Graph-based models of volumetric medical data and applications to topological correction

Quentin Fortier, ENS Lyon

Supervisors: Vincent Barra[1] , Jean-Marie Favreau[1], Marco Attene[2].

Abstract

After stating some results about surface cutting, this report gives an algorithm to "optimally" cut a volume, based on pants decomposition, and several variants. It introduces two new algorithms, the neighborhood algorithm and the single pant algorithm, and a C++ code named OC3D available[3] under GPL v3, which are my main contributions.

1 Introduction

The reader not familiar with basic algebraic topology will find in the appendix the definitions needed in this report.

1.1 Overview

The limited resolution or the noise may increase the genus of the surfaces given by medical images and we want to remove them, both to have a more accurate image and apply algorithms that need genus 0 surfaces. For that purpose, important works have been done in finding the shortest system of loops cutting reducing the genus to 0: several methods were proposed, for example in [1] or [2]. The goal of this internship is to extend these methods to a volumic topological correction, in which only few works have been done.

1.2 Modelling

For more precision, we first need some definitions:

Definition 1. A *volume* $\mathbb{V}$ (resp. *surface*) is a 3(resp. 2)-manifold embedded in $\mathbb{R}^3$. We assume a volume has one boundary $\partial\mathbb{V}$ and the genus g of $\mathbb{V}$ (i.e the genus of $\partial\mathbb{V}$) is assumed to be different from 0.

From an algorithmic point of view, instead of an abstract 3-manifold we deal with either a 3-mesh (using tetrahedra) or voxels, which are a volume in the

[1]Univ. Blaise Pascal - LIMOS - UMR 6158 Campus Scientifique des Cézeaux, 63177 Aubière Cedex, France, e-mail: {Jean-Marie.Favreau,Vincent.Barra }@isima.fr

[2]Istituto per la Matematica Applicata e le Tecnologie Informatiche, Consiglio Nazionale delle Ricerche, Via De Marini 6, 16149 Genova, Italy, e-mail: attene@ge.imati.cnr.it

[3]Source code, examples, documentation... can be found at http://perso.ens-lyon.fr/quentin.fortier/OC3D/OC3D.html

sense of the above definition. We refer to them as discrete volume. Discrete surfaces are either sets of triangles or of faces of a voxel set.
Moreover, we assume these discrete volumes or surfaces to have a weight on their faces or edges (typically, this weight is the length for an edge and the area for a face or the probability that the two volumic elements are adjacent, given by the segmentation step).

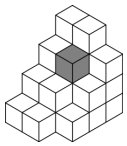

Figure 1: Voxels

Definition 2. Given a discrete volume $\mathbb{V}$, the *dual graph* $G^* = (\mathrm{V}, \mathrm{E})$ of $\mathbb{V}$ is built the following way:

- The vertices of G^* are the volumic elements (tetrahedra or voxels).
- Two vertices are connected if the corresponding two elementary volumes share a face, and the weight of this edge is the weight of the face.

Two edges of G^* are *adjacent* if the corresponding faces share an edge of $\mathbb{V}$.

We define similarly the dual graph of $\mathbb{S}$: we replace volumic elements by faces of the surface, and two vertices are connected if the corresponding faces share an edge.
Since every vertex in G^* has a degree at most d, with $\mathrm{d} = 6$ if $\mathbb{V}$ is a voxel set and $\mathrm{d} = 4$ if $\mathbb{V}$ is a 3-mesh, $2\,\mathrm{E} = \sum deg(v) \leqslant \mathrm{d}\,\mathrm{V}$, so $\mathrm{O(E)} = \mathrm{O(V)}$.

Definition 3. Let A, B be two sets such that $\mathrm{A} \subset \mathrm{B}$ and B is connected.
A is B-*non-separating* if B - A is connected.
Similarly, if $A_i \subset \mathrm{B}$, $\{A_1, \ldots, A_n\}$ is B-non-separating if B - $\bigcup A_i$ is connected.

Definition 4. A *1-cut* γ (on a surface $\mathbb{S}$) is a $\mathbb{S}$-non-separating loop.

On the dual of $\mathbb{S}$, a 1-cut is a simple cycle of edges.

Definition 5. A *2-cut* $\mathcal{C}$ (in $\mathbb{V}$) is a connected, $\mathbb{V}$-non-separating, genus-0 surface such that $\partial\mathcal{C}$ lies on $\partial\mathbb{V}$.

On G^*, a 2-cut is a set of adjacent edges (however it is not sufficient, since we require the set to be a manifold).

Definition 6. The *cutting product* of a surface $\mathbb{S}$ along a 1-cut γ (written ✂($\mathbb{S}$, γ)) is, informally, the surface produced by duplicating every edge and vertex and removing the adjacences between faces sharing an edge in γ.
The informal definition is similar for a volume and a 2-cut.
See [1], p. 35, for a more detailed definition.

Definition 7. A *system of* 2(resp. 1)-*cuts* of $\mathbb{V}$ (resp. $\mathbb{S}$) is a $\mathbb{V}$(resp. $\mathbb{S}$)-non-separating set of 2(resp. 1)-cuts such that the resulting cutting product has genus 0 and every pair of cuts doesn't intersect.

We can now give the main issue of this report:

Problem 1 (2-MIN-CUTTING)**.** *Given a discrete volume* $\mathbb{V}$*, find a minimum system of 2-cuts.*

We are also interested in the related problem on surface, which we investigate first:

Problem 2 (1-MIN-CUTTING)**.** *Given a discrete surface* $\mathbb{S}$*, find a minimum system of 1-cuts.*

These minimums exist since there are finitely many sets of g 1-cuts (resp. 2-cuts).

1.3 NP-hardness

An apparently similar problem is known to be NP-hard:

Problem 3 (1-MIN-CUTTING-BOUNDARY)**.** *Given a discrete surface* $\mathbb{S}$*, possibly with boundary, find a minimum cutting to obtain a single topological disk (i.e the resulting cutting product must have genus 0 and no boundary).*

Theorem 1. *1-MIN-CUTTING-BOUNDARY is NP-hard*

Proof:
[6] reduces this problem to the *rectilinear Steiner tree problem* (i.e, given n points in the plane, find a tree connecting them with only vertical and horizontal line segments), which is known to be NP-hard.

□

However, it is not known if 1-MIN-CUTTING and 2-MIN-CUTTING are NP-hard, but the two problems are linked.
In the case of voxels, we have:

Theorem 2. *1-MIN-CUTTING reduces to 2-MIN-CUTTING.*

Proof:
Let $\mathbb{S}$ be an instance of 1-MIN-CUTTING: according to Jordan's theorem, $\mathbb{S}$ bounds only one volume $\mathbb{V}$.
On each face of $\mathbb{V}$, we put a weight equals to the sum of the weights of the edges of this face that are in $\mathbb{S}$. This way the area of a 2-cut of $\mathbb{V}$ equals its perimeter. The size of $\mathbb{V}$ is polynomial in the size of $\mathbb{S}$ according to the isoperimetric inequality: $36\ \pi\ \mathrm{vol}(\mathbb{V})^2 \leqslant \mathrm{area}(\mathbb{S})^3$, and since the volume of $\mathbb{V}$ equals the number of voxels in $\mathbb{V}$ and the area of $\mathbb{S}$ equals the number of faces (squares) in $\mathbb{S}$.

□

The same theorem holds for tetrahedra if we restrict the problem to PLCs (*Piecewise Linear Complex*) which can be tetrahedrized (not all surfaces can be tetrahedrized).

1.4 Relations between the two problems

Although they have similarities, 1-MIN-CUTTING and 2-MIN-CUTTING are not identicals and especially, minimizing the area of a surface is not equivalent to minimizing its perimeter. Moreover, a surface with bounded area may have a perimeter equals to infinity, for example the Koch snowflake.

2 Previous works on surface cutting

Let $\mathbb{S}$ be a surface with genus g.
This section introduces two algorithms dealing with **surface cutting**, the first described in [1], chap. 3 and the second in [2], chap. 3. Both uses a similar approach to find 1-cuts: they first show that computing a 1-cut is easy if we add constraints to it and then they split the surface, to force the wanted constraints.

2.1 Cutting a surface to reduce its genus to 0

Using the following approach, we can get a good approximation of a minimum system of 1-cuts of $\mathbb{S}$.

Lemma 1. We can calculate the shortest 1-cut on $\mathbb{S}$ passing through a fixed point p in O(Vlog(V)).

Proof:
We use Dijkstra's algorithm from p to keep a growing surface. As soon as we meet a vertex v previously visited, we deduce a loop built with the two shortest paths to v and we look at the number c of connected components of the complementary of this loop: if c equals two we continue our search, otherwise (c equals one) we found a shortest 1-cut from p.

□

Lemma 2. We can compute a first system of 1-cut $\mathcal{Y}$ on $\mathbb{S}$ in O(V).

Proof:
Let t be a volumic element (triangle). We use the *cut locus* associated to t, i.e the set of all points having more than one shortest path from t: we take as $\mathcal{Y}$ the non arborescent part of this cut locus, called the *reduced* cut locus.

□

Lemma 3. Every 1-cut of $\mathbb{S}$ crosses $\mathcal{Y}$.

Proof:
If a 1-cut doesn't cross $\mathcal{Y}$, it is included in ✂$(\mathbb{S}, \mathcal{Y})$ which is a genus 0 surface. Every loop on such a surface is contractible so separating, a contradiction.

□

By calculating the shortest 1-cut passing through every triangle of $\mathcal{Y}$, we have:

Theorem 3. *We can compute a shortest 1-cut in O($\mathcal{Y}$ Vlog(V)).*

The algorithm consists in iteratively calculating the shortest 1-cut, while possible. This may not lead to the minimum system of 1-cuts.

2.2 Shortest homotopic cycles on a surface

This second algorithm uses *n-pants* (a genus-0 surface with n boundaries).

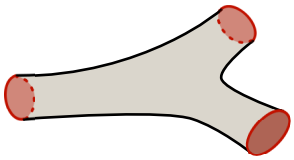

Figure 2: A 3-pant

Theorem 4. *In a 3-pant, we can compute the shortest 1-cut homotopic to a boundary of the pant in O(Vlog(V)).*

Proof:
To establish this result, we need (see proof in [4])

Lemma 4. In a cylinder, we can compute the shortest 1-cut homotopic to a boundary of the cylinder in O(Vlog(V))

We can use this lemma on a 3-pant by cutting along a shortest path from the two remaining boundaries.

□

In [3], one can find the following results:

Lemma 5. Every compact connex orientable surface unless the sphere, disk, cylinder and torus has a 3-pants decomposition (a set of disjoint 1-cuts cutting $\mathbb{S}$ into 3-pants).

Lemma 6. On such a surface, we can compute a first 3-pants decomposition $\mathbb{P}_0$ in O(gV).

Then we can give the pants optimization algorithm: compute a 3-pants decomposition and optimize locally every boundary of a pant, until no improvement is possible. [3] gives an upper bound for the number of iterations of this algorithm: the (difficult) proof uses rewriting on *crossing words* (words encoding crosses between curves). In particular, this algorithm is polynomial in its input.
[3] also proves that this algorithm is optimal, in this sense: each cut in the resulting pants decomposition $\mathbb{P}$ is the shortest cut homotopic to the initial corresponding cut in $\mathbb{P}_0$. However this algorithm doesn't solve 1-MIN-CUTTING: the homotopy classes of $\mathbb{P}$ are not, *a priori*, equal to the homotopy classes of a minimum system of 1-cuts (it is worth remembering that homotopy classes can't change while optimizing, with this algorithm).

2.3 First attempts to do a volume cutting

A first idea, suggested before the internship, is to put information on the surface.

Definition 8. The *medial axis* (or *skeleton*) of $\mathbb{V}$ is the set of all points (not in $\partial\mathbb{V}$) having more than one closest point to $\partial\mathbb{V}$, i.e the centers of the open balls tangent to $\partial\mathbb{V}$ in at least two points.

Definition 9. The *local feature size* of a point on $\partial\mathbb{V}$ is its distance to the medial axis.

The method consists in defining the length of an edge on $\partial\mathbb{V}$ as the product of the local feature sizes of the adjacent vertices and the Euclidean length of this edge. It is an estimation of the area that is introduced if we select this edge in a 1-cut. Then we can use a method from the previous section to compute an approximation of the shortest system of 1-cuts on $\partial\mathbb{V}$. However this doesn't use all the volume information and incoherent cuttings may happen.

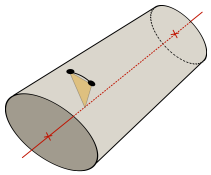

Figure 3: Length defined by the local feature size

Another idea, inspired by the algorithm of section 2.2, is to decompose a volume into volumic 3-pants and to optimize them. It is the main idea of the approach described next.

3 General algorithm

3.1 Overview

The generic algorithm is the following:

- Compute a first volumic pants decomposition.
- Optimize (i.e change the location while decreasing the weight) every boundary of the pants while possible.
- Among the resulting pants decomposition, select a system of 2-cuts.

Two algorithms result from this pattern, which I call *multi pants (MPA)* and *single pant (SPA) algorithms*.

3.2 First decomposition

To compute a first cutting, I already described the cut locus method: given a set of points A, the *A-cut locus* is the set of all points having more than one shortest path from A.
In the SPA, we simply choose an arbitrary point p, compute the reduced p-cut locus $\mathcal{Y}$ and take as an initial decomposition the n-pant which boundaries are the g connected components of $\mathcal{Y}$.

However, one may try to act more *locally* and to use *topological information* to guide the choice of the cuts (i.e to have an initial set of cuts "close" to the optimal) by decomposing the volume into many pants. For that purpose, the MPA first compute the skeleton $\mathcal{K}$ of the surface, using for example an *erosion* process (see [7] for a comprehensive survey of skeleton computation).

Figure 4: Left: Skeleton. Right: Cuts associated to this skeleton.

Let B be the set of all intersection points of $\mathcal{K}$: the MPA basically computes the reduced B-cut locus and uses its connected components as a pants decomposition.
However, we can have intersections in this reduced cut locus, and we are not sure to get 2-cuts: Marco Attene and Jean-Marie Favreau suggested a method to avoid such not manifold cuts, but it is rather complicated and I didn't have the time to investigate it further.

3.3 Pant(s) optimization

I first discuss how to optimize a single 2-cut, the next subsection will deal with the problem of selecting a cut to optimize.
We need to distinguish one kind of 2-cut:

Definition 10. A 2-cut $\mathcal{C}$ is an auto 2-cut if the two pants adjacent to $\mathcal{C}$ are actually the same pant.

Since the SPA deals with only one pant, every 2-cut is auto, and I first describe the optimization of an auto 2-cut $\mathcal{C}$, associated to a pant $\mathbb{P}$.

3.3.1 Optimization of an auto 2-cut $\mathcal{C}$

We first build the network R from G^* the following way:

- Add two vertices s and t to G^*.
- Link s to every vertex adjacent to $\mathcal{C}$ on one side and link the vertices adjacent to $\mathcal{C}$ on the other side to t.
- For each boundary $\mathcal{C}$' of $\mathbb{P}$ delete from G^* the edges in $\mathcal{C}$', and, if $\mathcal{C}' \neq \mathcal{C}$, delete the edges with an end point adjacent to $\mathcal{C}$'.

The second part of this last point is needed in order to avoid intersections between 2-cuts.

Then, we compute a max flow on R (using Ford Fulkerson algorithm, for example) and we use the min cut - max flow theorem to deduce a *min $\mathcal{C}$ s-t cut* (i.e, a min cut obtained by computing a max flow to optimize $\mathcal{C}$) $\mathcal{C}_{min}$. However, $\mathcal{C}_{min}$ is not necessarily a 2-cut (for example $\partial\mathcal{C}_{min}$ is not necessarily connected) but in most applications we hope that $\mathcal{C}_{min}$ is a 2-cut, and, with some caution we will see after, we assume that every min cut computed this way is a 2-cut.
It is easy to see that, throughout the SPA, the pant structure is kept (($\mathcal{Y}$- $\mathcal{C}$) $\cup$ $\mathcal{C}_{min}$ is a system of 2-cuts).
Moreover, the result of the SPA is likely to be, in practice, independant from the initial set of cuts (contrary to the MPA: see section 3.3.2).
However, it is possible that the best cut $\mathcal{C}$' intersects $\mathcal{C}$, and a max flow will not take it into account (both in the SPA and the MPA). To avoid this problem, I suggested to take a min $\mathcal{C}_{min}$ s-t cut as the new cut replacing $\mathcal{C}$ (so we do two consecutive max flows), which correctness is validated by the theorem:

Theorem 5. *$\mathcal{C}$' doesn't intersect $\mathcal{C}_{min}$.*

Proof:

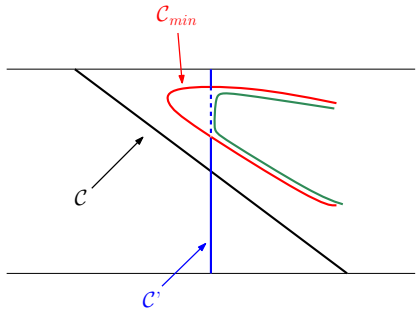

Figure 5: Illustration of the proof by contradiction.

Assume $\mathcal{C}$' intersects $\mathcal{C}_{min}$.
Then we can find a shortest $\mathcal{C}$ s-t cut (in green in figure 5) that doesn't intersect $\mathcal{C}$ by following $\mathcal{C}_{min}$, then $\mathcal{C}$' when $\mathcal{C}$' and $\mathcal{C}_{min}$ intersect.
This contradicts the fact that $\mathcal{C}_{min}$ is a min $\mathcal{C}$ s-t cut.

□

3.3.2 Optimization of a 2-cut $\mathcal{C}$ (not auto)

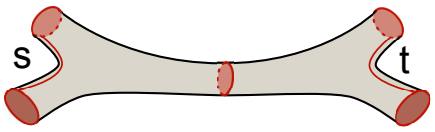

Figure 6: Optimization of a cut

Now we assume that $\mathcal{C}$ is adjacent to two different pants $\mathbb{P}_1$ and $\mathbb{P}_2$, let $\mathcal{C}_{i,j}$ be the boundaries of $\mathbb{P}_i$ different from $\mathcal{C}$. We need to be careful since we want to avoid disconnected min cuts.
We build R from G^* the following way:

- Find a shortest tree $\mathcal{T}_1$ (resp. $\mathcal{T}_2$) connecting $\bigcup \mathcal{C}_{1,j}$ (resp. $\bigcup \mathcal{C}_{2,j}$).
- For all j, link s to $\mathcal{C}_{1,j}$ and $\mathcal{T}_1$ and link $\mathcal{C}_{2,j}$ and $\mathcal{T}_2$ to t.

We need to use the trees $\mathcal{T}_i$ to avoid disconnected min cuts (a min $\mathcal{C}$ s-t cut can't intersect one of these trees). This way, every $\mathcal{C}$ s-t cut is homotopic to $\mathcal{C}$: **we don't change homotopy classes**, contrary to an optimization of an auto 2-cut.
The search of the trees is a problem known as the Steiner tree problem, which is NP complete: in pratice we may prefer to do an approximation of it (for example, in the C++ code, I compute $\mathcal{T}_1$ by iteratively selecting the closest cut (among the $\mathcal{C}_{1,j}$) adjacent to the current cut and linking them), we just want a tree that will not restrict too much the choice of the min $\mathcal{C}$ s-t cut.
However, we can't completely avoid the restriction of the homotopy classes: if $\mathbb{P}_1$ and $\mathbb{P}_2$ have respectively n and m borders, there are (n-1)(m-1) homotopy classes unreachable (i.e, that can't be in the resulting cutting of the MPA), as shown on figure 7. It means that the total weight of the resulting cutting of the MPA depends crucially on the initial cutting.

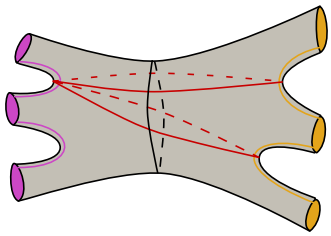

Figure 7: Some 2-cuts (red) unreachable during the MPA (since every cut optimization doesn't change homotopy classes)

3.4 Processing choice

Since every optimization of a 2-cut decreases the weight of the 2-cut system $\mathcal{Y}$, and as there are only finitely many possible total weights (for a discrete volume), our algorithms are *terminating*.
However, the final weight of $\mathcal{Y}$ and the complexity of these algorithms may depend on the order we choose cuts to optimize.
Several possible choices were considered:

- Randomly.
- Maximum weighted 2-cut first.
- (All-Min) I also suggested, especially in the SPA, to optimize the 2-cut $\mathcal{C}$ which lead to the minimum possible $\mathcal{C}_{min}$, and to never optimize $\mathcal{C}_{min}$ later.

Bounding the number of iterations required for the two first points is likely to be very difficult (personal communication with Éric Colin de Verdière).
However the All-Min choice requires $\Omega(g^2)$ max flow, with the SPA: at the beginning there are g 2-cuts to optimize, so we compute g max flow, then we fix one 2-cut so we compute g - 1 max flow, and so on. And indeed, g + g - 1 + ... + 1 = $\frac{g(g+1)}{2} = \Omega(g^2)$.
Unfortunately it is difficult to select a "best" method: we can, for each of these methods, contruct examples such that the resulting cutting given is not the optimal and the "quality" of the resulting cutting (i.e, its weight) may depend strongly on the volume we consider, or the initial cutting.
The main issue is the intersections between 2-cuts: however short it may be, a min cut $\mathcal{C}_{min}$ computed by a max flow is not necessarily in a shortest system of 2-cuts since it may intersect a lot of other small cuts. Bad intersections are more likely to occur with the MPA than the SPA, since the MPA uses more cuts than the SPA.

3.5 Parallelization

I suggested the following method to optimize simultaneously several cuts, with the MPA: compute the graph which vertices are the 2-cuts and with an edge between two vertices if the two corresponding cuts have an adjacent pant in common. Then find an independent set (or an approximation: finding an independent set is NP-hard) in this graph and optimize simultaneously every cut in this set. This way we are sure cuts we optimize will not interfere.
When all these optimizations are done, compute an independent set of the remaining vertices (which were not optimized) and optimize such a set. Continue this process until there is no more cut to optimize.

3.6 Selection of a cutting system

When the SPA is finished, there is only one choice for the final system of 2-cuts (we must take all the 2-cuts) and for the MPA we simply use the greedy choice: iteratively select the shortest 2-cut, while possible (i.e, until there are g 2-cuts selected).

4 Neighborhood

4.1 The algorithm

For simplicity, in this section we assume that our volumes are made of voxels, unless explicitly stated. The most time consuming part of both SPA and MPA being the computation of a max flow, one may try to improve its time complexity, taking advantage of the fact that G^* is the dual graph of a volume. The basic idea of the neighborhood algorithm (NA) is that, during Ford Fulkerson algorithm, we can select an augmenting path "near" the previous found paths, which restricts the number of edges to visit. To define precisely what the "near" means, we need a more dense graph, to avoid "discontinuities":

Definition 11. The dual-adjacency graph $\overline{G^*}$ is the graph with the same vertices as G^* and with an edge (with neither a weight nor a capacity) between two

vertices if the two corresponding volumic elements share at least one vertex.

Definition 12. If v is a vertex and p a set of vertices (for example, a path), d(v, p) is the minimal length of a shortest path, in $\overline{G^*}$, between v and a vertex of p.
Given two sets of vertices p and p', $d(p, p') = \max_{v \in p} d(v, p')$.
The p-neighborhood $\mathcal{N}_p$ is the set of all vertices v such that $d(v, p) \leqslant 1$.

Clearly, a path p' is included in $\mathcal{N}_p$ if and only if $d(p', p) \leqslant 1$. To compute a max flow, we can use the following algorithm (the algorithm keeps a growing neighborhood N where the paths are searched):

Algorithm 1: Neighborhood algorithm (NA)

Let p_i be a shortest path from s to t in R (using BFS) ;
$N \leftarrow \mathcal{N}_{p_i}$;
$f \leftarrow 0$;
while *There is an augmenting path p from s to t in* $R_f \cap N$ **do**
  Augment the flow f along p ;
  $N \leftarrow N \cup \mathcal{N}_p$;

R_f is the residual network of R (built according to section 3.3) and $R_f \cap N$ means: the graph with N as the set of vertices and with every edge of R_f linking two vertices of N.
Of course, the first path computed may visit all the reachable edges (as in the classic Ford Fulkerson algorithm), but all other paths are searched in the growing neighborhood.
The following theorem shows that we actually get a max flow at the end of the neighborhood algorithm:

Theorem 6. *Assume we are computing a max flow in* $\mathbb{V}$*, using neighborhood algorithm and that we already found a not empty set of augmenting paths* $\mathcal{P}$*.*
Let f be the current flow an R_f *the residual network.*
If there is no path from s to t in $R_f \cap \mathcal{N}_{\mathcal{P}}$ *then f is a max flow.*

Proof:
Assume f is not a max flow: there is at least one path p_0 from s to t in R_f.
$\{ d(p, \mathcal{P}), p \text{ path from s to t in } R_f \}$ is a finite, non empty ($d(p_0, \mathcal{P})$ belongs to it) set of $\mathbb{N}$ so it has a minimum m, let p_1 be a path such that $d(p_1, \mathcal{P}) = m$.
If $m \leqslant 1$, $p_1 \in R_f \cap \mathcal{N}_{\mathcal{P}}$ and the theorem is proved.
Otherwise, let v be a vertex in p_1 such that $d(v, \mathcal{P}) \geqslant 2$.
Since $d(v, \mathcal{P}) \geqslant 2$, $\mathcal{N}_{\mathcal{P}} \cap \mathcal{N}_v = \varnothing$ and every edge in $\mathcal{N}_v$ has a flow equals to 0 (i.e, these edges are R_f). Let q be the path $\mathcal{N}_v \cap p_0$.
Since all paths from s to t are homotopic (such that there is no "hole" between them), we can find a path q' in $\mathcal{N}_v$ with same extremities than q and replace q by q' such that for all vertices w added this way, $d(w, \mathcal{P}) < d(v, \mathcal{P})$.
If we do this with all such vertex v, we create a path p such that $d(p, \mathcal{P}) < m$, a contradiction. So $m \leqslant 1$.

□

4.2 Complexity

For simplicity, we assume in this subsection that every edge has capacity 1 (it is the case, for example, if the capacity of an edge is equal to the area of the corresponding face, for voxels).

Theorem 7. *Let p be the shortest path from s to t (we assume that it is already computed with a BFS), $\mathcal{C}$ a min s-t cut.*
The neighborhood algorithm (without the computation of the first path) has complexity $O(p|\mathcal{C}|^2)$.

Proof:
Throughout the algorithm, the neighborhood grows and at the end, its size is inferior to $p|\mathcal{C}|$ (the size of a complete cylinder with axis p). So we visit at most $p|\mathcal{C}|$ vertices for each BFS, and there are $|\mathcal{C}|$ such BFS.

□

This bound is likely to be much better than the $O(VE^2) = O(V^3)$ required by the Edmond-Karp implementation of the Ford Fulkerson algorithm.
In the case of tetrahedra, this lemma is false since we can have an arbitrary number of tetrahedra adjacent to one tetrahedron (however, the algorithm may perform very well on tetrahedra, see section 5.8).
The first path p can be computed in $O(\min(p^3, V))$, indeed we stop the BFS as soon as we reach t so we can't visit more than $\frac{4}{3}\pi\ p^3$ vertices (the size of a ball with radius p). Then the whole NA has complexity $O(\max(\min(p^3, V), p|\mathcal{C}|^2))$.

4.3 Variant: the Step Neighborhood Algorithm

Instead of augmenting the neighborhood whenever a s-t path is found, it may be interesting to find paths in the neighborhood while possible and to augment the neighborhood by step, only when no path is found. If no path is found after we increased the neighborhood, the algorithm stops and reports a max flow.

Algorithm 2: Step neighborhood algorithm (SNA)

```
Let p_i be a shortest path from s to t in R (using BFS) ;
N ← N_{p_i} ;
f ← 0 ;
while There is an augmenting path p from s to t in R_f ∩ N do
    P ← p ;
    Augment f along p ;
    while There is an augmenting path p' from s to t in R_f ∩ N do
        Augment flow along p' ;
        P ← P ∪ p' ;
    N ← N ∪ N_P ;
```

The advantage of this variant is that N is more "compact", we only augment it when needed so we visit less edges.
It also computes sligthy longer paths than the NA and computes more BFS since every time we augment N, we do a "useless" BFS which find no path, although this BFS is performed when $R_f \cap N$ contains few edges so it is cheap.

5 Implementation

5.1 Overview

Initially, it was planned that I develop the optimization part of the MPA, and that the rest of the team will develop the initial pants decomposition, all in C++. After two weeks working on the core algorithm, which deals with the dual graph so that it is independant from the representation (voxels or tetrahedra), I decided to use tetrahedra, although more complicated to implement than voxels, to be able to work on surfaces, with Blender[1] for example, and use TetGen[2] to tetrahedralize them. I also used TetMeshLib, a library to manage tetrahedral meshes that Marco Attene developed. To both automatize the different tools of the algorithm and visualize more easily, I wrote a script in Blender Python API. At the end of the internship, I worked together with Jean-Marie Favreau, using GitHub [3]. Globally, I wrote about 3000 lines in C++ and 500 in Python.

5.2 SGL

SGL (Simple Graph Library) is a graph library I started to develop few months before the internship and that I used in this internship. It relies on *generic programming*: an ADT (Abstract Data Type) defines the required methods and semantic for a class and is used through template parameter.
For example, the class BFS has a `Graph` template parameter, which must provide, among other, a class named `iterator` giving a way to iterate over all edges adjacent to a given vertex.
OC3D was designed in a similar way.

5.3 Data structures

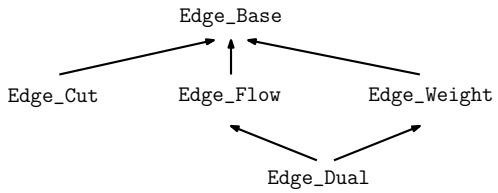

Figure 8: Hierarchy of the edge classes (an arrow means "inherits from")

Since we often need to iterate over all edges adjacent to a vertex, every graph is implemented as a adjacency list graph (`Graph_List<Edge>`). I use mainly three graphs: the pants graph, the dual graph G^* and the dual-adjacency graph $\overline{G^*}$.

Definition 13. The pants graph, associated to the current set of pants computed by the SPA or MPA, has one vertex for each pant and one edge between two vertices if the two corresponding pants share at least one 2-cut.

[1] www.blender.org
[2] http://tetgen.berlios.de/
[3] The project is available at http://github.com/jmtrivial/OC3D

It is needed, only for the MPA, during an optimization of a 2-cut.
The pants graph is directed, while G^* and $\overline{G^*}$ are undirected, this is needed by the way the residual graph is used by the implementation of Ford Fulkerson algorithm: we access the residual capacity of an edge of G^* (of type `Edge_Dual`) providing the direction of the edge (i.e, the end extremity), it uses less memory and time since we manipulate only one edge pointer (this edge pointer is stored in the adjacency list of both extremities). Since we want to be able to add flow along the two possible directions of every edge at the beginning of a max flow, every edge of G^* has a reverse (in the opposite direction) and a pointer to its reverse.
An edge (i.e, a 2-cut) of the pants graph is of type `Edge_Cut`: its extremities are the pants adjacent to the corresponding 2-cut and it stores the list of the pointers to `Edge_Dual` elements included in the 2-cut. Every `Edge_Dual` in a `Edge_Cut` has the same orientation, i.e their end extremities are in the same pant. This is needed to know to which vertices we must link the source (or sink) before running Ford Fulkerson algorithm, or to delete the edges out of the pant we optimize (section 5.5 explains how to give an orientation to a cut). To avoid time waste, every 2-cut has a reverse and a pointer to it.

5.4 Core algorithm

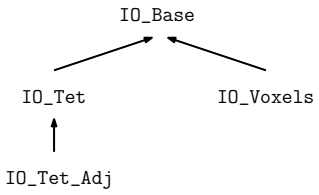

Figure 9: Hierarchy of the IO classes.

To improve genericity, the algorithm first build the dual graph (which depends on the the representation used - voxels or tetrahedra) and then works only on the dual graph.
For that purpose, an object is used to deal with representation-dependant operations, IO_Base implements the required methods of such an object. IO_Tet defines how to make the dual graph from tetrahedra and IO_Voxels (made by Jean-Marie Favreau) from a voxels image. IO_Tet_Adj also defines how to build the dual-adjacency graph from tetrahedra.
After the dual graph is made, we can search a max flow on it with a class derived from `Max_Flow`, and `Cut_vertices` is a class giving a min cut from a max flow, searching for the set S of every vertex reachable from the source s and setting as the cut the set of edges connecting S and S^c.

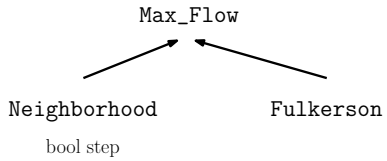

Figure 10: Hierarchy of the max flow classes.

5.5 Orientation

When we get a cut, selecting it with Blender for example, we must give it an orientation (such that every edge in the cut are oriented toward the same pant). After discussing it with Jean-Marie Favreau and Vincent Barra, the best solution may be the following:

- Take an element of the cut (a triangle or a square, say a triangle abc) and give it an orientation, which can be viewed as a permutation σ of its vertices.
- For each triangle uvw adjacent to it with an edge uv, and if $\sigma(u) = \text{v}$, defines the orientation on t according to the permutation vuw.
- Repeat the second point replacing abc by all new triangles with an orientation, until every triangle has an orientation (we assume here that the cut is connected).
- For each edge (of G^*) uv in the cut, associated to a triangle with orientation (abc), compute $\det(\overrightarrow{uv},\overrightarrow{ab},\overrightarrow{ac})$: if it is negative, replace uv by vu.

This algorithm is clearly linear.

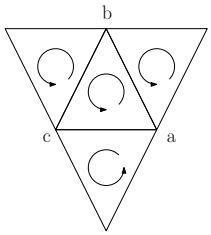

Figure 11: An orientation.

5.6 Pants graph

We assume here that every cut is oriented.
When we compute a first set of 2-cuts (selecting them manually or with a reduced cut locus) we then want to compute the pants graph. This is straightforward for the SPA (the pants graph is complete), I now focus on the MPA.
Let $\mathcal{C}$ be a 2-cut such that the adjacences of one of its side S is not defined and e an edge of $\mathcal{C}$ such that the first extremity v is in the side S.
My first idea was to do a BFS from v and adding to the list of adjacent cuts of

$\mathcal{C}$ every cut found by the BFS.
However, Jean-Marie Favreau noticed that we can have "thin" (and disconnected) pants between two cuts such that the BFS will not find every cut adjacent to $\mathcal{C}$. After discussing it, to solve this we can do more BFS on the extremities of the edges of $\mathcal{C}$ in S while possible and adding cuts found this way to the list of adjacences of $\mathcal{C}$ (avoiding multiple insertions of the same cut).
Moreover, we can't change the adjacences of the cuts after optimizing a cut in the MPA (since we replace a cut by one of its homotopy class), so we need to do the above algorithm only once.

5.7 Python script

The script[1] I wrote for Blender uses, with pipeline, OC3D_debug, a command-line debugger which can, among other, load a mesh, make the dual graph, optimize cuts step by step and output various information. To avoid freezing Blender while optimizing a cut I used several threads and I had to synchronize the threads, OC3D_debug and TetGen.

Figure 12: Script interface.

5.8 Running time comparison

To compare the algorithms of max flow (the classic Edmonds-Karp variant of the Ford Fulkerson algorithm, the NA and the SNA), I tested them on a slighty modified torus, with various number of vertices.

Complexity (edges + vertices) of the dual graph	Classic	NA	SNA
200K	111s	61s	29s
900K	1147s	236s	101s
1750K	2975s	318s	158s

Table 1: Running time in seconds of the optimization of one 2-cut on a torus using several algorithms of max flow (K stands for 1000).

The timing runs were performed on a Dell inspiron 1520 with Intel Core 2 Duo T7300 Processor (2.0GHz, 4MB L2 cache) and 2048 Mo memory. Compilation was done with Visual C++ 2008 with all optimization flags set for

[1] which can be found together with a video describing its working at http://perso.ens-lyon.fr/quentin.fortier/OC3D/Script/script.html

maximum speed.
The advantage of the NA and SNA increases significantly while the number of vertices of G^* increases and the times required by the SNA is barely greater than the time required by TetGen which has the complexity $O(\mathbb{S}^2)$ of the Delaunay tetrahedrization (about 80 seconds for the torus which dual has a complexity 1750000), see [8]. I noticed an improvement of the time required by the NA (and SNA) when the quality (inversely proportional to the radius-edge ratio of the tetrahedra) of the tetrahedrization increases, since if we increase the quality, we decrease the number of edges in $\overline{G^*}$ (and then the neighborhood is also reduced), see figure 13.
To perform tests, I used the basic option -q constraining the radius-edge ratio to be less than 2, but is also possible to constrain more the radius-edge ratio (up to 1.414 for example, see [8]), although this may increase the time required by TetGen. However a drawback of this contrained generation is that the number of vertices of the tetrahedrization increases slighty (for example, the tetrahedrized torus of the figure 13 has about 50K edges with -q and 40K without).

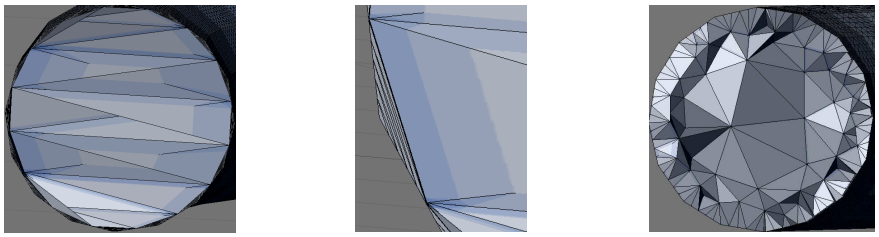

Figure 13: A section of a tetrahedrized torus. Left: without the option -q of TetGen. Middle: without -q, zoom on a border. Right: with -q.

I also compared, on the dual graph with complexity 900000, the time required for a BFS at different moments of the algorithms. I split the execution of the

	SNA			NA			Classic	
Paths	Edges in N	Total	Avg	Edges in N	Total	Avg	Total	Avg
1	101 (in path)	250	250	101 (in path)	250	250	250	250
21	2452	41	1,95	2452	49	2,33	4917	234,14
129	4240	283	2,19	5084	446	3,46	30140	233,64
363	9428	1295	3,57	11282	2500	6,89	86221	237,52
741	21822	5234	7,06	25930	10028	13,53	165232	222,99
1735	44944	22627	13,04	50886	46345	26,71	364113	209,86
2798	87172	66810	23,88	101262	144944	51,8	559327	199,9
249	161664	10568	42,44	184092	16781	67,39	48611	195,22
0	173368	22	22	189654	33	33	94	94

Table 2: Detailed total time and average time comparison on a torus with 700K edges (times are without units)

algorithms in different steps defined by the increase of the neighborhood used by the SNA (thus, the SNA found 1 path before augmenting, then 21, 129 and so on). Then I compared the number of edges of the neighborhood in which the paths were found, the total time during the step, and the average (shortened

Avg) time of a BFS. The advantage of a neighborhood decreases slighty during the algorithm, since the neighborhood increases as the number of edges visited by a BFS on the whole graph decreases (some edges have then zero capacity and disappear from the residual graph), so the algorithm is likely to be more efficient if the min cut is small (this is especially true for voxels with capacity one, since the size of the min cut is directly the number of iterations of the algorithm).
I didn't have the time to test this algorithm on voxels but, for non degenerated volumes, the first neighborhood will be at most eight times the length of the first path found (if the path is a straight path, there is exactly eight disjoint paths adjacent to it). For that example, it means that the first neighborhood is likely to be about 800 edges instead of 2452 for tetrahedra, three times less, so I expect the neighborhood algorithm to be more efficient with voxels.

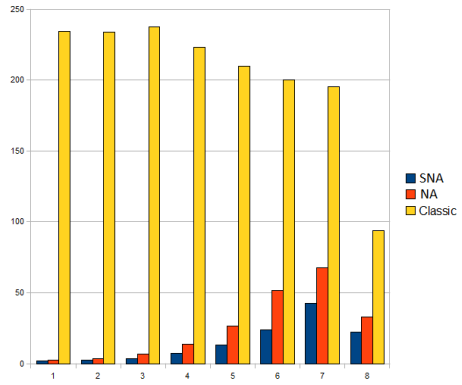

Figure 14: Average time of a BFS with respect to the step in the SNA.

5.9 Computation of the first path

Since I used a rather "big" min cut in the previous tests, the time required by the first BFS is negligible, but if the min cut is smaller it may be interesting to find a way to avoid this first BFS (for example if the min cut is such that the first 100 paths found gives a max flow, the first BFS takes a time equal to the half of the total time required by the SNA). I describe an algorithm to do this, with the SPA: assume we have a set of cuts $\mathcal{C}_1$, ..., $\mathcal{C}_g$ we want to optimize.
Before any max flow, we can compute a set of shortest paths p_1, ..., p_g such that p_j is a shortest path from one extremity of an edge of $\mathcal{C}_j$ to the other extremity, without intersecting any of the cuts (I call here such a path *a valid path* for $\mathcal{C}_j$). This can be done in O(gV) if we use g BFS. Then, if we optimize a cut (say, $\mathcal{C}_1$) we can use p_1 as the first path. The max flow on $\mathcal{C}_1$ gives a min $\mathcal{C}_1$ s-t cut $\mathcal{C}_{min}$.
Clearly, p_1 is also a valid path for $\mathcal{C}_{min}$, but this may not be the case for the other paths (p_j may intersect $\mathcal{C}_{min}$). Let $j \geqslant 2$: if p_j doesn't intersect $\mathcal{C}_{min}$, p_j is a valid path for $\mathcal{C}_j$.
Otherwise, we replace p_j by the path obtained by concatenating p_j before the intersection with $\mathcal{C}_{min}$, then p_1 and then the rest of p_j (see figure 15). This can be done in $O(\mathcal{C}_{min}p_j)$ (we just need to verify the intersection and to link

suitably p_j and p_1).

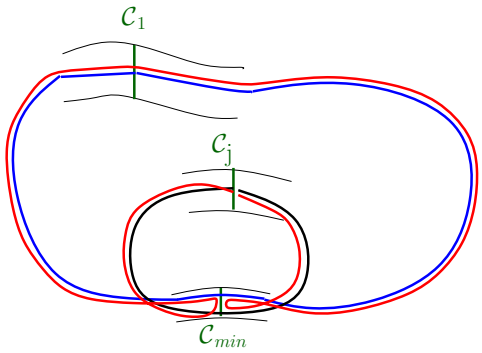

Figure 15: Illustration of the method to update a path p_j (black) to have a valid path (red). p_1 is blue.

Since we must do this for all (g-1) cuts, the complexity for "updating" all other paths is O((g-1) $\mathcal{C}_{min}$ $\max_j(p_j)$), which is likely to be better than the O(V) required by a BFS. However this may lead to sligthy longer paths, increasing the size of the neighborhood used for the next max flows.
With the MPA, we can simply use the skeleton as a set of first paths. There is no need to update them since these paths are always valid (homotopy classes of cuts can't change).

6 Conclusion

Many issues, either theoretical such as the NP-hardness of the problems studied or pratical such as the computation of the first cuts and a order of cuts processing for which we can prove correctness of the algorithm, remain after this internship.
Because of the large variety of possible configurations of the cuts in $\mathbb{R}^3$, it is difficult to manage all of them and, even after the internship, I discussed a lot with Jean-Marie Favreau about unexpected configurations.
The code developed will be integrated to an algorithm (of the Recods[1] project) aiming at robustly investigating the brain modifications to detect disorders such as Alzheimer's disease and to take into account anatomical variability among humans. More especially, the code will be used to remove handles resulting from the meshing or the segmentation steps, as explained in [5].

7 Acknowledgement

I thank Vincent Barra, for introducing me the subject and for giving me great liberty, Jean-Marie Favreau for letting me use his pictures and for the help he

[1] REconstruction, COrrection and DEformation of brain Surfaces, involving both LIMOS and the Australian e-Health Research Centre (CSIRO, Brisbane)

provided on various subjects. I also thank Marco Attene for the TetGen library he provided and Éric Colin de Verdière for the interesting discussion (by email) we had on theoretical aspects of this internship.

8 Appendix: mathematical background

In all this report, we only consider subsets of $\mathbb{R}^n$ (n = 2 or 3) with the Euclidean distance and norm $\| \ \|$.

Definition 14. f : X $\longrightarrow$ Y is an *homeomorphism* if it is a continuous bijection and if its inverse is continuous.

Definition 15. X and Y are *homeomorphic* if there is an homeomorphism from X to Y.

Definition 16. X is a *n-manifold (with boundary)* if every point of X has a neighborhood homeomorphic to the open ball $\{ x \in \mathbb{R}^n, \| x \| < 1 \}$ or to $\{ x = (x_1, \ldots, x_n) \in \mathbb{R}^n, \| x \| < 1$ and $x_1 \geqslant 0 \}$. The boundaries of X are the connected components of the points with a neighborhood homeomorphic to $\{ x = (x_1, \ldots, x_n) \in \mathbb{R}^n, \| x \| < 1$ and $x_1 \geqslant 0 \}$.

Definition 17. A *loop* on $\mathbb{S}$ is a continuous map $\gamma : [0, 1] \longrightarrow \mathbb{S}$ such that $\gamma(0) = \gamma(1)$.

Definition 18. Two continous maps $f_1, f_2 : [0, 1] \longrightarrow X$ are *homotopic* if there exists $H : [0, 1]^2 \longrightarrow X$ continuous such that $H(0, t) = f_1(t)$ and $H(1, t) = f_2(t)$, $\forall\ t \in [0, 1]$.
A loop is *contractible* if it is homotopic to a point.

Definition 19. The genus of a surface (i.e, a two-dimensional topological manifold) without boundary $\mathbb{S}$ is the maximum number of loops on $\mathbb{S}$ without intersections and such that they don't disconnect $\mathbb{S}$.

I don't give the precise definition of a 2(or 3)-dimensional *simplicial complex*[1] the reader can simply think of it as a "good" set of vertices, edges, triangles (and tetrahedra for 3-dimensional *simplicial complex*).

Definition 20. A n-dimensional simplicial complex which is a n-manifold is a *n-mesh*.

References

[1] J.-M. Favreau. Outils pour le pavage de surfaces. PhD thesis, Blaise Pascal university, 2009.

[2] É. Colin de Verdière. Raccourcissement de courbes et décomposition de surfaces. PhD thesis, Paris 7 university, 2003.

[3] É. Colin de Verdière and F. Lazarus. Optimal pants decompositions and shortest homotopic cycles on an orientable surface. Proceedings of the 11th Symposium on Graph Drawing , 2003.

[4] É. Colin de Verdière. Algorithms for graphs on surfaces, M.P.R.I. course Master 2. 2009-2010.

[1] see http://en.wikipedia.org/wiki/Simplicial_complex for a precise definition

[5] Cortical surface mapping using topology correction, partial flattening and 3D shape context-based non-rigid registration for use in quantifying atrophy in Alzheimer's Disease. Preprint submitted to International Journal of Biomedical Imaging, March 2010.

[6] J. Erickson, S. Har-Peledz. Optimally Cutting a Surface into a Disk. Discrete and Computational Geometry, July 2, 2002.

[7] N. D. Cornea, D. Silver, P. Min. Curve skeleton properties, applications and algorithms. IEEE Transactions on Visualization and Computer Graphics 13 (2007), pp 530-548.

[8] H. Si. TetGen: User's manual. 2006.