Progressive Hulls: Application on Biomedical Data

David Cholt*
Supervised by: Josef Kohout†

Department of Computer Science and Engineering
University of West Bohemia
Pilsen / Czech Republic

Abstract

A coarse outer hull of a mesh is a good tool used in computer graphics to reduce algorithm complexity, especially, in applications such as collision detection or ray-tracing. It is often required that the hull has some very specific parameters concerning its shape and quality since these influence general flexibility and numerical stability of the algorithms. This paper overviews existing problem approaches, their downsides for coarse outer hull creation, and describes *Progressive Hull* algorithm, which produces coarse hulls that maintain the general shape of the original triangulated mesh while containing the original mesh inside its interior. However, when this algorithm is used on a biomedical mesh extracted from volumetric data, one can observe frequent artifacts in the produced hull caused by local imperfections in the meshes. In this paper, we, therefore, present a few modifications to the original *Progressive Hull* algorithm that result not only in a suppression of hull artifacts and a better overall hull quality but also in a shorter execution time.

Keywords: Progressive Hull, Coarse Hull, Outer Hull, Mesh Decimation, Progressive Hull Application

1 Introduction

A coarse outer hull of a mesh is any hull that encapsulates the mesh completely and has a lower number of primitives (vertices, triangles) and, if possible, preserves the shape of the mesh. It can be used in many applications, for example in ray-tracing where one can detect the possibility of a ray intersecting the mesh by finding the intersection with the hull first. Given that the hull contains less polygons than the original mesh, the hull intersection test is faster. Furthermore it is more precise than tests using bounding box or convex hull, since the shape of the coarse outer hull is more similar to the shape of the mesh.

In our case we exploited the properties of the hull in in our project aiming at a simulation of human musculoskeletal system. Every muscle in this system is represented by a triangulated surface mesh which is wrapped around bones and gets deformed as these bones move. The deformation method, which is based on gradient domain deformation technique and described in [4], requires a very specific coarse outer hull of the muscle mesh as its input in order to speed up the deformation process. The deformation is performed on the hull and projected on the muscle mesh using barycentric coordinates that were previously constructed. This allows the method to be more numerically stable and faster. The coarse hull of the mesh therefore must meet following criteria:

- Low primitive count compared to original mesh - the hull has to be simple enough, so that later complex computations can be performed on a limited number of primitives. Less than $\frac{1}{10}$ of the original primitive count is desired
- Outer hull - all primitives of the hull must be outside of the mesh, leaving some spacing between the mesh and the hull. This is due to barycentric coordinates requirements
- Shape preservation - the hull has to "trace" the shape of the original mesh. In other words, the spacing between the original mesh and the hull has to be consistent
- Non-self-intersecting hull - edges and triangles of the hull may not intersect the hull as this would cause instability in deformation computation
- The hull has to be manifold and should be smooth (depending on the muscle data), otherwise this would cause deformation instability as well

Outline of this paper follows. Section 2 describes examples of methods used to obtain coarse outer hulls and their disadvantages, Section 3 describes *Progressive Hull* algorithm, which should produce hulls that meet the criteria above, Section 4 introduces our modifications of the algorithm for better results on our biomedical data, followed by Sections 5 presenting experimental results, a hull quality comparison and execution time measurements. Our paper is concluded in Section 6.

*cholt@students.zcu.cz
†besoft@kiv.zcu.cz

2 State of the art

In this section we briefly describe the methods, that can be used to obtain a coarse outer hull of a triangulated mesh. Since we use the coarse hull in the deformation method described in [4], here we discuss some possible approaches to obtain a specific hull that meets all the criteria described in Section 1. This paper does not provide an overview of methods used in collision detection, raytracing, silhouette clipping or another methods that use the coarse outer hull as we did not use it for those purposes.

Bounding box, as an example of a very simple coarse hull, does not preserve the shape of the original mesh. Convex hull meets more of the criteria specified, but the spacing between the mesh and the hull varies significantly, especially if the mesh is rugged.

Better approach to the problem is to create an *alpha shape* [2], which is an object created from a finite set of vertices, in our case the mesh vertices. For the sake of simplicity, we will describe the construction and problems of an *alpha shape* in two dimensions. The method has one tuning parameter α that defines a radius of an abstract disc. The method finds such discs that have the property that two of the vertices lie on their boundary and they do not contain any other vertices. Vertices of the *alpha shape* are generated at the intersection of two neighboring discs. The situation is similar for three-dimensional space, only one has to use spheres instead of discs and search for intersections of three neighboring spheres. We can additionally implement a restriction that the intersection must lie outside of the original mesh in order to achieve outer hull. This method for finding an "*alpha hull*" from a set of points allows one to create an object, that is not necessarily convex and therefore, to certain extent, resembles the shape of the original mesh. However, this approach has four main problems:

1. If the α parameter is too large, the resulting hull suffers from the same problem as convex hull, i.e. the spacing between the alpha hull and the mesh varies significantly (for $\alpha \to \infty$ the alpha hull is equal to convex hull; see Fig. 1a)

2. If the α parameter is too small, the hull would intersect the original mesh (see Fig. 1b). Very small α parameter also causes the hull to be divided into number of components that can not form the hull of the original mesh by definition.

3. Some (arguably important) details in the original mesh may be lost in the process (Fig. 1a)

4. Even if we would be able find an optimal α parameter that ensures the *alpha hull* is a coarse outer hull of the object, the primitive count of the hull would be too high, approximately the same as the primitive count of the original mesh. In general, the number of primitives in resulting hull cannot be controlled well enough in *alpha shapes*

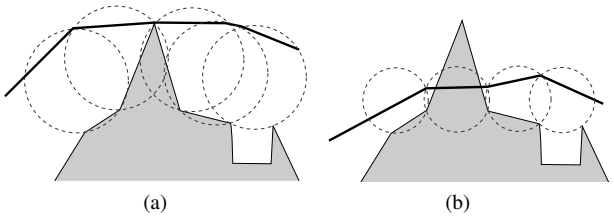


Figure 1: (a) *alpha shape* with too large α parameter. (b) *alpha shape* with too small α parameter.

Another possible approach is to decimate the mesh and therefore obtain a coarse mesh directly from the original mesh. Decimated mesh maintains the shape of the mesh by definition and has desired lower primitive count. One can assume that enlarging the mesh by moving its vertices outwards in the direction of their surface normal would create a coarse outer hull.

The problem is how much we need to enlarge the decimated mesh. If we enlarge the mesh too much, self-intersections may occur (see Fig. 2 for example). However, if we do not enlarge it enough, the decimated mesh would intersect the original mesh, as many decimation algorithms are based on volume preservation.

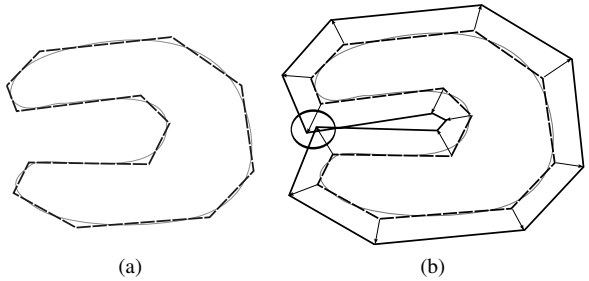


Figure 2: (a) Decimated mesh. (b) Enlarged decimated mesh with self-intersection introduced.

Nevertheless, this approach is simple to implement and has a sufficient shape and primitive count control, though it may give inconsistent results. We need an algorithm that creates an outer hull of the mesh by decimating and enlarging it at the same time, with a better control of how the decimation is performed.

3 Progressive hull

Progressive Hull [7] is a generalized mesh simplification process that meets all the criteria described in Section 1. The method is based on decimation of the mesh by a sequence of edge collapses, that ensures that progressively created hull contains the whole original mesh in its interior volume.

Figure 3 shows how the edge **e**, surrounded by faces $\{f_0, f_1, ..., f_m, f_{d1}, f_{m+1}, ..., f_n, f_{d2}\}$ and defined by vertices V_{e1} and V_{e2} collapses. Vertices V_{e1} and V_{e2} are joined

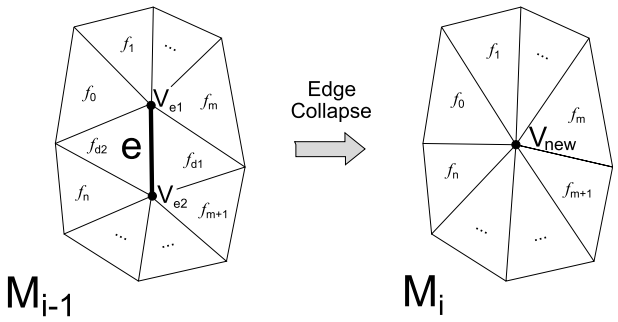


Figure 3: Edge collapse

by this operation to form a new vertex V_{new} and affected primitives are adjusted accordingly. Note that faces f_{d1} and f_{d2} are removed from the mesh and therefore every collapse reduces the number of faces in the mesh by two.

In order for the sequence of such collapses to result in a progressive hull, we need to calculate a specific position for the vertex V_{new}. Paper [7] shows that in order for the mesh M_i (after one edge collapse) in every iteration i to be an outer hull of the mesh M_{i-1} (before the collapse), the volume of the mesh M_i must be greater or equal to the volume of the mesh M_{i-1}. This can be achieved by placing the vertex V_{new} inside the intersection of half spaces above faces $\{f_0, f_1, ..., f_m, f_{d1}, f_{m+1}, ..., f_n, f_{d2}\}$ (see Fig. 4 for simplified two-dimensional case).

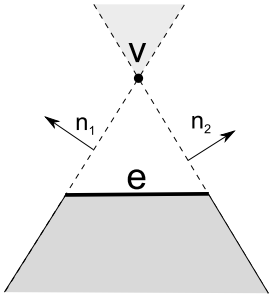


Figure 4: Position of vertex **V** constrained to lie in intersection of half spaces defined by planes with normals n_1 and n_2

The volumes of the meshes M_{i-1} and M_i are computed as a sum of the tetrahedral volumes defined by each mesh triangle's vertices and the origin point. Since we need the hull to be similar to the original mesh as much as possible, we place the vertex V_{new} in a way that it causes the smallest possible volume gain. That is a linear programming problem with an objective of mesh volume gain minimization and constrains defined by equations of half spaces above faces $\{f_0, f_1, ..., f_m, f_{d1}, f_{m+1}, ..., f_n, f_{d2}\}$.

Every mesh edge enters a priority queue with a priority based on the volume gain introduced by its collapse. The lower is the change, the higher is the priority. The algorithm follows:

1. For every edge in the mesh, compute the volume change that would be introduced by collapsing the edge. That requires solving linear programming problem. Store the solution for later use.
2. Insert the edge to a priority queue, low mesh volume gain represents a high priority in the queue
3. While the queue is not empty and the target primitive count is not reached:
   (a) Remove the edge from the priority queue and collapse it
   (b) Recalculate priority of every edge that was affected by this collapse and update their position in the priority queue by solving the linear programming problem again

One can see that this algorithm can be quite slow. Many of the linear programming problem solutions are redundant, as they are invalidated by a later nearby edge collapse. Additionally, a time-consuming solution of the linear programming problem is required for every edge priority update.

Platis and Theoharis [6] suggested using a faster approach to priority computation. Instead of using vertex V_{new} computed by solving the linear programming problem to determine the volume of the mesh M_i after the edge collapses, they proposed using an arithmetic average of the vertices in the one ring area surrounding the collapsing edge, V_{avg}.

The volume computed using this average scales similarly to the volume originally computed using vertex V_{new}, i.e. in relatively flat area adjacent to the edge **e**, the volume gain caused by imaginary collapse into the vertex V_{avg} is lower than in rugged area and therefore the induced priority is higher. This approach is considerably faster than solving the linear programming problem.

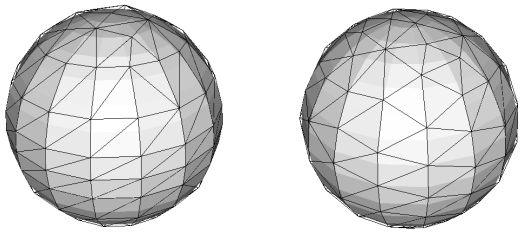
4 Modified progressive hull

Papers [7] and [6] do not mention any major problems regarding the quality of the resulting progressive hulls. However, our implementation of described methods performed irregular hull construction, showed significant numerical instability and produced low quality triangles in the hull. This was caused by imperfections in our meshes, presence of nearly or fully parallel triangles, local non-manifold areas in the mesh and other problems, which were probably not considered in [7] and [6]. Therefore, we introduced several modifications to the algorithm in order to resolve these issues.

4.1 Volume increase computation

The original paper [7] suggests the volume of object formed by faces $\{f_0, f_1, ..., f_m, f_{d1}, f_{m+1}, ..., f_n, f_{d2}\}$ before and after a single edge collapse (see Fig. 3) to be used

as collapse priority. Note that faces f_{d1} and f_{d2} become singular after the collapse and therefore their contribution to the volume after the collapse is zero.



(a) (b)

Figure 5: Sphere decimation from 480 to 250 triangles. (a) Irregular decimation (b) Regular decimation after priority computation fix

However, a priority computed from this volume gain makes the algorithm to process the shorter edges first, since their smaller adjacent triangles are more likely to introduce a smaller volume change. The areas formed by small triangles are progressively decimated into areas with presumably a bit larger, but still small enough triangles, forcing the algorithm to process them early again. As a result, the decimation runs irregularly around the mesh. Example can be seen in Figure 5a. One can see that the area around the sphere equator remains untouched as the caps of the sphere contain smaller areas with smaller local volumes.

To address this issue, we use global mesh volume gain to compute the priority. The volume gain caused by the edge collapse is computed relatively to the volume of the original mesh, not to the collapse area, i.e. as the sum of volume gains caused by all previous edge collapses and the volume gain caused by the edge collapse[1]. When the area with small triangles is progressively decimated, it grows outwards, the global volume gain increases and the computed priority decreases. Consequently the priority gets lower than the priority in unchanged areas with larger triangles, resulting in regularly performed decimation around the whole mesh (see Fig. 5b) and limiting the undesired variation of spacing between the hull and the original mesh.

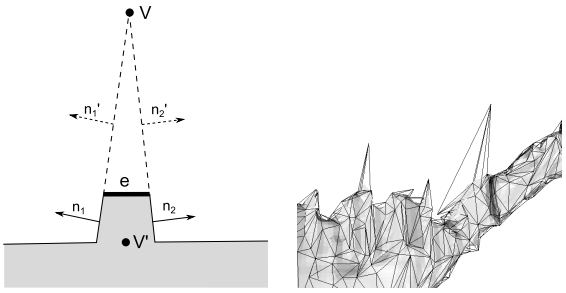
4.2 Algorithm stability

The mesh deformation method, which is described in [4], uses a hull of the mesh to speed up the computation. The vertices of the mesh are linked using barycentric coordinates to the near vertices of the hull, the deformation is then computed on the hull and the results are projected back to the mesh, deforming it accordingly. Because of this, the hull has to preserve the shape of the original mesh closely. Ideally, the spacing between the hull and the mesh is constant and the computed deformation is distributed evenly to the vertices of the original mesh. Note that one vertex of the hull represents a number of vertices of the original mesh and therefore any artifact present on the hull can cause the deformation projection to behave very unexpectedly.

When we used the original algorithm on the surface biomedical data extracted from volumetric data, we observed its high instability, artifacts resembling "spikes" were often present on the produced hull.

If there are two nearly parallel triangles in the edge collapse area (see Figure 6a), the optimal solution to the linear programming problem is a very distant vertex. Provided that the collapsing edge is short, the volume gain caused by the collapse is relatively small, the collapse priority is high and the edge is collapsed, resulting in a "spike" on the surface of the hull (see Figure 6b).

As the number of iterations rises, the artifacts are more common and accumulate. The resulting hull is then unusable for any later use. In order to prevent these artifacts from developing, we added a test that disallows the edge collapses causing them.



(a) (b)

Figure 6: Algorithm instability. (a) Cause of the problem (b) Spikes on the hull

Our test is inspired by Platis and Theoharis [6]. They proposed using Gueziec's [3] test of triangle deviation in progressive hulls to prevent creases on the hull. They test the angle between normals of triangles $\{f_0, f_1, ..., f_m, f_{m+1}, ..., f_n\}$ before and after collapsing the edge they are adjacent to (again, see Fig. 6a). However, as these normals are often unchanged, the spikes may still occur.

We use a slightly different method of checking the decimation quality. We compute angles α_n between normals of triangles $\{f_0, f_1, ..., f_m, f_{m+1}, ..., f_n\}$ adjacent to the vertex V_{new} in pairs. If any angle α_n is larger than the user-specified threshold α_t, we disallow the collapse, because a large angle between the normals of the triangles implies a small angle between the triangles. This small angle between two neighboring triangles indicates an undesired spike on the hull. Figure 7 shows, how are the angles the situation illustrated in Fig. 6a evaluated and since $\alpha_n > \alpha_t$, the collapse is therefore disallowed.

Since our biomedical data mainly consists of smooth surfaces (such as bones and muscles), any local decima-

[1] The collapse of the edge for which the priority is computed

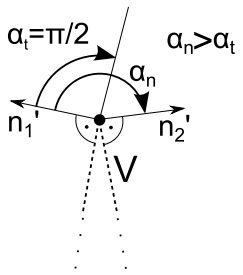


Figure 7: Triangle normal angle check

tion has to be relatively smooth, without any spikes. If some spikes are already present in the original mesh and would result in spike-like area on the hull, they are simply skipped. Experimental results show that these artifacts in the mesh are later "absorbed" by the hull as the surrounding areas are decimated. Therefore, these artifacts do not occur on the hull and the method becomes very stable in the required (and usually large) number of iterations.

4.3 Vertex and triangle quality

Many decimation methods (e.g. [5, 3, 6]) perform vertex valence test. Due to our better priority computation (see Section 4.1), our algorithm creates a regular hull of the mesh. Experimental results show that this test is not needed and the vertex valence is balanced automatically.

The shape of the triangles is also important. Generally, any computations are more stable if performed on compact triangles. Platis and Theoharis [6] suggest performing a triangle compactness test analogically to the test used by Gueziec [3]:

$$c = \frac{4\sqrt{3}a}{l_0^2 + l_1^2 + l_2^2}$$

where a is a positive area of the triangle and l_0, l_1 and l_2 are the lengths of its three sides. This number represents a quality of a triangle, smaller for "sliver" triangles that cause numerical instability in later use. Therefore, disallowing the edge collapse if the triangle compactness of any of the faces $\{f_0, f_1, ..., f_m, f_{m+1}, ..., f_n\}$ is lower than an user-specified threshold results in a hull, which contains only sufficient quality triangles.

We use a different simple test. We check the largest inner angle of each triangle (based on the largest dot product of two of the three triangle sides' vectors). If this angle is larger than the user-specified value, the triangle is considered to be narrow, and therefore undesired. If such an angle is present already in the original mesh, we check, if it gets smaller by the planned edge collapse. If the outcome of this test is negative, naturally, we disallow the collapse. This method is slightly more simple to implement and does not require additional triangle area computation, while the results are sufficiently good enough. On the other hand, if the mesh contains many narrow triangles, the algorithm disallows most of the edge collapses and desired number if primitives in the hull is not reached.

4.4 Self-intersection

Sander et al. [7] hypothesize that self-intersection prevention may be unnecessary. In our tests we did not observe any self-intersections caused by our algorithm.

Nevertheless, a self-intersection in the hull is caused by introducing a sharp crease in the mesh. This type of self-intersection may be introduced by possible local imperfections in our data. Figure 8 illustrates an example of this situation. One can see that by adjusting the α_t parameter (see Section 4.2), we can define what constitutes a sharp crease and therefore prevent the possible self-intersection.

This adjustment however globally affects the whole decimation process. Using a very small α_t parameter forces the algorithm to limit the degree of decimation in rough and creased mesh areas. This behavior can be desired (preserving subtle details in the mesh), but results in an uneven mesh decimation.

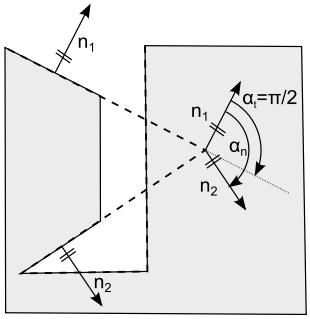


Figure 8: Possible introduction of hull self-intersection.

The self-intersection can also be caused by the very shape of the original mesh, where two mesh surfaces, even without any creases, are close together, resulting in a self-intersection in the hull. In this case, additional set of constrains defined by triangles that could be intersected by the edge collapse has to be added to the linear programming problem, as described in paper [7]. Our algorithm does not prevent this type of self-intersections, since our biomedical data consists mainly of convex and smooth surfaces.

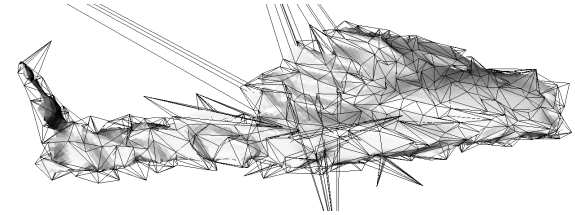
5 Experiments and Results

The method described above was implemented in C++ (MS Visual Studio 2010) and integrated into the MuscleWrapping application[2], which is a part of LHPBuilder software being developed within the VPHOP project [1]. The algorithm was tested on data sets included with this software. The software uses the Multimod Application Framework (MAF) [9], a rapid development visualization system mainly based on Visualization Toolkit (VTK) [8]. For our experiments, a Dell Precision 470 desktop computer (2x Intel Xeon 3.4 GHz, 2 GB DDR2 400MHz RAM, 2x HDD 137 GB SCSI with 10,000rpm, Windows XP Pro) was used.

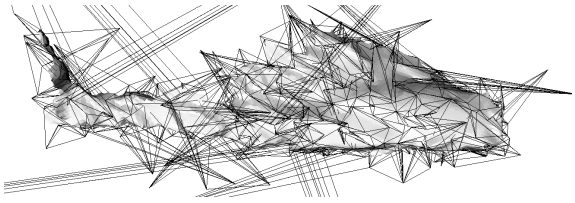
[2] http://graphics.zcu.cz/Projects/Muskuloskeletal-Modeling

5.1 Hull quality and spike prevention

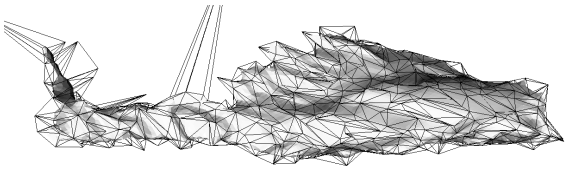
In this section we visually compare the stability of our algorithm to the stability of algorithms based on the *Progressive Hull* method described in this paper. The spike artifacts are commonly present on the hulls created by the original method [7] (see Figure 9a) and the method with a fast priority computation described in Section 3 (see Figure 9b). Hulls created by our implementation of these methods are obviously unusable for later use. Results of the method with a triangle deviation test used by Platis et al. [6] (see Section 4.2, Figure 9c) are slightly better, but the spikes are still present on hull. As we prevent the forming of the spikes specifically, the hull constructed by our algorithm does not contain any (see Figure 9d) and therefore is safely usable for later computations.



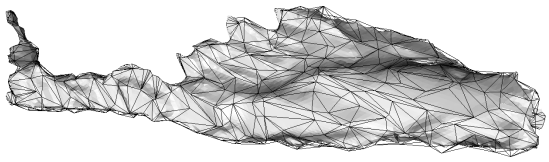
(a) Sander et al



(b) Faster priority computation



(c) Platis et al



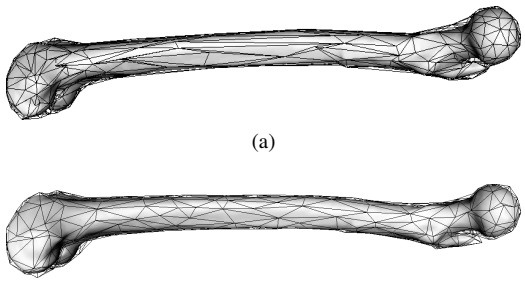
(d) Our method

Figure 9: Comparison of hull quality across the different *Progressive Hull* based methods

The tests shown in Figure 9 were performed on a biomedical mesh of the left *Piriformis* (again extracted from volumetric muscle data) that contains 15000 triangles. The target mesh decimation was set to 90% (90% of the primitives removed), the resulting hull in all four tests contained 1496 triangles.

5.2 Triangle quality improvement

Using the method described in Section 4.3, we achieved a reduction of narrow and sliver triangles on the hull. Visual example can be seen in Figure (see Fig. 10). One can observe a significant triangle shape quality increase.



(a)

(b)

Figure 10: Comparison of triangle quality (a) before triangle angle check (b) after triangle angle check

To confirm this observation, we analyzed the mesh and computed each triangle's compactness using the formula by Gueziec [3] described in Section 4.3. Histograms in Figure 11 show that the most of the low compactness and therefore low quality triangles are removed from the hull.

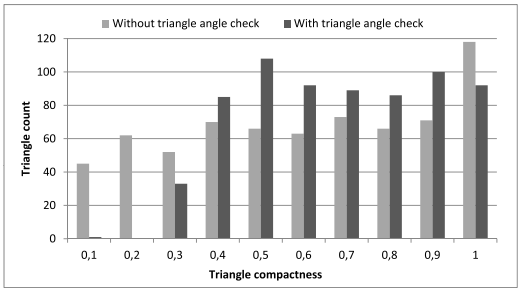


Figure 11: Triangle compactness histograms

In both tests, 90% target decimation was used. The *Femur* mesh was decimated from 13946 to 697 triangles.

5.3 Time consumption

In this section we compare the time consumption of our implementation of the original method [7], the modified method using faster priority computation [6] and our method. We performed tests on 3 meshes with the same setting of 90% decimation.

The results in Table 1 show that our method is approximately nine times faster than the original method by Sander et al. and it may be a couple of seconds slower than the method by Platis et al. [6], however, the additional time consumption is an acceptable trade-off for high quality hulls. Furthermore, considering that the method is typically used in pre-processing, we came to a conclusion that the slowdown of our algorithm in comparison with Platis et al. [6] is insignificant.

Mesh	Method execution time [s]		
	Sander et al.	Platis et al.	Our method
Gluteus Minimus Mesh: 7980 △ Hull: 776 △	104,3	8,6	14,7
Femur Mesh: 13946 △ Hull: 1384 △	215,6	17,9	19,9
Left Piriformis Mesh: 15000 △ Hull: 1496 △	193,4	28,6	23,6

Table 1: Method execution time comparison

5.4 Execution time dependency on the target decimation level

Figure 12 shows the execution time of the algorithm as a variable of a target decimation level. This test was performed on high polygon mesh representation of a *Pelvis* bone. The decimation level denotes how many primitives were removed from the mesh, for example 10% decimation level means that $\frac{1}{10}$ of the original primitive count is removed from the mesh. For this test and default settings, the maximum decimation level was 99.4% (1130 triangles). Note that the preparation time is constant as the measurements were performed using the same mesh and therefore the number of edges entering the priority queue is constant.

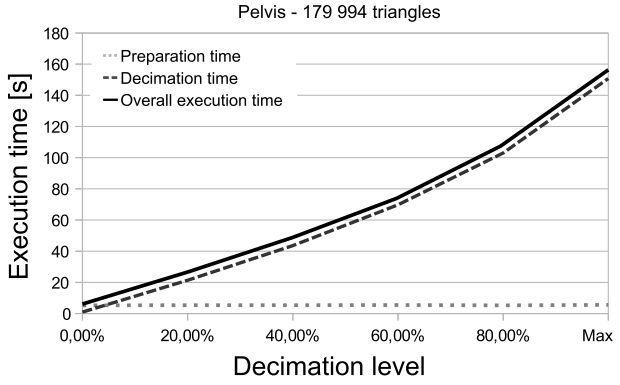


Figure 12: Execution time dependency on target decimation level

In the preparation phase, the priority queue is constructed. Since the priority queue is implemented using a heap data structure, the queue construction is equivalent to a sorting problem and therefore the complexity of the preparation phase is $O(C_p * N * log(N))$, where N denotes a number of edges in the mesh and C_p denotes a time needed to compute the priority.

The decimation itself performs edge collapse on every edge in the queue. The number of the edges in the queue decreases by three with every successful edge collapse. One edge is removed for the collapse itself and two are removed during the area reconstruction process (edges of the triangles, that become singular by this operation, see Section 3). The priorities of the affected edges are updated after every collapse, and the heap property is restored. Since we use a data structure that maintains the neighbors of every primitive in the mesh, the edges that need to be updated in the queue are found in constant time. If we presume that the number of primitives in collapse area is constant, the overall decimation phase complexity is $O(C_c * C_p * \frac{N}{3} * log(\frac{N}{3}))$, where N denotes a number of edges in the mesh, C_p denotes a time needed to compute the priority and C_c denotes a time needed to collapse the edge. Note that $C_c \gg C_p$, as collapsing the edge requires a linear programming problem solution.

5.5 Additional hull results

In figures 13-17 we present the results of our algorithm on several other biomedical meshes.

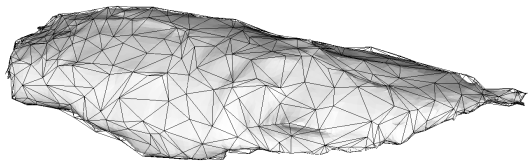


Figure 13: Vastus Lateralis, 19970 △; 1980 △ in the hull

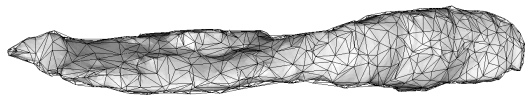


Figure 14: Vastus Medialis, 19986 △; 1982 △ in the hull

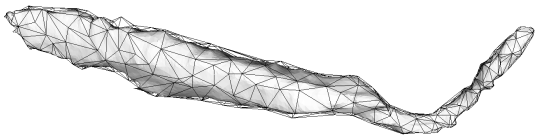


Figure 15: Psoas, 9988 △; 984 △ in the hull

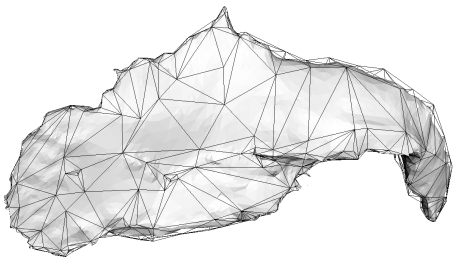


Figure 16: Iliacus, 9968 △; 964 △ in the hull

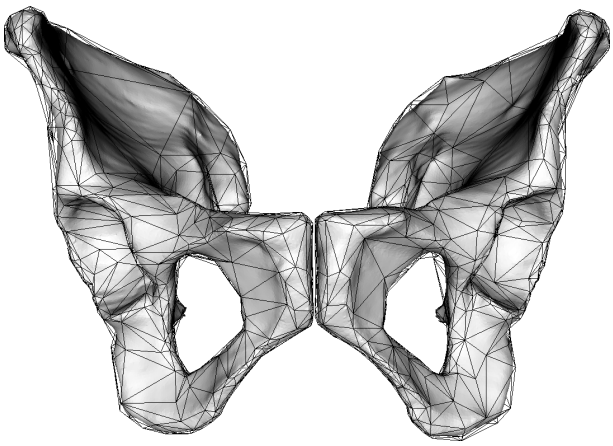


Figure 17: Pelvis, 179994 △; 1130 △ in the hull

6 Conclusion and future work

The proposed method constructs coarse outer hulls that do not contain artifacts that may appear when using the original *Progressive Hull* method [7]. It also offers a better overall quality results. Our algorithm was successfully used in a mesh deformation technique [4] and allowed it to be more precise, due to the fact that the *Progressive Hull* shape preservation is very high and consistent.

Time consumption of the algorithm is a problem that remains unsolved. The method, though it is nine times faster than the original one (see Section 5.3), is still time consuming. For many applications (including ours) this is not a serious drawback, since the method runs only once in the pre-processing.

The future and current work includes usage of a graphics processing unit (GPU) for a faster hull construction as well as further triangle quality enhancement using a better triangle compactness test in order to allow the algorithm to always perform enough iterations to create a coarse outer hull, that contains the desired number of primitives.

7 Acknowledgment

This work was supported by the Information Society Technologies Programme of the European Commission under the project VPHOP (FP7 ICT-223865) and by the Ministry of Education of The Czech Republic under the project 7E11016.

References

[1] VPHOP: The osteoporotic virtual physiological human. http://www.vphop.eu/.

[2] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shape. *ACM Transactions on Graphics*, volume 13(1):pp 43–72, 1994.

[3] André Guéziec. Locally toleranced surface simplification. *IEEE Transactions on Visualization and Computer Graphics*, volume 5:pp 168–189, April 1999. Chapter 5: Description of the Algorithm.

[4] Josef Kohout, Petr Kellnhofer, and Saulo Martelli. Fast deformation for modelling of musculoskeletal system. In *Proceedings of the International Conference on Computer Graphics Theory and Applications: GRAPP 2012*, pages 16–25, Rome, February 2012.

[5] Peter Lindstrom and Greg Turk. Fast and Memory Efficient Polygonal Simplification. *Visualization Conference, IEEE*, pages 279+, 1998.

[6] Nikos Platis and Theoharis Theoharis. Progressive hulls for intersection applications. *Comput. Graph. Forum*, volume 22(2):pp 107–116, 2003.

[7] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 328–329, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[8] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit, Third Edition*. Kitware Inc., August 2004. ISBN: 1-930934-12-2.

[9] Marco Viceconti, Luca Astolfi, Alberto Leardini, Silvano Imboden, Marco Petrone, Paolo Quadrani, Fulvia Taddei, Debora Testi, and Cinzia Zannoni. The multimod application framework. In *Proceedings of the Information Visualisation, Eighth International Conference*, pages 15–20, Washington, DC, USA, 2004. IEEE Computer Society.