Inferring topological operations on G-maps formalism: application to iterated function systems

Romain Pascual, Hakim Belhaouari, Agnès Arnould, Pascale Le Gall

Inferring topological operations on G-maps formalism: application to iterated function systems

Romain Pascual [1], Hakim Belhaouari [2], Agnès Arnould [2], and Pascale Le Gall [1]
[1] MICS, CentraleSupélec, Université Paris Saclay
[2] XLIM UMR CNRS 7252, Université de Poitiers

December 16, 2021

Abstract

The design of correct topological modeling operations is known to be a time-consuming and challenging task. However, these operations are intuitively understood via simple drawings of a representative object before and after modification. We propose to infer topological modeling operations from an application example. Our algorithm exploits the compact and expressive graph-based language used in the Jerboa platform. In this framework, topological modeling operations on generalized maps are represented as rules from the theory of graph transformations. Most of the time, operations are generic up to a topological cell (vertex, face, volume). Thus, the rules are annotated with variables indicating which kind of cells is involved. Our main idea is to infer a Jerboa rule by folding a graph representing the object before and after modification. We fold this graph according to the cell parametrization of the operation under design. We illustrate our approach with some examples of iterated function systems.

Keywords: Topology-based geometric modeling, Iterated function systems, Operation inference, Inference from examples, Topological operation, Computational topology

1 Introduction

Procedural modeling consists in producing complex objects by automated successive transformations [25, 24, 33, 34]. These transformations may exploit operations commonly known as L-system or Iterated Function Systems (IFS). Dedicated operations or dedicated adaptations of existing operations may also be relevant to ease the design process.

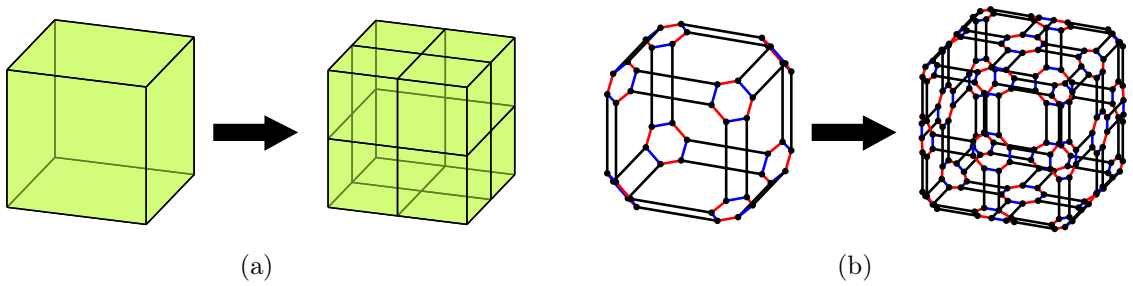
These dedicated (adaptations of) operations usually aim at simplifying the production of domain-specific objects. For instance, certain simplifications replicate a sequence of operations as a single one. We wish to provide specific optimizations of the aggregated transformation by inlining operations. Indeed, such transformation does not suffer for the proliferation of sub-routines. As a result, the generation time of complex scenes can be more easily monitored.

Our work lies in the field of topology-based geometric modeling [5]. Here, we tackle the topological structure of the objects. We leave aside geometrical aspects such as the position of vertices or any embedding information carried by the *topological cells* (volumes, faces, edges, and vertices). In particular, we exploit the formalism of *generalized maps*, or G-maps [5, 18, 19].

This model has the main benefit of being homogeneously defined in all dimensions in the form of graphs [27, 3].

Since objects are formalized using graphs, modeling operations can be studied as rules in the framework of graph transformations [31, 9, 13]. As operations should produce a well-formed object when applied to a well-formed object, the preservation of the topological constraints [26] and the geometric constraints [2] is ensured with syntactic conditions on rules.

Intuitively, a rule is written $A \rightarrow B$ and allows to transform object A into object B within a more general context. For example, the subdivision of the cube, illustrated in Figure 1(a), is expressed as a rule where object A is the cube before subdivision and object B is the object after subdivision. The same operation is given as a rule on the corresponding G-maps in Figure 1(b), it subdivides any quadrilaterally-faced hexahedron.



(a) (b)

Figure 1: Operation of quad subdivision: application to a cube (a) and produced operation on G-map (b).

The Jerboa framework [1] offers a compact and expressive graph-based language to design modeling operations. Topological cells, and more generally orbits, are used as rule variables to describe modifications of objects [3, 26]. These extended rules, called rule schemes, express transformations valid for all possible shapes of a given orbit type, providing the desired generalization. Rule schemes are instantiable to concrete rules, depending on the object under modification. Therefore, Jerboa allows the design of modeling operations via code generation.

Intuitively, Jerboa is a topology-based geometric modeling platform that allows for defining operations for specific topological cells. Our idea is to exploit this cell-based definition of modeling operations: we want to infer the topological rule corresponding to an example and a given cell type. For instance, the modeling expert will describe the quad subdivision operation by the cubes in Figure 1(a) and ask how this operation can be generalized to any surface. Apart from this cell-based definition of modeling operations, we chose to use the Jerboa platform because the inferred operations are correct by construction. Indeed, syntactic conditions on rules ensure that rewriting a G-map always results in a G-map. Besides, the operations can be directly exported in a viewer and applied to an object for visualization.

Despite the undeniable assets of Jerboa, rule schemes might be hard to write without double expertise in geometric modeling and graph transformations. In this paper, we present an algorithm to conceive topological modeling operations without knowledge on generalized maps or Jerboa's formalism. Our algorithm only needs two objects to reconstruct an operation. The first one is used as a reference, while the second describes the result of applying the operation. We will illustrate our approach with the help of iterated function systems. Section 2 recalls the formalism of generalized maps and modeling operations described as rules. Section 3 presents the algorithm for reconstructing rule schemes. Validation of our approach is given in Section 4

on iterated function systems. Section 5 discusses practical side-effects of our inference mechanism. Section 6 is dedicated to related works, while Section 7 provides concluding remarks.

2 Generalized maps

The model of generalized maps relies on a boundary representation (or B-rep) of objects [5, 18, 19]. G-maps are defined homogeneously for any dimensions with a graph representation.

2.1 Structure

We call *graph* a classical undirected graph, possibly with parallel arcs and loops. We use edge-labeled graphs to define generalized maps (G-maps). Nodes are called darts and arcs are called links. A link labeled with i will also be called an i-link or an α_i-link. A *generalized map* of dimension n, or n-G-map, is a graph G labeled on $[\![0, n]\!]$ satisfying the following topological constraints:

Incident arcs constraint any dart of G is the source of a unique α_i-link, for each dimension i in $[\![0, n]\!]$.

Cycle constraint any dart of G is the source of an $\alpha_i\alpha_j\alpha_i\alpha_j$-cycle, for each i and j in $[\![0, n]\!]$ such that $i + 2 \leq j$.

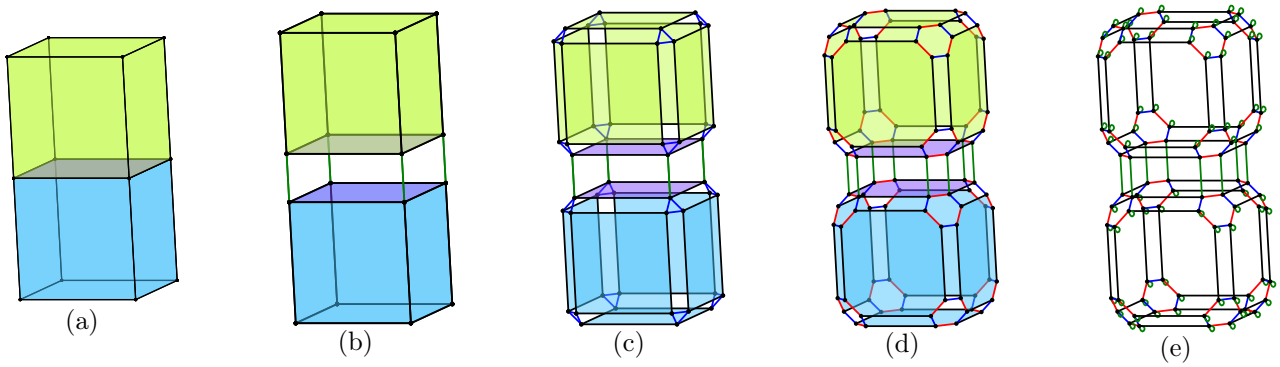


Figure 2: Topological decomposition of a geometric object in dimension 3: (a) two cubes sharing a face, (b) split on α_3, (c) α_2, (d) α_1, and (e) α_0. The graph G_{topo} (e) is the corresponding G-map.

The representation of a geometric object can be reconstructed from its decomposition into topological cells of decreasing dimensions. For instance, the 3D object of Figure 2(a) can be represented by a 3-G-map. First, the object is split into volumes linked along their shared face. This link means that the topological cells of dimension 3 or 3-cells (the volumes) share a common face. This first separation yields the Figure 2(b) where a green arc represents the 3-link. By iteration on decreasing dimensions, we obtain the Figure 2(c) after decomposition of the dimension 2 (blue arcs), the Figure 2(d) after the decomposition of dimension 1 (red arcs) and finally the Figure 2(e) after dimension 0 (black arcs). Loops are added on the free darts (for each possible dimension) to obtain the final object corresponding to the G-map displayed in Figure 2(e). Loops will sometimes be omitted to simplify the figures.

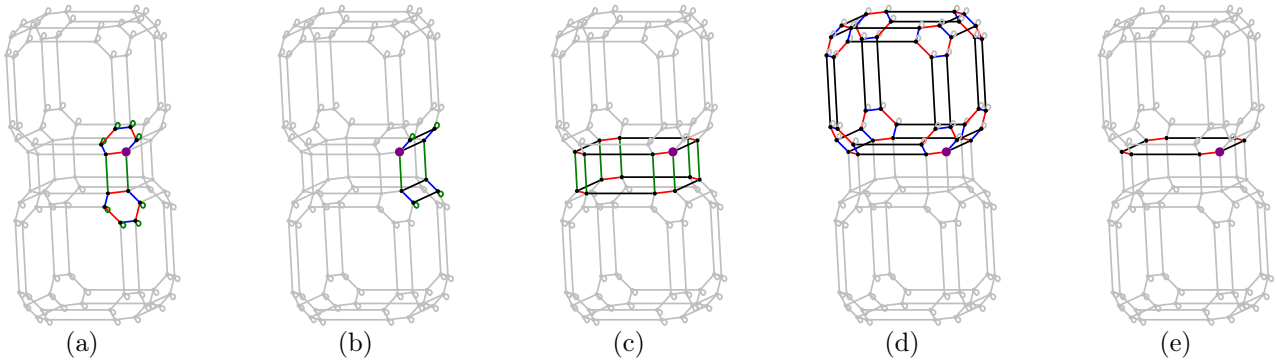


Figure 3: Orbits incident to the purple dart: (a) vertex $\langle 1,2,3\rangle(e)$, (b) edge $\langle 0,2,3\rangle(e)$, (c) face $\langle 0,1,3\rangle(e)$, (d) volume $\langle 0,1,2\rangle(e)$, and (e) half face $\langle 0,1\rangle(e)$.

In the sequel, we consider an n-G-map G. The topological cells (vertices, edges, faces, volumes) of the represented geometric object are not explicitly defined in a generalized map but can be implicitly retrieved. The cells can be computed via graph traversals restricted to a subset of dimensions. For instance, the cells incident to the pointed purple dart are depicted in Figure 3. Let the purple dart be called e. The 0-cell (the vertex) incident to dart e is given in Figure 3(a). This cell contains the dart, and every dart reachable by all links expect α_0 links (i.e., α_1, α_2, and α_3 links), along with the links themselves. This subgraph is written $G\langle 1,2,3\rangle(e)$. Similarly, Figure 3(b) displays the 1-cell incident to dart e, representing the edge incident to the purple dart. The subgraph $G\langle 0,2,3\rangle(e)$ corresponds to this edge. Likewise, the 2-cell $G\langle 0,1,3\rangle(e)$ incident to e is shown in Figure 3(c) and represents a face. Finally, the Figure 3(d) displays the volume incident to the purple dart e thanks to the subgraph $G\langle 0,1,2\rangle(e)$. More generally, the cells correspond to specific cases of orbits.

An *orbit* of G consists of a subgraph induced by all the darts reachable from an initial dart, only using links from a subset of $[\![0,n]\!]$. The orbit is written $G\langle o\rangle(v)$ with o subset of $[\![0,n]\!]$, or $\langle o\rangle(v)$ when there is no ambiguity on the graph. Such an orbit is said to be of type $\langle o\rangle$ or referred to as an $\langle o\rangle$-orbit. For example, the orbit $G\langle 0,1\rangle(e)$ described in Figure 3(e) represents the face of one volume (also called half face) incident to e. Since the full graph G_{topo} of Figure 2(e) is a complete connected component, it corresponds to the orbit $G\langle 0,1,2,3\rangle(e)$.

2.2 Modeling operations

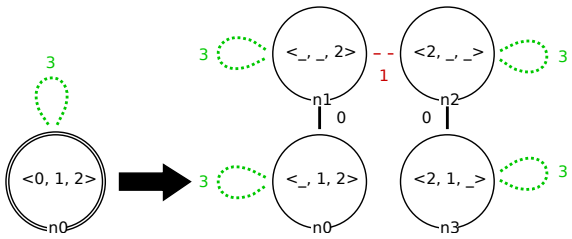
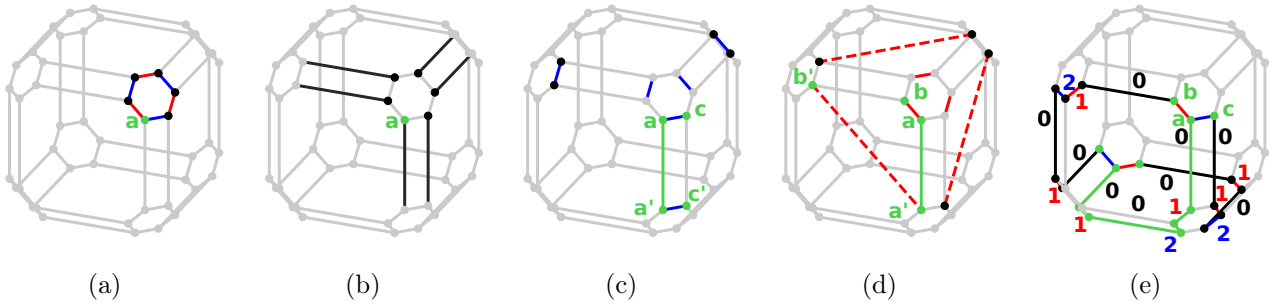


Figure 4: Rule scheme for the quad subdivision operation.

The subdivision illustrated in Figure 1 is called the quad subdivision. It is mainly used to refine the topological structure of meshes [4]. We can consider this operation as purely

topological, i.e., without considering the geometric properties. A new vertex is added to the center of the face and the middle of each edge. An edge then attaches the center of each face to the midpoints of each of the initial edges. When applied to a cube, the subdivision splits each face into four new faces (see Figure 1(a)). In the Jerboa platform, we can write this operation for an entire surface, i.e., on the volume cell $\langle 0, 1, 2\rangle$. The rule is provided in Figure 4.



(a) (b) (c) (d) (e)

Figure 5: Steps of the folding algorithm on the object. Step 1 (a) - construction of $G\langle 1, 2\rangle(a)$. Step 3 (b) - construction of the hook's explicit arcs. Step 4 (c) and (d) - construction of a node and its implicit arcs. The algorithm terminates (e).

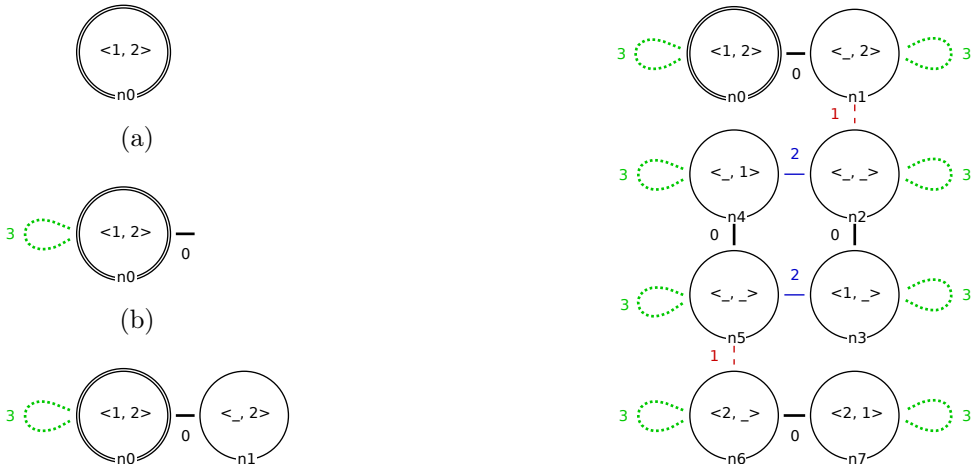


Figure 6: Different steps of the graph scheme in the folding algorithm: step 2 (a) - construction of the hook, step 3 (b) - construction of the hook's explicit arcs and step 4 (c) - construction of a node, (d) - result of the algorithm.

In the rule scheme of Figure 4, the left-hand side contains a single node. The double line around the node means that it is a *hook*. Intuitively, the hook describes the location where the modeling operation occurs. During the rule application, the hook matches the orbit incident to the selected dart. Then, the part to be modified (the subgraph) is retrieved thanks to the left-hand side of the rule. In our example, the node $n0$ of the left-hand side is labeled by $\langle 0, 1, 2\rangle$. Thus, the whole volume of the selected dart is matched. The semantics of Jerboa's rule requires that each node of the rule represents a copy of this subgraph, up to deletion and renaming of the links. For instance, the node $n0$ occurs in the left and right-hand sides of the rule. Thus, the rule preserves all darts matched by the node $n0$, i.e., belonging to the volume

incident to the selected dart. Nonetheless, the label of $n0$ is modified. The left orbit $\langle 0, 1, 2\rangle$ is substituted by $\langle \_, 1, 2\rangle$ in the right. This substitution specifies, by following the positions in the node label, that the 0 links are deleted (denoted by the underscore "_"), while the 1 and 2 links are preserved. The nodes $n1$, $n2$, and $n3$ appear only on the right-hand side of the rule. They correspond to creations and mean that the darts matched by the hook $n0$ are duplicated 3 times. The nodes $n1$ and $n2$ denote the mid-edge vertex, while node $n3$ is a part of the center-face vertex. The copy $n1$ removes the 0 and 1 links and keeps only the 2 links, as indicated by the label $\langle \_, \_, 2\rangle$ with two "_". The copy $n2$ renames all 0 links to 2 links and deletes the 1 and 2 links, as indicated by the label $\langle 2, \_, \_\rangle$ where the 0 has been substituted for a 2 and the 1 and 2 have been removed and replaced by "_". Finally, the node $n3$ renames the 0 links into 2 links, preserves the 1 links, and removes the 2 links. The links from the labels of the rule nodes are called *implicit* arcs.

On the other hand, the arcs between nodes of the rule are called *explicit* arcs. They allow linking copies of the same dart of the G-map. For example, the 0-arc between nodes $n2$ and $n3$ means that each copy associated with node $n2$ is linked by a 0-link to its counterpart in the copy associated with node $n3$. Node labels indicate how the copies of the matched cell are modified. They are variables, called *orbit variables*, and explicit the type of the matched cells (in the object under modification) and the copied cells (in the reconstructed object).

Rule schemes (i.e., rules with orbit variables) are an intermediate format for code generation of geometric modeling operations in the Jerboa platform. However, their authoring requires the learning of this specific language and a good knowledge of generalized maps. Therefore, we propose to infer them automatically based on a given cell and an example of the operation. This approach will exempt the designer from learning Jerboa's dedicated language.

3 Inferring operations

We want to provide the user with a mechanism to infer a generic modeling operation from an application example. In other words, we seek to reconstruct the rule scheme that corresponds to a modification of the object performed by a user who is not an expert in the underlying theory. For instance, we wish to infer the rule of Figure 4 from the objects of Figure 1 when specifying to infer an operation for the orbit $\langle 0, 1, 2\rangle$.

The user describes a starting object, modifies it, and asks what the corresponding operation is for a given orbit. Therefore, we aim at designing a mechanism for reconstructing a rule scheme given by an orbit (provided it exists), i.e., having an instantiation corresponding to the rule provided by the user. We call *operation inference* this mechanism.

3.1 Notations

The objective is to reconstruct the rule scheme that generalizes a specific modeling operation for a given orbit. Rather than directly reconstructing the rule scheme, we first design an algorithm to reconstruct a *graph scheme*. We will then explain how to modify the algorithm to build rule schemes. Intuitively, a graph scheme corresponds to the left (or right) hand side of a rule scheme. It is a graph where each node is associated with an orbit variable.

Our algorithm reconstructs a graph scheme $\mathcal{G}$ from a given G-map and a given orbit type $\langle o\rangle$ (with $o \subseteq [\![0, n]\!]$). In the graph scheme $\mathcal{G}$, nodes are generically named n plus an integer, e.g., $n0$, $n1$, while the darts of the G-map G are named with lowercase letters from the beginning

of the alphabet, e.g., a, b. The hook of the graph scheme is written h. First, we choose a dart a in the G-map. We assume that the orbit graph incident to a corresponds to the instantiation of the orbit variable of the hook h from the graph scheme $\mathcal{G}$. By local search, we then try to reconstruct the scheme. Intuitively the construction of the scheme $\mathcal{G}$ consists of folding the graph G along its i-links, for all dimensions i of the orbit type $\langle o \rangle$. If the construction fails, we start again with another initial dart (a) until a rule is found (or all darts have been tried).

3.2 Algorithm

We detail the algorithm that constructs a graph scheme for a given connected G-map G, a given orbit type $\langle o \rangle$, and a chosen dart a of G. The graph scheme $\mathcal{G}$ we seek to construct will have by construction a hook node h labeled by $\langle o \rangle$ if it exists.

In the sequel, we will illustrate the algorithm using the orbit type $\langle 1, 2 \rangle$ on the G-map of a cube, in other words, the left-hand side of the rule given in Figure 1(b). All darts are incident to a 3-loop not shown to avoid overloading the image.

Step 1 (Orbit graph) We build the orbit graph $G\langle o \rangle(a)$ on a in G.

The orbit graph $G\langle 1, 2 \rangle(a)$ on the cube is illustrated in Figure 5(a). Irrelevant links and darts have been dimmed. Thus, the orbit graph contains six darts and six links: three 1-links drawn in red and three 2-links in blue.

Step 2 (Construction of the hook) We initialize the scheme by its hook h labeled by the chosen orbit type $\langle o \rangle$. $G\langle o \rangle(a)$ is by construction a possible instantiation of this scheme.

In the case of the cube, we obtain the graph given in Figure 6(a) containing a simple hook $n0$ labeled $\langle 1, 2 \rangle$. The subgraph of Figure 5(a) is indeed one of its possible instantiations.

Step 3 (Construction of the hook's explicit arcs) The explicit arcs of the scheme, incident to the hook, are labeled by dimensions d not belonging to the orbit type $\langle o \rangle$. Each d-arc of the scheme must be instantiated into a d-link of G for each dart of $G\langle o \rangle(a)$. For such an instantiation to exist, two situations are possible:

1. The first possibility is that the d-arc of the schema is a loop and that all darts of $G\langle o \rangle(a)$ have a d-loop in G.

2. The second one is that the d-arc of the scheme binds the hook h to another node and that all the darts of $G\langle o \rangle(a)$ have a d-link to distinct darts outside of $G\langle o \rangle(a)$, in G.

If none of the above conditions are met, the construction fails. Take back our example. As mentioned previously, all darts of G, and thus of $G\langle 1, 2 \rangle(a)$ have 3-loops (not shown in the figures). Thus the hook has a 3-loop in the scheme being constructed. Likewise, the 6 darts of the vertex $G\langle 1, 2 \rangle(a)$ exhibit a 0-link to 6 distinct darts outside the vertex matched by the hook $n0$, as shown in Figure 5(b). We deduce the beginning of the graph scheme described in Figure 6(b). The scheme is incomplete at this point since the 0-arc is dangling.

Step 4 (Construction of a node) The construction of an explicit d-arc assumes that the target darts of the corresponding d-links are the instantiation of a new node m in the scheme graph. This node needs to be reconstructed, specifically its implicit arcs. We look for the existence of a label $\langle o^m \rangle$ whose instantiation provides the links that join the darts of the instance of m.

This boils down to identifying the implicit arcs that are preserved, deleted or renamed by the new node m. For every i of $\langle o \rangle$, one of the following condition should hold:

- If for any pair of i-linked darts, b and c of $G\langle o\rangle(a)$ (i.e., instance of the hook h), their respective d-neighbors, b' and c', are i-linked, then node m preserves the implicit arc i. Thus, i appears in $\langle o^m\rangle$ at the same position as in $\langle o\rangle$.

- If there is a dimension j such that for any pair of i-linked darts, b and c of $G\langle o\rangle(a)$, their respective d-neighbors, b' and c', are j-linked, then the node m renames the implicit arc i into j. In this case, j appears in $\langle o^m\rangle$ at the position of i in $\langle o\rangle$.

- Finally, if for any pair of i-linked darts, b and c of $G\langle o\rangle(a)$, their respective d-neighbors, b' and c', are not linked, then the node m deletes the implicit arc i. The symbol "_" appears in $\langle o^m\rangle$ at the position of i in $\langle o\rangle$.

If none of the above conditions are met, the construction fails. We exemplify this step on the dangling 0-arc of Figure 6(b) which instantiates into the six 0-links of the Figure 5(b). We add a new node $n1$ in the scheme and now have to construct its label. We try to construct a renaming of the 1 and 2 links of the initial orbit graph on this set of darts. As illustrated in Figure 5(c), we obtain a plausible preservation of the 2-link between a and c (since there is a 2-link between a' and c'). Conversely, as seen in Figure 5(d), there is no renaming of the 1-link between a and b; this link is deleted. We check that the renaming (the replacement of 2 by 2 and the deletion of 1) is correct for all darts. This renaming is valid, and we deduce the partial graph scheme given in Figure 6(c).

Note that, in general, two darts b and c of $G\langle o\rangle(a)$ can be connected by several i-implicit links (i.e., with different i of $\langle o\rangle$). One can then construct several different renamings. Our algorithm uses a heuristic to construct a plausible renaming among all possible ones. The possible plurality of inferred graph schemes (and therefore rule schemes) is further discussed in Section 5.3.

Step 5 (Traversal) Using a breadth-first traversal of G, we reconstruct a graph scheme $\mathcal{G}$ that yields G by instantiation. In other words, the graph scheme matches G from the dart a onto its hook node h.

This traversal consists of alternating steps 3 and 4 by generalizing them :

- A third case is added to the construction of explicit arcs, one where the d-arc connects the node being processed to another node already existing in the scheme. The construction of the explicit arc is then not accompanied by any node creation.

- The renaming between the implicit arcs $\langle o\rangle$ of the hook h and those $\langle o^m\rangle$ of the current node m is no longer built along a single explicit d-arc, because the hook h and the node m are not necessarily immediate neighbors. The renaming is then constructed along an explicit-arc path ω that connects h to m. This same path ω then connects the considered darts b and c, instances of the hook h, to their ω-neighbors b' and c', instances of the current node.

If the algorithm terminates without failure, we obtain a topologically correct graph scheme $\mathcal{G}$ for the chosen dart.

On the cube, the algorithm ends by constructing the last node $n7$. This node corresponds to the vertex opposite to the initial vertex, as shown in Figure 5(e). Following the exploration path 012010, we reach the darts of the opposite vertex, renaming the orbit variable $\langle 1, 2\rangle$ into $\langle 2, 1\rangle$. The resulting scheme is given in Figure 6(d).

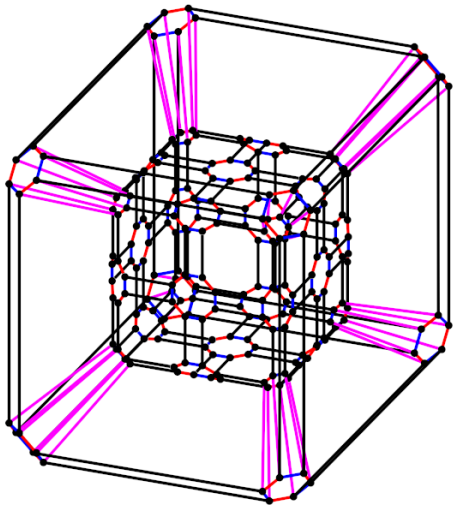


Figure 7: The graph $\kappa(\mathbf{L}, \mathbf{R})$ for quad subdivision of the cube.

3.3 Generalization to a rule scheme

To infer rule schemes coherent with a given rule, we propose considering the G-maps (left and right) as part of the same graph. From the two G-maps L and R, we construct the graph $\kappa(L, R)$ as the disjoint union graph of L and R, plus arcs labeled by κ (a particular symbol) connecting both copies of the preserved nodes. The graph $\kappa(L, R)$ constructed from the rule of Figure 1(b) is given in Figure 7. The κ-arcs are drawn in pink; the original cube is enlarged while the modified cube is shrunk.

To determine the scheme for a rule $r = L \twoheadrightarrow R$, we construct a scheme of the related graph $\kappa(L, R)$. We run the algorithm on $\kappa(L, R)$ by memorizing whether the darts belonged to L or R. The algorithm is identical on the graph $\kappa(L, R)$; we just need to add some additional conditions:

- We only consider darts of L to start the algorithm.
- At each identification of a renaming function from $\langle o \rangle$ to $\langle o^m \rangle$, κ is not a letter of $\langle o^m \rangle$. Therefore, κ-arcs can only occur as explicit arcs in the graph scheme.

Finally, the desired rule scheme is obtained by removing the κ-arcs from the output graph scheme.

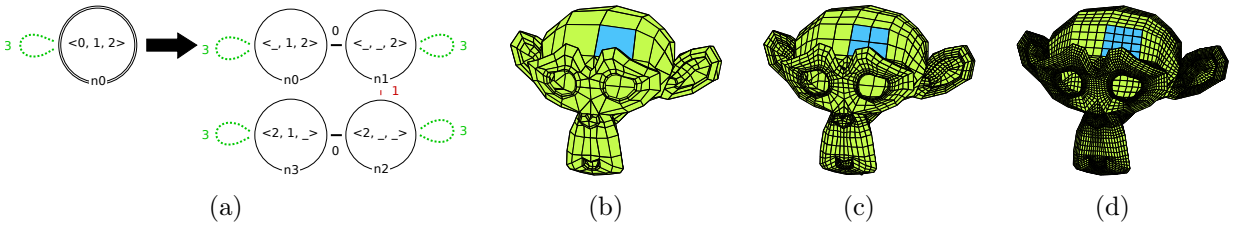
4 Examples of inference on iterated function systems (IFS)

We reconstructed several iterated function systems as a validation of our algorithm. We first present applications to surfaces (Sub-section 4.1). Afterwards, we illustrate our approach with operations on volumes (Sub-section 4.2).

4.1 IFS for surface refinement

We reconstructed three subdivision schemes for surfaces, i.e., with the orbit $\langle 0, 1, 2 \rangle$ as inference parameter. The first one, illustrated earlier, is the quad subdivision. Using the two generalized

maps of Figure 1(b) and the orbit $\langle 0, 1, 2\rangle$, we obtain the rule given in Figure 8(a). This rule corresponds to the one presented in Figure 4 made manually with Jerboa's rule editor.



(a) (b) (c) (d)

Figure 8: Quad subdivision operation: inferred rule (a) from the objects of Figures 1. Suzanne (b), first (c) and second (d) iterations of the quad subdivision.

The Doo-Sabin subdivision of surfaces [7] splits vertices. Figure 9(a) presents the initial object, while Figure 9(b) displays the first iteration of the subdivision. From these two objects, we infer the rule of Figure 9(c). We now have a standalone rule, applicable to any isolated surface, which refines objects as many times as desired. For instance, we can further subdivide the object of Figure 9(b) to obtain the second and third iterations, respectively illustrated in Figures 9(d), and 9(e). From these iterated applications, we can infer new operations directly performing the second or third iteration of the Doo-Sabin subdivision. The inferred rules for the second and third iterations (with the orbit type $\langle 0, 1, 2\rangle$) are illustrated in Figures 10(a) and 10(b). In other words, our mechanism allows for straightforward self-composition of operations. Inferring the rule of the third iteration took around 40ms, which means our approach is usable in practice.

Powell and Sabin studied various subdivision schemes on triangles [28] that ensure given values for the first derivatives on vertices. Such subdivisions are still studied to construct smooth finite element spaces [12]. The Powell-Sabin 6-split refines a triangle into six new triangles. The operation adds a vertex at the center of the face and links it with all vertices of the triangle as well as all midpoints of the initial edges. Assuming we only have three basic operations: split an edge, add a vertex, and link two vertices by an edge, we can reconstruct the Powell-Sabin 6-split refinement as described in Figure 11.

4.2 IFS for volume refinement

We reconstructed two iterated function systems for volume refinement, i.e., with the orbit $\langle 0, 1, 2, 3\rangle$ as inference parameter.

The Menger sponge is a 3D extension of the Cantor set (1D) or the Sierpinski carpet (2D). It can be computed as a fractal. Each refinement step can be described as follows. Take a cube and split each face into 9 squares to obtain 27 cubes. Remove all middle cubes (middle of faces and center of the initial cube). The 20 remaining cubes correspond to the iteration of the refinement step. This step is iterated on the newly obtained cubes. From the cube of Figure 12(a), we construct the first iteration of the Menger sponge illustrated in Figure 12(b). We can infer the operation by specifying that the operation occurs on the orbit $\langle 0, 1, 2, 3\rangle$ to obtain the rule of Figure 12(e). Note that this inferred rule has 20 nodes on its right-hand side, which might already prove challenging to write (or read). Our inference mechanism alleviates the fastidious task of manually designing such complex operations. After adding the missing geometry, we can iterate the inferred operation and obtain the following iterations of the Menger

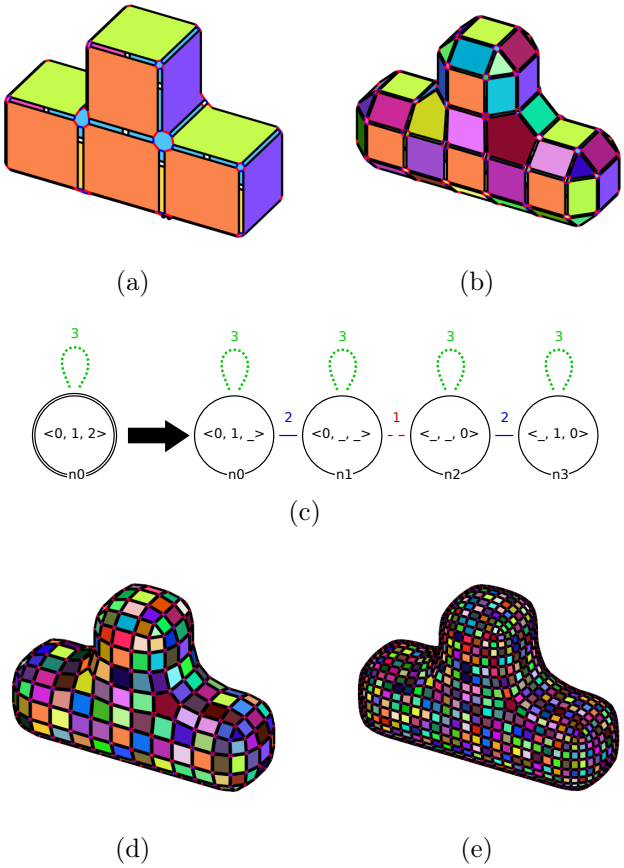


Figure 9: Doo-Sabin subdivision operation: the initial object (a), the first (b), second (d) and third (e) iterations of the subdivision. The inferred operation (c) from the objects (a) and (b).

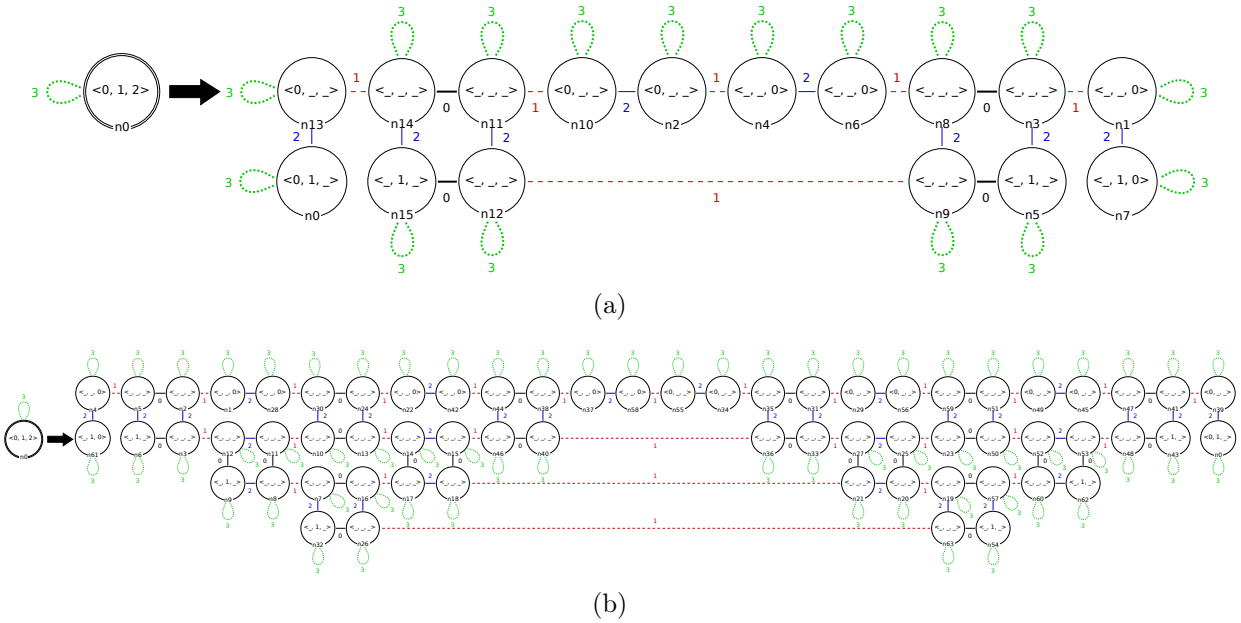


Figure 10: Inferred rules for the second (a) and third (b) iterations of the Doo-Sabin subdivision.

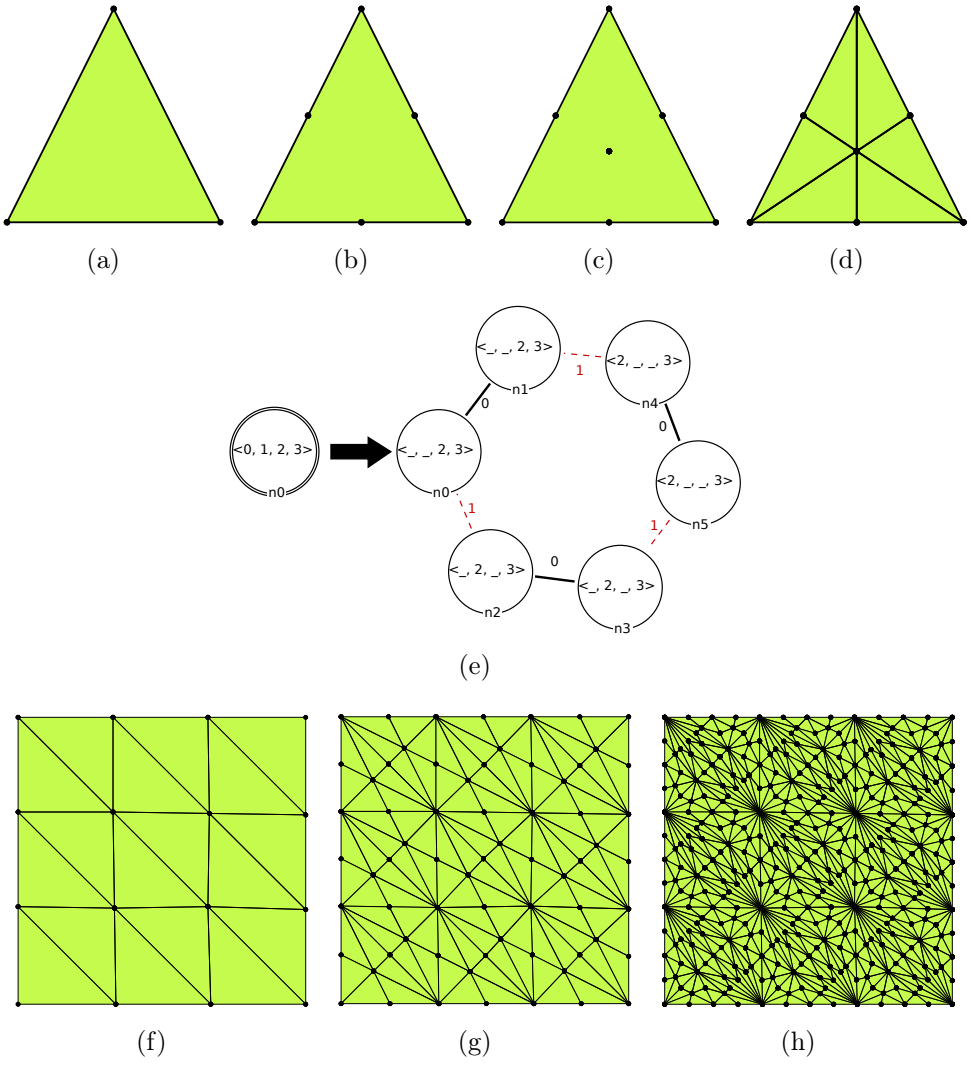


Figure 11: Powell-Sabin 6-split refinement: given a triangle (a), split the three edges (b), add a vertex (c), and link it with all other vertices (d). The inferred rule (e) is recursively applied on a triangulation of the unit square (f) (from [12]) to obtain more refined triangulations (g) and (h).

sponge (second and third iterations in Figures 12(c) and 12(d)). The inferred operation for the second iteration of the Menger sponge has 400 nodes and is too large to be properly drawn.

In [30], the authors proposed a generalization of the Menger sponge to Menger polycube. One iteration of the (L, M, N)-Menger operation transforms a polycube into a polycube with L holes along the x-axis, M holes along the y-axis, and N holes along the z-axis. Each hole has the same size as a one-unit cube and is separated from the nearest holes by a one-unit cube. From a cube, we built the first iteration of the $(2, 2, 2)$-Menger operation (see Figure 13(b)). The polycube is of genus 28 and consists of 81 volumes, 270 faces, 216 vertices. To our knowledge, there is no definition (either algorithmic or with a rule) of this operation. We built the polycube and used our algorithm to infer the operation. The reconstructed polycube is given in Figure 13(b). From the objects of Figures 13(a) and 13(b), we inferred the rule of Figure 13(e). We can now use this operation and build the second and third iterations,

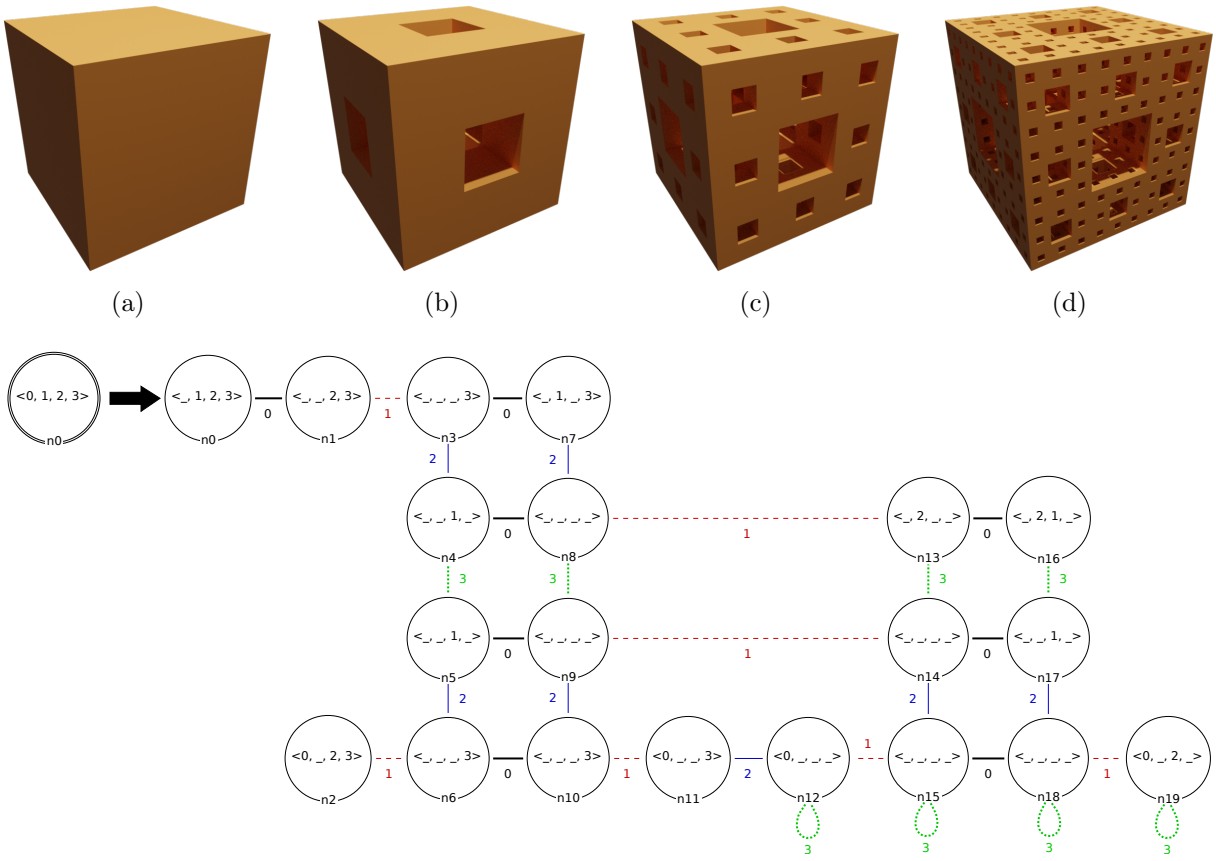


Figure 12: A cube (a), the first (b), second (c) and third(d) iterations of the Menger sponge. The inferred operation (e) from the objects of Figures (a) and (b).

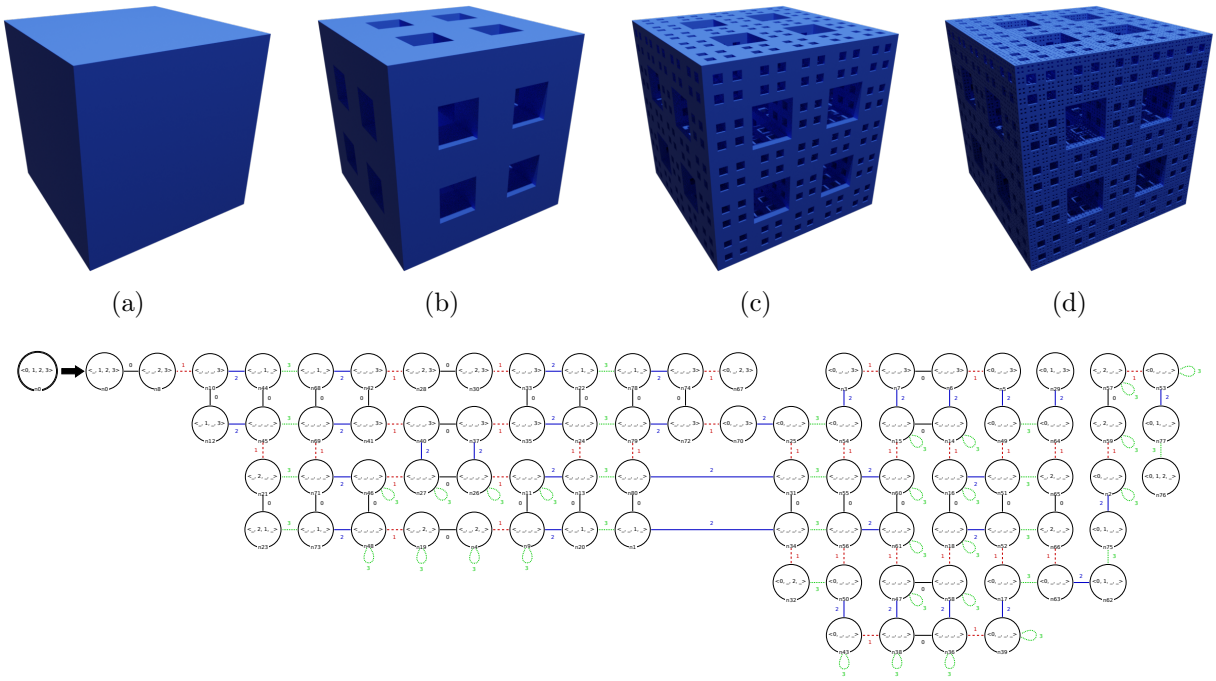
respectively illustrated in Figures 13(c) and 13(d). For information, it took around 100ms to infer the $(2, 2, 2)$-Menger operation.

5 Advanced exploitation

We discuss certain practical side-effects of our inference mechanism. The benefits and limitations are mainly related to Jerboa's formalism to handle modeling operations on generalized maps.

5.1 Target cell parameter

Operations are inferred for a given orbit. Intuitively, an orbit is an abstraction of a cell (or a subcell), described by a subset of dimensions. Therefore, the inferred rule for Powell-Sabin 6-split does not specify that it subdivides triangles: we can apply it to any surface. For instance, we can use this operation to triangulate the quad mesh of Figure 14(a). We obtain the triangulated mesh of Figure 14(b). We can iterate the subdivision to obtain the mesh of Figure 14(c). Similarly, the representation of Suzanne in Figure 8(b) is a surface that only

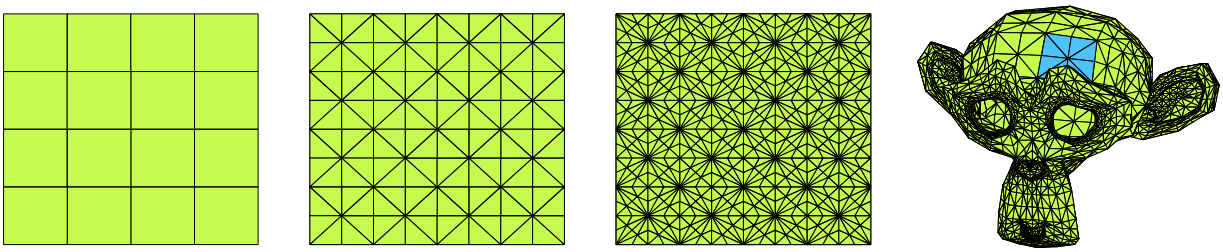


(a) (b) (c) (d)

(e)

Figure 13: A cube (a), the first (b), second (c) and third(d) iterations of the $(2, 2, 2)$-Menger operation. The inferred operation (e) from the objects of Figures (a) and (b).

consists of quads. We can use the inferred operation of Figure 11(e) to obtain a triangulation of the mesh, as illustrated in Figure 14(d).



(a) (b) (c) (d)

Figure 14: Powell-Sabin 6-split on quad meshes: (a) a quad subdivision of the unit square, (b) its triangulation with Powell-Sabin 6-split, (c) further refinement of the triangulation, and (d) the triangulation of Suzanne.

The proposed algorithm folds the graph correctly to obtain a valid rule by traversing all darts of the application example. The inferred operation is always valid for the provided example but might be too sensitive in some instances. Especially, the external, unmodified parts of the object may be captured by the generated operation, hindering its applicability. For instance, the inferred Menger operation assumes an isolated volume and cannot be iterated directly. Concretely, we inferred the operation on two adjacent cubes to ensure that the generated rule does not isolate the volume.

5.2 Inference of other operations

We presented several iterated function systems in Section 4. However, our approach is not limited to these specific operations. Indeed, the algorithm presented in Section 3 takes as input two generalized maps and an orbit type. Any topological operation can be inferred. For instance, if we transform the square face of Figure 15(a) into the cube of Figure 15(d), we infer the rule presented in Figure 15(g) when specifying the face as the input cell. This operation corresponds to the face extrusion. We can visualize the operation by applying it on the triangle of Figure 15(b) and the octagon of Figure 15(c) to obtain the prisms of Figures 15(e) and 15(f).

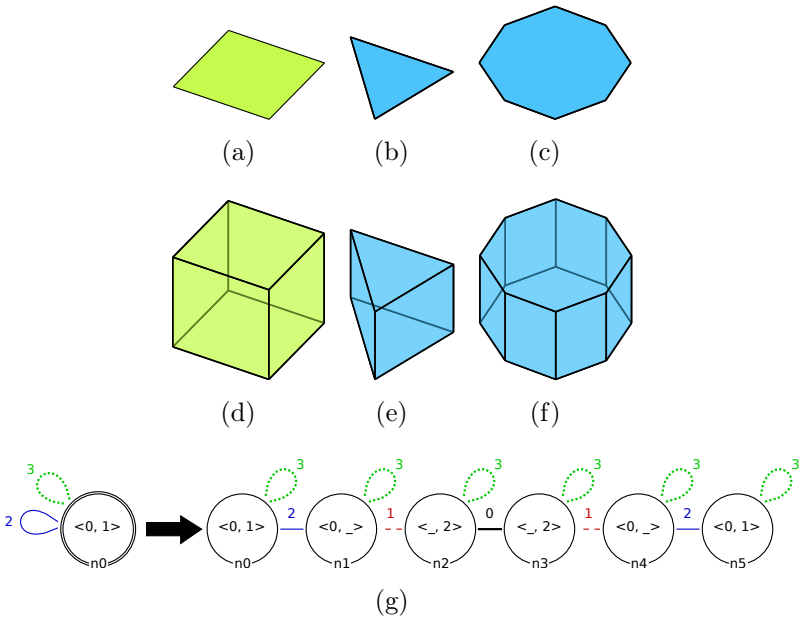


Figure 15: Face extrusions: (a) a square face and (d) its face extrusion as a cube, (g) the inferred rule and its application to faces (b) and (c) to obtain prisms (e) and (f).

5.3 Plurality of inferred operations

We explained in Section 3 our algorithm for inferring topological operations from an example for a given orbit. Intuitively, the algorithm tries to fold the $\kappa(L, R)$ graph for the provided orbit by choosing an initial dart. Throughout the paper, we illustrated our approach with iterated function systems where the left-hand side of the rule consists of only one node. Therefore, any initial dart yields the same rule. It could be the case that several rules exist, e.g., when the left-hand side of the rule has more than one node. Besides, there are possibilities for failure when either the implicit or explicit arcs cannot be reconstructed. Thus, we implemented a mechanism for marking the darts that correspond to the hook node during the computation of the algorithm. This mechanism avoids exploring darts for which we are assured to obtain a rule that has already been generated or to fail to result in a rule. On rules with only one node in the left-hand side, we only have to run the algorithm a single time. On the contrary, only a subset of the initial object's darts is marked when the left-hand side of the obtained rule contains several nodes. We can try the algorithm again with an unmarked dart until all darts are marked. Based on the symmetry of the initial and target objects, we might obtain the same

rule several times. To discard duplicated rules, we use a vertex invariant algorithm similar to [23] with the addition of the orbit variables as labels to speed up the computation [14]. Let us illustrate this mechanism with the folding of the cube from a vertex (see Figure 5). Since the algorithm terminates on a, we can mark all darts of $G\langle 1,2\rangle(a)$ and not use them again as a possible choice to start the algorithm. The orbit of a vertex on the cube always contains 6 darts. Therefore we reduce the computation from 48 to $48 \div 6 = 8$ initial darts (the number of vertices). We can run the algorithm on each vertex and obtain 8 valid rules. However, because the cube is completely symmetric, all 8 rules are isomorphic, and we only keep one.

6 Related Works

Procedural modeling refers to a set of techniques used in computer graphics to derive a model from a ruleset. These techniques avoid having to manually edit objects, proving fruitful to model regular objects, i.e., objects with many repetitions of sub-patterns. Procedural modeling techniques have been exploited to generate plants [29], terrain [33], buildings [24], or cities [25]. Since these techniques could not guarantee an output faithful to the designer's idea, they were extended to inverse procedural modeling. These new approaches try to discover the correct parametrized rules and values thanks to machine learning. Inverse procedural modeling techniques have proven successful in most of the domains where procedural modeling was used, namely trees [34], building facades [35], weather simulation on urban models [11], virtual worlds [10], and texture modeling [15]. Although the rules in Jerboa do not directly belong to procedural modeling techniques, the approach explained in this paper can be seen as similar to inverse procedural modeling. Indeed, we are trying to discover the correct rule parametrized by a given orbit.

Our approach is dual to the construction recently presented in [20]. This approach takes the Loop subdivision scheme [21] as the atomic operation for refinement. A neural network is then used to learn the geometric values for the subdivision. In comparison, we infer subdivisions from a topological perspective. Therefore, we can reconstruct any subdivision scheme.

Jerboa's formalism is a domain-specific language within graph transformations, namely for topology-based geometric modeling. In this paper, we presented an algorithm to reconstruct modeling operations from an example. Similar ideas were used in [22] to reconstruct a graphical modeling environment for domain-specific language using yED. In [6], the authors used graph transformations as a learning mechanism to detect and fix bugs in Javascript programs. A rule-based definition of geometric modeling operation has been used in [16] to predict operations.

In [36], the authors infer the sequence of modeling operations as a sequence of sketches, extrusions and boolean operations. It extends previous works such as [32], [8] or [17] using constructive solid geometry (CSG). These works retrieve a CSG tree to obtain a specific object while we infer a generalized operation.

7 Conclusion and perspectives

We presented an automated method to infer topological modeling operations from a representative example. Our algorithm takes as input an example of the application of the operation and the parametrization orbit. Objects are represented with the topology-based model of generalized maps, defined as edge-labeled graphs. Operations are inferred as graph transformation rules

with variables. These variables specify the orbit involved in the operation. Implementation was done in the Jerboa platform to exploit its formal language, ensuring the well-formedness of the inferred rules. Therefore, any inferred rule preserves the topological consistency of generalized maps. Rules are provided within the Jerboa platform with syntactic conditions ensuring that any modified object is a well-formed object according to the model of generalized maps. Our approach for inferring topological operations exploits the orbit-based definition of Jerboa's rule to fold objects along a given orbit. We detailed our algorithm on the folding of the cube with the vertex orbit type. The inference of a rule relies on the addition of the κ-links connecting the nodes preserved by the operation. Once the algorithm terminates, removing these arcs splits the scheme graph in two and yields the designed operation. The inferred operation is directly applicable to any object by matching the rule's hook into an orbit of the appropriated type in the object.

Our approach offers the following two main benefits:

- First, the automated inference allows a user unaccustomed to either topological models, i.e., generalized maps, or (graph) transformation rules can design operations exclusively from examples.

- Secondly, this process allows for inlining a sequence of elementary operations and generates a direct transformation that can speed up the design of complex scenes.

The algorithm assumes that the graph $\kappa(L, R)$ is connected. Nonetheless, if this graph is not connected, the operation we are trying to infer corresponds to the parallel application of several operations. It seems acceptable to restrict the inference of one operation at a time as a first step.

Another limitation of our algorithm is the lack of context consideration. Indeed, we construct rule schemes from two generalized maps before and after modification. The instantiation of the left-hand side, therefore, necessarily leads to a generalized map. It is then impossible to infer an operation that filters only a part of the G-map. We wish to integrate the detection of unmodified parts of the object thanks to geometric analysis.

Moreover, we only handled the topological structure of objects here. The inferred rules were manually edited to add the missing geometry before applying them again. Therefore, we ought to develop a method to infer computations of embeddings in the rule schemes. The inference of the missing geometry will need to be generic to benefit from the orbit-based generalization. We believe that the methods presented in [20] could be adapted to our needs.

Nonetheless, the approach presented here ensures that the topological part of modeling operations can be inferred without writing a single line of code in a standard programming language or designing a rule in Jerboa's expert language. It also hides the intern structure of Jerboa's rules which are cumbersome, if not impossible, to write or read when they have too many nodes. Finally, a rapid inference mechanism offers an optimist alternative to develop new topology-based geometric operations.

References

[1] H. Belhaouari, A. Arnould, P. Le Gall, and T. Bellet. Jerboa: A Graph Transformation Library for Topology-Based Geometric Modeling. In H. Giese and B. König, editors, *Graph*

Transformation (ICGT 2014), volume 8571 of *Lecture Notes in Computer Science*, pages 269–284, Cham, 2014. Springer International Publishing.

[2] T. Bellet, A. Arnould, H. Belhaouari, and P. Le Gall. Geometric modeling: Consistency preservation using two-layered variable substitutions. In J. de Lara and D. Plump, editors, *Graph Transformation (ICGT 2017)*, Lecture Notes in Computer Science, pages 36–53, Cham, 2017. Springer.

[3] T. Bellet, M. Poudret, A. Arnould, L. Fuchs, and P. Le Gall. Designing a topological modeler kernel: a rule-based approach. In *Shape Modeling International Conference (SMI)*, pages 100–112. IEEE, 2010.

[4] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin. Quad-Mesh Generation and Processing: A Survey. *Computer Graphics Forum*, 32(6):51–76, Sept. 2013.

[5] G. Damiand and P. Lienhardt. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. CRC Press, 2014.

[6] E. Dinella, H. Dai, Z. Li, M. Naik, L. Song, and K. Wang. Hoppity: Learning Graph Transformations to Detect and Fix Bugs in Programs. In *International Conference on Learning Representations*, 2020.

[7] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, Nov. 1978.

[8] T. Du, J. P. Inala, Y. Pu, A. Spielberg, A. Schulz, D. Rus, A. Solar-Lezama, and W. Matusik. InverseCSG: Automatic conversion of 3D models to CSG trees. *ACM Trans. Graph.*, 37(6), Dec. 2018.

[9] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin Heidelberg, 2006.

[10] A. Emilien, U. Vimont, M.-P. Cani, P. Poulin, and B. Benes. WorldBrush: interactive example-based synthesis of procedural virtual worlds. *ACM Transactions on Graphics*, 34(4), July 2015.

[11] I. Garcia-Dorado, D. G. Aliaga, S. Bhalachandran, P. Schmid, and D. Niyogi. Fast Weather Simulation for Inverse Procedural Design of 3D Urban Models. *ACM Transactions on Graphics*, 36(2), Apr. 2017.

[12] J. Guzmán, A. Lischke, and M. Neilan. Exact sequences on Powell–Sabin splits. *Calcolo*, 57(2):13, Mar. 2020.

[13] R. Heckel and G. Taentzer. *Graph Transformation for Software Engineers: With Applications to Model-Based Development and Domain-Specific Language Engineering*. Springer International Publishing, Cham, 2020.

[14] S.-M. Hsieh, C.-C. Hsu, and L.-F. Hsu. Efficient Method to Perform Isomorphism Testing of Labeled Graphs. In M. L. Gavrilova, O. Gervasi, V. Kumar, C. J. K. Tan, D. Taniar, A. Laganá, Y. Mun, and H. Choo, editors, *Computational Science and Its Applications - ICCSA 2006*, Lecture Notes in Computer Science, pages 422–431, Berlin, Heidelberg, 2006. Springer.

[15] Y. Hu, J. Dorsey, and H. Rushmeier. A novel framework for inverse procedural texture modeling. *ACM Transactions on Graphics*, 38(6), Nov. 2019.

[16] T. Igarashi and J. F. Hughes. A suggestive interface for 3d drawing. SIGGRAPH '07, page 20, New York, NY, USA, 2007. Association for Computing Machinery.

[17] K. Kania, M. Zięba, and T. Kajdanowicz. UCSG-Net – Unsupervised Discovering of Constructive Solid Geometry Tree. Oct. 2020. arXiv: 2006.09102.

[18] P. Lienhardt. Subdivisions of N-dimensional Spaces and N-dimensional Generalized Maps. In *Proceedings of the Fifth Annual Symposium on Computational Geometry*, SCG '89, pages 228–236, New York, NY, USA, June 1989. Association for Computing Machinery.

[19] P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(11):59–82, Feb. 1991.

[20] H.-T. D. Liu, V. G. Kim, S. Chaudhuri, N. Aigerman, and A. Jacobson. Neural subdivision. *ACM Transactions on Graphics*, 39(4), July 2020.

[21] C. Loop. Smooth Subdivision Surfaces Based on Triangles. Master's thesis, The University of Utah, January 1987.

[22] J. J. López-Fernández, A. Garmendia, E. Guerra, and J. de Lara. An example is worth a thousand words: Creating graphical modelling environments by example. *Software & Systems Modeling*, 18(2):961–993, 2019.

[23] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, Jan. 2014.

[24] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 614–623, New York, NY, USA, July 2006. Association for Computing Machinery.

[25] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 301–308, New York, NY, USA, Aug. 2001. Association for Computing Machinery.

[26] M. Poudret, A. Arnould, J.-P. Comet, and P. Le Gall. Graph Transformation for Topology Modelling. In H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, editors, *Graph Transformations (ICGT 2008)*, volume 5214 of *Lecture Notes in Computer Science*, pages 147–161, Berlin, Heidelberg, 2008. Springer.

[27] M. Poudret, J.-P. Comet, P. Le Gall, A. Arnould, and P. Meseure. Topology-based geometric modelling for biological cellular processes. In *International Conference on Language and Automata Theory and Applications*, pages 497–508, 2007.

[28] M. J. D. Powell and M. A. Sabin. Piecewise Quadratic Approximations on Triangles. *ACM Transactions on Mathematical Software*, 3(4):316–325, Dec. 1977.

[29] P. Prusinkiewicz, A. Lindenmayer, and J. Hanan. Development models of herbaceous plants for computer imagery purposes. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, volume 22 of *SIGGRAPH '88*, pages 141–150, New York, NY, USA, June 1988. Association for Computing Machinery.

[30] L. Richaume, G. Largeteau-Skapin, R. Zrour, and E. Andres. Unfolding Level 1 Menger Polycubes of Arbitrary Size With Help of Outer Faces. In *Discrete Geometry for Computer Imagery (DGCI)*, Paris, France, Mar. 2019.

[31] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations*, volume Foundations. World Scientific Publishing Co., Inc., USA, Feb. 1997.

[32] G. Sharma, R. Goyal, D. Liu, E. Kalogerakis, and S. Maji. CSGNet: Neural Shape Parser for Constructive Solid Geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5515–5523, June 2018.

[33] R. M. Smelik, K. J. De Kraker, T. Tutenel, R. Bidarra, and S. A. Groenewegen. A survey of procedural methods for terrain modelling. In *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, pages 25–34, jun 2009.

[34] O. Stava, S. Pirk, J. Kratt, B. Chen, R. Měch, O. Deussen, and B. Benes. Inverse Procedural Modelling of Trees. *Computer Graphics Forum*, 33(6):118–131, 2014.

[35] F. Wu, D.-M. Yan, W. Dong, X. Zhang, and P. Wonka. Inverse procedural modeling of facade layouts. *ACM Transactions on Graphics*, 33(4), July 2014.

[36] X. Xu, W. Peng, C.-Y. Cheng, K. D. Willis, and D. Ritchie. Inferring cad modeling sequences using zone graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6062–6070, June 2021.