Mesh Merging:

for 3D triangular surface meshes

Fredrik Viström

Project in Computational Science: Report

January 2014

Abstract

An algorithm for merging 3D triangular surface meshes is presented. The algorithm is based on a few key steps; finding the intersecting triangles, extracting boundary polygons, connecting boundary polygons and merge the pre-processed meshes. The algorithm shows promising results regarding the quality of the merging, but it can not handle all types of intersection. Future work would be generalizing the algorithm, validating and testing it more and parallelize the implementation.

Contents

Introduction

A mesh is a set of vertices and connections between them. The meshes that have been used throughout this report are *3D triangular surface meshes*, i.e. the vertices are points in 3D space (three coordinates) and the vertices are connected to form triangles. All the vertices are connected into triangles, whose union forms a closed surface, i.e. a volume is enclosed by the mesh. From here on a 3D triangular surface mesh will simply be refereed to as mesh. An example of such a mesh is presented in Figure 1.

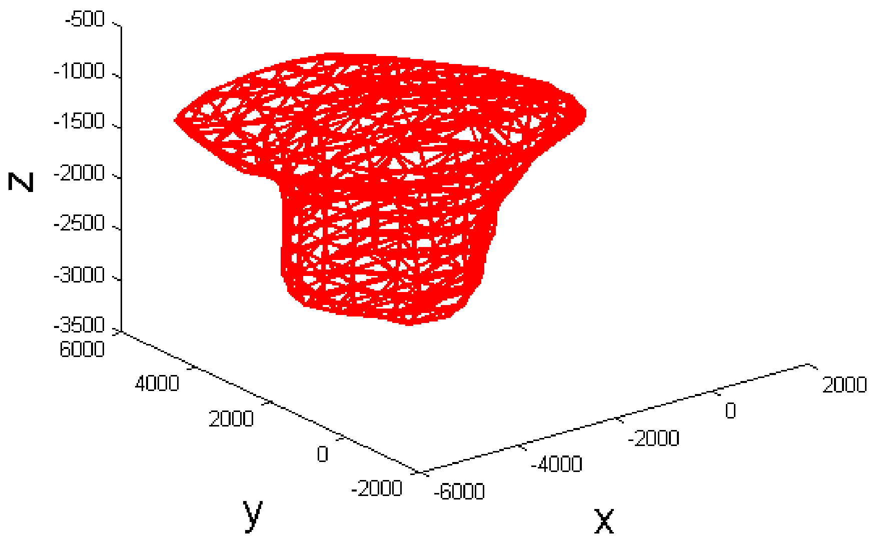

Figure 1: A 3D triangular surface mesh.

If there are two meshes, M_A and M_B, enclosing two different volumes, V_A and V_B, respectively, then the case can be that V_A and V_B are intersecting. This means that M_A and M_B are intersecting with each other as well. It can then be desired to create a new mesh, for the union of the two volumes. This can be performed by merging the two meshes into a mesh M_M such that the volume enclosed by M_M is V_M, where $V_M = V_A \cup V_B$. An example of two intersecting meshes is presented in Figure 2, where M_A is shown in red and M_B in blue.

Method

Given two meshes M_A and M_B with vertices v_i^A and v_j^B, for $i = 0, 1, ..., N_v^A - 1$ and $j = 0, 1, ..., N_v^B - 1$, and triangles t_i^A and t_j^B, for $i = 0, 1, ..., N_t^A - 1$ and $j = 0, 1, ..., N_t^B - 1$, respectively. Where N_v^A, N_v^B, N_t^A and N_t^B are respectively the number of vertices and triangles in M_A and M_B. The algorithm for merging the two meshes, into one mesh M_M, can be summarized in the following steps:

- Find the intersecting triangles from the two meshes.
- Extract a boundary polygon for each mesh from the intersecting triangles.

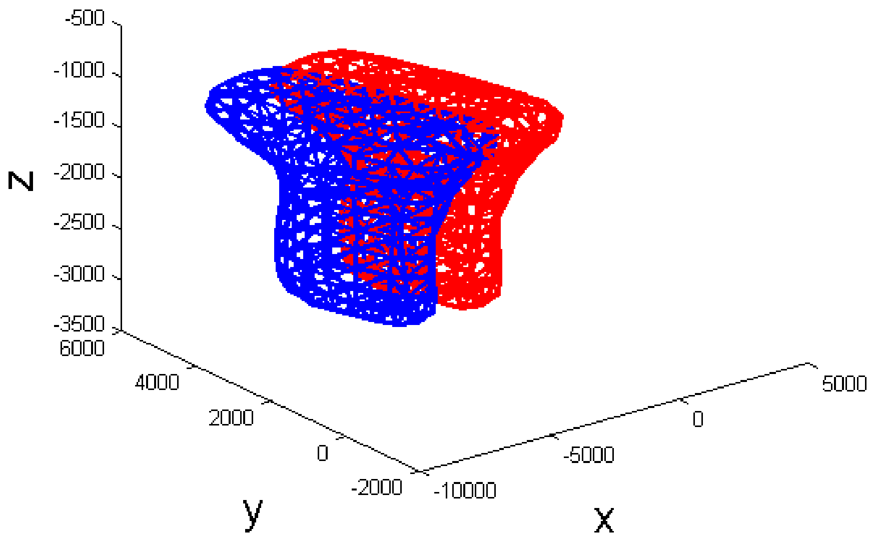

Figure 2: Two intersecting meshes, M_A shown in red and M_B in blue.

- Connect the two boundary polygons with triangles.
- Merge the pre-processed meshes.

This means that this algorithm is limited to the cases where there is one single intersection region, such that there is only one boundary polygon for each mesh. In order to handle all types of intersection the algorithm has to be generalized such that several boundaries for each mesh can be generated and then connecting the correct boundaries with each other.

Finding Intersecting Triangles

If a triangle from M_A intersects with any triangle from M_B, then it is an intersecting triangle from M_A. In order to find all the intersecting triangles from M_A the naive way is to test every triangle in M_A against all the triangles in M_B. The complexity of the naive way is $N_t^A \cdot N_t^B$, hence it will not scale well for meshes with a large number of triangles. In order to reduce the number of triangle-triangle intersection tests some other, cheaper, tests can be performed to see if a triangle-triangle intersection test is unnecessary.

A center point for M_A, defined by

$$c_A = \frac{\sum_{i=0}^{N_v^A-1} v_i^A}{N_v^A}, \tag{1}$$

can be computed and used in order to define a sphere, S_R^A, that contains the entire M_A. This sphere will have the center point c_A and a radius R_A, where

$$R_A = \max_i \; dist(v_i^A,\, c_A) \tag{2}$$

and $dist(v_i^A,\ c_A)$ is the Euclidean distance between v_i^A and c_A. A second sphere, S_r^A, can also be defined, that has the same center point but a different radius, defined by

$$r_A = \min_i\ dist(v_i^A,\ c_A). \tag{3}$$

The sphere is defined in a such a way that no triangles from M_A will lie inside S_r^A. Besides these two spheres the longest edge, l_A, in M_A is found. The longest edge means the maximum distance between any two vertices contained in the same triangle from M_A, that is

$$l_A = \max_i \left(\max_{v_n, v_m \in t_i^A} dist(v_n,\ v_m) \right). \tag{4}$$

The values computed in Equation (1)-(4) for M_A are also computed for M_B. From these mesh data some tests can be defined in order to reduce the number of triangle-triangle intersection tests. If a triangle, t_i^A, from M_A lies outside the sphere S_R^B it means that t_i^A can not intersect with any triangle from M_B. This means that all triangle-triangle intersection tests between t_i^A and any triangle from M_B can be skipped. A similar argument for triangles from M_A that lies inside S_r^B allows these triangle-triangle intersection tests to be skipped as well.

If a triangle t_i^A can not be rejected as a intersecting triangle by any of the two tests defined above it has to be tested whether it intersects with any triangle from M_B. However, a full triangle-triangle intersection test can still be avoided for some of the triangles from M_B. If the distance between t_i^A and t_j^B is too large, then the triangles can not intersect. Instead of calculating all possible distances between all vertices in t_i^A and t_j^B only one distance is calculated. If the distance between v_n and v_m is larger than l_A+l_B, it means that t_i^A and t_j^B can not intersect, where $v_n \in t_i^A$ and $v_m \in t_j^B$.

The triangle-triangle intersection test used here is presented by Tomas Möller [1]. The principle of the method will here be described shortly, for a more detailed explanation the reader is referred to the article. For determining whether two triangles, t_A and t_B, intersect the equations for the corresponding planes π_A and π_B are computed, where $t_A \in \pi_A$ and $t_B \in \pi_B$. Then the signed distances from the vertices of t_A to π_B are computed. The signed distance means that that the sign will positive on one side of the plane, and negative on the other side. The positive side will be the side in which the normal to the plane is pointing. If the signs of these three signed distances are the same it means that t_A lies on one side of π_B and that they do not intersect, hence does not t_A and t_B either. The same computations are then made for t_B and π_A. If the special case when the two planes are co-planar is rejected, then the intersection of π_A and π_B is a line L. Moreover, the triangles t_A and t_B will intersect with L, as the cases when they don't are already rejected. The intersection of t_A and L will be an interval on L, and the same goes for t_B. If these two interval on L overlaps, it means that t_A and t_B intersect.

The intersecting triangles from the meshes shown in Figure 2 are presented in Figure 3. In this example the red triangles come from M_A and the blue ones from M_B.

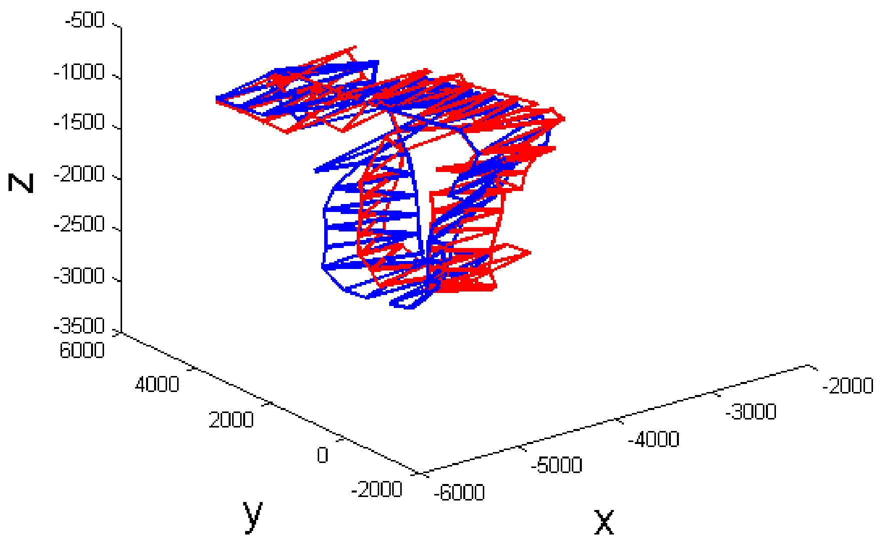

Figure 3: Intersecting triangles from M_A (red) and M_B (blue).

Finding Boundary Polygons

Due to the fact that M_A and M_B are surface meshes the intersecting triangles from M_A and M_B will both form a closed chain of triangles, meaning that all triangles share at least one edge with an other intersecting triangle. The special case is when a mesh only has one intersecting triangle, then the boundary polygon will be a list of the three vertices in that triangle. This further means that the intersecting triangles from M_A will form a surface S_A that works as a boundary between the triangles in M_A that lie outside M_B and the ones that lie inside M_B.

It is not a boundary of intersecting triangles that are sought after, but a boundary polygon. A boundary polygon is a closed chain of vertices. In order to extract a boundary polygon from the intersecting triangles all the edges from the intersecting triangles are stored. Each edge is stored once per triangle, meaning that a shared edge will be stored twice. The edges that are stored only once will form the boundary of S_A and the edges that are stored twice will lie in the interior of S_A. However, S_A will have two boundary polygons, one that lies inside M_B and one that lies outside M_B. The boundary polygon that lies outside M_B is the boundary polygon that is sought after. In order to determine which boundary polygon that lies outside M_B it is sufficient to check one vertex from one of the two boundary polygons. If this vertex lies outside M_B, than the entire boundary polygon does too. Otherwise, the entire boundary polygon lies inside M_B and it is the other boundary polygon that are sought after.

In order to determine whether a vertex lies inside a mesh a ray is cast from the vertex in any direction. Then the number of times the ray traverses through a triangle is counted. If it occurs an odd number of times, then the vertex lies inside the mesh, otherwise outside. To determine whether a ray and a triangle intersect or not a method presented by Thomas Möller and Ben Trumbdore [2] is used. The method is based on determining the Barycentric coordinates (u, v) (related to the triangle) for the point where the ray intersects with the plane in which the triangle lies and the signed distance, d_t, between the vertex and the plane. If the ray and the triangle intersect, then

$$\begin{aligned} u &\geq 0, \\ v &\geq 0, \\ u + v &\leq 1, \\ d_t &\geq 0. \end{aligned} \tag{5}$$

An important thing to remember is that the vertices of the boundary polygon are a subset of the mesh vertices and that the original indexing among the vertices has no impact on the order they appear in the boundary polygon. The boundary polygons from the example shown in this report are presented in Figure 4.

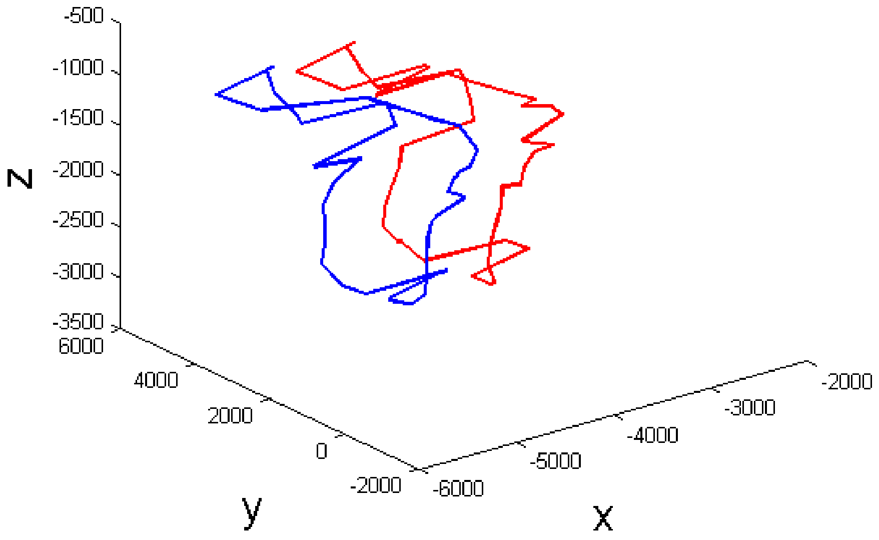

Figure 4: Boundary polygons, B_A shown in red and B_B in blue.

Connecting Boundary Polygons

From the intersecting triangles from M_A and M_B the boundary polygons B_A and B_B are extracted. B_A is a list of N_B^A vertices, where a vertex from B_A is denoted by $v_{B,i}^A$. The vertices $v_{B,i}^A$ and $v_{B,i+1}^A$ are connected, for $i = 0, 1, ..., N_B^A - 2$, and vertex $v_{B,0}^A$ and $v_{B,N_B^A-1}^A$ as well. The same goes for B_B. These boundary polygons are then connected by forming triangles, referred to as *boundary triangles*, between them in such a way that a single surface is formed, with boundaries B_A and B_B. This is performed by assuring three things:

- A boundary triangle takes two vertices from one of the boundary polygons and one from the other boundary polygon.

- Two adjacent vertices on a boundary polygon must also be connected in exactly one boundary triangle.
- Two non adjacent vertices from a boundary polygon can not be connected in any boundary triangle.

If B_A and B_B are of different length, $N_B^A \neq N_B^B$, then the shorter one is interpolated until $N_B^A = N_B^B$. If $N_B^A < N_B^B$, then the two vertices $v_{B,i}^A$ and $v_{B,i+1}^A$ (or $v_{B,0}^A$ and $v_{B,N_B^A-1}^A$) with the greatest distance between them are found and a new vertex is inserted between them. When a new vertex v_{new} is inserted between v_n and v_m the triangles with both v_n and v_m must be split into two new triangles. This means that a triangle with the vertices $\{v_n, v_m, v_o\}$ will be split into two new triangles with the vertices $\{v_n, v_o, v_{new}\}$ and $\{v_m, v_o, v_{new}\}$ respectively.

When connecting B_A and B_B, one of the boundary polygons is chosen as a base for the algorithm, i.e they are not treated in the same way. This means that there is a difference between *connect(B_A, B_B)* and *connect(B_B, B_A)*, where *connect(B_1, B_2)* is the algorithm for connecting boundary B_1 and B_2 with B_1 as the base. Below is the algorithm described with B_A as the base boundary polygon.

The first step in connecting B_A and B_B is to find the vertex $v_{B,b}^B$ that satisfies

$$\min_j \left(dist(v_{B,i}^A, v_{B,j}^B)^2 + dist(v_{B,i+1}^A, v_{B,j}^B)^2 \right), \tag{6}$$

for $i = 0$. The subscript b in $v_{B,b}^B$ is used to denote that this vertex will form a bound when connecting the boundary polygons. The three vertices $\{v_{B,0}^A, v_{B,1}^A, v_{B,b}^B\}$ will form the first triangle. The next step is to find the vertex from B_B that satisfies Equation (6) for $i = 1$. This vertex will be denoted by $v_{B,k}^B$ and the three vertices $\{v_{B,1}^A, v_{B,2}^A, v_{B,k}^B\}$ will form the second triangle. If $v_{B,k}^B = v_{B,b}^B$, $v_{B,k}^B$ is updated by taking the vertex that satisfies Equation (6) for $i = 2$. The procedure is repeated with for an increasing i until $v_{B,k}^B \neq v_{B,b}^B$. When this occur there is four possible cases:

$$\begin{aligned}
&\text{(a)} \quad k > b \text{ and } k - b \leq \frac{N_B^B}{2}, \\
&\text{(b)} \quad k > b \text{ and } k - b > \frac{N_B^B}{2}, \\
&\text{(c)} \quad k < b \text{ and } b - k \leq \frac{N_B^B}{2}, \\
&\text{(d)} \quad k < b \text{ and } b - k > \frac{N_B^B}{2}.
\end{aligned} \tag{7}$$

From these cases a direction of movement is defined. The direction of movement can be either increasing or decreasing. What it specifies is how one should step in B_B to get from b to k, along the shortest path. This means that even if $k < b$ it might be better to move in an increasing direction from b and jump from $N_B^B - 1$ to 0 before further increasing from 0 to k. Likewise for a decreasing direction when $k > b$. This leads to an increasing direction for case (a) and (d), in Equation (7), and a decreasing direction for case (b) and (c).

In order to fulfill the three statements, mentioned earlier in the bullet list, for assuring that the boundary polygons are connected in a admissible way the vertices that lie between $v^B_{B,b}$ and $v^B_{B,k}$ have to be connected to $v^A_{B,i}$, where i is the highest index used so far in Equation (6), i.e. the index that led to $v^B_{B,k} \neq v^B_{B,b}$. These triangles are $\{v^A_{B,i}, v^B_{B,j}, v^B_{B,j\pm 1}\}$, for $j = b, b \pm 1, ..., k \mp 1$, with the sign determined by the direction of movement. The set of indexes $b \pm 1, ..., k \mp 1$ is denoted by I.

The same procedure as described in the two last paragraphs is then repeated, in a slightly modified way. Once again the three statements for assuring an admissible connection between the two boundary polygons come in to play. Equation (6) is solved with j restricted to $j \notin I$ for increasing i until a new k is found such that $v^B_{B,k_{new}} \neq v^B_{B,k_{old}}$, where k_{old} refers to the previous k. Triangles are generated during this iteration as for the first triangles. And when k_{new} is found such that the stopping condition is fulfilled the triangles $\{v^A_{B,i}, v^B_{B,j}, v^B_{B,j\pm 1}\}$, for $j = k_{old}, k_{old} \pm 1, ..., k_{new} \mp 1$, are formed and the indexes $k_{old}, k_{old} \pm 1, ..., k_{new} \mp 1$ are added to I. This iteration is stopped when $i > N^A_B - 1$. In the last iteration, $i = N^A_B - 1$, the index $i+1$ in Equation (6) is replaced by index 0. In each iteration k_{new} is updated and k_{old} will be k_{new} from the previous iteration.

The last step when connecting the boundary polygons is to generate the triangles $\{v^A_{B,0}, v^B_{B,j}, v^B_{B,j\pm 1}\}$, for $j = k_{new}, k_{new} \pm 1, ..., b \mp 1$. No triangles are needed to be generated if $k_{new} = b$. The boundary triangles for the example followed in this report are presented in Figure 5.

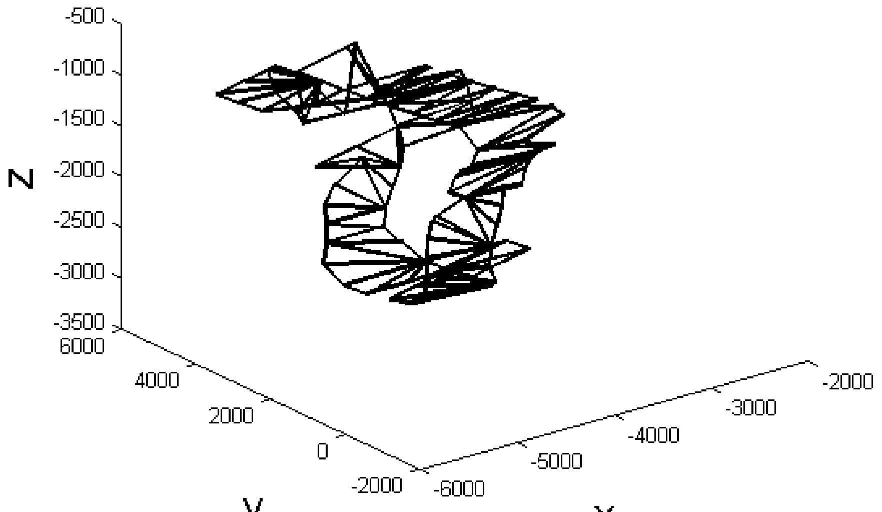

Figure 5: Boundary triangles.

Merging the pre-processed meshes

In order to get a complete mesh, among the already found boundary triangles, only some of the triangles from M_A and some of the triangles from M_B are used. Which triangles to use from M_A and M_B are determined when the intersecting triangles from the corresponding mesh is found. The triangles to use from M_A are the ones that lie outside M_B, and the triangles to use from M_B are the ones that lie outside M_A.

If a triangle from M_A lies outside S_R^B it is no longer a possible intersecting triangle, moreover it is then also known that it lies outside M_B and that it should be used in the merged mesh. Furthermore, we utilize that a triangles from M_A that lies inside S_R^B also lies inside M_B and that it should not be used in the M_M. If the triangle is an intersecting triangle it should not be used either. If the triangle does not fulfill any of the condition mentioned above it has to be tested specifically if it lies outside M_B or not. This is done by testing one of the vertices in the triangle and see if this vertex lies inside or outside M_B.

Result and Discussion

The algorithm for merging two meshes presented above has been implemented and tested in C#. Merging two meshes can be performed in several ways, affecting both the outcome mesh and the speed of the algorithm. Visual inspection has been used to qualitatively assess the output merged meshes, because there is not any efficient way for mathematically determine the quality of a mesh. During the visual inspection aspects that has been taken into account are the shape of the boundary triangles, they should preferably not differ to much from an equilateral triangle, and a vertex should not take part in too many boundary triangles. For the visualization of the meshes MATLAB has been used.

The implementation reads the two meshes from file and writes the merged mesh to its own file. If two meshes are given that do not intersect, the outcome is an empty file. This can be determined immediately after finding the intersecting triangles, because there will be no intersecting triangles. When measuring the execution times, the code was run on an Intel Core i5-3570K CPU @3.40 GHz processor and 8.00 GB RAM, running a 64-bit Windows 7 operating system. The implementation has not been parallelized, so it has only been running on one core.

In the example previously presented in this report the intersection between the two meshes is to be considered as deep. Besides this deep intersection other types of intersections have also been studied, namely a shallow intersection and a merge between two meshes without any intersection. The merged mesh for the deep intersection is presented in Figure 6, while the two meshes with shallow intersection and no intersection are presented in Figure 7 and Figure 8 respectively.

Besides the type of intersection between the meshes the number of triangles in the meshes has also been varied in the tests. In the meshes seen in Figure 2, 7 and 8 the number of triangles are around 750 for each mesh. Meshes with the same shape as in the three example given here but with around 20 000 triangles in each mesh have also been studied. The impact of the optimizations presented for reducing the number of triangle-triangle intersection tests have also been analyzed.

Execution times for several types of merges are presented in Table 1, where T_{it} is the time for finding the intersecting triangles and T_t is the total execution time, from which several conclusions can be drawn. First of all, one can see that the largest part of

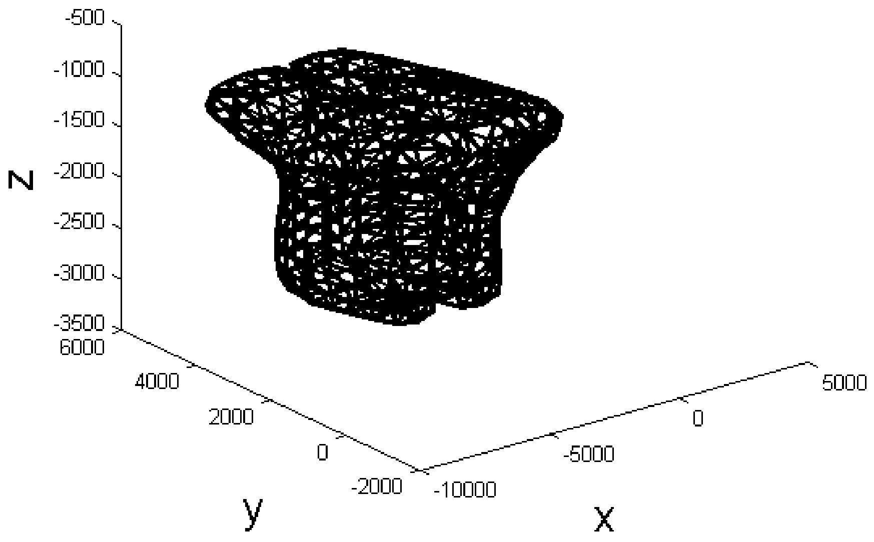

Figure 6: The resulting mesh for merging the meshes M_A and M_B from the example in this report.

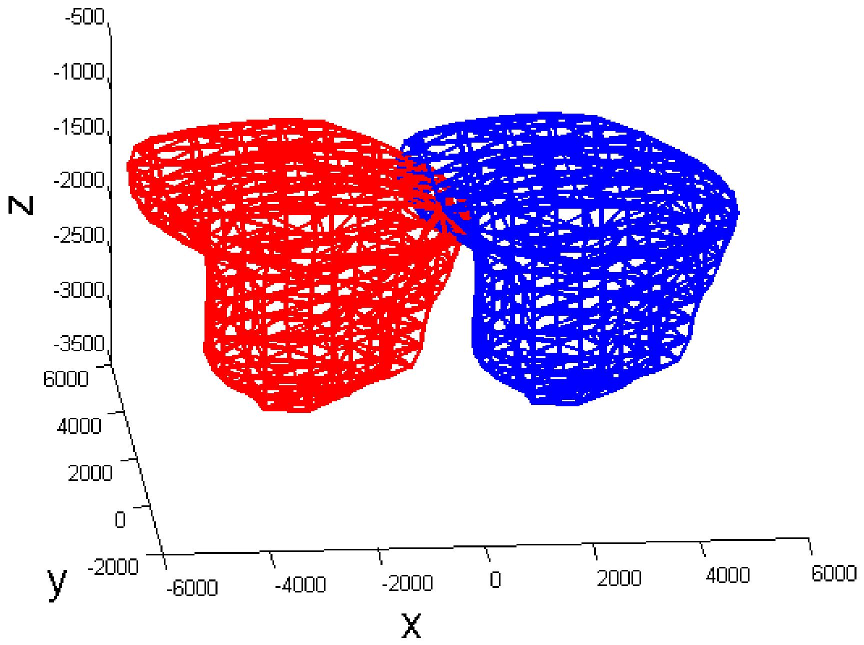

Figure 7: Two meshes with shallow intersection.

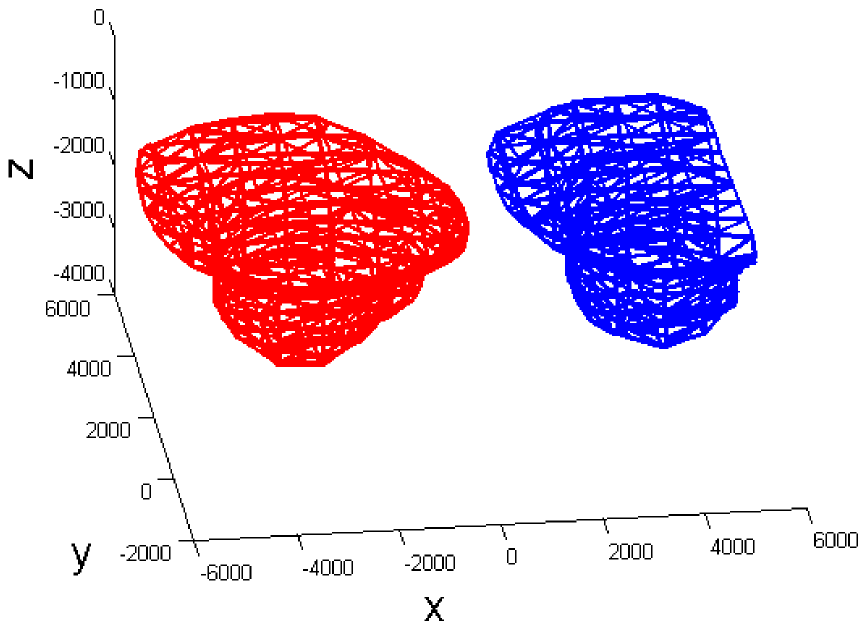

Figure 8: The meshes without any intersection.

the total execution time, in all cases, comes from finding the intersecting triangles. This means that any optimization for speeding up the algorithm should be focused on the part where the intersecting triangles are found, as it has been. When the optimizations have been used the execution time has been drastically reduced. It is also seen that the impact of the optimizations heavily dependent on the type of intersection the merging is performed on. If there is no intersection between the meshes to merge the impact is huge, but there is also a good performance boost for the deep intersection.

As stated earlier, the complexity of the naive way for finding the intersecting triangles is $N_t^A \cdot N_t^B$. This can also be seen from the execution times for the merge with deep intersection and no optimizations. The ratio between the number of triangles in the two cases is

$$\frac{19\,580 \cdot 19\,568}{764 \cdot 760} \approx 660, \tag{8}$$

and the ratio for the execution times is

$$\frac{392\,682}{613} \approx 641. \tag{9}$$

If the values from Equation (8) and (9) are compared one can see that the performance is slightly better than expected. This is due to the optimizations in the triangle-triangle intersection test itself. If one instead would look at the case when optimizations have been used the ratio between the execution times will be

$$\frac{129\,729}{234} \approx 554. \tag{10}$$

It is then seen that the complexity for finding the intersecting triangles are reduced when optimizations are used, since $554 < 641$.

Table 1: Execution times for merging meshes with different types of intersection, different sizes and whether to use optimizations or not.

Intersection	Optimization	$N_t^A \times N_t^B$	T_{it} (ms)	T_t (ms)
deep	yes	764 × 760	234	244
deep	no	764 × 760	613	623
deep	yes	19 580 × 19 568	129 729	129 863
deep	no	19 580 × 19 568	392 682	392 812
shallow	yes	764 × 764	99	106
shallow	no	764 × 764	625	632
shallow	yes	19 580 × 19 636	56 617	56 693
shallow	no	19 580 × 19 636	397 794	397 870
none	yes	764 × 544	7	7
none	no	764 × 544	449	449
none	yes	19 580 × 13 546	2 686	2 686
none	no	19 580 × 13 546	273 864	273 864

Conclusion

The algorithm shows promising results regarding the quality of the merging, but it is limited to the cases where there is only one intersection region. Some optimizations have been implemented in order to increase the speed of the algorithm, as it is a limiting factor for the algorithm to handle meshes with a large number of triangles. The next step for speeding up the execution would be to parallelize the implementation, in particular the part for finding the intersecting triangles. Further testing for validating the quality of the merged mesh is needed, as well as a generalization of the algorithm to handle all types of intersection.

References

[1] T. Möller. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 2(2):25–30, 1997.

[2] T. Möller and B. Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997.

Acknowledges

I would like to thank Nader Salman at Schlumberger for the help I have gotten during this project, such as the project itself, test data and guidance. I would also thank Maya Neytcheva as Uppsala University for hosting an interesting project course, where this project has been taken place, with instructive seminars about oral presentation and posters.