EG UK Computer Graphics & Visual Computing (2023)
D. Hunter and P. Vangorp (Editors)

Automatic detection of topological changes in modeling operations

M. Gaide[1], D. Marcheix[2], A. Arnould[3], X. Skapin[3], H. Belhaouari[3] and S. Jean[2]

[1]ISAE-ENSMA Poitiers, Université de Poitiers, LIAS, Poitiers, France
[2]Université de Poitiers, ISAE-ENSMA Poitiers, LIAS, Poitiers, France
[3]Université de Poitiers, Univ. Limoges, CNRS, XLIM, Poitiers, France

Abstract

Advanced geometric modelers require the detection of topological changes caused by modeling operations such as edge creation, face splitting or volume merging... Such a detection can be dynamically performed by comparing all topological cells (vertices, edges, faces, volumes) before and after each modification, which can be very time consuming. Then, for some events generated in a systematic way, it can also be performed statically before applying each operation, but it entails several hurdles due to the lack of formalization of such events: while some events may seem obvious, others may not appear intuitively or systematically, and this work of defining events needs to be done again for each newly developed operation.

In this paper, we propose to formalize the static detection of events and to automate this process based on automatic analysis of operations. To achieve this, we leverage on the formalism of graph transformation rules to describe geometric operations, and on the topological model of G-maps that enables homogeneous modeling of manifold geometric objects in any dimension. The syntactic analysis of rules enables the detection of all events that can be detected statically and also specifies the cells on which events that can only be detected dynamically could occur. With this approach, any new operation can be developed faster within the modeler, ensuring a complete, accurate and automatic event detection.

CCS Concepts

• Computing methodologies $\rightarrow$ ***Shape modeling; • Theory of computation*** $\rightarrow$ *Rewriting systems;*

Keywords *: Topology-based modeling; Topological change detection; Static analysis; Graph transformation rules; Generalized maps;*

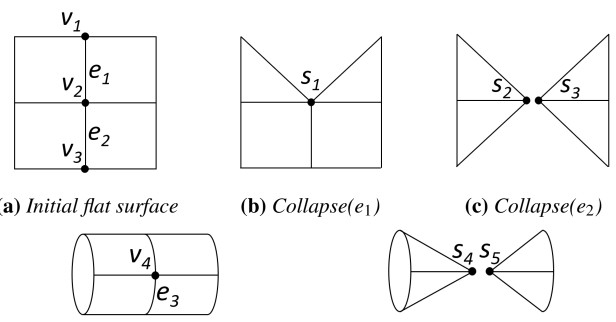
1. Introduction

CAD-oriented systems rely on geometric modeling kernels which allow users to design and maintain control over complex geometric models [Das23; Ope22; Sie22]. An essential feature of such kernels is event detection: the ability to track topological changes (creation, deletion, split, merging and modification) of cells (vertices, edges, faces, volumes) when a modeling operation is applied.

Event detection has a variety of purposes such as, for example, the construction of event logs, persistent naming schemes or modeling optimization. An *event log* is an history-based data structure describing the evolution of cells during the construction of a model. Such data are required for the internal execution of a number of operations provided by kernels and are also passed through the API in order to help developers using those kernels to maintain model consistency [ABC00; BMSB07; LLX*18; Ope22]. A *persistent naming scheme* allows the reevaluation of models regardless of parameter changes, or their transfer toward other systems, and directly or indirectly depends on events detection to propose unique and persistent names for cells [Kri95; MP02; CMHK12; BNB05; FH18]. Events detection is also useful for optimizing the modeling process. For example, operation conflicts in feature-based collaborative CAD systems can be detected and resolved by tracking topological entity changes [CHWZ16].

Most current tools detect events dynamically. A dynamic detection requires comparing a model before and after the application of any operation [Ope22; BMSB07]. On complex models that contain a significant number of cells, the cost in time to fully perform event tracking can be relatively high. To reduce this cost, it is possible to statically detect some local events. For example, the triangulation of a face generates vertex modifications, vertex creations, face subdivision, and so on. Static detection is based on analyzing operations and allows events to be described and calculated before those operations are actually performed. It is up to the developer of the operation to manually define and describe these events, which raises a number of predicaments. First, there is currently no standard formalization for describing these events. For instance, depending on the system, an edge flip operation between two triangles may be interpreted as a modification of both triangles or as a split and/or merging of those triangles. Second, the task of defining and describing events needs to be repeated every time a new operation is developed. Third, this manual definition can introduce errors because while some events may seem obvious, others may not be in-

DOI: 10.2312/cgvc.20231186



(a) *Initial flat surface* **(b)** *Collapse(e_1)* **(c)** *Collapse(e_2)*

(d) *Initial cylindrical surface* **(e)** *Collapse(e_3)*

Figure 1: *Various edge collapses.*

tuitively apparent or can be forgotten. Event detection may then be incomplete or erroneous. For example, consider the edge collapse operation used in geology for studying soil erosion [BCS*14]. This operation is illustrated in Fig.1. Quite intuitively, the collapse of an edge results in the merging of both vertices at its endpoints. This can be observed in Fig.1b, where vertices V_1 and V_2 are merged into a single vertex S_1. However, when the same operation is applied again (Fig.1c), it leads to the merging of vertices S_1 and V_3 and the subsequent split into two vertices S_2 and S_3, in the case of a classical manifold representation. The same operation applied to an edge of a cylinder can even result in no vertex merging at all, but only a split (Fig.1e).

In this paper, we propose to formalize the static detection of local events and automate this process based on automatic analysis of operations. To achieve this, we use the formalism of graph transformation rules to describe geometric operations, specifically using the Jerboa library [BALB14] which facilitates the development of modelers dedicated to specific applications. Jerboa is based on the Generalized Maps (or "G-maps") topological model [Lie91; DL14], which corresponds to a specific class of labeled graphs and enables homogeneous modeling of oriented or non-oriented manifold geometric objects in any dimension. The rules are constructed through the Jerboa interface, which allows for rapid development of modeling operations while ensuring the topological consistency of the underlying geometric model. As we show in the following, a syntactic analysis of each rule enables the detection of local events and also specifies the cells on which global events could occur (events that can only be detected dynamically). This detection becomes much faster, as only these cells need to be verified during the application of the operation. With this approach, any new operation can be developed within the modeler, ensuring an automatic, complete and accurate event detection.

Section 2 introduces the concepts of generalized maps and Jerboa's graph transformation rules, along with their associated vocabulary. Section 3 formalizes topological events and details the static analysis to perform in order to automatically detect these events. Section 4 presents an example of event log integrating a list of automatically detected events. Section 5 provides a time cost comparison between static and dynamic event detection. Section 6 concludes and proposes some perspectives.

2. Main concepts

In this section, we present the generalized maps, the graph transformation rules, and the concepts upon which our method is based.

2.1. Generalized maps

Generalized maps (or G-maps) [Lie91; DL14] allow the representation of manifold geometric objects (with or without boundaries), based on some cellular n-dimensional topological structure.

The representation of an object as a G-map intuitively comes from its decomposition into topological cells (vertices, edges, faces, volumes, and so on). For example, the 2D geometrical object shown in Fig. 2a is represented as a 2-dimensional G-map (or 2-G-map) as follows. The object is first decomposed into faces (Fig. 2b). These faces are *linked* along their common edge with a 2-link, drawn in blue. Each edge on the border of the object is connected to itself (the blue 2-link forms a loop). The index "2" means that the link connects two 2-dimensional (possibly a single one) faces. In order to simplify the reading, 2 labels will not be written in every subsequent figure. In the same way, faces are split into edges connected with red 1-links (Fig 2c). Lastly, edges themselves are split into vertices with black 0-links (Fig 2d) to produce the 2-G-map describing the objects shown in Fig. 2a. A G-map is therefore a graph, the nodes of which are the vertices (named *darts*) obtained at the end of the process and the edges are i-links.

G-maps have conditions guaranteeing objects consistency: for example, two faces are always linked along an edge.

Topological cells are not explicitly represented in G-maps but only implicitly defined as subgraphs. They can be computed using graph traversals defined by an originating node and by a given set of link labels named *orbit*. For example, the 2-cell adjacent to some dart a (or the object face matching a) (Fig. 3a) is the subgraph which contains a and all darts reachable from a, using links labeled 0 or 1 (*i.e.* darts a, b, c, d, e, f, g and h) and the links themselves. This subgraph is denoted by $G\langle 0,1\rangle(a)$ and models the face $BCED$ (Fig. 2a). $\langle 0,1\rangle$ is the *type* of the orbit. The 1-cell adjacent to a (or the object edge matching a) (Fig. 3b) is the subgraph $G\langle 0,2\rangle(a)$ which contains a and all reachable darts using links labeled 0 or 2 (*i.e.* darts a, b, l and m) and the corresponding links. It represents the topological edge BC. Thanks to 2-loops on the object border, $G\langle 0,2\rangle(c)$ (Fig. 3c) also represents the edge CE matching dart c. The 0-cell adjacent to a (or the object vertex matching dart a) (Fig. 3d) is the subgraph $G\langle 1,2\rangle(a)$ and represents vertex B. Note that orbits are more general than cells. For example, the face edge $G\langle 0\rangle(a)$ (Fig. 3e) is an $\langle 0\rangle$ orbit adjacent to a.

2.2. Graph transformation rules

Jerboas's [BALB14] graph transformation rules allow the formalisation of operations over G-maps.

In a few words, a *rule* $r : L \longrightarrow R$ and a *match* $m : L \rightarrow G$ to a G-map G, describe the transformation $G \longrightarrow^{r,m} H$ from G to H. m allows replacing a sub-graph of G described by the left-hand side of the rule L with another one described by the right-hand side R, in order to produce H.

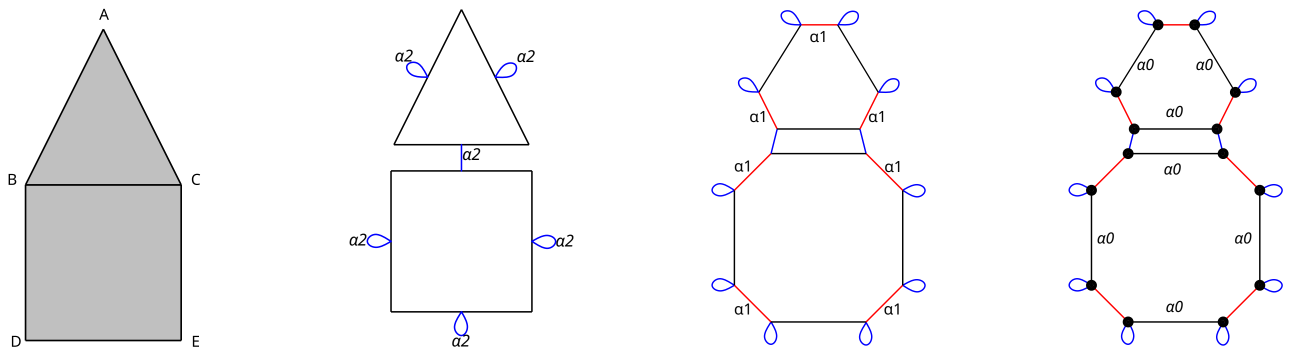


Figure 2: *Cell decomposition of a geometric 2D object*

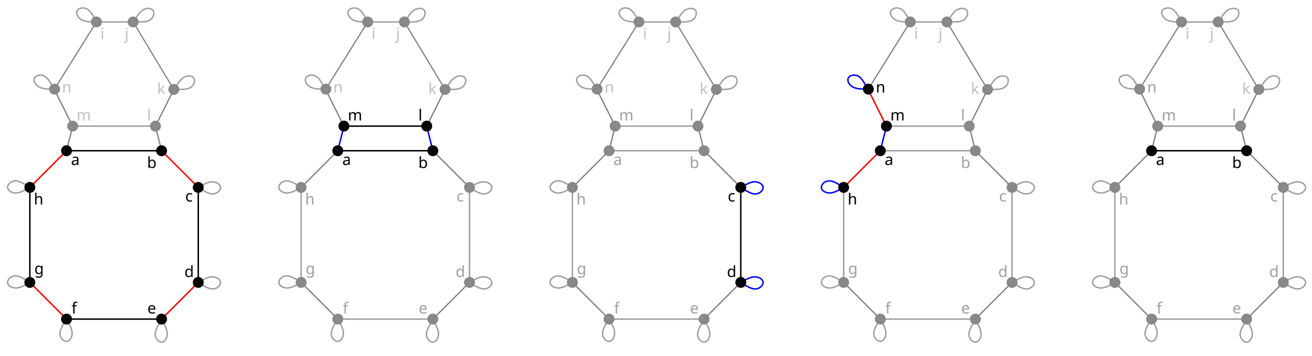


Figure 3: *Orbit decomposition of a geometric 2D object*

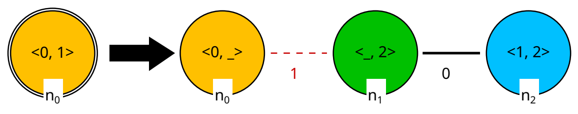
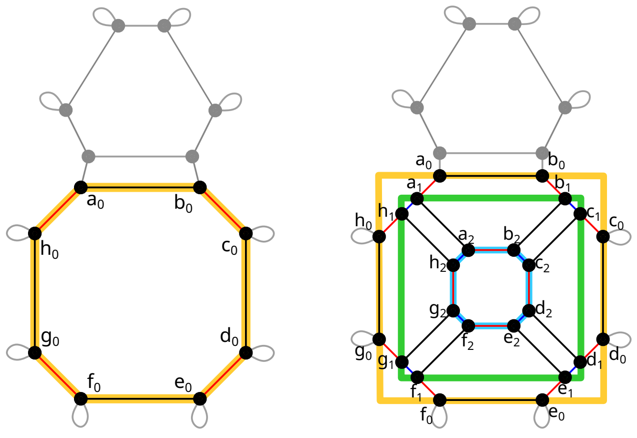


Figure 4: *Rule $L \longrightarrow R$ of face triangulation*



(a) *Graph G* **(b)** *Graph H*

Figure 5: *Application of the triangulation rule (Fig. 4) on matched dart a_0*

Informally, in the triangulation rule shown in Fig. 4, the left side is made of only one node n_0 labeled with the $\langle 0, 1\rangle$ (*i.e.* face) type: this way, it can match any face. Consider for instance the node n_0 from L and the dart a_0 from G shown in (Fig. 5a), respectively. Matching n_0 with a_0, the whole face $G\langle 0,1\rangle(a_0)$ is matched, highlighted in orange in the figure. 0- (resp. 1-) links are represented by black (resp. red) segments. On the right side, the node n_0 has label $\langle 0, \_\rangle$. This means that when applying this rule, 0-links of nodes n_0 have been preserved, while 1-links have been deleted. Hence, in H (Fig. 5b), the 0-links of the matched orange face have been preserved while 1-links have been deleted. In other words, the edges of the orange face are disconnected in H. The new node n_2 of R creates, in H, darts copied from the matched face. This is why there are eight blue darts (a_2, b_2 ... h_2) created from orange darts (a_0, b_0 ... h_0). Because n_2 is labeled $\langle 1, 2\rangle$, the orange 0- (resp. 1-) links on the left side of the rule are relabeled to 1- (resp. 2-) links in the blue copy. Therefore, the rule creates a dual vertex to the matched face. Finally, the node n_1 of R creates a green copy with eight darts (a_1, b_1 ... h_1), deletes the left-side 0-links and relabels the left-side 1-links to 2-links.

All the highlighted links (Fig. 5) are referred to as *implicit links* in the rule nodes. Conversely, the links connecting the nodes together in R are referred to as *explicit links*. For example, the explicit 0-links between n_1 and n_2 link one-to-one green and blue darts. Therefore, the rule creates the four new edges $\langle 0,2\rangle$ in the triangulation of the matched square face. By the same token, the explicit 1-link between n_0 and n_1 link one-to-one orange and green darts.

The n_0 node (Fig. 4) is a *preserved node* because it belongs to both the left and right sides of the rule. Nodes n_1 and n_2 are *created nodes* because they belong only to the right side of the rule. This rule does not contain any *deleted node* because none of its nodes belongs only to the left side.

The concept of *orbit* is extended to patterns of rules. For example, in the right-hand side of the triangulation rule (Fig. 4), the $\langle 0,2\rangle$-orbit (an edge) incident to node n_0 contains the single node n_0, and the $\langle 0,2\rangle$-orbit incident to n_1 contain the two nodes n_1 and n_2 and the explicit 0-link which connects them. Additionally, an $\langle o\rangle$-orbit is said to be *complete* if each node in the orbit matches one link per label of $\langle o\rangle$ either explicitly or implicitly. For example, the $\langle 0,2\rangle$-orbit incident to node n_0 in the right-hand side of the triangulation rule (Fig. 3) is not complete, because node n_0 has no 2-link, neither implicitly nor explicitly. Conversely, the $\langle 0,2\rangle$-orbit incident to n_1 is complete, because its two nodes n_0 and n_1 are incident to the explicit 0-link and both have an implicit 2-link. Note that an $\langle o\rangle$-orbit in a graph is entirely matched by a rule pattern if and only if the corresponding $\langle o\rangle$-orbit in the pattern is complete. For example, the $\langle 0,2\rangle$-orbit incident to a_0 in the graph H (Fig. 5b) is partially matched by the triangulation rule (only darts a_0 and b_0 are matched and not their 2 neighbors), because the $\langle 0,2\rangle$-orbit incident to n_0 in the rule is not complete. Conversely, the $\langle 0,2\rangle$-orbit incident to a_1 in graph H is entirely matched by the triangulation rule (both darts a_1 and h_1 are matched by node n_1 and darts a_2 and h_2 are matched by node n_2), because the $\langle 0,2\rangle$-orbit incident to n_1 in the rule is complete.

Jerboa's rules provide syntactic properties guaranteeing the preservation of G-maps consistency.

3. Event detection

Basically, detecting an event occurring on an orbit after the application of a rule implies comparing the graph before and after this application [TSRA17]. However, many events can be statically detected before the rule is applied. As seen previously, entrusting the developer with the task of detecting and formalizing these events for each new operation paves the way to various issues. We propose to formalize the different events (creation, deletion, split, merging, modification, and non-modification) and their detection in order to automate this process through analysis and comparison of rules' nodes.

3.1. Creation

Any node of a rule matches at least one dart. So when a rule creates a node, it also creates the corresponding darts. If the right-hand side of a rule contains a created orbit, then applying this rule on any object creates one or several orbits. For example, the triangulation rule (Fig. 4) creates the edge orbit $R\langle 0,2\rangle(n_1)$. Therefore, the application of this rule on the square face (Fig. 5) creates four edge orbits $H\langle 0,2\rangle(a_1)$, $H\langle 0,2\rangle(c_1)$, $H\langle 0,2\rangle(e_1)$ and $H\langle 0,2\rangle(g_1)$ in H.

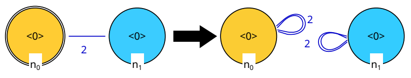


Figure 6: *Rule* $r : L \longrightarrow R$ *of unsewing* 2*-links along an edge* ($\langle 0,2\rangle$*-orbit)*

The creation of an orbit is defined as follows :

Definition 3.1.1 (Orbit creation) Let $r : L \longrightarrow R$ be a rule, $m : L \to G$ a match, $t : G \longrightarrow^{r,m} H$ the transformation of G by (r,m), $\langle o\rangle$ an orbit type, n a created node of R and d a dart of H matched by n.
An orbit $R\langle o\rangle(n)$ is *created* in the rule r if all of its nodes are created in r.
An orbit $H\langle o\rangle(d)$ is *created* in the transformation t if and only if all of its darts are created by t.

From the syntactic analysis of a rule, it is then possible to automatically detect the event of orbit creation using the following proposition :

Proposition 3.1.1 Let $r : L \longrightarrow R$ be a rule, $m : L \to G$ a match, $t : G \longrightarrow^{r,m} H$ the transformation of G by (r,m), $\langle o\rangle$ an orbit type, n a created node of R and d a dart of H matched by n.
The orbit $H\langle o\rangle(d)$ is created if and only if $R\langle o\rangle(n)$ is created.

Idea of proof: Thanks to the syntactic conditions on rules, the orbit $R\langle o\rangle(n)$ is complete. Thus, due to the application of rules, $H\langle o\rangle(d)$ is included in the image of $R\langle o\rangle(n)$. Therefore, all darts of $H\langle o\rangle(d)$ are created. Conversely, if $H\langle o\rangle(d)$ is created, then this orbit is included in the image of $R\langle o\rangle(n)$ and $R\langle o\rangle(n)$ is created. □

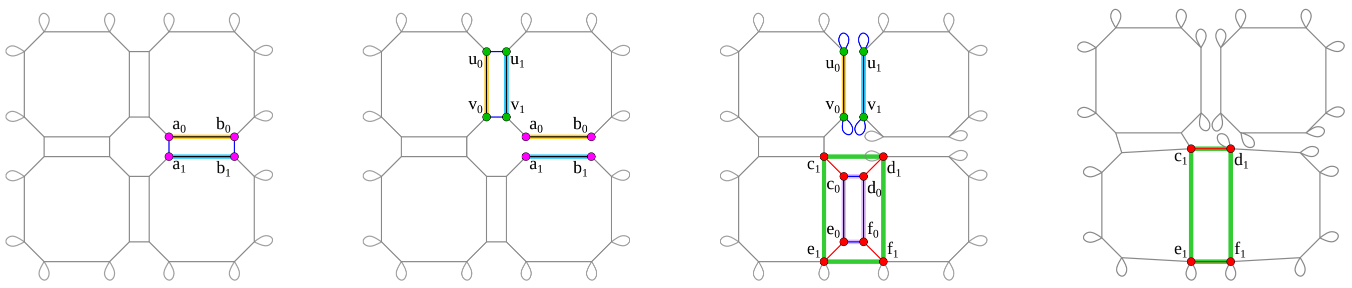
Back to the example of face triangulation (Fig. 4 and 5), we follow the creation of an edge. The edge orbit $R\langle 0,2\rangle(n_1)$ in the triangulation rule is created because both its nodes n_1 and n_2 are created. In the transformation, n_1 matches eight darts, including for example dart a_1, and n_2 also matches eight darts including dart a_2. The edge orbit $H\langle 0,2\rangle(a_1)$ is created as well as darts a_2, h_1 and h_2 (Fig. 5b).

3.2. Split

When a rule explicitly splits an orbit, it also splits the set of matched orbits. For example, the unsewing rule (Fig. 6) splits the edge orbit $L\langle 0,2\rangle(n_0)$ into a pair of edge orbits $R\langle 0,2\rangle(n_0)$ and $R\langle 0,2\rangle(n_1)$. Therefore, the first application of this rule (Fig. 7b) splits the edge into two edges $H\langle 0,2\rangle(a_0)$ and $H\langle 0,2\rangle(a_1)$.

A rule can also split an orbit implicitly. For example, the triangulation rule (Fig. 4) splits the face orbit $L\langle 0,1\rangle(n_0)$ along the second implicit links since on the right-hand side, every second implicit link of the nodes n_0, n_1 and n_2 belonging to the face orbit $R\langle 0,1\rangle(n_0)$ is either deleted or different from 0 and 1. Hence, applying this rule (Fig. 5) splits $G\langle 0,1\rangle(a_0)$ into four face orbits $H\langle 0,1\rangle(a_0)$, $H\langle 0,1\rangle(c_0)$, $H\langle 0,1\rangle(e_0)$, $H\langle 0,1\rangle(g_0)$.

The split of an orbit is defined as follows :



(a) *Graph G* **(b)** *Graph H* **(c)** *Graph I* **(d)** *Graph J*

Figure 7: *(a) Initial graph G. (b) First application of the* 2*-unsewing rule (Fig. 6): pink darts. (c) Second application of the* 2*-unsewing rule: green darts. (d) Application of the edge deletion rule (Fig. 9): red darts.*

Definition 3.2.1 (Orbit split) Let $r : L \longrightarrow R$ be a rule, $m : L \to G$ a match, $t : G \longrightarrow^{r,m} H$ the transformation of G by (r,m), $\langle o \rangle$ an orbit type, n a preserved node of r and d a preserved dart of t.
An orbit $L\langle o \rangle(n)$ is *split* in the rule r:

- Either *explicitly* if there exists a preserved node n' in $L\langle o \rangle(n)$ that does not belong to the same orbit in R, *i.e.* $R\langle o \rangle(n) \neq R\langle o \rangle(n')$;
- Or *implicitly along the k-th implicit link* if there exists in $L\langle o \rangle(n)$ a node for which the k-th implicit link belongs to $\langle o \rangle$ and for all nodes n' in $R\langle o \rangle(n)$, the k-th implicit link is renamed outside of $\langle o \rangle$.

An orbit $G\langle o \rangle(d)$ is *split* in the transformation t if and only if there exists a preserved dart d' in $G\langle o \rangle(d)$, such that $H\langle o \rangle(d) \neq H\langle o \rangle(d')$.

From the syntactic analysis of a rule, it is then possible to automatically detect the event of orbit split using the following proposition :

Proposition 3.2.1 Let $r : L \longrightarrow R$ be a rule , $m : L \to G$ a match, $t : G \longrightarrow^{r,m} H$ the transformation of G by (r,m), $\langle o \rangle$ an orbit type, n a preserved node of r.
If the orbit $L\langle o \rangle(n)$ is both complete and explicitly split in r, *i.e.* if there exists a preserved node n' in $L\langle o \rangle(n)$ such that $R\langle o \rangle(n) \neq R\langle o \rangle(n')$, then for any preserved darts d and d' of t respectively matching $R\langle o \rangle(n)$ and $R\langle o \rangle(n')$, $G\langle o \rangle(d)$ is split in t into two orbits $H\langle o \rangle(d) \neq H\langle o \rangle(d')$.
If the orbit $L\langle o \rangle(n)$ is both incomplete and implicitly split, the split may be confirmed dynamically on G.

Idea of proof: As a reminder, a complete orbit $L\langle o \rangle(n)$ entirely matches an orbit $G\langle o \rangle(d)$.

In the case of an explicit split in r from $L\langle o \rangle(n)$ to $R\langle o \rangle(n) \neq R\langle o \rangle(n')$, since $L\langle o \rangle(n)$ is complete, then due to rules syntactic conditions, so are $R\langle o \rangle(n)$ and $R\langle o \rangle(n')$. Let d and d' be two preserved darts of t such that d (resp. d') is matched by $L\langle o \rangle(n)$ and $R\langle o \rangle(n)$ (resp. $L\langle o \rangle(n')$ and $R\langle o \rangle(n')$). It follows that $G\langle o \rangle(d)$ is split into $H\langle o \rangle(d) \neq H\langle o \rangle(d')$.

If $L\langle o \rangle(n)$ is complete, and there is an implicit split along the k-th implicit links of $L\langle o \rangle(n)$ in r, then by definition, all the k-th implicit links of $R\langle o \rangle(n)$ are deleted or renamed outside of $\langle o \rangle$. Let n' be a preserved node of $L\langle o \rangle(n)$ with a k-th implicit i-link with i in $\langle o \rangle$, and d and d' two i-linked matched darts of n' in G. Then d and d' may either belong to two different orbits in H ($H\langle o \rangle(d) \neq H\langle o \rangle(d')$); belong to the same orbit in H if the sub-orbit is folded along the split i-link ($H\langle o \rangle(d) = H\langle o \rangle(d')$); or be the same dart if the link is a loop ($d = d'$ and therefore $H\langle o \rangle(d) = H\langle o \rangle(d')$).

Let us assume now that $L\langle o \rangle(n)$ is not complete. Let d and d' be two darts of G matched by $L\langle o \rangle(n)$ which are potentially split. Because $L\langle o \rangle(n)$ is not complete, G can contain some path between d and d' using only links labeled in $\langle o \rangle$ such that this path is not entirely matched by $L\langle o \rangle(n)$. Therefore, this path does not break during the transformation t and both d and d' remain in the same orbit in H: $H\langle o \rangle(d') = H\langle o \rangle(d'')$. □

Referring to the example of the unsewing rule (Fig. 6), we follow the explicit split of an edge. The rule splits the edge orbit $L\langle 0,2 \rangle(n_0)$ because both its nodes n_1 and n_2 form two distinct edge orbits $R\langle 0,2 \rangle(n_0)$ and $R\langle 0,2 \rangle(n_1)$. Rule application (Fig. 7b) entails n_0 (resp. n_1) matching both darts a_0 and b_0 (resp. a_1 and b_1). The edge orbit $G\langle 0,2 \rangle(a_0)$, containing the darts matched by both n_0 and n_1, is split into a pair of orbits $H\langle 0,2 \rangle(a_0)$ (*i.e.* a_0 and b_0) and $H\langle 0,2 \rangle(a_1)$ (*i.e.* a_1 and b_1).

Continuing with the face triangulation rule (Fig. 4), we follow the implicit split event of a face orbit type $\langle 0,1 \rangle$. The orbit $L\langle 0,1 \rangle(n_0)$ contains node n_0 and matches a single set of darts which form a face orbit. The first implicit link of n_0 is 0 and the second implicit link is 1; both belong to orbit type $\langle 0,1 \rangle$. However, the orbit $R\langle 0,1 \rangle(n_0)$ contains nodes n_0, n_1, n_2 and none of them has either 0 or 1 as a second implicit link. Under this condition, we can state that the face triangulation rule splits a face into two or more faces. Applying this triangulation rule on the square face (Fig. 5), $L\langle 0,1 \rangle(n_0)$ matches the face orbit $G\langle 0,1 \rangle(a_0)$). Then, the square face is split into the four faces matching $R\langle 0,1 \rangle(n_0)$: $H\langle 0,1 \rangle(a_0)$, $H\langle 0,1 \rangle(c_0)$, $H\langle 0,1 \rangle(e_0)$ and $H\langle 0,1 \rangle(g_0)$.

Note that both proposition 3.2.1 and its proof idea mention that a split of an $\langle o \rangle$-orbit is guaranteed to happen in a graph transformation when the split is explicit and the orbit is complete in the rule. For example, let us consider the unsewing edge rule (Fig. 6) and its application on a surface (Fig. 7) again but this time, we focus on the surface orbit $L\langle 0,1,2 \rangle(n_0)$. The rule's nodes of this orbit all miss an 1-link and, thus, $L\langle 0,1,2 \rangle(n_0)$ is not complete. As we can

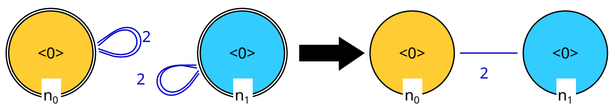


Figure 8: *Rule $r : L \longrightarrow R$ of 2-sewing along two edges ($\langle 0,2\rangle$-orbits)*

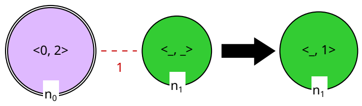


Figure 9: *Rule $r : L \longrightarrow R$ of edge deletion*

see, upon the first application of the rule (Fig. 7b), the surface is not split, although the conditions given in the definition 3.2.1 are met. Yet, in this case, applying the unsewing rule a second time (Fig. 7c) actually splits the surface. Indeed, when an orbit is not complete, the pattern matches only a part of the orbit and detected events must be confirmed dynamically. Fortunately, even if detecting this potential split is a dynamic process, it is not time-consuming, because we know exactly which cell needs to be traversed and which alpha links can potentially generate this split.

3.3. Merging

Similarly to orbit split, there exist explicit and implicit merging orbits.

For example, in the explicit case, the sewing rule (Fig. 8) is the opposite of the unsewing rule (Fig. 6) for both the result of its application and its construction. Let us consider the square plan (Fig. 7) from H to G (Fig. 7b to Fig. 7a). The sewing rule merges two edge orbits $L\langle 0,2\rangle(n_0)$ and $L\langle 0,2\rangle(n_1)$ into a single edge orbit $R\langle 0,2\rangle(n_0)$. Therefore, applying the sewing rule merges two edges of H, $H\langle 0,2\rangle(a_0)$ and $H\langle 0,2\rangle(a_1)$, into one edge $G\langle 0,2\rangle(a_0)$.

The same goes for the implicit merging. For example, in the implicit case, the edge deletion rule (Fig. 9) merges the face vertex orbits mapped to $L\langle 1\rangle(n_0)$ along the second implicit link. In fact, in the left-hand side of the rule, the second implicit link of n_0 is 2 and n_1 has no second implicit link. In the right-hand side of the rule, the second implicit link of n_1 is rewritten as 1. It follows that the application of this rule (Fig. 7d) merges both $I\langle 1\rangle(c_1)$, $I\langle 1\rangle(d_1)$ into $J\langle 1\rangle(c_1)$ and both $I\langle 1\rangle(e_1)$, $I\langle 1\rangle(f_1)$ into $J\langle 1\rangle(e_1)$.

The merging of an orbit is defined as follows :

Definition 3.3.1 (Orbit merging) Let $r : L \longrightarrow R$ be a rule, $m : L \to G$ a match, $t : G \longrightarrow^{r,m} H$ a transformation of G by (r,m), $\langle o\rangle$ an orbit type, n a preserved node of r and d a preserved dart of t.
An orbit $L\langle o\rangle(n)$ is *merged* by the rule r:

- Either *explicitly* if there exists a preserved node n', with $L\langle o\rangle(n) \neq L\langle o\rangle(n')$, such that both n and n' belong to the same orbit in R, *i.e.* $R\langle o\rangle(n) = R\langle o\rangle(n')$.
- Or *implicitly* along the k-th implicit link if there exists in $R\langle o\rangle(n)$ a node whose the k-th implicit link belongs to $\langle o\rangle$ and for all nodes n' in $L\langle o\rangle(n)$, the k-th implicit link does not belong to $\langle o\rangle$.

Two different orbits $G\langle o\rangle(d)$ and $G\langle o\rangle(d')$, with d' a preserved dart, are *merged* in the transformation t if and only if $H\langle o\rangle(d) = H\langle o\rangle(d')$.

From the syntactic analysis of a rule, it is then possible to automatically detect the orbit merging event :

Proposition 3.3.1 Let $r : L \longrightarrow R$ be a rule, $m : L \to G$ a match, $t : G \longrightarrow^{r,m} H$ a transformation of G by (r,m), $\langle o\rangle$ an orbit type, n a preserved node of r.
If the orbit $L\langle o\rangle(n)$ is both complete and explicitly merged, *i.e.* if there exists a preserved node n' in r such that $L\langle o\rangle(n) \neq L\langle o\rangle(n')$ and $R\langle o\rangle(n) = R\langle o\rangle(n')$, then for any preserved darts d and d' of t respectively matched by $L\langle o\rangle(n)$ and $L\langle o\rangle(n')$, $G\langle o\rangle(d)$ is merged with $G\langle o\rangle(d')$ in H, *i.e.* $G\langle o\rangle(d) \neq G\langle o\rangle(d')$ and $H\langle o\rangle(d) = H\langle o\rangle(d')$.
If the orbit $L\langle o\rangle(n)$ is both incomplete and implicitly merged, the merge may be confirmed dynamically on G.

Idea of proof: The idea is analogous to the one developed for Proposition 3.2.1.

If the orbit $L\langle o\rangle(n)$ is both complete and explicitly merged with another orbit $L\langle o\rangle(n')$, where n' is a preserved node of r, then $R\langle o\rangle(n) = R\langle o\rangle(n')$. Because $L\langle o\rangle(n)$ is complete, $L\langle o\rangle(n)$ entirely matches $G\langle o\rangle(d)$ for any preserved dart d of t matched by $L\langle o\rangle(n)$. Due to the injection property of matching, for any preserved dart d' of t matched by $L\langle o\rangle(n')$, d' is not a dart of $G\langle o\rangle(n)$. In other words, $G\langle o\rangle(d) \neq G\langle o\rangle(d')$. Finally, due to the rule application, $R\langle o\rangle(n) = R\langle o\rangle(n')$ implies $H\langle o\rangle(d) = H\langle o\rangle(d')$.

If $L\langle o\rangle(n)$ is complete and if there is an implicit merge along the k-th implicit links of $L\langle o\rangle(n)$ in r, then by definition, all the k-th implicit links of $L\langle o\rangle(n)$ are deleted or named outside of $\langle o\rangle$. Let n' be a preserved node of $L\langle o\rangle(n)$ such that the k-th implicit link of n' is i-labeled in R with i in $\langle o\rangle$. Let d and d' be two darts, matched by n', and i linked in H. Both d and d' belong to the same $\langle o\rangle$-orbit in H ($H\langle o\rangle(d) = H\langle o\rangle(d')$). Then d and d' may either belong to two different orbits in G ($G\langle o\rangle(d) \neq G\langle o\rangle(d')$); belong to the same orbit in G if the rule folds the sub-orbit along the i-link ($G\langle o\rangle(d) = G\langle o\rangle(d')$); or finally be the same dart if the i-link is a loop ($d = d'$ and therefore $G\langle o\rangle(d) = G\langle o\rangle(d')$).

Let us assume now that $L\langle o\rangle(n)$ is not complete. Let d and d' be two preserved darts of t matched by $L\langle o\rangle(n)$ which are potentially merged. Because $L\langle o\rangle(n)$ is incomplete, so is $R\langle o\rangle(n)$, and H can contain some path between d and d' using only links in $\langle o\rangle$ such that this path is not entirely matched by $R\langle o\rangle(n)$. Therefore, this path has not been built during the transformation t and d and d' were already in the same orbit in G ($G\langle o\rangle(d) = G\langle o\rangle(d')$). □

With the example of the sewing rule (Fig. 8), we follow the explicit merging of two edges. The edge orbits $L\langle 0,2\rangle(n_0)$ and $L\langle 0,2\rangle(n_1)$ are complete and merged in the rule because their respective nodes n_1 and n_2 are part of the same edge orbit $R\langle 0,2\rangle(n_0)$. Applying the transformation, n_0 (resp. n_1) matches the darts a_0 and b_0 (resp. a_1 and b_1). Edge orbits $H\langle 0,2\rangle(a_0)$ and $H\langle 0,2\rangle(a_1)$ are merged into orbit $G\langle 0,2\rangle(a_0)$ (*i.e.* a_0, b_0, a_1 and b_1), as shown in Fig. 7a.

With the edge deletion rule (Fig. 9), we follow the implicit merging event of the face vertices (orbit types $\langle 1\rangle$). The orbit $L\langle 1\rangle(n_0)$

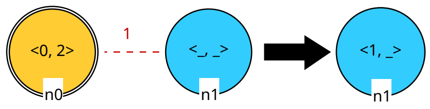
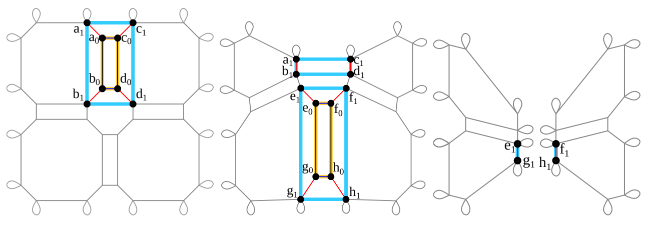


Figure 10: *Rule L $\longrightarrow$ R of edge collapse*



(a) *Graph G* **(b)** *Graph H* **(c)** *Graph I*

Figure 11: *Two successive applications of edge collapse rule (Fig. 10) on matched dart a_0 (a), then on on matched dart e_0 (b)*

contains both nodes n_0 and n_1 and matches a single set of darts representing four face vertices. In this example, since all nodes of $L\langle 1\rangle(n_0)$ do not have an implicit 1-link and since the only remaining node in $R\langle 1\rangle(n_1)$ has such a link, then we can state that face vertices are merged. Applying this edge deletion rule on the bottom vertical edge of the Fig. 7c, $L\langle 1\rangle(n_0)$ matches two pairs of face vertices $(I\langle 1\rangle(c_0), I\langle 1\rangle(d_0))$ and $(I\langle 1\rangle(e_0), I\langle 1\rangle(f_0))$. Then, the face vertices around the edge to delete is merged into the pair of face vertices matched from $R\langle 1\rangle(n_1)$ (*i.e.* $J\langle 1\rangle(c_1)$ and $J\langle 1\rangle(e_1)$).

Note that a potential merge of two faces is also detected in this edge deletion rule. Indeed, the second implicit link of n_1 in $R\langle 0,1\rangle(n_1)$ does not appear in all nodes of $L\langle 0,1\rangle(n_1)$ but this orbit is not complete. Thus, such a potential merge is automatically detected and must be confirmed dynamically depending if both faces incident to the removed edge are different (as shown in Fig. 7d). Similarly, let us consider the first example from Fig. 1 that uses the edge collapse rule introduced in Fig. 10 and the incomplete $L\langle 1,2\rangle(n_0)$ orbit. The first application of the rule (Fig. 11b) does not result in a vertex split, while the second application does (Fig. 11c).

3.4. Other events

The approach for the other events (*deletion*, *non modification* and *modification*) being relatively similar, we will describe them in a more concise way.

3.4.1. Deletion

This event mirrors the creation one. Let us get back to the edge deletion rule (Fig. 9). This rule, as its name suggests, deletes an edge orbit $L\langle 0,2\rangle(n_0)$. Its single node n_0, which by itself matches an edge because it is complete on $\langle 0,2\rangle$, no longer exists on the right side of the rule and, therefore, is deleted. Applying the rule on the darts shown in Fig. 7c, $L\langle 0,2\rangle(n_0)$ matches the orbit $I\langle 0,2\rangle(c_0)$ whose darts (c_0, d_0, e_0 and f_0 are then deleted in J (Fig. 7d), since there are no nodes in R to match them.

The deletion of an orbit is defined as follows :

Definition 3.4.1 (Orbit deletion) Let $r : L \longrightarrow R$ be a rule, $m : L \to G$ a match, $t : G \longrightarrow^{r,m} H$ the transformation of G by (r,m), $\langle o\rangle$ an orbit type, n a node of L and d a dart of G.
An orbit $L\langle o\rangle(n)$ is *deleted* in the rule r if all of its nodes are deleted in r.
An orbit $G\langle o\rangle(d)$ is *deleted* in the transformation t if and only if all of its darts are deleted by t.

From the syntactic analysis of a rule, it is then possible to automatically detect the orbit deleting event :

Proposition 3.4.1 Let $r : L \longrightarrow R$ be a rule, $m : L \to G$ a match, $t : G \longrightarrow^{r,m} H$ the transformation of G by (r,m), $\langle o\rangle$ an orbit type, n a node of L and d a dart of G.
An orbit $L\langle o\rangle(n)$ is deleted in the rule r if and only if the orbit $G\langle o\rangle(d)$ is deleted in the transformation t for any dart d matched by $L\langle o\rangle(n)$.

Idea of proof: Thanks to the syntactic conditions on the rules, a deleted orbit is always complete. Consequently, the corresponding orbits are also completely deleted in the transformation. □

3.4.2. Non-Modification

An orbit is said to be *not modified* when its nodes and links remain unchanged. For example, in the triangulation rule (Fig. 4), the face edge orbit $L\langle 0\rangle(n_0)$ only contains node n_0 and the same goes for the orbit $R\langle 0\rangle(n_0)$. Considering that the implicit 0-link of n_0 is unchanged and that there is no other node in its orbit, we can state that the triangulation rule does not modify the face edges. Applying this rule on the house (Fig. 5), the face edge orbits remain the same and $G\langle 0\rangle(a_0) = H\langle 0\rangle(a_0)$, $G\langle 0\rangle(c_0) = H\langle 0\rangle(c_0)$, $G\langle e_0\rangle = H\langle e_0\rangle$ and $G\langle 0\rangle(g_0) = H\langle 0\rangle(g_0)$ (Fig. 5b).

The non-modification of an orbit is defined as follows :

Definition 3.4.2 (Orbit non-modification) Let $r : L \longrightarrow R$ be a rule, $m : L \to G$ a match, $t : G \longrightarrow^{r,m} H$ the transformation of G by (r,m), $\langle o\rangle$ an orbit type, n a preserved node of r and d a preserved dart of t.
An orbit $L\langle o\rangle(n)$ remains *not modified* in the rule r if $L\langle o\rangle(n) = R\langle o\rangle(n)$ and all the nodes of the orbit have the same label on both sides of the rule, *i.e.* their implicit links are the same in L and R.
An orbit $G\langle o\rangle(d)$ is said to be *not modified* in the transformation t if $G\langle o\rangle(d) = H\langle o\rangle(d)$.

From the syntactic analysis of a rule, it is then possible to automatically detect the orbit non-modification event :

Proposition 3.4.2 Let $r : L \longrightarrow R$ be a rule, $m : L \to G$ a match, $t : G \longrightarrow^{r,m} H$ the transformation of G by (r,m), $\langle o\rangle$ an orbit type, n a preserved node of r and d a preserved dart of t.
If an orbit $L\langle o\rangle(n)$ remains not modified in the rule r, then $G\langle o\rangle(d)$ remains not modified in the transformation t for any dart d of G matched by $L\langle o\rangle(n)$.
If an orbit $G\langle o\rangle(d)$ is not matched by the rule, *i.e.* no dart of $G\langle o\rangle(d)$ is matched by L, then it remains not modified.

Idea of proof: This is obvious, because a rule may modify only the matched part of a graph. □

Orbite type	Detected event	Orbite type	Detected event
$\langle 0\rangle$	Creation $H\langle 0\rangle(a_1)$ Creation $H\langle 0\rangle(b_1)$ ... No modif $H\langle 0\rangle(a_0)$ No modif $H\langle 0\rangle(c_0)$ ...	$\langle 1\rangle$	Creation $H\langle 0\rangle(a_2)$ Creation $H\langle 1\rangle(c_2)$ ... Split $G\langle 1\rangle(a_0) \to \{H\langle 1\rangle(a_0), H\langle 1\rangle(h_0)\}$ Split $G\langle 1\rangle(c_0) \to \{H\langle 1\rangle(c_0), H\langle 1\rangle(b_0)\}$ ...
$\langle 2\rangle$	Creation $H\langle 2\rangle(a_1)$ Creation $H\langle 2\rangle(a_2)$ Creation $H\langle 2\rangle(b_1)$ Creation $H\langle 2\rangle(b_2)$ ... No modif $H\langle 2\rangle(a_0)$ No modif $H\langle 2\rangle(b_0)$ No modif $H\langle 2\rangle(c_0)$ No modif $H\langle 2\rangle(d_0)$ ...	$\langle 0,2\rangle$	Creation $H\langle 0,2\rangle(a_1)$ Creation $H\langle 0,2\rangle(b_1)$... No modif $H\langle 0,2\rangle(a_0)$ No modif $H\langle 0,2\rangle(c_0)$ ...
$\langle 0,1,2\rangle$	modif $H\langle 0,1,2\rangle(a_0)$	$\langle 0,1\rangle$	Split $G\langle 0,1\rangle(a_0) \to \{H\langle 0,1\rangle(a_0), H\langle 0,1\rangle(c_0), H\langle 0,1\rangle(e_0), H\langle 0,1\rangle(g_0)\}$
$\langle 1,2\rangle$	Creation $H\langle 1,2\rangle(a_0)$ Modif $H\langle 1,2\rangle(b_0)$... Creation $H\langle 1,2\rangle(a_2)$		

Table 1: *Extract from the Event log generated after applying the triangulation rule (Fig. 4) on the house in Fig. 5.*

3.4.3. Modification

Generally speaking, an orbit is *modified* when it is neither created, deleted, not modified, split or merged. More specifically, a modified orbit has either modified links, some created part or some deleted part. Also, such a change must not lead to a split nor a merging event. For example, once again considering the triangulation rule, vertex orbit $L\langle 1,2\rangle(n_0)$ contains node n_0 only, while $R\langle 1,2\rangle(n_0)$ contains both nodes n_0 and n_1. The orbit has a created part related to n_1, the second implicit link of n_0 is deleted, and there is no split, either implicit or explicit: thus, the orbit is modified. Applying this rule on the house (Fig. 5) entails vertex orbits $G\langle 1,2\rangle(a_0)$, $G\langle 1,2\rangle(c_0)$, $G\langle 1,2\rangle(e_0)$, $G\langle 1,2\rangle(g_0)$ to be modified with an added part of two new darts each.

4. Event log

Event detection method, defined in a generic manner on rules in section 3, allows automatic generation of an *event log* encompassing all events or potential events that occurred on the object when a rule is applied. To illustrate this with a simple example, let us consider the Event log in Table.1. It corresponds to the application of the triangulation rule (Fig. 4) to the bottom face of the geometric object (Fig. 5). As we can see, events can be automatically detected and listed for each orbit. For the sake of simplicity, we have chosen a rule that matches a complete orbit. Thus, all events can be detected statically. If not, the event log specifies the precise orbit that needs to be dynamically verified.

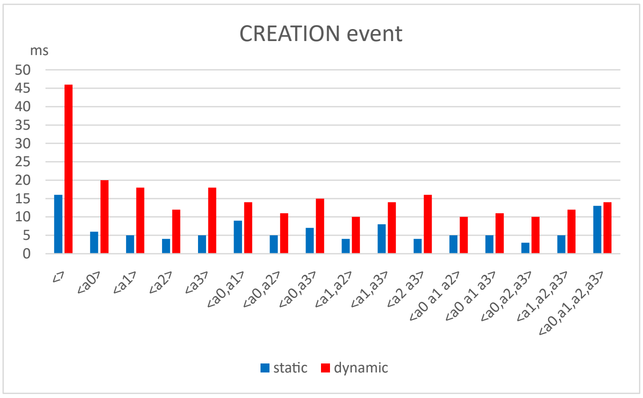
5. Static and dynamic comparison

This section compares a dynamic approach, in which the program manages the evolution of the orbits while traversing a 3D object, with a static approach which exploits the techniques presented in this article. The comparison has been done on an Intel i9-11950H with 32GB under JDK 11. We focus our study on the three main events: creation, split and merge operations. Fig. 12 displays experimental results of our study which measure the computation time of a single representative of the target orbit as the dart with the lowest identifier. Static detection aims at recomputing only modified orbits whereas the dynamic detection would cover the whole mesh.

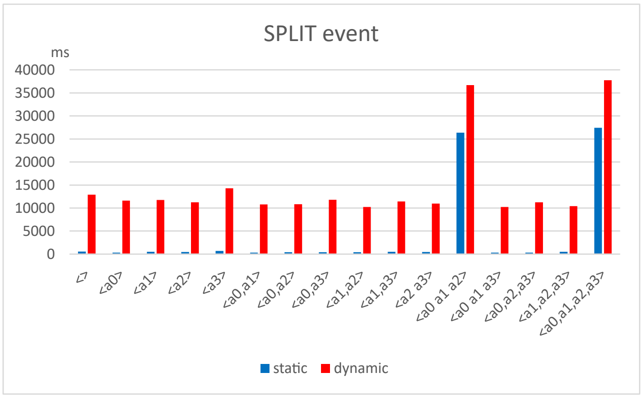
The performance of a creation operation, shown in Fig. 12a, results from a scenario where a shape is extruded several times so as to create a new volume (*i.e.* a new orbit $\langle 0,1,2\rangle$). For every orbit type, the static detection is invariably better than the dynamic one. We note a peak with regard to the dart orbit type, corresponding to a specific case where this orbit requires a substantial treatment as darts are the lowest dimension entities and a fair amount of them are created in this scenario. However, the static detection significantly limits execution time. Orbit $\langle 0,1,2,3\rangle$ yields similar values for static and dynamic detection, caused by the same process.

Fig. 12b shows execution time for the split operation. The study considers a cube where each face is split into four faces by applying the Catmull-Clark subdivision scheme [CC78]. Consequently, this example focuses on the detection of events over the face orbit $\langle 0,1\rangle$. The histogram confirms the efficiency of the static approach over dynamic detection. However, we note the presence of two peaks corresponding to volume orbits $\langle 0,1,2\rangle$ and connected component orbits $\langle 0,1,2,3\rangle$. Moreover, these two peaks have the same values. This case results from the input mesh: as we start with one volume which contains a single connected component, and since the operation just splits faces without adding new volumes, then the number of faces increases and directly impacts the performance for computing static and dynamic evolutions, even if the number of volumes and connected components is still one.

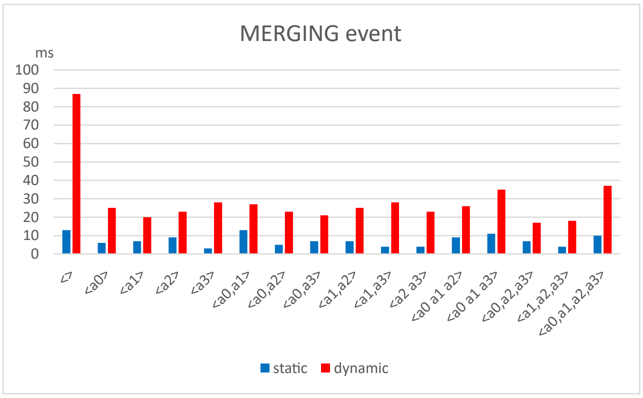
Fig. 12c shows execution time for a merging operation. The study starts from a stack of cubes where the faces shared between two cubes are successively deleted, resulting in the merging of the volumes and their adjacent border faces: we focus on detecting the merging events over volume face orbits $\langle 0,1\rangle$ Once again, the histogram confirms the efficiency of the static approach over dynamic detection.



(a) *Creation operation*



(b) *Split operation*



(c) *Merging operation*

Figure 12: *Comparison between static and dynamic event to track all orbits (the lower the better)*

6. Conclusions and perspectives

Event tracking is one of the core functionalities of geometric modeling kernels. This detection can be done dynamically by comparing each topological cell composing the geometric model before and after applying an operation, but it can represent a significant cost when the model becomes very large. To reduce this cost, it is possible to statically detect some specific events, meaning that events generated in a systematic way are directly defined during the development of the operation. The responsibility is then left to the developer of the operation to define these events, but it entails several hurdles. Firstly, there is no formalization to describe these events. Secondly, it can lead to errors because while some events may seem obvious, others may not appear intuitively or systematically, or may be forgotten. Ultimately, this work of defining events needs to be done again for each newly developed operation.

For these reasons, we propose in this paper to formalize the static detection of events and to automate this process based on automatic analysis of operations. To achieve this, we leverage on the formalism of graph transformation rules to describe geometric operations, and on the topological model of G-maps that enables homogeneous modeling of manifold geometric objects in any dimension. The syntactic analysis of the rule enables the detection of all events that can be detected statically and also specifies the cells on which events that can only be detected dynamically could occur. This dynamic detection becomes much faster, as only these cells need to be verified during the application of the operation. With this approach, any new operation can be developed faster within the modeler, ensuring a complete, accurate and automatic event detection.

This approach leads the way towards future works. First, as of now, the operations that can be analysed are defined with a single rule. An improvement will be to adapt this method to higher level operations defined as scripts combining several rules. Second, most current persistent naming methods rely on the tracking of topological entities evolutions. Working on a complex model, or in case of frequent reevaluations of parametric models, the static tracking of topological entities should allow a significant improvement of the efficiency of persistent naming methods. This could be accurately evaluated by integrating this static tracking into parametric systems.

References

[ABC00] Armstrong, Cecil, Bowyer, Adrian, and Cameron, Stephen. *Djinn: a geometric interface for solid modelling: specification and report*. Information Geometers, 2000 1.

[BALB14] Belhaouari, Hakim, Arnould, Agnès, Le Gall, Pascale, and Bellet, Thomas. "Jerboa: A graph transformation library for topology-based geometric modeling". *International Conference on Graph Transformation*. Springer. 2014, 269–284 2.

[BCS*14] Bézin, Richard, Crespin, Benoît, Skapin, Xavier, et al. "Generalized Maps for Erosion and Sedimentation Simulation". *Computers & Graphics* 45 (Dec. 2014), 1–16 2.

[BMSB07] Baba-Ali, Mehdi, Marcheix, David, Skapin, Xavier, and Bertrand, Yves. "Generic Computation of bulletin boards into Geometric Kernels". *Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*. 2007, 85–93 1.

[BNB05] Bidarra, Rafael, Nyirenda, Paulos J., and Bronsvoort, Willem F. "A Feature-Based Solution to the Persistent Naming Problem". *Computer-Aided Design and Applications* 2.1-4 (2005), 517–526 1.

[CC78] Catmull, Edwin and Clark, James. "Recursively generated B-spline surfaces on arbitrary topological meshes". *Computer-Aided Design* 10.6 (1978), 350–355. ISSN: 0010-4485. DOI: https://doi.org/10.1016/0010-4485(78)90110-0 8.

[CHWZ16] CHENG, YUAN, HE, FAZHI, WU, YIQI, and ZHANG, DEJUN. "Meta-operation conflict resolution for human–human interaction in collaborative feature-based CAD systems". *Cluster Computing* 19 (2016), 237–253 1.

[CMHK12] CHEON, SANG-UK, MUN, DUHWAN, HAN, SOONHUNG, and KIM, BYUNG CHUL. "Name matching method using topology merging and splitting history for exchange of feature-based CAD models". *Journal of mechanical science and technology* 26 (2012), 3201–3212 1.

[Das23] DASSAULT SYSTÈMES SPATIAL CORP. *3D ACIS Modeler product page*. https://www.spatial.com/products/3d-acis-modeling. Accessed on 2023-03-28. 2023 1.

[DL14] DAMIAND, GUILLAUME and LIENHARDT, PASCAL. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. A K Peters/CRC Press, Sept. 2014 2.

[FH18] FARJANA, SHAHJADI HISAN and HAN, SOONHUNG. "Mechanisms of Persistent Identification of Topological Entities in CAD Systems: A Review". *Alexandria Engineering Journal* 57.4 (2018), 2837–2849 1.

[Kri95] KRIPAC, JIRI. "A mechanism for persistently naming topological entities in history-based parametric solid models". *Proceedings of the third ACM symposium on Solid modeling and applications*. 1995, 21–30 1.

[Lie91] LIENHARDT, PASCAL. "Topological models for boundary representation: a comparison with n-dimensional generalized maps". *Computer-aided design* 23.1 (1991), 59–82 2.

[LLX*18] LIU, XIAOJUN, LI, XIANG, XING, JIALU, et al. "Integrating modeling mechanism for three-dimensional casting process model based on MBD". *The International Journal of Advanced Manufacturing Technology* 94 (2018), 3145–3162 1.

[MP02] MARCHEIX, DAVID and PIERRA, GUY. "A Survey of the Persistent Naming Problem". *Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications*. SMA '02. Saarbrücken, Germany: Association for Computing Machinery, 2002, 13–22 1.

[Ope22] OPEN CASCADE SAS. *Open CASCADE Technology collaborative development portal*. https://dev.opencascade.org/. Accessed on 2023-03-28. 2022 1.

[Sie22] SIEMENS. *Parasolid product page*. https://www.plm.automation.siemens.com/global/en/products/plm-components/parasolid.html. Accessed on 2023-03-28. 2022 1.

[TSRA17] TIERNEY, CHRISTOPHER M, SUN, LIANG, ROBINSON, TREVOR T, and ARMSTRONG, CECIL G. "Using virtual topology operations to generate analysis topology". *Computer-Aided Design* 85 (2017), 154–167 4.