solved using parallel squared linear solver. A straightforward way to parallelize solving of a linear system is an iterative Jacobi method [Acton 1990]. When solving the linear system $A^T Ax = A^T b$ (Equation 4), matrix A is decomposed into upper triangular matrix U, lower triangular matrix L and diagonal matrix D. Iterative evaluating of Equation 5 is needed to correct approximation of vector x. Each term of vector x can be computed independently, thus each vertex can be processed individually using Equation 6.

$$\sum_{j=i}^{n} a_{ij} x_i = b_i \tag{4}$$

$$\mathbf{x}^{(k)} = D^{-1}(L+U)\mathbf{x}^{(k-1)} + D^{-1}\mathbf{b} \tag{5}$$

$$x_i^{(k)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(k-1)}}{a_{ii}} \tag{6}$$

Jacobi method guarantee to converge, if matrix A is strictly diagonally dominant. Unfortunately, matrices obtained applying normal equations on Laplacian flow matrix do not have to satisfy this condition. Therefore Jacobi method will not converge for every input mesh and cannot be used.

The third option is a hypothesis that the problem can be solved locally. We construct the Laplacian locally and solve N small linear systems in parallel. For this we use only local neighborhoods of each vertex. This approach is described more detailed in the next section (Section 4).

4 Local Laplacian Curvature Flows on GPU

In this section we discuss construction of local Laplacian and solving of linear equations from parallel point of view focusing on GPU implementation. In the preprocessing stage (Subsection 4.1), a global connectivity matrix and local neighbourhood for each vertex are constructed. Those matrices are passed to GPU and VBO is created. As next step, OpenCL kernel is iteratively called (Subsection 4.2). Each kernel processes local neighbourhood of a vertex. Local neighbourhood of one vertex overlaps with local neighbourhoods of another vertices as can be seen in Figure 3. Thus, only new position of center vertex itself is written into VBO. New positions of other vertices from local neighbourhood is discarded. These vertices are computed by their own threads, computing theirs positions in parallel.

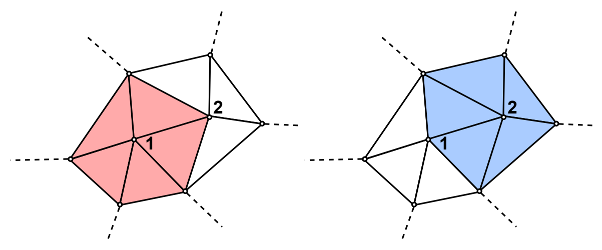

Figure 3: Local neighbourhood is created from one-ring vertices around each vertex. Local neighbourhood of vertex 1 is colored with red and local neighbourhood of vertex 2 is colored as blue.

4.1 Preprocessing Step

As mentioned before, in the preprocessing step the global connectivity matrix E is constructed from the input mesh. Furthermore, for each mesh vertex v_i, local neighbourhood is constructed from one-ring area around the vertex. From this local neighbourhood is created a vector n_i as $n_i = (s_i, l_0, l_1, l_2 \ldots l_{si})$, where s_i is the size of one-ring local neighbourhood and l_i are the indices of these vertices. From vector n_i for each vertex, a matrix N is composed and matrices E, N are passed into global OpenCL memory objects. Finally, VBO with vertex positions is created from the input geometry.

4.2 Iterations

Each iteration step is composed of few sub-steps. As first, local Laplacian curvature flow matrix L_i is constructed. For this, a cotangent Laplacian flow is used (Equation 2). As second, linear system is constructed, QR decomposition is performed and new vertex positions are computed. For all of this, only local GPU memory is used allocated for each kernel. The final position of vertex i is taken and written into VBO in parallel. When all kernels have finished, we have new mesh vertex positions stored in VBO.

4.3 Implementation Details

When porting our Laplacian system on GPU, we have to take care of the memory management. Matrices that are created in preprocessing step are written into global memory, so all kernel threads have access to them. This is global connectivity matrix E and matrix N holding neighbourhood information around each vertex. Variables that are used in the iteration are declared as private memory objects and they are unique for each kernel.

	#vertices	Global CPU Jama	VCL-QR	VCL-LSM	Local CPU Jama	OpenCL transfer	OpenCL interop
Iteration	648	0,07	0,147	0,083	0,19	0,35	0,19
	1416	0,51	2,51	0,405	0,75	1,3	0,295
	5664	32,5	130,5	15,23	9,35	3,38	0,975
	14952	701	1654	181,5	66,5	12,1	2,065
Preprocess	648	0,012	0,012	0,012	0,012	3,79	3,54
	1416	0,025	0,025	0,025	0,025	4,113	3,928
	5664	0,413	0,413	0,413	0,413	5,391	4,806
	14952	2,018	2,018	2,018	2,018	12,047	11,062
Overall	648	0,222	0,453	0,261	5,712	14,29	9,24
	1416	1,555	7,555	1,24	22,525	43,113	12,778
	5664	97,913	391,913	46,103	280,913	106,791	34,056
	14952	2105,018	4964,018	546,518	1997,018	375,047	73,012

Table 1: Table shows the running time (in seconds) of evaluated local and global methods for mesh contraction. The best speedup we have achieved for global method is using VCL-LSM (546,518s / 3 iterations) and for local method using OpenCL interop (73,012s / 30 iterations).

When rendering each step of the iterative mesh contraction, a feature we are using is OpenCL-OpenGL interoperability inside of multi-threading environment. The interoperability paradigm enables sharing contexts between OpenGL and OpenCL. Therefore, when processing geometry in OpenCL kernels, OpenCL gets locks for OpenGL VBO, processes the geometry and retrieves the lock to OpenGL so the rendering process may continue. All this is done strictly on GPU, without any CPU-GPU bandwidth. However, in our implementation we want to iteratively apply mesh processing kernel and after each iteration render the geometry from VBO to visualize the iteration step. Therefore, we are using two threads for this. One thread takes care of all OpenGL calls and rendering process. The second thread is geometry processing thread. It calls OpenCL kernels to contract the geometry. When the iteration is completed, the processing thread with shared context writes new positions into VBO and rendering thread can access the contracted geometry from the same VBO.

5 Results

All the input models are 2D manifolds with number of vertices shown in the Table 1. We have compared six different methods for mesh contraction (Table 2). There are three global (1,2,3) and three local methods (4,5,6). Methods 1 and 4 are computed sequentially on CPU using Jama library for QR decomposition, the rest is computed on GPU in parallel. Methods 2 and 3 use ViennaCL library for QR decomposition and methods 5 and 6 use local Laplacian schema implemented in OpenCL. The difference between QR decompositions using ViennaCL (VCL-QR and VCL-LSM) is that VCL-QR makes full QR decomposition on GPU and passes matrices Q and R on CPU, where final computation of $Rx = Q^T b$ is performed. Method VCL-LSM directly computes matrix R and $Y = Q^T b$ on GPU. We do not evaluate normal equations with Jacobi iterative method, because it does not numerically converge in the case of our geometric problem.

5.1 Convergence Problems

Local parallel method we have developed can be used for fast mesh fairing or small mesh contractions. Unfortunately, numerical tests show that local Laplacian schema do not converge into a zero volume meshes. This is, because global numerical constraints were removed by localizing the linear system. If global constraints are removed, contraction weights W_L and W_H do not control the contraction process anymore. Weighting matrices can be changed only

Method name	Laplacian flow	Processor
1. Global CPU Jama	global	CPU
2. VCL-QR	global	GPU
3. VCL-LSM	global	GPU
4. Local CPU Jama	local	CPU
5. OCL transfer	local	GPU
6. OCL interop	local	GPU

Table 2: Table shows description of methods used in the evaluation. We have evaluated three local and three global methods.

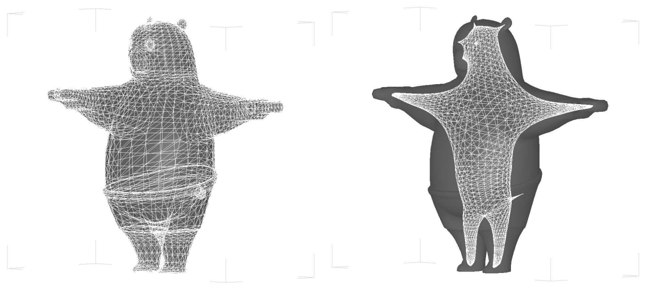

Figure 4: Local Laplacian flow fails to converge into zero volume skeleton even after 30 iterations.

locally and this does not drive the system into global minimum, only local minimum is achieved. An expansion of local neighbourhoods to bigger local patches was also evaluated. With bigger local neighbourhoods the method shrinks the mesh quicker, however still do not converge into zero volume. Shrinking process using local method can be seen in Figure 4. Here, 30 iterations of local Laplacian smoothing is applied on the mesh.

5.2 Computation Times

Running times of contraction methods is shown in Table 1. The algorithm is divided into a preprocessing step and an iteration. Overall timing is computed as 1x preprocessing step and 3x iteration step for global or 30x iteration step for local method. Speedup graphs can be seen in Figure 5. For contraction of mesh into zero volume, global method using parallel QR decomposition is the best choice (VCL-LSM). In this case, a speedup by one order of magnitude can be achieved using parallelization. When mesh fairing or only low contraction is the objective, local Laplacian flow can be used with much higher speedup (50-100x).

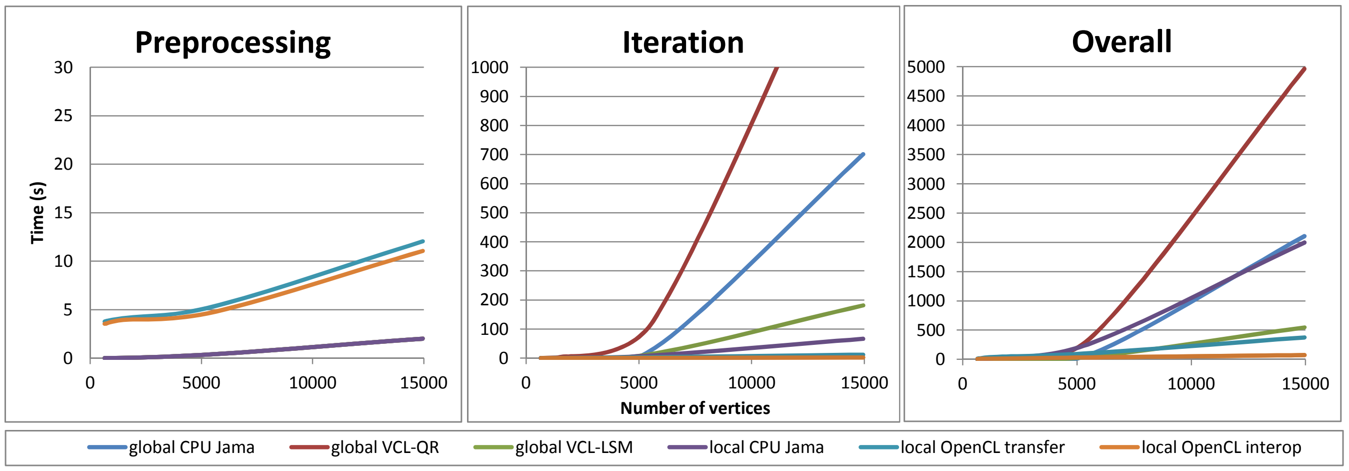

Figure 5: Graphs show execution times of each step of algorithm. Left: preprocessing, matrix creation, CPU-GPU transfer, OpenCL kernel building. Middle: one iteration of solving Laplacian linear system using QR decomposition. Right: overall running time, 3 iterations for global method and 30 iterations for local methods.

All the execution times are measured in seconds on Intel i7-2640 @ 2.8Ghz using single thread implementation and AMD Radeon 6630M with 480 stream processors.

6 Conclusion and Future Work

In this paper we offer three contributions concerning mesh smoothing and contracting. Firstly, we have developed a parallel schema using local Laplacian curvature flows for mesh contraction. This schema do not converge into zero volume as predicted and therefore cannot be used for skeleton extraction. However, local Laplacian schema is good tool for mesh fairing. Using one or more iterations, mesh can be slightly contracted and smoothed and finally the volume has to be restored as described in [Desbrun et al. 1999]. Secondly, the paper analyses the best parallelization of local and global method from the time complexity point of view. We compare six schemes for iterative mesh contraction with both, local and global characteristics implemented on GPU and CPU. As third, the best global method for skeleton extraction is picked and tested. When local Laplacian schema fails to converge, a global method using parallelized QR decomposition can be used alternatively as a best candidate. Parallel QR decomposition on GPU is stable, works well on all tested input meshes and still gives speedup by one order of magnitude if compared to sequentially CPU method.

As a future work we plan to investigate parallelization using normal equations and Jacobi method more deeply in a numerical way. Jacobi method does not converge into the solution, however so called Jacobi extrapolated method may converge in our case. We would like to investigate this method and try to implement it in parallel. Another idea is to use Gauss-Seidel method, that can be implemented parallel in some cases [Courtecuisse and Allard 2009] and this method should converge also for matrices that are not diagonally dominant, as in our this case.

7 Acknowledgment

Our project was supported from the grant No. UK/375/2013.

References

ACTON, F. S. 1990. *Numerical Methods That Work, 2nd printing*. Math. Assoc. Amer.

AU, O. K.-C., TAI, C.-L., CHU, H.-K., COHEN-OR, D., AND LEE, T.-Y. 2008. Skeleton extraction by mesh contraction. In *ACM SIGGRAPH 2008 papers*, 1–10.

BOTSCH, M., KOBBELT, L., AND LEVY, B. 2010. *Polygon Mesh Processing*. Ak Peters Series. A K Peters.

CAO, J., TAGLIASACCHI, A., OLSON, M., ZHANG, H., AND SU, Z. 2010. Point cloud skeletons via laplacian based contraction. In *Proceedings of the 2010 SMI Conf.*, 187–197.

COURTECUISSE, H., AND ALLARD, J. 2009. Parallel dense gauss-seidel algorithm on many-core processors. In *High Performance Computation Conference (HPCC)*, IEEE CS Press, 139–147.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 317–324.

GENTLE, J. E. 1998. *Numerical Linear Algebra for Applications in Statistics*. Springer-Verlag.

1953. *Principles of Numerical Analysis*. McGraw-Hill.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P., 1998. Interactive multi-resolution modeling on arbitrary meshes.

OPENCL, K., AND MUNSHI, A., 2013. The opencl specification version: 1.0 document revision: 48.

RUPP, K., 2013. ViennaCL.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '95, 351–358.

WEISSTEIN, E. W., 2013. Normal equation.