Discrete Mesh Optimization on GPU

Daniel Zint and Roberto Grosso

Abstract We present an algorithm called *discrete mesh optimization* (DMO), a greedy approach to topology-consistent mesh quality improvement. The method requires a quality metric for all element types that appear in a given mesh. It is easily adaptable to any mesh and metric as it does not rely on differentiable functions. We give examples for triangle, quadrilateral, and tetrahedral meshes and for various metrics. The method improves quality iteratively by finding the optimal position for each vertex on a discretized domain. We show that DMO outperforms other state of the art methods in terms of convergence and runtime.

1 Introduction

Simulations based on finite elements require meshes with high quality. Element shape has a strong impact on convergence [3, 32]. In cases like fluid simulations [1] anisotropy or a locally varying element size influence stability and accuracy. Such attributes of a mesh can be described with a quality metric. A vast range of smoothing methods considers purely element shape [7, 8, 11, 18, 17, 19, 21, 22, 24, 26, 37, 41, 42, 44, 45]. Changing the quality metric within these methods requires major changes in the algorithm structure.

We present *discrete mesh optimization* (DMO), a greedy approach to topology-consistent mesh smoothing. The exhaustive search efficiently exploits the parallel computing power of GPUs. Full utilization is achieved by applying a coloring scheme to update several independent vertices at once. Due to the versatility of

Daniel Zint
Computer Grahics Chair, Universität Erlangen-Nürnberg, Cauerstr. 11, 91058 Erlangen, Germany, e-mail: `daniel.zint@fau.de`

Roberto Grosso
Computer Grahics Chair, Universität Erlangen-Nürnberg, Cauerstr. 11, 91058 Erlangen, Germany, e-mail: `roberto.grosso@fau.de`

DMO it optimizes meshes of any kind for any quality metric without relying on differentiable functions. DMO aims to maximize the global mesh quality by iteratively finding the optimal position for each vertex with respect to its one-ring neighborhood. The optimal position is found by evaluating the quality metric on a coarse grid of candidate positions. The best candidate is chosen as new vertex position. The process is repeated several times while the grid spacing is reduced after every iteration. This results in a runtime comparable to Laplacian-based smoothing while creating high quality results. For example, with our setup DMO converges within 50 iterations on a mesh with 37 907 interior vertices taking around 7 ms. Besides the GPU version, DMO was also implemented on CPU. Even though DMO was developed to benefit from parallelization, it still outperforms other smoothing methods on CPU due to its fast convergence.

The next section introduces the *max-min* problem for mesh optimization. Sec. 3 gives an overview of smoothing methods. DMO is presented in Sec. 4. Examples and a performance analysis are stated in Sec. 5. Conclusions are given in Sec. 6.

2 The max-min Problem for Mesh Optimization

For each mesh element e_k a quality $q_\mathrm{k}^{(e)}$ is obtained by evaluating a quality metric $q^{(e)}(e_\mathrm{k})$. The quality metric should be strictly quasiconcave, continuous, and decaying from its unique global maximum. This holds for commonly used quality metrics. Other functions that do not fulfill this criteria do not guarantee optimal quality improvement.

The formulation of the mesh optimization problem is independent of the particular quality function $q^{(e)}$. Assume a mesh M in $\mathbb{R}^n$ which consists of n-dimensional elements. They do not have to be of one type, hybrid meshes are also possible.

For finite element simulations the minimal element quality is of importance [40]. Therefore, the quality $q_\mathrm{i}^{(v)}$ of a vertex v_i that is positioned at x is defined as the minimal quality of its incident elements $e_\mathrm{k} \in N_\mathrm{e}(v_\mathrm{i})$,

$$q_\mathrm{i}^{(v)}(x) = \min_{e_\mathrm{k} \in N_\mathrm{e}(v_\mathrm{i})} q_\mathrm{k}^{(e)}(x)\,, \tag{1}$$

where element quality $q_\mathrm{k}^{(e)}$ depends merely on the vertex position x as the one-ring neighborhood of v_i is kept fixed. If the element quality function $q_\mathrm{k}^{(e)}$ is quasiconcave, continuous, and decaying from its maximum, it follows that the vertex quality function $q_\mathrm{i}^{(v)}$ defined in (1) has a unique global maximum and is quasiconcave, [36].

From (1) follows the local optimization problem for finding the maximum quality $q_{\mathrm{i,max}}^{(v)}$ for a vertex v_i,

$$q_{\mathrm{i,max}}^{(v)} = \max_{x} q_\mathrm{i}^{(v)}(x) = \max_{x} \min_{e_\mathrm{k} \in N_\mathrm{e}(v_\mathrm{i})} q_\mathrm{k}^{(e)}(x)\,. \tag{2}$$

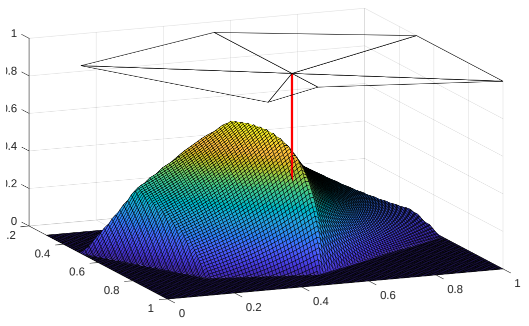


Fig. 1 Vertex quality function $q_{\mathrm{i}}^{(v)}$ for some one-ring neighborhood. Negative values were projected to zero. The red line points to the current quality of the vertex for the shown triangle fan.

The vertex quality function is nonlinear and nonsmooth (Fig. 1) and thus cannot be solved by any optimization method that requires derivatives. Because of the properties of the vertex quality function an efficient exhaustive search as proposed in our algorithm in Sec. 4.1 will certainly lead to the optimum of (2). The optimal position x^* for v_{i} is given as

$$x^* = \arg\max_x q_{\mathrm{i}}^{(v)}(x) = \arg\max_x \min_{e_{\mathrm{k}} \in N_{\mathrm{e}}(v_{\mathrm{i}})} q_{\mathrm{k}}^{(e)}(x)\,. \tag{3}$$

In the next section we summarize different smoothing methods intending to solve the *max-min* problem of mesh optimization. They mostly differ in their strategy of approximating eq. (3).

3 Related Work

Smoothing methods can be divided into three main groups, Laplacian-, physics-, and optimization-based. We will briefly discuss the three groups and give an overview of their advantages and disadvantages.

The classical Laplace-smoothing [15] is known to be fast but unstable in case of concave domains. A wide range of methods were proposed that modify the classical Laplace smoothing to overcome this problem [7, 8, 11, 19, 22, 24, 26, 29, 45]. Limitation still exists, for example angle-based smoothing [45] cannot deal with a strongly varying element size, Fig. 4. Another representative is the "Smart" Laplacian Smoothing of Freitag [19] which only performs smoothing when mesh quality is increased. Using "Smart" Laplacian Smoothing without further processing steps does not lead to satisfying results as it does not improve mesh quality considerably. Freitag proposed to use it in combination with an optimization-based method. This will be discussed later in this section. Laplacian-smoothers share the advantage of being fast but they also lead to suboptimal results. Many of these methods cannot guarantee that the quality will not decrease. DMO overcomes this problem by directly optimizing quality. Therefore, it will not converge to a suboptimal result and a non-decreasing quality can be guaranteed.

Physics-based methods consider the mesh as a physical model. Some examples are spring-mass systems [9, 14], truss networks [35], or elasticity models [4, 13, 38]. Just like Laplace-based methods they do not provide any guarantee of mesh improvement.

Optimization-based methods are named after their approach for optimizing a quality metric. Some methods try to overcome the problem of nondifferentiability by replacing eq. (2) with a smooth function [20, 27, 28, 43, 44]. They run into the same problems as Laplacian-based methods.

Another approach is taken by the previously mentioned method of Freitag [17, 18, 19, 21]. While in [18, 19] the optimization is done with an analogue of the steepest descent method for smooth functions, later versions use the simplex algorithm to solve a linear programming problem [17, 21]. To keep runtime low it combines Laplacian-based smoothing with an optimization-based approach. Thus, the method does not converge to the optimal solution in general. DMO avoids that by optimizing all vertices the same way.

A derivative-free approach is done by Park and Shontz in [33]. They use pattern search in combination with backtracking line search to find a better vertex position. The convergence is suboptimal as it depends on search directions.

Rangarajan and Lew introduced the directional vertex relaxation (DVR) algorithm [37]. It solves the optimization problem by breaking it down to one dimension. This is achieved by providing a smoothing direction which can be chosen either randomly or by using previous knowledge. Within this one dimension the optimal solution can be found analytically. The major concern about this method is its randomness of relaxation directions as it leads to an inefficient smoothing with slow convergence rates. In contrast, DMO is not restricted to search directions and therefore yields faster convergence than DVR or the method of Park and Shontz.

4 Discrete Mesh Optimization

DVR solves the *max-min* problem in eq. (2) analytically for only one direction. DMO uses a different strategy for solving the mesh optimization problem. We evaluate the vertex quality function with a greedy algorithm on a discretized domain. No derivatives are necessary which allows easy replacement of the quality metric.

In order to demonstrate the strength of DMO we use the standard mean ratio quality metric for triangles $q^{(e)}_{\mathrm{m_{tri}}}$ [2, 5, 6, 12, 17, 37],

$$q^{(e)}_{\mathrm{m_{tri}}} = 4\sqrt{3}\frac{A}{\sum_{i=1}^{3} l_i^2}, \tag{4}$$

where A is the signed area of the triangle and l_i is the length of its incident edges. The metric can be replaced by any other. For further informations on metrics see [30, 31, 40]. Some examples are given in Sec. 5.3.

In Sec. 4.1 our method is described. Sec. 4.2 explains a way of combining different quality metrics, here, performing density adaption while preserving mesh quality.

4.1 Algorithm

The method starts assigning each vertex to one of the sets S_m and S_f. Vertices that should be optimized are in set S_m, fixed vertices in S_f. In the given examples boundary vertices are fixed, see Algorithm 1. For each vertex the *argmax-min* problem of eq. (3) is solved on a discretized domain using a greedy approach. Vertices that are not adjacent can be smoothed in parallel. Thus, graph coloring is applied once as preprocessing step to create subsets of vertices for parallel optimization [25].

Solving the discretized *argmax-min* problem is done iteratively with a uniform grid, see Algorithm 2. The grid is positioned around the vertex that should be optimized. The grid size is defined by the axis aligned bounding box for the one-ring neighborhood and a scaling factor ω. In our case, using half the size of the bounding box, $\omega = 0.5$, as an initial scaling factor led to convenient results.

Each grid-point is considered as candidate position where eq. (1) is evaluated. The vertex is repositioned at the best candidate if this increases its quality, Fig. 2. After each iteration step the scaling factor ω is reduced such that the new grid size is twice the old grid spacing,

$$\omega \leftarrow \omega \cdot 2/(n-1), \tag{5}$$

where n is the number of grid points in one dimension, Fig. 3. Furthermore, the grid is repositioned around the current best vertex position. The process of candidate evaluation and mesh downscaling is repeated iteratively until the desired level of precision is reached.

We know that the vertex quality function stated in eq. (1) is decaying from its unique maximum. Thus, we state that our exhaustive greedy approach provides the optimal solution for the vertex within the one-ring in a discrete sense. Accuracy of the result can be increased up to floating point precision by adapting the number of iterations in the greedy algorithm. We found 3 iterations combined with $n = 8$ grid points in each dimension to be sufficient.

Working on a uniform grid enables efficient parallelization of the greedy algorithm. Using one block of 32 threads on a grid with 64 points leads to an optimal utilization of the GPU if enough vertices are processed in parallel.

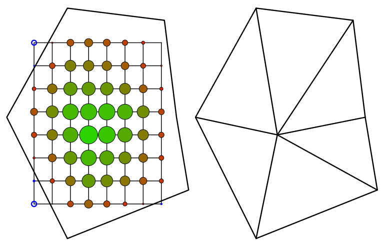


Fig. 2: *Left*: Uniform grid with quality metric evaluation for each candidate. A large green point represents good, a small red one bad quality. Positioning the vertex at one of the blue circles would result in triangle flipping. *Right*: the resulting fan after positioning the vertex at the best candidate.

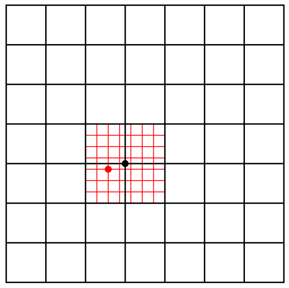


Fig. 3: Two iterations of the greedy method for finding the optimal vertex position. The dots represent the best candidate of the corresponding grid. The grid of the next iteration (red) is scaled down and placed at the best candidate of the previous grid (black).

Algorithm 1 Discrete mesh optimization

```
1: function SmoothMesh($M$,$n_{\text{iterations}}$)
2:     Vertex-set $S_m, S_f$
3:     for all $v \in M$ do
4:         if $v$ is boundary vertex then
5:             $S_f \leftarrow S_f \cup v$
6:         else
7:             $S_m \leftarrow S_m \cup v$
8:     Color-set $S_c$ = ColorMesh($S_m$, $M$)                  ▷ Each color is a subset of $S_m$
9:     for $i = 0$ to $n_{\text{iterations}}$ do                 ▷ Perform smoothing $n_{\text{iterations}}$ times
10:        for all $C \in S_c$ do                                 ▷ iterate over colors
11:            for all $v \in C$ do                               ▷ Iterate over vertices in $C$
12:                OptimizeVertexPosition($v$, $N_e(v)$, 8, 3)    ▷ see Algorithm 2
    return
```

4.2 Mesh Density Adaptive Optimization

We introduce a way of combining several quality metrics. As an example, we fuse the mean ratio metric $q^{(e)}_{\text{m}_{\text{tri}}}$ of eq. (4), and the density metric $q^{(v)}_{\text{d}}$, defined as the inverse of the squared distance between the position x_{k} of a vertex v_{k} and its optimal position x^*_{k} (in terms of density) relative to its one-ring neighborhood,

$$q^{(v)}_{\text{d,k}}(x_{\text{k}}) = \frac{1}{\|x_{\text{k}} - x^*_{\text{k}}\|^2}. \tag{6}$$

Algorithm 2 Discrete optimization of vertex position

```
1: function OptimizeVertexPosition($v$, $N_e(v)$, $n$, $n_{\text{greedy}}$)
2:     $\omega \leftarrow 0.5$                                                ▷ Scaling-factor for grid
3:     for $counter = 0$ to $n_{\text{greedy}}$ do                            ▷ Do $n_{\text{greedy}}$ iterations of the algorithm
4:         $(x_{\min}, y_{\min}, dx, dy)$ = GetGrid($v$, $N_e(v)$, $n$, $\omega$)   ▷ Get geometric grid specifications
5:         Create $quality\_grid$[n][n]
6:         for $i, j = 0$ to $n$ do
7:             $v' \leftarrow (x_{\min} + i \cdot dx, y_{\min} + j \cdot dy)$   ▷ Position $v'$ at grid-point (i,j)
8:             $quality\_grid$[i][j] $\leftarrow$ VertexQuality($v'$,$N_e(v)$)
9:         $q_{\max} \leftarrow -\infty,\ i_{\max} \leftarrow 0,\ j_{\max} \leftarrow 0$
10:        for $i, j = 0$ to $n$ do
11:            if $quality\_grid$[i][j] $> q_{\max}$ then                     ▷ Get argmax of $quality\_grid$
12:                $q_{\max} \leftarrow quality\_grid$[i][j]
13:                $i_{\max} \leftarrow i,\ j_{\max} \leftarrow j$
14:        if $q_{\max} >$ VertexQuality($v$,$N_e(v)$) then                   ▷ Check if initial position is better
15:            $v \leftarrow (x_{\min} + i_{\max} \cdot dx, y_{\min} + j_{\max} \cdot dy)$   ▷ Move vertex to new position
16:        $\omega \leftarrow \omega \cdot 2/(n-1)$                             ▷ Reduce scaling factor
       return
```

This metric tends to infinity the closer we get to the optimum and towards zero the further we move away. Computing the optimal position x_k^* is based on the spring-model for parametrization [16],

$$x_k^* = \frac{\sum_{j \in N(k)} D_j x_j}{\sum_{j \in N(k)} D_j}, \quad D_j = \frac{1}{h(x_j)}, \tag{7}$$

with $N(k)$ the indices of the one-ring neighborhood of v_k and $h(v_j)$ the size function at position x_j [34].

Assume we want to adapt mesh density while ensuring a minimal element quality $\hat{q}^{(e)}_{m_{tri}}$. We combine the two metrics to a new one as follows

$$q^{(v)}_{md,k}(x) = \begin{cases} q^{(v)}_{m_{tri}}(x) & \text{if} \quad q^{(v)}_{m_{tri}}(x) \leq \hat{q}^{(e)}_{m_{tri}} \\ \hat{q}^{(e)}_{m_{tri}} + q^{(v)}_{d,k}(x) & \text{otherwise} \end{cases}, \tag{8}$$

and solve the same *argmax-min* problem as before. An application of this metric is demonstrated in Sec. 5.3.

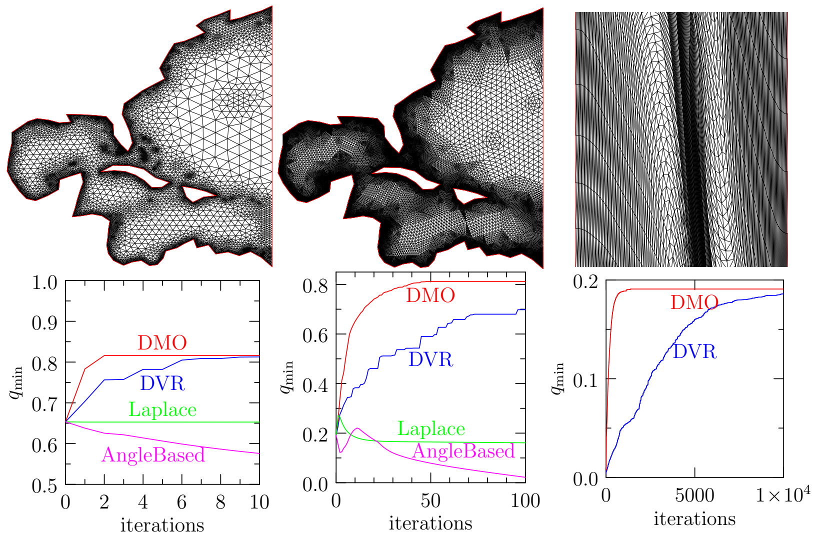
5 Results

DMO is compared to other methods in terms of convergence in Sec. 5.1 and performance in Sec. 5.2. Here, performance denotes the throughput of smoothed vertices per second. Examples for several mesh types and quality metrics are given in Sec. 5.3.

5.1 Convergence

We compare DMO with Laplace smoothing, angle-based smoothing, and DVR by analyzing convergence of minimal element quality. One iteration denotes that smoothing was applied to all vertices of S_m once. DMO and DVR optimize both for the mean ratio metric $q^{(e)}_{\mathrm{m_{tri}}}$ of eq. (4). Fig. 4a shows convergence on the triangle mesh *"east"*. The input mesh has already satisfactory quality. While DMO and DVR increase the minimal element quality, Laplace smoothing keeps it unchanged. Angle-based smoothing even decreases the quality. The bad behavior of angle-based smoothing is caused by the strongly varying element size within the mesh. DMO and DVR converge towards the same result. Nevertheless, DMO reached its optimum already after two iterations whereas DVR required 10 iterations.

A finer and block structured version of mesh *"east"* is shown in Fig. 4b. Here, Laplace smoothing is unstable. This follows from the topology which is not perfectly adapted to the geometry anymore. Angle-based smoothing decreases quality. DVR and DMO improve the mesh quality. However, DVR takes around 900 iterations to reach the same minimal quality that is achieved by DMO in 60 iterations. Thus, DMO converges 15 times faster than DVR.



(a) *"east"*, $n_m = 8249$ (b) *"east"*, $n_m = 37907$ (c) *"shashkov"*, $n_m = 3969$

Fig. 4: Improvement of minimal quality measured with the mean ratio metric.

The last example considers an anisotropic mesh for which it is not reasonable to optimize its elements for roundness. Nevertheless, it can be used to test convergence of smoothing methods in extreme scenarios. Depending on a search direction, especially on a random one, leads to very slow convergence. Fig. 4c shows an excerpt of the mesh ”shashkov” which was taken from the example files of the Mesquite Toolkit [10]. DMO converged within 1400 iterations, whereas DVR was not converged after the stated 10 000 iterations.

5.2 Performance

We compare the performance of DMO with Laplace smoothing and DVR on triangle grids. Performance is measured as smoothed vertices per second. Preprocessing steps like mesh loading or coloring were excluded from measurements. Each method performed 1000 iterations on each mesh, ignoring convergence as we are only interested in runtime per iteration. The performance of DMO was achieved on a Nvidia GTX 1070. The other methods and also a CPU version of DMO were profiled on an Intel i7-6700K CPU with 4.00 GHz and 4 cores. Note that the comparison was done using our own implementation of Laplace smoothing and DVR. DMO was performed using 8 grid points in each dimension and 3 iterations of the greedy algorithm. Several tests were run on different meshes, Fig. 5. DMO and Laplace smoothing reach around $2 \cdot 10^7$, DVR and DMO on CPU 10^5 vertices per second. The performance of DMO increases for larger meshes because the GPU is not fully utilized in case of small meshes.

On a mesh with 1 333 540 vertices in S_m and an initial minimal mean ratio quality of 0.033 DMO required 4 iterations and an execution time of 400 ms on GPU, including about 200 ms for graph coloring and copying data to the GPU, to converge towards a quality of 0.34. On CPU DMO ran for 31s to execute the same number of iterations and reach the same quality. DVR required 43 iterations which took about 435 s to converge towards the same quality. Thus, in this example DMO is 14 times faster on CPU than DVR. On GPU it runs more than 1000 times faster.

Another mesh is given with 8 429 vertices in S_m and an initial minimal mean ratio quality of 0.65. DMO converged within 4 iterations. On GPU it terminated in 4 ms, on CPU in 100 ms. DVR converged within 10 iterations and ran for 780 ms. All methods reached a minimal quality of 0.82. Here, DMO is 7.8 times faster on CPU than DVR and on GPU it is 195 times faster.

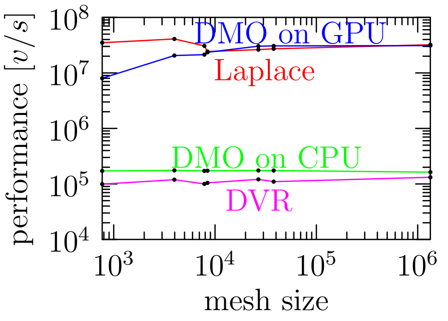


Fig. 5 Performance of smoothing methods for different mesh sizes. The performance is measured in smoothed vertices per second.

5.3 Different Quality Metrics

All results stated up to now were computed using the mean ratio metric for triangles, eq. (4). In this subsection we apply different metrics on various mesh types showing the flexibility of our method.

First, we smooth the block structured triangular mesh *”east”* (Fig. 6a) with respect to rectangularity [23],

$$q_{\text{rect}}^{(e)} = \max_{\text{i = 1,2,3}} q^{\text{i}}, \tag{9}$$

$$q^{\text{i}} = (1 - \frac{|\frac{\pi}{2} - \theta_{\text{i}}|}{\frac{\pi}{2}}) \cdot (1 - \frac{|e_{\text{ij}} - e_{\text{ik}}|}{\max(e_{\text{ij}}, e_{\text{ik}})}). \tag{10}$$

Here, q^{i} denotes the local quality of one vertex within the triangle, θ_{i} is the interior angle at this vertex, e_{ij} and e_{ik} are the incident edges. The right-angled quality criterion in [23] contains another factor that aligns the triangles according to a cross field. We skipped this term as our purpose is just to show the flexibility of DMO. Fig. 6c clearly states a rapid convergence of DMO to an optimal mesh. The resulting mesh is shown in Fig. 6b.

We also tested DMO with several other triangle shape quality metrics such as

- Max angle [3]: $\max_{\text{i = 1,2,3}} \theta_{\text{i}} \cdot \frac{3}{\pi}$
- Min angle [18]: $1.5(1 - \frac{\min_{\text{i = 1,2,3}} \theta_{\text{i}}}{\pi})$
- Radius ratio [40]: $2\frac{r}{R}$, where r is the incricle and R the circumcircle.

Fig. 6f states the convergence plots for these metrics, the resulting meshes for *radius ratio* and *max angle* are given in Fig. 6d and 6e.

In the following example we apply the mean ratio metric for quad meshes,

$$q_{\text{m}_{\text{quad}}}^{(e)} = 2\frac{A}{\sum_{i=1}^{4} l_i^2}. \tag{11}$$

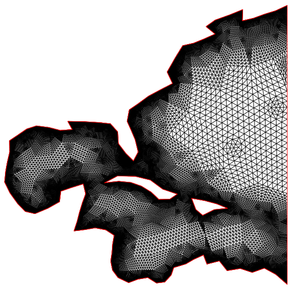
The mesh in Fig. 7a is a block structured quad version of *”east”*. The plot in Fig. 7c shows fast convergence and significant improvement in terms of the given quality metric. Fig. 7b shows the resulting mesh.

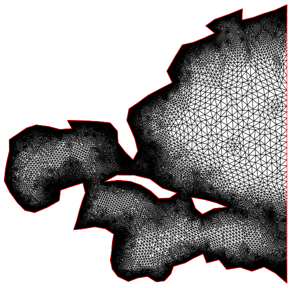
Adding a third dimension to DMO enables optimization for volume meshes. Fig. 8 shows a tetrahedral mesh where the mean ratio metric for tetrahedrons [17, 31, 37] was applied. The mesh quality was improved significantly within 10 iterations.

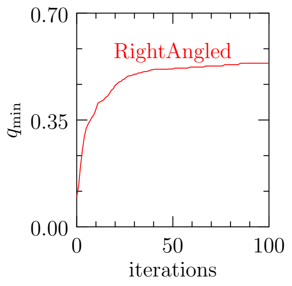
Finally, we want to show that our method is capable of adapting density as described in Section 4.2. We use the distance to some point (x_0, y_0) as size function,

$$h_1(x,y) = \sqrt{(x_0 - x)^2 + (y_0 - y)^2} \tag{12}$$

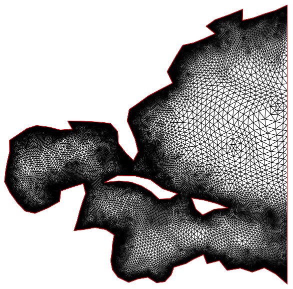
$$h_2(x,y) = \frac{1}{h_1(x,y)}. \tag{13}$$

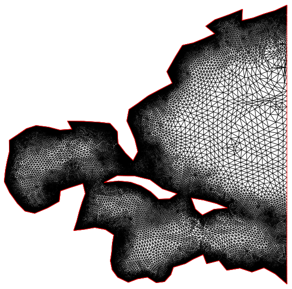

(a) Initial block structured *"east"*.

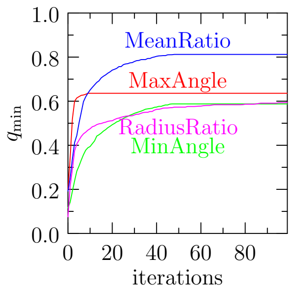

(b) Right-angled quality.



(c) Convergence for right-angled quality.

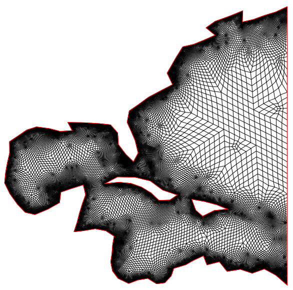

(d) Radius ratio metric.

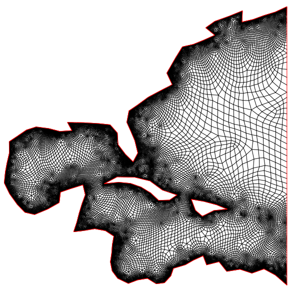

(e) Max angle metric.

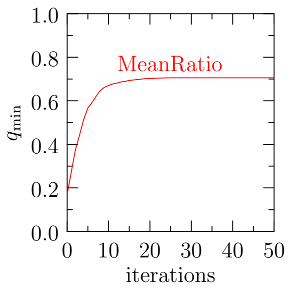


(f) Convergence for several shape metrics.

Fig. 6: Block structured triangle mesh *"east"* optimized with different quality metrics.


(a) "east" initial


(b) "east" smoothed



(c) *"east"* as a block structured quad mesh.

Fig. 7: Quad mesh *"east"* optimized with mean ratio metric.

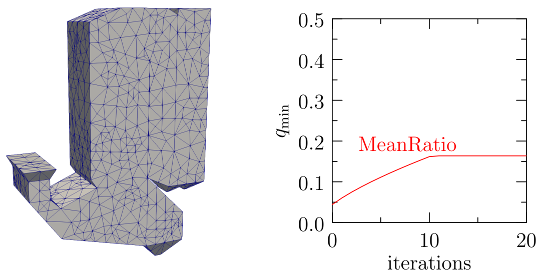
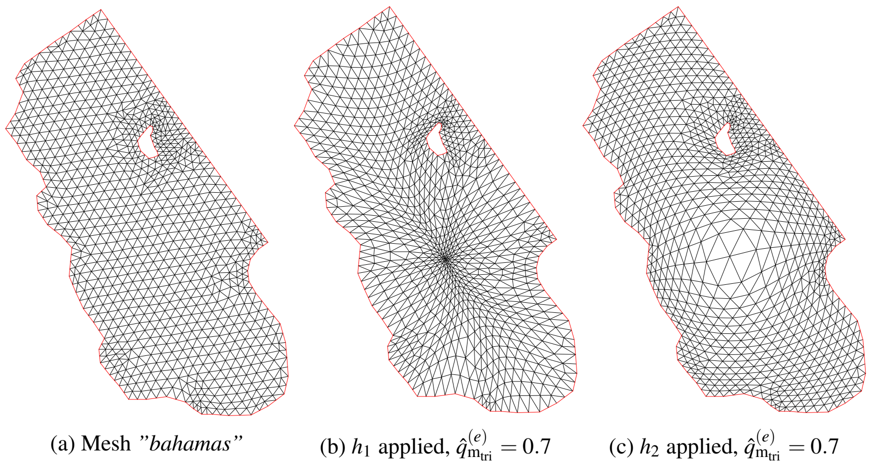


Fig. 8: Tetrahedral volume mesh *”tire”* from [39].

Fig. 9b shows the application of h_1 and Fig. 9c of h_2 on the triangle mesh *”bahamas”*, Fig. 9a. In both cases the minimal mean ratio $\hat{q}^{(e)}_{\mathrm{m_{tri}}} = 0.7$ was given.



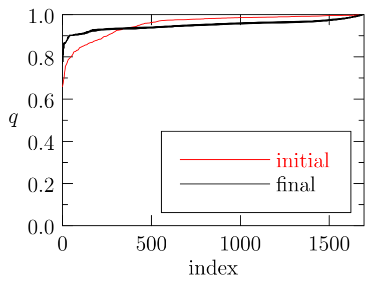
(a) Mesh *”bahamas”* (b) h_1 applied, $\hat{q}^{(e)}_{\mathrm{m_{tri}}} = 0.7$ (c) h_2 applied, $\hat{q}^{(e)}_{\mathrm{m_{tri}}} = 0.7$

Fig. 9: Density adaption on mesh *”bahamas”*.

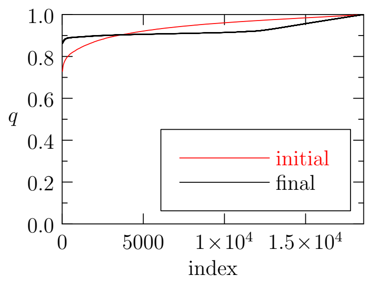
5.4 Robustness Against Smoothing Order

We give examples which indicate robustness of DMO against the order in which vertices are processed. We change the smoothing order by shuffling the vertex indices before applying DMO. We measure the quality of the mesh by lexicographically ordering the elements by their quality. Fig. 10 shows the results of two meshes which already have an initial good quality. The smoothing was done 50 times with different orders and for each time a black line was added to the plot. These lines mostly overlap which corresponds to equal overall mesh quality.

Fig. 11 shows the same plot for a Delaunay triangulation of 100 randomly distributed points. Here, a larger deviation in quality can be observed. This is caused by the extremely low quality of the input mesh. Due to the large movements during smoothing the order in which vertices are processed has a strong influence on the output mesh. Fig. 12 gives two examples. Both meshes have approximately the same minimal element quality but look completely different. This behavior could only be observed in such extreme cases as a Delaunay triangulation of randomly distributed points. If an input mesh has a reasonable quality, the result of our optimization algorithm is expected to be independent from the order in which the vertices are processed.

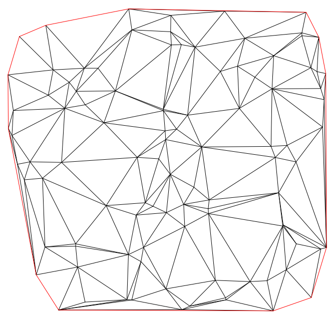


(a) *"bahamas"*



(b) *"east"*

Fig. 10: Quality with different smoothing orders on good initial meshes.



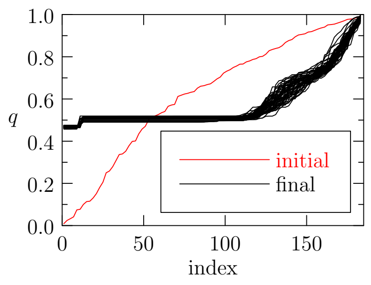


Fig. 11: Quality with different smoothing orders on a Delaunay triangulation (*left*).

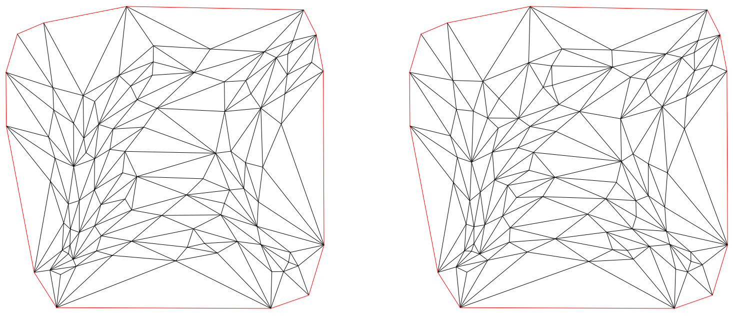


Fig. 12: Results for different smoothing orders.

6 Conclusion

We presented a greedy approach to mesh optimization with fast convergence and adaptability to any kind of mesh or quality metric. This method can take full advantage of the parallelism of a consumer GPU. The minimal quality of a mesh is either improved or at least not decreased within each iteration. The interchangeability of the quality metric enables various cases of practical relevance, e.g. adaptive mesh smoothing.

In future work we will apply DMO to block structured ocean meshes. Creating a block structured mesh from an unstructured mesh results in a loss of minor features along the boundary. We aim to reposition the boundary of a block structured mesh to restore minor features while preserving high mesh quality.

References

1. Vadym Aizinger and Clint Dawson. A discontinuous galerkin method for two-dimensional flow and transport in shallow water. *Advances in Water Resources*, 25(1):67–84, 2002.
2. Nina Amenta, Marshall Bern, and David Eppstein. Optimal point placement for mesh smoothing. *Journal of Algorithms*, 30(2):302–322, 1999.
3. Ivo Babuška and A Kadir Aziz. On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, 13(2):214–226, 1976.
4. Timothy J Baker. Mesh movement and metamorphosis. *Engineering with Computers*, 18(3):188–198, 2002.
5. Randolph E Bank and R Kent Smith. Mesh smoothing using a posteriori error estimates. *SIAM Journal on Numerical Analysis*, 34(3):979–997, 1997.
6. RE Bank. A software package for solving elliptic partial differential equations–users guide 7.0. *Frontiers in Applied Mathematics*, 15, 1998.
7. Ted D Blacker and Michael B Stephenson. Paving: A new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 32(4):811–847, 1991.
8. Ted D Blacker, Michael B Stephenson, and Scott Canann. Analysis automation with paving: a new quadrilateral meshing technique. *Advances in engineering software and workstations*, 13(5-6):332–337, 1991.

9. Frederic J Blom. Considerations on the spring analogy. *International journal for numerical methods in fluids*, 32(6):647–668, 2000.
10. Michael L Brewer, Lori Freitag Diachin, Patrick M Knupp, Thomas Leurent, and Darryl J Melander. The mesquite mesh quality improvement toolkit. In *IMR*, 2003.
11. Scott A Canann, Yong-Cheng Liu, and Anton V Mobley. Automatic 3d surface meshing to address today's industrial needs. *Finite Elements in Analysis and Design*, 25(1-2):185–198, 1997.
12. Scott A Canann, Joseph R Tristano, Matthew L Staten, et al. An approach to combined laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes. In *IMR*, pages 479–494. Citeseer, 1998.
13. Valmor F De Almeida. Domain deformation mapping: application to variational mesh generation. *SIAM Journal on Scientific Computing*, 20(4):1252–1275, 1999.
14. Ch Farhat, C Degand, B Koobus, and M Lesoinne. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Computer methods in applied mechanics and engineering*, 163(1-4):231–245, 1998.
15. David A Field. Laplacian smoothing and delaunay triangulations. *International Journal for Numerical Methods in Biomedical Engineering*, 4(6):709–712, 1988.
16. Michael S Floater. Parametrization and smooth approximation of surface triangulations. *Computer aided geometric design*, 14(3):231–250, 1997.
17. Lori Freitag, Mark Jones, and Paul Plassmann. A parallel algorithm for mesh smoothing. *SIAM Journal on Scientific Computing*, 20(6):2023–2040, 1999.
18. Lori Freitag, P Plassmann, and M Jones. An efficient parallel algorithm for mesh smoothing. Technical report, Argonne National Lab., IL (United States), 1995.
19. Lori A Freitag. On combining laplacian and optimization-based mesh smoothing techniques. *ASME applied mechanics division-publications-amd*, 220:37–44, 1997.
20. Lori A Freitag and Patrick M Knupp. Tetrahedral mesh improvement via optimization of the element condition number. *International Journal for Numerical Methods in Engineering*, 53(6):1377–1391, 2002.
21. Lori A Freitag, Paul Plassmann, et al. Local optimization-based simplicial mesh untangling and improvement. *International Journal for Numerical Methods in Engineering*, 49(1):109–125, 2000.
22. Paul-Louis George and Houman Borouchaki. Delaunay triangulation and meshing. 1998.
23. Christos Georgiadis, Pierre-Alexandre Beaufort, Jonathan Lambrechts, and Jean-François Remacle. High quality mesh generation using cross and asterisk fields: Application on coastal domains. *arXiv preprint arXiv:1706.02236*, 2017.
24. Leonard R Herrmann. Laplacian-isoparametric grid generation scheme. *Journal of the Engineering Mechanics Division*, 102(5):749–907, 1976.
25. Tommy R Jensen and Bjarne Toft. *Graph coloring problems*, volume 39. John Wiley & Sons, 2011.
26. RE Jones. Qmesh: A self-organizing mesh generation program. Technical report, Sandia Labs., Albuquerque, N. Mex.(USA), 1974.
27. Jibum Kim. A multiobjective mesh optimization algorithm for improving the solution accuracy of pde computations. *International Journal of Computational Methods*, 13(01):1650002, 2016.
28. Patrick Knupp. Updating meshes on deforming domains: An application of the target-matrix paradigm. *International Journal for Numerical Methods in Biomedical Engineering*, 24(6):467–476, 2008.
29. Patrick M Knupp. Winslow smoothing on two-dimensional unstructured meshes. *Engineering with Computers*, 15(3):263–268, 1999.
30. Patrick M Knupp. Achieving finite element mesh quality via optimization of the jacobian matrix norm and associated quantities. part iia framework for volume mesh optimization and the condition number of the jacobian matrix. *International Journal for numerical methods in engineering*, 48(8):1165–1185, 2000.
31. Patrick M Knupp. Algebraic mesh quality metrics. *SIAM journal on scientific computing*, 23(1):193–218, 2001.

32. Michal Křížek. On the maximum angle condition for linear tetrahedral elements. *SIAM Journal on Numerical Analysis*, 29(2):513–520, 1992.
33. Jeonghyung Park and Suzanne M Shontz. Two derivative-free optimization algorithms for mesh quality improvement. *Procedia Computer Science*, 1(1):387–396, 2010.
34. Per-Olof Persson. Mesh size functions for implicit geometries and pde-based gradient limiting. *Engineering with Computers*, 22(2):95–109, 2006.
35. Per-Olof Persson and Gilbert Strang. A simple mesh generator in matlab. *SIAM review*, 46(2):329–345, 2004.
36. Ramsharan Rangarajan. On the resolution of certain discrete univariate max–min problems. *Computational Optimization and Applications*, 68(1):163–192, 2017.
37. Ramsharan Rangarajan and Adrian J Lew. Provably robust directional vertex relaxation for geometric mesh optimization. *SIAM Journal on Scientific Computing*, 39(6):A2438–A2471, 2017.
38. Martin Rumpf. A variational approach to optimal meshes. *Numerische Mathematik*, 72(4):523–540, 1996.
39. Shewchuk. *Stellar: A tetrahedral mesh improvement program*, 05-23-2018. URL: `https://people.eecs.berkeley.edu/~jrs/stellar/input_meshes.zip`.
40. Jonathan Shewchuk. What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures (preprint). *University of California at Berkeley*, 73:137, 2002.
41. Suzanne M Shontz and Stephen A Vavasis. A mesh warping algorithm based on weighted laplacian smoothing. In *IMR*, pages 147–158, 2003.
42. Hongtao Xu and Timothy S Newman. 2d fe quad mesh smoothing via angle-based optimization. In *International Conference on Computational Science*, pages 9–16. Springer, 2005.
43. Kaoji Xu, Xifeng Gao, and Guoning Chen. Hexahedral mesh quality improvement via edge-angle optimization. *Computers & Graphics*, 70:17–27, 2018.
44. Pablo D Zavattieri, Enzo A Dari, and GUSTAVO C Buscaglia. Optimization strategies in unstructured mesh generation. *International Journal for Numerical Methods in Engineering*, 39(12):2055–2071, 1996.
45. Tian Zhou and Kenji Shimada. An angle-based approach to two-dimensional mesh smoothing. In *IMR*, pages 373–384, 2000.