Rule-based generative design of translational and rotational interlocking assemblies

Pierre Gilibert, Romain Mesnil, Olivier Baverel

▶ **To cite this version:**

Pierre Gilibert, Romain Mesnil, Olivier Baverel. Rule-based generative design of translational and rotational interlocking assemblies. Automation in Construction, 2022, 135, pp.104142. 10.1016/j.autcon.2022.104142 . hal-04582391

HAL Id: hal-04582391
https://hal.science/hal-04582391

Submitted on 22 Jul 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

Version of Record: https://www.sciencedirect.com/science/article/pii/S0926580522000152
Manuscript_5a2d99fba90fca9f5895494cc0abf9c1

Rule-based generative design of translational and rotational interlocking assemblies

Pierre Gilibert[a], Romain Mesnil[a], Olivier Baverel[a,1]

[a]*Laboratoire Navier UMR 8205, Ecole des Ponts Paristech, Marne-la-Vallée, 77455, MLV Cedex 2, France*
[b]*GSA ENS Architecture Grenoble, France*

Abstract

Despite being a notoriously difficult task, efficient design of interlocking assemblies could greatly impact the construction sector and reduce its environmental footprint by helping in the design of demountable buildings and the reuse of structural members. While a growing research effort in this direction is being undertaken by the computer graphics and structural engineering communities, most of the algorithms proposed so far imply restrictions on assembly directions or prior knowledge on the joint's geometry.While relevant, such tools do not fully explore the space of possible assemblies and often fail to produce surprising results. Moreover, these designs are always assembled through translational motions, and, to the best of our knowledge, very little research has been conducted to address the challenges of designing an assembly for rotation motions.

Building on recent advances in assembly design, this study investigates the automatic generation, using a Markov process and turtle graphics, of 2D interlocking sequential assemblies that can be assembled for any prescribed combination of translations and rotations. This generative approach shall be an aid to the engineer to explore the space of geometrical form-fitting connections and represents a first step towards an end-to-end workflow to design interlocking assemblies.

© 2022 published by Elsevier. This manuscript is made available under the CC BY NC user license
https://creativecommons.org/licenses/by-nc/4.0/

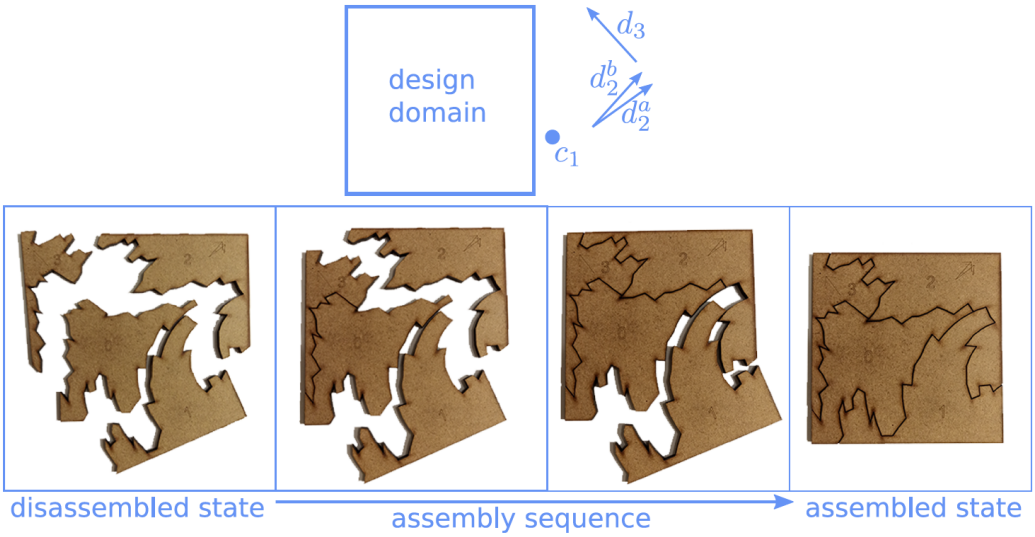


Figure 1: Top row is our tool's input: the disassembling motions are given by the location of a centre of rotation, a cone of translation, and a single direction of translation. Bottom row: the fabricated raw output and an assembly sequence.

Keywords: Design for assembly, Kinematics-aware fabrication, Shape partitioning, Interlocking assembly, Markov process, Turtle graphics

1. Introduction

Even the simplest buildings are constituted of tens of thousands of components which need to be assembled *in-situ* or in factory [1]. In civil engineering, an assembly may be defined as a collection of parts that are connected to each other through various kinds of means: nails, glue, etc. Despite their mechanical relevance, a major inconvenient with such standard connectors is that they are irreversible: at the end of the lifetime of a building, assemblies are often destroyed and prevent the reuse of structural components which has a significant impact on the environment: according to the French government [2] the construction sector generated almost 70% of the waste in France in 2017. Conversely, new construction materials have a serious environmental impact, up to 7% of the emission of greenhouse gas in France in 2016 [3]. This paper represents a humble step towards the necessary decarbonisation and waste reduction of the construction industry by studying the generation of reversible interlocking assemblies, i.e. assemblies that are tightly

held together through geometrical features. Interlocking assemblies can be (dis)assembled at will and could potentially be used to design reusable structural members in buildings in close spirit to traditional Japanese architecture, see FIGURE 2.

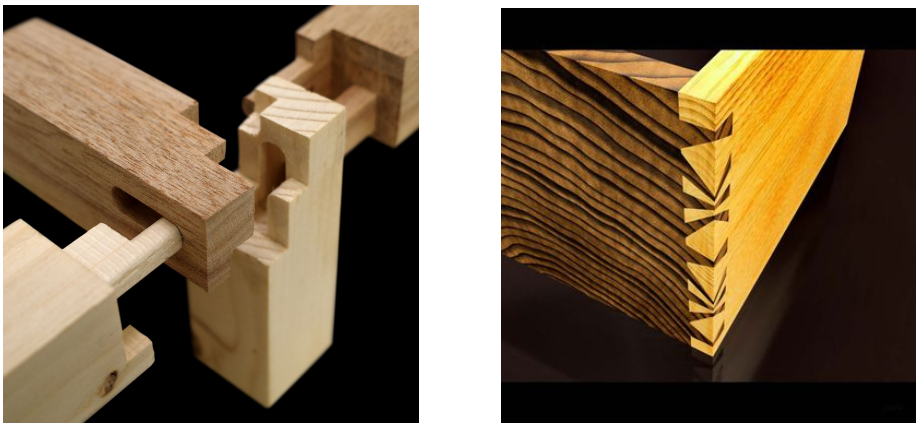


Figure 2: Examples of 3D sequential assemblies. Left: a CNC-milled assembly (source [4]). Right: a traditional Nejiri Arigata assembly (source Internet).

Interlocking assemblies are defined by Song *et al.* in [5] as an assembly of rigid parts such that only one of them, the *key*, is movable while any other part or subset of parts are immobilised relative to one another. The literature on the subject is rich with, for instance, [6] who designed furniture joinery and study the stability of the structure, [4] who introduces a remarkable software to design wood joints with a special focus on fabricability, or [7] who build a framework aimed at generating novel assemblies and presents examples of voxelised puzzles. These approaches can be categorised in two families: they are *catalogue-based*, meaning that possible joints are predefined in some catalogue (or similarly that the user is supposed to already have some knowledge on the geometry of the joint) or *voxel-based* which limits the space of accessible shapes for a given voxel resolution. In these approaches, the final assembly can only be assembled along directions of translation defined in a pre-existing discrete set: a single sliding axis in [4], a set of 26 arbitrary directions in 3D (8 in 2D) for [6] or the three canonical directions of space as a result of the voxelisation in [7]. As such, existing methods greatly reduce design and motion freedoms and leave completely unaddressed the challenges of designing assemblies that can be built using rotational motions.

Contribution . The aim of this paper is to introduce a Markov process and combine it with an agent to automatically generate 2D interlocking sequential assemblies working in translation and in rotation. The focus of our work is on the assembly node, the joint: parts shall be carved out of a design domain resulting from the intersection of structural members, see FIGURE 3. The main features of our approach follow:

- Generality: our approach deals with translations, rotations, and a combination of both. All translation directions or centres of rotation can be selected.
- Exhaustivity: any interlocked geometry can be generated.
- Injectivity: any feasible geometry can be produced by one sequence only.

The interested reader is referred to our video [8] which exemplifies the ideas and concepts developed in this paper.

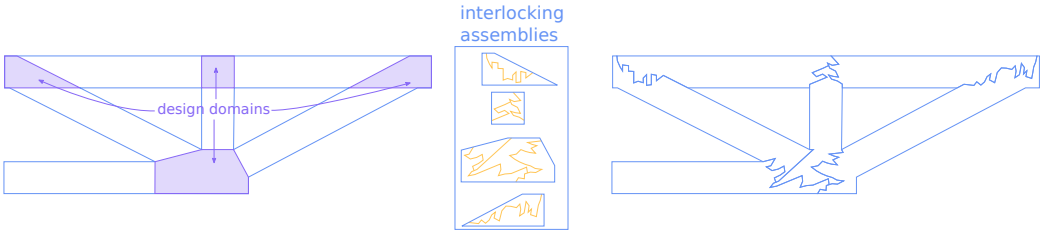


Figure 3: The aim of this study is to automatically generate a 2D interlocking assembly in a possibly non-convex design domain resulting from the intersection of several structural members. Here the algorithm presented in this paper was ran independently on each of the four design domains.

After discussing related works in SECTION 2 we sum up elementary results about blocking relationships in interlocking assemblies based on graph analysis and introduce the concepts of Markov process and turtle graphics in SECTION 3. Section 4 explains how graphs, turtle graphics and a Markov chain are combined to define an agent that generates interlocking assemblies and presents results. Finally SECTION 6 lists the shortcomings of our approach.

2. Related work

2.1. Designing interlocking assemblies

The literature on the design of interlocking assemblies can broadly be divided into two categories with a porous border.

Catalogue-based designs:
. Significant work has been made to automatically generate assemblies made of parts chosen in a catalogue: [9] builds hollow 3D shape using a set of Lego bricks; [10] explores the partitioning of 3D shapes into a set of 6-parts burr puzzles; Fu and coauthors [11] developed a method aimed at creating global interlocking furniture assembly from a model of orthogonally intersecting 3D shapes and generated the joints from a lookup table. In a different spirit, [12] partitions a 3D shape into printable parts and assembles them through mortise and tenon kind of joints and [13] presents a material-aware algorithm to modify the overall shape of a structure but connects parts with mortise and tenon joints.

Voxel-based designs:
. Most of the other methods available to generate interlocking assemblies are voxel-based which restrict the assembling motions to the three canonical orthogonal directions of $\mathbb{R}^3$: [5] focuses on recursive interlocking puzzles where at each step of the assembly sliding sequences of 3 parts are tightly interlocked. Thus only one assembling sequence is possible. This work was later refined in [14] who carefully subdivides an input mesh into a set of voxelised parts such that every $K \geq 3$ parts are tightly interlocked.

Yao *et al.* [6] implemented a tool that asks the user for the exterior appearance of the joints between structural components and automatically computes the internal solid geometry needed to connect and assemble the parts. Remarkably they were able to rediscover many traditional Japanese joints with intricate geometry. The main drawback of their approach is that the assembling motions are restricted to a set of 26 directions of translation in 3D (8 in 2D).

A compelling study has recently been made by the authors of [4]: their approach is the exact complementary to ours in the sense that their work about the design of wood joints is both voxel-based and catalogue-based (but supports adaptation to non-orthogonal and non square joints by linearly deforming the grid of voxels) and primarily focuses on interaction with the

human user, fabrication and mechanical relevance.

2.2. Disassembly planning

Assembly planning (or its pendant disassembly planning) refers to the problem of finding a sequence to fully (dis)assemble the parts constituting an assemblage, [15]. Several methods were developed and are thoroughly reviewed by [16], [17]. An interesting approach, on which we put an emphasis, was first proposed in [18]: given an assembly made of various parts, the authors introduced the concept of *Non Directional Blocking Graph* (NDBG) to encode blocking relations between the parts in directed graphs. By analysing these graphs the authors are able to find, for each step of the (dis)assembly process, which set of parts to move and what motion to follow to perform the task. While the method works theoretically both for translation and rotation, in practice they implemented an algorithm that "*considers all pure translations plus some suggested generalised motions*" without adding more details. Their method, for translation only, was later improved in [19] who makes further use of local contact information between parts.

Wang and coauthors in [7] were the first to leverage the kind of graph analysis introduced in [18] to automatically generate voxel-based interlocking assemblies. They designed an efficient puzzle generator that can be assembled along orthogonal translations. Even though they restrict themselves to voxelised structures, the authors fully explore the accessible design space and successfully manage to generate globally interlocking pieces without much computational effort. Their work serves as the basis of ours.

Generating interlocking assemblies is a difficult geometric challenge ([5]) and the methods reviewed in the literature attempt to simplify the problem by making strong assumptions on the shape of the assembly and the (dis)assembling motions which negatively impact the freedom needed to design novel assemblies. It can be argued that these assumptions ultimately stem from the fact that designing interlocking assemblies is essentially a *wicked problem*. Indeed, each problem is unique, can be approached by many different methods, infinitely many designs are solutions to it, and because of competing goals (ease of fabrication, ease of assembly, mechanical relevance, etc.) no solution is the best, one can only say that some designs are better than others. More formally, as for any structural object, the quality of an

interlocking assembly strongly depends on the interaction of the five axes of design proposed by [20], namely form, force, structure, material, and technology. For instance, Larsson *et al.* in [4], while delivering stunning results, had to assume an assembly (structure) made of wood (material), carrying most probably bending moments (force), milled with a 3-axis CNC machine (technology), with a grid of voxel as a design space (form), as well as additional assumptions such as a cube as a design domain and a single axis of assembly. Any change in those premises, for instance switching to a 5-axis CNC machine, greatly impacts the space of solutions and requires another algorithm to search it. More generally a good approach to designing assemblies, shown in FIGURE 5, would be through a multi-criteria optimisation where several designs are proposed to the designer who makes the final choice as to orient her work. These criteria (choice of material, technology, etc) are problem-dependent and could therefore be implemented *a posteriori* to curate the space of solutions, once the user knows how to navigate and explore the field of possible assemblies. As an example, for timber assemblies, the milling technology (3 to 5 axes, size of milling tools) and mechanical performance (governed by the strong anisotropy of wood) are obvious practical constraints that will dictate the performance of the assembly and, thus, the subset of suitable assembly shapes. Other technologies and material, like 3d-printed steel nodes [21], would come with different sets of feasibility constraints which would be met by different geometries of assembly.

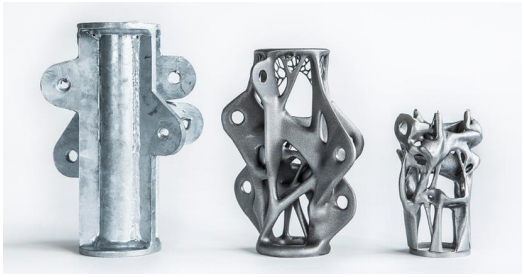


Figure 4: Digital technology makes it possible to envisage completely different methods of application, beyond traditional carpentry (source: [21]).

The focus of this paper is precisely on the exploration of the space of 2D interlocking sequential assemblies made of polygonal parts. To that end, we introduce a Markov process that maps surjectively to the space of polygonal lines, to ensure that the whole search space of such assemblies is reachable

through our method, as will be proved in SECTION 4.1.2. We insist that, in this paper, we focus only on the geometrical aspect of an assembly, and, even though they are fundamental questions, we leave to future work the consideration of structural properties and manufacturability that a real-life assembly with application in engineering or architecture must possess. In other words, in this preliminary work, we address the challenges of generating a puzzle rather than a load-bearing, structurally sound, assembly.

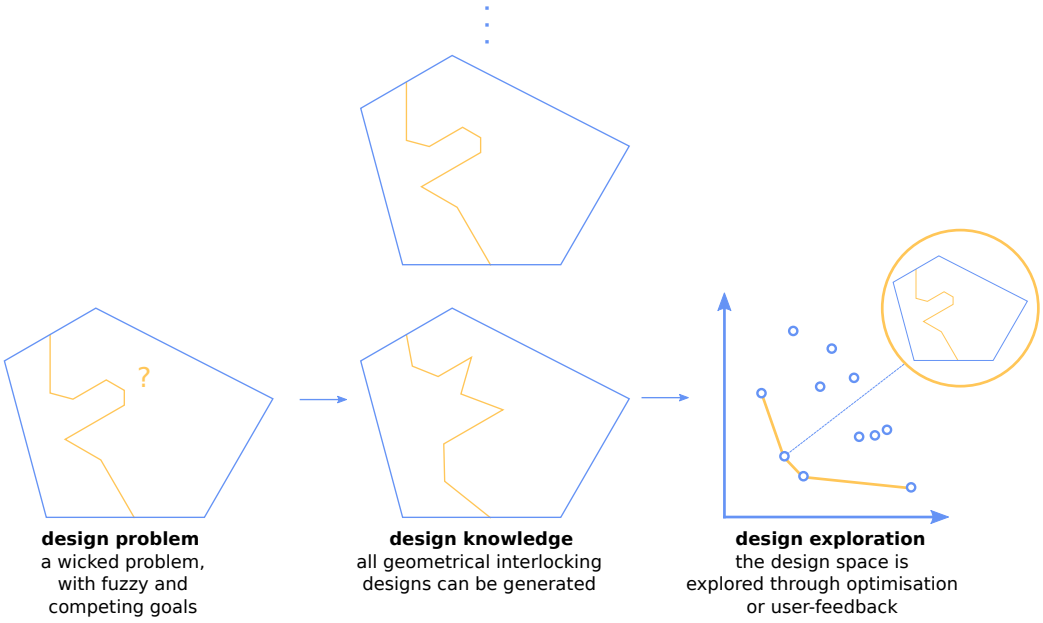


Figure 5: The focus of this paper is on the *design knowledge* step of the process to design interlocking assemblies: we shall generate all possible geometrical interlocking design.

3. Methodology

Using the terminology presented in [17], we restrict our study to 2D interlocking sequential assemblies made of polygonal parts. Since our work is in the spirit of [7] we will use the same notation as they did. Given a polygonal design domain (which throughout this paper will simply be a square) and an ordered list of N motions for disassembling, our goal is to partition the domain into $N+1$ parts forming a sequential assembly $A = \{P_0, P_1, ..., P_N\}$. For instance, on FIGURE 1, the design domain is partitioned into 4 parts ($N = 3$) $P_0, ...P_3$. While the reference part P_0 is fixed each P_i, $i \geq 1$, can

only move along its prescribed motion in an infinitesimal sense, meaning that a motion of arbitrarily small magnitude does not lead a part to collide with an other. Moreover, P_1 is the key - as long as P_1 has not been removed from the assembly, no other part can move - and as such the assembly is interlocked. Also, the disassembling sequence "*remove* P_1*, then* P_2*, ..., then* P_N" always exists, possibly among others. We will say that for $i > 0$ part P_i *obeys* motion $\boldsymbol{d}$ (be it a rotation or a translation) if and only if it can freely translate/rotate along $\boldsymbol{d}$ for an infinitesimal motion without colliding with another part P_j, $j \neq i$.

3.1. Blocking relationships in assemblies

For this exposition to be self-contained we recall some elementary facts about blocking relationships in assemblies. Additional details and proofs can be found in [18]. For the sake of simplicity and illustrative purposes, until SECTION 4.2, we only consider assemblies obeying translation motions.

3.1.1. Cone of translational freedom

We first focus our attention on the design problem where we want to partition a design domain into a polygonal assembly $A = \{P_0, P_1\}$ such that P_1 obeys an infinitesimal translation along one or many vector(s) $\boldsymbol{d} \in \mathbb{R}^2$. To that end, we are going to reverse-engineer the relationship between a given assembly $A = \{P_0, P_1\}$ and the set of translational motions P_1 can obey.
The easiest assembly made of two polygonal parts we can think of is such that the dividing curve between P_0 and P_1 is simply a line segment.

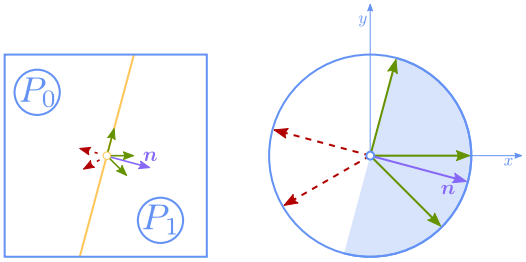


Figure 6: On the left an assembly $A = \{P_0, P_1\}$. On the right, the hemisphere in a darker shade of blue represents the half-space of motion of P_1.

On FIGURE 6 the design domain is the blue square on the left, the dividing curve is the yellow line segment and the green arrows represent several valid directions of translation such that P_1 can move along them without colliding

with P_0. The red dashed ones depict invalid directions of translation as moving P_1 along them will lead the two parts to intersect. The set of all valid translations constitutes a so-called *half-space of motion*. On FIGURE 6, let $\boldsymbol{n}$ be the unit normal vector of the yellow line segment separating the two parts oriented from P_0 to P_1 (the purple arrow). Then we can state that P_1 may obey a direction $\boldsymbol{d} \in \mathbb{R}^2$ if and only if $\boldsymbol{n} \cdot \boldsymbol{d} \geq 0$. A somewhat more complex result can be obtained by analysing a dividing polyline made of two line segments:

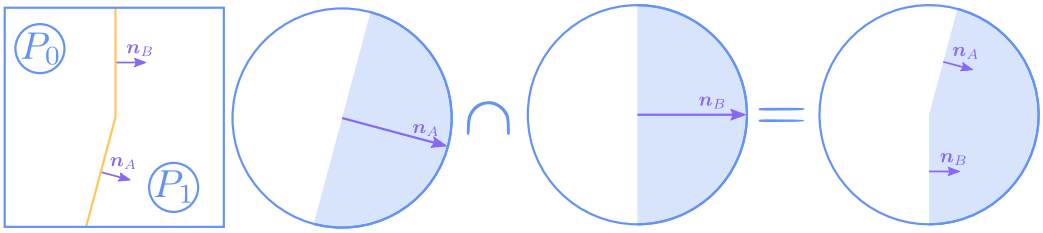


Figure 7: On the left an assembly $A = \{P_0, P_1\}$. On the right, the cone of translational freedom results from the intersection of the half-space of motion of each yellow line segment.

On FIGURE 7 each line segment defines a half-space of motion. P_1 can obey any $\boldsymbol{d}$ that is in both half-spaces of motion i.e. any $\boldsymbol{d}$ such that $\boldsymbol{n}_A \cdot \boldsymbol{d} \geq 0$ and $\boldsymbol{n}_B \cdot \boldsymbol{d} \geq 0$. We call *cone of translational freedom* the cone resulting from the intersection of the half-spaces of motion defined by the normal of each line segment of the yellow polyline.

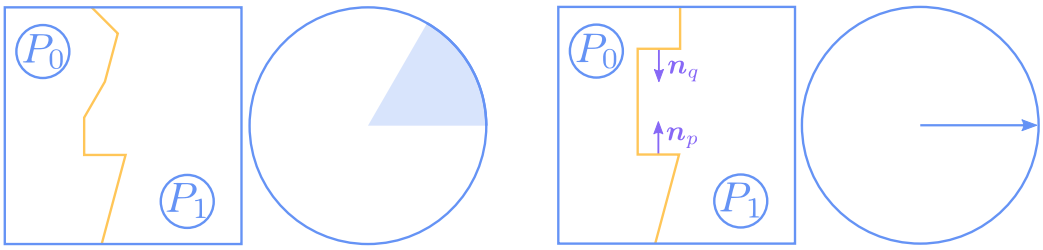


Figure 8: By intersecting half-spaces of motion, we can define the cone of translational freedom of P_1. Note that this cone may be reduced to a single direction as highlighted in the example on the right.

More generally for any polyline made of k line segments, any vector $\boldsymbol{d}$ in

the cone of translational freedom is a solution to the system

$$\begin{cases} \boldsymbol{n}_1 \cdot \boldsymbol{d} \geq 0 \\ \vdots \\ \boldsymbol{n}_k \cdot \boldsymbol{d} \geq 0 \end{cases} \tag{1}$$

FIGURE 8 illustrates two more complex examples where on the right the cone of translational freedom is reduced to a single direction. This event is characterised by the fact that at least two unit normals $\boldsymbol{n}_p$ and $\boldsymbol{n}_q$ are such that $\boldsymbol{n}_q \cdot \boldsymbol{d} = 0$ and $\boldsymbol{n}_p = -\boldsymbol{n}_q$. In both case any $\boldsymbol{d}$ in the cone can be obeyed to by P_1.

This small study gives us one important insight regarding the generation of assemblies obeying translations. Assume the user wants to design a part P_1 that must obey a cone of translational freedom bounded by two prescribed vectors $\boldsymbol{d}_A$ and $\boldsymbol{d}_B$ (possibly with $\boldsymbol{d}_A = \boldsymbol{d}_B$). Given this cone we want to generate a polyline such that the normal vectors of the line segments are solutions to SYSTEM (1). In other words, we want the Gauss map of the polyline to lie in the arc of circle resulting from the intersection of the two semi-circles oriented by $\boldsymbol{d}_A$ and $\boldsymbol{d}_B$. FIGURE 9 illustrates this process, where the Gauss map is figured by the collection of normal vectors depicted with purple arrows.

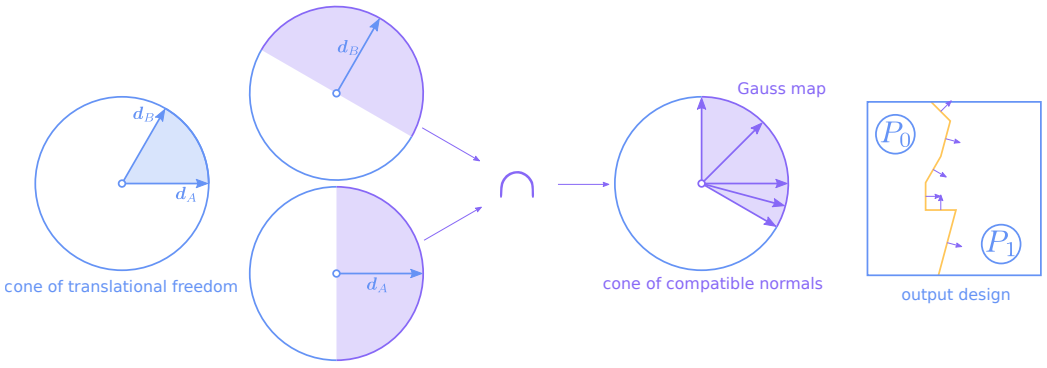


Figure 9: The desired cone of translational freedom is bounded by two vectors which define two semi-circles whose intersection becomes the cone of compatible normals. The Gauss map of the dividing curve, represented here using purple arrows, must lie in the latter.

We have now understood which criterion should be met for a part with a prescribed cone of translational freedom. Yet this section tells us nothing

on how to make sure that an assembly made of several parts is interlocked, which is the focus of the next two sections.

3.1.2. Directional Blocking Graph - DBG

Wilson and coauthors introduced in [18] the concept of *Directional Blocking Graph* (DBG) for a given assembly A and a motion $\boldsymbol{d}$ that conveniently sums up the blocking relationships between the parts P_i of A for an infinitesimal motion along $\boldsymbol{d}$. Such DBG is a directed graph denoted $G(\boldsymbol{d}, A)$ whose vertices P_i correspond to each of the $N+1$ parts in the assembly A and an arc $e_{i\to j}$ from vertex P_i to vertex P_j means that part P_i *is blocked by* part P_j for an infinitesimal motion of part P_i along direction $\boldsymbol{d}$ while holding part P_j in place. For instance, on FIGURE 10, edge $e_{0\to 1}$ in $G(\boldsymbol{d}_h, A)$ encodes that P_0 is blocked by P_1 for an infinitesimal translation along the horizontal direction $\boldsymbol{d}_h$ and (since there is no $e_{1\to 0}$) that P_1 is free to translate along $\boldsymbol{d}_h$: P_1 obeys $\boldsymbol{d}_h$. On the other hand, since both edges $e_{0\to 1}$ and $e_{1\to 0}$ exist in $G(\boldsymbol{d}, A)$ then both parts P_0 and P_1 block each other for a translation along $\boldsymbol{d}$: none can obey $\boldsymbol{d}$.

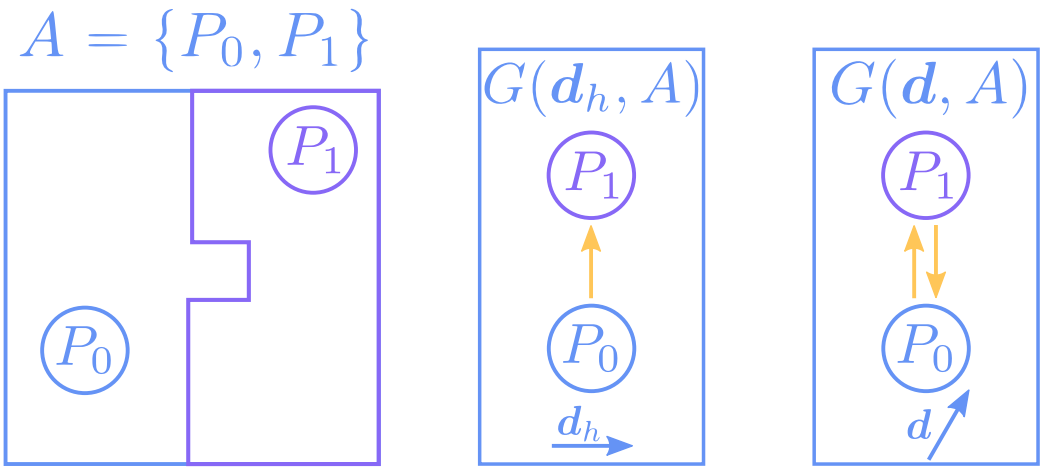


Figure 10: An assembly A made of two parts P_0 and P_1 and two DBGs.

A useful property of a DBG $G(\boldsymbol{d}, A)$ is that we can easily deduce whether the assembly A is interlocked for $\boldsymbol{d}$ by checking the strong-connectedness[1] of

[1]Informally speaking, a directed graph is **strongly connected** if one can walk along a path respecting the orientation of the edges between any couple of vertices (P_i, P_j).

the graph: if the graph is strongly connected then every part is blocked by another. If not, then it can be decomposed into strongly connected components (possibly reduced to a single vertex) where at least one component can obey $\boldsymbol{d}$.

For instance, in FIGURE 11, top row, the DBG associated with the horizontal direction of motion has two strongly connected components that are colour-coded: $\{P_1\}$ in purple and $\{P_0, P_2\}$ in blue. Indeed, starting from P_1 one cannot reach any other node but P_1 while following the edges' orientation and, similarly, if one starts from any node in $\{P_0, P_2\}$, one can only reach these two nodes. Moreover, no edge starts from P_1: it is not blocked by any other node and hence it obeys $\boldsymbol{d}$. Still on the top row, the DBG to the right associated with the upwards direction of motion is strongly connected: its strongly connected component is $\{P_0, P_1, P_2\}$ as one can convince oneself by walking from any node to any other. Thus since every part is blocked by another and the assembly is deadlocked for an upwards motion of translation.

On the bottom row of FIGURE 11 part P_1 has been removed and the assembly is now made of two parts $\{P_0, P_2\}$. The DBG associated to the horizontal direction of motion is strongly connected and the one associated to the upwards translation has two strongly connected components, namely $\{P_0\}$ (blue) and $\{P_2\}$ (purple). Moreover, since no edge starts from P_2 the latter is not blocked by P_0 for that upwards direction $\boldsymbol{d}$: it can obey such $\boldsymbol{d}$.

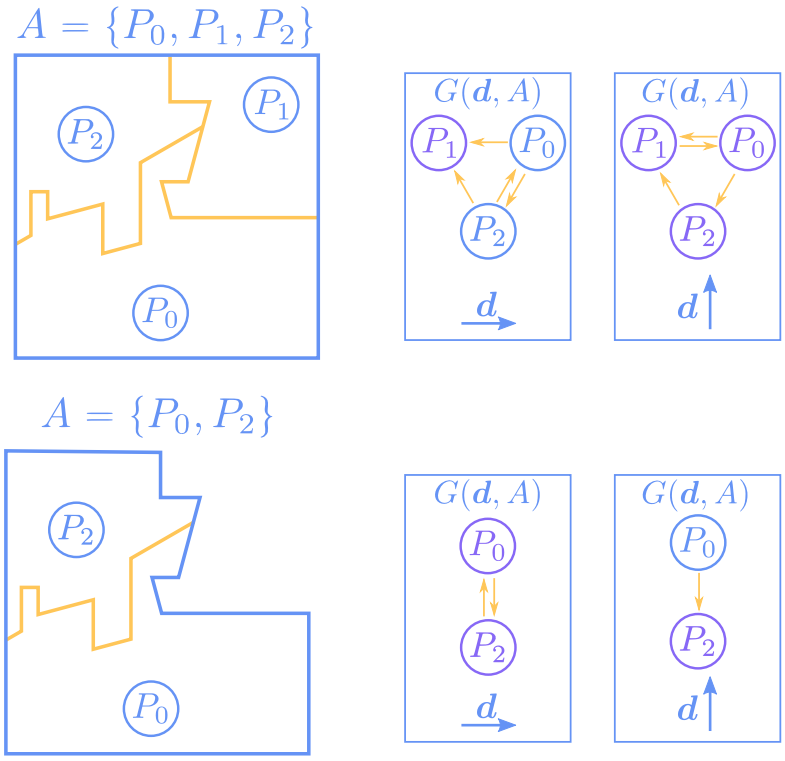


Figure 11: The strongly connected components of each DBG are colour-coded (blue and purple). Top row: an assembly made of 3 parts and the DBGs associated to the horizontal to the right and vertical upwards directions of translation. Bottom row: assembly obtained after removing P_1 and the corresponding DBGs.

3.1.3. Non Directional Blocking Graph - NDBG

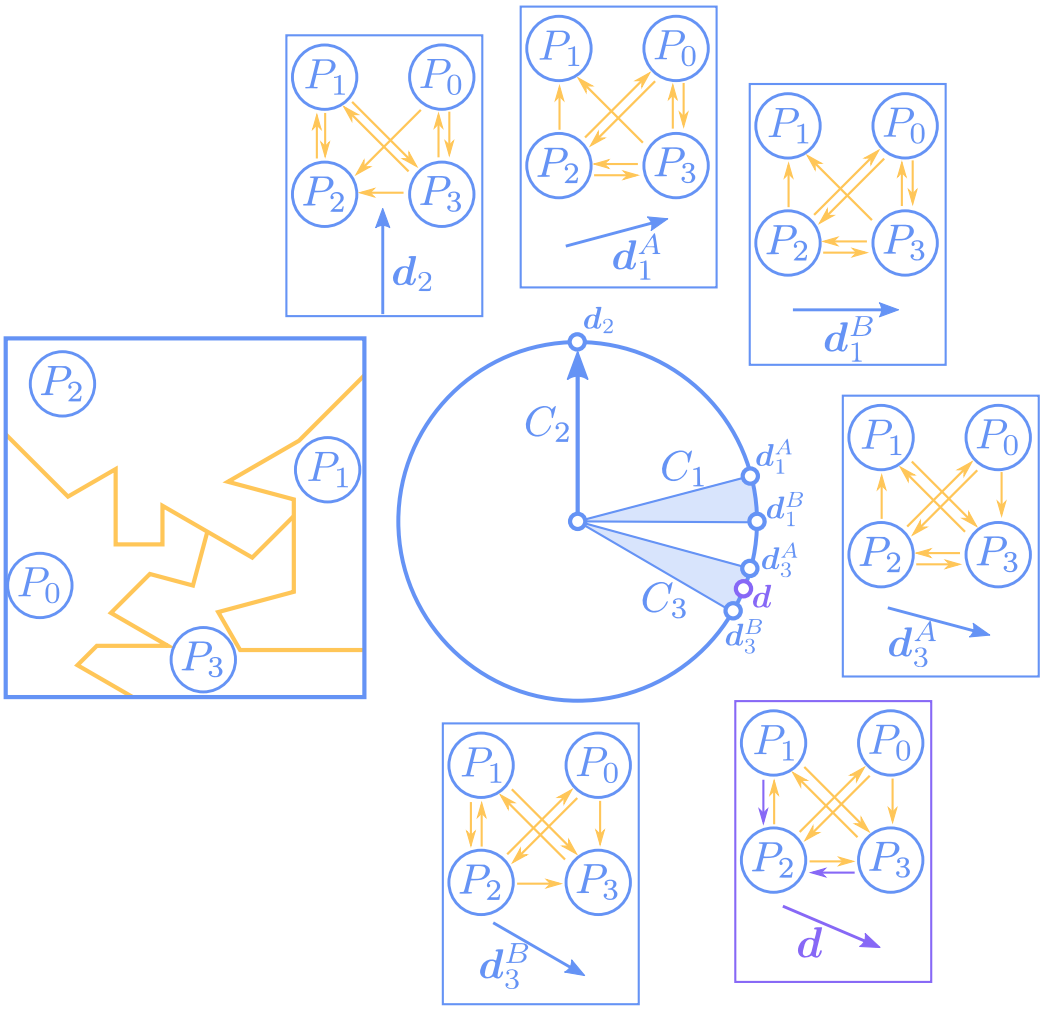


Figure 12: On the left, an assembly A made of 4 parts and a representation of its NDBG (restricted to translation). For $i \in \{1, 2, 3\}$ part P_i obeys cone C_i (C_2 being reduced to a single direction). The DBGs associated with several directions $\boldsymbol{d}_j$ (seen as points on the unit circle) are also depicted.

A *Non Directional Blocking Graph* (NDBG) is simply the concatenation of all DBGs $G(\boldsymbol{d}, A)$ for all possible motions $\boldsymbol{d}$. The NDBG of A can be represented on the 2D unit circle, seen as the locus of all directions of translation in 2D.

As illustrated on FIGURE 12, a useful property of an NDBG is that if a part P_i

obeys the two directions $\boldsymbol{d}_p$ and $\boldsymbol{d}_q$ bounding a cone of translational freedom C_i then [18] showed that for any $\boldsymbol{d}$ in the interior of C_i $G(\boldsymbol{d}, A)$ is obtained by performing a union operation of the two DBGs $G(\boldsymbol{d}_p, A)$ and $G(\boldsymbol{d}_q, A)$. In other words one only needs to compute the DBGs of the boundaries of a cone to know every possible DBG in that cone.
For instance, on FIGURE 12 cone C_3 is bounded by directions $\boldsymbol{d}_3^A$ and $\boldsymbol{d}_3^B$ and the DBG of the purple point $\boldsymbol{d}$ in the open arc $]\boldsymbol{d}_3^A, \boldsymbol{d}_3^B[$ is the union of the DBGs associated to the bounding directions $\boldsymbol{d}_3^A$ and $\boldsymbol{d}_3^B$. To see this one can notice that edge $e_{1\to 2}$ (resp. $e_{3\to 2}$) is present (resp. absent) in $G(\boldsymbol{d}_3^B, A)$ and absent (resp. present) in $G(\boldsymbol{d}_3^A, A)$ but both are present in $G(\boldsymbol{d}, A)$ (the corresponding arrows are highlighted in purple).
Moreover, as stated by [7], half of the circle is redundant as $G(-\boldsymbol{d}, A)$ can be obtained by reversing the direction of the edges in $G(\boldsymbol{d}, A)$. In that sense we can fully assess the NDBG, i.e., the blocking relations of a sequential assembly, by studying the strong connectedness of a finite number of DBGs $G(\boldsymbol{d}, A)$ obtained for the endpoints $\boldsymbol{d}$ of the different user-prescribed cones of translational freedom C_i. Following [7] we call such DBGs the *base DBGs* of A.

3.2. Turtle graphics and Markov process

Turtle graphics is a popular way to introduce children to the basics of coding: a virtual `Turtle` is displayed on the screen of the computer and moves in the 2D plane according to instructions given by the user while leaving a trace on its path. These instructions are of the form "*Walk by l unit*"; "*Rotate by θ radians*". The children are then tasked to find and code a sequence of instructions that lead the `Turtle` to draw some objective design: a square, a star of David, or any more complex shape. Similarly to [22] we will use such `Turtle` as an agent that draws the polylines partitioning the design domain into parts constituting our assembly.

In probability theory, a **Markov process**, or Markov chain, ([23]) is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event (it is *memoryless*). A state is said absorbing if once entered the probability to leave it is 0. More specifically a discrete-time Markov process is a Markov chain with a countable number of states and the chain iteratively transitions between states at discrete time steps according to some probabilistic rules.

In the present study, we introduce a discrete-time Markov chain with a finite number of states and one absorbing state, herein referred to simply as Markov chain, to play the role of children in Turtle graphics as the one making up the sequence of orders to move the `Turtle` with. Formally speaking, such a Markov process $\mathcal{M}$ is defined as a tuple $(\mathcal{V}, \mathcal{P})$ where

- $\mathcal{V}$ is the set of possible states that the chain will transition between. In our case $\mathcal{V} = \{\texttt{start}, \texttt{rotate}, \texttt{walk}, \texttt{snap}, \texttt{end}\}$.
- State `end` is absorbing: once $\mathcal{M}$ reaches this state it cannot leave it.
- $\mathcal{P}$ is the set of probabilistic rules specifying which transitions are available as well as their weights. It defines mappings $p : \mathcal{V} \to \mathcal{V}$. We provide here a succinct description of the rules emitted by the chain $\mathcal{M}$ and how they are interpreted by the `Turtle`. More details are given SECTION 4.
  Seven rules are defined in $\mathcal{P}$:

  0. $\texttt{start} \mapsto \texttt{rotate}$

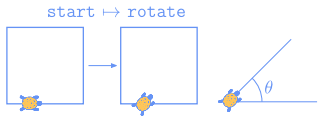


Figure 13: The `Turtle` is randomly initialised on the boundary of the design domain and this order simply tells it to choose a random orientation parameterised by angle θ.

  1. $\texttt{rotate} \mapsto \texttt{walk}$

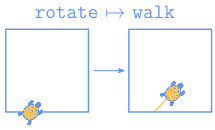


Figure 14: The `Turtle` has already chosen an orientation and must now walk forwards by a random amount.

  2. $\texttt{rotate} \mapsto \texttt{end}$

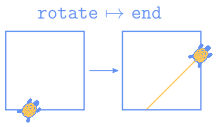


Figure 15: The `Turtle` has already chosen an orientation and walks forward until meeting an edge of the design domain.

3. `walk` ↦ `rotate`

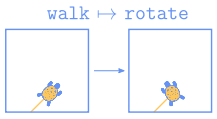


Figure 16: The `Turtle` has walked to a new position and must now choose a new random orientation.

4. `walk` ↦ `snap`

   The `snap` order ensures that the assembly is compatible with the prescribed disassembling motions by orienting the `Turtle` collinearly with the bounds $\boldsymbol{d}^A$ and $\boldsymbol{d}^B$. Details are provided SECTION 4.

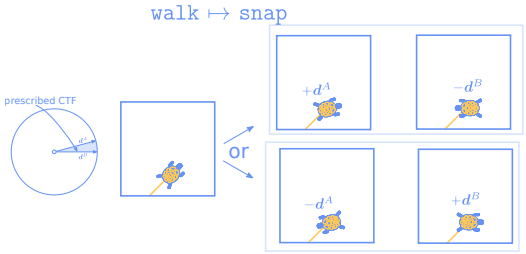


Figure 17: Context: the leftmost blue cone represents the cone of translational freedom (CTF) prescribed by the user: the final design must obey any direction in the cone bounded by $\boldsymbol{d}^A$ and $\boldsymbol{d}^B$. The `Turtle` has walked to a new position and must now choose to orient along either $\pm\boldsymbol{d}^A$ or $\mp\boldsymbol{d}^B$.

5. `snap` ↦ `walk`

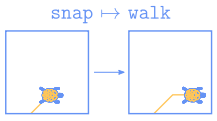


Figure 18: Similar to `rotate` ↦ `walk`: the `Turtle` has snapped to an orientation and walks forward by a random amount.

6. `snap` ↦ `end`

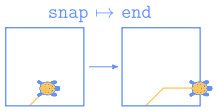


Figure 19: Similar to `rotate` ↦ `end`: the `Turtle` has snapped to an orientation and walks forward until meeting an edge of the design domain.

When two rules apply to the same left-hand side (for instance `rotate` ↦ `walk` and `rotate` ↦ `end`) then the Markov chain $\mathcal{M}$ randomly chooses one of the two with some predefined probability as depicted on FIGURE 20

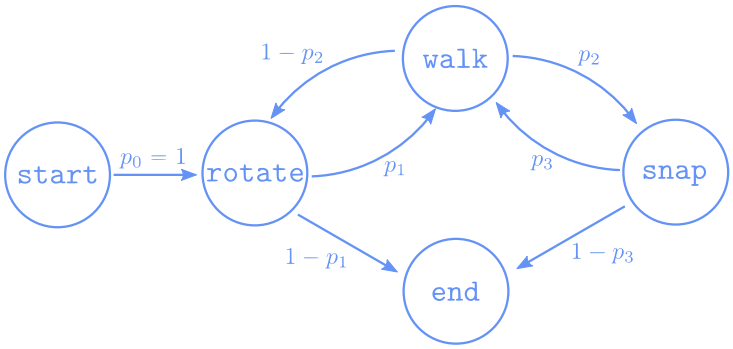


Figure 20: $\mathcal{M}$ transitions between states with some predefined probabilities p_i.

These strings are iteratively composed into a random sentence, for instance on FIGURE 21 the full sequence emitted by $\mathcal{M}$ is `start` ↦ `rotate` ↦ `walk` ↦ `snap` ↦ `walk` ↦ `rotate` ↦ `walk` ↦ `rotate` ↦ `walk` ↦ `rotate` ↦ `walk` ↦ `snap` ↦ `walk` ↦ `rotate` ↦ `walk` ↦ `rotate` ↦ `end`. At each iteration of the algorithm the `Turtle` receives one of these 5 strings, interprets it as an order and acts accordingly.

Since a sequence of the form [`rotate` or `snap`] ↦ `walk` defines a segment and a polyline is simply constituted of line segments jointly concatenated, this Markov process $\mathcal{M}$ is sufficient to draw a polygonal assembly. We do not need to add rules of the kind `rotate` ↦ `rotate` as the final orientation could very well be obtained after a single `rotate` order. Similarly a `walk` ↦ `walk` can be obtained with the sequence `walk` ↦ `rotate` ↦ `walk` where the `Turtle`

happens to keep the same orientation before and after the `rotate` order. We will prove in SECTION 4.1 that the Markov chain $\mathcal{M}$ aided by the `Turtle` can reach the full search space of polygonal assembly.

4. Algorithm and results

We propose to define a Markov process $\mathcal{M}$ that will instruct a `Turtle` so that the latter draws a polyline dividing the design domain into two parts, one of which obeying a user-prescribed motion $\boldsymbol{d}$. By repeatedly running this algorithm we can partition the domain into several polygonal parts P_i defining an assembly A. By checking the strong-connectedness of the $G(\boldsymbol{d}, A)$ for the different user-prescribed $\boldsymbol{d}$ we can assess whether A is globally interlocked. Let us dive into the details of the generation of, first, parts obeying a translation and, then, of parts obeying a rotation.

4.1. Parts obeying a translation

4.1.1. On the creation of a single part

The user decides on two vectors of translation $\boldsymbol{d}_1^A$ and $\boldsymbol{d}_1^B$ bounding the cone of translational freedom of the would-be part P_1. Enforcing $\boldsymbol{d}_1^A = \boldsymbol{d}_1^B$ leads to the special case where the cone is reduced to a single vector that is to say where P_1 must translate along one direction only. Then a Markov chain $\mathcal{M}$ emits the `start` order which initialises a `Turtle` randomly on the boundary of the design domain. In subsequent iterations $\mathcal{M}$ tells the `Turtle` whether to orient itself or to move. To comply with a `walk` order, the length l by which the `Turtle` moves is randomly picked in a user-defined interval $[l_{\min}, l_{\max}]$ (to have consistent step size, or edge length). To obey a `rotate` order, the rotation angle θ is randomly chosen in an interval such that the normal vector $\boldsymbol{n}$ of the line segment is such that $\boldsymbol{n} \cdot \boldsymbol{d}_1^A \geq 0$ and $\boldsymbol{n} \cdot \boldsymbol{d}_1^B \geq 0$. The reader's attention is drawn to the fact that this constraint on θ is enough to prevent the `Turtle` from crossing its own path.

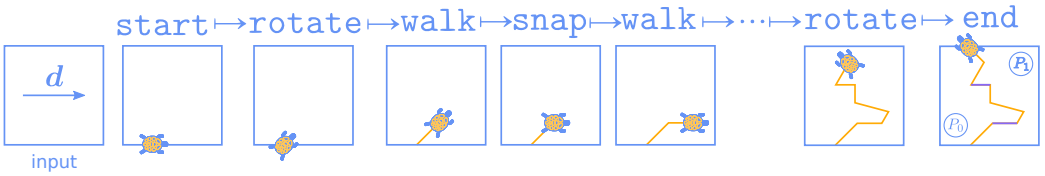


Figure 21: A step-by-step decomposition of the `Turtle`'s trajectory. Highlighted in purple on the far right are the line segments corresponding to a `snap` order.

If the sequence of orders and random values leads the `Turtle` to wander outside of the design domain, a backtracking procedure is executed to replace it inside. At the end of an iteration, $\mathcal{M}$ randomly applies a production rule on the latest order it gave to get the one for the next iteration. Finally, when $\mathcal{M}$ emits the `end` order, the `Turtle` walks until meeting an edge of the design domain.

Obviously, letting the `Turtle` move like this is likely to yield very poor results as the random separating polyline obtained at the end will be such that the actual cone of translational freedom of P_1 strictly includes the user-prescribed cone bounded by $\boldsymbol{d}_1^A$ and $\boldsymbol{d}_1^B$. As an extreme example imagine that the user specified a vector $\boldsymbol{d}_1^A = \boldsymbol{d}_1^B \equiv \boldsymbol{d}_1 = (1, 0)^T$ aligned with the x axis (meaning that the user wants P_1 to translate along the horizontal axis only) and the `Turtle` drew a single line segment, as illustrated on the top row of FIGURE 22. Even though P_1 does obey $\boldsymbol{d}_1$ it also obeys a full half-space of motion which is an undesirable behaviour.

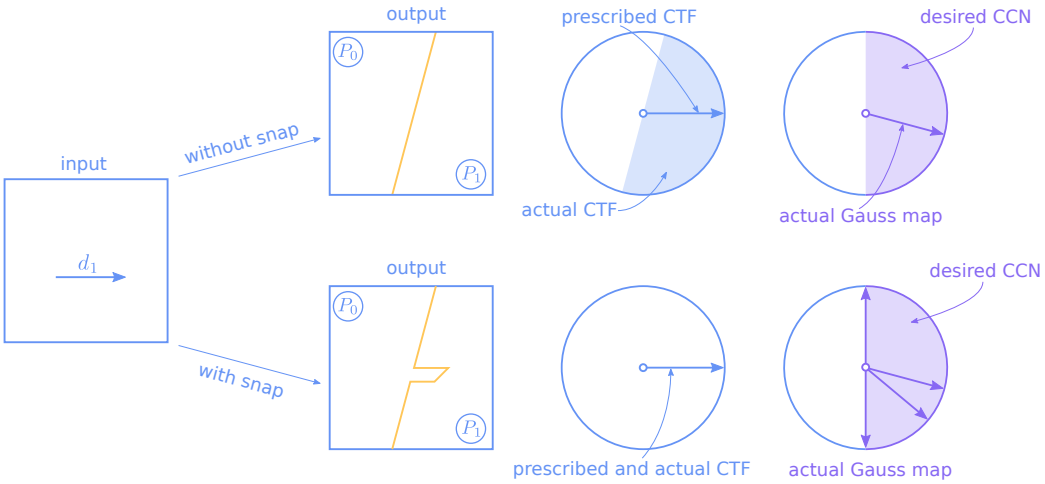


Figure 22: CTF and CCN respectively stand for *Cone of translational freedom* and *Cone of compatible normals*. The Gauss map of the dividing polyline is depicted using purple arrows on the rightmost circles.

In order to reduce the actual cone of translational freedom of P_1 to the user-prescribed one, the Markov chain $\mathcal{M}$ is enriched by a special state that we call `snap`: it forces the parameter θ to be chosen such that the `Turtle` is oriented either along $\pm\boldsymbol{d}_1^A$ or $\mp\boldsymbol{d}_1^B$. The signs ± 1 and ∓ 1 are randomly chosen at the initialisation of the `Turtle`. This ensures that when the `Turtle` `snaps` it draws a line segment whose unit normal vector $\boldsymbol{n}$ is such that either

$\boldsymbol{n} \cdot \boldsymbol{d}_1^A = 0$ and $\boldsymbol{n} \cdot \boldsymbol{d}_1^B \geq 0$ or *vice-versa* by switching the superscripts ^{A} and ^{B}. As such, the `Turtle` draws a valid polyline (i.e., such that the cone of translational freedom of P_1 exactly matches the user-prescribed cone) if and only if it snapped at least twice, once along $\pm\boldsymbol{d}_1^A$ and once along $\mp\boldsymbol{d}_1^B$, which gives a computationally light manner to check whether a polyline is valid. On FIGURE 22, bottom row, one `snap` order ensures that the Gauss map of the polyline includes a normal vector bounding the cone of compatible normals. Snapping at least twice in opposite directions ensures that two normal vectors will match the bounds of the cone of compatible normals (vertical upwards and downwards purple arrows on the right).

4.1.2. Surjectivity of the Markov process $\mathcal{M}$

We justify here that the Markov process $\mathcal{M}$ associated with the `Turtle` are sufficient to reach any polygonal assembly, i.e the mapping from the set made of $\mathcal{M}$ and the space of the `Turtle`'s parameters (l and θ) to the space of polygonal assembly is surjective.

Any polyline separating two parts must fit in the design domain. This observation gives an obvious upper bound on the value of $l_{\max}$, which could be the length of the diagonal of the bounding square of the design domain. In addition, setting $l_{\min} = 0$ ensures that the `Turtle` can draw infinitely small line segments and as such the full space of polygonal parts can be reached. But the mapping is not injective: for instance it is possible, although unlikely, that the magnitudes to walk or rotate by chosen by the `Turtle` for the sequence `rotate` $\mapsto$ `walk` $\mapsto$ `rotate` $\mapsto$ `walk` on the one hand and `rotate` $\mapsto$ `walk` on the other lead the same line segment see FIGURE 23. As such two identical polylines can be obtained through two different sequences of orders and the mapping is not injective. Moreover from a practical point of view letting the `Turtle` draw infinitely small segments might not be desirable and the user may want to reduce the search space to the subset of polylines having a minimal segment length $l_{\min} > 0$. The upper bound $l_{\max}$ can also be reduced to some smaller value as any polyline with a segment length greater than $l_{\max}$ can still be reached by walking several times in the same direction. Thus the mapping to this subset is still surjective.

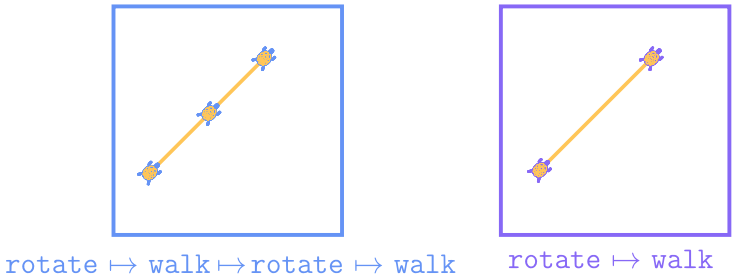


Figure 23: The same line segment can be reached by walking twice in the same direction by a magnitude $\frac{l}{2}$ or once by a magnitude l, for some $l \in [l_{\min}, l_{\max}]$, and thus the map is not injective.

Note that this mapping can be made injective by reducing a sequence emitted by $\mathcal{M}$ to the smallest possible word by tracking the times where the `Turtle` rotated by 0 rad (or snapped consecutively) and replacing instructions "$\texttt{rotate}(\theta_i) \mapsto \texttt{walk}(l_i) \mapsto \texttt{rotate}(0) \mapsto \texttt{walk}(l_{i+1})$" with "$\texttt{rotate}(\theta_i) \mapsto \texttt{walk}(l_i + l_{i+1})$".

4.1.3. On the creation of an interlocked assembly

To proceed with creating an assembly $A = \{P_0, ..., P_N\}$ we simply repeatedly run the above algorithm in order to subdivide the remaining part P_0 into two parts at each step (see FIGURE 24). At the end of every iteration the base DBGs are computed and if they are all strongly connected (but the one associated with the first cone of motion that must say that P_1 is free to translate in this cone) the latest polyline drawn by the `Turtle` is accepted and the next iteration starts. If not, this latest polyline is discarded and the `Turtle` must start drawing it again. FIGURE 24 illustrates the creation of a translational assembly made of 3+1 parts.

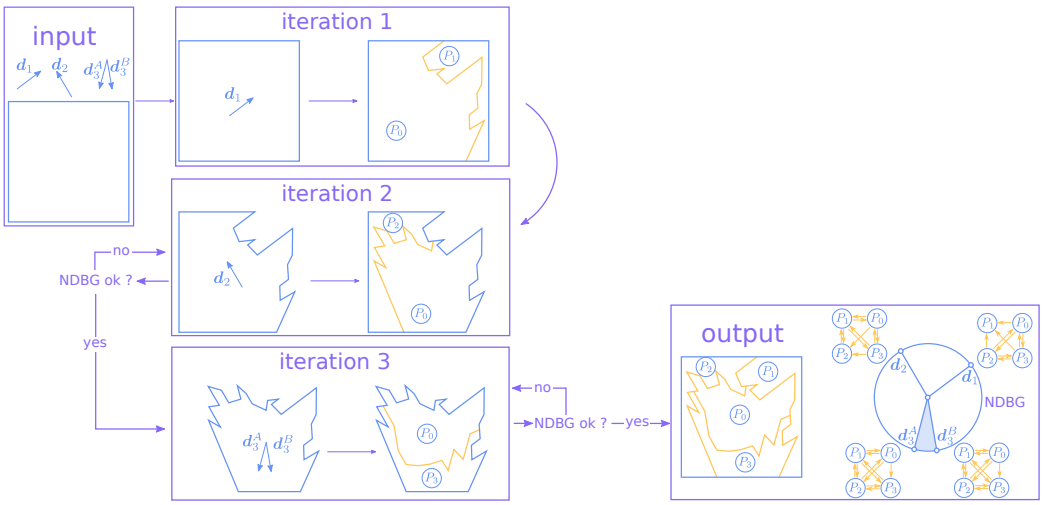


Figure 24: Workflow of the generation of a translational assembly made of 3+1 parts

A necessary condition for a part P_i to be blocked by an other part P_j, $0 < j < i$, is for the two of them to share a boundary. To ensure contact and thus increase the odds of the DBGs being strongly connected we bias the `Turtle` to always choose its starting point on a boundary shared by an other P_j, $0 < j < i$.
The reader's attention is drawn to the fact that the user cannot prescribe two cones of translational freedom related to two successive parts, say P_i and P_{i+1}, to intersect: indeed it would mean that there exists a direction in the intersection of the cones such that P_i and P_{i+1} can simultaneously obey it and in practice, one part could push the other out of the way. However the user can specify intersecting cones related to non-successive parts, say P_i and P_j with $|i - j| > 1$, and leave it to the `Turtle` to create an interlocked assembly.

Figure 25 presents four assemblies obeying translations generated by our method where the leftmost column depicts the user-prescribed cone of translational freedom.

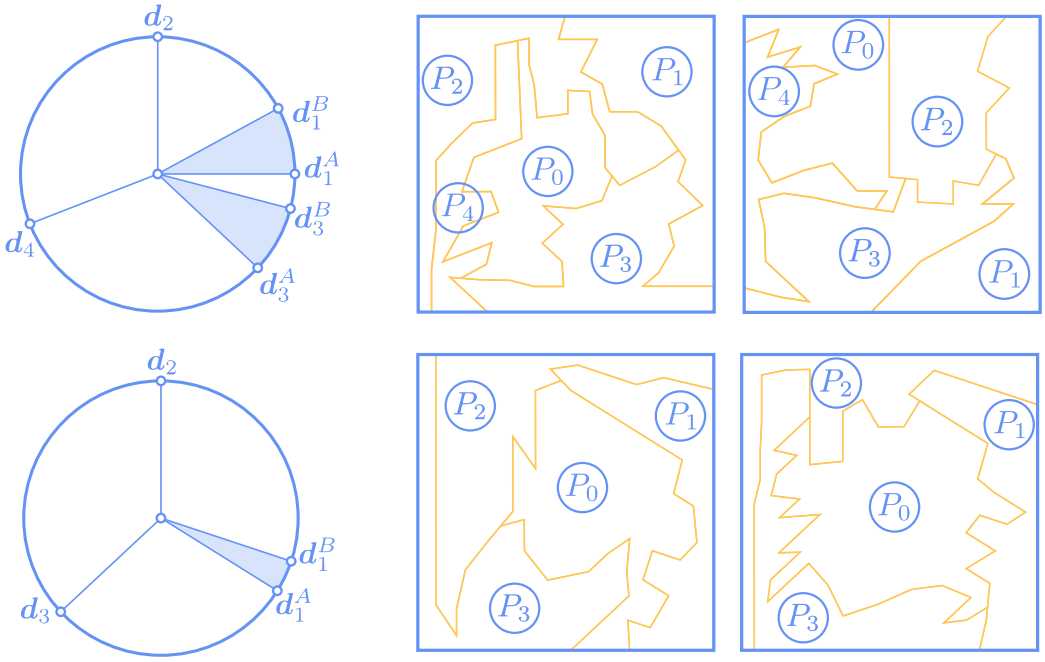


Figure 25: Assemblies working purely in translation. $N = 4$ on the top row and $N = 3$ on the bottom row.

A demonstration of the assembling and disassembling motions on a real laser-cut assembly obeying translations is proposed at 0:31 in our video [8].

4.2. Parts obeying a rotation

The workflow to generate an assembly obeying rotations is essentially the same as the one for translational assemblies except that the angular interval in which parameter θ, the `Turtle`'s orientation, is chosen must be dynamically updated as it depends on both the `Turtle`'s current position and the step size l.

In a general setting the point $\hat{\boldsymbol{x}} = (\hat{x}, \hat{y})$ obtained by rotating a point $\boldsymbol{x} = (x, y)$ by an angle ψ around a centre point $\boldsymbol{d} = (x_d, y_d)$ is given by:

$$\begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} = \begin{pmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} x - x_d \\ y - y_d \end{pmatrix} + \begin{pmatrix} x_d \\ y_d \end{pmatrix}$$

Because we restrict this study to infinitesimal motions, we assume $|\psi| \ll 1$

and a first-order Taylor expansion yields

$$\begin{pmatrix}\hat{x}\\ \hat{y}\end{pmatrix} = \begin{pmatrix}x\\ y\end{pmatrix} + \psi \begin{pmatrix}y_d - y\\ x - x_d\end{pmatrix} \tag{2}$$

EQUATION (2) states that an infinitesimal rotation of $\boldsymbol{x}$ around $\boldsymbol{d}$ is the same as an infinitesimal translation of $\boldsymbol{x}$ along the vector $\boldsymbol{m}_x = \begin{pmatrix}y_d - y\\ x - x_d\end{pmatrix}$. We call $\boldsymbol{m}_x$ the *instantaneous direction of motion* of point $\boldsymbol{x}$. Note that this vector is orthogonal to the line going through $\boldsymbol{x}$ and $\boldsymbol{d}$.
A necessary and sufficient condition for the part P_1 of an assembly $A = \{P_0, P_1\}$ to obey a rotation around $\boldsymbol{d}$ is to have the instantaneous directions of motion of all points $\boldsymbol{x}$ on the boundary of P_1 **not** pointing towards the interior of P_0. Indeed if there is one point $\boldsymbol{x}$ on the boundary of P_1 such that $\boldsymbol{m}_x$ points towards the interior of P_0 then an infinitesimal rotation around $\boldsymbol{d}$ will send that point to collide with P_0 which exactly means that P_1 does not obey $\boldsymbol{d}$. Moreover, since we focus on infinitesimal motions, we only need to study the points on the boundary of both P_0 and P_1. In other words the fact that P_1 obeys $\boldsymbol{d}$ depends only on the geometry of the polyline separating P_0 from P_1.

While the `Turtle` is walking we do not know on which side of its path P_1 will be. What we do know is that P_1 will always be on the same side of this path: from the `Turtle`'s point of view, P_1 will either always be to the left or always be to the right. That is to say the instantaneous direction of motion $\boldsymbol{m}_x$, of each point $\boldsymbol{x}$ on the `Turtle`'s path, shall always be pointing either "*to the left*" or "*to the right*" of the path. Mathematically speaking, for any point $\boldsymbol{x}$ on the `Turtle`'s path we denote by θ the orientation of the line segment of the polyline on which $\boldsymbol{x}$ is and the condition becomes:

$$s \begin{pmatrix}\cos\theta\\ \sin\theta\end{pmatrix} \times \boldsymbol{m}_x \geq 0 \tag{3}$$

where $\times$ denotes the 2D cross-product and $s = \pm 1$ is a constant sign, randomly decided at the initialisation of the `Turtle`, that stipulates on which side of the path $\boldsymbol{m}_x$ shall be pointing to. After a few calculations, developed in appendix A, one derives the formula:

$$\text{EQUATION}(3) \Leftrightarrow \forall i \ \begin{cases} s\boldsymbol{l}_i \times \boldsymbol{m}_{x_i} \geq 0 \\ s\boldsymbol{l}_i \times \boldsymbol{m}_{x_{i+1}} \geq 0 \end{cases} \tag{4}$$

where $\boldsymbol{m}_{x_{i+1}} = \boldsymbol{m}_{x_i} + l_i \begin{pmatrix} -\sin\theta_i \\ \cos\theta_i \end{pmatrix}$ and $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{l}_i$, with vector $\boldsymbol{l}_i = l_i \begin{pmatrix} \cos\theta_i \\ \sin\theta_i \end{pmatrix}$. Generally speaking SYSTEM (4) admits an angular interval as a solution.

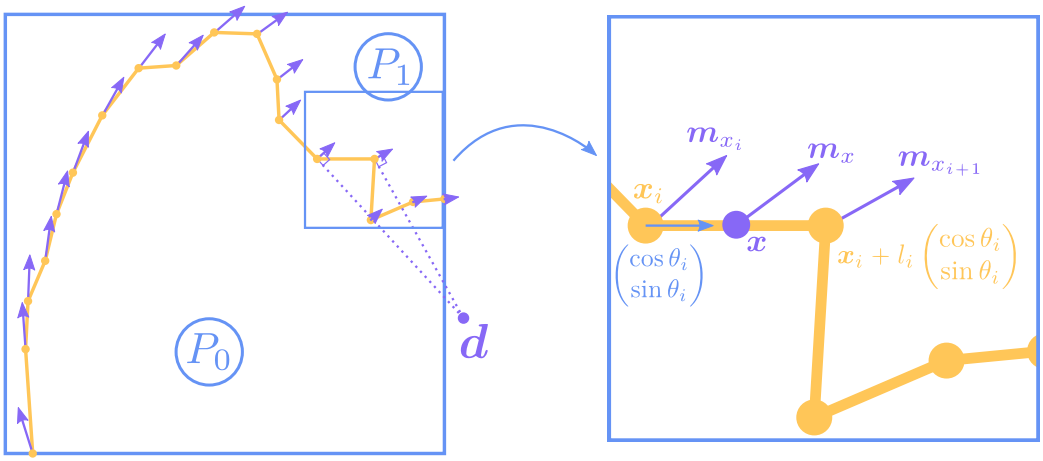


Figure 26: The purple arrows represents the instantaneous directions of motions of each yellow point. A zoom on the right shows that the cross product between $\begin{pmatrix} \cos\theta_i \\ \sin\theta_i \end{pmatrix}$ and vectors $\boldsymbol{m}_{x_i}$, $\boldsymbol{m}_{x_{i+1}}$ and $\boldsymbol{m}_x$ are all of the same sign meaning that, at least for the line segment under scrutiny, a rotation around centre $\boldsymbol{d}$ is possible.

Concretely when the `Turtle` is at position $\boldsymbol{x}_i$ it chooses a step size l_i and SYSTEM (4) is solved for to get an angular interval in which to randomly choose the `Turtle`'s orientation θ_i. This ensures that all the points $\boldsymbol{x}$ on segment $[\boldsymbol{x}_i, \boldsymbol{x}_{i+1}]$ obey a rotation around centre $\boldsymbol{d}$, see FIGURE 26. Observe that we can still define a `snap` order in this case: when $\mathcal{M}$ tells the `Turtle` to `snap`, the latter's orientation θ_i is chosen as one of the two bounds of the angular interval defined by SYSTEM (4). This ensures that one of the two inequalities of SYSTEM (4) becomes an equality and geometrically speaking the `Turtle` moves radially with respect to centre $\boldsymbol{d}$.

Now that we have understood how to compute the angular interval in which to choose the orientation θ, we can simply plug this calculation in the algorithm described in SECTION 4.1 and generate an assembly obeying rotations:

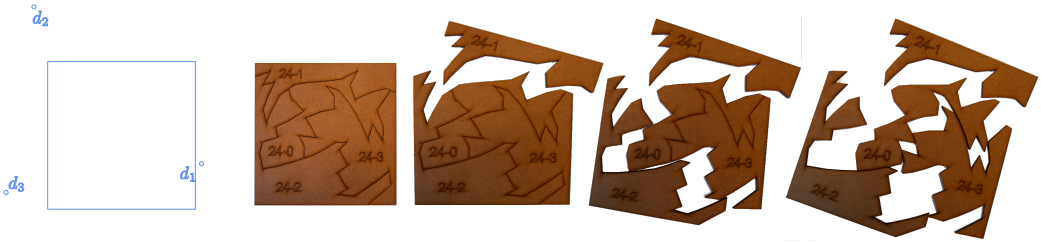


Figure 27: An assembly made of 3+1 parts, each obeying a rotation.

As the `Turtle` walks it is possible to define the on-going cone of translational freedom of the polyline under construction. As the current part shall rotate but not translate the algorithm keeps computing what angle θ would close such cone of freedom (meaning it would find θ such that the normal vector $\boldsymbol{n}$ of the current line segment would not be compatible with SYSTEM (1) for any direction of translation). If such θ happens to be solution to SYSTEM (4), then it is greedily chosen to orient the `Turtle`, which ensures that the newly created part will obey its rotation but will not obey any translation. The parts depicted on FIGURE 27 are obtained through this procedure. A demonstration of the (dis)assembling motions as well as the interlocking aspect of the design presented on FIGURE 27 is available at 0:06 in our video [8].

A note on the NDBG: SECTION 3.1 shows that for an assembly working in translation the directions $\boldsymbol{d}$ of the DBGs $G(\boldsymbol{d}, A)$ are vectors and can be seen as points on the unit circle (and the NDBG can be fully represented on the unit circle). For an assembly working in rotation the motions $\boldsymbol{d}$ are centres of rotation, i.e., points anywhere in the plane. As such the NDBG of such an assembly is represented on the tuple made of twice the $\mathbb{R}^2$ plane, where each point is a centre of rotation and is taken twice to account for both clockwise and counterclockwise rotations.

FIGURE 25 presents four assemblies obeying rotations generated by our method. The leftmost column depicts the user-prescribed positions of the centres of rotation with respect to the square design domain. The assemblies were generated for $N \in \{3, 4\}$.

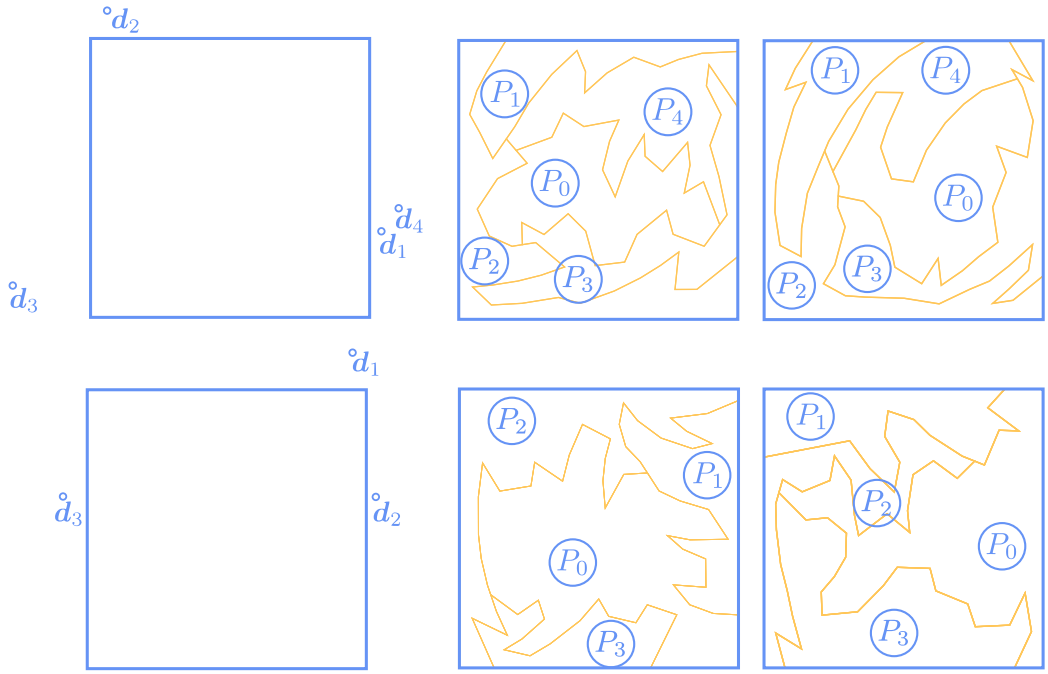


Figure 28: Assemblies working purely in rotation.

4.3. Parts obeying both a translation and a rotation

Building on top of the two previous sections it is quite easy to adapt our algorithm to design parts obeying both translations and rotations. Say the user wants P_i to obey simultaneously a translation along any vectors in the cone bounded by the vectors $\boldsymbol{d}_i^A$ and $\boldsymbol{d}_i^B$ and a rotation around the centre $\boldsymbol{d}_i^C$. To succeed, one simply needs to intersect the angular interval defined for the translation cone and the angular interval defined by SYSTEM (4) and choose the orientation θ in the resulting interval.

Figure 29: An assembly made of 3+1 parts: P_1 obeys a rotation around $\boldsymbol{d}_1^r$; P_2 obeys both a translation along $\boldsymbol{d}_2^t$ and a rotation around $\boldsymbol{d}_2^r$; P_3 obeys a translation along $\boldsymbol{d}_3^t$

In such case the Markov chain $\mathcal{M}$ distinguishes between a `snapT` (translation) and a `snapR` (rotation) and the rest of the algorithm stays the same. Note that special care must be taken by the user to define $\boldsymbol{d}_i^A$, $\boldsymbol{d}_i^B$ and $\boldsymbol{d}_i^C$ to be compatible with each other: one cannot ask for a rotation around $\boldsymbol{d}_i^C$ that would lift P_i up and at the same time define $\boldsymbol{d}_i^A$, $\boldsymbol{d}_i^B$ to be pointing downwards. FIGURES 1 and 29 present laser-cut puzzles generated by our `Turtle` where each part obeys a rotation and/or a translation. On FIGURE 30 part P_1 obeys both a rotation ($\boldsymbol{d}_1^r$) and translation ($\boldsymbol{d}_1^t$) while parts P_2 and P_3 must rotate ($\boldsymbol{d}_2^r$ and $\boldsymbol{d}_3^r$).

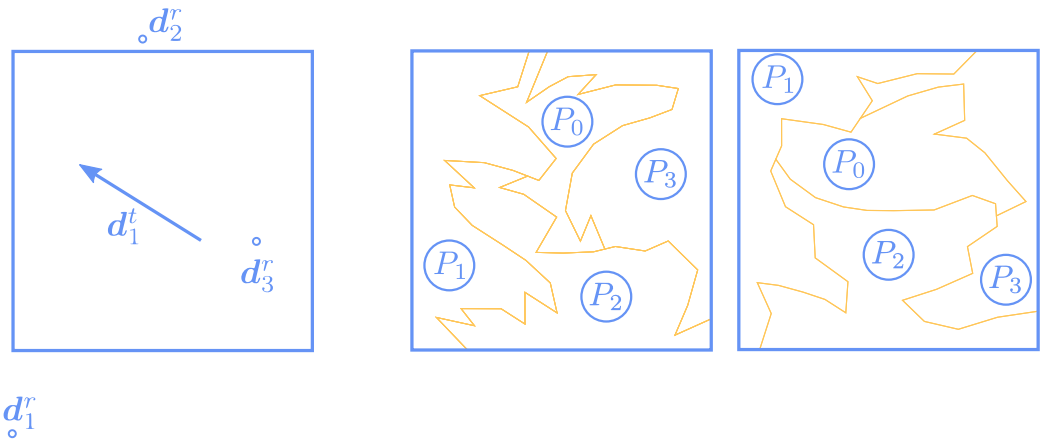


Figure 30: Assemblies working both in rotation and in rotation.

5. Computation time

By letting the algorithm run the way it has been described so far, the greater the number of parts N the less likely is it that a solution assembly will be found. In fact, the algorithm may fail to draw a polyline (as soon as $N \geq 2$). Indeed, referring to FIGURE 24, the design domain accessible to the `Turtle` when it is tasked to draw the next part P_{i+1} is what was the remaining part, P_0, at the previous iteration. Thus the bigger are the previous parts P_j, $0 < j < i$, the smaller is the design domain at iteration $i+1$, and if it is too small there is simply not enough space to successfully create the next part given the step resolution interval $[l_{\min}, l_{\max}]$. Consequently, the greater N the more the more difficult it is to generate successive parts and the completion time may become infinite.

As a consequence, two measures have been implemented:

- To ensure a design domain large enough at all iterations, a part is discarded if its area is greater than $\frac{A}{N+1}$ (A being the area of the initial design domain, the square in our examples) regardless of the NDBG, and has to be drawn again. This constraint ensures that all parts are of similar areas and that the remaining part P_0 is relatively big at all iterations.

- If the creation of a part P_{i+1} fails $M \in \mathbb{N}^*$ successive times (i.e. the related DBGs are not strongly-connected and/or its area is above the threshold) then it is assumed that part P_i, even though valid, has strange geometrical features making it burdensome for the `Turtle` to succeed. Consequently, the algorithm backtracks: P_i is deleted and must be generated again.

These two measures ensure that a valid interlocked assembly will eventually be found but give no certainty on the completion time (empirically, in translation, it seems to be in $\mathcal{O}(N^4)$). To understand it we need to quickly prove some results: for $0 < k \leq N$, at iteration k, let A_k be the area of the design domain in which part P_k shall be drawn (for instance $A_1 = A$, the area of the initial design domain). At this k^{th} step $k-1$ parts P_j, $0 < j < k$, already exist, each with an area smaller than $\frac{A}{N+1}$. As such $A_k \geq A - \sum_{j=1}^{k-1} \frac{A}{N+1} = \frac{A(N+2-k)}{N+1}$. Thus the probability $\mathbb{P}^k$ to draw a part P_k with an area less than $\frac{A}{N+1}$ is $\mathbb{P}^k = \frac{\frac{A}{N+1}}{A_k} \leq \frac{1}{N+2-k}$. Moreover, for $k > 1$, for a part P_k to be valid all the relevant DBGs in the NDBG must be strongly-connected, which is not a given, i.e. the probability that P_k meet this constraint is strictly less than 1 (it is not the case for P_1 as there is no NDBG to compute). Thus for $k > 1$ the probability that a part P_k meets both constraints (area below the threshold and strongly-connected DBGs) is strictly less than $\frac{1}{N+2-k}$ (and is exactly $\frac{1}{N+1}$ when $k = 1$).
In practice, since P_1 only needs to meet the area constraint (probability $\frac{1}{N+1}$), it happens to be quite fast to generate. The crux of the issue lies with part P_2: it has to meet both constraints and the probability to get a valid part is strictly less than $\frac{1}{N}$ (and empirically, it seems to be much lower). Once such P_2 is obtained, it becomes easier and easier to draw the subsequent parts and in particular P_N is also quite fast to generate. However, see the second bullet point, if the algorithm fails M times to draw a part P_{i+1} then P_i is deleted: when that happens for part P_3 and P_2 (which is quite likely for a sufficiently large N, see FIGURE 31), P_2 has to be redrawn from scratch

which again takes a lot of computational time. Thus the value of M has to be carefully picked: if M is too small then a part P_i might be deleted whereas a solution would have been found had the algorithm ran longer; conversely if M is too big the algorithm keeps trying to generate P_{i+1} whereas the shape of P_i makes it challenging to succeed and simply deleting P_i and redrawing it anew would have lead to a faster generation of P_{i+1}, see TABLE 2 where the optimal value of M seems to be around 15.

	Translation			Rotation		
# parts	Min (ms)	Average (ms)	Max (ms)	Min (ms)	Average (ms)	Max (ms)
1	1	4	12	2	29	97
2	6	57	192	131	2048	8197
3	37	237	726	523	5288	14780
4	228	697	1469	7127	35416	89994
5	540	2039	5031	-	-	-
6	1607	8070	19302	-	-	-

Table 1: Completion time for assemblies made of {1,2,3,4,5,6} + 1 parts working either in translation or in rotation.

TABLE 1 shows the minimum, mean, and maximum times needed by the algorithm to generate an assembly made of $N + 1$ parts (for $N \in [\![1, 6]\!]$) working either purely in translation or purely in rotation in a square design domain with $[l_{\min}, l_{\max}] = [0.05L, 0.1L]$, where L is the length of the diagonal of the square. The motions to obey, $\boldsymbol{d}$, were sampled randomly, and the completion times were averaged on 1050 designs.

M	5	10	15	20	50	100	200
Min (s)	-	5.1	2.4	2.1	5.7	9.0	6.4
Average (s)	-	15.9	13.1	13.8	15.8	20.8	30.7
Max (s)	-	26.1	23.6	29.7	26.4	35.4	69.7

Table 2: Completion time for assemblies made of 5 + 1 parts working purely in rotation for various values of M.

The parameters used to obtain TABLE 2 are the same as those used for TABLE 1. This table confirms that the value of M strongly influences the completion time. In particular, not a single complete assembly was found for $M = 5$.

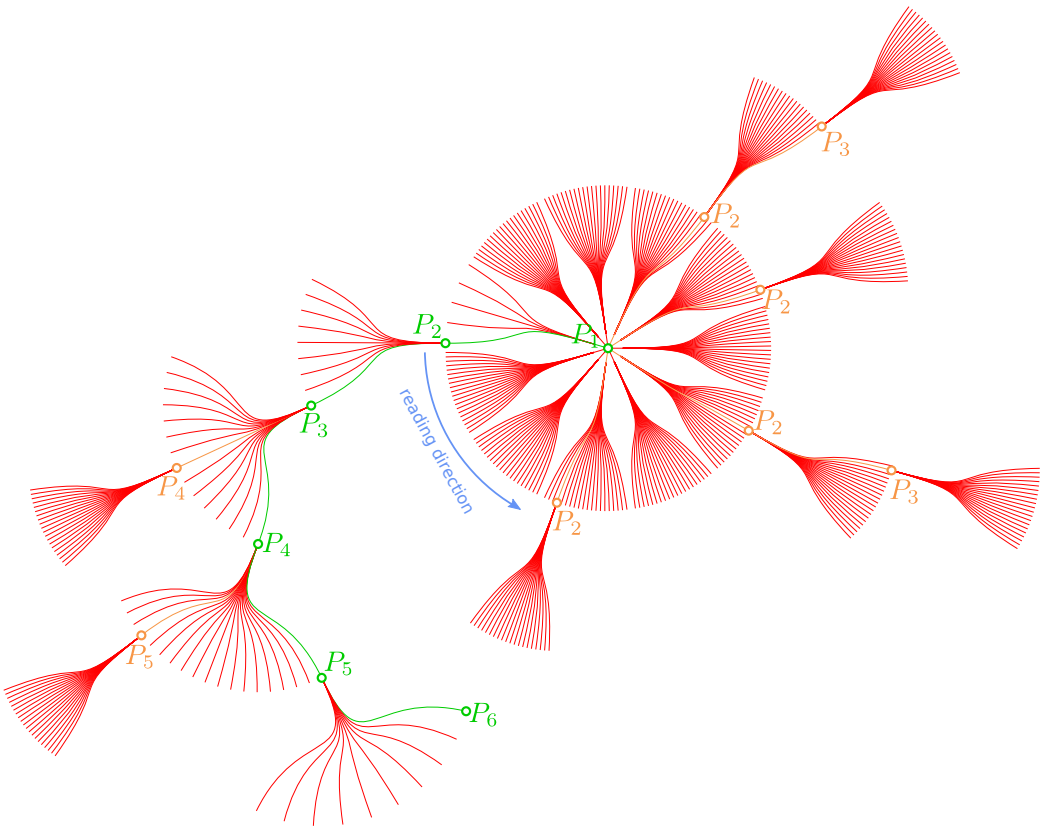


Figure 31: Regarding an assembly made of 6+1 parts, this tree illustrates the search carried out by the `Turtle` before finding a correct assembly. Here $M = 20$.

On FIGURE 31 the goal was to generate an assembly made of 6+1 parts. Each of the designs was recorded and arranged in a tree. For clarity, paths related to the generation of P_1 were omitted. Red paths represent fruitless paths: the `Turtle` drew a part P_i that does not meet one of the two aforementioned constraints. After failing $M = 20$ times the `Turtle` backtracks to the previous part. The orange paths depict paths such that a valid part P_i was successfully drawn but a subsequent part P_j, $j > i$; had we wanted $N = i$ parts the algorithm would have stopped at the latest at that path (it may have stopped earlier as the area threshold would have been less stringent and parts that have been discarded for $N = 6$ would have been accepted for $N < 6$). The green path is the valid path at the end of which the `Turtle` partitioned the design domain into a valid assembly. This figure clearly illustrates that most of the computation time was lost on generating fruitless

P_2 and that the greater i the fewer trials the algorithm needs before drawing a valid P_i.

6. Conclusion

We introduce a novel method to generate 2D sequential interlocking assemblies obeying translation, rotation, or a combination of both motions using an agent, called the `Turtle`, that partitions a design domain into polygonal parts. One of the main hypotheses of our work is the fact that the parts are polygonal, i.e. the separating curve between two parts is a polyline. Yet, as we derive the mathematical formula governing a valid assembly, one notices that these equations could readily be used to explore a broader solution space and design non-polygonal assemblies, for instance using nurbs instead of polylines to create the separating curves. A key feature of our work is the surjectivity of the mapping to the solution space. This ensures that, given enough trials and computation time, any polyline can be generated and thus that we explore homogeneously the full space of polygonal assemblies. Our approach yields surprising and novel assemblies but its main drawback is computational time: for some problems, especially the ones involving a combination of translation and rotation, the time needed to draw a valid assembly may become quite long. Future work shall be oriented towards a speeding up of the algorithm, possibly by running multiples `Turtles` in parallel, or by building a database and learning the statistical distribution of what makes an assembly valid and then sampling that distribution. Also, as highlighted by FIGURE 31, most of the computation time is lost on generating parts P_2 not meeting the area threshold or the NDBG constraint. The current, naive, approach to deal with such unfeasible designs is to backtrack, delete, and start drawing them anew. Yet, Wang *et al.*, in [24], suggests an interesting road to explore: for a given interlocked assembly, the authors optimise the geometry of the components to increase a stability score, and gradually make the assembly more structurally stable. In future work, their study shall be adapted so that for each design drawn by the `Turtle`, the polyline is optimised to increase the likeliness of the NDBG being strongly connected. Moreover, it would be interesting to enrich the set of states accessible to the Markov process $\mathcal{M}$, using a more interesting syntax, so that the `Turtle` can automatically draw predefined features that are of interest in the construction sector.

This study focuses on infinitesimal motions only. Whilst it concerns only

a small minority of the assemblies generated so far, it is quite possible for some designs that a finite motion to disassemble a part is impossible as it will collide with another part that cannot be removed before. Our video [8] illustrates such issue at 1:05. Furthermore, in this paper, we did not prove that if a part obeys a rotation about a centre $\boldsymbol{d}$ then $\boldsymbol{d}$ is the only centre of rotation of that part. The fact that a part may rotate around any point in the vicinity of centre $\boldsymbol{d}$ may prove useful in the context of toleranced assembly. Also, this work focuses only on the geometrical aspect of an assembly: as shown on FIGURE 5 the scope of this paper is on the design knowledge step. Yet for any real problem, the exploration step is crucial and one may want to assess and optimise some user-defined metrics such as the mechanical relevance or fabricability of the generated parts. Such optimisation is entirely possible given the parametrisation of the separating polylines drawn by the `Turtle` (simply a list of segment lengths and orientations) but has not been investigated so far. Yet, we may share some thoughts on the matter: referring to translational assemblies only, a Combescure transformation preserves the normals and, thus, the cinematics of (dis)assembly. It only involves the finding of a basis of the kernel of a matrix, which is a cheap calculation to perform. The linear space spanned by this basis encodes all geometrically valid designs in the vicinity of the solution found by our algorithm. A gradient descent on some predefined objective by the user, or a more open multi-criteria optimisation, can then be performed locally to find better designs. Performing this operation in parallel over multiple different designs found by our algorithm could generate many solutions that could be ranked according to the objectives defined by the user. Finally, an obvious research path shall be explored: the generation of 3D polyhedral assemblies.

Acknowledgements

. We are grateful to the anonymous reviewers for their thoroughness and helpful comments.

This work has been supported by the project DiXite. Initiated in 2018, DiXite (Digital Construction Site) is a project of the I-SITE FUTURE, a French initiative to answer the challenges of sustainable city.

References

[1] T. Bock, T. Linner, Advanced Construction and Building Technology, Cambridge University Press, 2015, ISBN: 978-1-107-07638-9, pp. 1–17.

doi:10.1017/CBO9781139924146.002.

[2] Déchets chiffres-clés, accessed: 2021-04-09 (2017).
URL https://www.ademe.fr/sites/default/files/assets/documents/dechets-chiffrescles-edition2020-3-010692.pdf

[3] Neutralité & bâtiment, accessed: 2021-04-09 (2016).
URL https://ile-de-france.ademe.fr/sites/default/files/neutralite-carbone-batiment.pdf

[4] M. Larsson, H. Yoshida, N. Umetani, T. Igarashi, Tsugite: Interactive design and fabrication of wood joints, in: Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, UIST '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 317–327. doi:10.1145/3379337.3415899.

[5] P. Song, C.-W. Fu, D. Cohen-Or, Recursive interlocking puzzles, ACM Transactions on Graphics 31 (6) (November 2012). doi:10.1145/2366145.2366147.

[6] J. Yao, D. M. Kaufman, Y. Gingold, M. Agrawala, Interactive design and stability analysis of decorative joinery for furniture, ACM Transactions on Graphics 36 (4) (March 2017). doi:10.1145/3072959.3054740.

[7] Z. Wang, P. Song, M. Pauly, Desia: A general framework for designing interlocking assemblies, ACM Transactions on Graphics 37 (6) (December 2018). doi:10.1145/3272127.3275034.

[8] P. Gilibert, R. Mesnil, O. Baverel, Rule-based generative design of translational and rotational interlocking assemblies (2021). doi:10.5281/zenodo.5158465.
URL https://doi.org/10.5281/zenodo.5158465

[9] R. Testuz, Y. Schwartzburg, M. Pauly, Automatic Generation of Constructable Brick Sculptures, in: M.-A. Otaduy, O. Sorkine (Eds.), Eurographics 2013 - Short Papers, The Eurographics Association, 2013. doi:10.2312/conf/EG2013/short/081-084.

[10] S. Xin, C.-F. Lai, C.-W. Fu, T.-T. Wong, Y. He, D. Cohen-Or, Making burr puzzles from 3d models, ACM Transactions on Graphics 30 (4) (July 2011). doi:10.1145/2010324.1964992.

[11] C.-W. Fu, P. Song, X. Yan, L. W. Yang, P. K. Jayaraman, D. Cohen-Or, Computational interlocking furniture assembly, ACM Transactions on Graphics 34 (4) (July 2015). doi:10.1145/2766892.

[12] L. Luo, I. Baran, S. Rusinkiewicz, W. Matusik, Chopper: Partitioning models into 3d-printable parts, ACM Transactions on Graphics 31 (6) (November 2012). doi:10.1145/2366145.2366148.

[13] Y.-L. Yang, J. Wang, N. J. Mitra, Reforming shapes for material-aware fabrication, in: Proceedings of the Eurographics Symposium on Geometry Processing, SGP '15, Eurographics Association, Goslar, DEU, 2015, pp. 53–64. doi:10.1111/cgf.12696.

[14] P. Song, Z. Fu, L. Liu, C.-W. Fu, Printing 3d objects with interlocking parts, Computer Aided Geometric Design 35-36 (2015) pp. 137–148. doi:10.1016/j.cagd.2015.03.020.

[15] P. Jimenez, Survey on assembly sequencing: A combinatorial and geometrical perspective, Journal of Intelligent Manufacturing 24 (2011) pp. 1–16. doi:10.1007/s10845-011-0578-5.

[16] A. Lambert, Disassembly sequencing: A survey, International Journal of Production Research 41 (2003) pp. 3721–3759. doi:10.1080/0020754031000120078.

[17] Z. Wang, P. Song, M. Pauly, State of the art on computational design of assemblies with rigid parts, Computer Graphics Forum 40 (2021) pp. 633–657. doi:10.1111/cgf.142660.

[18] R. H. Wilson, J.-C. Latombe, Geometric reasoning about mechanical assembly, Artificial Intelligence 71 (2) (1994) pp. 371–396. doi:10.1016/0004-3702(94)90048-5.

[19] B. Romney, C. Godard, M. Goldwasser, G. Ramkumar, An efficient system for geometric assembly sequence generation and evaluation, 1995, pp. 699–712. doi:10.1115/CIE1995-0800.

[20] M. Bagneris, R. Motro, B. Maurin, N. Pauli, Structural morphology issues in conceptual design of double curved systems, International Journal of Space Structures 23 (2008) pp. 79–87. doi:10.1260/026635108785260560.

[21] Optimizing structural building elements in metal by using additive manufacturing, accessed: 2021-12-07 (2015).
URL `https://www.ingentaconnect.com/content/iass/piass/2015/00002015/00000002/art00016`

[22] R. Oval, M. Rippmann, R. Mesnil, T. Mele, O. Baverel, P. Block, Feature-based topology finding of patterns for shell structures, Automation in Construction (February 2019). doi:10.1016/j.autcon.2019.02.008.

[23] P. A. Gagniuc, Markov Chains: From Theory to Implementation and Experimentation, John Wiley & Sons, Inc., ISBN: 978-1-119-38759-6 978-1-119-38755-8. doi:10.1002/9781119387596.

[24] Z. Wang, P. Song, F. Isvoranu, M. Pauly, Design and structural optimization of topological interlocking assemblies, ACM Transactions on Graphics 38 (6) (November 2019). doi:10.1145/3355089.3356489.

Appendices

A. Parts obeying a rotation

Let $\{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_k\}$ be the vertices of the polyline drawn by the `Turtle`. Let (l_i, θ_i) be the tuple defining segment $[\boldsymbol{x}_i, \boldsymbol{x}_{i+1}]$: $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{l}_i$ with vector $\boldsymbol{l}_i = l_i \begin{pmatrix} \cos\theta_i \\ \sin\theta_i \end{pmatrix}$. By linearity of EQUATION (2), EQUATION (3) can be rewritten as follows:

$$\text{EQUATION (3)} \Leftrightarrow \begin{cases} s \begin{pmatrix} \cos\theta_i \\ \sin\theta_i \end{pmatrix} \times \boldsymbol{m}_{x_i} \geq 0 \\ s \begin{pmatrix} \cos\theta_i \\ \sin\theta_i \end{pmatrix} \times \boldsymbol{m}_{x_{i+1}} \geq 0 \end{cases}$$

where we assume the index i to be such that $\boldsymbol{x} \in [\boldsymbol{x}_i, \boldsymbol{x}_{i+1}]$. Thus:

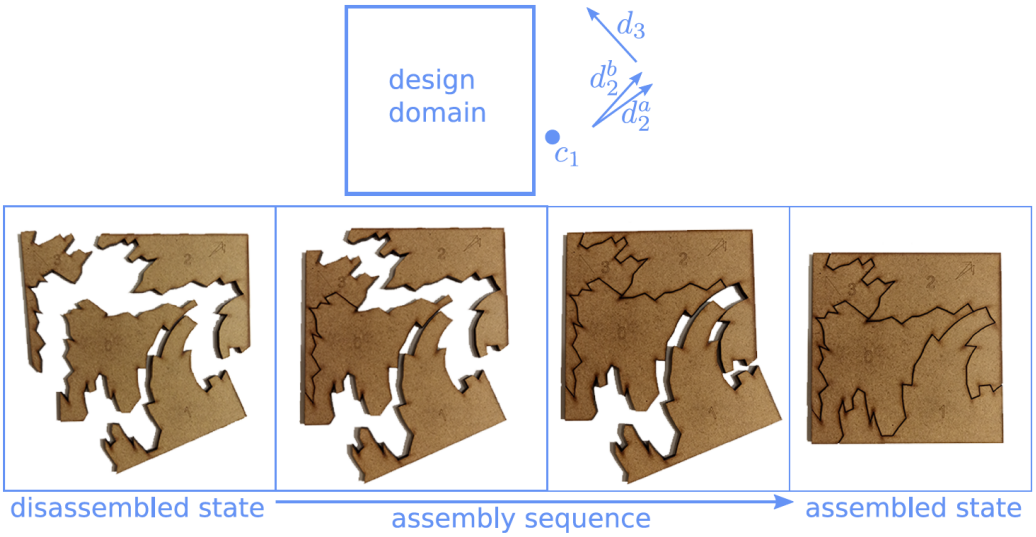
$$
\begin{aligned}
P_1 \text{ obeys } \boldsymbol{d} &\Leftrightarrow \forall\, 0 \le i < k \ \begin{cases} s(\boldsymbol{x}_{i+1} - \boldsymbol{x}_i) \times \boldsymbol{m}_{x_i} \ge 0 \\ s(\boldsymbol{x}_{i+1} - \boldsymbol{x}_i) \times \boldsymbol{m}_{x_{i+1}} \ge 0 \end{cases} \\
&\Leftrightarrow \forall\, 0 \le i < k \ \begin{cases} s\boldsymbol{l}_i \times \boldsymbol{m}_{x_i} \ge 0 \\ s\boldsymbol{l}_i \times \boldsymbol{m}_{x_{i+1}} \ge 0 \end{cases}
\end{aligned}
$$

where $\boldsymbol{m}_{x_{i+1}} = \boldsymbol{m}_{x_i} + l_i \begin{pmatrix} -\sin\theta_i \\ \cos\theta_i \end{pmatrix}$.

Graphical Abstract

Rule-based generative design of translational and rotational interlocking assemblies

Pierre Gilibert, Romain Mesnil, Olivier Baverel



Highlights

Rule-based generative design of translational and rotational interlocking assemblies

Pierre Gilibert, Romain Mesnil, Olivier Baverel

- Generality: our approach deals with the generation of 2D interlocking assemblies working in translations, rotations, and a combination of both. All translation directions or centres of rotation can be selected.
- Exhaustivity: any interlocked geometry can be generated.
- Injectivity: any feasible geometry can be produced by one sequence only.