Practical Polyhedral Model Checking*

A Gentle Introduction

Yuri Andriaccio[1][0009−0004−3250−2918], Vincenzo Ciancia[2][0000−0003−1314−0574], Diego Latella[3][0000−0002−3257−9059], and Mieke Massink[2][0000−0001−5089−002X]

[1] Scuola Superiore Sant'Anna, Pisa, Italy
yuri.andriaccio@santannapisa.it
[2] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche, Pisa, Italy
{Vincenzo.Ciancia, Mieke.Massink}@cnr.it
[3] Formerly with Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche, Pisa, Italy (ret.)
diego.latella@actiones.eu

Abstract. We illustrate the potential of spatial model checking of polyhedral models on a number of selected examples. In computer graphics polyhedral models can be commonly found in the form of triangular surface meshes or tetrahedral volume meshes. Polyhedral model checking is used to analyse spatial properties of interest of such models expressed in a suitable spatial logic. For this work we use the recently developed geometric spatial model checker `PolyLogicA`, the visualiser `PolyVisualizer` and the polyhedral semantics of the Spatial Logic for Closure Spaces `SLCS`.

Keywords: Spatial logics · Polyhedral models · Polyhedral model checking · Logical equivalence · Spatial bisimulation relations

1 Introduction

Polyhedra are an interesting class of spatial models. They form the mathematical basis for the visualisation of objects in *continuous* space. We find such structures in domains that exploit mesh-processing such as in computer graphics. Fig. 1

* The authors are listed in alphabetical order, as they equally contributed to the work presented in this paper. This is a post-print of the paper "Practical Polyhedral Model Checking — A Gentle Introduction", by Yuri Andreaccio, Vincenzo Ciancia, Diego Latella and Mieke Massink. In: M. H. ter Beek, Stefania Gnesi, Anne E. Haxthausen, Laura Semini (eds), Journeys Between Formal Methods and the Railway Industry, Essays Dedicated to Alessandro Fantechi on the Occasion of His 70th Birthday. Proceedings volume 16470 of Lecture Notes in Computer Science, pages 138-159. Springer, February 5, 2026. ISBN: 978-3-032-12483-8 (Soft-Cover print), 978-3-032-12484-5 (e-Book); DOI: 10.1007/978-3-032-12484-5_8. Springer, 2026, available at: https://doi.org/10.1007/978-3-032-12484-5_8

shows an image of a metro carriage and its triangular surface mesh. Such triangular surface meshes and tetrahedral volume meshes can be very large. Spatial model checking for such structures can be used to highlight or identify interesting aspects of such structures.

In [7] we presented the theory we developed for such continuous spaces and related model checking algorithms. We also presented the `PolyLogicA` model checker we developed for polyhedral model checking and an associated visualiser, `PolyVisualizer`, for the graphical presentation of the model checking results. More specifically, we considered *polyhedral models* and $\texttt{SLCS}_\gamma$, a variant of the Spatial Logic for Closure Spaces (`SLCS`) for such models. A polyhedral model is composed of a *polyhedron* and a *valuation function*. A polyhedron $|K|$ is the set union of all the components of a simplicial complex K, which, in turn, is a collection of *simplexes* in $\mathbb{R}^n$, for some dimension n, satisfying certain construction constraints. For our purposes, we set n to 3, so that, by definition, the kind of simplexes we are concerned with are just points, line segments, triangles and tetrahedra. For such simplexes we consider also their relative interior, i.e. the "open" variants of segments, triangles and tetrahedra — the relative interior of the point is the point itself — that form the so called *cells* of the polyhedral model. Polyhedral model checking consists in the automatic verification of a spatial property on a polyhedral model and gives as a result the set of cells in which the property holds. Polyhedral models used for model checking are such that all points in the same cell of the model satisfy the same predicate letters.

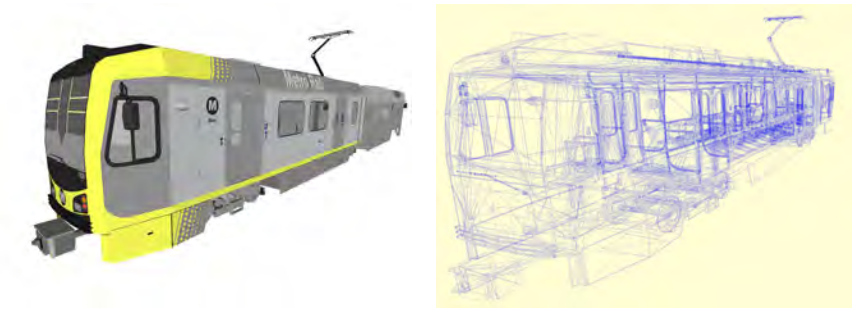


Fig. 1: Image of a metro carriage (left) and its triangular surface mesh (right). Image obtained from cgtrader (www.cgtrader.com) on 2025-07-29. Author: vodolagavaceslav. Light Rail Train Kinki Sharyo P3010.

Original contributions. The original contributions of this paper are twofold. As a first contribution, we illustrate how a polyhedral model can be enriched by adding a new predicate letter to those cells of the original model that satisfy the checked property in a previous model checking session. This is useful as it provides a way to reduce the length of formulas to check on such models

and to obtain more insightful results when these models are used for graphical visualisation. As a second contribution, we show that this form of enrichment also enables practical model minimisation providing deeper insights in the basic spatial structure of the model in terms of the spatial logic properties it enjoys. The work in this paper is performed using `PolyLogicA`, `PolyVisualizer` and the polyhedral semantics of the Spatial Logic for Closure Spaces `SLCS`. We first briefly recall an earlier example from [7] to illustrate polyhedral model checking on a synthetic example so that the reader gets acquainted with this novel form of spatial model checking. We then illustrate the contributions using two different case studies originating from the large set of available mesh models in Wavefront .obj format in the field of computer graphics.

This paper is part of the Festschrift in honour of the 70th birthday of Prof. Alessandro Fantechi. We would like to thank Alessandro for his contributions to the Formal Methods and Tools Laboratory at ISTI-CNR, Pisa, Italy. In particular, the work of Alessandro — together with Stefania Gnesi — on logics and model-checking has been an inspiration for us and for our own recent research on spatial logics and related tools. Some of the authors have started to explore the use of spatial logics and model checking also for applications in the railway domain. See, for example, the work on spatial model checking for smart stations [5] and the work on strategy synthesis for dealing with railway junctions [4]. We hope that the examples on polyhedral model checking, that we describe in the present paper, will spawn new ideas that lead to future applications in the railway domain too. Last, but not least, we want to thank Alessandro for offering us, every autumn, the delicious and authentic specialty "schiacciata con l'uva", that he brings to Pisa directly from a special bakery in Florence! And ..... by train of course!

Related work. Polyhedral model checking is also addressed in [12]. Therein a logic is defined that shares a similar syntax as `SLCS` but is provided with different semantics than those used in the present paper. The focus of their work is on modelling and analysis of groups of interactions and their higher-order relationships. In particular, in their work the domain upon which formulas are interpreted are (sets of) simplexes and not (sets of) points in a polyhedra as in the present paper. In [9, 8, 6] notions of bisimulation for polyhedral models, and for their discrete representation as cell partial order (poset) models, have been proposed. In particular, in [6] an effective model minimisation procedure based on spatial bisimilarity has been presented, as well as its implementation.

From the tools point of view, the python library pymeshlab [14], which is able to programmatically modify 3D meshes based on pre-built operators (mostly traditional 3D imaging filters) is somewhat related to our work. However, a direct comparison of this library and polyhedra model checking would be misleading as the latter uses a declarative language based on `SLCS`, with automatic parallelisation, and automatic memoization (caching) of intermediate results instead of providing a library of pre-built operators.

Synopsis. Section 2 recalls notation and provides a gentle introduction to relevant concepts concerning polyhedral models, spatial logic, model checking and minimisation. Section 3 presents a selection of examples of surface and volume meshes and illustrates various ways in which spatial polyhedral model checking can be used on such models. Section 4 concludes the paper.

2 Background and Notation

Below we briefly, and mostly informally, recall some basic notions, assuming that the reader has some familiarity with topological spaces, Kripke models, and partially ordered sets (posets). All relevant details can be found in [6]. Intuitively, a *simplex* is the convex hull of a set of affinely independent points. In the context of this paper, a simplex can be a point, or a segment, or a triangle, or a tetrahedron. A simplicial complex is a non-empty collection of simplexes that respect some spatial constraints (see [6] for details). The polyhedron $|K|$ of simplicial complex K is the set union of the points belonging to the components of K.[4]

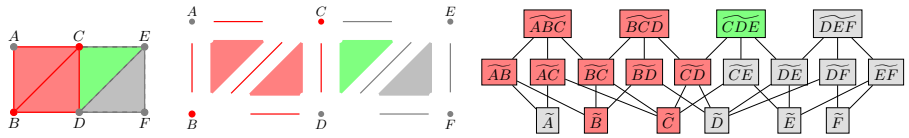
For the purpose of polyhedral model checking, it is convenient to consider, for each simplex σ, its *relative interior*, or *cell*, $\widetilde{\sigma}$. The cell of a point is just the point itself, whereas the cell of any other kind of simplex coincides with the topological interior of the simplex. For instance, the cell of a line segment is the segment itself without its end-points. Note that, given a simplicial complex K, the set $\widetilde{K}$ of its cells is a partition of the polyhedron $|K|$. Furthermore, it is easy to see that the cells of a simplicial complex K form a partial order $(W, \preccurlyeq)$, where $W = \widetilde{K}$ and $\widetilde{\sigma}_1 \preccurlyeq \widetilde{\sigma}_2$ if and only if $\widetilde{\sigma}_1$ is included in the topological closure of $\widetilde{\sigma}_2$. Finally, for each $x \in |K|$, we let $\mathbb{F}(x) \in \widetilde{K}$ denote the unique cell containing x. Given a set $\mathtt{PL}$ of proposition letters, and a simplicial complex K, a *polyhedral model* $\mathcal{P}$ is a pair $(|K|, \mathcal{V}_\mathcal{P})$ where $\mathcal{V}_\mathcal{P}$ maps proposition letters to sets of points in $|K|$.[5] It is required that the image of any predicate letter be a union of cells. On the basis of the valuation function $\mathcal{V}_\mathcal{P}$ of a polyhedral model $(|K|, \mathcal{V}_\mathcal{P})$, we can define the valuation function $\mathcal{V}_\mathcal{F}$ of its associated poset model[6] $\mathcal{F} = \mathbb{F}(\mathcal{P}) = (W, \preccurlyeq, \mathcal{V}_\mathcal{F})$: for each predicate letter p we have that $\widetilde{\sigma} \in \mathcal{V}_\mathcal{F}(p)$ if and only if $\widetilde{\sigma} \subseteq \mathcal{V}_\mathcal{P}(p)$.

Fig. 2 shows a polyhedral model illustrating the various concepts. There are three predicate letters, **red**, **green**, and **gray**, shown by different colours (2a). The model is "unpacked" into its cells in Fig. 2b. The latter are collected in the cell poset model, whose Hasse diagram is shown in Fig. 2c.

[4] Note furthermore that different simplicial complexes can give rise to the same polyhedron.

[5] In the sequel, for the sake of readability, when referring to a polyhedral model, we will often indicate the relevant simplicial complex, say K, only indirectly, via the polyhedron it generates, i.e. $|K|$.

[6] With a little bit of overloading, we let $\mathbb{F}(\mathcal{P})$ denote the poset model associated with polyhedral model $\mathcal{P}$.



(a) (b) (c)

Fig. 2: A polyhedral model $\mathcal{P}$ (2a) with its cells (2b) and the Hasse diagram of the related cell poset (2c).

Besides predicate letters, negation and conjunction, $\texttt{SLCS}_\gamma$ provides a reachability operator, γ. Informally, a point x in a polyhedral model satisfies the conditional reachability formula $\gamma(\Phi_1, \Phi_2)$ if there is a topological path, i.e. a continuous function from the interval $[0, 1]$ to $|K|$, starting from x, ending in a point y that satisfies Φ_2, and such that all the intermediate points of the path between x and y satisfy Φ_1. Note that neither x nor y is required to satisfy Φ_1. More formally, $\texttt{SLCS}_\gamma$ can be defined as follows (see also [7]). For $p \in \texttt{PL}$ the syntax of the logic is the following:

$$\Phi ::= p \mid \neg\Phi \mid \Phi \wedge \Phi \mid \gamma(\Phi, \Phi)$$

The satisfaction relation for $\gamma(\Phi_1, \Phi_2)$, for a polyhedral model $\mathcal{P} = (P, \mathcal{V}_\mathcal{P})$, with $P = |K|$ for some simplicial complex K, and $x \in P$, as defined in [7], is recalled below:

$$\mathcal{P}, x \models \gamma(\Phi_1, \Phi_2) \Leftrightarrow \text{a topological path } \pi : [0, 1] \to P \text{ exists such that } \pi(0) = x, \\ \mathcal{P}, \pi(1) \models \Phi_2, \text{and } \mathcal{P}, \pi(r) \models \Phi_1 \text{ for all } r \in (0,1).$$

It is worth pointing out that the definition of the satisfaction relation does *not* depend on the specific simplicial complex K that generates the polyhedron $|K|$. In other words: given polyhedral models $\mathcal{P}' = (P, \mathcal{V}_{\mathcal{P}'})$ with $P = |K'|$ and $\mathcal{P}'' = (P, \mathcal{V}_{\mathcal{P}''})$ with $P = |K''| = |K'|$ and $\mathcal{V}_{\mathcal{P}'} = \mathcal{V}_{\mathcal{P}''}$ for all $\texttt{SLCS}_\gamma$ formulas Φ and all $x \in P$ the following holds: $\mathcal{P}', x \models \Phi$ iff $\mathcal{P}'', x \models \Phi$.

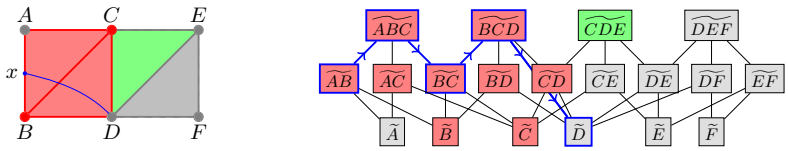
Many interesting properties, such as proximity (in the topological sense, i.e. "being in the topological closure of") or "being surrounded by" can be expressed using reachability. In particular, the near operator $\vec{\Diamond}$, corresponding to the classical closure operator, can be defined as a derived operator in this setting: $\vec{\Diamond}\,\Phi = \gamma(\Phi, \texttt{true})$.

We have also provided an interpretation of $\texttt{SLCS}_\gamma$ on poset models using the notion of $\pm$-path as an image on cell poset models of topological paths in polyhedra. An example of the correspondence between a topological path and its $\pm$-path is shown in Fig. 3. We refer the interested reader to [7] for the relevant formal definitions, here noting only that a $\pm$-path over a poset is a discrete, finite, undirected path $(w_0, w_1, \ldots, w_{n-1}, w_n)$ such that $w_0 \preccurlyeq w_1$ and $w_{n-1} \succcurlyeq w_n$ (see Fig. 3b). The interpretation of γ on a poset model $\mathcal{F} = (W, \preccurlyeq, \mathcal{V}_\mathcal{F})$ is the following: $w \in W$ satisfies $\gamma(\Phi_1, \Phi_2)$ if there is a $\pm$-path starting from w, end-

ing in $w' \in W$ that satisfies Φ_2, and such that all the intermediate elements of the path between w and w' satisfy Φ_1. Note that neither w nor w' is required to satisfy Φ_1. More formally, the satisfaction relation for $\gamma(\Phi_1, \Phi_2)$, for a poset model $\mathcal{F} = (W, \preccurlyeq, \mathcal{V}_\mathcal{F})$ and $w \in W$, is defined as follows:

$$\begin{aligned}\mathcal{F}, w \models \gamma(\Phi_1, \Phi_2) \Leftrightarrow \text{a } \pm\text{-path } \pi : [0; \ell] \to W \text{ exists for some } \ell \geq 2 \text{ s.t. } \pi(0) = w,\\ \mathcal{F}, \pi(\ell) \models \Phi_2, \text{and}\\ \mathcal{F}, \pi(i) \models \Phi_1 \text{ for all } i \in \{n \in \mathbb{N} \mid 0 < n < \ell\}.\end{aligned}$$

In [7] it has been shown that, for all $x \in |K|$ and $\texttt{SLCS}_\gamma$ formulas Φ, we have: $\mathcal{P}, x \models \Phi$ if and only if $\mathbb{F}(\mathcal{P}), \mathbb{F}(x) \models \Phi$. This result is the theoretical foundation for the model checking algorithm for $\texttt{SLCS}_\gamma$ on polyhedral models proposed in [7] that actually works on the poset models. The algorithm has been implemented in the tool `PolyLogicA`.[7]



(a) (b)

Fig. 3: (3a) A topological path from a point x to vertex D in the polyhedral model $\mathcal{P}$ of Figure 2a. (3b) The corresponding $\pm$-path (in blue) in the Hasse diagram of the cell poset model $\mathbb{F}(\mathcal{P})$.

Finally, note that, in the context of cell poset models, $w \models \vec{\Diamond} \Phi$ means that there exists a $\pm$-path starting from w leading in one step going up to a cell that satisfies Φ.

Bisimulation-based model minimisation. In [7], *simplicial bisimilarity*, a novel notion of bisimilarity for polyhedral models, has been defined that uses a subclass of topological paths and it has been shown to enjoy the classical Hennessy-Milner property: two points $x_1, x_2 \in |K|$ are simplicial bisimilar, written $x_1 \sim^{\mathcal{P}}_{\triangle} x_2$, if and only if they satisfy the same $\texttt{SLCS}_\gamma$ formulas, i.e. they are equivalent with respect to the logic $\texttt{SLCS}_\gamma$, written $x_1 \equiv^{\mathcal{P}}_{\gamma} x_2$.

The result has been extended to $\pm$-*bisimilarity* on finite cell poset models, a notion of bisimilarity based on $\pm$-paths: $w_1, w_2 \in W$ are $\pm$-bisimilar, written $w_1 \sim^{\mathcal{F}}_{\pm} w_2$, if and only if they satisfy the same $\texttt{SLCS}_\gamma$ formulas, i.e. $w_1 \equiv^{\mathcal{F}}_{\gamma} w_2$ (see [9] for details). In summary, we have:

$$x_1 \sim^{\mathcal{P}}_{\triangle} x_2 \text{ iff } x_1 \equiv^{\mathcal{P}}_{\gamma} x_2 \text{ iff } \mathbb{F}(x_1) \equiv^{\mathbb{F}(\mathcal{P})}_{\gamma} \mathbb{F}(x_2) \text{ iff } \mathbb{F}(x_1) \sim^{\mathbb{F}(\mathcal{P})}_{\pm} \mathbb{F}(x_2).$$

[7] `PolyLogicA` 0.4 and `PolyVisualizer` are available in the branch polyhedra of the main `VoxLogicA` repository, see https://github.com/vincenzoml/VoxLogicA.

In [8] we showed a similar result for a weaker logic, namely $\texttt{SLCS}_\eta$, where $\gamma(\Phi_1, \Phi_2)$ is replaced by $\eta(\Phi_1, \Phi_2)$, which also requires the satisfaction of Φ_1 on the starting point of the path. Intuitively, $\eta(\Phi_1, \Phi_2)$ is the same as $\Phi_1 \wedge \gamma(\Phi_1, \Phi_2)$. Weaker notions of bisimilarity, $\approx_\triangle$ and $\approx_\pm$, have been introduced and the following has been proven:

$$x_1 \approx_\triangle^{\mathcal{P}} x_2 \text{ iff } x_1 \equiv_\eta^{\mathcal{P}} x_2 \text{ iff } \mathbb{F}(x_1) \equiv_\eta^{\mathbb{F}(\mathcal{P})} \mathbb{F}(x_2) \text{ iff } \mathbb{F}(x_1) \approx_\pm^{\mathbb{F}(\mathcal{P})} \mathbb{F}(x_2).$$

In [6] we presented an effective toolchain for model minimisation modulo this weaker bisimilarity, and thus also modulo $\texttt{SLCS}_\eta$. Let us provide a small example to illustrate spatial bisimulation. With reference to Figure 2a, we have that no red point, call it y, in the open segment CD is simplicial bisimilar to the red point C. And, in fact, although both y and C satisfy $\gamma(\textbf{green}, \texttt{true})$, we have that C satisfies also $\gamma(\textbf{gray}, \texttt{true})$, which is not the case for y. Similarly, with reference to Figure 2c, cell $\widetilde{C}$ satisfies $\gamma(\textbf{gray}, \texttt{true})$, which is not satisfied by $\widetilde{CD}$.

Extension of $\texttt{SLCS}_\gamma$. For the examples in this work it is convenient to add a further operator, converse near ($\overleftarrow{\Diamond}$), that expresses the converse of $\overrightarrow{\Diamond}$. The satisfaction relation for $\overleftarrow{\Diamond}\,\Phi$, for a polyhedral model $\mathcal{P} = (|K|, \mathcal{V}_\mathcal{P})$ and $x \in |K|$, is defined as follows, where for set A, $\mathcal{C}_T(A)$ denotes the topological closure of A:

$\mathcal{P}, x \models \overleftarrow{\Diamond}\,\Phi \Leftrightarrow x' \in \mathcal{C}_T(\mathbb{F}(x))$ exists s.t. $\mathcal{P}, x' \models \Phi$.

The satisfaction relation for $\overleftarrow{\Diamond}$, for a poset model $\mathcal{F} = (W, \preccurlyeq, \mathcal{V}_\mathcal{F})$ and $w \in W$, is defined as:

$\mathcal{F}, w \models \overleftarrow{\Diamond}\,\Phi \Leftrightarrow w'$ exists s.t. $w' \preccurlyeq w$ and $\mathcal{F}, w' \models \Phi$.

We leave a more detailed treatment of spatial bisimulation and minimisation for the extended logic for future work.

Polyhedral model checking and visualisation. As we have seen above, $\texttt{SLCS}_\gamma$ properties of polyhedra can be transformed into equivalent properties on cell poset models. This result is exploited in the polyhedra model checker `PolyLogicA` and in the tool `PolyVisualizer` to visualise the model checking results. `PolyLogicA` is a global model checker that checks a given formula for all cells of the model at once. The algorithm takes as input a finite poset model $\mathbb{F}(\mathcal{P})$ of polyhedron $\mathcal{P}$ and an $\texttt{SLCS}_\gamma$ formula ϕ. The output is the satisfaction set $\mathsf{Sat}(\phi) = \{\widetilde{\sigma} \in \widetilde{K} \,|\, \mathbb{F}(\mathcal{P}), \widetilde{\sigma} \models \phi\}$ of nodes in the poset model $\mathbb{F}(\mathcal{P})$ that correspond to the set of cells of $\mathcal{P}$ that satisfy formula ϕ. The satisfaction set Sat is defined recursively on the structure of $\texttt{SLCS}_\gamma$ formula [7].

The actual input language of `PolyLogicA`, called `ImgQL` (Image Query Language), is an extended set of operations that has $\texttt{SLCS}_\gamma$ as its core language. The `PolyLogicA` model checker is written in `FSharp`. The model checking results of `PolyLogicA` are generated as a `json` file. This file contains the name of the

property as defined in the ImgQL specification, followed by a list of true and false following the order of the definition of the cells in the poset model. Cells for which the corresponding position in the json list is true satisfy the property, cells corresponding to false do not satisfy it. The result file is used by the PolyVisualizer tool to present the model checking results for the polyhedron in a graphical way as follows. Cells that satisfy the property are highlighted in their original colour, those that do not satisfy the property are shown in a semi-transparent way. We will see examples of this use in the next section where model checking results are shown as screenshots of the 3D images produced by the PolyVisualizer tool. Details on the polyhedra model checking algorithm and the visualiser can be found in [7].

3 Potential of Polyhedral Model Checking

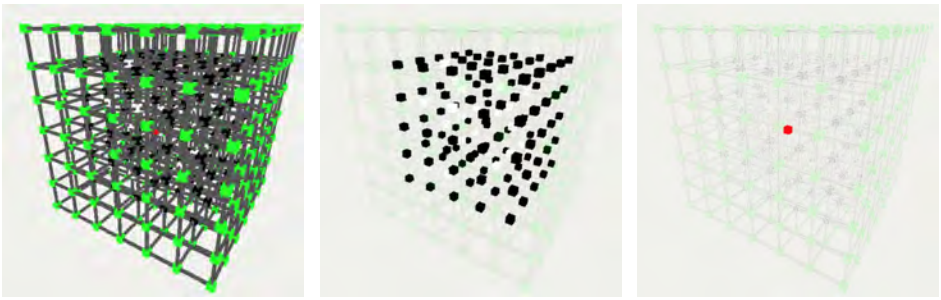
We illustrate the potential of polyhedral model checking for three case studies, each illustrating some different uses. The first is a 3D maze model. The second is a model of a tree-shaped coral. The third is a model of a simple aircraft. The original models of the latter two were provided by a third party in the Wavefront .obj format [15], which is a format widely used in computer graphics. We have developed dedicated python scripts to convert Wavefront .obj images into an input format suitable for polyhedra model checking with PolyLogicA and for viewing with the PolyVisualizer. An example of such a script can be found in [2].

3.1 Analysing Reachability in a 3D Maze Model

As a first example we consider a volumetric tetrahedral mesh of a 3D maze originally presented in [7]. Its purpose here is to illustrate how PolyLogicA can be used to reason about cell colours of an object, where the colours are treated as predicate letters. The maze, shown in Figure 4a, is a 3D cube structure that is 7 'rooms' wide, 7 'rooms' long and 7 'rooms' high and in which such rooms are connected by 'corridors'. Each room and corridor is itself a polyhedron composed, in turn, by a number of cells such as vertices, segments, triangles and tetrahedra.

The rooms at the outermost sides of the cube are green. Inside the cube we find black and white rooms (see Fig. 4b) and one red room (see Fig. 4c). The corridors are all dark grey. The images in Fig. 4 have all been obtained by polyhedral model checking with PolyLogicA. In particular, Fig. 4b shows all cells that satisfy the formula $B \vee W$, where B is the predicate letter for black cells and W the predicate letter for white cells. The cells that satisfy the formula are shown highlighted in their original colour. The cells that do not satisfy the formula are shown in a pale version of their original colour. The cells satisfying the predicate letter R (red cells) are shown in a similar way in Fig. 4c.

Fig. 5 shows $\texttt{SLCS}_\gamma$ formulas that characterise various kinds of corridors, for example, corridors connecting white rooms on both sides are defined using the reachability operator γ, where $\gamma(C, W)$ requires the corridor (C) to connect to



(a) (b) (c)

Fig. 4: 3D maze (4a), black and white (4b) and red rooms (4c) in the 3D maze.

$$
\begin{array}{ll}
\mathsf{corridorWW} & \equiv \gamma(C, W) \wedge \neg\gamma(C, G \vee B \vee R) \\
\mathsf{corridorWG} & \equiv \gamma(C, W) \wedge \gamma(C, G) \\
\mathsf{corridorWR} & \equiv \gamma(C, W) \wedge \gamma(C, R) \\
\mathsf{corridorWB} & \equiv \gamma(C, W) \wedge \gamma(C, B) \\
\mathsf{whiteToGreen} & \equiv \gamma((W \vee \mathsf{corridorWW} \vee \mathsf{corridorWG}), G) \\
\mathsf{Q1} & \equiv \mathsf{whiteToGreen} \vee \gamma(G, \mathsf{whiteToGreen}) \\
\mathsf{Q2} & \equiv \gamma((\mathsf{Q1} \vee \mathsf{corridorWR}), R) \vee \gamma((R \vee \mathsf{corridorWR}), \mathsf{Q1}) \\
\mathsf{Q3} & \equiv (W \vee \mathsf{corridorWW}) \wedge \neg\mathsf{whiteToGreen}.
\end{array}
$$

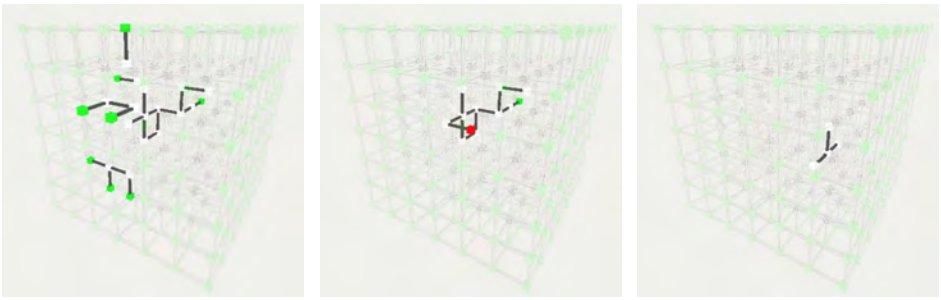
Fig. 5: $\texttt{SLCS}_\gamma$ formulas expressing properties Q1, Q2 and Q3; predicate letters G, W, B, R, C are assumed given and their meaning is the obvious one (C for "corridor", G for green and similarly for the other colours).

a white (W) room and $\neg\gamma(C, G \vee B \vee R)$ makes sure it does not connect to a room of any other colour used in this model.

Formulas Q1, Q2 and Q3 express a few less basic $\texttt{SLCS}_\gamma$ properties. Q1: White rooms and their connecting corridors from which a green room can be reached not passing by black or red rooms, including the green room that is reached; Q2: White rooms and their connecting corridors from which both a red and a green room can be reached not passing by black rooms; Q3: White rooms and their connecting corridors with no path to green rooms. The model checking results of Q1, Q2 and Q3 are shown in Fig. 6, again by highlighting the cells that satisfy the specific formula in their original colour. The full `ImgQL` specification can be found in [2].

3.2 Analysing the Branching Hierarchy of a Coral Model

The second example is used to illustrate a semi-automatic approach to find the various branches in a tree-shape coral (see [1, 3]). In this example, `PolyLogicA` is used in combination with the tool MeshLab [10]. MeshLab is a free and open-source software product for the processing and editing of 3D triangular meshes. It



(a) Q1 (b) Q2 (c) Q3

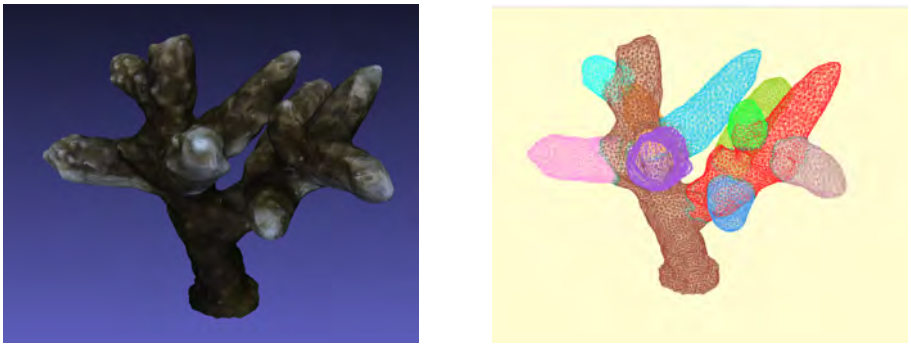
Fig. 6: Spatial model checking results of the properties in Figure 5 for the 3D maze of Figure 4.

provides a wide range of tools for operations such as cleaning, repairing, inspecting, rendering, texturing, and converting meshes. We used MeshLab to perform some initial manual annotations of the model and then to save it in the Wavefront `.obj` format. We then converted the model, with the help of a dedicated python script (see [2]), into an input format that is suitable for further analysis using `PolyLogicA`.

In the following, we show that the results of `PolyLogicA` model checking can be used to enrich the model itself with additional predicate letters. The enriched model is then used, in turn, for further model checking or for model minimisation. This novel way of enriching a model with previously obtained spatial model checking results is very interesting and has many advantages. First of all, it can be used to keep the formulas of interest to check in each model checking session much shorter, resulting in reduced model checking times. This is important as models tend to be very large. Second, enriching/changing models this way makes it possible to study their core structure through $\texttt{SLCS}_\eta$-property preserving minimisation of the model [6]. Third, such minimised models may, in turn, be used to speed up further model checking.

The surface mesh coral model is shown in Fig. 7a as a visualisation in MeshLab in a reduced resolution with respect to the original Wavefront `.obj` file. In Fig. 7b the same model is shown after its conversion into the input format for model checking and visualisation with `PolyVisualizer`. The identification of the branches have been obtained through model checking in the following way. First the borders between the branches have been marked using MeshLab (shown in Fig. 9a). Then some selected vertices of each branch have been marked manually, using a different mark for each branch. Fig. 9b illustrates such a marking for a single branch.

The $\texttt{SLCS}_\gamma$ formulas to identify the various branches are straightforward and shown in Fig. 8. For example, formula `b1` is satisfied by all cells in the model that are not part of a border and through which one can reach a selected cell satisfying



(a) (b)

Fig. 7: (a) Tree-shape coral in MeshLab; (b) Coral with model checking results for finding branches.

$$
\begin{array}{l}
\mathsf{b0} \equiv \gamma(\neg \mathsf{border}, \mathsf{bSel0}) \\
\mathsf{b1} \equiv \gamma(\neg \mathsf{border}, \mathsf{bSel1}) \\
\ldots \\
\mathsf{b13} \equiv \gamma(\neg \mathsf{border}, \mathsf{bSel13})
\end{array}
$$

Fig. 8: Predicate letters border, bSel0, bSel1, etc. of selected triangular faces for the various branches are assumed given; b0 through b13 characterise the cells belonging to the various branches.

bSel1, passing only by such non-border elements (except possibly for the first element of such a path). Providing the visualiser with the model checking results, together with a definition of the colour in which to show the cells satisfying the various properties b0 to b13 and the borders, we obtain the polyhedron shown in Fig. 7b.

Coral complexity. An interesting aspect of tree-shape corals is to find the branching hierarchy in the coral structure. Such branching hierarchy provides information about the complexity of the corals. It is known from the literature that the covering of the ocean floor with corals, and particularly its covering with branching corals, are positively correlated with the structural complexity of coral reefs. In turn, structural complexity of coral reefs has been shown to positively influence several measures of biodiversity in such reefs. Branching coral cover of the ocean bottom may be particularly likely to contribute fine-scale structural complexity to reefs, which can be important to a range of organisms, such as fish and mobile invertebrates (see e.g. [11] page 322). One way to measure the complexity of a coral is through investigating the parent-child hierarchy of branches. In the following we will use `PolyLogicA` to colour the various groups of branches according to their rank in such a hierarchy. The rank of a branch is its position in an in-order traversal of the tree-shaped coral starting from the

(a) (b)

Fig. 9: (a) Borders between branches; (b) Example of marked vertex (see arrow) to identify a branch.

root. The root branch of the coral has rank 1, branches attached to the root have rank 2, and so on.

The procedure follows two alternating phases starting from the root of the coral. We define two properties for each rank level n. The first property, indicated by rankPn, defines all the cells that belong to a certain rank level. The root is the first rank level. The second property, indicated by rankAn, defines all the cells that belong to rank level n or lower, including the borders between such branches. We define two derived operators grow(x,y) and throughCom(x,y,z). Informally, the operator grow(x,y)= x $\vee$ (y $\wedge\gamma$(y,x)) extends an area of cells each satisfying x with an adjacent area of cells each satisfying y. The operator throughCom(x, y, z) combines the γ operator with the grow operator. Informally, this formula is satisfied by any cell from which a $\pm$-path starts that passes by cells satisfying x and reaches a cell satisfying grow(z,y). When instantiated as throughCom($\neg$border, border, rank), this operator is used to find cells in the branches, that are not border cells, but via which one can reach border cells passing through cells in branches with a lower rank than that specified in the formula. We also define an auxiliary operator difference(x,y). The latter property is satisfied by cells that satisfy x but not y. The specification in Fig. 10 shows the formulas for identifying three rank levels. Of course, these can be extended in the obvious way to identify the rank of branches in coral with a larger branching structure. The specification assumes that the predicate letters border and rootSel, some selected cells of the root branch, are given. Both were obtained by annotating the model with the tool MeshLab.

The model checking results of the specification in Fig. 10 are shown in Fig. 11a. Rank level rankP1 is shown in purple, rank level rankP2 in green and rank level rankP3 in orange. Borders between branches are shown in brown. The full ImgQL specifications for the coral example can be found in [2].

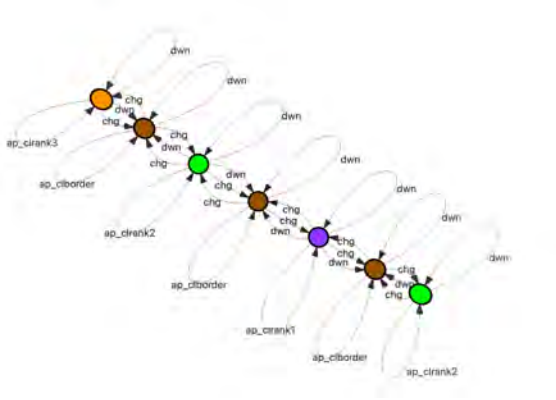
grow(x,y)	$\equiv \mathsf{x} \vee (\mathsf{y} \wedge \gamma(\mathsf{y}, \mathsf{x}))$
throughCom(x,y,z)	$\equiv \gamma(\mathsf{x}, \mathsf{grow}(\mathsf{z}, \mathsf{y}))$
difference(x,y)	$\equiv \mathsf{x} \wedge \neg\mathsf{y}$
root	$\equiv \gamma(\neg\mathsf{border}, \mathsf{rootSel})$
rankP1	$\equiv \mathsf{root}$
rankA1	$\equiv \mathsf{root}$
rankP2	$\equiv \vec{\Diamond}\,(\mathsf{difference}(\mathsf{throughCom}(\neg\mathsf{border}, \mathsf{border}, \mathsf{rankP1}), \mathsf{rankA1}))$
rankA2	$\equiv \mathsf{rankP2} \vee \mathsf{grow}(\mathsf{rankA1}, \mathsf{border})$
rankP3	$\equiv \vec{\Diamond}\,(\mathsf{difference}(\mathsf{throughCom}(\neg\mathsf{border}, \mathsf{border}, \mathsf{rankP2}), \mathsf{rankA2}))$
rankA3	$\equiv \mathsf{rankP3} \vee \mathsf{grow}(\mathsf{rankA2}, \mathsf{border})$

Fig. 10: Finding the rank of coral branches. Predicate letters border and rootSel are assumed given.

Core structure of corals. One might be interested also in the core structure of a branching coral. There are several known computer graphics methods that can be used to obtain such a structure, for instance methods that extract a topological 'skeleton' from a triangular surface mesh, i.e. a sort of thin version of the shape that is equidistant to its borders. In our setting we could obtain an abstract version of the structure by minimising the structure w.r.t. $\approx_{\pm}$, the weaker bisimulation related to $\mathtt{SLCS}_{\eta}$ (see Section 2). The result of minimising the structure in Fig. 11a is shown as a Labelled Transition System (LTS) in Fig. 11b, where the states in the LTS have been given colours that correspond to the rank levels in Fig. 11a. The labels of the self-loops in the LTS, such as ap_clrank1 (denoting the first rank in the hierarchy) and ap_clborder, denote the predicate letters of the states, the other transition labels, chg and dwn, are auxiliary labels used in the minimisation procedure (see [6] for details). This LTS does not give exactly the same information as a topological skeleton, but may reveal other interesting aspects. For example, it shows that there exists at least one branch, attached to the root, from which further branches emanate, leading to a maximum ranking level of 3. This is represented by the longer chain of states in the LTS, starting from the purple state (clrank1) and passing by a brown state (clborder), a green state (clrank2), a brown state again and finally an orange state (clrank2). It also shows that there are branches that emanate from the (purple) root that do not further split into other branches in turn. This is represented by the shorter chain of states in the LTS starting from the purple state.

Let us illustrate how we obtain the minimal LTS model of a coral. First of all, we use `PolyLogicA` to find the rank levels as described previously. The model checking result for each rank level can be saved in a simple .json format that provides for each cell in the model the value 'true', in case it is part of that rank level, and the value 'false' if it is not. We can use this outcome to add new predicate letters to the original model. For example, in case we have the results for rank level 1, we add the predicate letter `"clrank1"` to each cell that satisfies the rank1 property in the model file (but not `"border"`), replacing the predicate

(a)



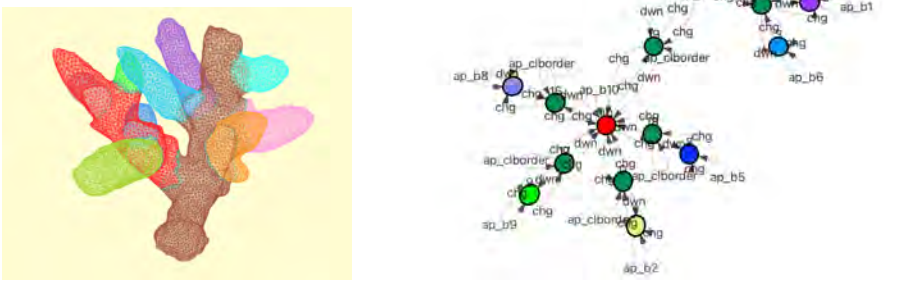
(b)

Fig. 11: (a) Rank levels in the same coral (turned 180 degrees around its vertical axes w.r.t. Fig. 7); (b) Minimised model as LTS.

letters that were already there. This is easy to do in an automatic way using a simple python script because in the result file of the model checking session the results are given in exactly the same order as the cells defined in the model file. If we do this for every rank, we get a model with only clrank1, clrank2, clrank3 and clborder as predicate letters, and each cell is labelled by exactly one predicate letter. Applying the toolchain described in [6] for minimising the model modulo $\approx_{\pm}$, the weaker bisimulation related to $\texttt{SLCS}_{\eta}$, we obtain the minimised LTS as shown in Fig. 11b (apart from colouring the states, which was done manually to facilitate the interpretation of the state labels).

We can perform a similar analysis on the coral model in which all branches were given a different colour as in Fig. 7b. The result of this minimisation is the LTS shown in Fig. 12b. In the latter, the colours of the states representing equivalence classes w.r.t. $\texttt{SLCS}_{\eta}$ reflect the colours of the branches of the coral in Fig. 12a. The states representing the border classes are in dark green. This makes it easy to recognise the full structure of the coral reflected in the minimised LTS model. A closer look also reveals an interesting aspect. In the upper right part of the LTS the blue branch and the purple branch are shown to share a border. This was not the case in the minimal model in Fig. 11b, since there these two branches had the same colour because they belonged to the same rank. Furthermore, the orange branch is now a single branch. The ranking procedure had split that branch into two different parts, assigning different ranks.

Note that the above analyses are presenting an interesting and innovative use of model minimisation. Usually model minimisation is applied to increase the efficiency of spatial model checking for large models [8]. Actually, the possibility to add (or replace) predicate letters in a model as we did above can be of great use also in other settings, as it makes it possible to insert intermediate model



(a) (b)

Fig. 12: (a) Branches of the coral with individual colours; (b) Minimised model as LTS with state colours reflecting that of corresponding branches in (a), which states representing borders in dark green.

checking results in the model itself so that further formulas to check can be of much shorter length, saving precious computing resources.

3.3 Studying Curvature Properties in a Simple Aircraft Model

In this third example we show how `PolyLogicA` can be used to study curvature properties of an object, in this case a simple aircraft model[8] chosen for illustrative purposes. The aircraft is composed of simple surfaces that are connected together. Such connections form angles that can be concave or convex and that can be more or less sharp. The surfaces themselves are mainly flat, but there are some small ridges as well. A view of the aircraft is shown in Fig. 13a and a version in which concave and convex edges are shown in different colours is presented in Fig. 13b.

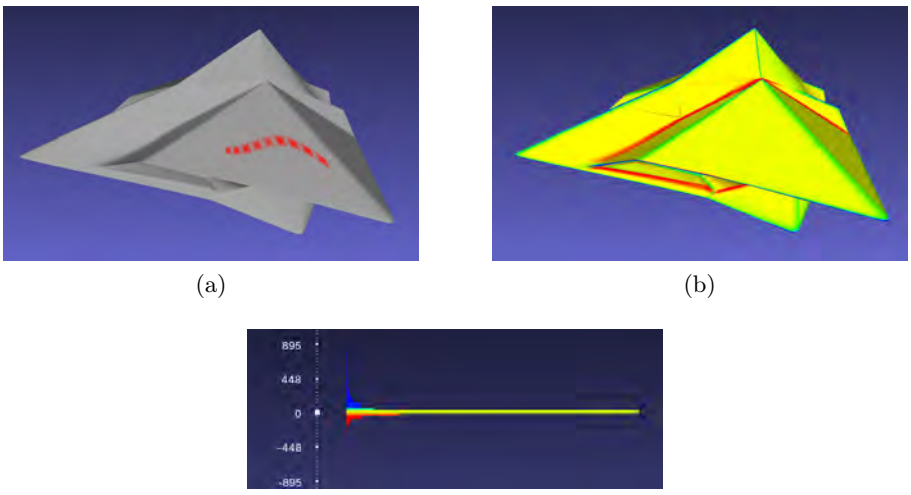
The image in Fig. 13b has been obtained by computing discrete mean curvature values [13] for each vertex of the triangular mesh with MeshLab [10]. The resulting image was saved as a Wavefront `.obj` image in which the curvature values are saved as colours for each *vertex*. Blue-coloured vertexes are representing points on a locally highly convex surface (i.e. a high positive mean curvature value). Green vertexes represent points on the surface with a less pronounced

[8] Original model downloaded from Free3D at https://free3d.com/3d-model/ufo-triangle-v1--354611.html. Accessed on 21 May 2024.

convexity (i.e. with a less high but still positive mean curvature value). Red vertexes represent points on a concave surface (i.e. points with a negative mean curvature value). Yellow vertexes represent points where the surface is essentially flat (or, in a few cases, a saddle point).

As described in the previous case study, this .obj image can be converted using a python script into an input format suitable for polyhedra model checking with `PolyLogicA` and for viewing with `PolyVisualizer`. In the python script we can define the relation between the colours of the vertexes and the names of the predicate letters. For example, we defined four predicate letters, with the names 'flat', 'concave', 'convex' and 'convexsharp', corresponding to the four colours used in the histogram in Fig. 13c that was produced with MeshLab by rendering the curvature measure as a vertex quality.

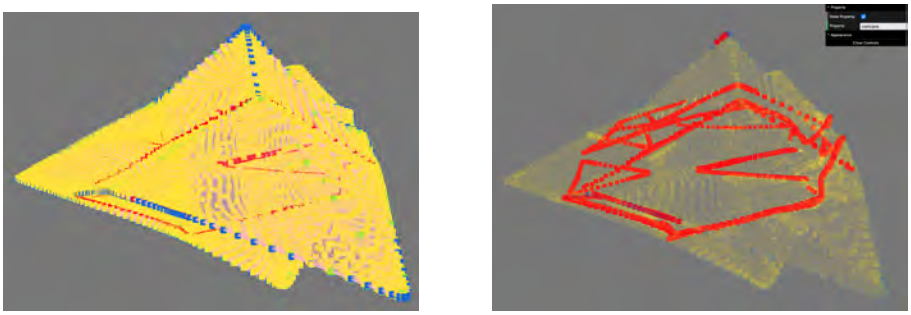
A vertex (or point) satisfies the predicate letter 'concave' if it is red (in rgb terms [255, 0, 0]) within an error margin of 30. Similarly for convex (rgb [0, 255, 0]) and convexsharp (rgb [0, 0, 255]). We gave all remaining vertices the predicate letter 'flat' and all remaining triangles and segments of the aircraft the predicate letter 'ufo' (the original name of this aircraft model). The result of this conversion is shown in the `PolyVisualizer` visualisation of the .json model in Fig. 14a, where we chose to show vertexes satisfying flat in yellow, vertices satisfying concave in red, and so on. We can now use `PolyLogicA` on the generated .json model to check the predicate letters one by one and generate the associated result files for visualisation with `PolyVisualizer`. For example, in



(a) (b)

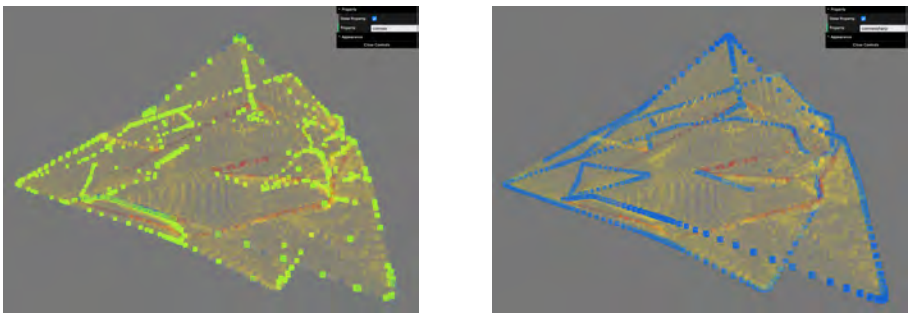
(c)

Fig. 13: (a) Aircraft in MeshLab [10]; (b) Discrete Mean Curvature values of the aircraft model as computed by MeshLab [10]: Flat surface in yellow, concave edges in red, convex edges in green and sharp convex edges in blue, as also shown in the histogram of curvature values for vertices (c).



(a) (b)

Fig. 14: (a) Aircraft in `PolyVisualizer`; (b) Vertices satisfying 'concave' highlighted in red.



(a) (b)

Fig. 15: (a) Vertices satisfying 'convex' highlighted in green; (b) Vertices satisfying 'convexsharp' highlighted in blue.

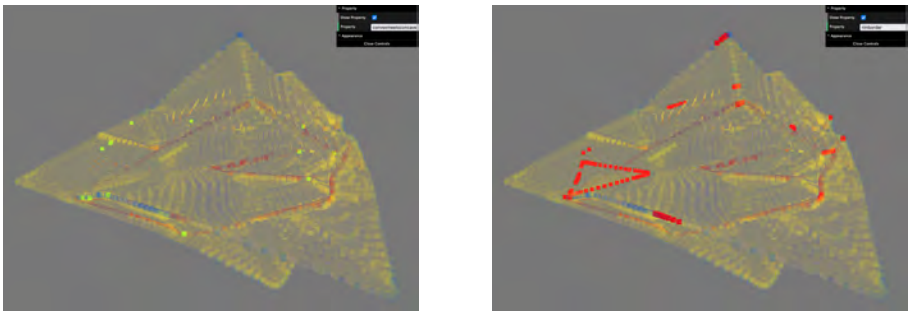
Fig. 14b all vertices satisfying concave are highlighted in red. In Fig. 15 vertices satisfying convex and convexsharp are shown, highlighted in green (Fig. 15a) and highlighted in blue (Fig. 15b), respectively.

More interesting properties are specified in Fig. 16. The formula $\overleftarrow{\Diamond}\mathsf{concave}$ is satisfied by all segements and triangles that are connected to a vertex satisfying 'concave'. We can then define $\mathsf{convexmeetsconcave} = \mathsf{convex} \wedge \gamma(\overleftarrow{\Diamond}\mathsf{concave},\mathsf{convex})$. The model checking results are shown in Fig. 17a where the highlighted green vertices satisfy the formula.

Finding sharp ridges. A rather subtle property is to find areas that are surrounded by a sharp ridge. We see such an area for example as a triangle shape in the bottom of the aircraft as shown in Fig. 18a. Such ridges are characterised by a thin concave area next to a sharp convex area. One can define such a ridge as $\mathsf{ridgeborder} = \mathsf{concave} \wedge \gamma(\overleftarrow{\Diamond}\ \mathsf{convexsharp}, \mathsf{concave})$ and then look for an area surrounded by such a ridge defining $\mathsf{flatridgearea} = \overleftarrow{\Diamond}\ \mathsf{flat} \wedge\ (\neg\ \mathsf{grow}(\mathsf{ridgeborder})$,

grow(x,y)	$\equiv x \vee (y \wedge \gamma(y, x))$
convexmeetsconcave	$\equiv$ convex $\wedge\, \gamma(\overleftarrow{\Diamond}$ concave, convex)
ridgeborder	$\equiv$ concave $\wedge\, \gamma(\overleftarrow{\Diamond}$ convexsharp, concave)
flatridgearea	$\equiv \overleftarrow{\Diamond}$ flat $\wedge$ ($\neg$grow(ridgeborder), $\overleftarrow{\Diamond}$ flat)).

Fig. 16: Derived $\texttt{SLCS}_\gamma$ operator grow(x,y); $\texttt{SLCS}_\gamma$ formulas expressing the properties convexmeetsconcave, ridgeborder and flatridgearea; predicate letters concave, convex, convexsharp and flat are assumed given.



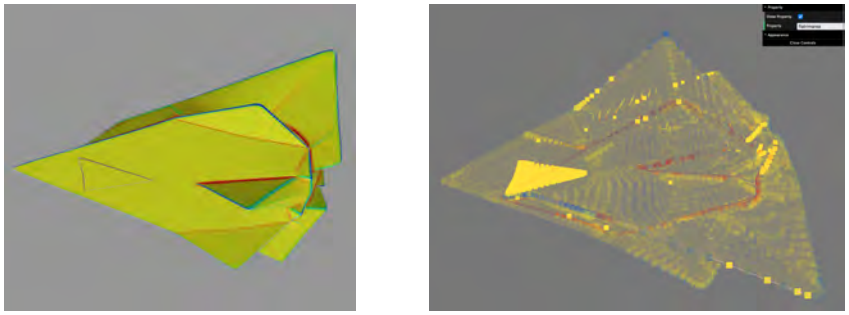
(a) (b)

Fig. 17: (a) convex meeting concave highlighted in green; (b) Vertices satisfying ridgeborder highlighted in red.

$\overleftarrow{\Diamond}$ flat)), where $\overleftarrow{\Diamond}$ flat are the vertices satisfying flat extended with their directly adjacent segments and triangles. The result of `PolyLogicA` model checking of this formula is shown in Fig. 18b, where the area that satisfies the property is highlighted in yellow. It is easy to see that besides the triangular shape in the bottom of the aircraft there are also other small areas that have this property in the rear of the aircraft and along the edges of the wings. This illustrates how `PolyLogicA` could be used to find aspects of the curvature of an object that are difficult to detect, which might be very useful, for example, when looking for small defects in constructions. The full `ImgQL` specifications for the aircraft model can be found in [2].

3.4 Preliminary Performance Data and Discussion

We have presented three different case studies to illustrate a variety of ways in which polyhedral model checking can be applied. The first case study concerned a synthetic example, i.e. a 3D polyhedral model that was automatically generated in the format required for the `PolyLogicA` model checker. This case study served the purpose of illustrating the potential of the conditional spatial reachability operator γ. It also provided a first test case for studying the practical feasibility of polyhedral model checking.



(a) (b)

Fig. 18: (a) Bottom view of aircraft; (b) Vertices satisfying '`flatridgearea`' highlighted in yellow.

The second case study started from an existing triangular surface mesh of a tree-shaped coral. In this case, the surface mesh still needed to be converted into the format for the `PolyLogicA` model checker. The converter was developed by one of the authors in such a way that it could be easily adapted to be used to convert other `.obj` surface meshes as well. Furthermore, MeshLab turned out to be a very useful tool set to annotate the original mesh to enable and facilitate further polyhedral model checking. MeshLab was also used to experiment with rescaling of the coral model and subsequent polyhedral model minimisation to study and visualise the core structure of the coral.

The third case study started from an existing triangular surface mesh obtained from a publicly accessible large repository of such meshes. Also in this case we have successfully converted the obtained model from its original `.obj` format into the model checking format. Furthermore, in this case MeshLab has been used to apply one of its algorithms to visualise curvature aspects of the model, which have subsequently been investigated using polyhedral model checking. Even if in this paper we illustrated the approach on a simple case study, it should be evident that polyhedral model checking provides interesting potential to analyse such models in more detail than would be possible (or convenient) by pure human visual inspection.

We close this discussion by providing a few preliminary indications on the time performance of `PolyLogicA` model checking on the models introduced in this paper. This work shows only a small part of the potential uses, but it serves as a first step to illustrate ideas and to, hopefully, stimulate others to use `PolyLogicA` in a fruitful way in further case studies.

Preliminary model checking performance data. Although it is beyond the scope of this paper to study the performance aspects of polyhedral model checking with `PolyLogicA` in detail, we provide some preliminary indications in

Table 1 for each of the three examples[9]. Since this is the first polyhedral model checker in its kind, to the best of our knowledge, we have no data to compare our performance results with that of other tools.

model	**nr. of vertexes**	**nr. of cells**	**MC time incl. IO**	**pure MC time**
maze	11,319	147,245	5.97	0.38
coral (branches)	8,123	48,728	2.7	0.20
coral (hierarchy)	67,573	405,426	22.4	5.94
aircraft	31,810	190,850	10.01	3.35

Table 1: For each model the number of vertexes, total number of cells, model checking time including IO (i.e. reading input model and producing result file) and pure model checking time are shown. All times are in seconds. The properties for the maze are the ones shown in Fig. 5, for the coral (branches) those in Fig. 8, for the coral (hierarchy) those in Fig. 10 and for the aircraft those in Fig. 16. The full `ImgQL` specifications can be found in [2].

The results in Table 1 show that pure model checking times range from 0.38 seconds for the maze to about 6 seconds for the analysis of the ranks in the coral model. Considering the number of cells that the models consist of, ranging from several tens of thousands to several hundreds of thousands, these performance times are very encouraging and show the practical feasibility of polyhedral model checking. We note that a significant amount of time is spent performing I/O operations, i.e. the reading and writing of the model and result files, respectively. This is due to the fact that in the current prototype we are using plain text files for models and results. In future work we plan to integrate the various components of the tool chain so that writing and reading intermediate results can be avoided, reducing the time needed for I/O.

The general computational complexity of the polyhedral model checking algorithm (which includes both the translation of the polyhedron into a poset and the model checking algorithm itself), given a fixed maximal dimension of the simplicial complexes in the polyhedral model, is $\mathcal{O}(n \cdot h)$, where n is the number of nodes of the poset and h is the cardinality of the set of different $\texttt{SLCS}_\gamma$ subformulas present in the `ImgQL` specification that need to be checked. Further details on the model checking algorithm and its complexity can be found in [7].

4 Conclusion

We have shown various ways in which polyhedral model checking can be used to analyse aspects of triangular surface meshes and tetrahedral volume meshes that

[9] All experiments in this paper were performed with `PolyLogicA` 0.4 on a Mac M2 pro with 12 cores and 32GB of memory.

are abundant in the domain of computer graphics. To this end, we have used our first-in-its-kind spatial polyhedral model checker `PolyLogicA` in combination with the well-established computer graphics tool MeshLab [10]. The latter provides a huge library of operations on surface and volume meshes, among which manual annotation of objects and operations such as computing the curvature characteristics of objects and many more. The results of such operations can be saved as a mesh object and then converted into the input format of the model checker inserting useful predicate letters in the model. This way models can be formally and automatically analysed from a wide range of perspectives. This paper has only illustrated a small subset of the possibilities.

In future work we plan to address further models from various domains, illustrating also how `PolyLogicA` could be used to check properties from more than one perspective (e.g. curvature, distance, texture) of a model at once. Recent work on geometric analysis through spatial logics to check the consistency of 3D geological models [16] also seems a promising domain of application, besides a continuation of the application of polyhedral model checking in the medical domain [7]. We also plan to investigate its application in the railway domain.

Acknowledgments. Research partially supported by Bilateral project between CNR (Italy) and SRNSFG (Georgia) "Model Checking for Polyhedral Logic" (#CNR-22-010); European Union - Next GenerationEU - National Recovery and Resilience Plan (NRRP), Investment 1.5 Ecosystems of Innovation, Project "Tuscany Health Ecosystem" (THE), CUP: B83C22003930001; European Union - Next-GenerationEU - National Recovery and Resilience Plan (NRRP) – MISSION 4 COMPONENT 2, INVESTMENT N. 1.1, CALL PRIN 2022 D.D. 104 02-02-2022 – (Stendhal) CUP N. B53D23012850006; MUR project PRIN 2020TL3X8X "T-LADIES"; CNR project "Formal Methods in Software Engineering 2.0", CUP B53C24000720005. We thank the anonymous reviewers for their careful review and useful observations.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Andriaccio, Y.: Skeletonization, segmentation and phenotyping of branching corals 3D replicas (2022), B.Sc. Thesis, AA 2021/2022, Available from the author
2. Andriaccio, Y., Ciancia, V., Latella, D., Massink, M.: Practical exploration of polyhedral model checking. CoRR **abs/2506.20176** (2025). https://doi.org/10.48550/ARXIV.2506.20176, https://doi.org/10.48550/arXiv.2506.20176
3. Andriaccio et al., Y.: Leveraging 3D geometric processing for the phenotyping of branching corals (2024), https://www.reeffutures.com/, oral presentation at Reef Futures
4. Basile, D., ter Beek, M.H., Bussi, L., Ciancia, V.: A toolchain for strategy synthesis with spatial properties. Int. J. Softw. Tools Technol. Transf. **25**(5), 641–658 (2023). https://doi.org/10.1007/S10009-023-00730-1, https://doi.org/10.1007/s10009-023-00730-1

5. ter Beek, M.H., Ciancia, V., Latella, D., Massink, M., Spagnolo, G.O.: Spatial model checking for smart stations - research challenges. In: Lluch-Lafuente, A., Mavridou, A. (eds.) Formal Methods for Industrial Critical Systems - 26th International Conference, FMICS 2021, Paris, France, August 24-26, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12863, pp. 39–47. Springer (2021). https://doi.org/10.1007/978-3-030-85248-1_3, https://doi.org/10.1007/978-3-030-85248-1_3
6. Bezhanishvili, N., Bussi, L., Ciancia, V., Gabelaia, D., Jibladze, M., Latella, D., Massink, M., de Vink, E.P.: Weak simplicial bisimilarity and minimisation for polyhedral model checking (2024), https://arxiv.org/abs/2411.11428
7. Bezhanishvili, N., Ciancia, V., Gabelaia, D., Grilletti, G., Latella, D., Massink, M.: Geometric Model Checking of Continuous Space. Log. Methods Comput. Sci. **18**(4), 7:1–7:38 (2022), https://lmcs.episciences.org/10348, DOI 10.46298/LMCS-18(4:7)2022. Published on line: Nov 22, 2022. ISSN: 1860-5974
8. Bezhanishvili, N., Ciancia, V., Gabelaia, D., Jibladze, M., Latella, D., Massink, M., de Vink, E.P.: Weak simplicial bisimilarity for polyhedral models and SLCSη. In: Castiglioni, V., Francalanza, A. (eds.) Formal Techniques for Distributed Objects, Components, and Systems - 44th IFIP WG 6.1 International Conference, FORTE 2024, Held as Part of the 19th International Federated Conference on Distributed Computing Techniques, DisCoTec 2024, Groningen, The Netherlands, June 17-21, 2024, Proceedings. Lecture Notes in Computer Science, vol. 14678, pp. 20–38. Springer (2024). https://doi.org/10.1007/978-3-031-62645-6_2, https://doi.org/10.1007/978-3-031-62645-6_2
9. Ciancia, V., Gabelaia, D., Latella, D., Massink, M., de Vink, E.P.: On bisimilarity for polyhedral models and SLCS. In: Huisman, M., Ravara, A. (eds.) Formal Techniques for Distributed Objects, Components, and Systems - 43rd IFIP WG 6.1 International Conference, FORTE 2023, Held as Part of the 18th International Federated Conference on Distributed Computing Techniques, DisCoTec 2023, Lisbon, Portugal, June 19-23, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13910, pp. 132–151. Springer (2023). https://doi.org/10.1007/978-3-031-35355-0_9, https://doi.org/10.1007/978-3-031-35355-0_9
10. Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G.: MeshLab: an Open-Source Mesh Processing Tool. In: Scarano, V., Chiara, R.D., Erra, U. (eds.) Eurographics Italian Chapter Conference. The Eurographics Association (2008). https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136
11. Graham, N.A.J., Nash, K.L.: The importance of structural complexity in coral reef ecosystems. Coral Reefs **32**(2), 315–326 (2013). https://doi.org/10.1007/s00338-012-0984-y, https://doi.org/10.1007/s00338-012-0984-y
12. Loreti, M., Quadrini, M.: A spatial logic for simplicial models. Log. Methods Comput. Sci. **19**(3) (2023). https://doi.org/10.46298/LMCS-19(3:8)2023, https://doi.org/10.46298/lmcs-19(3:8)2023
13. Meyer, M., Desbrun, M., Schröder, P., Barr, A.H.: Discrete differential-geometry operators for triangulated 2-manifolds. In: Hege, H.C., Polthier, K. (eds.) Visualization and Mathematics III, pp. 35–57. Springer-Verlag, New York (2003). https://doi.org/10.1007/978-3-662-05105-4_2
14. Muntoni, A., Cignoni, P.: PyMeshLab. Zenodo (Jan 2021). https://doi.org/10.5281/zenodo.4438750

15. Murray, J.D., vanRyper, W.: Encyclopedia of graphics file formats, 2nd Edition. Springer (1996), http://oreilly.com/catalog/9781565921610
16. Parquer, M.N., de Kemp, E.A., Brodaric, B., Hillier, M.J.: Checking the consistency of 3d geological models. Geoscientific Model Development **18**(1), 71–100 (2025). https://doi.org/10.5194/gmd-18-71-2025, https://gmd.copernicus.org/articles/18/71/2025/