Volume definition from incomplete and unreliably oriented surfaces

Emile Martinez
ENS de Lyon

Directed by:
Marco Attene
IMATI, CNR

May 1, 2022 - July 22, 2022

Contents

1 Introduction

A common way to represent a surface in computer science is to discretize and represent it as a triangular mesh. A 3D surface is often used to indirectly represent an enclosed solid object, and being able to know if a point is inside or outside such a solid is important for many applications. This inside/outside classification problem is inherently ill-posed when the surface is open.

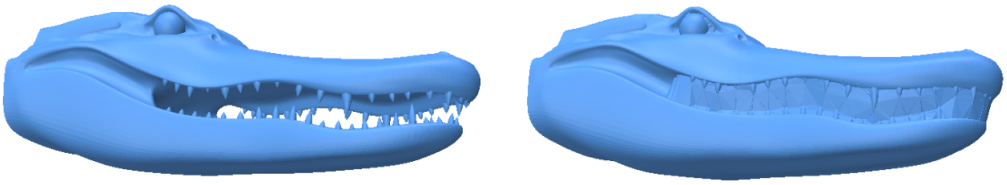


Figure 1: Left: Expected output, Right: Actual output when the orientation is random, with the method using the orientation

In these cases, existing techniques strongly rely on the orientation of the triangle. Indeed, it is quite common that the order in which the vertices of the triangles are given is giving which side of the triangle is inside, and which is outside. Our point is thus here to find a way to close a surface, to differentiate the outside from the inside, without using the triangle orientation.

This work is built from a previous work from Marco ATTENE and Lorenzo DIAZZI [4]. In this previous work, we obtain a polyhedral subdivision of the space respecting the input triangles. After this subdivision, each cell is classified between internal and external. This classification is strongly relying on the orientation. For example, if the orientation is random, the result lose his sense, like Figure 1 is showing. Thus, we will try here to classify the cells without the use of the triangle orientations.

To do so, our method rely on the use of ray shooting. Indeed, whenever a surface is closed, to know whether a point is inside or outside, we can shoot a ray from the point to the infinity, and count the number of times the ray intersect the mesh. If the number of intersection is odd, the point was inside, otherwise, it was outside. Here, we do the same, but as the ray can cross a hole, we may have the wrong information. Thus, we take different direction, hoping that the majority of ray won't cross a hole. The all point of the work is then to try to do this efficiently.

In this report, we will then see the different way by which we can obtain the ray information. We can indeed use a collision detector, but it is not the only way. We can also use the fact that we have a polyhedral subdivision of the space, which allow us to navigate inside the mesh, and then to propagate efficiently information.

After having perform the ray shooting step, the point is then to try to exploit it efficiently. In order to do that, we get the dual graph of the subdivision (cells being nodes, and facets being edges). Different way to classify have been tried but the most effective seemed to be a min-cut, minimizing then the added area to close the surface, while maximizing the correlation with the ray information.

All the code as been implemented in C++, and tested on the Thingi10k dataset [16] giving in the general case satisfying result, without aberration, but outputting in some specific cases not the expected surface -but outputting anyway something with sense-. If on the well oriented surfaces, it is a bit less efficient than the method we compare with, it is of course for far way more effective on misoriented surfaces.

2 State of the art

Depending on the point of view we adopt on this works, the states of the art will be different. From the pragmatical point of view, we are just taking a polyhedral space division fitting with a set of triangles, and we decide which cell should be inside, and which should

be outside. In this context, there are few works, but the existing one seems to all use the triangle orientation, like in the works we are building from [4], explaining thus the need of our work. Having the same basis, we will compare our work with this methods. (It exists works which are classifying cells without orientation but they don't use a space division fitting with the set of triangles, they are using other subdivision, having others required properties as uniformity for example.)

From an outside point of view, we can consider this work as a hole filling method. Then there are plenty different works. In hole filling methods, there are two main methods: the volume-based one and the surfacic-based one [7].

Surfacic-based methods The surfacic-based methods are trying to create a watertight surface by directly adding triangles. These methods are often acting locally, fulfilling holes one by one. C. Feng *et al.* [5] are performing a naive triangulation, which can be -depending on the size of the hole- refined and then harmonize with the rest of the mesh (using the normals of the triangle, and the angle created). The same procedure but with different algorithms at each step is apply, for example by P. Liepa [9]. L. Tekumalla and E. Cohen [14] are directly proceeding to a good triangulation, using notably the projection of a point on a implicit surface defined by a set of point to guess where to add a point. With the same basis, W. Zhao *et al.* [15] are also performing a smoothing using the poisson equation (equivalent in this case in a minimizing problem involving the gradient of the surface). These methods can give great results but may face difficulties when the holes are disconnecting the surface.

Volume-based methods The volume-based methods are working in two steps: they first subdivide the space in convex polyhedral cells, and then determine which is inside and which is outside. [11] The completed surface is then the frontier betweeen the inside and the outside. Here we can distinguish between two kind of algorithm: the one with an approximated rendered surface, and the one with an exact surface away from the hole.

Approximative methods Most of the time, these methods are building a regular grid of cube. The solid being voxelized, when we rebuild the surface we have then an approximated surface. To classify between inside and outside, Davis *et al.* [3] are using a signed distance function defined in the vicinity of the hole, and then the function is diffused to extend the surface, but to determine the signed distance function, the triangle orientation is required. Without the orientation, Nooruddin and Turk [10] are performing a classification very close from the one presented in this report. Indeed, they draw rays in many directions to determine if a voxel is inside or outside by a majority vote, each ray giving the information for every crossed voxel. Nevertheless, the subdivision of the space they are using being quite different from the one we use, their method -and the way to apply it- and their results remain quite different from ours.

Methods preserving the original mesh On the other hand, some methods are preserving the original mesh away from the holes. Podolak and Rusinkiewicz [12] are building an octree, being refined near the hole until each cube of the octree does not contain holes, or contains few triangles enough. Then, a cube which doesn't contains a hole can be split in two cells (not necessarly convex) (one labeled inside, one labeled outside), and the cube which are on the border of a hole are tetrahedrized. Thus, we obtain a subdivision of the space respecting the input triangles. A min-cut algorithm is then performed (the nodes being the cells, and the edges being the faces of the cell), minimizing the added area and separating the cells classified outside and the cells classified inside. With a close idea, Attene *et al.* [4] are performing a polyhedral subdivsion of the space fitting with the input mesh. If the point of this work is to perform a robust subdivision, in an efficient way, (subdivision we

will start from) they also define the interior and the exterior. To do so, they take the same graph and perform also a min-cut algorithm minimizing the added area, but also trying to maximize the right labeling of the cells whose label can be given by the input triangles, through their orientation. Both of these methods are quite efficient, nevertheless, they both strongly rely on the orientation of the triangles to determine when a cell is to be inside or outside, to guide the min-cut algorithm. Another method to be mentioned has been made by Jacobson *et al.* [6]. Here, in the min-cut algorithms, we try to minimize the added area but also to classify the cells with respect to their winding number. This number can be interpreted as the signed number of turns you will over yourself if you follow a curve (or a surface), and in a close surface, the parity of this number directly gives the classification. Nevertheless, once again the orientation is required to compute efficiently these numbers whenever the surface is not closed. We can note an attempt to define the orientation when the orientation is not reliable by Takayama *et al.* [13] using ray shooting.

3 Methods

We built our work from a polyhedral subdivision of the space. Then, our work, is just about the classification of this subdivision We don't work try to refine the subdivision to have a better one. The subdivision is guaranteed to be non-degenerated, meaning that every face and every cell is convex, with a non null measure. The mesh is store with the classical structure, each cell storing its faces, each faces storing its edges and the connected cells, and each edges storing its two points. Here, we will call the faces that are included in the union of the triangle input, the "black facets", and the faces whose interior is disjoint from the interior of the input triangles the "white facets" (every facet is guaranteed to be either black, either white).

We are going to use the ray shooting principle. Indeed, with a closed surface, one can determine if we are inside or outside just by shooting a ray to the infinite. With a closed surface, every time the ray cross a black facet, we switch between inside and outside. As the infinite is outside, the parity of black facets crossed give us the right classification. Of course, when the surface is open, its works only if we don't cross a hole. We have then to use the power of statistics which will grant us, if we shoot enough rays, in enough direction, clear data, that will have to be post processed. It will then reduce the influence of holes and allow us to ignore ill cases as ray being tangent to a cell.

We are then going to put into practice different methods to get this information and to treat them.

3.1 Gathering information on the potential classification of each cell

For a triangular mesh M (i.e. a set of triangles), let's also note M the union in $\mathbf{R}^3$ of the triangles of M. For $x \in \mathbf{N}$, we define $x\%2$ as the remainder of the euclidean division of x by 2. Let's also define for $A \in \mathbf{R}^3$ and $r \in \mathbf{R}$, $S(A, R)$ the sphere of center A and radius R, and $R = \max_{B \in M} |A - B|$

Let A and B be two points in $\mathbf{R}^3$, and let $M = |M \cap [A, B]|$ denote the number of intersections between the segment $[A, B]$ and the surface M. For any point A in $\mathbf{R}^3$, we define its "innership" through the following integral $\int_{B \in S(A,R)} \big|M \cap [A, B]\big|\%2 \; dB$ divided by the total sphere area. If this quotient is more than 0.5, we declare A to be inside M, otherwise A is outside.

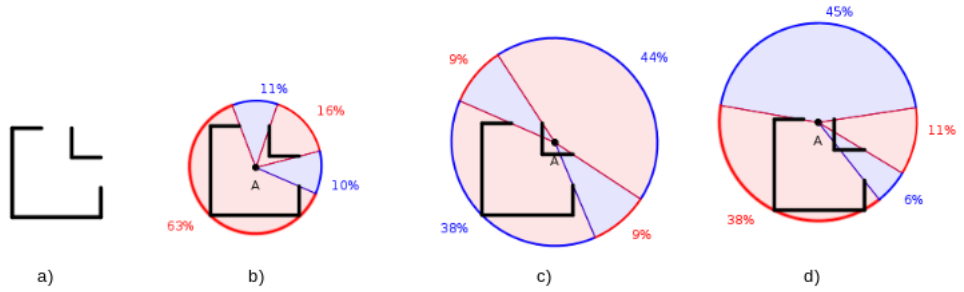


Figure 2: 2D representation of our definition, with in red the direction saying inside, and in blue the direction saying outside: a) input mesh b) Case when A is inside (indeed, the length of the red arcs is bigger than the blue ones) c) Case when A is outside d) Case when we are close from a hole, where hole farther away have importance

Here, we are assuming that, for most of the point, there is no hole in the majority of the directions (a hole being defined relatively to a way to close the surface). Whenever the holes are small, and we are not too close from it, this assumption can hold, as figure 2 is showing.

Since the problem is inherently ill-posed, our definition may sometimes produce unexpected results (see 5.

3.1.1 Using AABB detector

The first approach would consist in calculating the integral by discretizing the sphere. This method being from far too expansive, we won't use it. As we just want to compare the measure of two set, we are just going to use a probabilistic approach.

Here what we want to do is, giving a subset S_1 of $S(A,R)$, to know if the measure of S_1 is more than the half of the one of $S(A,R)$ (it is exactly the definition of section 3.1 with $S_1 = \{B \in S(A,R) / \, |M \cap [A,B]| \,\%2 = 1\}$). To do so, we are going to take random elements of $S(A,R)$, and count if there is more than half of the point in S_1. To have a concluding enough result, we are going to take random point until the number of point in S_1 is far enough from the mean. If we note s_1 the measure of S_1 and s the measure of $S(A,R)$, then we want to determine whether $s_1 > \frac{s}{2}$ or not. To do so, we have access to independent random variables following a Bernoulli law of parameter $p = \frac{s_1}{s}$. Let's note them $(X_n)_{n \in \mathbf{N}}$. To know if we can stop -or if we should compute more of the random variables outputs-, we want to determine, for $n \in \mathbf{N}$, representing the number of element we picked, $\max \left\{ k \in [\![0,n]\!] \Big/ \mathbb{P}\left(p < 0.5 \Big| \sum_{j=0}^{n} X_j \leq k \right) > 0.95 \right\}$. Indeed this threshold means the biggest k for which if we have less than k elements picked which are in S_1, then we are 95% sure that $s_1 < \frac{s}{2}$. The protocol consists then just in picking elements until we are below this number, or above n minus this number (if S_1 is big).

Nevertheless, to do the rigorous calculus of this maximum, we should be able to determine the probability law of p, which can depend a lot on the mesh. As we want something which does not depend on the distribution of p, our result is going to be mathematically wrong.

So what we do is to take as threshold, for $n \in \mathbf{N}$, $\max \{ k \in [\![-1,n]\!] / \mathbb{P}(Y_n \leq k) \leq 0.05 \}$ where Y_n follow a binomial law of parameter n, $\frac{1}{2}$. What the threshold means is that if p

is greater than 0.5, then the probability to be below this threshold is less than 0.05. Of course, it is not what we wanted but it doesn't require to know the probability law of p. The thresholds for the twenty five first n are -1 , -1 , -1 , -1 , -1 , 0 , 0 , 0 , 1 , 1 , 1 , 2 , 2 , 3 , 3 , 3 , 4 , 4 , 5 , 5 , 5 , 6 , 6 , 7 , 7.

To classify the cells, we then take the barycenters of the cells and we determine if these barycenters should be inside or outside which will give the classification for the cells. To do so, we will shoot rays, look at the parity of the number of intersection, and then conclude, the number of shot being determined by the previously mentioned values. To shoot rays here, we preprocess the triangles to create an octree where we keep track of which triangle cross which cube. Then, we just determine which cells are crossed by the ray shot and we test the intersection between the triangles contained in the crossed cells and the ray. (Most of the geometric predicates used are described in [4])

3.1.2 Using the structure of the mesh

Another way to get the ray information would be to use the structure of the mesh. Indeed, if we can indeed use a classical collision detection like in 3.1.1, we could also take benefit of the mesh structure. To do so, we shoot a ray from the barycenter of cells and we check the intersection with all the facets of the cell. Thus, by taking the connected cell of the facet which is crossing the ray we can find the next cell crossed by the ray etc ... We keep track of all the cells the ray has been through, and when the facet crossed was black. Then, knowing that after the last cell we are supposed to be outside, by switching each time the facet was black, we can determine which cell is outside and which is inside according to this ray (as shown in figure 3).

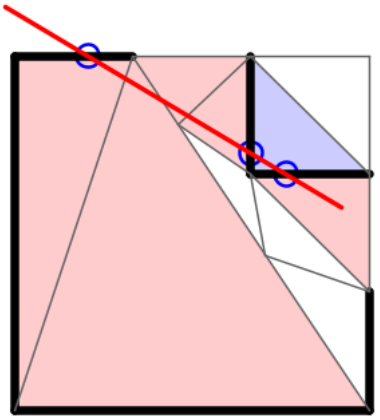


Figure 3: Ray shot from a cell in 2D. The bold lines represent the original mesh, the blue circle the intersection with the black facets, the red triangles get one more Inside while the blue one get one more Outside

Then we maintain two arrays whose indexes are the indexes of the cell: the first one contain the number of ray which concluded Inside for each cell, the second the one who concluded Outside. To know from which cell we should shoot a ray, we are using a structure which will give us in $O(1)$ the cell which has been the less crossed and will update in $O(1)$ the number of ray which has crossed a cell (as the number of ray can only increase by one each time, we can do better than a classical min-heap).

With this method, the bigger and the more central a cell is, the more information we will have about it. Furthermore, we also obtain new information because on the big cells, we will not just have information about the barycenter, but also about other part of the cell.

3.1.3 Using uniformly sampled grids

To accentuate this distribution of the information depending on the size of the cells, shared over all the cells, we can try an approx where the ray are chosen from an outside point of view.

Let's take again our mesh M and the notation of the section 3.1. We also take the definition that for $A, B \in \mathbf{R}^3$, $\overleftrightarrow{A,B}$ is the line $\{A + \lambda.(B-A)/\lambda \in \mathbf{R}\}$ and $\overrightarrow{A,B}$ is the ray $\{A + \lambda.(B-A)/\lambda \geq 0\}$. Let's define the set of all lines of $\mathbf{R}^3$ as $\overleftrightarrow{\mathbf{R}^3}$

Then, for a point $A \in \mathbf{R}^3$,

$$\begin{array}{c} \displaystyle\int_{B \in S(A,R)} \left|M \cap [A,B]\right| \%2 \; dB \; > \frac{1}{2} \int_{B \in S(A,R)} dB \\ \Updownarrow \\ \displaystyle\int_{B \in S(A,R)} \left(2 \times \left|M \cap [A,B]\right| \%2 - 1\right) \; dB \; > 0 \\ \Updownarrow \\ \displaystyle\int_{B \in S(A,R)} \left(2 \times \left|M \cap \overrightarrow{A,B}\right| \%2 - 1\right) \; dB \; > 0 \end{array}$$

We notice then that for every $B \in S(A,1)$, the line $\overleftrightarrow{A,B}$ can give us two information, one with $\overrightarrow{A,B}$ and one with $\overrightarrow{A,-B}$. Let's define

$$\delta_A \; : \; \left(\begin{array}{ccc} \left\{\overleftrightarrow{A,B} / B \in S(A,1)\right\} & \longrightarrow & \{-1,0,1\} \\ \overleftrightarrow{A,B} & \longmapsto & \left|\overrightarrow{A,B} \cap M\right| \%2 + \left|\overrightarrow{A,-B} \cap M\right| \%2 - 1 \end{array} \right) \qquad (1)$$

δ_A is well define as if we choose another representation of the line, then we choose $-B$, and we obtain the same result. And thanks to this function, we can notice now that actually, our heuristic of section 3.1 is equivalent to say that $\displaystyle\int_{\{l \in \overleftrightarrow{\mathbf{R}^3} / A \in l\}} \delta_A(l) dl \; > 0$.

But as we are working on cells, let's try to generalize this notion to cells. Let's assimilate a cell C of M and its geometric representation $C \subset \mathbf{R}^3$ to a convex polyhedron. For a cell C of M, we define, for $l \in \overleftrightarrow{\mathbf{R}^3}$, $\delta_C(l) = \left\{ \begin{array}{ll} 0 & \text{if } l \cap \dot{C} = \emptyset \\ \delta_A(l) & \text{with } A \in l \cap \dot{C} \text{ if } l \cap \dot{C} \neq \emptyset \end{array} \right.$. This definition doesnot depend on the choice of A as $M \cap \dot{C} = \emptyset$ and C is convex (thus, between two points of $\dot{C} \cap l$, there is no input triangle). With this function, we can then define if C should be classify Inside or Outside by $\displaystyle\int_{l \in \overleftrightarrow{\mathbf{R}^3}} \delta_C(l) dl \; < \; 0$.

To compute efficiently that, we are going first of all to use the fact that computing $\delta_C(l)$ for a cell C and a line l, if we do it in the good way, will also give us $\delta_D(l)$ for every cell D. Then, we will discretize $\overleftrightarrow{\mathbf{R}^3}$. There were no point to do it in section 3.1.1, but here discretizing will give us data on the whole cell, not just the barcyenter, and the most important cells will be crossed by more lines, which is better considering their importance. To proceed to this discretization, in order to be able to compute efficiently δ, we are going to discretize the possible direction of the lines, and then for each direction, we will do a regular grid, orthogonal to the direction, each node -associated to the direction- giving a line.

Technically, to perform efficiently this method, we will use algorithm 1 whose big steps are:

1. Rotating the whole structure (representing the choice of direction)
2. Creating an array representing the grid
3. For each facet of the mesh, find which line will cross it, and add the facets to the corresponding cell of the array representing the grid

Algorithm 1 Atomic step of the method using uniformly sample grids

$matrix \leftarrow$ a random rotation matrix of $\mathbf{R}^3$
for *point* **in** *mesh.points* **do**
    $point \leftarrow matrix \times point$ ▷ We rotate the whole mesh by multiplyong every point by $matrix$
end for

for every three axis a **do**
    $(y, z) \leftarrow$ the two other axis than a
    $N \leftarrow \sqrt[3]{number\ of\ cell\ in\ the\ mesh}$
    $max\_y \leftarrow$ the maximum y-coordinate on the mesh
    $min\_y \leftarrow$ the minimum y-coordinate on the mesh
    $max\_z \leftarrow$ the maximum z-coordinate on the mesh
    $min\_z \leftarrow$ the minimum z-coordinate on the mesh
    $interval\_y \leftarrow (max\_y - min\_y)/N$
    $interval\_z \leftarrow (max\_z - min\_z)/N$
    $classification \leftarrow N \times N$ array of empty list
    for $face$ **in** $mesh.faces$ **do**
        Project $face$ on the plane (y, z)
        for all (i, j) such that $(min\_y + i \times interval\_y,\ min\_z + j \times interval\_z)$ is in the projection **do**
            $x \leftarrow$ the a coordinates of the intersection between $face$ and $\mathbf{R} \times (min\_y + i \times interval\_y,\ min\_z + j \times interval\_z)$
            $classification[i][j].add((face, x))$
        end for
    end for

    for i **from** 0 **to** $N - 1$ **do**
        for j **from** 0 **to** $N - 1$ **do**
            Sort $classification[i][j]$ by the second argument
            $actual\_cell \leftarrow outside$
            $is\_inside \leftarrow False$
            for $(face, \_)$ **in** $classification[i][j]$ **do**
                if $face$ is black **then** $is\_inside \leftarrow True$
                end if
                $actual\_cell \leftarrow$ the other cell connected to $face$ than $actual\_cell$
                if is_inside **then**
                    Add one Inside to the counter of $actual\_cell$
                else
                    Add one Outside to the counter of $actual\_cell$
                end if
            end for
            if is_inside **then** ▷ Because at the end we are supposed to be outside
                Cancel all the counter update made in the previous loop
            end if
        end for
    end for
end for

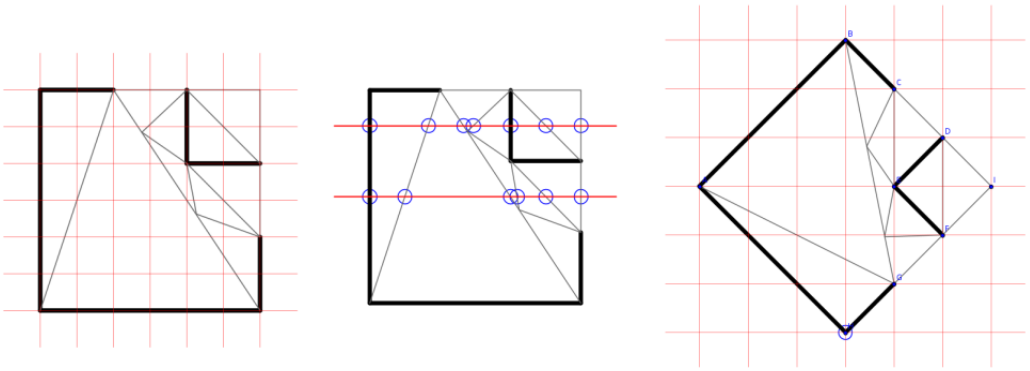


Figure 4: 2D example of the method using a regular grid. From left to right: Example of a grid, each line being in red. Example of two lines, one succeeding, one crossing a hole (and being detected), the blue circles representing the intersection points, which once sorted by their x coordinates will give us which cell is being crossed the first. Example of rotation, with then a new grid, new projection, etc...

4. In each cell of the array, sort the facets by order of crossing
5. Using the mesh structure -the fact that each facet contain the two connected cell-, rebuild the path of the crossed cells, remembering when we crossed a black facet
6. If the line ends saying outside, added the Inside/Outside information for each cell w

The condition on the 6-th steps is important, and come from Nooruddin and Turk [10], because here, we are just taking the first ray of the definition of δ_A 1. The second ray giving a different output, is equivalent to an odd number of intersection between the line and the mesh. Thus, these cases having to count 0, we just don't add the information. Removing this line is equivalent to remove the line crossing an odd number of hole (including 1, and thus removing most of the wrong rays). This method is illustrated with the figure 4

To have a better discretization, and to take benefit of the rotation, whenever we do this rotation, we consider successively as direction both the three axis. Thus, we have three very different directions (which will not be likely to go through the same hole), and we just have to project on the axis (making the projection and then the detection of the right lines much easier). And to keep something linear (we need something linear to have the same order of magnitude of lines as cells, while keeping something reasonably quick), we take a grid of size $\sqrt{3}N \times \sqrt{3}N$ where N is the number of cell in the mesh.

We didn't mention in our survey some safeguard, here to remove wrong information being obtained because the line is tangent to a cell or some rounding errors (which can happen often seeing the number floating operation, and the unability to maintain the robustness of the mesh creation).

3.2 Post processing

Once we have the ray information about the cells, which are more or less at this step two arrays, indexed by the cells, containing respectively the number of Inside and the number of Outside for each cell. Nevertheless, just taking the majority right now may not be enough (it can be with the method 3.1.1 and 3.1.2 whenever the holes are small, but it is not general, and even in this case there may be some irregularities, as show in figure 5).

In all the smoothing, we are going to use a dual graph of our mesh. In this graph, the nodes are the cells, and the edges are faces. We thus have as many nodes as cells, and two

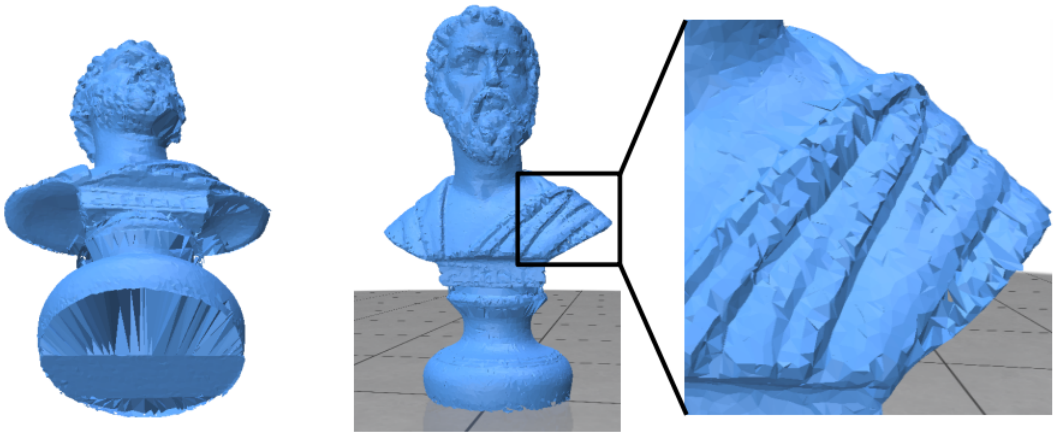


Figure 5: Left: Result we can obtain when we apply the method 3.1.1 or 3.1.2 without smoothing. Right: Result we can obtain when we use uniformly sampled grids without smoothing

nodes are connected if they share a facet. The weight of an edge is the area of the facet shared by the two cells corresponding to the two nodes of the edge, and we add a label to each edge: 1 if the facet is white, -1 is the facet is black. We can note $G = (V, E, W, L)$ such a graph with V the nodes, E the edges, W the weight function and L the labeling function.

3.2.1 Local smoothing using area

A first approach would be to classify the cells using a majority vote and then to check for each cell what is the classification of the connected cells, (where the majority is weighted by the area of the shared facet), and switch if it was not the right one. This amounts to check if changing the classification of this cell will reduce the total added area. Of course, when we compute the majority, if the shared facet is black, then we switch the classification in the calculus (because we consider that it is preferable if both side of a black facet have different classification). More rigorously, if we denote by C the classification ($C(V) = \{-1, 1\}$), for each cell $x \in V$, we switch $C(x)$ if and only if $\sum_{\substack{(x,y)\in E \\ L(x,y)\times C(y) = C(x)}} W(x,y) < \sum_{\substack{(x,y)\in E \\ L(x,y)\times C(y) \neq C(x)}} W(x,y)$.

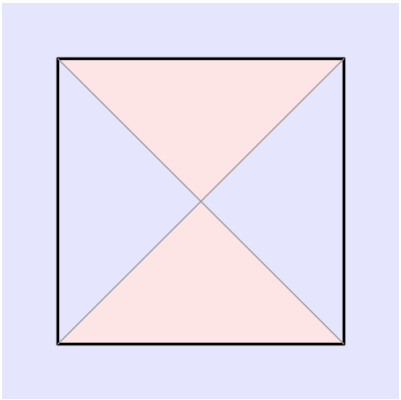


Figure 6: 2D example where the method 3.2.1 will alternate between two states

We then iterate the process until we reach convergence (convergence is not guaranteed, it frequently happens that we alternate between different position (often 2, see figure 6), but we can detect this situation and stop anyway).

3.2.2 Propagating the confidence we have in the classification of a cell

An other approach we can try is to propagate the information. Through this, we will be able to touch the untouched cells and to rectify the cell which has been fooled. To do so, for $x \in V$, let $outside[x]$ (resp. $inside[x]$) be the number of Outside (resp. Inside) we got for the cell corresponding to x. Then our smoothing is:

$$\forall x \in V, \begin{cases} inside[x] = \sum\limits_{\substack{y \in V \\ (x,y) \in E}} inside[y] \times W(x,y) \times \frac{L(x,y)+1}{2} + outside[y] \times W(x,y) \times \frac{1-L(x,y)}{2} \\ outside[x] = \sum\limits_{\substack{y \in V \\ (x,y) \in E}} outside[y] \times W(x,y) \times \frac{L(x,y)+1}{2} + inside[y] \times W(x,y) \times \frac{1-L(x,y)}{2} \end{cases}$$

doing all of it simultaneously and we iterate the procedure, until we are not far from reaching convergence.

a geometrical interpretation could be added here to explain in which way it is just like spreading things from the black facets and then cutting where both side of black facets influence is equal

3.2.3 Using a min-cut algorithm

Another approx would be to try to directly close with the surface maximizing the area of black facets we take, and minimizing the area of white facet. And here, we exactly end up on the definition of a min-cut problem. Indeed, we want a subset $I \subset V$ minimizing the value $\sum\limits_{\substack{x \in I, y \notin I \\ (x,y) \in E}} W(x,y) \times L(x,y)$ where I contains the internal cells, which is exactly the Min-cut problem with as weight function $W \times L$. In this case the black facets have the same weight than the white ones (but opposite), we can handle that by replacing the value -1 of L by $-M$ where we can choose M to decide the importance of keeping the black facets in the final surface (and ∞ if we want to be sure to keep them).

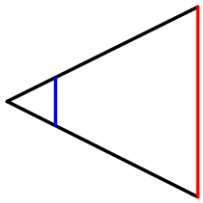


Figure 7: In black the input segment (because it is in 2D), in red the expected way to close it and in blue a better way to close it if we do a min-cut algorithm just removing the edges corresponding to black facets

Nevertheless, if the min-cut problem can be easy to resolve, it is when all the weight are positive. Whenever the weight can be negative, we have something equivalent to the max-cut problem, which is NP-complete. What we could do then is just to remove all edges connecting black facets, to enforce the cut to use them. Nevertheless, it still doesn't work because with that we are not forcing both cells -from each sie of a black facets- to be classified differently, and then the optimal solution is to draw a minimal surface between few black facets (see figure 7).

To solve this issue, we can use a slightly different problem. We can add weight to each cell according to their classification. We are now allowed to use a function $A : V \times \{0,1\} \to \mathbf{R}$ and the minimizing problem becomes to minimize $\sum\limits_{\substack{x \in I, y \notin I \\ (x,y) \in E}} W(x,y) \times \frac{L(x,y)+1}{2} + \sum\limits_{x \in I} A(x,1) + \sum\limits_{x \notin I} A(x,0)$ and thanks to the wildly used graph-cut works of Boykov *et al.* [2] [1] and Kolmogorov *et al.* [8], this can be proceeded very efficiently.

Diazzi and Attene [4] -which we take the classification from- are using the orientation of the facets to set up A. Indeed, if the mesh is

oriented, every black facet is giving information about the connected cells. The cell below a black facet is supposed to be inside while the cell above is supposed to be outside. Thus, for $x \in V$, we can take

$$A(x,1) = \sum_{\substack{f \text{ facet of } x \\ x \text{ is above } f}} \text{area of } f \qquad A(x,0) = \sum_{\substack{f \text{ facet of } x \\ x \text{ is below } f}} \text{area of } f$$

But in our case, we don't have access to such information. Thus, we are going to use the one we computed in the first part. What we would want to do, is to have $A(x,0) = inside[x]$ and $A(x,1) = outside[x]$. Nevertheless, the sum to minimize wouldn't be homogeneous. We have then two options. First, in the general case, we can set

$$A(x,0) = \frac{inside[x]}{inside[x] + outside[x]} \times \text{area of all facets of } x$$
$$A(x,1) = \frac{outside[x]}{inside[x] + outside[x]} \times \text{area of all facets of } x$$

Otherwise, if in the first step we are using uniformly sampled grids 3.1.3, we can associate to each ray the area of the surface it was representing on the grid (with the notation of algorithm 1, it is $interval\_y \times interval\_z$) -or the mean of this surface on all ray, if we want an equal weight for each-. Anyway, both of this method are supposed to -and are- giving similar result.

4 Results

4.1 The most efficient method

4.1.1 The choice of the method for getting information

Except for really small meshes, both the two first methods are becoming way too slow. Indeed, the geometric predicates being more time consuming than in the method using uniformly sampled grid, it becomes really longer (can be 100 times longer for big mesh).

Number of triangles	method 3.1.1	method 3.1.2	method 3.1.3
10 000	12s	15s	2s
30 000	375s	589s	9s

4.1.2 The choice of the smoothing

Whenever it comes to the choice of the smoothing, the time doesn't matter as working directly on the graph doesn't have almost any cost compare to all the geometric predicates that are to be made before.

1. The first method of smoothing (3.2.1) may has problems because we are working only on the neighbors of a cell and then sometimes many cells are required to switch as the same time. So it doesn't work perfectly. Furthermore, the surface it creates is not guaranteed to be smooth as figure 8 b) is showing. Nevertheless, on most of the case, or whenever the holes are small enough, it works quite well. It is not the best but it works.

2. The second smoothing method (3.2.2) is also quite efficient in most of the case, nevertheless it often creates small default, and in some rare cases, but cases that happen, it can produce mesh without sense, just like in figure 8 c). Indeed, if with the other

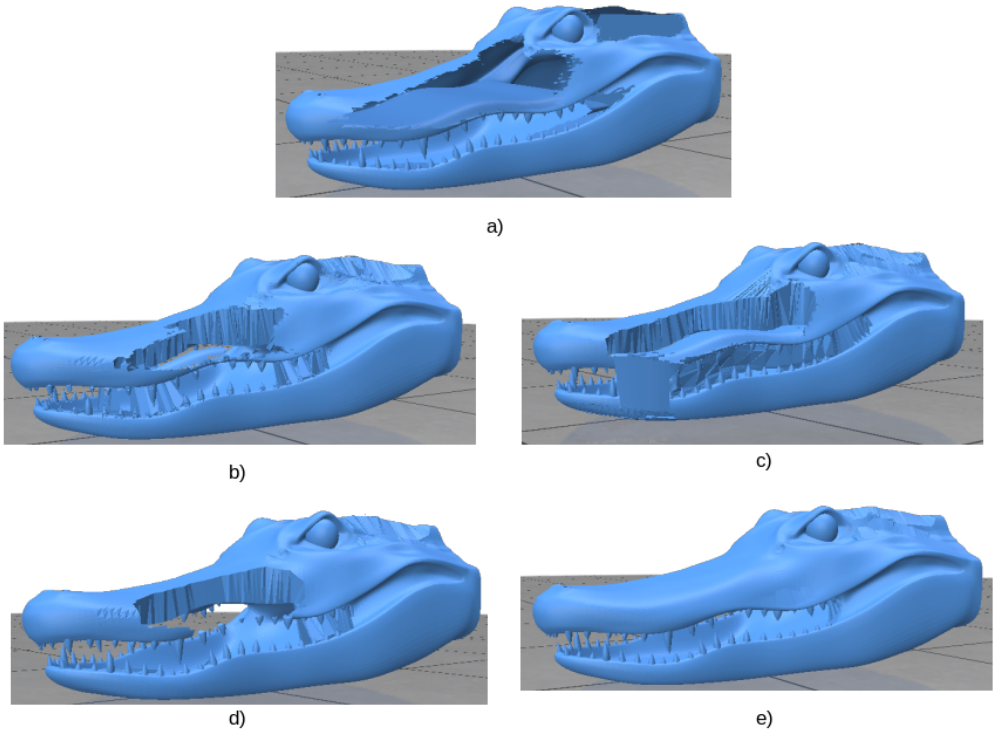


Figure 8: Comparison of result of the different smoothing on one of the worst case for our method
a) The original mesh with holes
b) Completed with the first smoothing method with the local area minimization3.2.1
c) Completed with the second smoothing method using propagation 3.2.2
d) Completed using our third method using a min-cut algorithm 3.2.3
e) Completed using the method with orientation

Type of holes	Proportion of better result with our method	Mean volume error for our method	Mean volume error using orientation	Median volume error for our method	Median volume error using orientation
15 holes representing 2% of the surface	32%	2.2%	0.54%	0.29%	0.12%
3 holes representing 3% of the surface	27%	3.3%	1.9%	1.5%	0.44%
5 holes representing 8% of the surface	20%	7.6%	4.7%	5.6%	2.8%
20 holes representing 15% of the surface	21%	23%	12%	11%	5.2%

Table 1: Comparison of the method of this work (using uniformly sampled grid and min-cut algorithm) and of the work we build from (using orientation)
The error is computed relatively to the original mesh

method it is rare to have a facet which will end up wrongly oriented -what is most likely to happen is that they may disappear-, with this method in can happen, producing real strange mesh. Of course it is quite rare, but it may happen, specifically when we have several layer of mesh close one to the other.

3. The last smoothing using the min-cut algorithm seems to always produce result which have sense. Even in the worst cases (figure 8), the output is not the one expected but it still have sense. Furthermore, it creates smooth surfaces, specifically compare to the other methods.

Thus, we are going to evaluate more specifically the method were we get ray information using uniformly sampled grid and where we smooth using a min-cut algorithm.

4.2 Quality of results

In most of the cases, the final result looks good as figure 9 is showing. Even in the problematic cases, where we are recovering something else than the initial mesh, the result remains pretty smooth (as shown in the last line of figure 9, where we have two aligned holes, and thus a big part is removed, outputting something which has sense but really different from the input).

In table 1, we compare our method to previous one, using orientation. The comparison is made on the well oriented closed surface of the database Thingi10k. To do the comparison, we take the close surface, and we create hole by selecting a random triangle and then by progressively extending the hole until we reach a certain limit, limit which is define by the total number of hole we want and the total percentage of area we want to remove. Then, we calculate the difference of volume defined by the original mesh and the redefined volume (in percentage of the initial total volume).

This result are showing that if with this kind of comparison, the method using orientation is better (which is logical as it uses more information), the result are in the same magnitude (around 25% of the volume defined without orientation are better than when defined with orientation). The remaining difference is not always a sign of bad quality. Indeed, we based our method on the fact that we want to close the surface by adding the as less area as possible (we are not trying to guess what should be the volume). Whenever we lose too many information with the holes, having something closer from the original mesh is not always better, and can appear a bit stranger. It is for example the case in the first line of figure 10 where an additional face, which was in the original mesh is kept because of the weight brought by the orientation, but as the holes changed the topology, it looks stranger

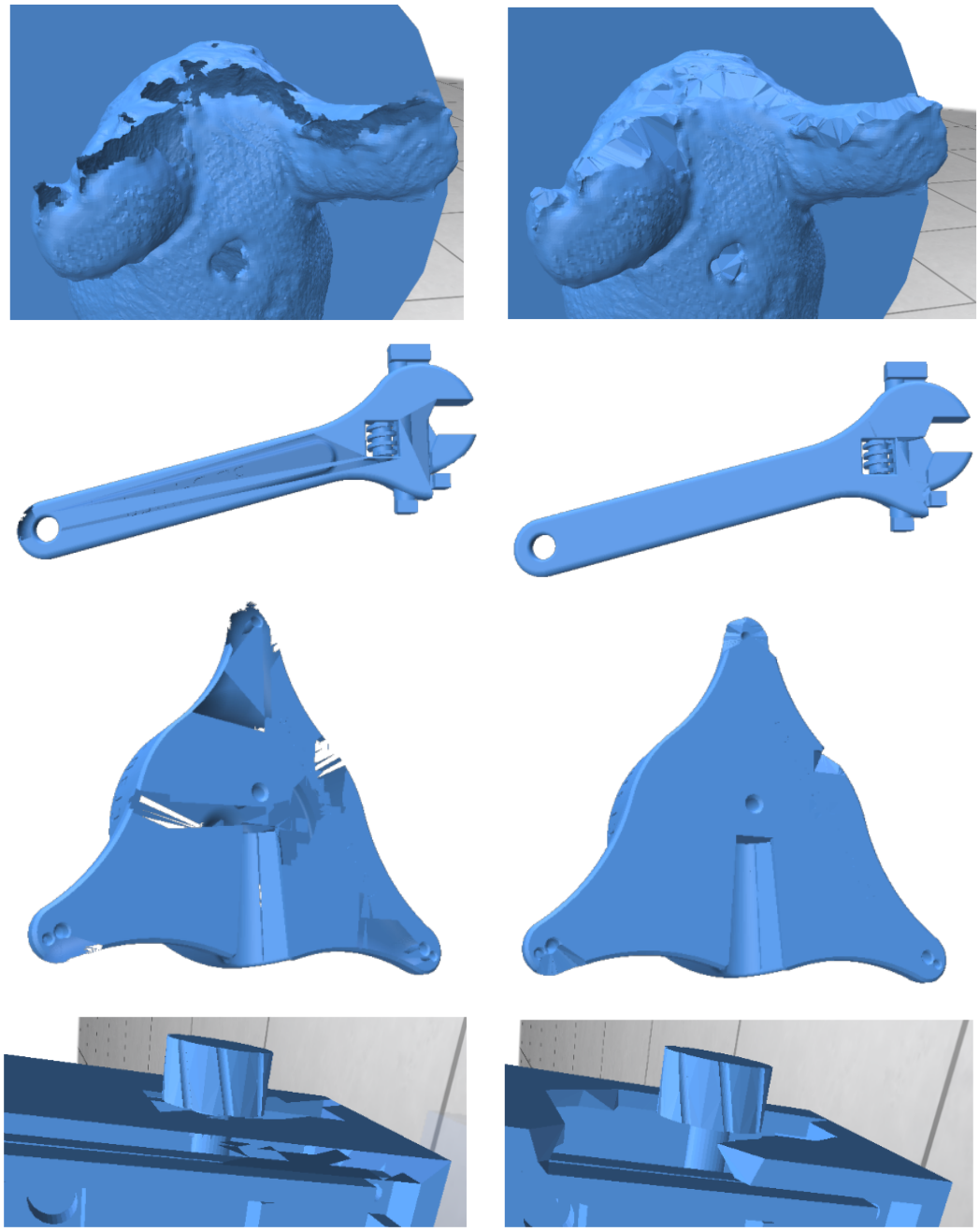


Figure 9: With at the left the original mesh with holes, and at the right the reconstructed mesh with our method

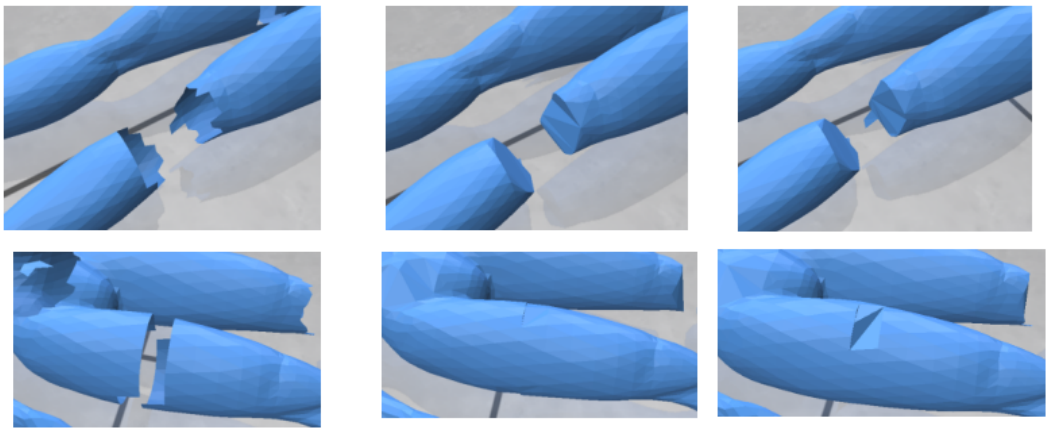


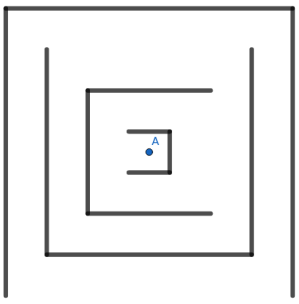
Figure 10: Comparison of the closure of a mesh. Left: The holed mesh. Middle: The reconstructed mesh with our method without orientation. Right: The reconstructed mesh using orientation

-even if closer to the original volume-.

It can also happen that the mesh is simply better at all. It is not common but we have an example in the second line of figure 10.

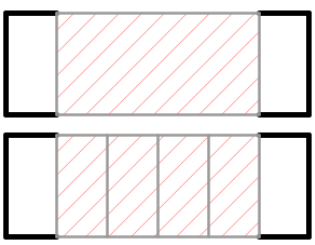
Of course as our method is computing geometric predicates in addition to the min-cut algorithm and the subdivision, it takes more time. Nevertheless, once again we are on the same order of magnitude (being in average 5 times longer -counting in both cases both the subdivision and the classification-). We can speed this up by reducing the number of direction we take (we usually perform 5 rotations, and thus we take 15 directions) or the size of the grid. But if we don't care about the time, we can also increase them to increase the quality of the classification.

5 Potential issues



The idea describe at the section 3.1 being an heuristic, sometimes it fails. In some specific examples, it can be fully wrong, like for example in figure inset where whatever the direction we take, from A, we will cross an hole. It also often fails near from hoels. When a hole is to be closed with a plane surface, if we are close from this plane, both area can be similar (see figure 2 d)). Then, an other hole further away could interfere. If these issues are most of the times not concerning enough issues, it can become problematic when the mesh has the shape of a membrane because then the problem showed in the figure 2 d) occurs in the whole cells if we are near a hole (because the cells are thin due to the membrane topology). This issue explains the result of the figure 8 and of the last line of the figure 9.

Our method is general and does not rely on the way the space subdivision has been done (as long as the cells are convex). Thus, depending on the purpose, we can take another subdivision of the space (to have a more or less refined mesh, or to make it fit in parallel to another mesh, etc...). Neverthe-



less, in our method, the mesh can change the topology and not only because it would not be precise enough. For example, in the example beside, the more we subdivide the hatched area, the more the sum of the length (area in 3D) of the segment (facets in 3D) of cell inside this hatched area will be high. Thus, as in our smoothing the weight of classifying a cell is proportional to the area of its facets, the more we subdivide a region of the space, the more it will have importance compared to other region, but also compared to the area minimization (i.e. with the notation of the section 3.2.3, A will take a bigger weight than W).

In addition, variation are to be noticed. It rarely changes the topology but it can create significant switching (even if, in good meshes, the output is alwyas good, and thus stable). Nevertheless, this problem can be partly tackle by increasing the precision of the sampling of ray (increasing the number of direction and the precision of the grid).

6 Conclusion

We manage to provide a classification of a polyhedron subdivision of the space respecting an input triangular mesh, which does not rely on the orientation of the facets. We tried different method to be able to select the right one. This method runs in a reasonable time and outputs satisfying volumes. The mission is thus fulfilled.

However, this method cannot be fully trusted as some issues may occur. If they are not that problematic, it makes a bit harder the result to be predicted and thus depending on the purpose, we may not rely on it. This method is more easily removing triangle from the initial mesh than expected, which is a problem when we fully trust the input (if we are sure that the all the input triangle are good, just some are missing) but can become an advantage when there are some mistakes.

One could also consider that not enough C++ tricks have been used and thus the code could become faster, to reach something as long as the subdivision, or to be able to improve the quality by increasing the size of the sampling.

I want to thank all the IMATI lab in which I have been making my internship for their welcome, which has been great, and my supervisor Marco Attene for his availability, his smile and the fact that he was never short of valuable advice.

References

[1] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1124–1137, 2004.

[2] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001.

[3] James Davis, Stephen R Marschner, Matt Garr, and Marc Levoy. Filling holes in complex surfaces using volumetric diffusion. In *Proceedings. First international symposium on 3d data processing visualization and transmission*, pages 428–441. IEEE, 2002.

[4] Lorenzo Diazzi and Marco Attene. Convex polyhedral meshing for robust solid modeling. *CoRR*, abs/2109.14434, 2021.

[5] Chao Feng, Jin Liang, Maodong Ren, Gen Qiao, Wang Lu, and Shifan Liu. A fast hole-filling method for triangular mesh in additive repair. *Applied Sciences*, 10(3), 2020.

[6] Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)*, 32(4):1–12, 2013.

[7] Tao Ju. Fixing geometric errors on polygonal models: A survey. *Journal of Computer Science and Technology*, 24(1):19–29, 2009.

[8] Vladimir Kolmogorov and Ramin Zabin. What energy functions can be minimized via graph cuts? *IEEE transactions on pattern analysis and machine intelligence*, 26(2):147–159, 2004.

[9] Peter Liepa. Filling holes in meshes. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 200–205, 2003.

[10] Fakir S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, 2003.

[11] Emiliano Pérez, Santiago Salamanca, Pilar Merchán, and Antonio Adán. A comparison of hole-filling methods in 3d. *International Journal of Applied Mathematics and Computer Science*, 26(4):885–903, 2016.

[12] Joshua Podolak and Szymon Rusinkiewicz. Atomic volumes for mesh completion. In *Symposium on Geometry Processing*, volume 10. Citeseer, 2005.

[13] Kenshi Takayama, Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. A simple method for correcting facet orientations in polygon meshes based on ray casting. *Journal of Computer Graphics Techniques*, 3(4):53, 2014.

[14] Lavanya Sita Tekumalla and Elaine Cohen. A hole-filling algorithm for triangular meshes. *School of Computing, University of Utah, UUCS-04-019, UT, USA*, 2, 2004.

[15] Wei Zhao, Shuming Gao, and Hongwei Lin. A robust hole-filling algorithm for triangular mesh. *The Visual Computer*, 23(12):987–997, 2007.

[16] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016.