Simple and Effective GPU-based Mesh Optimization

Eric Shaffer[1], Zuofu Cheng[1], Raine Yeh[1], George Zagaris[2], and Luke Olson[1]

[1] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 shaffer1@illinois.edu
[2] Kitware Inc. george.zagaris@gmail.com

Summary. We present a GPU-based algorithm for optimizing the shape of elements in tetrahedral volume meshes. To exploit the massive parallelism available in modern GPU hardware, optimization occurs on a per-vertex basis using only local neighborhood information. A classical derivative-free numerical optimization method is employed to optimize the minimum dihedral angle of the component tetrahedron elements. Preliminary results demonstrate the GPU as a promising platform for mesh optimization with notable speedups. The final mesh exhibits high-quality comparable to state-of-the-art CPU techniques. Ongoing efforts are focused on further performance optimizations as well as scaling to multiple GPUs.

Key words: Mesh optimization; parallel algorithms.

1 Introduction

Mesh quality is a key concern in engineering and scientific computing applications. The shape of mesh elements can significantly impact the efficiency and accuracy of simulation codes. In this paper we describe what is to our knowledge the first application of GPU computing to the problem of element shape optimization. Optimization is accomplished on a per-vertex basis, with each GPU thread assigned the task of optimizing a vertex position by employing a classic derivative-free numerical optimization technique. This algorithm is simple to implement and fast. In our experiments, the GPU-based algorithm was shown to converge to a high-quality solution up to three times faster than a state-of-the-art serial method for large meshes. These results demonstrate the scalability and effectiveness of the GPU as a platform for mesh optimization.

2 Background

Mesh optimization is a well-researched activity with an abundant body of literature that aims to improve the quality of the mesh by altering mesh characteristics without fundamentally impacting the accuracy with which the mesh models a domain. For example, an early mesh optimization (or smoothing) technique was based on repeated averaging of a vertex position in a neighborhood [9]. Newer techniques cast mesh smoothing as a numerical optimization problem. Here, an objective function is defined using a quality metric measuring desired geometric properties. Examples include optimizing the inverse mean-ratio metric [4, 10] or the condition number of the Jacobian matrix [7]. Freitag, Knupp and others developed Mesquite [3], a linkable software library for mesh quality improvement through vertex repositioning which we have chosen to measure the efficacy of our proposed algorithm.

Our approach to GPU-based mesh optimization was informed by the work of Freitag *et al.* on parallel mesh optimization in a classical distributed memory computing environment [2]. Their algorithm seeks out parallelism by decomposing the mesh optimization problem on a per-vertex basis. When optimizing the position of a single vertex v, the domain of the optimization problem becomes the set of elements E_v neighboring v. An objective function is created by combining the quality measures of the elements in E_v in some manner, which was shown to generate good results for cluster-style parallelism in [2]. Our work as demonstrated in [12] show that for surface mesh optimization this fine-grained decomposition is appropriate for the massively-threaded nature of modern GPU processors.

3 Volume Mesh Optimization

We start the optimization problem by defining the quality metric of a tetrahedron as the minimum dihedral angle. This measure is commonly used and very intuitive, and therefore well-suited to our purpose. For each vertex, we consider the best position as the one which results in the minimum objective function evaluation, which in our case is the reciprocal of the minimum dihedral angle among the tetrahedron that include that vertex. Notice that this this objective function is not smooth; discontinuities can occur at points where the minimum dihedral angle shifts from one element to another as the vertex is moved. This motivates our choice of derivative-free optimization methods, similar to the multi-directional search algorithm proposed by [11]. Therefore, we have chosen to employ the Nelder-Mead Simplex method [1].

3.1 GPU Implementation

The first step in the GPU optimization algorithm is to find the surface vertices, which are fixed to prevent alteration of the shape of the mesh. The non-surface

vertices are then labeled and parsed by the CPU to yield a precomputed independent neighborhood, which corresponds to the coordinate positions of the vertices which the objective function is evaluated among. Each vertex must not be simultaneously optimized as its neighbor, which is guaranteed by a simple First Fit labeling of the vertices [8] included in the pre-computation. This data is initially stream into the GPU *global memory*, but is immediately loaded in parallel to the *shared memory* of each thread block, which is either 16 or 32 threads. The shared memory acts as a manually managed cache, allowing us to evaluate the objective function without having to read out of global memory. The variability in block size is due to the variable neighborhood size of each tetrahedron and the fact that all the neighborhood vertices for the 16 or 32 vertices to be optimized must fit into 48 kilobytes of shared memory. Each thread then generates a local simplex used for optimization, which is a tetrahedron given by corners perturbed by a factor scaled by the minimum length of incident edges to the mesh vertex.

Once the initial simplices are computed, each vertex begins its independent optimization following the Nelder-Mead algorithm. For each of the four points in the simplex, the objective function is given by

$$Q(p_v) = \frac{1}{min(\theta_v)}$$

where p is the position of each simplex of vertex v and θ_v are the dihedral angles of the tetrahedron elements that include vertex v. There is an additional condition which assigns a high value for $Q(p_v)$ if the proposed p_v is outside a pre-determined allowable radius to prevent mesh inversion. Based on the relative values of the function at the sampled points, the simplex is transformed using a combination of expansion, reflection, and contraction operations. These operations move one or more corners of the locally stored simplex based on a step-size parameter. Ultimately, the simplex should flow to areas yielding lower function values and contract around a minimum, yielding the optimal position of the vertex. For a complete explanation of the Nelder-Mead algorithm, the book by Conn *et al.*[1] is an excellent resource. After the last iteration the result is written out and replaces the original mesh vertex position with the newly found best simplex vertex. When all the threads of the given set have completed, the updated vertex positions are copied back into the host memory and a new kernel is launched with the next independent set.

3.2 Experimental Results

In assessing the effectiveness of the proposed algorithm, we consider its ability to improve the minimum dihedral angle found in a mesh. Our code demonstrably accomplishes this in all our test meshes except for the Large Rocket mesh in which the minimum dihedral angle is locked by its occurrence on the boundary. Even so, the minimum interior dihedral angle moves from 13 to 22

degrees. Angle improvements among other meshes range from around 30 to 122 percent.

Table 1. Performance on tetrahedral meshes for the GPU algorithm and Mesquite. Vertex count includes only interior vertices. Interior angles for Big Rocket indicated in parentheses.

Mesh	Number of Vertices	Number of Tets	Method	Converge Time	Dihedral Angle (°)		Inv. Mean Ratio	
					min	avg min	max	avg
Small Rocket	58981	468623	unopt.	—	12.5	47.5	2.2	1.1
			GPU	7s	21.2	47	2.6	1.1
			Mesquite	4s	12.2	49	2.2	1.1
Small Sphere	70849	1166714	unopt.	—	5	40	5.7	1.6
			GPU	5s	7	39	5.8	1.6
			Mesquite	12s	6	41	3.8	1.4
Big Sphere	290739	4720255	unopt.	—	4.5	41	8.6	1.6
			GPU	20s	6	41	5.3	1.6
			Mesquite	2m	5.4	41	5.5	1.5
Big Rocket	2202793	14992367	unopt.	—	11 (13)	46	2.5	1.2
			GPU	1m17s	11 (22.5)	46	2.5	1.2
			Mesquite	3m50	11 (19)	48	2.3	1.1
Wing	4484039	27725125	unopt.	—	5.8	51	6.1	1.1
			GPU	4m27s	12.9	51	3.3	1.1
			Mesquite	14m40s	7.2	52	3.7	1.1

We chose to compare the GPU-based algorithm with a mesh optimization code provided by Mesquite, which implements a current state-of-the-art algorithm as shown in Freitag *et. al*[5, 6]. The serial Mesquite algorithm employs gradient-based optimization which typically converges much faster than a derivative-free method. In order to achieve fine-grained parallelism, our algorithm formulates mesh optimization in terms of finding the maximum of a non-smooth function. This makes it problematic to employ the faster gradient-based algorithms. However, as shown in Table 1, the GPU-based algorithm exhibits a significant speedup over Mesquite for large meshes due to the parallelism offered by the fine-grained approach. For the largest mesh, the Wing, the GPU algorithm was 3 times faster than Mesquite and generated slightly better mesh quality. For smaller meshes, as seen in the Small Rocket, Mesquite converged faster than the GPU-based algorithm. Comparison of the GPU-based method to serial derivative-free optimization would exhibit even greater speedup, as demonstrated by the results for surface mesh optimization in [12] . In measuring performance, we compared the CPU time for Mesquite running on a Xeon X5550 to the combined CPU pre-processing and GPU time on the same computer with the addition of a Tesla C2070.

4 Conclusion and Future Work

The overall results in our experiments show that volume mesh optimization by GPU can be significantly faster than equivalent serial methods, suggesting that the GPU offers a very promising platform for mesh optimization. Moreover, we believe the local mesh optimization framework employed on the GPU will ultimately prove more scalable than global optimization techniques. We expect future work will focus on maximizing the scalability of GPU-based mesh optimization, allowing for out-of-core optimization as well as scaling to multi-core GPU systems.

References

1. Andrew R. Conn, Katya Scheinberg, and Luis N. Vicente. *Introduction to Derivative-Free Optimization.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.
2. Lori Freitag, Mark Jones, and Paul Plassmann. A parallel algorithm for mesh smoothing. *SIAM J. Sci. Comput.*, 20(6):2023–2040, 1999.
3. Lori Freitag, Patrick Knupp, Thomas Leurent, and Darryl Melander. MESQUITE design: Issues in the development of a mesh quality improvement toolkit. In *Proceedings of the 8th International Conference on Numerical Grid Generation in Computational Field Simulations*, pages 159–168, 2002.
4. Lori Freitag, Patrick Knupp, Todd Munson, and Suzanne Shontz. A comparison of two optimization methods for mesh quality improvement. In *Proceedings, 11th International Meshing Roundtable*, pages 29–40, September 2002.
5. Lori Freitag, Patrick Knupp, Todd Munson, and Suzanne Shontz. A comparison of inexact newton and coordinate descent mesh optimization techniques. In *Proceedings of the 13th International Meshing Roundtable*, pages 243–254, Williamsburg, VA, September 2004.
6. Lori Freitag, Patrick Knupp, Todd Munson, and Suzanne Shontz. A comparison of two optimization methods for mesh quality improvement. *Invited Submission. Engineering with Computers*, 22(2):61–74, May 2006.
7. Lori Freitag and Patrick M. Knupp. Tetrahedral element shape optimization via the jacobian determinant and condition number. In *Proceedings of the 8th International Meshing Roundtable*, pages 247–258, 1999.
8. A. Gyrfs and J. Lehel. On-line and first fit colorings of graphs. *Journal of Graph Theory*, 12(2):217–227, 1988.
9. Peter Hansbo. Generalized Laplacian smoothing of unstructured grids. *Communications in Numerical Methods in Engineering*, 11(5):455–464, 1995.
10. Todd Munson. Optimizing the quality of mesh elements. *SIAG/Optimization News and Views*, 16:27–34, 2005.
11. Jeonghyung Park and Suzanne M. Shontz. Two derivative-free optimization algorithms for mesh quality improvement. *Astrophysical Journal Supplement Series*, 186:457–484, 2010.
12. Eric Shaffer and George Zagaris. GPU accelerated derivative-free mesh optimization. In *GPU Computing Gems Jade Edition.* Morgan Kaufmann, 2011.