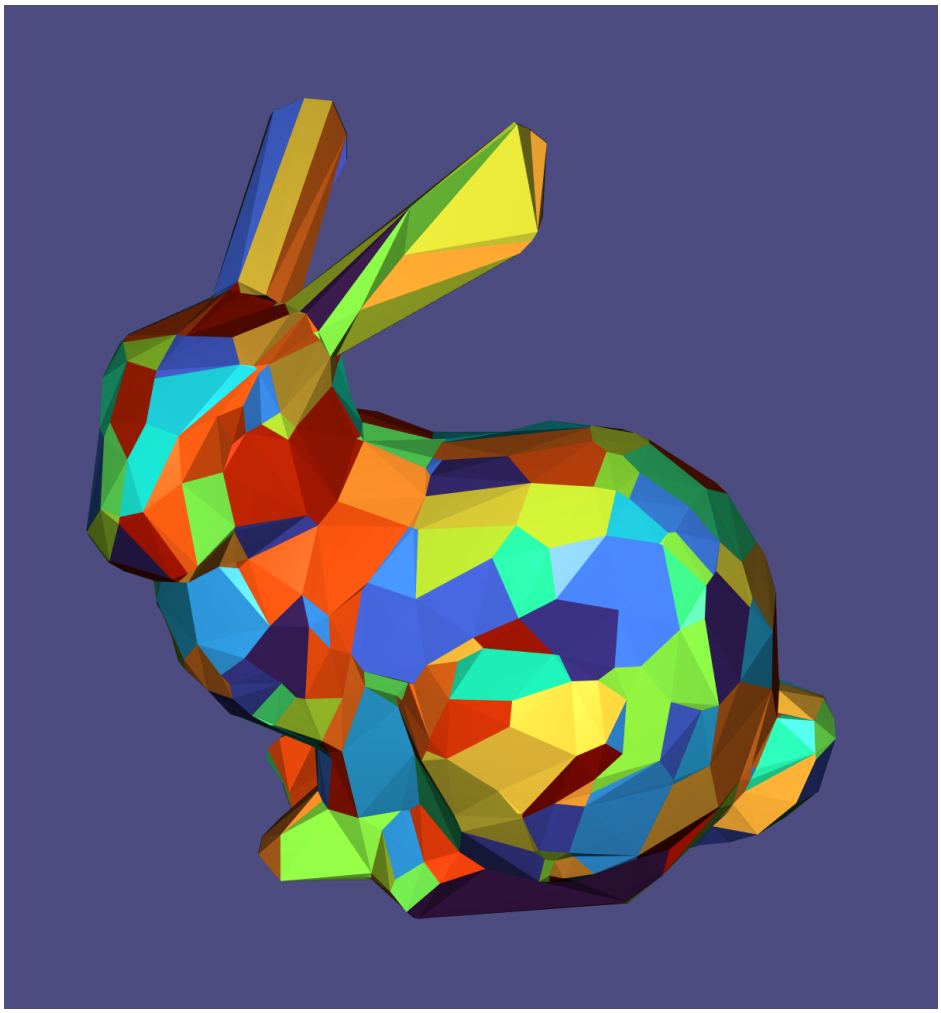
INF574 Project: Variational Shape Approximation

Capucine Leroux and Nissim Maruani

December 2020



Contents

1 Problem description and main ideas

The method presented here is called *Variational Shape Approximation* and was first introduced by David Cohen-Steiner, Pierre Alliez and Mathieu Desbrun [1]. The goal of this method is to approximate a given shape with a small number of geometric proxies. This allows to compress the representation without losing too much precision.

When we look at a 3D object, we pay attention to two different things: the overall shape of the object and its lighting (in fact, we are more sensitive to a normal change than to a position change). A good compressed mesh preserves these two aspects.

To do so, the main idea of the *Variational Shape Approximation* algorithm is to cluster the mesh into a fixed number of regions as to minimize a distortion error. The regions are then triangulated to produce a compressed mesh. The paper describes the possible distortion error distances they use and the implementation of the algorithm. Unlike most methods, this paper introduces an error-driven approximation, which means that the topology might not be conserved on some details but the distortion will be minimized.

1.1 Distances

Each region of faces will be approximated by a proxy $P(X, N)$, where X is a point and N a normal. To evaluate if a proxy makes a good approximation, we need an error metrics. Two distances are presented in the article :

$$\begin{aligned} L^2(R_i, P_i) &= \iint_{x \in R_i} ||x - \Pi_i(x)||^2 dx \quad \text{where } \Pi_i \text{ is the orthogonal projection on } P_i. \\ L^{2,1}(R_i, P_i) &= \iint_{x \in R_i} ||N(x) - N_i||^2 dx \quad \text{where } N(x) \text{ is the normal of the point's face.} \end{aligned}$$

L^2 is an extension of the classic Hausdorff metrics, and $L^{2,1}$ is an innovation of the paper. It better captures the anisotropy of the surface and enables an easier implementation of the optimal proxy.

1.2 The algorithm

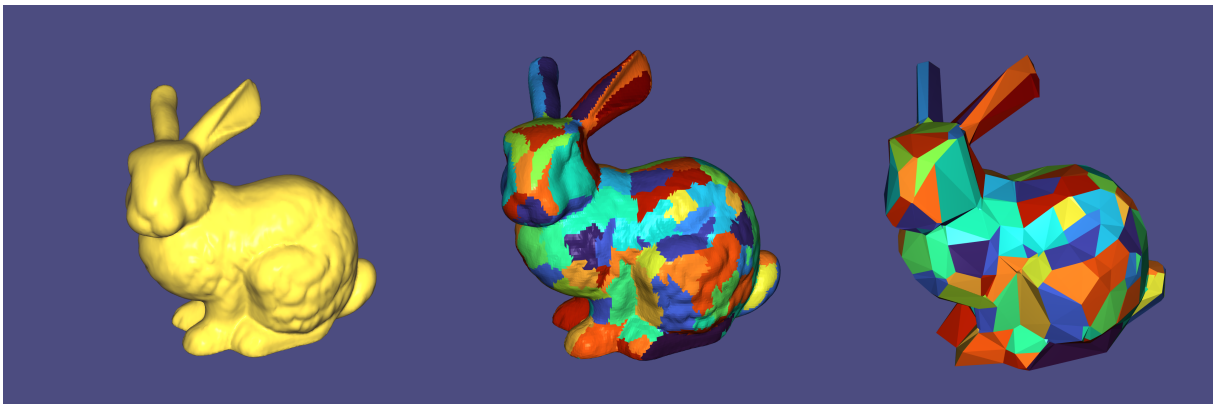


Figure 1: From a 70k faces bunny to a 1K one

The algorithm is inspired from the Lloyd's clustering algorithm (k-means clustering). It alternates between two steps, geometry partitioning and proxy fitting, until the global error converges or cycles.

Geometry Partitioning: We initialize the k regions with one triangle face per region and an associated proxy. Then we flood the rest of the triangles, assigning them to the region from which they are the closest to (minimizing L^2 or $L^{2,1}$ metrics).

Proxy fitting: Once we have k regions of triangle faces, we find the optimal set of proxies to minimize the global

distortion error :

$$E = \sum_{i=1}^{k} L^{2 \ or \ 2,1}(R_i, P_i)$$

1.3 Remeshing

Once we have k optimal proxies for each of the regions, we want to produce a final mesh. To do so, we define anchor vertices on the border of each region that will be the vertices of our new mesh. We then connect these vertices with edges to define the triangular faces of the new mesh.

2 Implementation

We implemented the partitioning, the proxy fitting and the re-meshing pipeline described in the paper. In addition, we chose to implement one of the improvement suggested: the Farthest-point Initialization.

2.1 Structures used

Mesh: We store the mesh as a matrix V of size $v * 3$, and a matrix F of size $f * 3$. On top of that we added a $f * 3$ matrix, Ad, to store the adjacency of the original faces: each one has at most 3 neighbours (we work on triangular meshes). We also use the HalfEdge structure seen in the TDs to efficiently run through the boundary edges of the regions.

Proxies: Each proxy is stored as two 3D vectors in the matrix *Proxies*, one for its position and one for its normal.

Regions: The regions are stored in a vector R of size f: each face is assigned to a region.

2.2 Distances

We first needed to have a practical implementation of the L^2 and $L^{2,1}$ error metrics. For that, we used the formulas described in Appendix B of the paper. To optimize our code, we pre-compute the area and normal of each triangle of the original mesh.

2.3 Initialization of the proxies

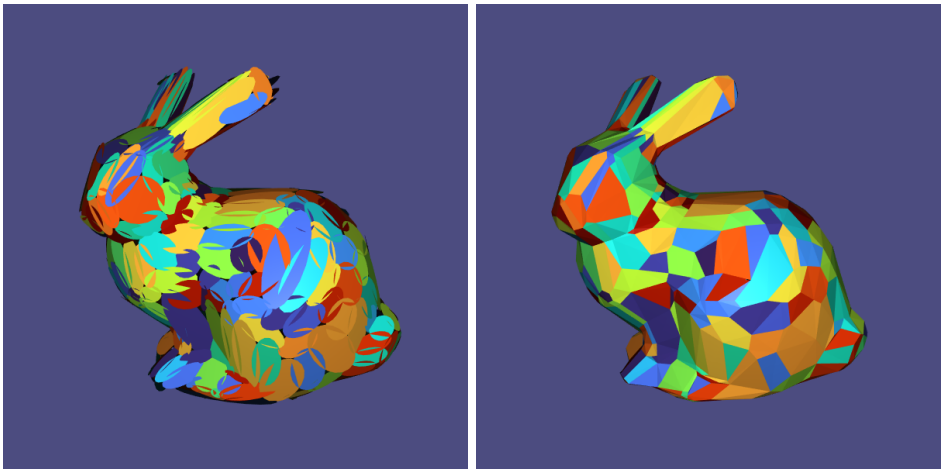
We first find k triangles and use their normal and centre of gravity to define the first set of proxies. To find the triangles, we implemented two methods:
- Random, with a uniform distribution
- Farthest-point Initialization, where we choose the first triangle at random and then select at each iteration the triangle which it farthest to its proxy to define a new one.

2.4 Geometry Partitioning

- For each proxy p, we find its closest triangle (with the L^2 or $L^{2,1}$ metric), set its region to p in R, and insert its neighbours in a priority queue q with the following information: distance to the proxy, face, region.
- While q is not empty, we de-pop its head. If the triangle's region hasn't been set yet, we update it in R and insert its neighbours in q.

2.5 Proxy Fitting

Once the regions $(R_i)_{1\leq i\leq k}$ are defined, we need to find the proxies $(P_i)_{1\leq i\leq k}$ minimizing the global error. The formulas of the optimal proxies are described in Appendix B of the paper. We used the Eigen solver for the eigenvectors and the eigenvalues needed for L^2



(a) (b)

Figure 2:
(a) The proxies defined after convergence
(b) The new triangulated mesh

2.6 Remeshing

The Remeshing part includes three main steps :
- Definition of anchor vertices and edges
- Triangulation
- Polygonalization

We decided not to implement the last part which consists in removing unnecessary edges to simplify faces, because we wanted to use a triangle mesh structure for input and output.

Anchor vertices and edges:

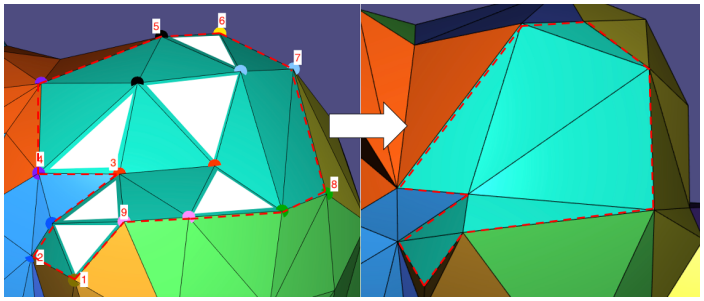
The anchor vertices are based on the vertices of the original mesh. Each vertex that is at the intersection of three or more regions defines an anchor. We then proceed to add more anchors on the regions' boundaries: if $x1$ and $x2$ are two neighbours anchors, we find the vertex y that maximizes the orthogonal distance $d(y)$ to $(x1, y1)$. If $d(y) * sin(N_{region_a}, N_{region_b})/||x1x2||$ is larger than some criterion (we found parameters between 0.1 and 0.4 work best in practice), y becomes an anchor. We repeat this process until each vertex on the boundary is either an anchor or satisfies the criterion.

Triangulation:

To triangulate a region, the main idea of the algorithm is to adapt the Djikstra's shortest-path algorithm to deduce Delaunay triangles starting from the anchor vertices. We used an half-edge structure and priority queues for this part.
The anchor vertices are the sources, they each have a different color (represented as an integer value). Each edge or

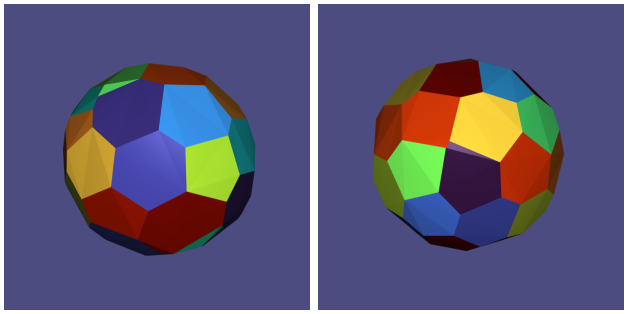
half-edge has a weight (its length). The goal is to assign to every vertex within the region the color of the sources it is closest to.
We flood the boundary vertices first to force the boundaries to be in the triangulation, then the interior vertices. To do the flooding, we used two priority queues, one for each flooding. The halfedge structure enables us to first flood vertices around the boundaries only, each time a vertex is assigned its color, it will add its two neighbors on the boundary to the first queue, and all of its interior neighbors to the second queue.
Once we have assigned a color to each vertex, we can deduce the triangulation by looking at every face that has three different color vertices (white triangles in Figure 3), the final triangle will be made out of the three corresponding anchor vertices.

Figure 3: Flooding and Triangulation



3 Results

3.1 General observations



(a) L^2 with 50 proxies (b) $L^{2,1}$ with 50 proxies

Figure 4: Similarities on spherical shapes

Both distances provide very conclusive results on simple closed shapes. On more complicated meshes, we tend to have better results with the $L^{2,1}$ metrics as it better captures details and anisotropy. However, on spherical shapes, the two norms act similarly (Figure 4).

The algorithm works better on a reduction of a large factor (> 10), when the number of regions is small compared to the number of faces of the original mesh.

3.2 Performances

Since the goal here is to reduce the meshes' size, speed isn't critical: the article mentions times between 3 seconds and 10 minutes to simplify a mesh. Although our algorithm isn't as optimized as the one described in the article, we could still compute 70k faces meshes (like the bunny on the Figure 1) in reasonable time (5 minutes) on our personal computers.

In our implementation, partitioning is the most time-consuming step (proxy fitting is pretty quick). Thanks to the adjacency matrix, the neighbours of each triangle are found in $O(1)$. The asymptotic complexity of each step is thus $O(f * log(f))$ which is clearly optimal (we sort the distances of the triangles to the proxies in the priority queue). The number of step required however varies, and it depends on the convergence criterion.

3.3 Convergence

The algorithm converges very quickly even though there is no mathematical convergence guarantee. We can observe that we do not need more than a hundred iterations to have a quasi-systematic stable distortion error (Figure 5).

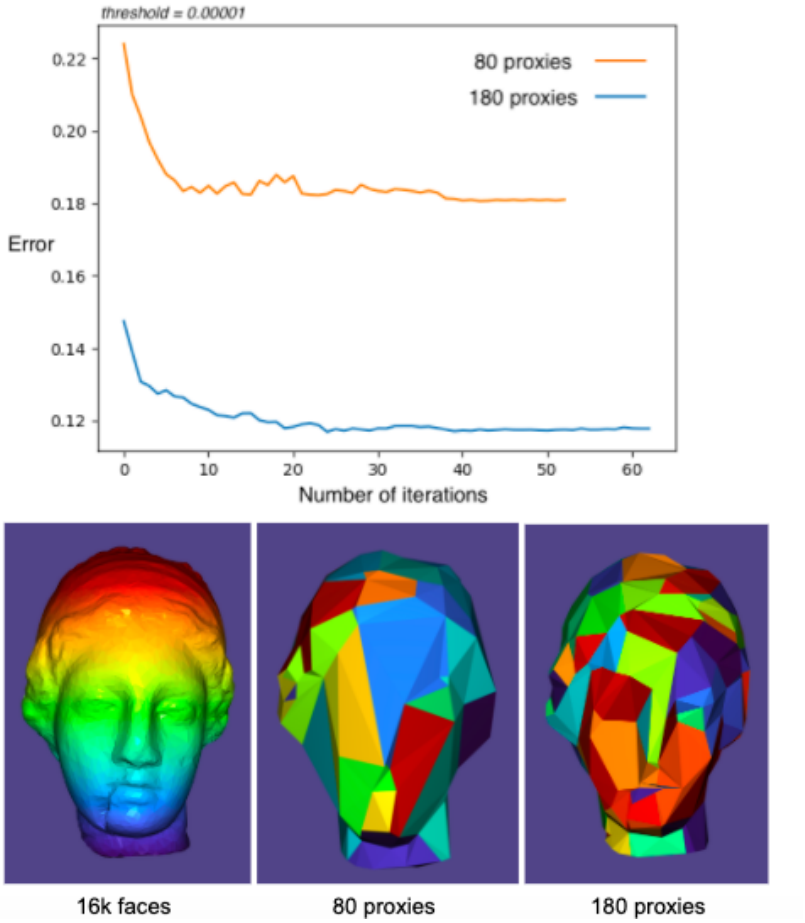


Figure 5: Convergence of the distortion error

3.4 Farthest triangle initialization

We find that the farthest triangle initialization method sometimes produces better looking result, but without significantly reducing the global error (Figure 6). In this case, both have a similar global errors, but we can see that the ears and legs of the bunny with the farthest triangle initialization are better defined. We conclude that the $L^{2,1}$ error metric isn't enough to describe the appearance of the compressed mesh.

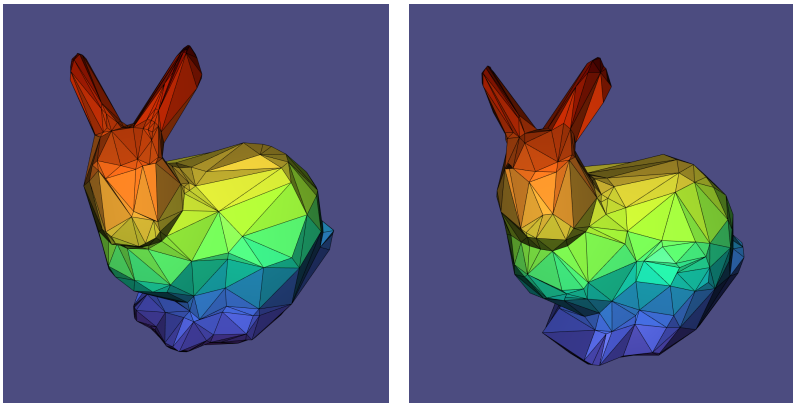


Figure 6: Farthest (0.0069 error) vs random initialization (0.0060 error)

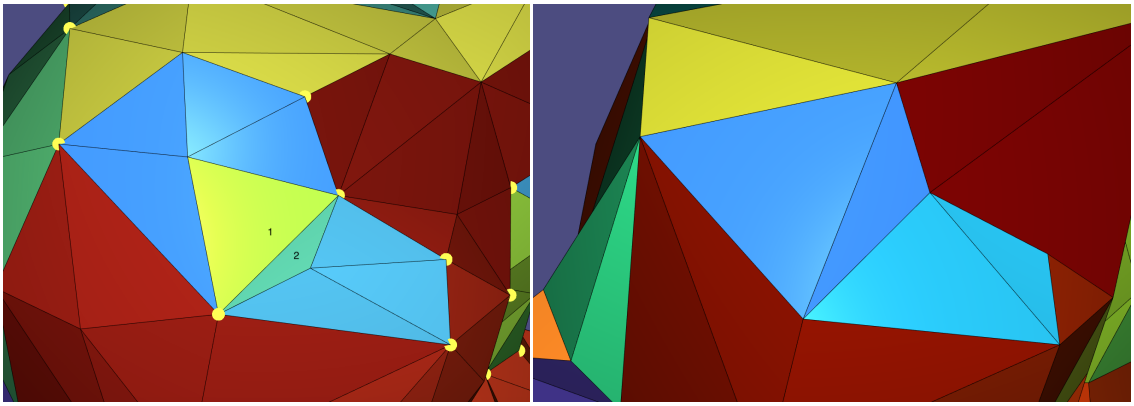


Figure 7: Regions with less than 3 anchors

3.5 Regions with less than 3 anchors

The article was rather elusive on adding special anchors when a region has less than three. If a region is included in another one, our algorithm does not find any anchor vertices and therefore cannot proceed to a triangulation. In this case, we decided to merge the two regions. If a region has less than three anchors, we decided to simply ignore it since it does not contribute much to the shape of the final mesh (see Figure 7).

3.6 Topology of the mesh

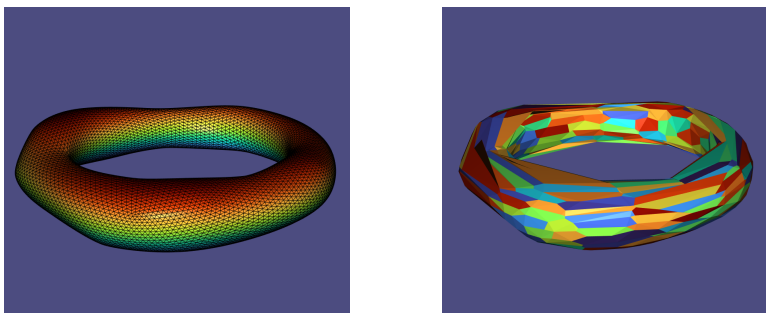


Figure 8: 16k to 1k torus

Due to the structure of our code, we can only use meshes that are closed (ie each edge separates two faces), but the topology of the mesh can vary, as seen on Figure 8.

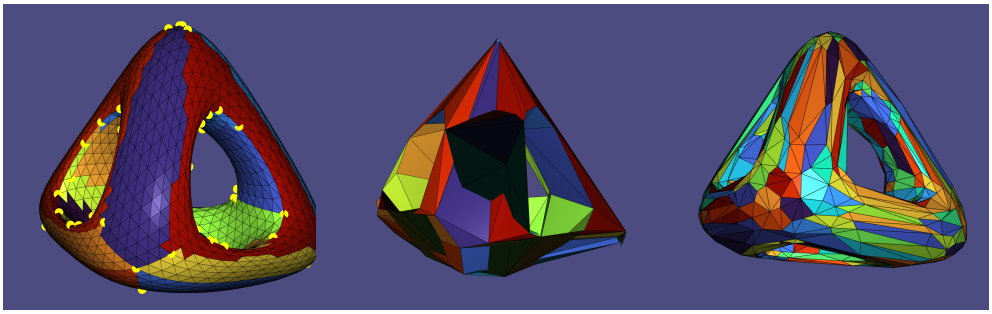


Figure 9: An example of incorrect and good approximation

However, if the mesh has some thin areas, problems can appear when using too few proxies: faces can intersect each other, as seen on Figure 9. The purple region is too big compared to the thickness of the model: the curvature isn't taken into account. The solution is to use more regions, as seen on the right.

3.7 Recursive anchors addition

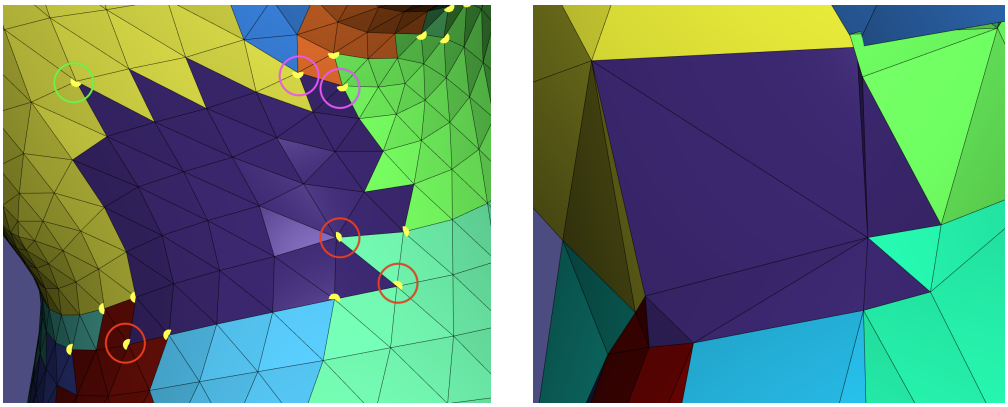


Figure 10: Useful and useless anchors on a 7k faces mesh

We think the algorithm defining the anchors might not be optimal in the case of coarse meshes. Consider the example of the Figure 10, the anchor circled in green is essential to define the purple regions, but the anchors circled in red are not. The anchors circled in pink should be merged as they are only separated by one edge and the tiny triangle they create doesn't contribute to the appearance of the rabbit. This is due to the lack of smoothness of the region's border: if the border separating the anchors is not long enough, the recursion might start adding every vertex that is on it.

4 Conclusion

Implementing this algorithm was challenging, but we found the result to be pretty satisfactory: we were able to compress relatively large meshes on our personal computers.

We could go further in the partitioning part, and implement other improvements. To simplify our code, we decided to only work on closed meshes, but we could extend it to meshes with free boundaries.

We could also improve our pipeline with several additional functions mentioned in the scientific publication, such as being able to remove or add regions at anytime during the process, change manually the number of proxies, smooth the normal field, or correct manually the level of details of some regions.

Bibliography

[1] *Variational shape approximation*
Cohen-Steiner, David and Alliez, Pierre and Desbrun, Mathieu
ACM SIGGRAPH 2004 Papers, 905–914, 2004