Scalable Mesh Refinement for Canonical Polygonal Schemas of Extremely High Genus Shapes

Marco Livesu, CNR IMATI

Abstract—Any closed manifold of genus g can be cut open to form a topological disk and then mapped to a regular polygon with $4g$ sides. This construction is called the *canonical polygonal schema* of the manifold, and is a key ingredient for many applications in graphics and engineering, where a parameterization between two shapes with same topology is often needed. The sides of the $4g-$gon define on the manifold a system of loops, which all intersect at a single point and are disjoint elsewhere. Computing a shortest system of loops of this kind is NP-hard. A computationally tractable alternative consists of computing a set of shortest loops that are not fully disjoint in polynomial time using the greedy homotopy basis algorithm proposed by Erickson and Whittlesey [1], and then detach them in post processing via mesh refinement. Despite this operation is conceptually simple, known refinement strategies do not scale well for high genus shapes, triggering a mesh growth that may exceed the amount of memory available in modern computers, leading to failures. In this paper we study various local refinement operators to detach cycles in a system of loops, and show that there are important differences between them, both in terms of mesh complexity and preservation of the original surface. We ultimately propose two novel refinement approaches: the former greatly reduces the number of new elements in the mesh, possibly at the cost of a deviation from the input geometry. The latter allows to trade mesh complexity for geometric accuracy, bounding deviation from the input surface. Both strategies are trivial to implement, and experiments confirm that they allow to realize canonical polygonal schemas even for extremely high genus shapes where previous methods fail.

Index Terms—Topology, polygonal schema, cut graph, homology, homotopy

1 Introduction

Any closed orientable surface with genus g has exactly $2g$ classes of homotopically independent loops. A system of loops containing one loop from each such class is also a homotopy basis [1]. If cut along its homotopy basis the surface becomes a topological disk, hence it can be flattened to the plane. In particular, if all loops emanate from a single source and are disjoint elsewhere, cutting the surface yields a polygon with $4g$ sides, called the *canonical polygonal schema* of the surface [2] (Figure 1, right). This construction is a topological invariant, hence any two shapes with same genus share the same polygonal schema, which can be used as a medium to initialize a cross parameterization between them [3], [4], [5]. For practical reasons cut graphs made of shortest (geodesic) paths are often preferred [6], [7]. However, this latter condition makes the problem NP-hard. A practical alternative consists in computing in polynomial time a shortest system of loops that possibly overlap at some mesh edge, using the greedy homotopy basis algorithm proposed in [1], and then detach such loops in post processing via mesh refinement. Despite apparently trivial, this refinement operation hides some difficulty. In fact, for high genus shapes the system will contain a big number of loops, which will largely snap to the same chains of edges, requiring massive mesh refinement to fully detach them.

To better understand this observation, we recall that the $2g$ loops of a discrete manifold with genus g should all intersect at a unique mesh vertex. For these loops to be fully disjoint, such vertex should have at least $4g$ incident chains of edges. For example, in a manifold with genus 100 the system of loops should centered at a mesh vertex with at least 400 neighbors. Having a mesh with such a connectivity is in practice extremely unlikely to happen. In fact, it is known that the average vertex valence for triangle meshes is equal to 6, which means that already for a manifold with genus 2 the chances that all loops in the basis will be fully disjoint are tiny, and mesh refinement is necessary.

Li and colleagues [3] proposed to use edge splits to detach loops in a greedy homotopy basis, but their refinement scheme does not scale well on high genus shapes, and in our tests it failed in 25% of the cases because the mesh became so big that it did not fit the memory available in our machine (Section 5). In this short paper we analyze alternative refinement strategies to detach loops in a given homotopy basis. Our main result consists in showing that the vertex split operator introduces far less new elements in the mesh, exhibiting a much better scalability on shapes with extremely high genus. Despite cheaper, the vertex split may occasionally introduce deviation from the input mesh. We therefore propose two alternative refinement strategies: the first one simply substitutes the edge split with the vertex split, significantly reducing mesh growth; the second one takes also geometry into account, and tries to use as many vertex splits as possible, switching to the costly edge split only when the former would introduce excessive deviation from the reference geometry. Experiments confirm that our refinement strategies outperform previous techniques, turning this strong theoretical framework into a practical algorithm to robustly initialize canonical polygonal schemas for shapes of any complexity.

2 Background and prior works

Before providing a precise formulation of our problem, we briefly introduce basic notions from computational topology and discuss previous works, also fixing the notation.

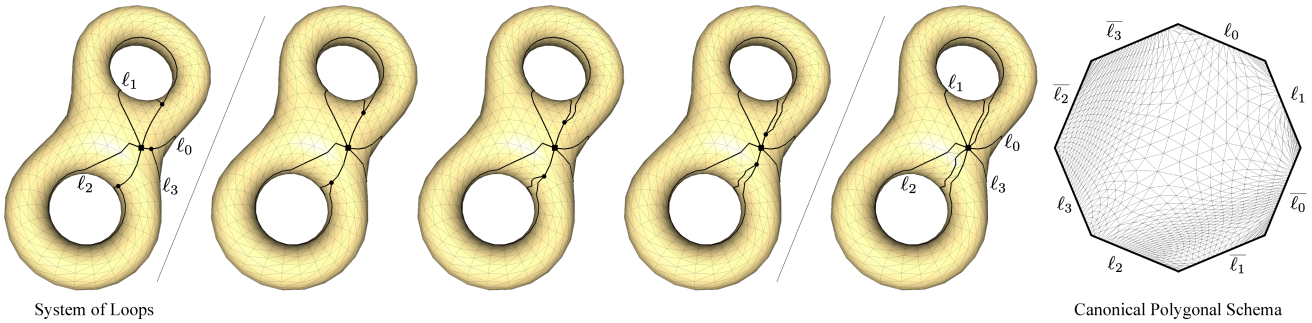

Fig. 1. Left: the greedy homotopy basis algorithm generates a system where loops are not fully disjoint (ℓ_0 and ℓ_1 merge at the black circle on top, ℓ_0 and ℓ_3 merge at the black circle in the middle, ℓ_2 and ℓ_3 merge at the black circle at the bottom). Middle: merging points are iteratively pushed towards the origin of the basis (black square) until they all vanish to it. Right: the associated canonical polygonal schema.

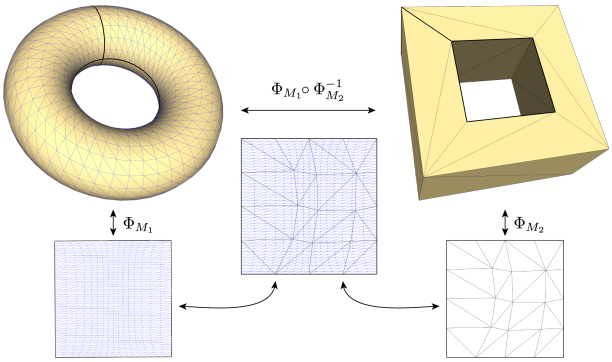

Fig. 2. A cross parameterization between a torus and its polycube (computed with [8]). Maps between any two homotopic shapes can be obtained by firstly projecting each shape to its canonical polygonal schema (left,right), and then using it as a medium to travel from one shape to the other (middle). Note that there are $4g$ possible ways to overlap the canonical polygons of two manifolds with genus g.

A $2-$manifold M is a topological space where each point is locally homeomorphic to $\mathbb{R}^2$. In the discrete setting, manifolds are typically represented as triangle meshes. With abuse of notation, in remainder of the paper we will use the symbol M to denote both the manifold and its combinatorial realization. The interpretation will become evident from the context.

Any discrete manifold M can be cut through a subset of its edges to form a topological disk. This set of edges is called the *cut graph* of M, and its nodes and arcs define the points and edges of a 2D polygon, called *polygonal schema* of M [9]. The *canonical* polygonal schema is a mapping of M to a regular polygon with $4g$ sides, where g is the genus of M. The cut graph associated to such a schema designs on M a system of $2g$ loops $L = \{\ell_0, \ell_1, \dots, \ell_{2g}\}$ that are fully disjoint except at a common vertex, called the *origin* (or *root*) of the system. The corners of the $4g-$gon are the images of the origin, and the edges are images of the loops, which are ordered according to the gluing scheme

$$\ell_0, \ell_1, \overline{\ell_0}, \overline{\ell_1}, \dots, \ell_{2g-1}, \ell_{2g}, \overline{\ell_{2g-1}}, \overline{\ell_{2g}} \ ,$$

with ℓ_i and $\overline{\ell_i}$ being two copies of a loop $l_i \in L$ (Figure 1). The canonical polygonal schema has two fundamental properties:

- it is a topological invariant, meaning that two manifolds with same genus map to the same polygon (up to a rotational degree of freedom);
- it is optimal, in the sense that among all the possible polygonal schemas, the canonical polygon has the least number of edges (i.e. there exists no $k-$gon with $k < 4g$ that is the cut graph of a manifold M with genus g [10])

Polygonal schemas play a central role in computer graphics, where they are at the basis of numerous applications, such as texture mapping [11], remeshing [12], compression [13], and morphing [14], to name a few. In particular, the properties of the canonical schema make it an appealing starting point to initialize a mapping between two shapes with same genus [3], [4], [5]. In fact, as shown in Figure 2, given two manifolds M_1, M_2 with genus g, and denoting with Φ_{M_1} and Φ_{M_2} their one-to-one maps to the canonical polygon P_{4g} :

$$\begin{matrix} \Phi_{M_1} : M_1 \leftrightarrow P_{4g} \\ \Phi_{M_2} : M_2 \leftrightarrow P_{4g} \end{matrix} \ ,$$

a cross parameterization $\Phi : M_1 \leftrightarrow M_2$ can be obtained through the composition

$$\Phi = \Phi_{M_1} \circ \Phi_{M_2}^{-1} \ .$$

Topologists and pratictioners in computer graphics have widely investigated the problem of computing cut graphs for discrete manifolds. Typically, the goal is to find the cut graph with minimal length, or the one that contains the least number of edges. Erickson and Har-Peled showed that both problems are NP-hard, and proposed a greedy algorithm to compute a $O(\log^2 g)-$approximation of the minimum cut graph in $O(g^2 n \log n)$ [9]. The so generated cut graphs may not be canonical. Dey and Shipper [10] propose a linear time method to compute a polygonal schema using a breadth-first search on the dual graph. Their cut-graph is not guaranteed to be shortest, and may not yield the canonical schema as well. In [15] Colin de Verdière and Lazarus propose a polynomial time algorithm that inputs a system of loops, and shrinks it in order to find the shortest system of loops in the same homotopy class. To mimic the continuous framework, the authors *"allow the loops to share edges and vertices in the mesh, provided that they can be spread apart on the surface with a thin space so that they become simple and disjoint except at the origin"*. The authors do not explain how this operation can be performed, and what impact it has on mesh size. In this

paper we focus on this very specific problem, aiming to find the mesh refinement strategy with minimal impact on the input manifold, both in terms of number of discrete elements and geometric fidelity. In [7] and [2] methods to compute system of loops that realize a canonical polygonal schema are presented. As already acknowledged by Lazarus and colleagues in their final remarks *"the obtained loops look too much jaggy and complex to be of any use for practical applications. More work needs to be done in this direction taking into account the geometry of the surface"* [7]. Dey and colleagues [6] propose a method based on persistent homology to robustly detect all handle and tunnel loops in a discrete mesh. This methods uses geodesic paths to grant geometry awareness, but does not produce a polygonal schema. Jin and colleagues [16] also use geodesics, and rely on the the Hyperbolic Ricci Flow to define the polygonal schema. Problems arising from multiple cuts that snap to the same mesh edges are not taken into account. At the time of writing, the greedy homotopy basis algorithm of Erickson and Whittlesey [1] can be considered to be the state of the art for computing arbitrary polygonal schemas on discrete manifolds. Their method uses the tree-cotree decomposition [17], and is guaranteed to find the shortest system of loops centered at a given mesh vertex in $O(n \log n)$, and – by testing each point in the mesh – the globally shortest system of loops in $O(n^2 \log n)$. It is interesting to notice that while the computation of the shortest cut graph is NP-hard, the shortest system of loops is easy to compute. The difference between these entities relies in how lengths are computed: in a cut graph, each edge in the cut counts once; in a system of loops, each edge counts as many times as the number of loops in the system that traverse it. It follows that the systems of loops computed with the greedy homotopy basis algorithm are practically useful (because they are shortest), but do not allow to realize a canonical schema (because multiple loops snap to the same mesh edges), hence cannot be used to initialize a cross parameterization between two manifolds.

The use of mesh refinement to detach loops in a given homotopy basis is mentioned in a few aforementioned papers, although only [3] provided an actual algorithm. A similar method was possibly proposed already in [2], but the manuscript misses some technical details, and the algorithm is hardly reproducible. To the best of the author's knowledge, no alternative refinement schemes have ever been proposed in previous literature.

3 Problem Statement and Overview

Given a discrete manifold M with genus g, our objective is to generate a cut graph that realizes a canonical polygonal schema of M, enabling a map to a regular $4g-$gon. Our algorithm inputs M and a system of loops $L = \{\ell_0, \ell_1, \ldots, \ell_{2g}\}$. Loops in L are assumed to all emanate from the same origin $O(L)$, but may not be fully disjoint, thus violating the necessary condition to realize a canonical schema, that is

$$\bigcap_{\ell_i \in L} \ell_i = O(L)\,. \tag{1}$$

Our method outputs a refined manifold M' and a new system of loops L', such that L' satisfies Equation 1, and the refinement of M' is minimal. Without loss of generality, we assume that the input L is computed with the greedy homotopy basis algorithm [1]. This is just a practical choice to ensure that loops are shortest. The method works also if loops are not shortest paths, provided that if at some point two loops merge, they follow the same path until they reach the origin $O(L)$.

3.1 General Algorithm

To devise a refinement algorithm we start from a basic observation: loops in the system may be partially overlapping, but can never be entirely coincident. This is ensured by the fact that L is a system of loops in the sense of [15], hence it is also a cut graph of M. If two loops were coincident, $M \setminus L$ would not be a topological disk, thus L could not be a cut graph in the first place. It follows that if two loops share a portion of their path towards the origin of the system, there should be a mesh vertex where they begin to coincide. We call this point a *merging* vertex. Figure 3 (left) shows an example of merging vertex where two loops collapse into a single discrete path that takes to the origin of the system. Note that the number of loops incident to a merging vertex can be much higher (for a manifold with genus g the worst case scenario is $2g-1$). Moreover, each incoming path can be either a single loop or a bundle of multiple loops that already joined at a previous merging vertex. From a computational perspective there is no difference between these cases, single loops or bundles of loops can all be locally detached using the same refinement operators.

The main idea of the algorithm is to iteratively push each merging vertex one step forward towards the origin of the system of loops $O(L)$, until all merging points converge to it and Equation 1 is satisfied. In the initialization step, all the merging vertices in L are identified and stored in a queue Q. Then, merging vertices v_m are iteratively extracted and the mesh is locally refined, making sure that all incoming loops traverse the one ring of v_m along a dedicated path. After refinement, the merging point of all such loops has moved to a new mesh vertex which was originally in the one ring of the current v_m. If such a point is not the origin of the system of loops, it is added to the queue. The algorithm stops when Q is empty. At that point there won't be any merging vertex in L but $O(L)$, thus Equation 1 is satisfied, and a canonical polygonal schema of the refined manifold M' along the newly generated system L' can be computed (Figure 1). Note that the algorithm above does not provide any detail on how the local refinement is performed. There are several options, which produce different results in terms of number of new elements inserted in the mesh, and geometric distance between M and M'. In Section 4 we present all the possible alternatives, discussing pros and cons of each strategy.

4 Local Refinement Operators

In this section we explore all the alternative ways to split the elements of a simplicial mesh to detach a set of loops around a merging vertex. The basic ingredients for this operation are illustrated in Figure 4. The refinement strategy based on the edge split operator discussed in Section 4.1 was already presented in [3]. To the best of our knowledge, the alternatives presented in Section 4.2 and 4.4 are novel.

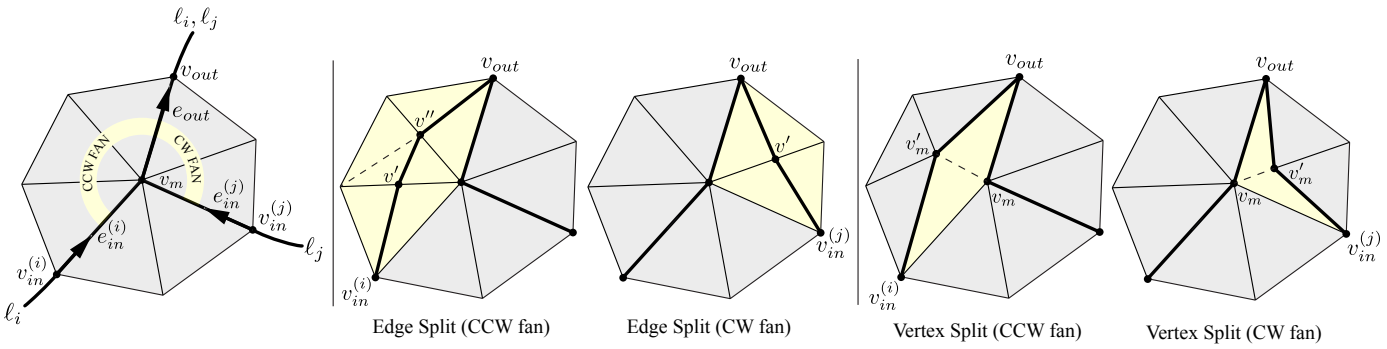

Fig. 3. Left: loops ℓ_i, ℓ_j meet together at a merging vertex v_m. From that point on, they travel together towards the origin of the loop system, $O(L)$. Edges incident to v_m that are traversed by ℓ_i, ℓ_j can be locally oriented such that there is one outgoing edge e_{out}, traversed by ℓ_i, ℓ_j, and two ingoing edges, traversed by one loop each (see black arrows). Rotating from e_{out} in both directions until the first ingoing edges are found defines two fans of mesh elements (CW and CWW). Middle: using the edge split to to locally detach ℓ_i and ℓ_j around v_m using the CWW fan and the CW fan. Right: same result, obtained using the vertex split operator.

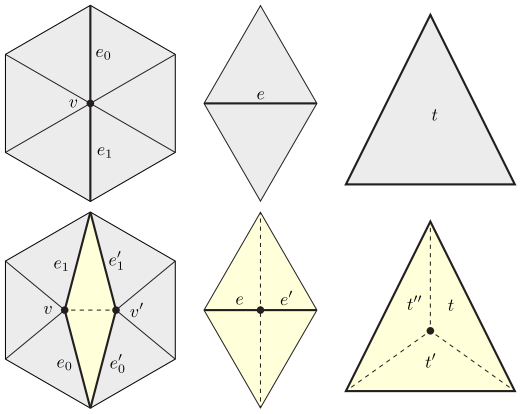

Fig. 4. The three possible refinement operators for a triangle mesh. Left: splitting vertex v along its incident edges e_0, e_1; middle: splitting edge e at its midpoint; right: split a triangle t into three sub triangles.

The typical configuration is the one shown in Figure 3, where two loops, ℓ_i, ℓ_j meet at merging vertex and, from that point on, proceed together towards the origin of the system of loops $O(L)$. Edges traversed by some loop can be locally oriented, such that there is one outgoing edge e_{out} that points towards the origin of the system, and two (or more) ingoing edges v_{in}, which all converge to the merging vertex. Rotating from the outgoing edge e_{out} clock-wise and counter clock-wise towards the first ingoing edges, defines two ordered fans of mesh elements. These are the two alternative domains that can be used to locally refine the mesh, defining two disjoint paths for ℓ_i and ℓ_j within the umbrella of their merging vertex v_m. In the following sub-sections we will detail how each splitting operator can be used to perform such operation.

4.1 Edge Split

Considering a merging vertex v_m and the ordered fan of edges $E = \{e_1, \dots, e_n\}$ in between an ingoing edge e_{in} and an outgoing edge e_{out}, a unique path connecting the associated ingoing vertex v_{in} and the outgoing vertex v_{out} can be obtained by splitting all edges in E. Denoting with v_i the splitting point of edge e_i, the path $\{v_{in}, v_0, \dots, v_n, v_{out}\}$ is entirely defined within the triangle fan span by E, and is also completely disjoint from any other path connecting v_{in} and v_{out}. Figure 3 (middle) shows its application to the CCW and CW fans of edges around the merging point v_m. Note that splitting the CCW edge fan introduces 2 new vertices and 4 new triangles, whereas splitting the CW edge fan introduces 1 new vertex and 2 new triangles. In the general case, the mesh grows linearly with the size of the edge fan, and the growth amounts to $|E|$ new vertices, and $2|E|$ new triangles. Since there are always two alternative edge fans to be split (CW or CCW), to minimize mesh growth it is preferable to always split the fan with smallest size. Note that the edge split requires that $|E| > 0$. If the fan of elements in between e_{in}, e_{out} contains only one triangle and zero edges, loops can be locally split only with the vertex or the triangle split operators.

4.2 Vertex split

Considering the same edge fan $E = \{e_0, e_1, \dots, e_n\}$ around a merging vertex v_m, a unique path connecting v_{in} and v_{out} can also be obtained by splitting v_m along the ingoing and outgoing edges e_{in}, e_{out} that bound E. Figure 3 (right) shows an application of this refinement scheme to the CCW and CW fans around the merging point v_m. Note that in both cases the number of new mesh elements amounts to 1 new vertex and 2 new triangles. Differently from the edge split case, this growth is invariant and does not depend on the local complexity of the mesh. Although preferable from a topological point of view, the vertex split operator has a geometric limitation: depending on the geometry of the mesh, the two new triangles incident to the new edge (v_m, v'_m) will not adhere to the original mesh, introducing a deviation from the target geometry. An example of failure case is depicted in Figure 5. In general, any time the fan of triangles span by E is not planar, the vertex split operator introduces such a deviation.

4.3 Triangle split

Differently from the edge split and the vertex split operator, the triangle split operator can be used to locally detach a pair of loops if and only if the ingoing and the outgoing edges share the same triangle. In that case, adding a new vertex

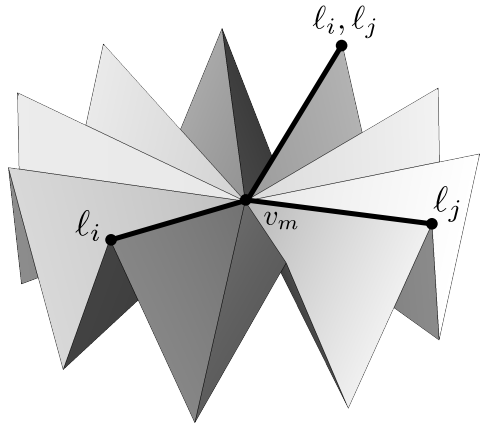

Fig. 5. Detaching loops ℓ_i, ℓ_j around their merging vertex v_m using the vertex split operator unavoidably deviates from the mesh. Denoting with v'_m the split copy of v_m, the two triangles incident to edge (v_m, v'_m) deviate from the reference geometry, unless v_m and v'_m coincide. In such case, both triangles will be degenerate.

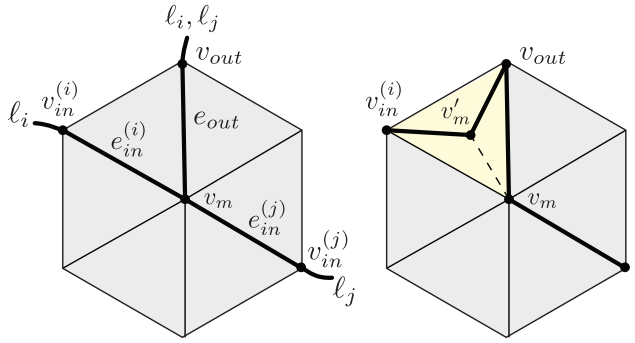

Fig. 6. Splitting triangle $v_{in}^{(i)}, v_m, v_{out}$ locally detaches loops ℓ_i, ℓ_j. Note that ℓ_i is not a shortest loop, because $|v_{in}^{(i)} - v_{out}| < |v_{in}^{(i)} - v_m| + |v_m - v_{out}|$. Also note that the triangle split is conceptually equivalent to splitting vertex v_m along the edges $e_{in}^{(i)}, e_{out}$.

inside the triangle and connecting it to the three corners with new edges generates an alternative path from the ingoing vertex v_{in} and the outgoing vertex v_{out}, without passing from the merging vertex v_m (Figure 6). Note that this operation is equivalent to performing a vertex split of v_m along the edges e_{in}, e_{out}. Also note that if the input system of loops is shortest – as in the case of [1] – this configuration will never occur. If fact, due to the triangular inequality

$$|v_{in} - v_{out}| < |v_{in} - v_m| + |v_m - v_{out}|$$

the path $\{v_{in}, v_{out}\}$ will always be shorter than the path $\{v_{in}, v_m, v_{out}\}$, hence v_m would not be a merging vertex. Considering its limited applicability and the fact that, even when usable, the triangle split is equivalent to the vertex split, this is not a suitable operator to detach loops in a cut graph.

4.4 Hybrid split

Considering the ability of the vertex split to introduce less elements in the mesh, but also its lack of geometric fidelity if the edge fan is not locally planar, we introduce here a fourth option which consists in a hybrid operator that uses as many vertex splits as possible to reduce mesh growth, and switches to the costly edge split to avoid excessive deviation from the surface. The method is extremely simple, and seamlessly integrates in the global detaching algorithm described in Section 3.1. Given a merging vertex, we first check whether the CW or CCW fans of elements aside the outgoing edge are planar. If so, we split the merging vertex along the ingoing and outgoing edges that bound such fan (Section 4.2). If none of the fans are roughly planar, we locally refine the mesh using the edge split operator (Section 4.1.)

To measure planarity we simply consider the maximum angle between the normals $\mathbf{n}_i, \mathbf{n}_j$ of two triangles i, j in the fan of triangles

$$\arg\max_{i,j} \angle(\mathbf{n}_i, \mathbf{n}_j) \tag{2}$$

If the angle above evaluates zero, the fan is planar and the vertex split operator can be used without introducing any deviation from the input surface. In all other cases some deviation from the reference geometry will occur. Assuming the mesh is planar and the vertex split operator is used, positioning the new vertex v'_m in the one ring of the merging vertex v_m is also critical to ensure that no triangle will flip its orientation in the refined mesh. Making sure that v'_m stays inside the polygon defined by the boundaries of the edge fan is not enough, because such polygon may be non convex (e_{in} and e_{out} may form a concave angle). We practically solve this issue by initializing the new point as

$$v'_m = (1 - \lambda)v_m + \lambda v_e$$

where λ is initially set to 0.75, and v_e is the vertex opposite to v_m along the edge e, which is median in the edge fan being split. If any of the triangles incident to v'_m is flipped, we halve λ and update its position, until a valid position is found. Such a position always exists if e_{in}, e_{out} do not coincide.

In practice, we use the vertex split operator even when the fan of elements is roughly planar. To do so we simply test Equation 2 with a threshold angle of 5 degrees, which we empirically found to provide a good balance between the amount of vertex splits executed and surface deviation. Users can easily trade mesh size for geometric fidelity by acting on this parameter.

5 Results and Discussion

In this section we analyze the performances of the three refinement strategies presented in Section 4. We considered a set of 75 triangle meshes, mostly gathered from the Thingi10K dataset [18]. Since we are interested in the scalability of refinement operators we focused our analysis on high genus meshes, including just a few low genus models for completeness. Overall, the genus g varies from 2 to 632 (with average 157), and the initial mesh size varies from 1K to 370K vertices (with average 35K). Our experimental setup is as follows: for each model we first compute a generic system of loops centered at a random mesh vertex with [1]. We then apply the three refinement algorithms to detach all loops except at their basis, producing three alternative cut graphs that admit a canonical polygonal schema (Figure 7). Assuming an infinite amount of memory, all the refinement operators are guaranteed to converge to a canonical cut

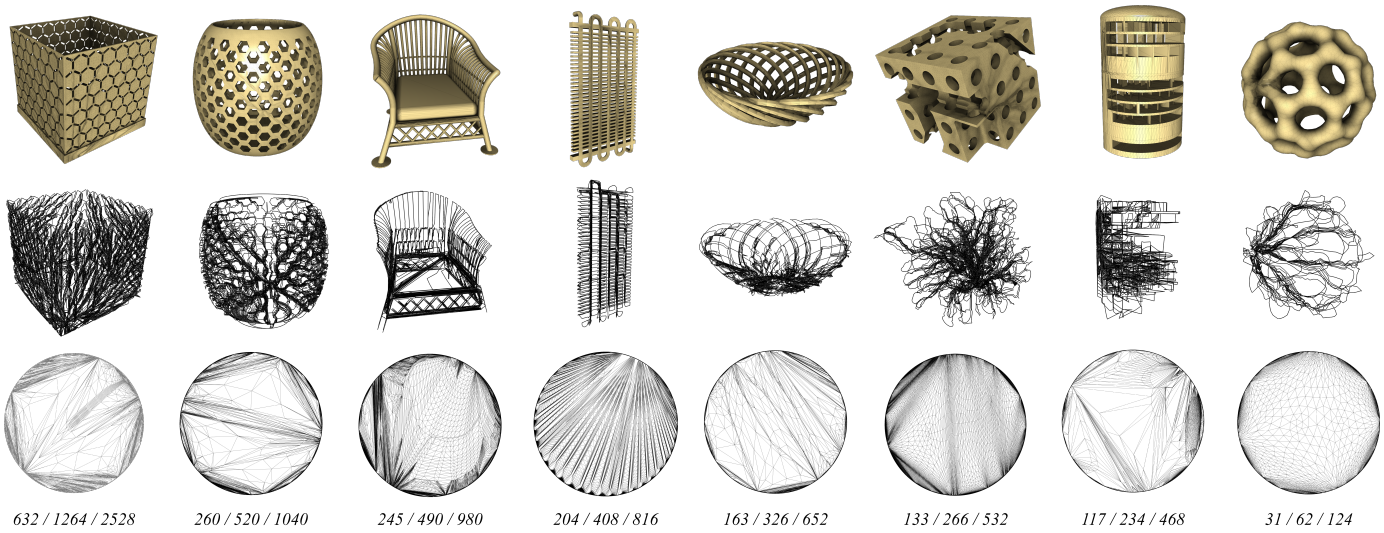

Fig. 7. A few results from our testing dataset, obtained by applying the hybrid splitting scheme. From top to bottom we show: input shape, system of loops, and associated canonical polygonal schema (using the Tutte embedding for interior points). The three numbers at the bottom report: genus, number of loops in the system, sides of the canonical polygon.

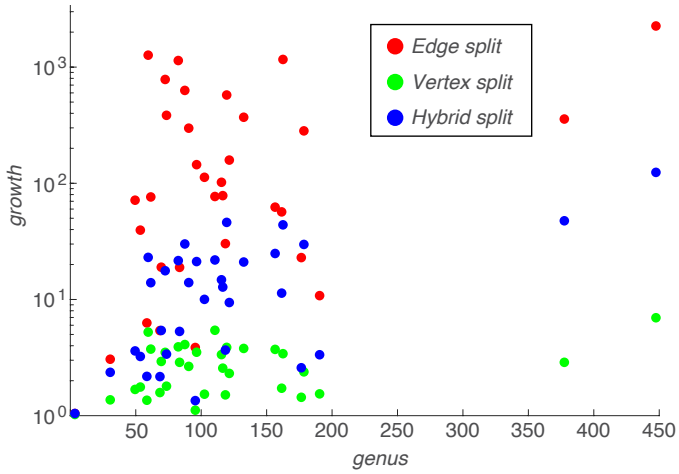

Fig. 8. Scatter plot of all the models in our test set that were successfully processed with all three refinement operators. Mesh growth is measured as the ratio between the number of points in the refined mesh and in the input mesh. To better inspect results and differences between the various approaches we use a log scale for the vertical axis.

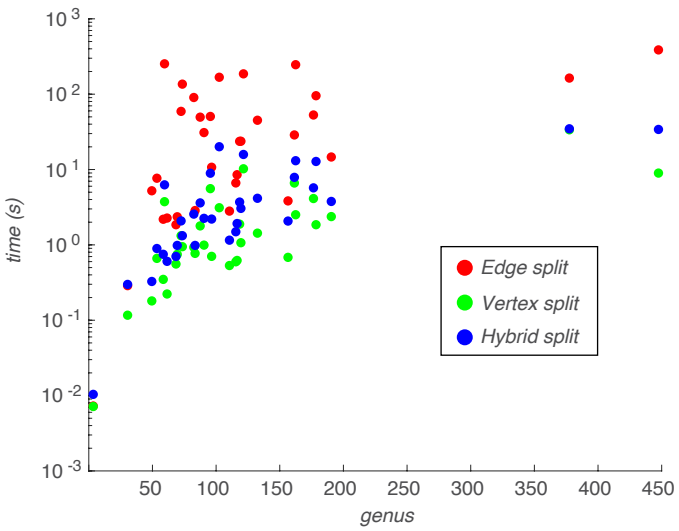

Fig. 9. Running times for all the models in our test set that were successfully processed with all three refinement operators. The vertical axis is in log scale.

graph. We run our software prototypes as single threaded applications on a MacBook Pro with 16GB of RAM, aborting processes that consume a bigger amount of memory and considering them as failure cases.

In terms of mesh growth, experimental results confirm our hypothesis and clearly show the ability of the vertex split to introduce far less new mesh elements than the edge split (Figure 8). With the edge split operator the ratio between the number of vertices in the refined mesh and in the original one stayed in between 1.03 and 2229 (from 5.5K to 12.2M vertices for a mesh with genus 448). Conversely, for the vertex split operator the same ratio is always lower than 7.5 (from 5.7K to 41.4K vertices for a mesh with genus 451). Moreover, the edge split strategy started to consistently fail for meshes with genus higher than 180, with only three exceptions ($g = 191, 378, 448$). Overall, we counted a total of 19 failures (25% of the dataset), while the vertex split succeeded in producing a valid cut graph for all the meshes in the dataset, always using less than 10GB of RAM. The hybrid split stays in between, with a vertex ratio that grew up to 476 (from 9.7K to 64.6K vertices for a mesh with genus 568), and failed on three very high genus shapes ($g = 358, 451, 697$), succeeding on all the others (4% failure rate). Overall, the average vertex ratio was 2.66 for the vertex split, 38.66 for the hybrid split, and 240.18 for the edge split.

Running times are heavily influenced by mesh growth. The vertex split refinement is the fastest to compute, followed by the hybrid split and with the edge split at least one order of magnitude slower (Figure 9). Overall, timings vary from 0.007s to 32.9s for the vertex split (average 2.3s), from 0.007s to 431s for the edge split (average 61.4s), and from 0.1s to 34s for the hybrid split (average 4.9s).

As can be noticed from the growth and time analysis numbers fluctuate, and do not monotonically grow with

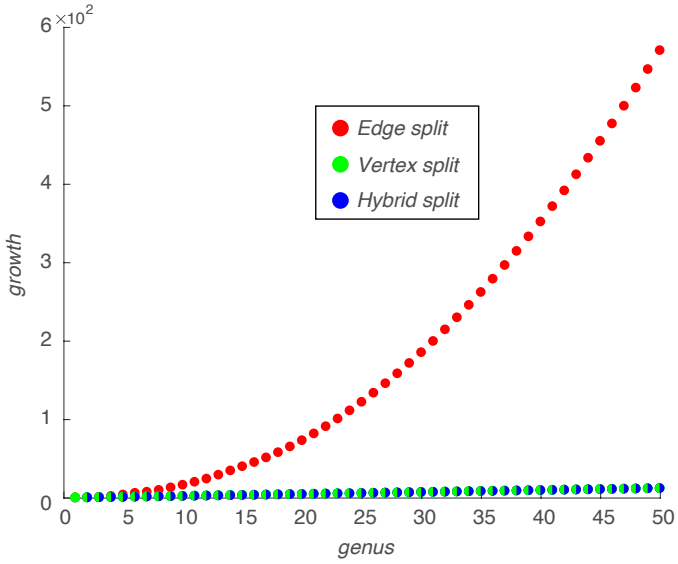

Fig. 10. Mesh growth obtained by using the edge split and vertex split operators to refine a greedy homotopy basis of a sequence of polycubes with increasing genus.

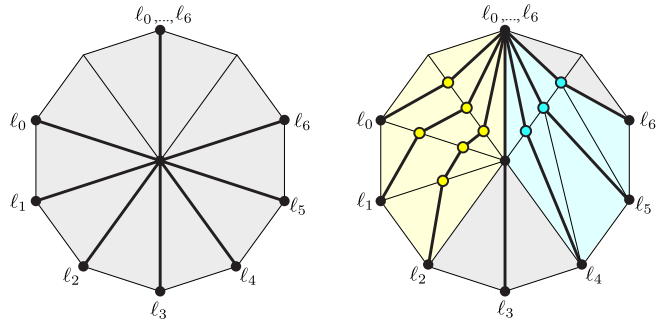

Fig. 11. Processing of a dense merging vertex: if the vertex split operator is used (CW fan, cyan area) detaching n loops requires the insertion of n new mesh vertices. If the edge split operator is used (CWW fan, yellow area) detaching n loops requires the insertion of $n(n+1)/2$ new mesh vertices.

H	1×10^{-4}	3×10^{-4}	6×10^{-4}	1×10^{-3}	3×10^{-3}	6×10^{-3}
%	3.1	18.7	47.1	70.8	92.1	99.9

TABLE 1
Average results obtained by processing all models in our dataset with a modified hybrid operator that always attempts to use a vertex split, and switches to the edge split if the newly inserted vertex would increase the Hausdorff distance H above a given threshold. In the top line we indicate the threshold on surface deviation, and in the bottom line the average percentage of vertex splits executed w.r.t. the total amount of splits.

the genus. Although this might seem counter intuitive, it is actually not surprising and can be explained by observing that the mesh genus is just one the factors that impact the performances. The other factors are: the initial resolution of the mesh, the type of connectivity (i.e. the distribution of vertex valences), the geometry, and also the position of the root of the system, which affects how shortest loops wind around the handles and how much they tend to accumulate on specific areas of the mesh, requiring more or less refinement to detach them.

To better isolate the impact of mesh genus we also conducted a second experiment: we constructed a sequence of synthetic meshes with growing genus, obtained by concatenating multiple occurrences of the polycube shown in Figure 2. We fixed the root of the system of loops at a prescribed vertex, and applied the three mesh splitting schemes for a sequence of meshes going from genus 1 to 50. Results are presented in Figure 10 and clearly show that the vertex split operator grows linearly with mesh genus, whereas the edge split has a super quadratic growth. This difference in the asymptotic behaviour can be explained by observing that to detach n loops around a merging vertex the edge split may introduce up to $n(n+1)/2$ new vertices, whereas the vertex split always introduces n vertices (Figure 11). Moreover, if the portion of loop connecting the merging vertex with the origin of the system counts m vertices, the same operation will be repeated at each such point, leading to a $O(mn^2)$ complexity for edge split, and $O(mn)$ for the vertex split.

For the geometric fidelity, the edge split is the only operator that guarantees no deviation from the reference geometry. For the vertex split, we observed that when many loops concentrate in a single chain of edges and the refinement insists on the same area, surface deviation accumulates, creating visible artifacts. In the worst case we found that the Hausdorff distance was 0.2 w.r.t. the diagonal of the bounding box, whereas the average distance was 3e-2 across all the runs. The hybrid split allows to better control surface deviation, and in all our tests the average and Hausdorff distances were consistently below 4e-5 and 1e-3, respectively. Even though our heuristic based on the planarity check proved to be effective at bounding surface deviation, there are countless alternative options to combine edge and vertex splits to reduce mesh growth while satisfying some user prescribed requirement. We chose the planarity check because it naturally preserves sharp creases and the local smoothness or planarity of the shape (e.g. for the polycube case).

Another natural alternative consists in directly bounding the Hausdorff distance, always attempting a vertex split, and reverting it switching to the edge split if the surface deviation grows above a given threshold. We implemented this option using VCG [19] to compute the two sided Hausdorff distance, and tested it against all the models in our dataset. In Table 1 we report the average percentage of vertex splits successfully executed (i.e. not reverted) for growing bounds on the maximal distance between the input and refined meshes.

6 Conclusions and Future Works

We showed that although theoretically correct, detaching cycles in a system of loops using the edge split operator triggers a mesh growth that explodes with genus, leading to numerous failures in practice. In alternative we propose two novel refinement operators. The first one simply substitutes the edge split with a vertex split, obtaining much better performances in terms of running times, memory footprint and mesh growth, at the cost of a possible deviation from the reference geometry. The second one addresses this limitation, and proposes to use as many vertex splits as possible, switching to the costly edge split only when significative surface deviation occurs. Numerous valid heuristics could be used to decide how to combine these operators. We showed the effectiveness of a heuristic based on a testing local mesh

planarity, and also implemented an alternative strategy based on a direct control on the Hausdorff distance.

We support our claims with a variety of results, obtained on discrete manifolds that span from low to extremely high genus, and from smooth to CAD-like shapes. The proposed algorithms are based on well established local operators for simplicial meshes. These operators are already implemented in many geometry processing toolkits, making our results easy to reproduce. Nevertheless, we release a reference implementation of all the splitting methods presented in this paper (including the basic edge split strategy) inside Cinolib [20].

Despite conceptually simple, we believe that this work makes one step forward towards the robust and computationally affordable generation of cross maps between complex shapes. Interesting results have already been presented for disk-like topologies [21], and we expect more and more papers to come in future years. In the same spirit of recent works for the robust computation of planar maps, which start with Tutte's embedding and then cure distortion [22], [23], [24], we foresee a similar pipeline for cross maps between shapes, where manifolds are first cross mapped via their canonical schema, and then the polygon is evolved to minimize distortion. Note that this problem is much harder: partly because distortion minimization should consider the composition of two maps that overlap to one another, but more importantly because there are $4g$ alternative ways to overlap two $4g-$gons (i.e., which handle maps to which?), which makes it a problem with mixed discrete and continuous degrees of freedom for which, to the best of our knowledge, no effective solution is available in literature.

REFERENCES

[1] J. Erickson and K. Whittlesey, "Greedy optimal homotopy and homology generators," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2005, pp. 1038–1046.

[2] G. Vegter and C. K. Yap, "Computational complexity of combinatorial surfaces," in *Proceedings of the sixth annual symposium on Computational geometry*. ACM, 1990, pp. 102–111.

[3] X. Li, Y. Bao, X. Guo, M. Jin, X. Gu, and H. Qin, "Globally optimal surface mapping for surfaces with arbitrary topology," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 4, pp. 805–819, 2008.

[4] C. Carner, M. Jin, X. Gu, and H. Qin, "Topology-driven surface mappings with robust feature alignment." in *IEEE Visualization*, vol. 2005, 2005, pp. 543–550.

[5] H. Wang, Y. He, X. Li, X. Gu, and H. Qin, "Polycube splines," in *Proceedings of the 2007 ACM symposium on Solid and physical modeling*. ACM, 2007, pp. 241–251.

[6] T. K. Dey, K. Li, J. Sun, and D. Cohen-Steiner, "Computing geometry-aware handle and tunnel loops in 3d models," in *ACM SIGGRAPH 2008 papers*, 2008, pp. 1–9.

[7] F. Lazarus, M. Pocchiola, G. Vegter, and A. Verroust, "Computing a canonical polygonal schema of an orientable triangulated surface," in *Proceedings of the seventeenth annual symposium on Computational geometry*. ACM, 2001, pp. 80–89.

[8] M. Livesu, N. Vining, A. Sheffer, J. Gregson, and R. Scateni, "Polycut: Monotone graph-cuts for polycube base-complex construction," *ACM Transactions on Graphics (Proc. SIGGRAPH ASIA 2013)*, vol. 32, no. 6, 2013.

[9] J. Erickson and S. Har-Peled, "Optimally cutting a surface into a disk," *Discrete & Computational Geometry*, vol. 31, no. 1, pp. 37–59, 2004.

[10] T. K. Dey and H. Schipper, "A new technique to compute polygonal schema for 2-manifolds with application to null-homotopy detection," *Discrete & Computational Geometry*, vol. 14, no. 1, pp. 93–110, 1995.

[11] A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogomyakov, "Abf++: fast and robust angle based flattening," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 2, pp. 311–330, 2005.

[12] P. Alliez, M. Meyer, and M. Desbrun, "Interactive geometry remeshing," in *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3. ACM, 2002, pp. 347–354.

[13] G. Taubin and J. Rossignac, "Geometric compression through topological surgery," *ACM Transactions on Graphics (TOG)*, vol. 17, no. 2, pp. 84–115, 1998.

[14] V. Kraevoy and A. Sheffer, "Cross-parameterization and compatible remeshing of 3d models," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 861–869, 2004.

[15] É. C. De Verdière and F. Lazarus, "Optimal system of loops on an orientable surface," *Discrete & Computational Geometry*, vol. 33, no. 3, pp. 507–534, 2005.

[16] M. Jin, J. Kim, F. Luo, and X. Gu, "Discrete surface ricci flow," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 5, pp. 1030–1043, 2008.

[17] D. Eppstein, "Dynamic generators of topologically embedded graphs," in *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2003, pp. 599–608.

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

[19] Visual Computing Lab, *The VCG Library*, Italian National Research Council - ISTI, 2018. [Online]. Available: http://vcg.isti.cnr.it/vcglib/

[20] M. Livesu, "cinolib: a generic programming header only c++ library for processing polygonal and polyhedral meshes," *Transactions on Computational Science XXXIV*, 2019, https://github.com/mlivesu/cinolib/.

[21] P. Schmidt, J. Born, M. Campen, and L. Kobbelt, "Distortion-minimizing injective maps between surfaces," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–15, 2019.

[22] M. Rabinovich, R. Poranne, D. Panozzo, and O. Sorkine-Hornung, "Scalable locally injective mappings," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 37a, 2017.

[23] J. Smith and S. Schaefer, "Bijective parameterization with free boundaries," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 70, 2015.

[24] L. Liu, C. Ye, R. Ni, and X.-M. Fu, "Progressive parameterizations," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, p. 41, 2018.

Marco Livesu is a tenured researcher at the Institute for Applied Mathematics and Information Technologies of the National Research Council of Italy (CNR IMATI). He received his PhD at University of Cagliari in 2014, after which he was post doctoral researcher at the University of British Columbia, University of Cagliari and CNR IMATI. His main research interests are in computer graphics and geometry processing.