



HAL
open science

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-04582391v1>

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

16 Rule-based generative design of translational and
17 rotational interlocking assemblies

18 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*

19 **Abstract**

20 Despite being a notoriously difficult task, efficient design of interlocking as-
21 semblies could greatly impact the construction sector and reduce its environ-
22 mental footprint by helping in the design of demountable buildings and the
23 reuse of structural members. While a growing research effort in this direc-
24 tion is being undertaken by the computer graphics and structural engineering
25 communities, most of the algorithms proposed so far imply restrictions on
26 assembly directions or prior knowledge on the joint's geometry. While rele-
27 vant, such tools do not fully explore the space of possible assemblies and
28 often fail to produce surprising results. Moreover, these designs are always
29 assembled through translational motions, and, to the best of our knowledge,
30 very little research has been conducted to address the challenges of designing
31 an assembly for rotation motions.

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

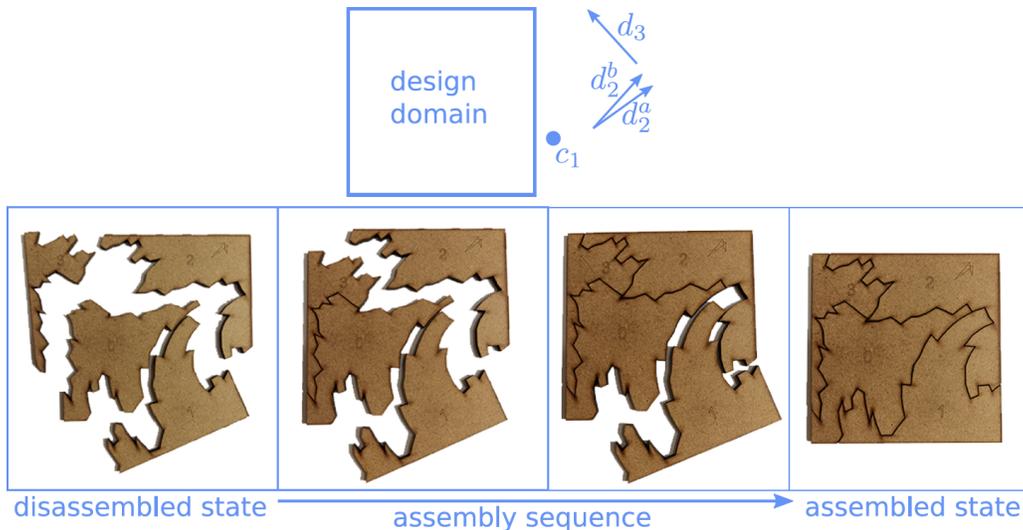


Figure 1: Top row is our tool’s input: the disassembling motions are given by the location of a centre of rotation, a cone of translation, and a single direction of translation. Bottom row: the fabricated raw output and an assembly sequence.

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

41 1. Introduction

42 Even the simplest buildings are constituted of tens of thousands of compo-
 43 nents which need to be assembled *in-situ* or in factory [1]. In civil engineering,
 44 an assembly may be defined as a collection of parts that are connected to
 45 each other through various kinds of means: nails, glue, etc. Despite their
 46 mechanical relevance, a major inconvenient with such standard connectors is
 47 that they are irreversible: at the end of the lifetime of a building, assemblies
 48 are often destroyed and prevent the reuse of structural components which has
 49 a significant impact on the environment: according to the French government
 50 [2] the construction sector generated almost 70% of the waste in France in
 51 2017. Conversely, new construction materials have a serious environmental
 52 impact, up to 7% of the emission of greenhouse gas in France in 2016 [3].
 53 This paper represents a humble step towards the necessary decarbonisation
 54 and waste reduction of the construction industry by studying the genera-
 55 tion of reversible interlocking assemblies, i.e. assemblies that are tightly

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

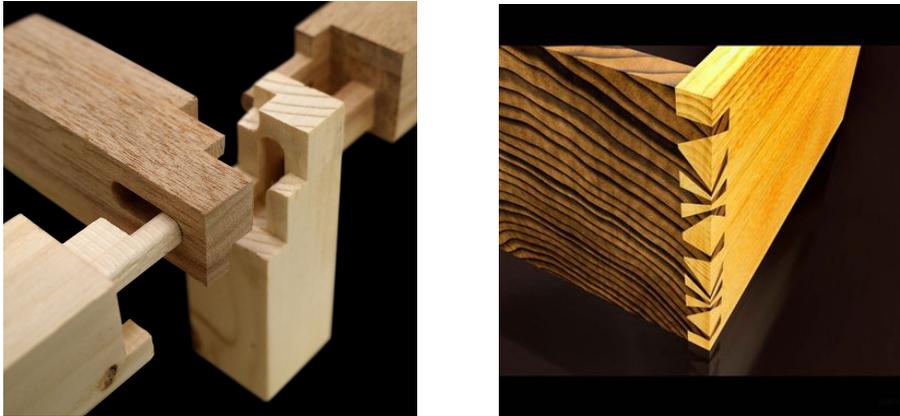


Figure 2: Examples of 3D sequential assemblies. Left: a CNC-milled assembly (source [4]). Right: a traditional Nejiri Arigata assembly (source Internet).

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

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

- 84 • Generality: our approach deals with translations, rotations, and a com-
 85 bination of both. All translation directions or centres of rotation can
 86 be selected.
- 87 • Exhaustivity: any interlocked geometry can be generated.
- 88 • Injectivity: any feasible geometry can be produced by one sequence
 89 only.

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

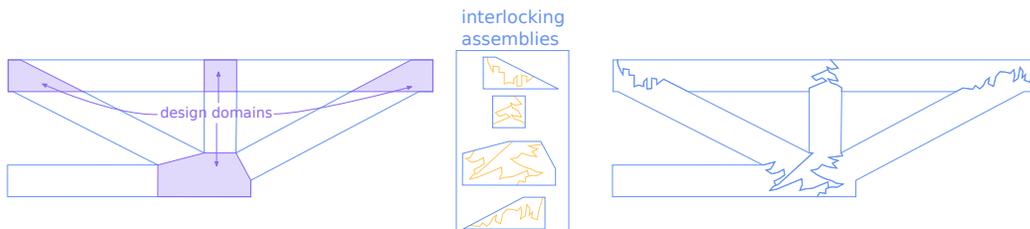


Figure 3: The aim of this study is to automatically generate a 2D interlocking assembly in a possibly non-convex design domain resulting from the intersection of several structural members. Here the algorithm presented in this paper was ran independently on each of the four design domains.

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

99 **2. Related work**

100 *2.1. Designing interlocking assemblies*

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

103 *Catalogue-based designs:*

104 . Significant work has been made to automatically generate assemblies made
105 of parts chosen in a catalogue: [9] builds hollow 3D shape using a set of Lego
106 bricks; [10] explores the partitioning of 3D shapes into a set of 6-parts burr
107 puzzles; Fu and coauthors [11] developed a method aimed at creating global
108 interlocking furniture assembly from a model of orthogonally intersecting 3D
109 shapes and generated the joints from a lookup table. In a different spirit,
110 [12] partitions a 3D shape into printable parts and assembles them through
111 mortise and tenon kind of joints and [13] presents a material-aware algorithm
112 to modify the overall shape of a structure but connects parts with mortise
113 and tenon joints.

114 *Voxel-based designs:*

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

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

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

134 human user, fabrication and mechanical relevance.

135

136 2.2. Disassembly planning

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

151

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

159

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

171 interlocking assembly strongly depends on the interaction of the five axes of
172 design proposed by [20], namely form, force, structure, material, and technol-
173 ogy. For instance, Larsson *et al.* in [4], while delivering stunning results, had
174 to assume an assembly (structure) made of wood (material), carrying most
175 probably bending moments (force), milled with a 3-axis CNC machine (tech-
176 nology), with a grid of voxel as a design space (form), as well as additional
177 assumptions such as a cube as a design domain and a single axis of assembly.
178 Any change in those premises, for instance switching to a 5-axis CNC ma-
179 chine, greatly impacts the space of solutions and requires another algorithm
180 to search it. More generally a good approach to designing assemblies, shown
181 in FIGURE 5, would be through a multi-criteria optimisation where several
182 designs are proposed to the designer who makes the final choice as to orient
183 her work. These criteria (choice of material, technology, etc) are problem-
184 dependent and could therefore be implemented *a posteriori* to curate the
185 space of solutions, once the user knows how to navigate and explore the field
186 of possible assemblies. As an example, for timber assemblies, the milling
187 technology (3 to 5 axes, size of milling tools) and mechanical performance
188 (governed by the strong anisotropy of wood) are obvious practical constraints
189 that will dictate the performance of the assembly and, thus, the subset of
190 suitable assembly shapes. Other technologies and material, like 3d-printed
191 steel nodes [21], would come with different sets of feasibility constraints which
192 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]).

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

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

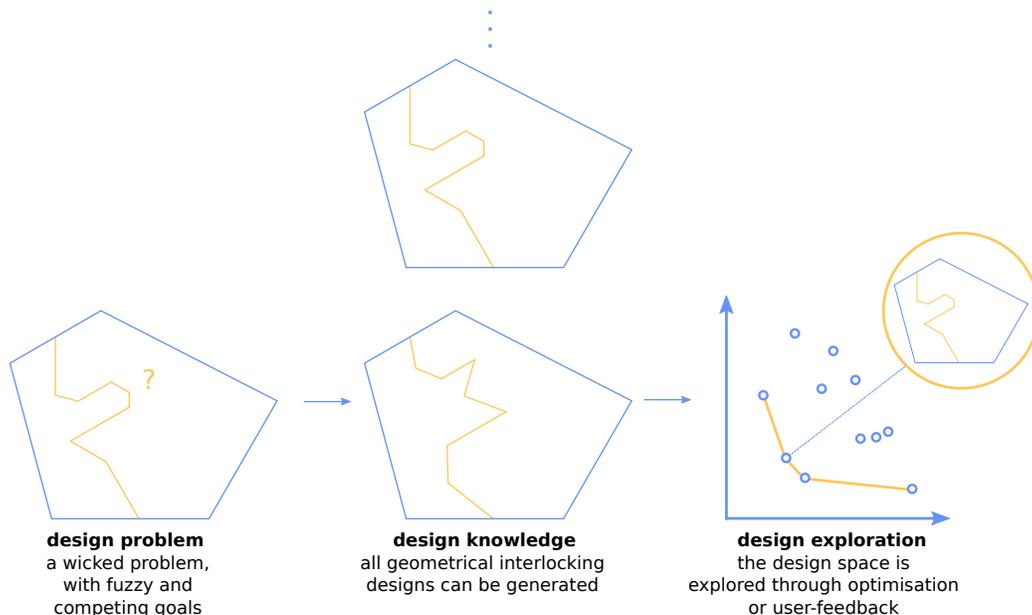


Figure 5: The focus of this paper is on the *design knowledge* step of the process to design interlocking assemblies: we shall generate all possible geometrical interlocking design.

204 3. Methodology

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

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

222 3.1. Blocking relationships in assemblies

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

227 3.1.1. Cone of translational freedom

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

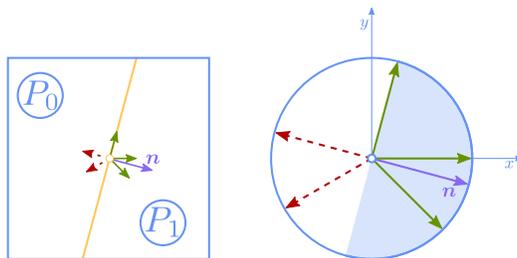


Figure 6: On the left an assembly $A = \{P_0, P_1\}$. On the right, the hemisphere in a darker shade of blue represents the half-space of motion of P_1 .

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

238 with P_0 . The red dashed ones depict invalid directions of translation as
 239 moving P_1 along them will lead the two parts to intersect. The set of all
 240 valid translations constitutes a so-called *half-space of motion*. On FIGURE
 241 6, let \mathbf{n} be the unit normal vector of the yellow line segment separating the
 242 two parts oriented from P_0 to P_1 (the purple arrow). Then we can state that
 243 P_1 may obey a direction $\mathbf{d} \in \mathbb{R}^2$ if and only if $\mathbf{n} \cdot \mathbf{d} \geq 0$. A somewhat more
 244 complex result can be obtained by analysing a dividing polyline made of two
 245 line segments:

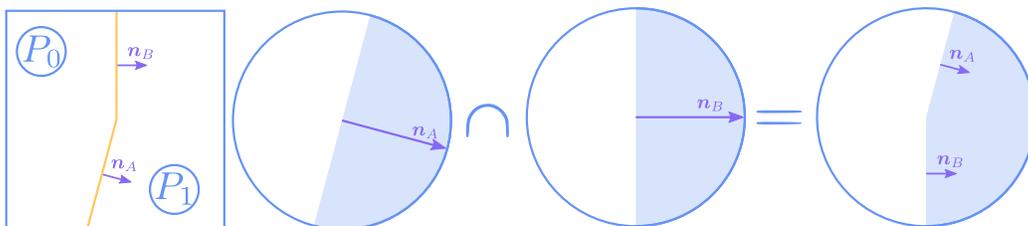


Figure 7: On the left an assembly $A = \{P_0, P_1\}$. On the right, the cone of translational freedom results from the intersection of the half-space of motion of each yellow line segment.

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

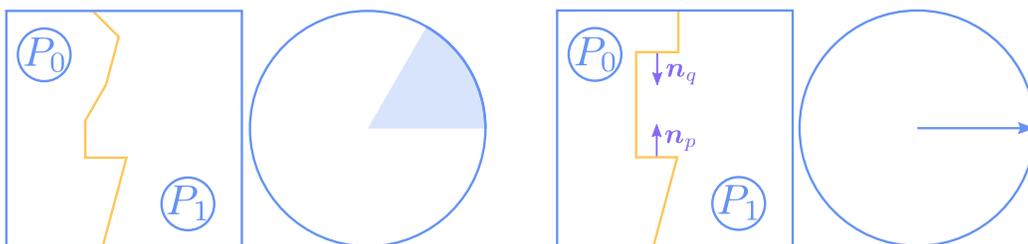


Figure 8: By intersecting half-spaces of motion, we can define the cone of translational freedom of P_1 . Note that this cone may be reduced to a single direction as highlighted in the example on the right.

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

the cone of translational freedom is a solution to the system

$$\begin{cases} \mathbf{n}_1 \cdot \mathbf{d} \geq 0 \\ \vdots \\ \mathbf{n}_k \cdot \mathbf{d} \geq 0 \end{cases} \quad (1)$$

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

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

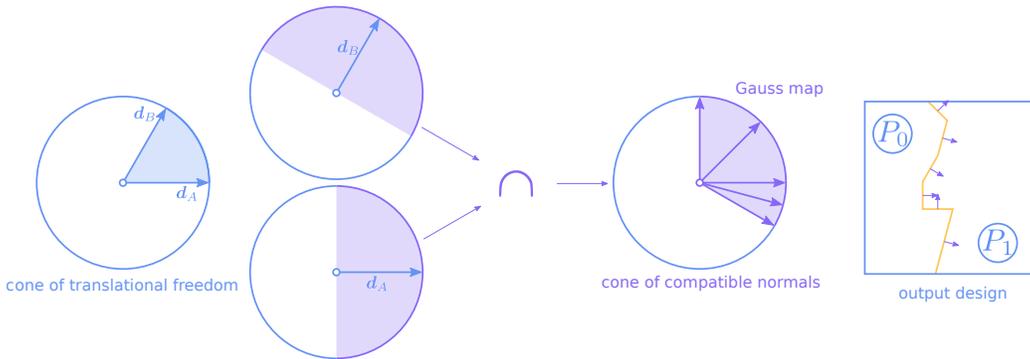


Figure 9: The desired cone of translational freedom is bounded by two vectors which define two semi-circles whose intersection becomes the cone of compatible normals. The Gauss map of the dividing curve, represented here using purple arrows, must lie in the latter.

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

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

270 *3.1.2. Directional Blocking Graph - DBG*

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

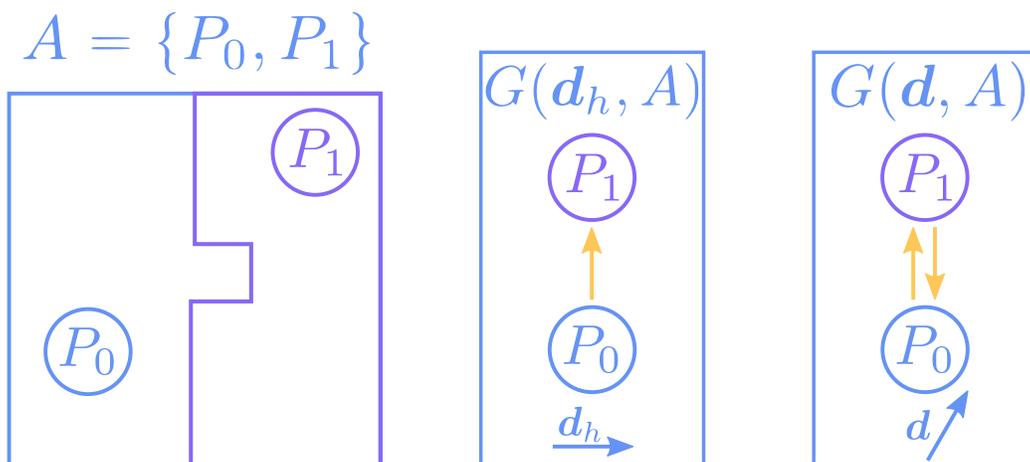


Figure 10: An assembly A made of two parts P_0 and P_1 and two DBGs.

284 A useful property of a DBG $G(\mathbf{d}, A)$ is that we can easily deduce whether
 285 the assembly A is interlocked for \mathbf{d} by checking the strong-connectedness¹ of

¹Informally 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) .

286 the graph: if the graph is strongly connected then every part is blocked by
287 another. If not, then it can be decomposed into strongly connected compo-
288 nents (possibly reduced to a single vertex) where at least one component can
289 obey \mathbf{d} .

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

301 On the bottom row of FIGURE 11 part P_1 has been removed and the as-
302 sembly is now made of two parts $\{P_0, P_2\}$. The DBG associated to the
303 horizontal direction of motion is strongly connected and the one associated
304 to the upwards translation has two strongly connected components, namely
305 $\{P_0\}$ (blue) and $\{P_2\}$ (purple). Moreover, since no edge starts from P_2 the
306 latter is not blocked by P_0 for that upwards direction \mathbf{d} : it can obey such \mathbf{d} .

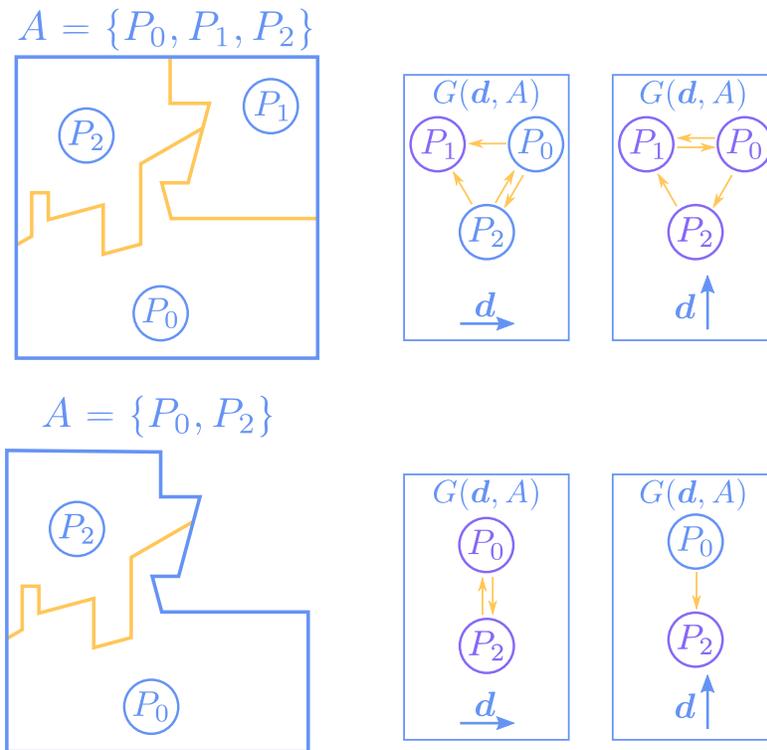


Figure 11: The strongly connected components of each DBG are colour-coded (blue and purple). Top row: an assembly made of 3 parts and the DBGs associated to the horizontal to the right and vertical upwards directions of translation. Bottom row: assembly obtained after removing P_1 and the corresponding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

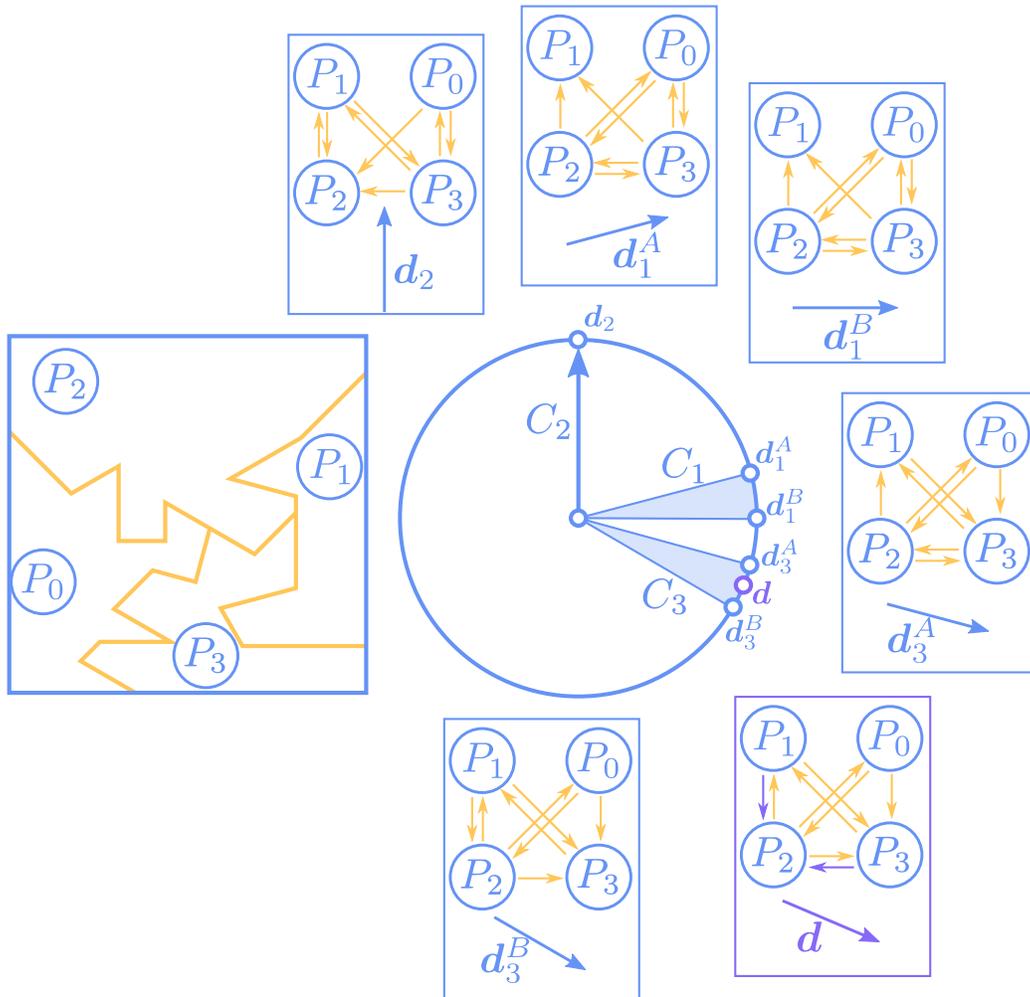


Figure 12: On the left, an assembly A made of 4 parts and a representation of its NDBG (restricted to translation). For $i \in \{1, 2, 3\}$ part P_i obeys cone C_i (C_2 being reduced to a single direction). The DBGs associated with several directions \mathbf{d}_j (seen as points on the unit circle) are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

308 A Non Directional Blocking Graph (NDBG) is simply the concatenation
 309 of all DBGs $G(\mathbf{d}, A)$ for all possible motions \mathbf{d} . The NDBG of A can be
 310 represented on the 2D unit circle, seen as the locus of all directions of trans-
 311 lation in 2D.

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

313 obeys the two directions \mathbf{d}_p and \mathbf{d}_q bounding a cone of translational freedom
 314 C_i then [18] showed that for any \mathbf{d} in the interior of C_i $G(\mathbf{d}, A)$ is obtained
 315 by performing a union operation of the two DBGs $G(\mathbf{d}_p, A)$ and $G(\mathbf{d}_q, A)$.
 316 In other words one only needs to compute the DBGs of the boundaries of a
 317 cone to know every possible DBG in that cone.

318 For instance, on FIGURE 12 cone C_3 is bounded by directions \mathbf{d}_3^A and \mathbf{d}_3^B
 319 and the DBG of the purple point \mathbf{d} in the open arc $] \mathbf{d}_3^A, \mathbf{d}_3^B [$ is the union of
 320 the DBGs associated to the bounding directions \mathbf{d}_3^A and \mathbf{d}_3^B . To see this one
 321 can notice that edge $e_{1 \rightarrow 2}$ (resp. $e_{3 \rightarrow 2}$) is present (resp. absent) in $G(\mathbf{d}_3^B, A)$
 322 and absent (resp. present) in $G(\mathbf{d}_3^A, A)$ but both are present in $G(\mathbf{d}, A)$ (the
 323 corresponding arrows are highlighted in purple).

324 Moreover, as stated by [7], half of the circle is redundant as $G(-\mathbf{d}, A)$ can
 325 be obtained by reversing the direction of the edges in $G(\mathbf{d}, A)$. In that sense
 326 we can fully assess the NDBG, i.e., the blocking relations of a sequential
 327 assembly, by studying the strong connectedness of a finite number of DBGs
 328 $G(\mathbf{d}, A)$ obtained for the endpoints \mathbf{d} of the different user-prescribed cones
 329 of translational freedom C_i . Following [7] we call such DBGs the *base DBGs*
 330 of A .

331 3.2. Turtle graphics and Markov process

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

341
 342 In probability theory, a **Markov process**, or Markov chain, ([23]) is a
 343 stochastic model describing a sequence of possible events in which the prob-
 344 ability of each event depends only on the state attained in the previous event
 345 (it is *memoryless*). A state is said absorbing if once entered the probability
 346 to leave it is 0. More specifically a discrete-time Markov process is a Markov
 347 chain with a countable number of states and the chain iteratively transitions
 348 between states at discrete time steps according to some probabilistic rules.

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

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

363 0. `start` \mapsto `rotate`

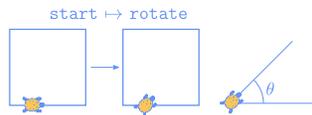


Figure 13: The `Turtle` is randomly initialised on the boundary of the design domain and this order simply tells it to choose a random orientation parameterised by angle θ .

364 1. `rotate` \mapsto `walk`

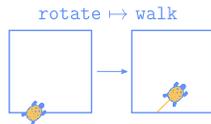


Figure 14: The `Turtle` has already chosen an orientation and must now walk forwards by a random amount.

365 2. `rotate` \mapsto `end`

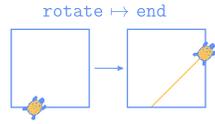


Figure 15: The **Turtle** has already chosen an orientation and walks forward until meeting an edge of the design domain.

366 3. walk ↦ rotate

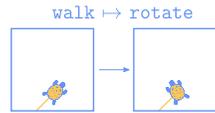


Figure 16: The **Turtle** has walked to a new position and must now choose a new random orientation.

367 4. walk ↦ snap

368 The **snap** order ensures that the assembly is compatible with
 369 the prescribed disassembling motions by orienting the **Turtle**
 370 collinearly with the bounds \mathbf{d}^A and \mathbf{d}^B . Details are provided SEC-
 371 TION 4.

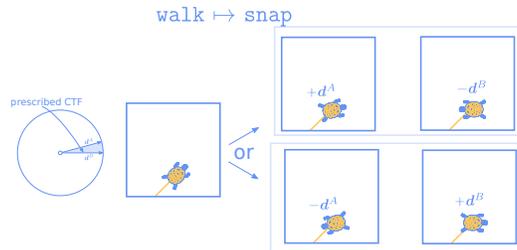


Figure 17: Context: the leftmost blue cone represents the cone of translational freedom (CTF) prescribed by the user: the final design must obey any direction in the cone bounded by \mathbf{d}^A and \mathbf{d}^B . The **Turtle** has walked to a new position and must now choose to orient along either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

372 5. snap ↦ walk

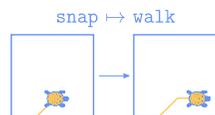


Figure 18: Similar to **rotate ↦ walk**: the **Turtle** has snapped to an orientation and walks forward by a random amount.

373

6. snap \mapsto end

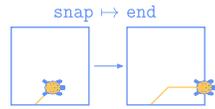


Figure 19: Similar to `rotate \mapsto end`: the Turtle has snapped to an orientation and walks forward until meeting an edge of the design domain.

374

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

375

376

377

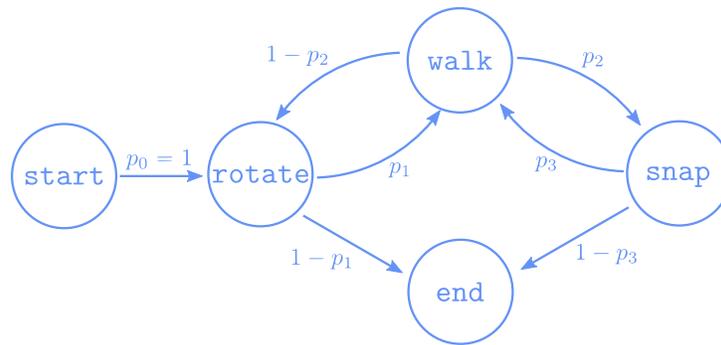


Figure 20: \mathcal{M} transitions between states with some predefined probabilities p_i .

378

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

384

385

Since a sequence of the form `[rotate or snap] \mapsto 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 \mapsto rotate` as the final orientation could very well be obtained after a single `rotate` order. Similarly a `walk \mapsto walk` can be obtained with the sequence `walk \mapsto rotate \mapsto walk` where the Turtle

386

387

388

389

390

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

394 4. Algorithm and results

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

403 4.1. Parts obeying a translation

404 4.1.1. On the creation of a single part

405 The user decides on two vectors of translation \mathbf{d}_1^A and \mathbf{d}_1^B bounding the
 406 cone of translational freedom of the would-be part P_1 . Enforcing $\mathbf{d}_1^A = \mathbf{d}_1^B$
 407 leads to the special case where the cone is reduced to a single vector that
 408 is to say where P_1 must translate along one direction only. Then a Markov
 409 chain \mathcal{M} emits the `start` order which initialises a `Turtle` randomly on the
 410 boundary of the design domain. In subsequent iterations \mathcal{M} tells the `Turtle`
 411 whether to orient itself or to move. To comply with a `walk` order, the length
 412 l by which the `Turtle` moves is randomly picked in a user-defined interval
 413 $[l_{\min}, l_{\max}]$ (to have consistent step size, or edge length). To obey a `rotate`
 414 order, the rotation angle θ is randomly chosen in an interval such that the
 415 normal vector \mathbf{n} of the line segment is such that $\mathbf{n} \cdot \mathbf{d}_1^A \geq 0$ and $\mathbf{n} \cdot \mathbf{d}_1^B \geq 0$.
 416 The reader’s attention is drawn to the fact that this constraint on θ is enough
 417 to prevent the `Turtle` from crossing its own path.

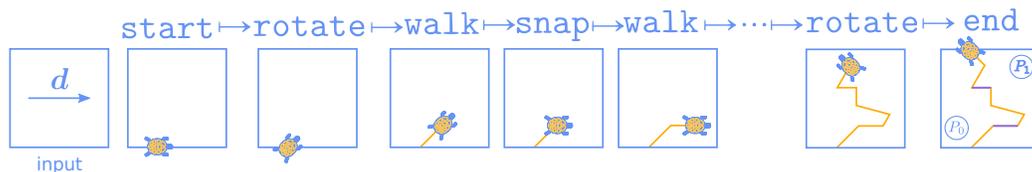


Figure 21: A step-by-step decomposition of the `Turtle`’s trajectory. Highlighted in purple on the far right are the line segments corresponding to a `snap` order.

418 If the sequence of orders and random values leads the **Turtle** to wander
 419 outside of the design domain, a backtracking procedure is executed to replace
 420 it inside. At the end of an iteration, \mathcal{M} randomly applies a production rule
 421 on the latest order it gave to get the one for the next iteration. Finally, when
 422 \mathcal{M} emits the **end** order, the **Turtle** walks until meeting an edge of the design
 423 domain.
 424 Obviously, letting the **Turtle** move like this is likely to yield very poor results
 425 as the random separating polyline obtained at the end will be such that the
 426 actual cone of translational freedom of P_1 strictly includes the user-prescribed
 427 cone bounded by \mathbf{d}_1^A and \mathbf{d}_1^B . As an extreme example imagine that the user
 428 specified a vector $\mathbf{d}_1^A = \mathbf{d}_1^B \equiv \mathbf{d}_1 = (1, 0)^T$ aligned with the x axis (meaning
 429 that the user wants P_1 to translate along the horizontal axis only) and the
 430 **Turtle** drew a single line segment, as illustrated on the top row of FIGURE
 431 22. Even though P_1 does obey \mathbf{d}_1 it also obeys a full half-space of motion
 432 which is an undesirable behaviour.

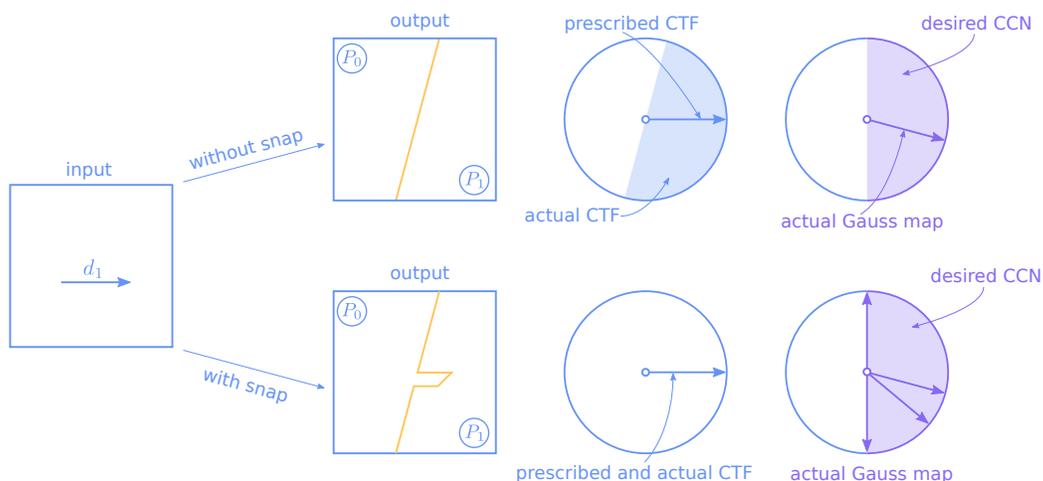


Figure 22: CTF and CCN respectively stand for *Cone of translational freedom* and *Cone of compatible normals*. The Gauss map of the dividing polyline is depicted using purple arrows on the rightmost circles.

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

439 $\mathbf{n} \cdot \mathbf{d}_1^A = 0$ and $\mathbf{n} \cdot \mathbf{d}_1^B \geq 0$ or *vice-versa* by switching the superscripts A and
 440 B . As such, the `Turtle` draws a valid polyline (i.e., such that the cone of
 441 translational freedom of P_1 exactly matches the user-prescribed cone) if and
 442 only if it snapped at least twice, once along $\pm \mathbf{d}_1^A$ and once along $\mp \mathbf{d}_1^B$, which
 443 gives a computationally light manner to check whether a polyline is valid.
 444 On FIGURE 22, bottom row, one `snap` order ensures that the Gauss map of
 445 the polyline includes a normal vector bounding the cone of compatible nor-
 446 mals. Snapping at least twice in opposite directions ensures that two normal
 447 vectors will match the bounds of the cone of compatible normals (vertical
 448 upwards and downwards purple arrows on the right).
 449

450 4.1.2. Surjectivity of the Markov process \mathcal{M}

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

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

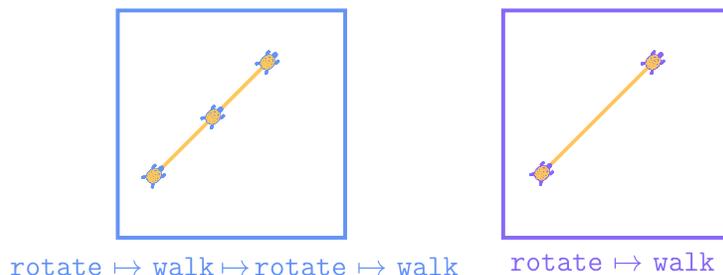


Figure 23: The same line segment can be reached by walking twice in the same direction by a magnitude $\frac{l}{2}$ or once by a magnitude l , for some $l \in [l_{\min}, l_{\max}]$, and thus the map is not injective.

472 Note that this mapping can be made injective by reducing a sequence
 473 emitted by \mathcal{M} to the smallest possible word by tracking the times where the
 474 **Turtle** rotated by 0 rad (or snapped consecutively) and replacing instructions
 475 “rotate(θ_i) \mapsto walk(l_i) \mapsto rotate(0) \mapsto walk(l_{i+1})” with “rotate(θ_i) \mapsto
 476 walk($l_i + l_{i+1}$)”.

477 4.1.3. On the creation of an interlocked assembly

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

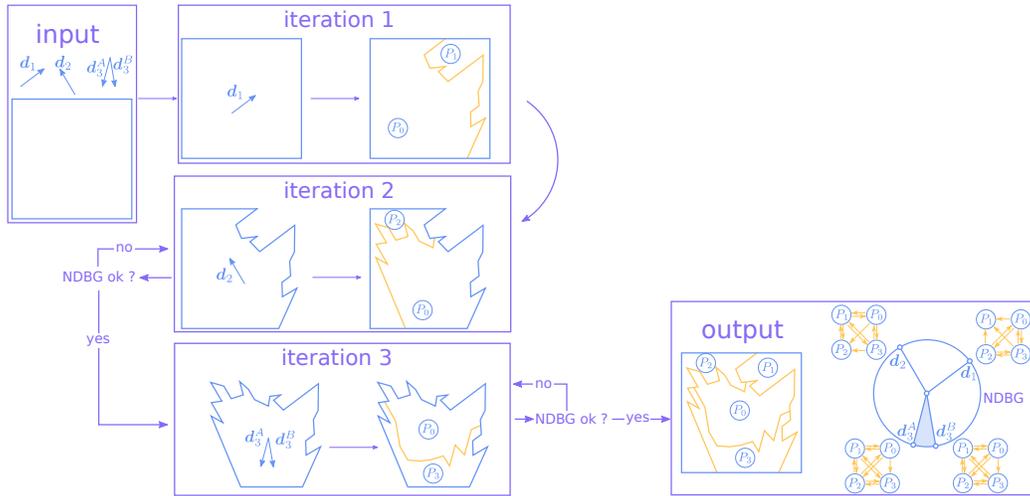


Figure 24: Workflow of the generation of a translational assembly made of 3+1 parts

487 A necessary condition for a part P_i to be blocked by another part P_j ,
 488 $0 < j < i$, is for the two of them to share a boundary. To ensure contact
 489 and thus increase the odds of the DBGs being strongly connected we bias
 490 the **Turtle** to always choose its starting point on a boundary shared by an
 491 other P_j , $0 < j < i$.

492 The reader's attention is drawn to the fact that the user cannot prescribe
 493 two cones of translational freedom related to two successive parts, say P_i
 494 and P_{i+1} , to intersect: indeed it would mean that there exists a direction in
 495 the intersection of the cones such that P_i and P_{i+1} can simultaneously obey
 496 it and in practice, one part could push the other out of the way. However
 497 the user can specify intersecting cones related to non-successive parts, say P_i
 498 and P_j with $|i - j| > 1$, and leave it to the **Turtle** to create an interlocked
 499 assembly.

500

501 Figure 25 presents four assemblies obeying translations generated by our
 502 method where the leftmost column depicts the user-prescribed cone of trans-
 503 lational freedom.

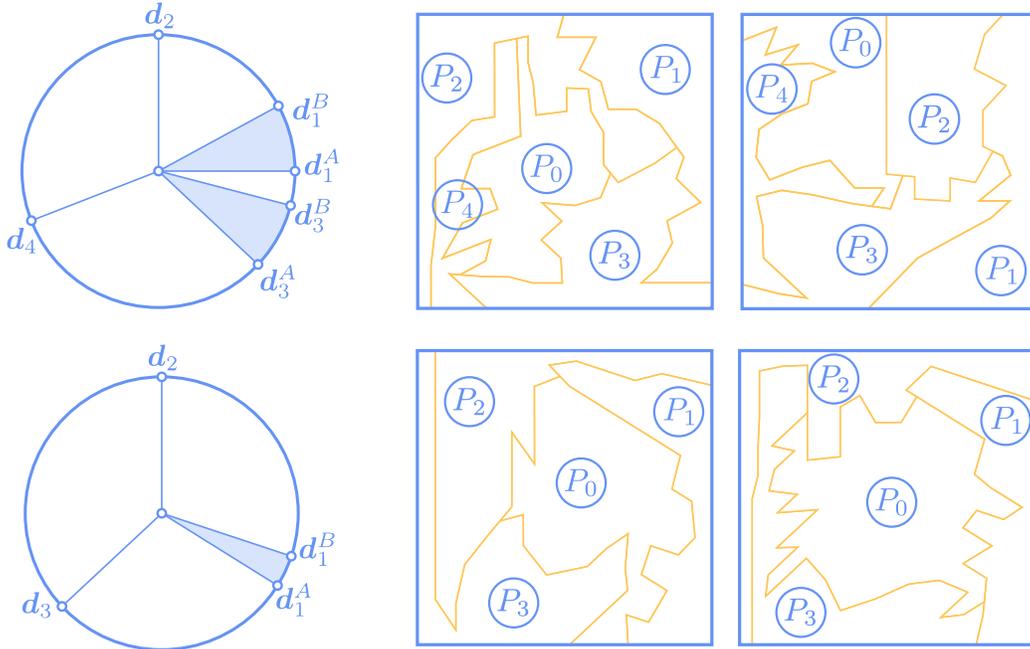


Figure 25: Assemblies working purely in translation. $N = 4$ on the top row and $N = 3$ on the bottom row.

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

506 4.2. Parts obeying a rotation

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

512

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

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

513 EQUATION (2) states that an infinitesimal rotation of \mathbf{x} around \mathbf{d} is the same
 514 as an infinitesimal translation of \mathbf{x} along the vector $\mathbf{m}_x = \begin{pmatrix} y_d - y \\ x - x_d \end{pmatrix}$. We
 515 call \mathbf{m}_x the *instantaneous direction of motion* of point \mathbf{x} . Note that this
 516 vector is orthogonal to the line going through \mathbf{x} and \mathbf{d} .

517 A necessary and sufficient condition for the part P_1 of an assembly $A =$
 518 $\{P_0, P_1\}$ to obey a rotation around \mathbf{d} is to have the instantaneous directions
 519 of motion of all points \mathbf{x} on the boundary of P_1 **not** pointing towards the
 520 interior of P_0 . Indeed if there is one point \mathbf{x} on the boundary of P_1 such that
 521 \mathbf{m}_x points towards the interior of P_0 then an infinitesimal rotation around \mathbf{d}
 522 will send that point to collide with P_0 which exactly means that P_1 does not
 523 obey \mathbf{d} . Moreover, since we focus on infinitesimal motions, we only need to
 524 study the points on the boundary of both P_0 and P_1 . In other words the fact
 525 that P_1 obeys \mathbf{d} depends only on the geometry of the polyline separating P_0
 526 from P_1 .

527

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 \mathbf{m}_x , of each point \mathbf{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 \mathbf{x} on the **Turtle**'s path we denote by θ the orientation of the
 line segment of the polyline on which \mathbf{x} is and the condition becomes:

$$s \begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix} \times \mathbf{m}_x \geq 0 \quad (3)$$

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 \mathbf{m}_x shall be pointing to. After a few calculations, developed
 in appendix A, one derives the formula:

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

528 where $\mathbf{m}_{x_{i+1}} = \mathbf{m}_{x_i} + l_i \begin{pmatrix} -\sin \theta_i \\ \cos \theta_i \end{pmatrix}$ and $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{l}_i$, with vector $\mathbf{l}_i =$
 529 $l_i \begin{pmatrix} \cos \theta_i \\ \sin \theta_i \end{pmatrix}$. Generally speaking SYSTEM (4) admits an angular interval as a
 530 solution.

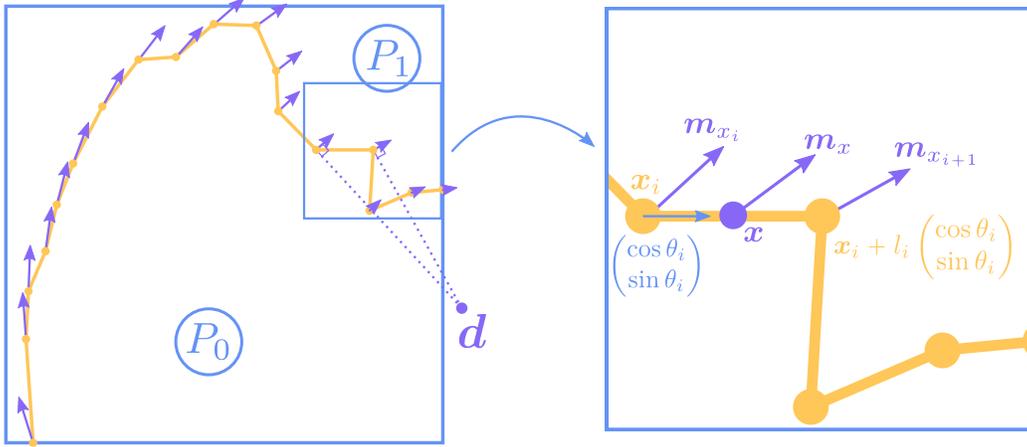


Figure 26: The purple arrows represents the instantaneous directions of motions of each yellow point. A zoom on the right shows that the cross product between $\begin{pmatrix} \cos \theta_i \\ \sin \theta_i \end{pmatrix}$ and vectors \mathbf{m}_{x_i} , $\mathbf{m}_{x_{i+1}}$ and \mathbf{m}_x are all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

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

540

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



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

545 As the `Turtle` walks it is possible to define the on-going cone of transla-
 546 tional freedom of the polyline under construction. As the current part shall
 547 rotate but not translate the algorithm keeps computing what angle θ would
 548 close such cone of freedom (meaning it would find θ such that the normal vec-
 549 tor \mathbf{n} of the current line segment would not be compatible with SYSTEM (1)
 550 for any direction of translation). If such θ happens to be solution to SYSTEM
 551 (4), then it is greedily chosen to orient the `Turtle`, which ensures that the
 552 newly created part will obey its rotation but will not obey any translation.
 553 The parts depicted on FIGURE 27 are obtained through this procedure. A
 554 demonstration of the (dis)assembling motions as well as the interlocking as-
 555 pect of the design presented on FIGURE 27 is available at 0:06 in our video [8].
 556

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

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

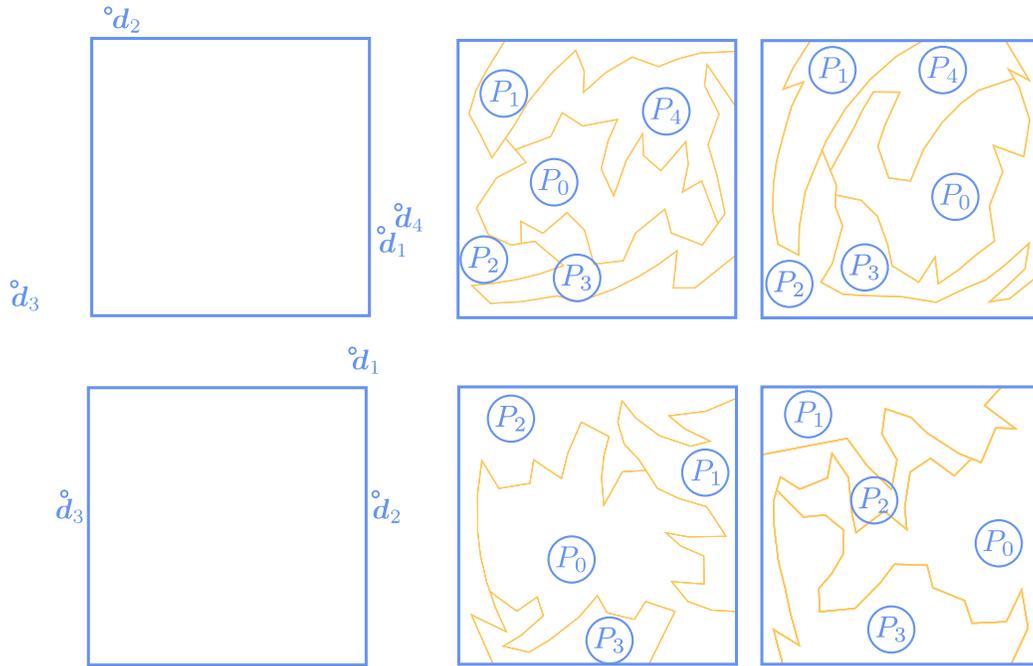


Figure 28: Assemblies working purely in rotation.

569 *4.3. Parts obeying both a translation and a rotation*

570 Building on top of the two previous sections it is quite easy to adapt our
 571 algorithm to design parts obeying both translations and rotations.

572 Say the user wants P_i to obey simultaneously a translation along any vectors
 573 in the cone bounded by the vectors \mathbf{d}_i^A and \mathbf{d}_i^B and a rotation around the
 574 centre \mathbf{d}_i^C . To succeed, one simply needs to intersect the angular interval
 575 defined for the translation cone and the angular interval defined by SYSTEM
 576 (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 \mathbf{d}_1^r ; P_2 obeys both
 a translation along \mathbf{d}_2^t and a rotation around \mathbf{d}_2^r ; P_3 obeys a translation along \mathbf{d}_3^t

577 In such case the Markov chain \mathcal{M} distinguishes between a **snapT** (trans-
578 lation) and a **snapR** (rotation) and the rest of the algorithm stays the same.
579 Note that special care must be taken by the user to define \mathbf{d}_i^A , \mathbf{d}_i^B and \mathbf{d}_i^C
580 to be compatible with each other: one cannot ask for a rotation around \mathbf{d}_i^C
581 that would lift P_i up and at the same time define \mathbf{d}_i^A , \mathbf{d}_i^B to be pointing
582 downwards. FIGURES 1 and 29 present laser-cut puzzles generated by our
583 **Turtle** where each part obeys a rotation and/or a translation. On FIGURE
584 30 part P_1 obeys both a rotation (\mathbf{d}_1^r) and translation (\mathbf{d}_1^t) while parts P_2
585 and P_3 must rotate (\mathbf{d}_2^r and \mathbf{d}_3^r).

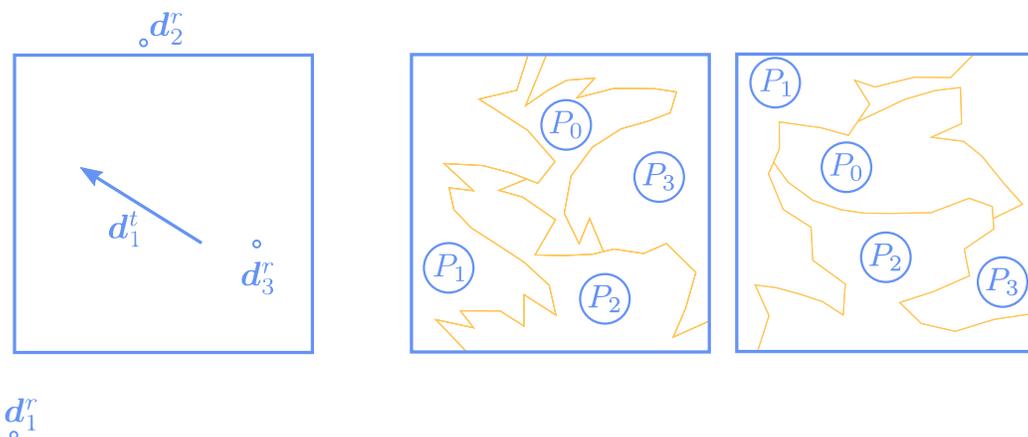


Figure 30: Assemblies working both in rotation and in translation.

586 5. Computation time

587 By letting the algorithm run the way it has been described so far, the
588 greater the number of parts N the less likely is it that a solution assembly
589 will be found. In fact, the algorithm may fail to draw a polyline (as soon as
590 $N \geq 2$). Indeed, referring to FIGURE 24, the design domain accessible to the
591 **Turtle** when it is tasked to draw the next part P_{i+1} is what was the remaining
592 part, P_0 , at the previous iteration. Thus the bigger are the previous parts P_j ,
593 $0 < j < i$, the smaller is the design domain at iteration $i + 1$, and if it is too
594 small there is simply not enough space to successfully create the next part
595 given the step resolution interval $[l_{\min}, l_{\max}]$. Consequently, the greater N the
596 more the more difficult it is to generate successive parts and the completion
597 time may become infinite.

598 As a consequence, two measures have been implemented:

- 599 • To ensure a design domain large enough at all iterations, a part is
600 discarded if its area is greater than $\frac{A}{N+1}$ (A being the area of the initial
601 design domain, the square in our examples) regardless of the NDBG,
602 and has to be drawn again. This constraint ensures that all parts are
603 of similar areas and that the remaining part P_0 is relatively big at all
604 iterations.
- 605 • If the creation of a part P_{i+1} fails $M \in \mathbb{N}^*$ successive times (i.e. the re-
606 lated DBGs are not strongly-connected and/or its area is above the
607 threshold) then it is assumed that part P_i , even though valid, has
608 strange geometrical features making it burdensome for the `Turtle` to
609 succeed. Consequently, the algorithm backtracks: P_i is deleted and
610 must be generated again.

611 These two measures ensure that a valid interlocked assembly will eventu-
612 ally be found but give no certainty on the completion time (empirically, in
613 translation, it seems to be in $\mathcal{O}(N^4)$). To understand it we need to quickly
614 prove some results: for $0 < k \leq N$, at iteration k , let A_k be the area of
615 the design domain in which part P_k shall be drawn (for instance $A_1 = A$,
616 the area of the initial design domain). At this k^{th} step $k - 1$ parts P_j ,
617 $0 < j < k$, already exist, each with an area smaller than $\frac{A}{N+1}$. As such
618 $A_k \geq A - \sum_{j=1}^{k-1} \frac{A}{N+1} = \frac{A(N+2-k)}{N+1}$. Thus the probability \mathbb{P}^k to draw a part P_k
619 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
620 a part P_k to be valid all the relevant DBGs in the NDBG must be strongly-
621 connected, which is not a given, i.e. the probability that P_k meet this con-
622 straint is strictly less than 1 (it is not the case for P_1 as there is no NDBG
623 to compute). Thus for $k > 1$ the probability that a part P_k meets both con-
624 straints (area below the threshold and strongly-connected DBGs) is strictly
625 less than $\frac{1}{N+2-k}$ (and is exactly $\frac{1}{N+1}$ when $k = 1$).

626 In practice, since P_1 only needs to meet the area constraint (probability $\frac{1}{N+1}$),
627 it happens to be quite fast to generate. The crux of the issue lies with part
628 P_2 : it has to meet both constraints and the probability to get a valid part
629 is strictly less than $\frac{1}{N}$ (and empirically, it seems to be much lower). Once
630 such P_2 is obtained, it becomes easier and easier to draw the subsequent
631 parts and in particular P_N is also quite fast to generate. However, see the
632 second bullet point, if the algorithm fails M times to draw a part P_{i+1} then
633 P_i is deleted: when that happens for part P_3 and P_2 (which is quite likely
634 for a sufficiently large N , see FIGURE 31), P_2 has to be redrawn from scratch

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

# parts	Translation			Rotation		
	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.

642 TABLE 1 shows the minimum, mean, and maximum times needed by the
 643 algorithm to generate an assembly made of $N + 1$ parts (for $N \in \llbracket 1, 6 \rrbracket$)
 644 working either purely in translation or purely in rotation in a square design
 645 domain with $[l_{\min}, l_{\max}] = [0.05L, 0.1L]$, where L is the length of the diagonal
 646 of the square. The motions to obey, \mathbf{d} , were sampled randomly, and the
 647 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 .

648 The parameters used to obtain TABLE 2 are the same as those used for
 649 TABLE 1. This table confirms that the value of M strongly influences the
 650 completion time. In particular, not a single complete assembly was found for
 651 $M = 5$.

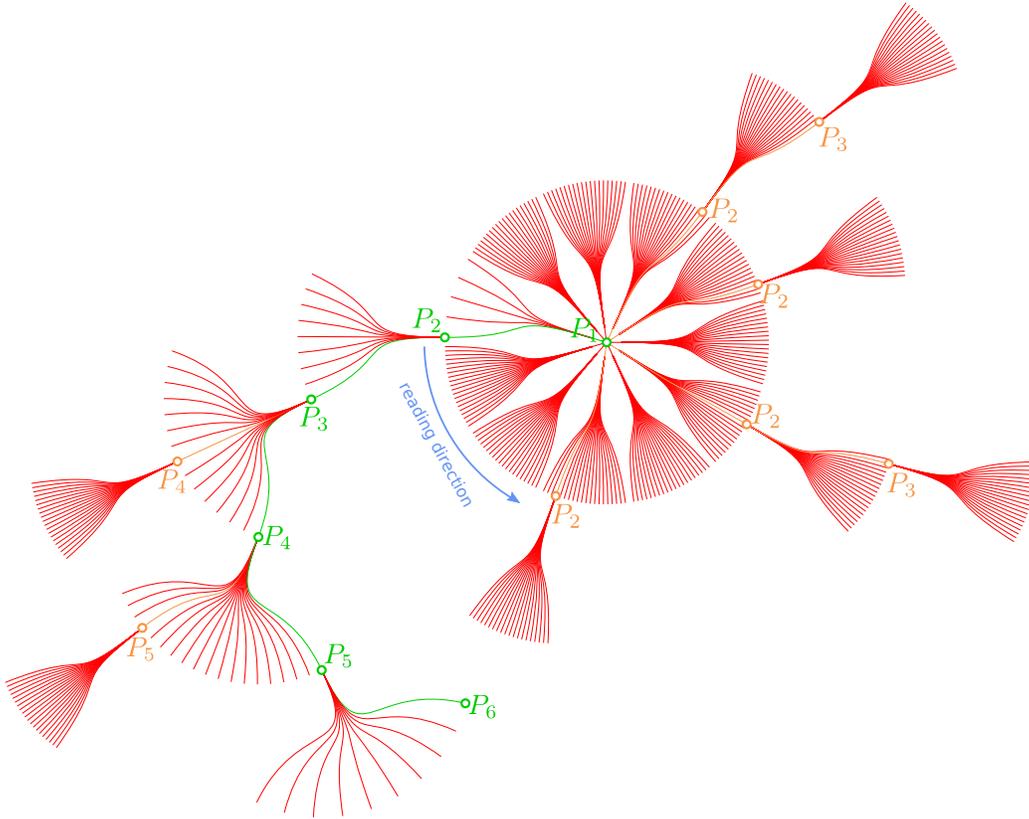


Figure 31: Regarding an assembly made of 6+1 parts, this tree illustrates the search carried out by the `Turtle` before finding a correct assembly. Here $M = 20$.

652 On FIGURE 31 the goal was to generate an assembly made of 6+1 parts.
 653 Each of the designs was recorded and arranged in a tree. For clarity, paths
 654 related to the generation of P_1 were omitted. Red paths represent fruitless
 655 paths: the `Turtle` drew a part P_i that does not meet one of the two afore-
 656 mentioned constraints. After failing $M = 20$ times the `Turtle` backtracks
 657 to the previous part. The orange paths depict paths such that a valid part
 658 P_i was successfully drawn but a subsequent part P_j , $j > i$; had we wanted
 659 $N = i$ parts the algorithm would have stopped at the latest at that path (it
 660 may have stopped earlier as the area threshold would have been less stringent
 661 and parts that have been discarded for $N = 6$ would have been accepted for
 662 $N < 6$). The green path is the valid path at the end of which the `Turtle`
 663 partitioned the design domain into a valid assembly. This figure clearly il-
 664 lustrates that most of the computation time was lost on generating fruitless

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

667 **6. Conclusion**

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

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

727

728 *Acknowledgements*

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

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

734 **References**

- 735 [1] T. Bock, T. Linner, *Advanced Construction and Building Technology*,
736 Cambridge University Press, 2015, ISBN: 978-1-107-07638-9, pp. 1–17.

- 737 doi:10.1017/CBO9781139924146.002.
- 738 [2] Déchets chiffres-clés, accessed: 2021-04-09 (2017).
739 URL [https://www.ademe.fr/sites/default/files/assets/](https://www.ademe.fr/sites/default/files/assets/documents/dechets-chiffrescles-edition2020-3-010692.pdf)
740 documents/dechets-chiffrescles-edition2020-3-010692.pdf
- 741 [3] Neutralité & bâtiment, accessed: 2021-04-09 (2016).
742 URL [https://ile-de-france.ademe.fr/sites/default/files/](https://ile-de-france.ademe.fr/sites/default/files/neutralite-carbone-batiment.pdf)
743 neutralite-carbone-batiment.pdf
- 744 [4] M. Larsson, H. Yoshida, N. Umetani, T. Igarashi, Tsugite: Interactive
745 design and fabrication of wood joints, in: Proceedings of the 33rd Annual
746 ACM Symposium on User Interface Software and Technology, UIST '20,
747 Association for Computing Machinery, New York, NY, USA, 2020, pp.
748 317–327. doi:10.1145/3379337.3415899.
- 749 [5] P. Song, C.-W. Fu, D. Cohen-Or, Recursive interlocking puz-
750 zles, ACM Transactions on Graphics 31 (6) (November 2012).
751 doi:10.1145/2366145.2366147.
- 752 [6] J. Yao, D. M. Kaufman, Y. Gingold, M. Agrawala, Interactive design and
753 stability analysis of decorative joinery for furniture, ACM Transactions
754 on Graphics 36 (4) (March 2017). doi:10.1145/3072959.3054740.
- 755 [7] Z. Wang, P. Song, M. Pauly, Desia: A general framework for designing
756 interlocking assemblies, ACM Transactions on Graphics 37 (6) (Decem-
757 ber 2018). doi:10.1145/3272127.3275034.
- 758 [8] P. Gilibert, R. Mesnil, O. Baverel, Rule-based generative de-
759 sign of translational and rotational interlocking assemblies (2021).
760 doi:10.5281/zenodo.5158465.
761 URL <https://doi.org/10.5281/zenodo.5158465>
- 762 [9] R. Testuz, Y. Schwartzburg, M. Pauly, Automatic Generation of Con-
763 structable Brick Sculptures, in: M.-A. Otaduy, O. Sorkine (Eds.), Eu-
764 rographics 2013 - Short Papers, The Eurographics Association, 2013.
765 doi:10.2312/conf/EG2013/short/081-084.
- 766 [10] S. Xin, C.-F. Lai, C.-W. Fu, T.-T. Wong, Y. He, D. Cohen-Or, Making
767 burr puzzles from 3d models, ACM Transactions on Graphics 30 (4)
768 (July 2011). doi:10.1145/2010324.1964992.

- 769 [11] C.-W. Fu, P. Song, X. Yan, L. W. Yang, P. K. Jayaraman, D. Cohen-
770 Or, Computational interlocking furniture assembly, *ACM Transactions*
771 *on Graphics* 34 (4) (July 2015). doi:10.1145/2766892.
- 772 [12] L. Luo, I. Baran, S. Rusinkiewicz, W. Matusik, Chopper: Partitioning
773 models into 3d-printable parts, *ACM Transactions on Graphics* 31 (6)
774 (November 2012). doi:10.1145/2366145.2366148.
- 775 [13] Y.-L. Yang, J. Wang, N. J. Mitra, Reforming shapes for material-aware
776 fabrication, in: *Proceedings of the Eurographics Symposium on Geome-*
777 *try Processing, SGP '15*, Eurographics Association, Goslar, DEU, 2015,
778 pp. 53–64. doi:10.1111/cgf.12696.
- 779 [14] P. Song, Z. Fu, L. Liu, C.-W. Fu, Printing 3d objects with interlocking
780 parts, *Computer Aided Geometric Design* 35-36 (2015) pp. 137–148.
781 doi:10.1016/j.cagd.2015.03.020.
- 782 [15] P. Jimenez, Survey on assembly sequencing: A combinatorial and geo-
783 metrical perspective, *Journal of Intelligent Manufacturing* 24 (2011) pp.
784 1–16. doi:10.1007/s10845-011-0578-5.
- 785 [16] A. Lambert, Disassembly sequencing: A survey, *International*
786 *Journal of Production Research* 41 (2003) pp. 3721–3759.
787 doi:10.1080/0020754031000120078.
- 788 [17] Z. Wang, P. Song, M. Pauly, State of the art on computational design
789 of assemblies with rigid parts, *Computer Graphics Forum* 40 (2021) pp.
790 633–657. doi:10.1111/cgf.142660.
- 791 [18] R. H. Wilson, J.-C. Latombe, Geometric reasoning about mechan-
792 ical assembly, *Artificial Intelligence* 71 (2) (1994) pp. 371–396.
793 doi:10.1016/0004-3702(94)90048-5.
- 794 [19] B. Romney, C. Godard, M. Goldwasser, G. Ramkumar, An efficient
795 system for geometric assembly sequence generation and evaluation, 1995,
796 pp. 699–712. doi:10.1115/CIE1995-0800.
- 797 [20] M. Bagneris, R. Motro, B. Maurin, N. Pauli, Structural mor-
798 phology issues in conceptual design of double curved systems,
799 *International Journal of Space Structures* 23 (2008) pp. 79–87.
800 doi:10.1260/026635108785260560.

- 801 [21] Optimizing structural building elements in metal by using additive man-
802 ufacturing, accessed: 2021-12-07 (2015).
803 URL [https://www.ingentaconnect.com/content/iass/piass/](https://www.ingentaconnect.com/content/iass/piass/2015/00002015/00000002/art00016)
804 [2015/00002015/00000002/art00016](https://www.ingentaconnect.com/content/iass/piass/2015/00002015/00000002/art00016)
- 805 [22] R. Oval, M. Rippmann, R. Mesnil, T. Mele, O. Baverel, P. Block,
806 Feature-based topology finding of patterns for shell structures, *Automa-*
807 *tion in Construction* (February 2019). doi:10.1016/j.autcon.2019.02.008.
- 808 [23] P. A. Gagnicuc, *Markov Chains: From Theory to Implementation and*
809 *Experimentation*, John Wiley & Sons, Inc., ISBN: 978-1-119-38759-6
810 978-1-119-38755-8. doi:10.1002/9781119387596.
- 811 [24] Z. Wang, P. Song, F. Isvoranu, M. Pauly, Design and structural opti-
812 mization of topological interlocking assemblies, *ACM Transactions on*
813 *Graphics* 38 (6) (November 2019). doi:10.1145/3355089.3356489.

814 Appendices

815 A. Parts obeying a rotation

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

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

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

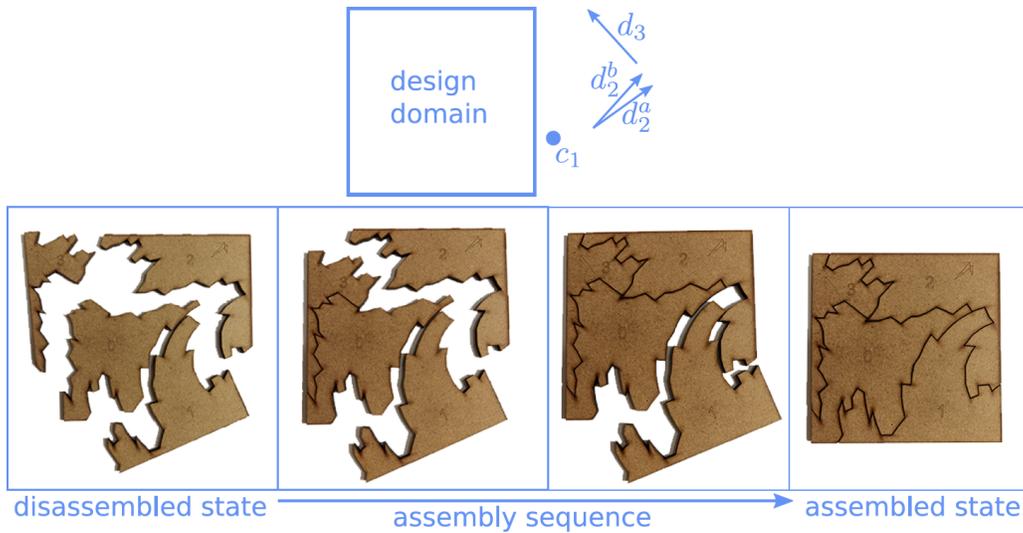
$$\begin{aligned}
P_1 \text{ obeys } \mathbf{d} &\Leftrightarrow \forall 0 \leq i < k \begin{cases} s(\mathbf{x}_{i+1} - \mathbf{x}_i) \times \mathbf{m}_{x_i} \geq 0 \\ s(\mathbf{x}_{i+1} - \mathbf{x}_i) \times \mathbf{m}_{x_{i+1}} \geq 0 \end{cases} \\
&\Leftrightarrow \forall 0 \leq i < k \begin{cases}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

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

1 Graphical Abstract

2 **Rule-based generative design of translational and rotational interlocking assemblies**

3
4 Pierre Gilibert, Romain Mesnil, Olivier Baverel



6 Highlights

7 **Rule-based generative design of translational and rotational inter-** 8 **locking assemblies**

9 Pierre Gilibert, Romain Mesnil, Olivier Baverel

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