EUROGRAPHICS 2026 / B. Masia and J. Thies
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 45 (2026), Number 2

DiskScissors: Cutting Arbitrary-Topology Solids for Bijective Mapping

S. Hinderink and M. Campen

Paderborn University, Germany

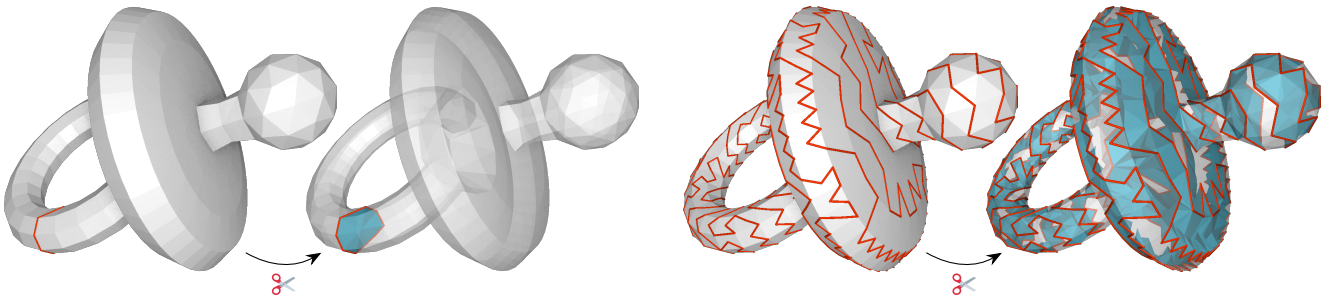


Figure 1: *Given a loop (red) on the surface of a solid (discretized as a tetrahedral mesh) of arbitrary topology (here genus $g = 1$), our algorithm constructs a disk-topology surface (blue) inside the solid, bounded by the loop, that cuts the solid. As demonstrated on the right, with an extremely convoluted loop (actually homotopic to the one on the left), this reliably holds regardless of the problem's geometric complexity: the resulting cut surface again is of disk topology, even if hard to see visually here.*

Abstract

An algorithm for cutting solid objects in a topology-controlled manner is presented. Concretely, given a loop on the object boundary, a disk-topology cut surface bounded by the loop is constructed in the interior. In contrast to various previous approaches, both disk topology and conformance to the prescribed loop are ensured by construction, while supporting not only contractible but also incontractible loops on the boundaries of manifold objects of higher genus and arbitrary non-trivial topology. We describe an implementation of this algorithm in the discrete setting, with triangle mesh cut surfaces embedded in tetrahedral mesh objects. Making use of this novel cutting algorithm, we describe a method for the reliable construction of bijective volumetric maps between solid objects, demonstrating the algorithm's utility. This mapping method overcomes restrictions of the state of the art to topological balls, extending coverage to objects of arbitrary genus, specifically so-called 1-handlebodies.

CCS Concepts

• ***Computing methodologies*** → ***Mesh models; Volumetric models;***

1. Introduction

Cutting is a basic operation in geometry processing and generally when dealing with geometric objects. As such it has many applications. This work is specifically concerned with cutting *solid* objects. As is apparent from pertinent literature, this is relevant for, e.g., object decomposition or segmentation [XGZ11, GMD*16], mesh generation [Tak19, BC23], spline modeling [LLWQ13], and volumetric mapping [HBC24], the latter in turn being relevant for parametrization, deformation, or structure transfer, to name a few.

Although it seems like an elementary problem, cutting in a way that reliably satisfies the concrete needs of an application can be difficult depending on the input as well as on the output requirements. Relevant desiderata are:

- control over the cut surface's topology,
- control over the cut surface's boundary geometry,
- no restrictions regarding the solid's topology.

In particular, all of the above mentioned works require cut surfaces of *disk* topology for their use cases.

We present an algorithm that, for a given loop on the boundary of a solid, reliably computes a surface of disk topology within the solid, bounded exactly by the prescribed loop, provided it exists. Solids of arbitrary manifold topology are supported, in particular there are no restrictions regarding their genus.

In recent work an algorithm to compute boundary-constrained disk-topology cut surfaces was proposed [HBC24]. It is fundamentally limited, though, to loops that are contractible on the boundary, which is generally the case only for ball-topology solids. Our algorithm's central principle is reducing the general problem to multiple instances of this simpler contractible-loop-on-ball-topology-solid scenario. We then solve these instances using an extended variant of the above work's *surface shift* approach, and combine them to a solution of the general problem. We describe an implementation of this algorithm in a discrete setting. The solid is represented by a tetrahedral mesh, the cut surface is a triangle mesh discretely embedded in the tetrahedral mesh.

Moreover, we present an application of this cutting algorithm in a volumetric mapping method, demonstrating its utility. Concretely, while recent volumetric map construction methods which offer certain guarantees regarding map bijectivity are all limited to solids of ball topology [HC23, NCB23, HBC24, NCB24], by means of our reliable cutting algorithm we are able to extend one of these to support solids of arbitrary genus, more precisely 1-handlebodies (defined in Section 4.1).

After discussing related work on solid cutting and mapping in Section 2, Section 3 deals with the cutting algorithm, while its application in the context of bijective mapping is described in Section 4. An empirical evaluation follows in Section 5.

An open source implementation of the algorithm is available online at `https://github.com/SteffenHinderink/DiskScissors`.

2. Related Work

2.1. Cutting

The problem of constructing surfaces within solids has been addressed in quite a variety of ways in previous works, often to perform some form of cutting, segmentation, or decomposition. We discuss these in the following, pointing out how each of these does not meet one or more of the three above mentioned key desiderata: cut topology control, cut boundary control, arbitrary-topology solid support.

Ambient explicit surfaces One possibility is to construct in ambient space (essentially ignoring the solid) some disk-topology mesh bounded by a given boundary loop, and then extrinsically fair it – as done for solid partitioning for spline domain construction [LLWQ13, §4.2]. While this trivially ensures that the mesh is a disk and matches the prescribed loop, it does not necessarily lie entirely within the solid. The implied cut surface, i.e. that part of the mesh actually intersecting the solid, may be of non-disk topology, may have a boundary that differs from the prescribed loop, and may self-intersect.

Ambient implicit surfaces Another option is to define the surface, again in ambient space, as a level set of an implicit function, modeled as, e.g. Hermite thin-plate splines, as done for block-structured hexahedral meshing [Tak19, §4.1], hex-dominant meshing [LPP*20], and for isogeometric analysis using hexahedral segmentations [HJ17]. Due to their ambient nature, again there is no guarantee regarding the resulting cut surface topology and its boundary. Furthermore, a given boundary loop (or general curve) is generally matched only approximately.

Intrinsic implicit surfaces Other approaches make use of implicit functions defined on a discretization of the solid, in particular a tetrahedral mesh. A popular choice is a discrete harmonic field, obtained by solving the discrete Laplace equation on the mesh, subject to constraints – as done for structured volume decomposition for parametrization [GMD*16, §3.2] and mapping [HBC24, §5.2.1]. Using suitable constraints, the (discrete) zero-isosurface of this field can be forced to meet the solid's boundary (at least) along a prescribed loop (or general curve). Unfortunately, the isosurface's topology is uncontrolled [GS25]; it cannot, e.g., be constrained to disk topology.

Intrinsic explicit surfaces The surface can also be computed as some form of minimal surface, discretely embedded in the tetrahedral mesh, spanning given boundary loops. This can be approached using graph partitioning techniques [XGZ11, §4.3, LAPS17, §6] or using (relaxed integer) linear program (LP) formulations [Gra06]. Unfortunately, also methods based on intrinsic minimal surfaces do not offer control over or guarantees regarding the resulting cut surface's topology [DHS10].

Surface shifting Lastly, in case a given loop bounds a disk in the solid's boundary, this disk-topology subset of the boundary can be used as initialization for the cut surface. This initial surface can then be discretely shifted into the interior of the solid, keeping only the boundary loop fixed and maintaining topology [BC23, §6.1.2, HBC24, §5.2.2]. This guarantees boundary loop conformance and disk topology. However, since this relies on the initialization already having disk topology, this approach is restricted to loops that are contractible on the solid's boundary; the particularly important case of cutting handles of higher-genus solids (as in Figure 1), for instance, is not covered by this approach.

2.2. Mapping

Injective and bijective maps between shapes, or between shapes and parameter domains, are a fundamental tool in many applications. While for the 2D/surface case the construction and optimization of such maps are well-understood problems, with relatively simple sound solutions such as Tutte's embedding, the matter is more intricate in the 3D volumetric case.

Variational Many volumetric mapping methods aim for bijectivity by optimizing for some injectivity or bijectivity promoting distortion objective function defined on a tetrahedral mesh [NZZ20, DAZ*20, GKK*21, DKZ*22, JWP*22, ASGS23, POK23]. While supporting solids of arbitrary topology, these optimization-driven approaches do not guarantee yielding a bijective result.

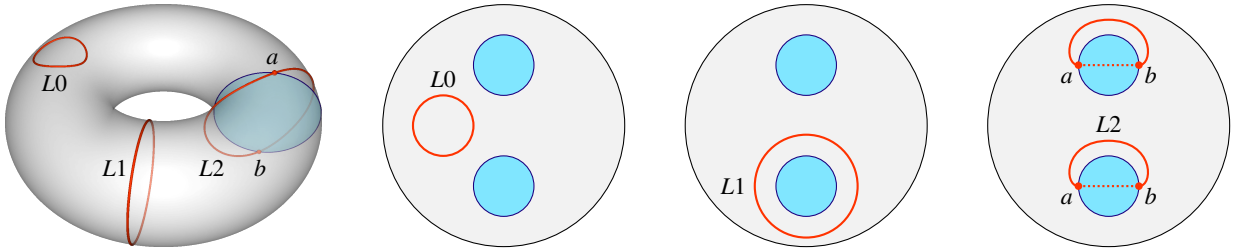


Figure 2: *Three loops (L0, L1, and L2, red) on the boundary of a solid torus M (left). A virtual cut (blue) yields a simply connected boundary B; three copies of a 2D embedding of B (punctured at some point) in 2D is shown on the right to clearly depict the configuration of the individual loops. The two (opposite) inner parts of B, that lie along the two sides of the virtual cut, are marked in blue. L0 bounds a disk on $\partial M \subseteq B$, which can be shifted inside (based on M) to obtain disk $\Delta 0$. L1 bounds a disk on B, which can be shifted inside (based on the virtually cut M) to obtain $\Delta 1$. L2 is split into two paths on B at the vertices a and b. The algorithm connects the open ends across the blue inner parts (dotted curves). This yields multiple loops that bound disks on B that all can be shifted inside to combinedly form disk $\Delta 2$.*

Combinatorial On the other hand, bijective volumetric mapping methods based on combinatorial principles have been described for increasingly general target shapes (alongside combinatorial map construction methods for the 2D case [Liv24a, Liv24b, FBRCA23, CFLF25]). Maps to spheres and cubes can be defined through foliations [CSZ16, HG15]. Extension to more general star-shaped targets is achieved by expanding on this idea and locally applying it in so-called galaxies [HC23] or contracting all inner vertices and then expanding them within certain cones [NCB23, NCB24]. By compatibly decomposing source and target into star-shaped components, multiple such maps can be assembled to a map between arbitrary shapes of ball topology [HBC24]. While offering bijectivity-related guarantees, all these methods are restricted to (subclasses) of ball-topology solids.

Our cutting algorithm allows generalizing the latter method to arbitrary genus solids, formally 1-handlebodies. In essence, using the algorithm we are able to reliably perform the compatible solid decomposition into star-shaped components on this broad class of objects.

3. Cutting

3.1. Problem Statement

We begin by formally and precisely defining the problem setting.

Let S be a pure geometric simplicial d-complex in 3D. If $d = 3$, S is a *tetrahedral mesh*. If $d = 2$, it is a *triangle mesh*. If $d = 1$, it is an *edge mesh*. The *carrier* of S is the union of all its simplices, $[S] = \bigcup_{s \in S} s \subseteq \mathbb{R}^3$. The *closure* $\langle \cdot \rangle$ of a subset of simplices of S is the smallest simplicial complex containing them; for instance, the closure of a set of 2-simplices (triangles) includes their incident 1-simplices (edges) and 0-simplices (vertices). The *boundary* ∂S is the closure of all $d-1$-dimensional simplices in S that are faces of only one d-dimensional simplex in S.

An edge mesh with a connected and 1-manifold carrier is a *path*. A path without boundary is called a *loop*. We call a loop L *contractible* in a mesh $S \supseteq L$ if its carrier $[L]$ is contractible in $[S]$ in the topological sense [Hat02].

Input The input is a tetrahedral mesh M, such that $[M]$ is 3-manifold with boundary, together with a prescribed loop $L \subseteq \partial M$ contractible in M.

Output The output is a triangle mesh $\Delta \subseteq M'$, discretely embedded in a possibly refined version M' of M, such that $\partial M' \cap \Delta = \partial \Delta = L$ and $[\Delta]$ has *disk topology*.

Note that loops that are not topologically contractible *in* M do not permit a solution by definition. If M consists of multiple connected components, only the one containing L is relevant to solve the problem; hence, w.l.o.g. we assume M to be this single connected component in the following.

3.2. Approach Overview

The *surface shift* technique detailed by Hinderink et al. [HBC24, §5.2.2] following an idea of Brückler and Campen [BC23, §6.1.2] solves this problem for the special case of L bounding a disk in ∂M. We provide a recap of this technique in Section 3.5. This is the case iff the loop is contractible on the boundary, such as $L0$ in Figure 2. We call such loops *trivial* in the following.

But on objects with boundaries of genus $g > 0$, there are *non-trivial* boundary loops, that do not bound disks on the boundary. Our algorithm's underlying idea to deal with this is two-fold. First, we virtually cut the solid M such that the resulting boundary $B \supseteq \partial M$ is simply connected, of sphere-topology. Intuitively, one can think of B as the membrane of a balloon (of genus 0) that is maximally inflated inside M. Every loop on ∂M that also forms a loop on this new boundary B, e.g. $L0$ and $L1$ in Figure 2, is contractible on B and therefore can be handled by applying the surface shift technique on the virtually cut version of M.

Second, we note that loops on M may cross the virtual cut, such as $L2$ in Figure 2, thus may form not a loop on B. In such cases the sought disk surface Δ must also cross the virtual cut, i.e. it must intersect B in the interior of M. We algorithmically determine a suitable intersection pattern on B, creating paths on the inner parts of B. These paths close the partial loops that the original loop is split into by the virtual cut, forming multiple loops on B. The above

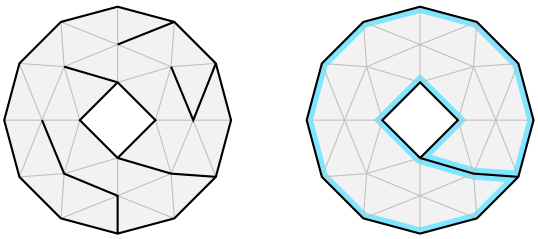


Figure 3: *2D example for the virtual cut. The edges in F are marked in black, before (left) and after (right) the pruning step. The resulting boundary B is indicated in blue.*

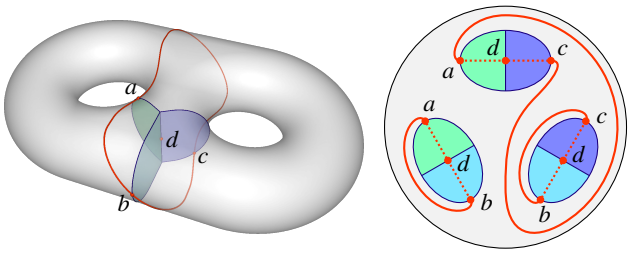


Figure 4: *A simply connected boundary B with three sides and two parts per side. Pairs of opposite parts are marked in the same color. The loop L is split into three paths on B at the vertices a, b, and c. The connecting paths (dotted curves) cross a copy of the vertex d each.*

technique can then be applied to each of these, such that the resulting disks, Δ_i, together form $\Delta = \bigcup_i \Delta_i$.

These two key ingredients, ball cutting and loop closing, are detailed in Section 3.3 and Section 3.4, respectively, followed by the adaptation of the surface shift technique to this setting in Section 3.5.

3.3. Ball Cutting

First, M is virtually cut to ball topology, yielding a simply connected boundary B. The procedure is similar to algorithms for cutting surfaces to disk topology [GGH02, LBHH23].

3.3.1. Balloon inflation

Pick an arbitrary tetrahedron in M, mark it as visited, and initialize a set F, conceptually the initial balloon membrane, with its four triangles. F is then inflated discretely by iterating the following: Let c be a tetrahedron adjacent to a visited one and f the shared triangle between them. The triangles of c are added to F, f is removed from F, and c is marked as visited. This is repeated until all tetrahedra have been visited. Equivalently, F can be obtained as the set of triangles of M that are not crossed in a breadth-first search, starting from the seed tetrahedron, on the dual graph of M.

By construction, the solid region enclosed by the triangles F maintains its initial (ball) topology throughout the process. Hence, in the end, when all tetrahedra are enclosed, $[M] \setminus [\langle F \rangle]$ is of ball topology. In other words, $\langle F \rangle$ cuts M to ball topology. Figure 3 illustrates the outcome on a 2D example.

Pruning Many triangles can be removed from F without changing the topology of the cut solid $[M] \setminus [\langle F \rangle]$. For efficiency of the subsequent steps, it is advisable to do so. To this end, after F is obtained as described above, any triangle whose removal from F does not create a hole in F and does not cut a bridge in F is greedily removed from F. This is repeated until no more triangles can be removed. Figure 3 shows this for the 2D example. Note that $\langle F \rangle$ consists of ∂M and interior components (the actual cut).

3.3.2. Boundary extraction

Based on F, the boundary B is created. For every simplex $s \in \langle F \rangle$ and every side of F it lies on, a copy of s is added to B and connected accordingly: Let C be the set of tetrahedra incident to s. F partitions C into subsets C_i of tetrahedra on different sides – in Figure 2 at most two, in Figure 4 at most three. For each C_i a copy s_i of s is added to B. The connectivity of simplices in B is defined as follows: Let $C(s_i) := C_i$. A simplex a is a face of a simplex b in B if $C(a) \subseteq C(b)$.

Note that B is not a geometric triangle mesh in the sense of Section 3.1 because it contains geometrically coincident elements, the multiple copies of certain simplices. Combinatorially, though, it is a closed manifold triangle mesh. Furthermore, as it forms the boundary of the ball-topology cut M, B is combinatorially simply-connected, i.e. of genus 0. Note that this also allows us to visualize B using 2D plane embeddings, which is utilized in the 2D views in Figure 2 and following. They show plane embeddings of B punctured at some point; the circular boundary of the disk corresponds to this point.

3.4. Loop Closing

If L is a loop also on B (as $L1$ in Figure 2), the surface shift algorithm can now be applied directly to yield a disk Δ. But L may fall into multiple pieces on B, like $L2$ in Figure 2. In this case the loop in general forms an edge mesh $X \subseteq B$ containing multiple paths on B. To the best of our knowledge, there is no known way to reliably prevent this, i.e. to construct F dependent on a given loop L such that they are guaranteed not to interfere.

Note that $\langle F \rangle$ can be non-manifold, not just where its interior components meet the boundary but also within these interior components, in the interior of M as in Figure 4. We also call an edge of B *non-manifold*, if it corresponds to a non-manifold edge in $\langle F \rangle$. The non-manifold edges partition B into *parts*. Parts coinciding with ∂M are *boundary parts*, the others are *inner parts*. Inner parts come in pairs, each inner part has an *opposite* part. In simple cases, each inner part is surrounded in B by one boundary part (Figure 2). More generally, inner parts can be adjacent in B; we say that edge-connected components of inner parts in B form a *side* (Figure 4).

L *touches* an inner part if it contains a vertex but no incident edges corresponding to that part. L *runs along* an inner part if it contains an edge corresponding to that part. If L only touches inner parts, X is uniquely determined, as for every edge in L there is a unique corresponding edge in B. Where L runs along inner parts,

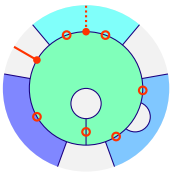
however, there is ambiguity due to multiple copies of these edges in B. While any copy can validly be picked in such a case, it is preferable to minimize the resulting number of components in X to reduce effort in subsequent steps. Hence, we determine X by walking along L, edge by edge, and always picking that copy in B for inclusion in X that is adjacent to the edge previously added to X, if available.

The loop pieces X lie in the boundary parts, their open ends ∂X lie on the boundary of inner parts. Our goal is to bridge these gaps across inner parts, so as to form loops again, by adding paths inside the inner parts that connect open ends. We construct these additional *connecting paths* incrementally, as detailed in the following. In this we respect the requirement that these connecting paths are consistent across opposite parts, i.e. when a path is added to one part, an identical (or rather mirrored) copy needs to be added to its opposite part. This is done until X, supplemented by the connecting paths, consists of only loops $\{L_i\}$. The connecting paths essentially define the intersection of the sought disk surface Δ with B in the interior of M, hence the above consistency requirement.

3.4.1. Connecting path

Let O be the vertices of X with an odd valence in X. In order to form loops, every vertex of X must have an even valence, so paths must be appended to the vertices O. Note that typically $O = \partial X$, i.e. the open ends (valence 1) form O, but in the course of the algorithm, vertices with an odd valence > 2 may arise in X and these must be treated as open as well.

To reduce algorithmic complexity, we add paths on the level of parts rather than on the level of sides (which can consist of multiple parts in structurally diverse configurations). As, however, the connection of two open ends may need to span multiple parts within a side (see Figure 4), we add *virtual* open ends to O. These do not have to, but may be used. For this, one vertex is added per *unique* connection of inner parts. In the inset figure, for one exemplary part (green) the vertices of O are marked in red, circles indicate virtual ones. The grey parts are the boundary parts.



Let $s,t \in O$ be any two vertices on the boundary of one inner part $I \subseteq B$. A connecting path $P \subseteq I$ between them, i.e. with $\partial P = \{s,t\}$, in the form of a sequence of edges is searched using a breadth-first search applied in I, excluding its boundary ∂I (except s and t) and other connecting paths possibly already constructed on I. We detail in Section 3.4.3 which pairs s,t are selected to be connected.

Refinement If a path cannot be found we search for a dual path instead, i.e. a triangle strip, between two triangles incident to s and t. As I is one connected component of edge-connected triangles, this dual path exists regardless of tessellation (in contrast to the above primal edge path), as long as t is continuously reachable from s. This dual path is then turned into a primal path, using the edges along the left or the right of the strip. Where these edges cannot be used (because they run along ∂I or another path), the mesh is refined by splitting inner edges of the strip to pave the way for the primal path. If the strip consists of a single triangle, a 1-to-3 split of this triangle may be needed. A similar refinement strategy is also described by Kraevoy and Sheffer [KS04]. To maintain conformance, the splits are also performed in M and on the opposite part in B.

Opposite path Since the path P on I does not contain non-manifold edges of B, every edge in P has a unique counterpart on the opposite part. These counterpart edges form the opposite path $\bar{P}$. Both paths, P and $\bar{P}$, are then added to X, i.e. X is set to $X \cup P \cup \bar{P}$, before the next connecting path is constructed. We remark that, in general, $\bar{P}$ may connect to X at vertices not yet in O – this is where vertices of valence > 2 in X arise. It may also not be connected to X at all, or only at one end, yielding new open ends in O.

3.4.2. Untangling

Once X only has vertices of even valence, i.e. O (ignoring virtual vertices) is empty, it forms a collection of loops $\{L_i\}$. In the typical case, all vertices in X have valence 2, such that the connected components of X readily form these loops. In general, though, loops may touch, i.e. share single vertices. These are the vertices of (even) valence > 2 in X. As we require non-crossing loops, at such vertices in X an incident edge needs to be part of the same loop as either its clockwise or counterclockwise neighbor. This leaves two connectivity choices per vertex of valence > 2, both lead to non-crossing loops by definition, so we can pick either. If a resulting loop touches itself at some vertex, we split it into two (touching) loops – recursively, where necessary, until all loops are simple.

3.4.3. Connection pattern

In simple cases, if the input loop L intersects only components of the virtual cut that do not contain inner non-manifold edges, i.e. if it only interacts with one-part sides in B, and this not more than twice per side (like all loops in Figure 2), there is only one combinatorially unique way to add connecting paths: per part there are only no or two open ends that need to be connected (independently, i.e. in arbitrary order).

Then there are cases, again involving interactions only with one-part sides, but more than two per part, as in Figure 5. In such a part, with $2n$ open ends, there are C_n combinatorially distinct ways of connecting them using non-crossing paths, C_n being the n-th Catalan number

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

In some of these cases any of these C_n options can be picked, in others only some choices ultimately lead to a valid result. Instead of adding algorithmic complexity to predetermine and plan ahead which choices these are – which is a hard non-local problem due to interplay of all such parts – we opt to enumerate these choices until a valid one is found, motivated by the observation that high-n scenarios are theoretically interesting but practically atypical.

The most complex class of cases are those with the input loop intersecting also non-manifold components of the virtual cut. In such cases, in general, also sides of non-trivial (e.g. annulus) topology can arise, causing a potentially infinite number of topologically distinct connection patterns (technically facilitated by the virtual open

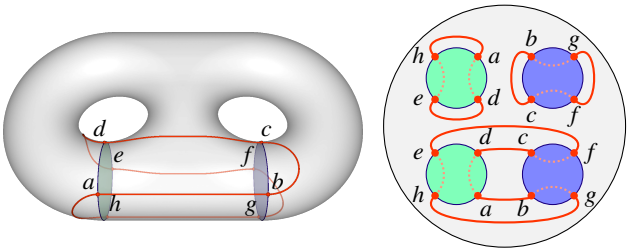


Figure 5: *Two times two opposite parts (marked in the same color) with four initial open ends each. The loop L is split into eight paths on B at the vertices a–h. If the open ends are connected with paths ad, bg, cf, and eh (as indicated by the dotted curves),* $|\{L_i\}| = 3$ *and* $\chi = -1 \neq 1$*. The implied surface has the topology of a torus minus a disk. The other three ways of connecting the open ends lead to* $|\{L_i\}| = 5$ *and* $\chi = 1$*, implying a valid surface.*

ends). Nevertheless, the space of options is *countably* infinite, patterns can be enumerated, such that also in such – rare but possible – cases the trial-until-valid strategy is applicable.

Stopping criterion Before we discuss the enumeration of connection patterns for trial, let us clarify when to stop, i.e. how to determine if a connection pattern is valid. Regardless of the chosen pattern (that leaves no open ends), we yield a set of non-crossing loops $\{L_i\}$, contractible on B. Hence, for each of them a disk Δ_i in the interior of M can be constructed (see Section 3.5); doing this sequentially allows ensuring they are disjoint away from their boundaries. Due to connecting path compatibility across opposite parts, these disks Δ_i join to a connected surface, bounded only by L. The only aspect that may vary across connection patterns is the topology of this joint surface. If it has disk topology, the current pattern is valid, the enumeration can stop. Otherwise, it needs to continue.

This can be checked without explicitly creating the disks Δ_i and joining them to Δ, since $[\Delta]$ has the same topology as the the abstract *polygon mesh* δ in which every loop L_i bounds one polygon. It has disk topology *iff* it is manifold and its Euler characteristic $\chi = 1$. Both conditions only depend on the connectivity of δ. Let $V_M \subseteq M$ be the set of vertices and $E_M \subseteq M$ the set of edges of which a copy exists in X. Then V_M are the vertices and E_M the edges of δ. By construction, the surface is manifold at most of these; only vertices for which X has vertices of valence > 2 need to be checked for manifoldness. The Euler characteristic can be calculated as

$$\chi = |V_M| - |E_M| + |\{L_i\}|.$$

An example of a connection pattern that implies a surface that is manifold but with $\chi \neq 1$ is given in Figure 5. Conversely, there exist triangle meshes bounded by a loop and with $\chi = 1$, but non-manifold, that could result, e.g. a Möbius strip with a sphere attached at a non-manifold vertex. Hence, both properties indeed need to be checked.

Enumeration First, all ways of adding one connecting path (one step of adding P and $\bar{P}$) are tried. Their number is the product of the inner parts' Catalan numbers C_n, and they are easily enumerated lexicographically. Next, for each of these, all possibilities of adding a second connecting path are tried, and so on. This can be implemented as a breadth-first search over the states of X with different connections made. Whenever a complete pattern is reached, the above stopping criterion is checked. Figure 6 shows an example of this search.

3.4.4. Termination

If the input loop L permits a disk topology surface, it is eventually found by the above algorithm. To see this, let $\tilde{\Delta} \subseteq [M]$ be such a surface with $\partial\tilde{\Delta} = [\partial M] \cap \tilde{\Delta} = [L]$ and disk topology. We can, if necessary, locally perturb its interior to be in general position, meaning it intersects $\langle F \rangle$ only in curves, and non-manifold M-interior edges of $\langle F \rangle$ only in points, i.e. tangential intersections are avoided. Likewise, if $\partial\tilde{\Delta}$ locally runs along the virtual cut, i.e. along non-manifold edges of $\langle F \rangle$ on ∂M, we can perturb the interior of $\tilde{\Delta}$ such that at least an ε-strip along that edge lies on that side of the virtual cut that the corresponding edge in L is mapped to in B to form the initial X (as described in Section 3.4).

Now, $\tilde{\Delta}$ intersects B in finitely many closed curves, i.e. loops. If one of these loops, L°, is entirely contained in one (inner) side, and bounds a disk region D in that side, this indicates that the surface $\tilde{\Delta}$ locally protrudes through this part (and possibly further ones). We further deform the interior of $\tilde{\Delta}$ so as to retract such local protrusions, getting rid of all such side-contained intersections. Concretely, L° partitions $\tilde{\Delta}$ into a disk (the protrusion) and an annulus; cutting out the disk and replacing it by disk D, followed by local perturbation to resolve the tangential contact, yields the retracted result.

The intersection pattern that $\tilde{\Delta}$ now forms with B, consisting of loops, is one that our algorithm can represent in terms of connectivity: No loop is contained entirely in an inner part, but either spans multiple parts or lies on just a boundary part, by construction. The set O of possible connection path endpoints in our algorithm, with virtual open ends, (Section 3.4.1) is designed to permit exactly all different crossing patterns across parts a loop may take within a side, as long as the loop is not a trivial loop entirely contained in a part. The algorithm will thus eventually encounter this connection pattern in the enumeration. When it is encountered, the surface $[\Delta]$ constructed by the algorithm will be homeomorphic to $\tilde{\Delta}$, thus satisfy the disk topology requirement, such that the algorithm terminates.

Excessive search space If $|\partial X|$ is initially large because L crosses the virtual cut formed by F many times, many connecting paths are necessary, thus the enumeration of many (partial) connection patterns is required. This can lead to excessive run time. In our experiments we limit the algorithm to explore no more than 1 000 000 states. This limit was reached in about 0.1 % of our tests (presented in Section 5). In such cases the implementation picks a different random seed to determine F and restarts; this led to success within the limit also for those few cases. Potential improvements are discussed in Section 5.3.

Invalid input loop L must be valid, i.e. permit a disk-topology surface. If L is invalid, e.g. if it is a tunnel loop [DLSCS08], the

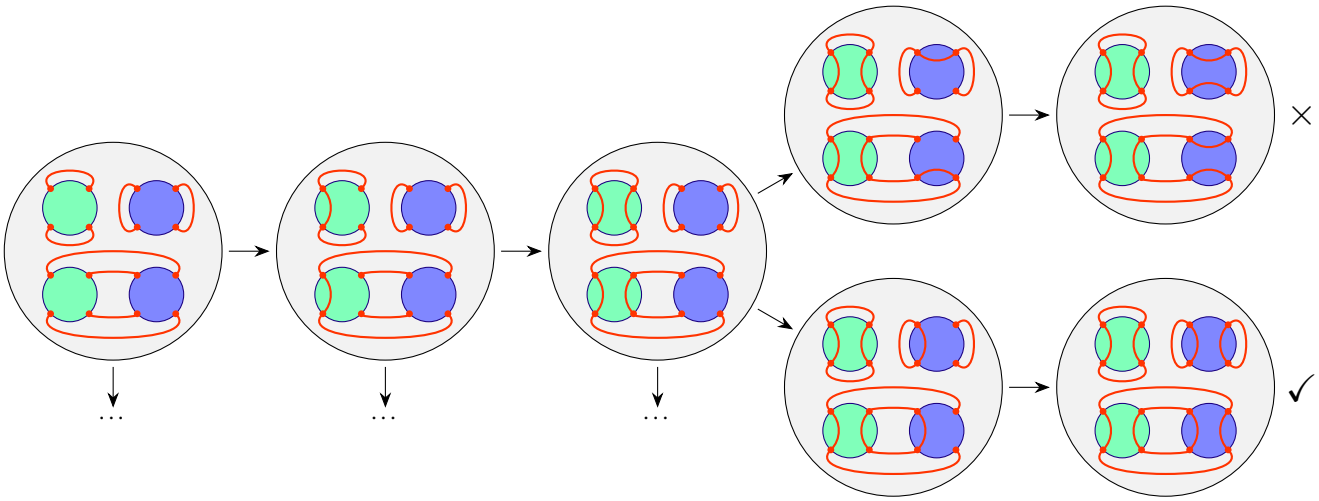


Figure 6: *Part of the search graph for the instance from Figure 5. Every node represents a different state of X. With every step, one path and its opposite is added. While the first way of connecting all open ends (top) yields a non-disk surface (cf. Figure 5), the second way (bottom) yields a disk surface. This state (or another of the valid final states) is eventually reached and the search terminates.*

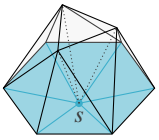
algorithm may not terminate. It will terminate if the search space is finite (see the discussion in Section 3.4.3). Invalidity can then be concluded from the stopping condition not being satisfied. We can also conclude invalidity (and terminate early) if initially any side has an odd number of open ends (not shared with an adjacent side).

3.5. Sub-Disk Construction

Once the search for a valid pattern terminates, the final task is to construct the disks Δ_i in the loops $\{L_i\}$. A loop L_i divides B into two halves because it is simply connected. Each of these halves has the connectivity of a 2-manifold surface of disk topology. We initialize $\Delta_i \subseteq B$ as one of these halves – we pick the smaller one. Then Δ_i is shifted into M, off of B, keeping only its boundary fixed at L_i. This is done using the surface shift algorithm [HBC24, §5.2.2], discussed below.

3.5.1. Surface shifting

Let us recap the surface shift algorithm. It operates incrementally and discretely, applying a local shift operator to individual simplices of the surface to be shifted. For a simplex $s \in \Delta_i$, the *dome* of s (illustrated in the inset) is the closure of the set of tetrahedra incident to s in front of Δ_i. The *floor* (blue) of a dome are the simplices on its boundary of which s is a face. The *ceiling* of a dome is the complement of its floor's closure in the dome's boundary. A simplex in the ceiling is *forbidden* if it is in ∂M or in Δ_i. By splitting certain dome simplices, a buffer between the floor and the forbidden simplices can be created; the new (lower) ceiling then is entirely non-forbidden. The shift operator, applied at s, removes the floor from Δ_i and adds the ceiling to it, effectively shifting the surface away from s. This operator is applied to the inner vertices, then the remaining inner edges, and then the remaining triangles (if any) of the initial Δ_i. See Figure 7 for an example.



Non-manifold extension Note that while Δ_i is 2-manifold in terms of connectivity, its embedding $[\Delta_i]$ may initially be non-manifold geometrically, e.g. due to containing opposite parts. We extend the surface shift algorithm to handle also such initial configurations. Concretely, multiple copies of the same simplex of M can occur in Δ_i on different sides. For each copy a shift operation is performed to its respective side. We redefine the floor to be the set of all simplices of Δ_i of which s is a face. This differs slightly from the previous definition and is necessary to properly cover also the case of the initial Δ_i being folded onto itself, back to back, such that all tetrahedra incident to certain s lie on the front, i.e. the dome is effectively folded onto itself and contains the floor in its interior rather than on its boundary.

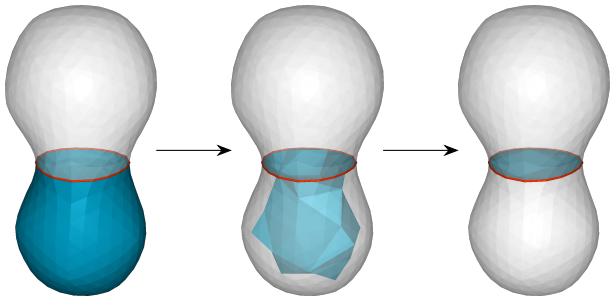


Figure 7: *Left: Surface (blue) bounded by the loop (red) initialized as the bottom half of the mesh boundary. Center: It is shifted away from the boundary using atomic operations that shift across a dome of tetrahedra; after having applied this operation to all boundary elements in the interior of the surface, it only intersects the mesh boundary along the loop, while disk topology is preserved. Right: Through further shifts, this disk surface can be shrunk for simplification.*

Furthermore, in the context of our algorithm, besides ∂M and Δ_i, also B as well as previously shifted sub-disks $\Delta_j, j < i$, need to be considered as forbidden. This is to ensure that the different sub-disks do not intersect each other, such that the final Δ does not self-intersect.

Note that although the domes and floors may be non-manifold, the ceilings still are manifold. Therefore, the shifted Δ_i ends up manifold also in terms of its embedding $[\Delta_i]$, as required.

3.5.2. Surface shrinking

As optional result simplification, after Δ is created, we apply further shifts (not requiring refinement) across domes wherever this decreases the surface area of Δ, cf. Figure 7. B can now be disregarded for this, only ∂M and Δ must be considered forbidden.

4. Mapping

Making use of the presented cutting algorithm, we are able to generalize the bijective volumetric mapping method by Hinderink et al. [HBC24] from ball-topology solids to a much broader class.

Concretely, the set of all solids, i.e. compact 3-manifolds with non-empty boundary embedded in $\mathbb{R}^3$, can be classified into so-called 0-handlebodies, 1-handlebodies, and 2-handlebodies [Mat02] – more details below. While the former method supports only 0-handlebodies (that all have the topology of a ball only), our generalization adds support for 1-handlebodies (often referred to simply as handlebodies). This is a topologically much larger class, containing solids with boundaries of arbitrary genus.

The only class of solids that remains unsupported are true 2-handlebodies. These are rather rare in practice. For instance, only 1.0 % of the manifold solids from the Thingi10K dataset [ZJ16], generated by TetWild [HZG*18], are not 1-handlebodies but 2-handlebodies.

4.1. Handlebodies

A k-handlebody is any 3-manifold that can be obtained by repeatedly *attaching* λ-*handles*, with $\lambda \leq k$ [Mat02]. Attaching a 0-handle just means adding a disjoint topological ball, i.e. another component. Attaching a 1-handle means gluing a topological ball to existing components in two places, with both gluing sites of disk topology. This, e.g., can turn a ball into a solid torus, a solid torus into a solid double-torus, etc. Attaching a 2-handle means gluing a topological ball to an existing component along an annulus-topology gluing site. This, e.g., can turn a ball into a ball with a ball-topology inner cavity. The above mentioned 1.0 % of true 2-handlebodies in that dataset contain one or more cavities (of varying topology) each. Besides such solids with cavities, another (very rare) type of solids whose construction requires attaching 2-handles (thus are not 1-handlebodies) are nontrivial knot complements, i.e. solids pierced by tunnels that are knotted.

4.2. Method Generalization

Besides lifting the ball-topology requirement, the overall setting for the mapping remains unaltered: The input is a source tetrahedral mesh M, such that $[M]$ is a 1-handlebody, and an orientation-preserving bijective continuous boundary map $\psi : \partial[M] \to \partial\Omega$, linear per boundary triangle, onto target shape Ω. The output is a bijective continuous map $\Psi : [M] \to \Omega$. The target shape Ω is defined by ψ and has the same topology as $[M]$, enabling the homeomorphism.

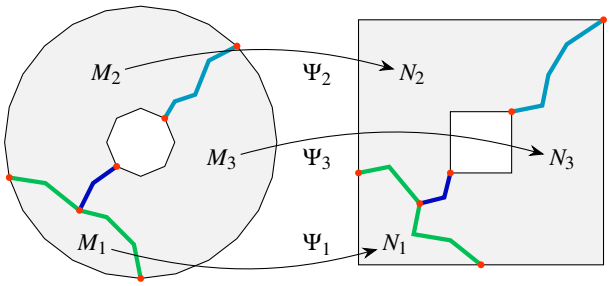


Figure 8: *2D example for the mapping method based on compatible decomposition. N is decomposed into star-shaped components $\{N_i\}$. The vertices (red, analogous to loops in the 3D case) bounding the interfaces (thick lines) between the components of N are mapped to M with the given boundary map ψ^{-1}. In M, the cutting algorithm is then utilized to create structurally compatible interface curves (disk surfaces in the 3D case).*

In the following we reiterate the method's main steps, detailing the change that involves the introduction of our novel cutting algorithm. For details on the unaltered parts we refer to the original publication [HBC24]. Note that the reason for the limitation to 1-handlebodies (which includes 0-handlebodies) lies in the fact that we make use of cutting using disks – by means of the above proposed cutting algorithm. 2-handlebodies require more complex (annulus-topology) cuts to be completely decomposed into topologically trivial components, as required in the mapping method.

A 2D illustration of the method's principle is given in Figure 8.

4.2.1. Star decomposition

Ω is tetrahedrized to a tetrahedral mesh N. N is partitioned into star-shaped components $\{N_i\}$. We replace the generic LP solver used in previous work for this partitioning by the algorithm of Seidel [Sei91], which we find to be significantly faster. This is due to the employed LP being low-dimensional, having only four variables.

4.2.2. Compatible source decomposition

Next, M needs to be decomposed into components $\{M_i\}$, such that this decomposition is compatible, structurally identical to the decomposition of N. This is done proceeding component by component. A component N_i is picked (e.g. N_1 in Figure 8), its interface with the rest of the mesh being of disk topology. The boundary loop of this disk is pulled-back from N to M using the prescribed boundary map's inverse. A disk-topology cut surface bounded by this loop is computed in M, forming the interface in correspondence with the one in N. Then the components N_i and M_i are virtually discarded, and the remainder processed further. After all components

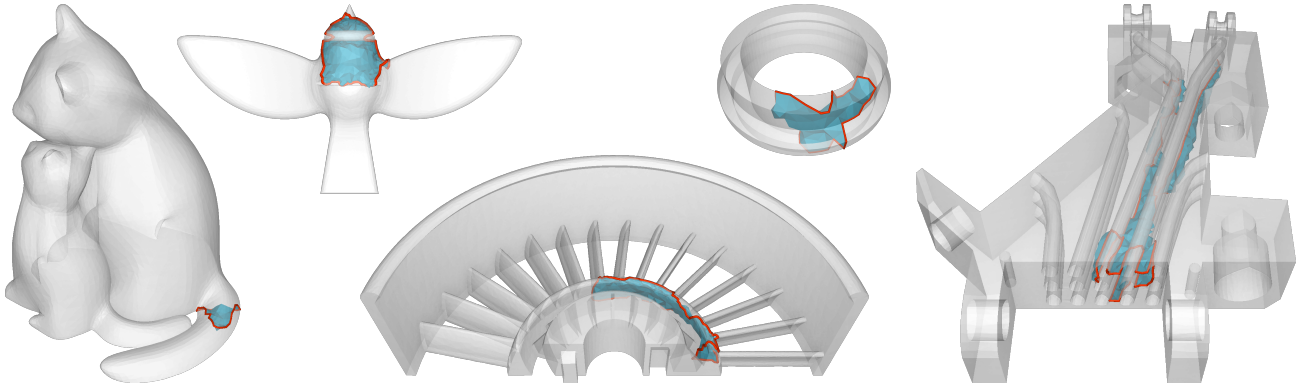


Figure 9: *Five meshes (IDs 444 375, 488 050, 669 971, 99 933, and 376 245), each with one of the boundary loops (red) generated by HanTun [DLSCS08] and the resulting disk surface (blue). The dataset includes cutting instances of varying complexities, from relatively simple (left) to complex (right).*

are processed, the compatible decomposition is complete. We remark that the star-shapedness requirement only applies to the target, not the source.

In the case that M is of ball-topology, it could always be ensured that the interfaces are single disks and their boundaries trivial loops. In our more general setting, they may consist of multiple disks and their boundary loops may be non-trivial, most importantly when a component is processed whose removal reduces the genus – such as N_2 in Figure 8, after N_1 was discarded. By sequentially treating these multiple disks and employing our proposed cutting algorithm from Section 3 that supports non-trivial loops, also such cases are covered. The implementation needs to take care that the component is only discarded once all its interface disks are processed, and that the possibly multiple cutting disks constructed for one component remain disjoint – effectively by marking them forbidden for purposes of the surface shifting.

The total number of cut disk problems to be solved changes in this generalized setting from $|\{N_i\}|-1$ to $|\{N_i\}|+g-1$ (per component, if $[M]$ consists of multiple connected components), where g is the genus of 1-handlebody M.

It was pointed out [HBC24] that, in theory, no N_i with a disk-topology interface may be selectable at some point, but that this could be remedied via refinement of the components. The argument for this remedy relies on the ball topology of N. Using our cutting algorithm, based on g non-homotopic handle loops, M and N could first be compatibly cut to ball topology using g cut disks each, the prescribed boundary map be extended onto these (as in Section 4.2.3), effectively resulting in a ball-topology mapping problem – to which the argument applies again. But, similar to the observation in previous work, this scenario appears to be of merely theoretical relevance. No such refinement or special handling was required in any experiment.

4.2.3. Boundary map extension

The extension of ψ onto the component interfaces is performed as in the original work, employing the recursive *triangulation alignment* algorithm.

4.2.4. Per-component mapping

As in the orignal work, for the final step of creating bijections $\Psi_i : [M_i] \to [N_i]$ per component, existing methods that support at least star-shaped targets [HC23, NCB23, NCB24] can be used. Together, these form the sought bijection Ψ.

5. Evaluation

We evaluate the proposed cutting algorithm (Section 5.1) as well as the presented general bijective mapping method that makes use of it (Section 5.2). All experiments were conducted using single-threaded implementations on a system with Intel Xeon Platinum 8462Y+ processor @ 2.8 GHz.

5.1. Cutting

5.1.1. Dataset

We make use of the tetrahedral mesh version of the Thingi10K dataset [ZJ16] from the supplemental material of TetWild [HZG*18]. It contains 8399 manifold tetrahedral meshes.

To all these we apply HanTun [DLSCS08] to generate multiple topologically distinct handle loops per mesh, which are incontractible on the solid's boundary, thus cannot be handled by previous work [HBC24, §5.2.2]. A tetrahedral mesh together with one such loop is one cutting problem instance. Note that HanTun also generates tunnel loops; these are not valid inputs for the cutting method as they, by definition, do not permit a solid-contained disk.

The HanTun implementation provided by the authors is not unconditionally robust, so it does not yield the theoretical maximum of the number of handle loops. Yet, a total of 35 790 problem instances are obtained in this way, available at `https://github.com/SteffenHinderink/DiskScissors`.

5.1.2. Cutting results

The proposed cutting algorithm succeeds on all 35 790 instances. In Figure 9, some of these instances are presented together with their resulting cut surfaces.

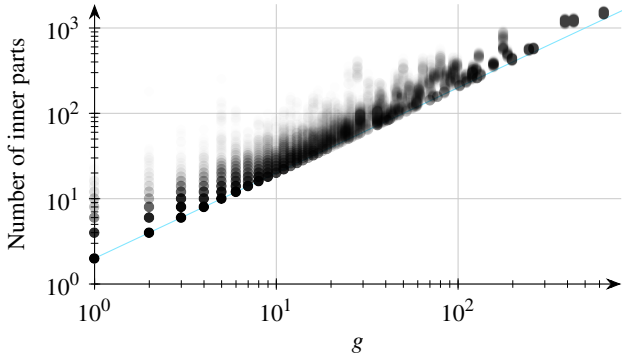


Figure 10: *Genera g of the boundary component of M containing L versus the number of inner parts in B resulting from the algorithm for all cutting problem instances. At least $2g$ (indicated by the blue line) inner parts are necessary to yield a simply connected boundary.*

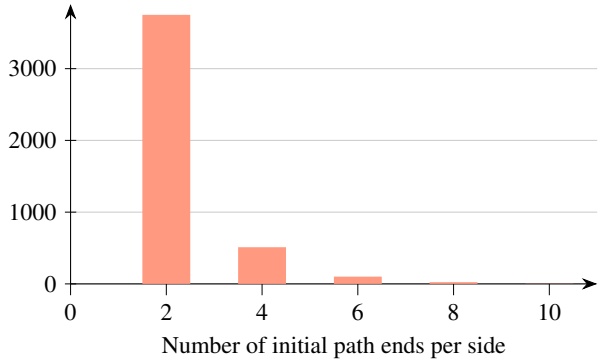


Figure 11: *Histogram of the number of initial open ends incident to the same side over all sides in the instances that are actually hit.*

The instances contain a variety of genera. The required complexity of the virtual cut to yield the simply-connected boundary B, i.e. the lower bound of the number of inner parts, grows with the genus of the mesh. Fewer inner parts are preferable in terms of loop closing efficiency; Figure 10 gives an idea of the level of effectiveness of the simple balloon inflation approach in that regard.

On these inner parts fewer initial open ends are preferable. Especially the number of open ends per side is a major factor in the size of the search space, as these need to be connected in some way. Figure 11 shows that this number is indeed pretty small over the dataset.

Since a side can be partitioned into multiple parts, and connecting paths are constructed on the level of parts, the required number of connecting paths can be higher than the initial number of loop pieces in X, or half the number of open ends. Figure 12 shows these numbers based on the number of paths that L is initially split into. These numbers reveal that the vast majority of connections between initial open ends must be direct connections, crossing only a single part.

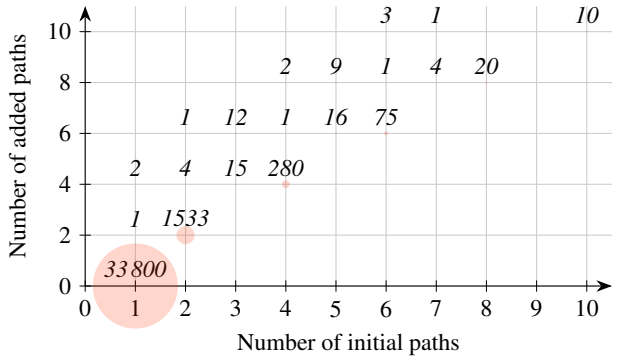


Figure 12: *The number of paths that loop L is initially split into on B, versus the number of connection paths added to close the loops, i.e. twice the depth of the breadth-first search. The area of the dots is proportional to the number of instances with the respective numbers of initial and added paths. In most instances, L remains a loop on B. In the remaining* 1990 *instances, it is split into multiple pieces. The three instances with a single initial path yet requiring additional connecting paths may seem surprising: in these the loop is split at a single point, where an inner piece of B touches ∂M.*

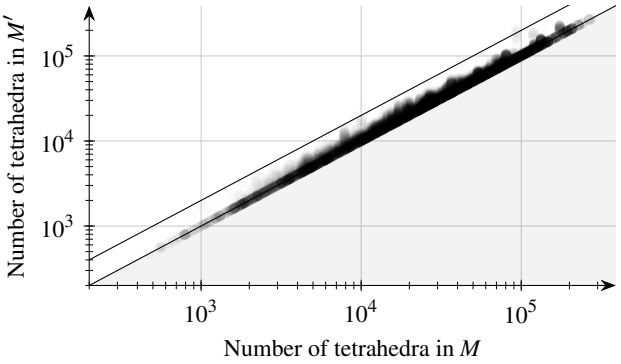


Figure 13: *The number of tetrahedra in the initial mesh M versus in the final mesh M'. The lower line indicates the identity. Instances on this line are not refined at all. The upper line indicates a refinement factor of* 2*, i.e. for instances on that line the number of tetrahedra is doubled. Only* 17 *instances lie above that line, the maximum refinement factor is below* 2.6*.*

Together, these insights indicate that, typically, the search space is actually relatively simple and only few steps of adding connecting paths and enumerating connection patterns are necessary to close the loops. This motivates the use of the relatively simple breadth-first search strategy rather than adding additional algorithmic complexity in that part of the algorithm.

Refinement Combinatorial bijective volumetric mapping algorithms [CSZ16, HC23, NCB23, NCB24, HBC24] suffer from relatively large amounts of on-demand mesh refinement caused by their execution. Since a key application of the cutting algorithm is in this context of mapping, we also analyze this aspect of refine-

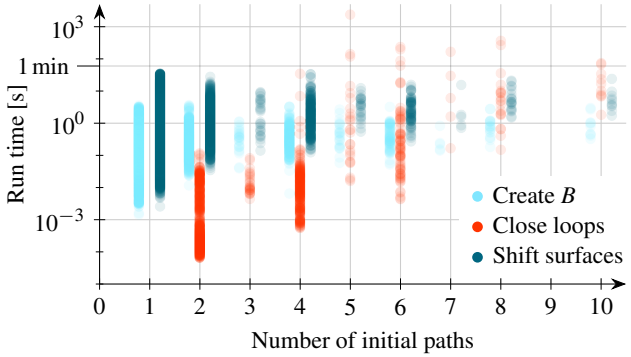


Figure 14: *The number of paths that loop L is initially split into versus the run time of the different stages of the algorithm. The stages are described in Section 3.3, Section 3.4, and Section 3.5, respectively. To avoid visual clutter, the dots of the different stages are displaced a bit left and right.*

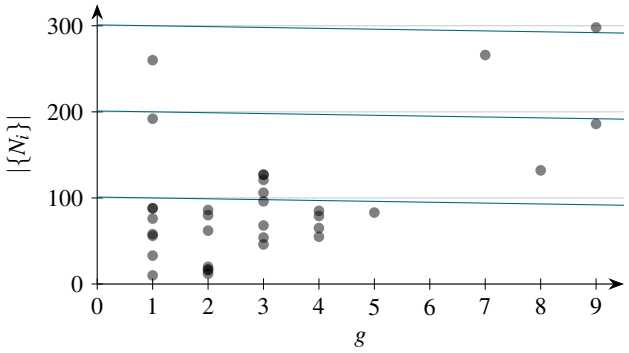


Figure 15: *Genera g of M and N versus the number $|\{M_i\}| = |\{N_i\}|$ of components they are each decomposed into over all mapping problem instances. The dark blue lines indicate where the number $|\{N_i\}| + g - 1$ of required disk cuts is* 100, 200, *and* 300.

ment introduced by the cutting algorithm. Refinement can occur in two steps: First, triangles and edges may be split to ensure the existence of the path P (Section 3.4.1). Second, dome simplices may be split to enable surface shifts (Section 3.5.1). Figure 13 shows the total amount of refinement incurred. We observe that it is typically relatively low in comparison to that of mapping methods, thus should not have a significant negative effect in that context.

Run time Figure 14 shows the run time of the different stages of the algorithm, relative to instance complexity in terms of the number of open ends to be connected. The times to create B and to shift the surfaces are relatively consistent (differing mainly depending on mesh size). The time to close the loops, by contrast, grows with the number of paths that L is initially split into. This is expected as this number is strongly correlated with the number of paths added (Figure 12), which in turn affects the number of connection patterns potentially explored. For a few complex instances the run time can reach several minutes. The average run time of the full algorithm is 2.6 s.

5.1.3. Comparison

We compare our algorithm's performance to the intrinsic alternative approaches, implicit and explicit, discussed in Section 2.1. In the implicit case, Δ is determined as the zero-isosurface of the discrete harmonic field x on the dual of M, i.e. with values on the tetrahedra of M, calculated by solving $Ax = 0$, with A being the (uniform) Laplacian matrix. Values for x are constrained to -1 and 1 for the boundary tetrahedra adjacent to the loop L to its left and right, respectively. This is based on similar setups (with varying constraints) used in former work [GMD*16, HJ17, HBC24]. In the explicit case, Δ is computed by an LP that solves for the discrete solid-contained minimal surface spanned by the loop L [Gra06].

To aid these methods, we pre-refine M to ensure that each edge of L has at least one incident triangle that is interior.

The harmonic field linear system is solved using the Eigen library. The resulting isosurface-implied cut surface turns out to be valid (manifold disk topology, intersecting the solid's boundary in L and only in L) for 24 269 of our test instances ($\approx 68.0\,\%$) and the average run time is 1.1 s.

The discrete minimal surface LP is solved using Clp. The resulting cut surface is valid for 27 114 instances ($\approx 75.8\,\%$) with an average run time of 4.1 s.

We remark that both of these approaches could certainly be improved in some ways, e.g., with additional tailored mesh refinement heuristics or post-process topology modifications, but to the best of our knowledge no concrete such strategies have been detailed in the literature, in particular not ones that would reliably yield the 100 % success rate of our proposed method.

5.2. Mapping

The dataset† of Du et al. [DAZ*20] contains 33 volumetric mapping problem instances for which the object pairs have genus $g > 0$ and the prescribed boundary map ψ is actually bijective. In particular, these are mapping problems of solids into polycube target shapes. TetGen [Si15] is used to tetrahedralize the targets. We evaluate our generalized bijective mapping method on these mapping problem instances.

All 33 instances are successfully handled. Figure 15 shows the genera of the instances as well as the number of components in the respective decompositions, and thus the number of cuts performed.

Since the harmonic isosurface approach and the minimal surface approach (Section 5.1.3) both have a relatively consistently low run time, for each disk cut to be performed in the course of the method, we try them first and escalate to our reliable cutting algorithm if they fail. Statistics about this are shown in Table 1. As can be seen, the alternatives combined succeed in $\approx 34\,\%$ of cases. In $\approx 66\,\%$ of cases the surface shift based technique needs to come into play. In

† https://github.com/duxingyi-charles/Locally-Injective-Mappings-Benchmark

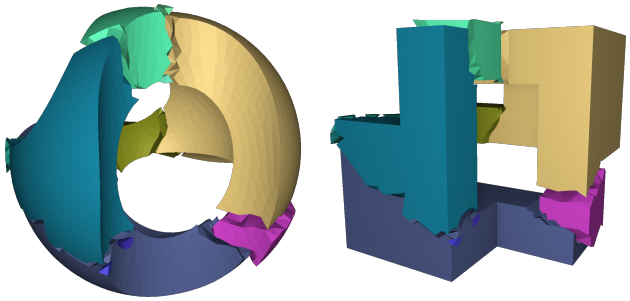


Figure 16: *Star decomposition of the target mesh (right) and compatible decomposition of the source mesh (left). These meshes are of genus $g = 2$ and are decomposed into 20 components, thus 21 cuts (along disk surfaces) are performed.*

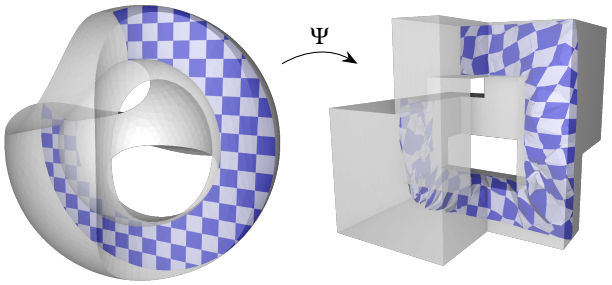


Figure 17: *Visualization of the volumetric map constructed based on the decomposition in Figure 16. The components were mapped using the method of Hinderink and Campen [HC23]. The image of the uniform checkerboard pattern on a cross section of the solid shows the map's behavior in the interior. No form of map optimization was applied; the constructed initial bijective output is shown.*

21 cases our novel arbitrary-topology treatment is crucial; without it, 12 of the 33 mapping problem instances would have failed.

The decomposition of one of these instances (*sculpt*) is visualized in Figure 16, the resulting bijective map in Figure 17.

Table 1: *Number of times the different cutting approaches succeed in the mapping algorithm, grouped by trivial and non-trivial boundary loops. If an approach fails, the one in the next row is tried. Ours succeeded in all remaining cases. The last column and row contain the respective sums.*

	Trivial	Non-trivial	
Harmonic isosurface	784	63	847
Minimal surface LP	214	19	233
Ours (Section 3)	2115	21	2136
	3113	103	3216

5.3. Limitations and Future Work

Some aspects of our cutting algorithm (Section 3) offer opportunities for future work. First, implementation improvements are possible: In the breadth-first search, redundancies could be eliminated by identifying already visited states of X. Splits performed to enable paths that are later not used could be undone. The connections could also be planned more abstractly before adding actual paths.

A limitation of the algorithm is the potentially very large number of steps required for the loop closing in cases where L crosses B very often, e.g. zigzagging over the virtual cut. As this is often not topologically inherent, this could be circumvented in two different ways: Either B is modified by shifting such that L crosses it less, or topologically trivial protrusions of L across the virtual cut are retracted by shifting on the boundary, then the cut disk is computed, and this resulting surface shifted reversely, un-retracting its boundary loop.

Non-manifoldness of virtual cuts (as in Figure 4) is the main factor increasing the algorithm's complexity. If B could be generated such that it virtually cuts M along disjoint disks only, the search space of path combinatorics would be simplified significantly. It would also make it finite in general, enabling a complete check for the input loop's validity. The B generated in the way described, however, can be such that there is no obvious simple way to untangle it into disjoint disks; Figure 18 shows an example that evades such attempts.

The generalized mapping method (Section 4) can lead to maps that, before some potential optimization in a post-process, exhibit high distortion – depending on the base method employed per component of the decomposition. Figure 19 shows that for the choice of base method used in our experiments [HC23], it can indeed be extremely high, much higher than necessary, as can be estimated when comparing to the distortion of a map globally optimized for low distortion, "[GKK*21] global" in the plots. To get an idea of the extent to which this initial excess distortion is due to the particular choice of base method and to which extent due to our cutting method inducing distortion on top, the plots also show the distortion of a map optimized for low distortion *per component*, fixing the

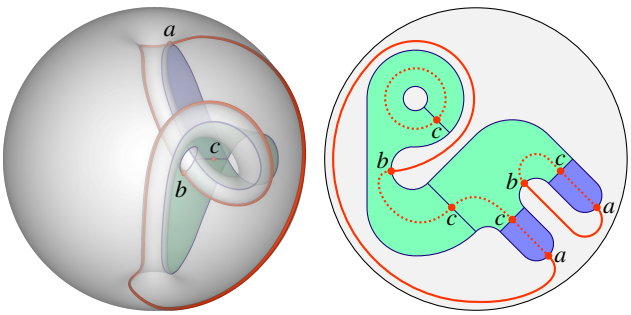


Figure 18: *Example of a cut surface possibly resulting from balloon inflation (in a genus 1 object) that has a non-manifoldness that is not resolvable in any obvious systematic way. Also shown is a loop (red) and the simplest way of closing its pieces on B (right), that yields a disk surface.*

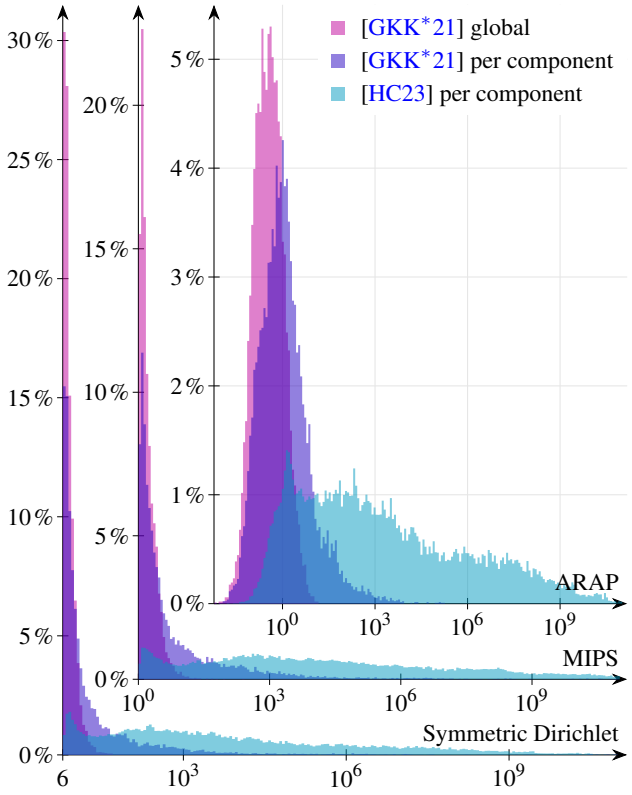


Figure 19: *Histograms of multiple distortion measures [ASGS23] over the tetrahedra of a mapping instance (double_torus) from Section 5.2. Our arbitrary-genus mapping method largely inherits the characteristics of the specific genus-0 mapping method chosen for per-component mapping [HC23], leading to partially extreme initial distortion (■). For reference, also shown are the distortion distributions of alternative maps optimized for low distortion [GKK*21], either likewise per component (■), or globally (■).*

map along the cuts, the component interfaces, "[GKK*21] per component". The smaller difference between the latter two indicates that a smaller share of the excess distortion is due to the cuts (in particular their shape), while the largest share is inherited from the chosen base method. It is thus worth exploring the use of alternative base methods [NCB23, NCB24] as well as efficient post-process distortion optimization of such high-distortion (but bijective-by-construction) maps.

Figure 20 illustrates an example of a unfavorably shaped cut surface generated by our method in the mapping context – a source of the cut-related part of the above distortion. An improved non-greedy version of the cut surface shrinking (Section 3.5.2), or even a technique that coordinates the cut surface shape between source and target object, would be valuable.

Finally, non-1-handlebodies, i.e. true 2-handlebodies, are the last class of solids not yet covered by reliable bijective map construction methods. While rare in practice, at least conceptually this remains an interesting problem.

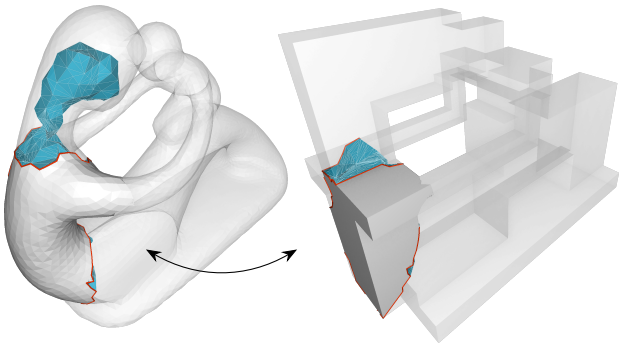


Figure 20: *One component of the decomposition of a mapping instance (fertility) in the source mesh (left) and its corresponding component in the target mesh (right, star-shaped) with corresponding interfaces highlighted (blue, red boundary). The discrete surface shrinking here did not manage to yield a particularly simple cut surface shape (left). The implied unnecessarily large shape difference between source and target component implies some additional distortion (and possibly mesh refinement) in the per-component maps – which can then only be taken care of in a post-process distortion optimization of the joint map.*

6. Conclusion

We have presented DiskScissors, an algorithm for reliably cutting solid objects along given boundary loops with disk-topology surfaces. The loop is possibly split into multiple paths, which are then connected through the solid's interior to form loops again. These subloops are such that they are contractible on a certain domain, such that disk-topology surfaces can be more easily constructed for these. Together, these form the desired disk-topology cut surface.

Using this algorithm to generalize a former star-decomposition based mapping method for simply connected solids, now enables most solids (1-handlebodies) to be constructively mapped in a continuous and bijective manner.

Acknowledgments

This work was funded by the Deutsche Forschungsgemeinschaft (DFG) - 497335132. Open Access funding enabled and organized by Projekt DEAL.

References

[ASGS23] ABULNAGA S. M., STEIN O., GOLLAND P., SOLOMON J.: Symmetric volume maps: Order-invariant volumetric mesh correspondence with free boundary. *ACM Trans. Graph. 42*, 3 (2023), 25:1–25:20.

[BC23] BRÜCKLER H., CAMPEN M.: Collapsing embedded cell complexes for safer hexahedral meshing. *ACM Trans. Graph. 42*, 6 (2023), 180:1–180:24.

[CFLF25] CHENG Y.-Y., FANG Q., LIU L., FU X.-M.: Divide-and-conquer embedding. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers* (2025), pp. 98:1–98:10.

[CSZ16] CAMPEN M., SILVA C. T., ZORIN D.: Bijective maps from simplicial foliations. *ACM Trans. Graph. 35*, 4 (2016), 74:1–74:15.

[DAZ*20] DU X., AIGERMAN N., ZHOU Q., KOVALSKY S. Z., YAN Y., KAUFMAN D. M., JU T.: Lifting simplices to find injectivity. *ACM Trans. Graph. 39*, 4 (2020), 120:1–120:17.

[DHS10] DIERKES U., HILDEBRANDT S., SAUVIGNY F.: Minimal surfaces. In *Minimal Surfaces*. Springer, 2010, pp. 53–90.

[DKZ*22] DU X., KAUFMAN D. M., ZHOU Q., KOVALSKY S., YAN Y., AIGERMAN N., JU T.: Isometric energies for recovering injectivity in constrained mapping. In *SIGGRAPH Asia 2022 Conference Papers* (2022), pp. 36:1–36:9.

[DLSCS08] DEY T. K., LI K., SUN J., COHEN-STEINER D.: Computing geometry-aware handle and tunnel loops in 3D models. *ACM Trans. Graph. 27*, 3 (2008), 45:1–45:9.

[FBRCA23] FINNENDAHL U., BOGIOKAS D., ROBLES CERVANTES P., ALEXA M.: Efficient embeddings in exact arithmetic. *ACM Trans. Graph. 42*, 4 (2023), 71:1–71:17.

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. *ACM Trans. Graph. 21*, 3 (2002), 355–361.

[GKK*21] GARANZHA V., KAPORIN I., KUDRYAVTSEVA L., PROTAIS F., RAY N., SOKOLOV D.: Foldover-free maps in 50 lines of code. *ACM Trans. Graph. 40*, 4 (2021), 102:1–102:16.

[GMD*16] GAO X., MARTIN T., DENG S., COHEN E., DENG Z., CHEN G.: Structured volume decomposition via generalized sweeping. *IEEE Transactions on Visualization and Computer Graphics 22*, 7 (2016), 1899–1911.

[Gra06] GRADY L.: Computing exact discrete minimal surfaces: Extending and solving the shortest path problem in 3D with application to segmentation. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2006), pp. 69–78.

[GS25] GOATES C., SHEPHERD K. M.: Counterexamples to proofs for volumetric parameterization of topological sweeps, 2025. arXiv:2503.01573.

[Hat02] HATCHER A.: *Algebraic topology*. Cambridge University Press, 2002.

[HBC24] HINDERINK S., BRÜCKLER H., CAMPEN M.: Bijective volumetric mapping via star decomposition. *ACM Trans. Graph. 43*, 6 (2024), 168:1–168:11.

[HC23] HINDERINK S., CAMPEN M.: Galaxy maps: Localized foliations for bijective volumetric mapping. *ACM Trans. Graph. 42*, 4 (2023), 129:1–129:16.

[HG15] HUYNH L., GINGOLD Y.: Bijective deformations in $\mathbb{R}^n$ via integral curve coordinates, 2015. arXiv:1505.00073.

[HJ17] HABERLEITNER M., JÜTTLER B.: Isogeometric segmentation: Construction of cutting surfaces. *Computer-Aided Design 90* (2017), 135–145.

[HZG*18] HU Y., ZHOU Q., GAO X., JACOBSON A., ZORIN D., PANOZZO D.: Tetrahedral meshing in the wild. *ACM Trans. Graph. 37*, 4 (2018), 60:1–60:14.

[JWP*22] JI Y., WANG M.-Y., PAN M.-D., ZHANG Y., ZHU C.-G.: Penalty function-based volumetric parameterization method for isogeometric analysis. *Computer Aided Geometric Design 94* (2022), 102081.

[KS04] KRAEVOY V., SHEFFER A.: Cross-parameterization and compatible remeshing of 3D models. *ACM Trans. Graph. 23*, 3 (2004), 861–869.

[LAPS17] LIVESU M., ATTENE M., PATANÉ G., SPAGNUOLO M.: Explicit cylindrical maps for general tubular shapes. *Computer-Aided Design 90* (2017), 27–36.

[LBHH23] LIU C., BÉNARD P., HERTZMANN A., HOSHYARI S.: ConTesse: Accurate occluding contours for subdivision surfaces. *ACM Trans. Graph. 42*, 1 (2023), 5:1–5:16.

[Liv24a] LIVESU M.: Advancing front surface mapping. *Computer Graphics Forum 43*, 2 (2024), e15026.

[Liv24b] LIVESU M.: Stripe embedding: Efficient maps with exact numeric computation. *ACM Trans. Graph. 43*, 6 (2024), 167:1–167:14.

[LLWQ13] LI B., LI X., WANG K., QIN H.: Surface mesh to volumetric spline conversion with generalized polycubes. *IEEE Transactions on Visualization and Computer Graphics 19*, 9 (2013), 1539–1551.

[LPP*20] LIVESU M., PIETRONI N., PUPPO E., SHEFFER A., CIGNONI P.: LoopyCuts: Practical feature-preserving block decomposition for strongly hex-dominant meshing. *ACM Trans. Graph. 39*, 4 (2020), 121:1–121:17.

[Mat02] MATSUMOTO Y.: *An Introduction to Morse Theory*. American Mathematical Society, 2002.

[NCB23] NIGOLIAN V. Z., CAMPEN M., BOMMES D.: Expansion cones: A progressive volumetric mapping framework. *ACM Trans. Graph. 42*, 4 (2023), 131:1–131:19.

[NCB24] NIGOLIAN V. Z., CAMPEN M., BOMMES D.: A progressive embedding approach to bijective tetrahedral maps driven by cluster mesh topology. *ACM Trans. Graph. 43*, 6 (2024), 170:1–170:14.

[NZZ20] NAITSAT A., ZHU Y., ZEEVI Y. Y.: Adaptive block coordinate descent for distortion optimization. *Computer Graphics Forum 39*, 6 (2020), 360–376.

[POK23] POYA R., ORTIGOSA R., KIM T.: Geometric optimisation via spectral shifting. *ACM Trans. Graph. 42*, 3 (2023), 29:1–29:15.

[Sei91] SEIDEL R.: Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry 6*, 3 (1991), 423–434.

[Si15] SI H.: TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw. 41*, 2 (2015), 11:1–11:36.

[Tak19] TAKAYAMA K.: Dual sheet meshing: An interactive approach to robust hexahedralization. *Computer Graphics Forum 38*, 2 (2019), 37–48.

[XGZ11] XIAN C., GAO S., ZHANG T.: An approach to automated decomposition of volumetric mesh. *Computers & Graphics 35*, 3 (2011), 461–470.

[ZJ16] ZHOU Q., JACOBSON A.: Thingi10K: A dataset of 10,000 3D-printing models, 2016. arXiv:1605.04797.