Engineering Efficient and Massively Parallel 3D Self-Reconfiguration Using Sandboxing, Scaffolding and Coating

Pierre Thalamy[1], Benoît Piranda[1,*], Julien Bourgeois[1]

Univ. Bourgogne Franche-Comté, FEMTO-ST Institute, CNRS

1 cours Leprince-Ringuet, 25200, Montbéliard, France

Abstract

Programmable matter based on modular self-reconfigurable robots could stand as the ultimate form of display system, through which humans could not only see the virtual world in 3D, but manipulate it and interact with it through touch. These systems rely on self-reconfiguration processes to reshape themselves and update their representation, using methods that we argue, are currently too slow for such applications due to a lack of parallelism in the motion of the robotic modules.

Therefore, we propose a novel approach to the problem, promising faster and more efficient self-reconfigurations in programmable matter display systems. We contend that this can be achieved by using a dedicated platform supporting self-reconfiguration named a *sandbox*, acting as a reserve of modules, and by engineering the representation of objects using an internal *scaffolding* covered by a *coating*.

This paper introduces a complete view of our framework for realizing this approach on quasi-spherical modules arranged in a face-centered cubic lattice. After thoroughly discussing the model, motivations, and making a case for our method, we synthesize results from published research highlighting its benefits and engage in an honest and critical discussion of its current state of implementation and perspectives.

Keywords: Programmable Matter, Distributed Algorithms, Modular Robotics, Self-Reconfiguration, Multi-agent Systems

1. Introduction

Through human history, humans have constantly been on a quest to represent the world around them, perhaps as a way of communicating and collaborating more efficiently with their peers and passing on their culture to future generations. One common way of representing the world is through the use of *display systems*, a term which can widely fit any medium, device, or system that can be used as way to display information. Some of the first ever display systems were most arguably something like cave paintings, a very primitive way of representing a 3D world with simple 2D drawings. For the most part of human history, these display systems have remained tragically two-dimensional, failing to accurately describe a world with an extra dimension. Some technologies have made an exception, such as sculpture, but they usually required very high effort and skills. With

*Corresponding author

Email addresses: pierre.thalamy@femto-st.fr (Pierre Thalamy), benoit.piranda@femto-st.fr (Benoît Piranda), julien.bourgeois@femto-st.fr (Julien Bourgeois)

URL: www.femto-st.fr/en/femto-people/pthalamy (Pierre Thalamy), www.femto-st.fr/en/femto-people/bpiranda (Benoît Piranda), www.femto-st.fr/en/femto-people/jbourgeo (Julien Bourgeois)

[1]Univ. Bourgogne Franche-Comté (UBFC), FEMTO-ST Institute, CNRS, France

recent advances in digital technologies, 2D display systems are now ubiquitous. Most allow interaction with the user, through touch for instances, but they largely still fail to account for that extra dimension of the real world. Low effort visual 3D display systems have nonetheless existed for quite some time in the collective imagination (e.g., holographic displays in science fiction), and are starting to materialize with relatively recent technologies such as virtual reality (VR) and augmented reality (AR). However, we want to take 3D display systems one step further, and to be able to display objects and data in an interactive and tangible way through the use of *programmable matter*.

Programmable matter is usually defined as matter that is able to **autonomously** change its physical properties (shape, color, density, etc.) programatically in reaction to an event, internal or external. Although several technologies could be considered a kind of programmable matter, we will be focusing in this paper on programmable matter based on **modular self-reconfigurable robotic systems** (MSRS). Those are robotic systems made from an arrangement of independent but interconnected robotic modules that must coordinate in order to achieve a common goal, usually while remaining connected to each other at all times. Modular robotic systems provide programmability, autonomy, interactivity, and evolutivity through the motion of their parts, making them, we believe, the current best substrate for programmable matter. They come in different architectures Ahmadzadeh et al. (2016), mainly as *chain* modular robots where modules act as joints and links to form chains, or as *lattice* modular robots were modules assemble to form 2D or 3D lattice structures that can be easily modeled as a discrete-space grid. Finally, *hybrid* modular robots belong to both classes at once. Of modular robotic systems, their most interesting feature is their ability to *self-reconfigure*, to change their morphology from an *initial* configuration to a *goal* configuration of its modules through the motion of their parts. As it turns out, the *self-reconfiguration problem*, whose goal is to find a sequence of (potentially concurrent) module motions transforming an initial configuration into a goal one is in fact a very complex problem Bourgeois et al. (2016). First, given a modular robot with n modules, the number of possible configurations that can be made with such a robot is huge: $(c.w)^n$, with c the number of possible connections per module, and w the number of ways of connecting them together Park et al. (2008). Second, the branching factor of the configuration space, describing all possible configurations that can be achieved from a given configuration is very high: $O(m^n)$, with m the number of possible movements and n the number of modules free to move Barraquand and Latombe (1991). This leads to a combinatorial explosion making the search space between any two configurations exponential in the number of modules, resulting in the intractability of the self-reconfiguration problem. It has been proved to be NP-complete for chain-type MSR by reduction to 3-PARTITION Hou and Shen (2010) and Probabilistic SATisfiability (PSAT) Gorbenko and Popov (2012), and is also suspected to be NP-complete for lattice MSRS.

Not only is self-reconfiguration computationally intractable when computing it in a centralized fashion, but most modular robotic systems must avoid relying on centralized computing and use a distributed paradigm instead. The difficulty of distributed self-reconfiguration algorithms then also comes from the incredible level of coordination required for many modules to move in parallel while avoiding collisions between each other, and creating *deadlocks* during the construction process, situations where part of the construction cannot be realized because some modules are blocking part of the goal shape, perhaps because of an unfeasible construction scheduling.

In fact, a self-reconfiguration algorithm could well proceed without any parallel motion of the modules, but the critical parameter that must be optimized in self-reconfiguration is *reconfiguration time*. That is the time to transform the initial configuration $\mathcal{I}$ into the goal configuration $\mathcal{G}$. This reconfiguration time is usually expressed in number of discrete time steps, and as a function of the number of modules in the configurations. It is evidently much slower to perform the transformation one module at a time than with parallel motions, and sequential algorithms are hence mostly impractical due to their slow speed. Other metrics for self-reconfiguration algorithms include the total number of motions performed by the modules, the memory cost of the algorithm, or the

number of messages exchanged.

Several research communities have become interested in the self-reconfiguration problem, with very different viewpoints Thalamy et al. (2019b). On one end of the spectrum, roboticists seek to design algorithms to implement specific behaviors in their hardware systems; on the other, theoretical computer scientists study this problem from the standpoint of computational geometry Michail et al. (2017) — mostly on 2D problems until now. Finally, in-between these two extremes, other research communities such as in distributed systems study self-reconfiguration algorithms without necessarily being involved in hardware research. We mostly belong to the latter, even though as we will see we are now making progress in the design and manufacturing of hardware micro-scale modular robots.

On the robotics side, early work on self-reconfiguration considered only a very limited number of modules in the system (dozens), and with intricate geometries that made self-reconfiguration excessively complex (bipartite systems, for instance Kotay and Rus (2000); Ünsal et al. (2000), or others Yoshida et al. (1998)). The field also moved away from heavily centralized approaches to more adequate distributed methods, better fitted for large-scale reconfiguration and the dynamic nature of the underlying systems. While most self-reconfiguration methods are deterministic, a number of stochastic methods can be found in the literature. Early attempts based on stochastic relaxation Yoshida et al. (1998) or simulated annealing Kurokawa et al. (1998) suffered from a difficulty to converge into the goal shape as they were getting trapped into local minima. Nonetheless, more recent attempts such as Fitch and McAllister (2013)—based on a Markov decision process to optimize the number of connections/disconnections of modules—have proven very promising as they can easily be made generic and applied on diverse hardware systems and models. Stochastic methods might also be by themselves more robust to faults in hardware systems during reconfiguration, which is critical, while deterministic methods would need additional correction mechanisms.

On the other hand, deterministic approaches generally have a guaranteed convergence into the goal shape, but might need additional correction mechanisms in case of hardware failures as opposed to the built-in robustness of stochastic methods. The most popular self-reconfiguration model is by far the simple *sliding-cube*, which resides in a cubic lattice and can perform translations and convex rotations on the surface of other modules, or only one of the former in some models. Approaches vary from disassembly/reassembly through an intermediate shape Fitch et al. (2003), tunneling through the shape with sliding-only cubes Kawano (2015) (both with quadratic operating time cost), to more specialized methods such as Bie et al. (2018) which can build branching structures in a linear number of module motions using Lindenmayer-systems and cellular automata Bie et al. (2018); Zhu et al. (2017).

The self-reconfiguration problem evidently stands as one of the major foundational issues of programmable matter. In this context, algorithmic solutions are required to exhibit some particular properties, that we discuss in Thalamy et al. (2019b) through an analysis of the state of the art of self-reconfiguration juxtaposed to the expectations of programmable matter.

To solve the reconfiguration problem more efficiently we propose, therefore, two optimizations. The first one is to change the way we define an object: rather than constructing an object filled with micro-robots, we define it using its boundary representation. Second, we propose, based on previous research Kotay and Rus (2000); Ünsal and Khosla (2001); Lengiewicz and Holobut (2019); Støy (2006); Støy and Nagpal (2007) to build an object using an internal *scaffold* that leaves internal holes inside the shape to facilitate motion and coordination. This scaffold can then be coated by modules to restore the external aspect of the object. Accordingly, while the object will look like a plain object from the outside, it will actually be composed exclusively of a scaffold with an added coating. The resulting object will thus contain fewer micro-robots than it would otherwise, and these micro-robots will be able to move inside the object; these two features significantly contribute to decreasing the reconfiguration time. Furthermore, while traditional self-reconfiguration research assumes systems where all the number of modules in the initial and goal configurations are equal, we drop this constraint and assume that our self-reconfigurations take place in a dedicated environment named a *sand-*

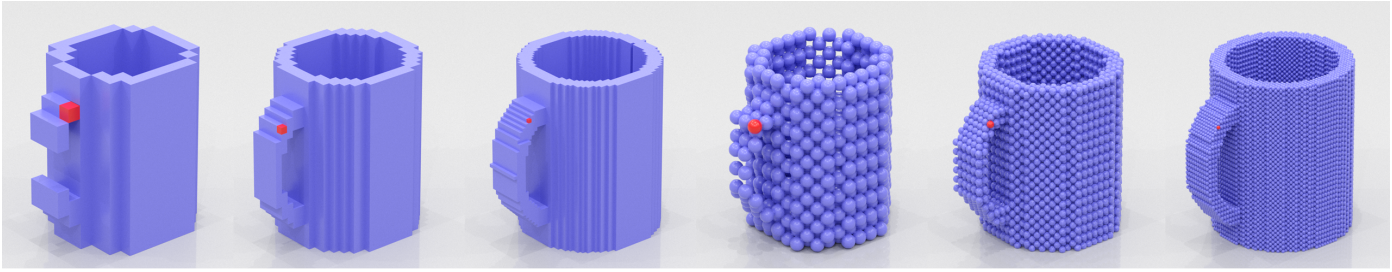

Figure 1: Visual comparison of cups made of modular-robot-based programmable matter with cubic and spherical modules and at various resolutions in terms of the size of the modules.

box, that acts as a reserve of modules placed underneath the reconfiguration scene and that can supply and discard modules from the reconfiguring ensemble.

As an anecdotal example of self-reconfiguration speeds, an estimate of the reconfiguration time for our sliding blocks Piranda et al. (2013) and using the metrics introduced in Zhu and El Baz (2019) is 12 hours for 800 blocks. This is just a rough estimate to stress that time is really an issue in self-reconfiguration algorithms. Besides, it was shown in practice that it could take 11.66 h to reconfigure an ensemble of 1,000 Kilobots Rubenstein et al. (2012). Given a rotation time of 20 ms, as gauged by our latest hardware experiments with our quasi-spherical rotating 2 mm *3D Catom* modules, and using scaffolding, we simulated that it would take roughly 6 s to build the scaffold of a cube of size $19 \times 19 \times 19$ modules (110 cm^3) and made of nearly 1,200 modules— while the filled cube of the same size would consist of 6,859 modules. Such a tremendous reconfiguration time decrease can seem impossible and suspicious. Indeed, this is more of an order of magnitude than a precise result, as reconfiguration time also has to factor in the time spent processing and communicating by the modules, control loops, synchronization, etc. This figure is thus most likely a very low approximation of expected self-reconfiguration time. Nonetheless, even if the actual duration of ends up $\times 50$ times greater, the total reconfiguration time for more than a thousand modules would still be a matter of minutes, not hours. This hints that together, electrostatic actuation and the reconfiguration method we propose can dramatically reduce self-reconfiguration time, enabling practical applications for programmable matter.

In this paper, we introduce in detail our proposed approach to the fast, efficient, and high-fidelity self-reconfiguration of modular robotic systems for display systems based on programmable matter. We aim to establish the foundations of this new self-reconfiguration paradigm, and highlight analyses and simulation results from published research showing that it delivers on its promises. We begin with a thorough presentation and discussion of the robotic model under consideration and its constraints in Section 2. Based on this model and constraints, as well as the current state of the art of the field, we explain in Section 3 how the traditional challenges of self-reconfiguration and that of our own model can be solved or simplified using a combination of existing techniques (e.g., scaffolding, local motion rules), and new additions (i.e., sandboxing, coating). Section 4 then gives an account of the various construction algorithms for constructing the scaffold and the coating in a sandbox environment based on our published research, and highlights the key performance takeaways from our analyses and simulations of these algorithms. Finally, Section 5 takes some distance from the our approach and discuss the strength and current limits of our implementation and its underlying robotic model. Last but not least, Section 6 draws a number of perspectives for improving and enriching our implementations, and generalizing our method to other models.

1.1. The Programmable Matter Project

The work presented in this paper is part of a much larger effort to realize the concept of programmable matter. A consortium around this effort has been built under the name of *The Programmable Matter Consortium*, and has rallied numerous partners from major research institutions (see Programmable Mat-

ter Consortium Web Site[2]), industrial leaders and manufacturers, and even art studios tasked with exploring the concept from their perspective.

This partnership goes beyond the mere financial support of research efforts through research grants, as it enables frequent real-world interactions and collaboration between partners. The value of this partnership is the open and global exchange of expertise between research teams with diverse and complementary specialties, each capable of addressing some of the many challenges standing between the current state of technology and the real-world programmable matter of our future.

Our consortium is involved in research on both software and hardware aspects of the technology. On the software side, the objectives of the consortium are to establish strong theoretical and algorithmic foundations (synchronization, leader election, distributed coordination, etc.) for programmable matter systems, and further our understanding of the capabilities of metamorphic self-reconfiguring systems—which is where the topic of this work fits.

On the hardware side, the current main focus of this joint enterprise is to push against the current limits of miniaturization of modular robotic systems and start a new era of systems at the micro-millimeter scale. This requires major innovations in several areas of research from the fields of micro- and nano-electro-mechanical systems and electrical engineering, such as computer miniaturization and integration, electrostatic actuation, nanoscale 3D printing technologies, etc. Bourgeois and Goldstein (2012) The robotic architecture under development is also the one that is taken into consideration in this paper: the *3D Catom*. The target size of the current prototype is a quasi-spherical autonomous robot of 3.6mm diameter (See Section 2.4 for more information).

If we believe that miniaturization is such an important aspect of modular-robot-based programmable matter, it is because the size of the modules has such a tremendous impact on the fidelity of programmable matter object with regard to their inert counterpart. To better illustrate the influence of the geometry and size of the modules, Figure 1 shows a visual comparison of programmable matter cups made from cubic and spherical modules at various sizes.

[2] https://www.programmable-matter.com

2. *3D Catom* Model and Motion Constraints

In this section, we introduce the model that stands as the basis of this work, both in terms of the robotic and computing model. The exact characteristics of the envisioned *3D Catom*, its underlying theoretical model, and its current state of hardware development, are thoroughly discussed.

2.1. The 3D Catom Modular Micro-Robot

As it has just been pointed out, the distributed algorithms presented in the following sections consider the self-reconfiguration of modular robots named **3D Catom**s. We will thereafter refer to a single unit from this modular robot as a *3D Catom* module, or simply as a **module**. *3D Catoms* are quasi-spherical rotating modules, with an expected diameter in the micro-millimeter range. These modules can attach to each other and rotate around one another without moving parts through electrostatic actuation.

2.1.1. Geometry

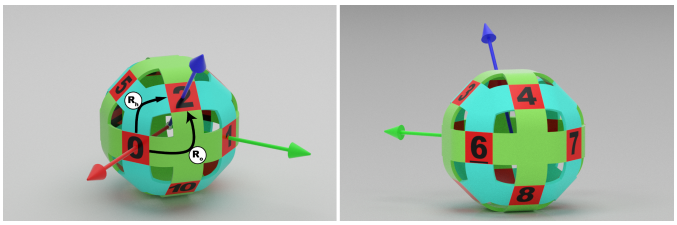

Figure 2: **The *3D Catom*:** geometry from two opposite angles, skewed coordinate system, and the two possible paths for the motion of a neighbor on its surface, using and a hexagonal actuator (R_h) or an octagonal actuator (R_o).

3D Catoms have a quasi-spherical geometry which consists of 12 flat squares (the connectors, drawn in red in Figure 2a, linked by curves. Connectors are centered and tangent to the contact points of a dense set of spheres placed in a **Face-Centered Cubic (FCC) lattice** (see Piranda and Bourgeois (2018) for relative contact points coordinates). There are two kinds of curves that are placed between connectors to allow the rotation of a *3D Catom* around another: The first shape, the *hexagonal actuator* (drawn in blue in Figure 2) is made of a triangle and 3 sections of the body of a cylinder; the second shape, the *octagonal actuator* (drawn in green in Figure 2)

is made of a square and 4 sections of the body of a cylinder.

3D Catoms assemble by latching onto each other using one of the 12 electrostatic connectors (also, interfaces) on their surface (numbered from #0 to #11 for identification purpose), and form FCC lattice structures—see Figure 3 for the positions of all 12 neighbors (modules directly connected to a given module) of a *3D Catom*, across three layers. Each position within a *3D Catom* lattice can also be referred to as a *lattice cell* (or simply *cell*), and is assigned a unique discrete coordinate. We assume that modules can only sense the presence or absence of their immediate neighbors, through their interfaces.

Furthermore, as can be seen in Figure 3, modules on adjacent horizontal layers of modules are staggered. For that reason, different coordinate systems can be pertinent depending on the task at hand. We choose to use a coordinate system with a skewed $\vec{z}$ axis (cf. Figure 2)—as it circumvents the trouble of a straight $\vec{z}$ axis caused by having different top and bottom neighbor positions depending on the parity of the current horizontal layer.

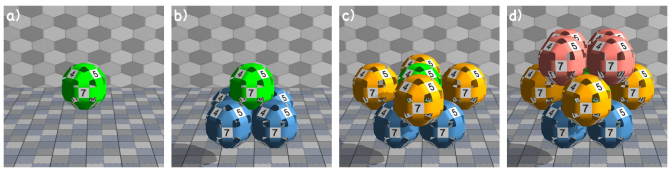

Figure 3: Arrangement of a 1-ball of *3D Catoms* in a Face-Centered Cubic (FCC) Lattice.

2.1.2. Rotations

3D Catoms move around the FCC grid by rotating on the surface of the surrounding modules, using their orthogonal or hexagonal actuators. Individual movements consist in a rotation from one connector of a neighbor module to another connector on the surface of this same module, which acts as a *pivot*. Note that a module acting as a pivot is not allowed to perform a motion while it is actuating for another module, and that a moving module cannot carry another during its motion.

Figure 4 shows the two possible ways of performing rotations, using an octagonal actuator on the left (rotating around the green pivot), and using a hexagonal actuator on the right (rotating around the yellow pivot). Each rotation displaces the rotating module from one cell to an adjacent one within the FCC lattice. More complex motions comprised of several steps are therefore sequences of individual rotations on the surface of neighbor modules. All lattice cells are not accessible to every module, these restrictions are the result of motion constraints.

First, a module can perform a motion from its current position to an adjacent one if and only if one of its immediate neighbors can act as a pivot for that motion, meaning that this neighbor module must be both a neighbor of the module seeking to move and of the target position of the movement, which must also be free. However, this condition is not sufficient condition to guarantee the success of the motion, as there could be a module in the neighborhood of the pivot whose presence could produce a collision. For instance, if a module is attached to connector #0 of the *3D Catom* in Figure 2 (acting as a pivot in this case) and seeks to move to connector #2 of the pivot, it would not be able to do so using the octagonal actuator if the pivot has a neighbor on its interface #5. Accordingly, using the hexagonal actuator of the pivot would not be possible either if it has a neighbor on its interface #10, in which case that module would not an appropriate pivot for that motion.

It thus becomes apparent that locally planning motion is not a trivial task for *3D Catoms*. We will say that a module is *mobile* if it can perform at least one motion in any direction. Furthermore, *3D Catoms* do not undergo any deformation when rotating, as deformation is likely to require moving parts, and *3D Catoms* are meant to be inexpensive and mass producible by design. This raises, however, an important constraint on the movement of modules, as **the geometry of *3D Catoms* thus does not allow a module to enter or leave a position that is surrounded by two opposing modules** (as illustrated in the rightmost part of Figure 4). In terms of the module, this means that a module cannot move if it has neighbors connected to two of its interfaces that are opposite, such as connectors #2 and #8, or #1 and #7, in Figure 2.

This means for instance that in the case of two lines of modules growing into each other, it would not be possible to insert the last module required to bridge the gap between the two lines. This is a major constraint on any self-reconfiguration using *3D*

Catoms, as this means that the construction of any shape must follow a strict set of ordering principles and construction rules so as to avoid the occurrence of deadlocks during construction. From here on, we will refer to this constraint as the **bridging constraint**.

Furthermore, any motion that would result in a collision with another module is prohibited. There are two possible scenarios in which that could happen. On the one hand, there is the presence of a module that is not in the local neighborhood (i.e., the 12 immediate neighbors) of a mobile module, and that nonetheless blocks this module from entering its target cell, resulting in a collision. While a module can directly sense its immediate neighbors to ensure that none might impede on its motion, there is no local way of doing such verification on a wider radius in our model. We will refer to this problem as the **remote blocking conundrum**. On the other hand, a collision could occur between two mobile modules if the two have planned concurrent and intersecting motions, which is also a scenario that cannot be prevented solely from the local scope of a module. This is the **motion coordination challenge**. We will discuss potential solutions to these problems after having introduced the communication capabilities of *3D Catoms*. Figure 5 illustrates these two problems: On the left, the motion of the green module will collide with the orange module even though its local neighborhood and the one of its pivot (magenta) are clear for this motion; On the right, both yellow modules are attempting to enter the same lattice position, as no local constraint prevents it, which will result a collision between the two.

2.1.3. Communication

Latching and motion actuation are not the sole purpose of the electrostatic connectors on the surface of *3D Catoms*. We consider in our *3D Catom* model that all communication is performed locally, according to the distributed communication paradigm, and in a peer-to-peer fashion between connected modules through their electrostatic connectors. Hence, *3D Catoms* cannot rely on global communication to receive essential data about the state of the system, but must instead propagate information distributively.

2.2. Motion Constraints and Challenges

As a consequence, when considering motion and communication constraints together, we see that modules can only sense their local neighborhood (immediate neighbors) through the connection of their interfaces, which they can also use to communicate with their first degree neighbors exclusively. However, modules in their second-degree neighborhood (the neighbors of their neighbors), can still cause trouble (cf. the **remote blocking conundrum**). Validating a candidate motion thus involves ensuring that no position in the first and second degree neighborhoods of a module are blocking that motion. The former is done locally, therefore, while the second has to be done through remote communication, distributively searching the graph of the configuration until all lights are green. Unfortunately, this necessarily means performing an entire flooding of the *3D Catom* network every time a motion must occur, which is insanely prohibitive due to the sheer number of messages that such verification would require (as well as the time and energy cost). Besides, the **motion coordination challenge**, that is in itself a *race condition*, can be solved by more local mechanisms, such as the virtual locking of lattice cells surrounding a moving module, much like *mutual exclusion* around a shared resource in programs with concurrency situations. While this is not local to the mobile modules, it is at least regional, as they would only need to lock all the cells in a one-module-thick FCC ball (cf. Figure 3d) around their initial position. Nonetheless, the lack of shared memory primitives between nearby modules still makes such strategy quite burdensome and impractical.

There is one last motion constraint that has not yet been mentioned, named the **connectivity constraint**. It states that all modules in a *3D Catom* ensemble must remain connected as a single ensemble at all times. In other words, the *connectivity graph* $G = (V, E)$ where V is the set of all modules in the system and E the interconnection and between modules through their interfaces. According to the connectivity constraint, G must remain a *connected* graph at all times, that is to say that no module motion that would split the graph G into two disconnected subparts is allowed (even temporarily). An example is given in Figure 5, where all the modules outlined

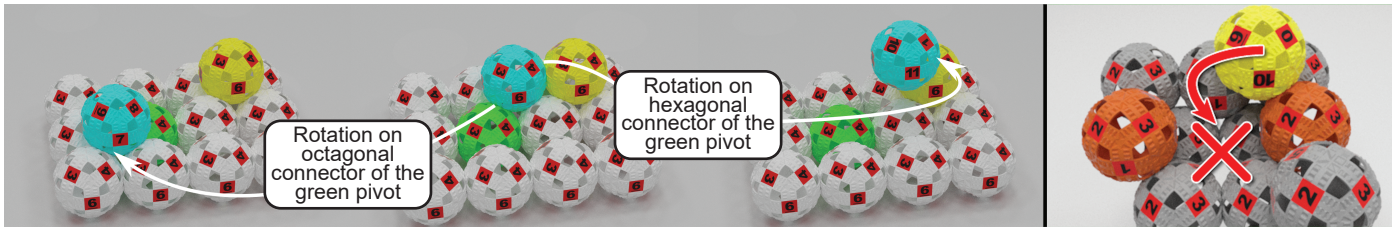

Figure 4: **(Left)** Two additional sample motions on octagonal and hexagonal actuators. **(Right)** The *bridging constraint*, in which the yellow module is unable to reach its destination because of the two blocking modules in orange.

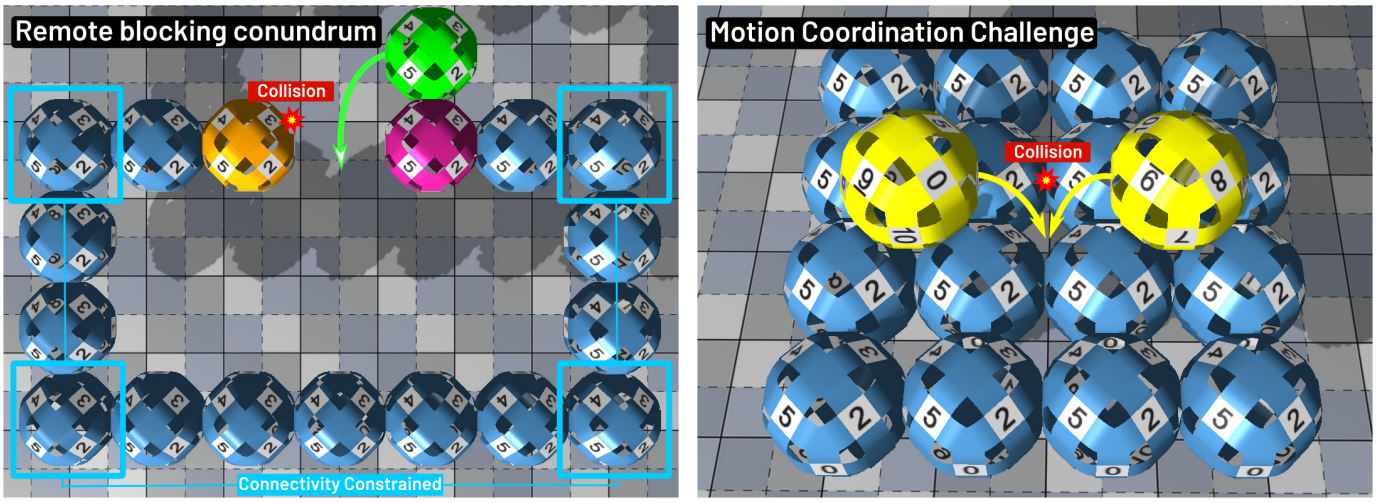

Figure 5: The two main motion challenges posed by the *3D Catom* model: the **remote blocking conundrum** and the **motion coordination challenge**.

with a light blue square are not allowed to move due to the connectivity constraint, even though they are mobile in all other aspects. The purpose of the connectivity constraint is to guarantee that modules can always act as a single global entity (i.e., they can always communicate with each other), and that the ensemble is therefore never split into two or more disjoint parts. Furthermore, in some modular robots, this also ensures that power can always be supplied to all modules in the system. Indeed, some modular robotic systems are likely to require an external source of a power, that would have to be routed to all modules in the system Daymude et al. (2020)—a daunting challenge.

Much like for the **remote blocking conundrum**, checking the connectivity constraint also requires a massive communication overhead, as it involves evaluating whether a module seeking to move is an *articulation point* of the configuration graph. Støy (2004) showed that this could be one by using a *connection gradient* propagated from modules that are in their final positions, and with an *invariant* on module motion stating that the motion should not alter the connection gradients of the surrounding modules.

We have thus seen that motion constraints exist at two levels for a *3D Catom*: **local** motion constraints and **global** ones. All motion constraints imposed on *3D Catom* modules are summarized below, with for each of them the corresponding condition that has to be evaluated by the module before attempting a motion:

Local motion constraints can be directly evaluated by a module:

- Pivot constraint: *"Can one of my nonmoving neighbors act as a pivot to get me to the cell I am trying to reach?"*
- Local criterion of the bridging constraint: *"Do I have two neighbors that are on opposite connectors?"*

Global motion constraints, however, must be resolved through regional or systemwide distributed communication:

- Global criterion of the bridging constraint: *"Are there two opposite neighbors surrounding the position that I am trying to reach?"*

- Remote blocking conundrum: *"Is there a non-connected module that might impinge on my motion path?"*
- Motion coordination challenge: *"Is my motion crossing the path of another module?"*
- Connectivity constraint: *"Will this motion split the 3D Catom ensemble in two disconnected subparts?"*

2.3. Programming Model and Assumptions

As we have seen, while the movement of a single module can seem trivial, the intricacy of self-reconfiguration becomes apparent when considering *3D Catoms* in a swarm context, with multiple modules attempting to perform their respective tasks in parallel. Below are a number of additional assumptions that govern the *3D Catom* ensembles under consideration in our work.

We model a modular robot consisting of a connected ensemble of *3D Catoms* as a distributed system, where:

- The system is **fully distributed**
- each module is assigned a **unique identifier**;
- all modules are identical and execute the exact same distributed program—*3D Catoms* thus form a **homogeneous modular robot**;
- modules can only react to either the reception of a message, to the connection/disconnection of a neighbor, or to an internal event such as a timed interruption or the start or the end of a motion;
- computation is only performed locally to each *3D Catom*;
- communication is also performed exclusively in a local fashion, with modules only communicating with their immediate neighbors on the FCC grid, and through a **message-passing scheme**;
- message sending and message propagation time are negligible against the rotation time of *3D Catoms*;
- all modules share a common coordinate system and have a global knowledge of the goal shape Tucci et al. (2017) for self-reconfiguration tasks;
- modules perform everything **asynchronously**.

2.4. First Hardware Prototype

Although this can seem like a strictly abstract model, several partners from the Programmable Matter Consortium are actively engaged in creating hardware *3D Catoms* and will soon complete the production of the first *3D Catom* prototype.

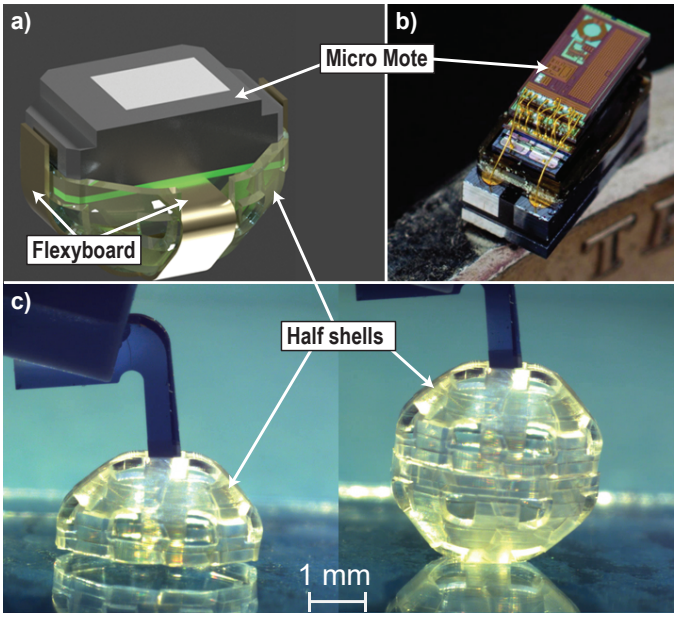

Figure 6: First *3D Catom* prototype. a) A synthetic model showing the design of the several parts of the *3D Catom*. b) A picture of the M^3 millimeter scale computer. c) Two steps of the assembly of the two half shells of the *3D Catom* using micro manipulators.

The *3D Catom* design under realization therefore consists in a 3D-printed envelope referred to as *the shell*, (cf. Figure 6c, assembling picture, courtesy of David Hériban - Percipio-robotics, showing a 3.6 mm diameter shell), with electrostatic actuators on its external surface (cf. flexyboard on Figure 6a), and with a processor, battery, and high voltage driver for commanding the actuators embedded inside (cf. Micro Mote on Figure 6b). The brain of a *3D Catom* is the Michigan Micro Mote (M^3) Pannuto et al. (2013), the world's smallest computer at the time of writing, which is a testimony to the cutting-edge nature of the *3D Catom* technology. The overall size of the final first prototype is 3.6mm.

3. Engineering Fast and Efficient Self-Reconfiguration

Throughout the past two sections, we have discussed the relevant state of the art of self-reconfigu-

ration (Section 1), and identified the modular robotic model, constraints, and assumptions under which we consider self-reconfiguration (Section 2).

Building upon this knowledge, we describe in this section how we propose to accelerate the process of self-reconfiguration and attenuate the complexity of module motion planning. This is done by slightly tweaking the parameters and setting of the self-reconfiguration, and by forcing an arrangement of the matter that structurally mitigates concurrency issues.

We will see that this strategy is not suitable for all kinds of context for self-reconfiguration. *"There ain't no such thing as a free lunch"*, saloon owners and economists would tell you alike, and this is no exception. In this particular case, we are conceding to sacrifice ubiquity and mobility (of the system as a whole), to gain a boost in reconfiguration speeds. This may not be acceptable for many applications of metamorphic robots (such as search and rescue robots), but we believe that for the purpose of object representation and display, it is.

As we have seen in Section 2.2, the *3D Catom* model imposes highly restrictive constraints on the motion of the modules, with some that are not even local to the modules themselves. We have discussed the tremendously prohibitive messaging overhead required to clear the **remote blocking conundrum** or the **connectivity constraint** for any motion. While these motion constraint issues are exacerbated by the FCC structure of the lattice and the ensuing combinatorial explosion of a 12-neighbor grid, they are fairly standard among modular robotic models. There is, however, one particular technique that researchers in the field have used to remediate some of these issues and facilitate planning: **scaffolding** — which was introduced in Section 1. In this work, we build upon the idea of a scaffolding, and propose a geometry and construction method for an FCC lattice scaffolding, from which we derive very important self-reconfiguration benefits.

As a reminder, scaffolding consists in arranging the internal structure of the modular ensemble as a porous and highly regular scaffold, intentionally reducing the density of the matter so as to ease the internal motion of modules through it. This can be done by removing modules that do not contribute to the overall shape or to the structural strength of the configuration Støy (2004). Nonetheless, this idea had only ever been applied to modules in a Square Cubic (SC) lattice (a regular 3D grid), and with holes no larger than one module in size. The best example of such apparatus can be found in the works of Støy Støy (2004); Støy and Nagpal (2007) (see Figure 7). The same scaffold geometry later inspired Lengiewicz and Holobut (2019), with a resembling model but solving reconfiguration through a max-flow search to optimize the flow of modules between the boundaries of the initial shape and those of the goal shape. Both achieved self-reconfiguration with a number of individual movements linear in the number of modules present in the system.

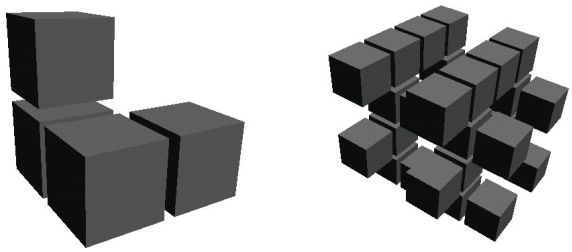

Figure 7: Scaffolding structure in a Square Cubic (SC) lattice as proposed by Støy and Nagpal (2007).

Scaffolding is a powerful tool for self-reconfiguration and presents numerous advantages, which are discussed below.

3.1. Motion Support

The primary motivation for using a scaffolding structure is that it greatly relieves the burden of concurrent motion planning that has to be done by the programmer of the system, as we will see. But on a higher level, a scaffold also has an impact on the actual possible trajectories of module motions: By lowering the density of the configuration and leaving space inside the forming objects, modules can now flow through the object, while with a higher density only surface motions are possible. This means a different reconfiguration paradigm where not only more paths are available for module motions, but they are also shorter and more direct.

Furthermore, thanks to the regularity and thus to the predictability of the internal structure of the scaffold, the interior of the growing object can be segmented into deterministic and parallel (non-intersec-

ting or at least seldom-intersecting) motion paths through which modules can flow without risk of concurrency issues. This process is named **pipelining**. This transforms the programming challenge from a motion coordination problem to the one of the resource allocation or traffic regulation across motion paths, with the coordination issues between modules now only local to specific locations where motion paths are susceptible to intersect. But this predictability can also directly benefit solving the **remote blocking conundrum**. By increasing the spacing between modules and reducing the number of neighborhood locations of a module that can be occupied, planning motions becomes easier with scaffolding. Careful readers might point out that reducing the number of potentially blocking positions still means having to check the presence of modules in several positions before a motion, which still generates an unreasonable communication overhead — no matter how reduced it is. We propose, however, to remove that communication phase altogether, by artificially reducing the set of motions that a module can undertake to a number of predetermined motion *paths* between two scaffold positions, where all motions in the path cannot be blocked by a scaffold module Thalamy et al. (2020b) will cover that further. Finally, scaffolding can also have a positive effect on the other dreadful global motion constraints: the **connectivity constraint**. Once again, this depends on the exact design of the scaffolding structure, which will be discussed later on, but with our scaffold design, every move that is not locally immobilized (by local motion constraints) cannot break the connectivity constraint either. One last thing remains unaddressed by the scaffold at this point is the mutual exclusion problem and motion coordination challenge. On the one hand, mutual exclusion over a particular pipeline or motion path is ensured thanks to a local traffic-light-style motion coordination protocol Thalamy et al. (2020b). On the other hand, mutual exclusions at path intersections are handled by the reconfiguration algorithm itself.

As a summary, Table 1 shows the impact of scaffolding on the various motion constraints introduced in Section 2.2.

Motion Constraint	**Aided by Scaffold**
Pivot Constraint	✗
Bridging Constraint (Local)	✗
Bridging Constraint (Global)	✓
Remote Blocking Conundrum	✓
Motion Coordination Challenge	✓
Connectivity Constraint	✓

Table 1: Summary of motion constraints that can be easier to satisfy in a scaffold setting.

3.2. Reduced Matter Usage

Another advantage of scaffolding is that it reduces the number of modules that constitute the target object. However, this also means that scaffold-based self-reconfigurations have also *less modules to displace*, which also contributes to a faster reconfiguration time. Its purpose is therefore twofold, both having a positive impact on reconfiguration time: reducing the amount of matter that must be displaced to perform the reconfiguration, and supporting module motions for an easier displacement and coordination of the remaining modules.

As there are no existing scaffolding models for the FCC lattice and our module geometry, we propose a novel scaffold model that suits our needs in Section 3.6.

3.3. The Limits of Scaffolding

There is, however, a major downside to scaffolding, and it is that it greatly alters the quality of the visual aspect of the represented objects. Indeed, on a structural level using scaffolding for self-reconfiguration can be seen much like audio compression methods: audio compression methods discard information that is not essential to the integrity of the track for the sake of a reduced memory footprint, but it might impact audio quality; scaffolding saves modules and eases processing at the expense of visual quality, by only keeping modules that are mechanically/structurally essential.

Notwithstanding, we believe that this can be mitigated by **coating** the scaffolded object after the reconfiguration with a thin layer of modules, in order to achieve the benefits of scaffold usage at a lower cost to the external aspect of objects. Thalamy et al. (2020a) further explores this idea, and Figure 8 illustrates this concept of coating through a side-by-side

comparison of regular, scaffold-based, and coated objects.

3.4. The Sandbox: Enabling Anisonumeric and Clustered Reconfiguration

Virtually all self-reconfiguration problems that have been posed until now have assumed that the initial and the goal configuration had *the same number of modules*, which we will refer to as **isonumeric** reconfiguration. This makes perfect sense for modular robotic applications in unknown and unstructured environment such as exploration or search and rescue missions, or any other applications where versatility/ubiquity is important or where the integrity of the robotic unit has to be maintained. However, in other applications such as our object representation in our case, where self-reconfiguration could be confined to a *dedicated environment*, *anisonumeric* reconfiguration can be considered — reconfigurations where the number of modules in the initial configuration $\mathcal{I}$ and goal configuration $\mathcal{G}$ differ ($|\mathcal{I}| \neq |\mathcal{G}|$). **anisonumeric** reconfiguration thus covers two possible cases in the relation between the number of modules during reconfiguration: **hyponumeric** reconfigurations, where the goal configuration has fewer modules than the initial one ($|\mathcal{I}| > |\mathcal{G}|$), and **hypernumeric** reconfigurations, which is the opposite ($|\mathcal{I}| < |\mathcal{G}|$).

This constraint on the size of the configurations also relates to the connectivity constraints, as attracting new modules coming from outside the system requires some sort of wireless or global communication channel since they are not connected to the configuration. Accordingly, discarding modules from the initial configuration would mean that these modules cannot be summoned back into the reconfiguring robot later without external communication means. Some authors have thus relied on wireless communication for that purpose, or structured (grid-like) environments supplying energy and used as a global communication bus between modules across the grid Spröwitz et al. (2014).

In a nutshell, *hyponumeric* reconfiguration thus requires the extra modules can be discarded somewhere near the reconfiguration scene, and *hypernumeric* reconfiguration requires a reserve of modules from which additional modules can be attracted. Both require a means of communication between modules in the configuration and modules that are attracted or discarded.

Nonetheless, it has not yet been pointed out what makes *anisonumeric* reconfiguration attractive in the first place. On a basic level, the fact that modules cannot be discarded or (re)introduced into the configuration at any time means that all modules in the initial configuration will have to move to a goal position in the final configuration, potentially having to traverse the entire configuration (and potentially wait for its turn to do so). Instead, through *anisonumeric* reconfiguration, a module that is too distant to where it is needed might instead be discarded, while another module is introduced right next to the position that needs to be filled. Naturally, this can only be beneficial if modules can be discarded and introduced at various locations, and it has a cost in terms of actual module units and energy. But, as **speed** is our critical parameter, that is a very interesting property.

For these reasons, we develop in this work a framework for *anisonumeric* reconfiguration, in the form of a **dedicated self-reconfiguration platform**, which we name **Sandbox**.

The *Sandbox* consists in a reserve of modules that is located underneath the reconfiguration scene, and that is able to introduce modules at various locations regularly placed on the ground of the reconfiguration scene (cf. Section 3.7).

One major advantage of having these locations for introducing/discarding modules regularly placed on the reconfiguration scene means that the resource allocation concerns of the self-reconfiguration can be segmented or *clustered* into areas of the goal configuration located around each of these *sandbox entry points* (cf. Figure 9). In other words, the initial and goal configurations can be discretized into areas that are located around (and above) the entry points, so that modules can self-organize in each area to transform this area from the initial configuration into the matching area from the goal configuration through displacement of configuration modules, feeding of modules of the sandbox, or the discarding of modules to the sandbox, depending on the local problem. Modules from one area almost never have to cross into a nearby area, thus easing coordination and further increasing the predictability of the self-reconfig-

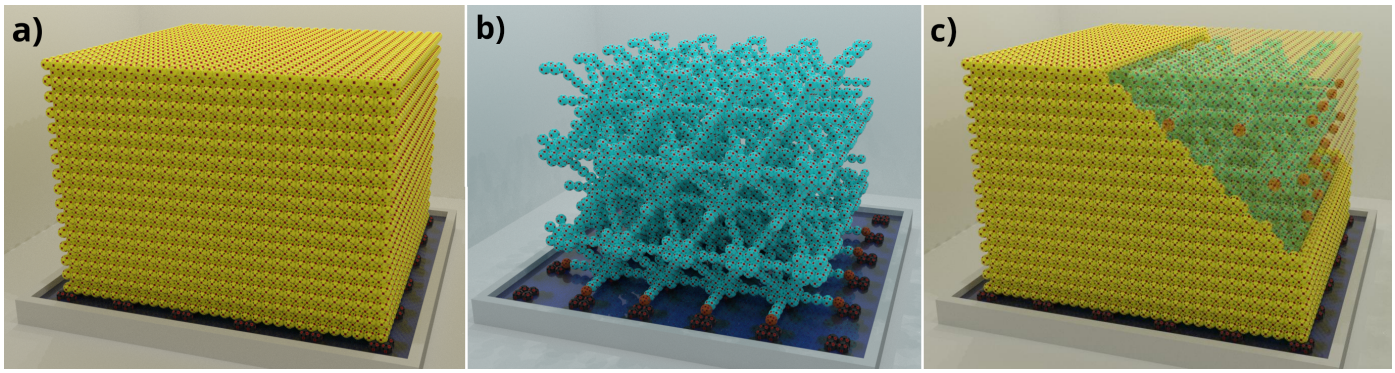

Figure 8: Side-by-side comparison of: **(a)** regular cube made of *3D Catoms*; **(b)** scaffold version of the object; **(c)** scaffold cube with added coating.

uration from the point of view of the modules.

Things really become exciting when using scaffolding in conjunction with the sandbox, however, as combining the two together enables unprecedented reconfiguration ease and speeds thanks to a multi-level **pipelining**: pipelining at the level of the shape thanks to the sandbox, where each part of the shape can be constructed in parallel once high-level construction rules are observed; and pipelining at the shape areas, thanks to the dedicated motion paths offered by the scaffold.

3.5. Relationship to the Self-Reconfiguration Literature

Our approach is somewhat conceptually similar to Dewey et al. (2008), where modules are arranged into regular multi-module units (*metamodules*, which can be in an empty state (only structural modules of the unit), or in a filled state (surplus of modules in the unit). Modules flow through the growing shape from filled metamodules to empty metamodules guided by a planner and achieve a completion time linear with the diameter of the ensemble. They did not address, however, the resource allocation aspect of the reconfiguration, that is to say how to decide on which part of the initial shape will fill each part of the goal shape, an inescapable and complex problem.

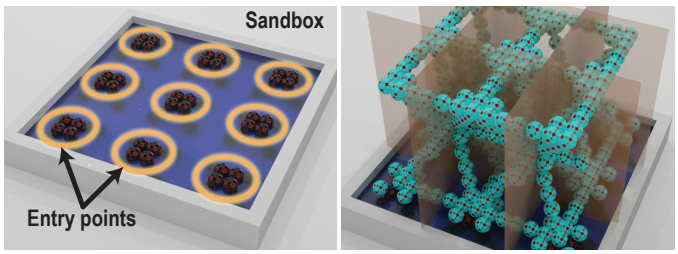

Figure 9: **(Left)** Overview of the sandbox, with the entry points used for supplying and discarding modules circled in orange. **(Right)** Scaffold of a cube of side 13 modules over the sandbox. Brown planes divide the object into several vertical areas. Scaffold modules from each area can be supplied exclusively through the sandbox entry points directly below them, enabling *pipelined* reconfiguration.

Furthermore, previous scaffolding approaches mentioned in previous paragraphs considered an initial shape as a prebuilt scaffold but none addressed how to construct the scaffolding structure from a mass of modules, which is the topic of this work. We are also, therefore, putting forward an original solution to a previously unstudied problem, as shape assembly work from the modular robotic literature is usually more concerned with the final latching of modules at specific locations than the planning of the motion that led them there Tucci et al. (2018), and classical self-reconfiguration approaches with massive ensembles generally transform a shape into another rather than build one from the ground up Dewey et al. (2008); Butler et al. (2004); Lengiewicz and Holobut (2019). Because of this, identifying bases of comparison for evaluating this work in regard to other assembly or self-reconfiguration solutions is arduous and the resulting findings might be inconclusive. As a matter of fact, this a much deeper problem in this line of work, as traditional (i.e., shape to shape) self-reconfiguration works are already afflicted by this evaluation conundrum, due to the variance in robotic models, capabilities, and modes of motion Thalamy et al. (2019b); Ahmadzadeh et al. (2016).

A comprehensive account of the structure (when relevant) of each of the proposed components is provided in the following sections.

3.6. Scaffolding and Structural Engineering

This section focuses on the anatomy and construction of the scaffolding structure introduced in the introduction.

As a reminder, we aim to build an internal scaffolding of a goal object as fast and efficiently as possible. This scaffold, which forms a sort of highly regular skeleton of an object, is composed of an arrangement of regular units sharing a common structure named scaffold **tiles**.

3.6.1. Structure of a Scaffold Tile

The scaffold tile is the parameterizable unit of the scaffold. All the tiles composing a *3D Catom* scaffold share a common geometry, but their exact structure can vary depending on the specific location of the tiles within the shape.

A tile consists of a number of components placed in an appropriate coordinate system, where $\overrightarrow{x}$ and $\overrightarrow{y}$ are classical orthogonal axes but where the vertical axis $\overrightarrow{z}$ is skewed and defined as $\overrightarrow{z} = (\frac{\sqrt{2}}{2}; \frac{\sqrt{2}}{2}; \frac{1}{2})$. These components are:

- A root module at the center of the tile, to which we will refer hereinafter as the **tile root** or simply R module (in white in Figure 10a).
- Two horizontal branches placed orthogonally across the $\overrightarrow{x}$ and $\overrightarrow{y}$ axes named the **X** and **Y** branches (in red and green in Figure 10b, respectively).
- Four upward branches ascending at a 45° angle and placed orthogonally to each other: the **Z** branch along the $\overrightarrow{z}$ axis, and the **RZ**, **RevZ**, and **LZ** branches at 90°, 180°, and 270° clockwise from Z (therefore following axes $(1, -1, 1)$, $(-1, -1, 1)$ and $(-1, 1, 1)$, respectively—all in light blue in Figure 10b.
- Four **support** modules: S_Z, S_{RevZ}, S_{LZ}, and S_{RZ}; one under each of the ascending branches at respective positions $(1, 1, 0)$, $(-1, -1, 0)$, $(-1, 1, 0)$, and $(1, -1, 0)$ relative to the tile root R. Supports are absolutely necessary for modules coming from below the tile so that they can traverse it vertically, as imposed by the *bridging constraint* (in yellow in Figure 10b).

3.6.2. Parameters and Conditional Structure

Let b be the parameter of the scaffold that defines the length of the branches of the tiles in number of modules. There is a lower bound on the value of b as under four modules in length tiles become too dense to allow module movement through all of their internal paths. Furthermore, an upper bound on the value of b is given by the mechanical strength of the connectors of the hardware *3D Catoms*. Although, the final prototype is not yet functional, physical simulations and preliminary experiments with the electrodes have shown a latching force able to carry more than 10 robots with one connector (cf. Piranda and Bourgeois (2016)). Varying the length of tile branches allows control on the resolution of the target shape and thus the speed of self-reconfiguration, as higher b values would result in less dense shapes with fewer modules to place but also might result in a lower fidelity for the details of the shape. Throughout this paper, we will assume $b = 6$ as the length of the branches, as it is a very reasonable value mechanically speaking.

When necessary, we will refer to a specific module of the tile with a name formed from its branch followed by its order within that branch (e.g., $RevZ_i$, where $i \in [1, b-1]$), or from S with the branch above it in subscript for **support** modules (e.g. S_{LZ}). Note that for any branch, the module of order 0 is always the **tile root**.

Furthermore, while b defines the *maximum* length l of the branch of a tile, a branch can have anywhere between 1 and b modules when part of the scaffold. A length of 1 means that the branch should not be grown for that tile, and only the tile root remains—tiles can therefore have a variable number of grown branches. A length of b means that the branch must be grown and it is likely that another tile will be grown from the tip of that branch once complete. A length anywhere between the two means that due to the geometry of the shape and placement of the tile within that shape, the full branch must not be grown and a child tile will not be grown from this branch.

To sum it all up, a tile always has a **root** module, and can grow between 0 and 6 branches, each between 2 (1 module + the **tile root**) and b modules long. Furthermore, **support** modules need only be present if the upward branch below it ingoing to

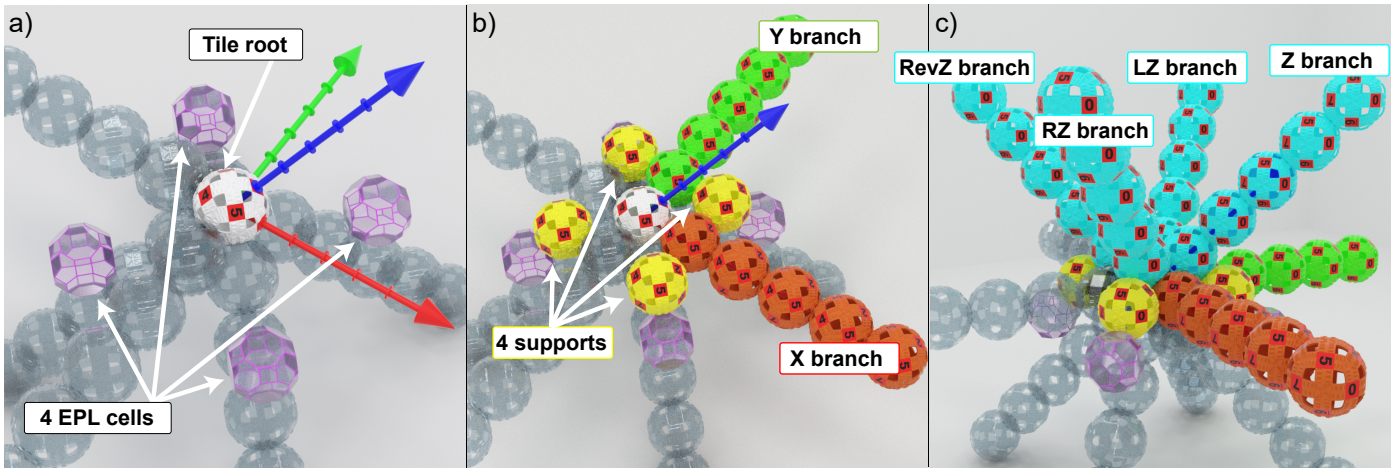

Figure 10: **Anatomy of a scaffold tile**: **(a) Tile root** and vertical entry point locations, ingoing branches from parent tiles in transparency; **(b) Supports** and outgoing horizontal branches; **(c)** Outgoing upward branches.

its tile has been grown. Thus, a full tile (with all branches grown), can have anywhere between 1 (the R module), and $1 + ((b-1) \times 6) + 4$ modules.

3.6.3. Connecting Tiles

Tiles assemble by connecting the tip of a fully grown branch (i.e., where $l = b$) to the tile root of another, as demonstrated in Figure 11.

Much like for the tile itself, we must enforce a construction order for the scaffold. This construction order follows the diagonal of the shape to be built. This means that the first tiles to be built (named **seed tiles**) will be on a corner of the base of the object, and the last one will be the one on the opposite corner of its top layer. We arbitrarily choose this corner as the one with minimal x and y coordinates. Therefore, the growth of the shape will by default (there are exceptions) proceed along the $\overrightarrow{x}$ and $\overrightarrow{y}$ axes for a given plane, and from bottom to top. Then we can say that a tile that has been built before another that is connected to it (i.e., a **neighbor tile**) is a **parent tile** of the latter — hence the other is a **child tile** of the *parent*. A tile usually has more than 1 parent and up to 6 (one for each outgoing branch) if all ingoing branches are grown. Parent tiles are responsible for the growth of their children tile by feeding them modules through the connecting upward branch. A tile will only start its construction once all incoming branches from parent tiles are complete. This synchronizes the construction of the shape according to our rule-based construction plan.

By generalization, we can generate a polytree, named **construction polytree**, representing the growth of a scaffold into a given object, where nodes are tiles and edges express construction precedence, with the seed tile as the root of the underlying tree (see Figure 12).

3.6.4. Entry Points into the Tile

Module navigation from one tile to another is supported by special positions around the base of each tile, named **Entry Point Locations** (**EPL** hereinafter). There are 4 EPL for a tile, one on each of the ingoing upward branches (see Figure 10, with entry points in transparent pink and ingoing branches in transparent blue). Entry points are located over the second-last module of the ingoing upward branches, and right below the support module for that branch, which guarantees the reachability of the higher portion of the tile.

Any module entering a tile will do so from one of the four EPL, that is to say, modules always flow through the scaffold from the lower tiles to the tiles above, and always do so through the connecting ascending branches—and therefore never through the horizontal branches, except strictly within a particular tile during its construction. What motivates this mode of operation is that it severely limits the number of possible intersecting paths along the scaffold, which lowers the risks of motion disturbance between modules and eases coordination. Entry points also have a crucial functional role to play in module navigation across the tiles and scaffold as a whole, which will be addressed later on.

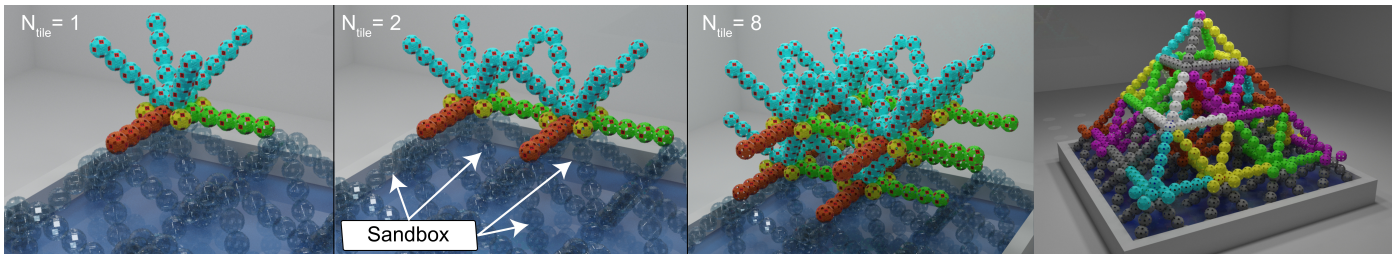

Figure 11: **Anatomy of the entire scaffold:** (Left) Breakdown of a sample scaffold consisting of an arrangement of 8 tiles with all branches grown, directly over the sandbox (branches from sandbox tiles in transparency). (Right) Scaffold of a pyramid with each tile highlighted in a different color.

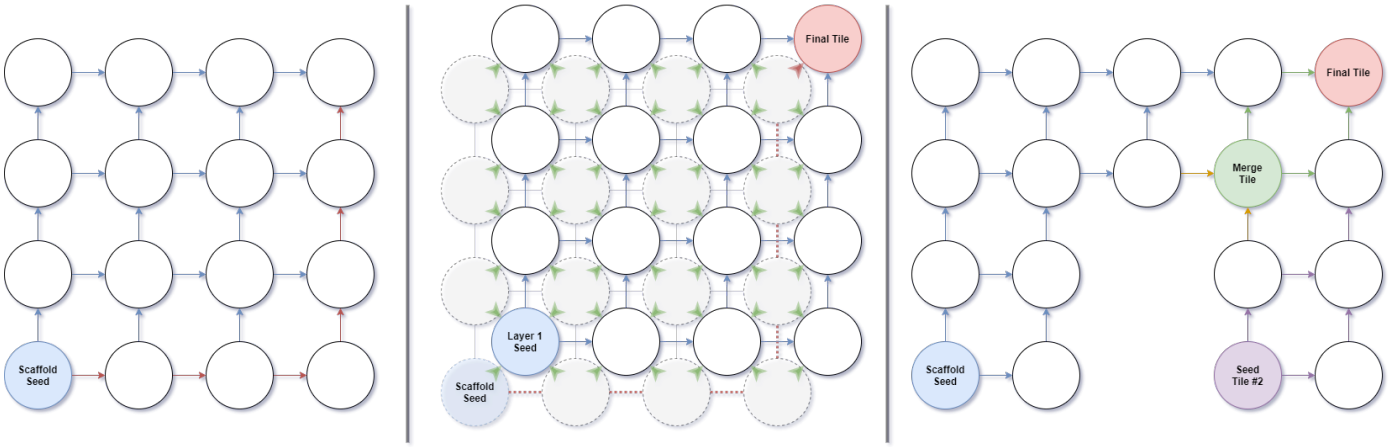

Figure 12: Diagram of the **construction polytree** of a $4 \times 4 \times 2$-tile cube. **(Left)** Bottom tile layer; **(Center)** Top tile layer, with green arrows the edges between the bottom and above layer. Red edges highlight a possible **critical path**. **(Right)** Polytree of a scaffold with multiple seed tiles.

As a consequence, a tile will have a maximum of four incoming flows of modules, which is the number of different usable paths leading to it. One of the main challenges is hence to coordinate these flows of modules such that they cannot intersect and impinge on each other's courses.

Note that for the generalization of this construction methods to all morphologies of scaffolds, a topic touched on in Thalamy et al. (2020b), it is possible that tiles are fed through horizontal branches, but this can only happen if they have no ingoing vertical branches because of concavities in the shape of the scaffold.

3.7. A Dedicated Self-Reconfiguration Platform

This section briefly discussed the **sandbox** — our dedicated self-reconfiguration platform that supports and manages the supply and withdrawal of modules to and from the reconfiguring ensemble. The nature and features have been discussed in the introduction. The design of such system will require considerable future research work, which unfortunately cannot fit into this paper. Therefore, it is impossible to present the exact design and structure of the sandbox at this point. We have, however, imagined that the sandbox could be structured internally exactly as the scaffold (with the same parameter b) and contains a surplus of modules along its branches, which can then be called in for the reconfiguration above. Or have this sandbox scaffold connected to a mass of modules that can climb onto the scaffold and into the reconfiguration scene. For the purpose of this work, we presently assume that the top of the sandbox shares the same structure as the scaffold, and consists in fully grown scaffold tiles and branches meeting at the ground level, providing platforms for starting new tiles with 4 incoming branches (and thus feeding paths) to each platform (see Figure 11).

This work focuses on the coordinated construction of the scaffold of a shape from an ordered reserve of modules rather than on the transformation of a prebuilt shape into another, which will be further addressed. Therefore, our initial state is an empty reconfiguration scene, and all the modules taking part

to the reconfiguration will have to be introduced to the growing shape through one of the ground tiles at the top of the sandbox. The reconfiguring modules will remain connected to the modules from the sandbox at all times (as per the *connectivity constraint*), thus sandbox modules can be attracted onto the reconfiguration scene thanks to messages propagated through the scaffold, in the same way that modules are attracted to various parts of scaffold in our reconfiguration algorithms presented in the next Section.

3.8. Visual Aspect Preservation Through Coating

As stated in the introduction, we propose to compensate the negative impact of scaffolding on the visual aspect of objects by covering the surface of the porous objects formed by the scaffolding with a single layer of modules. We call this process **coating**. This scaffold and coating method can be seen as a special case of self-reconfiguration, that takes place among obstacles (the scaffold, constraining the motions and assembly of modules), and from a reserve of modules,

This section introduces the coating problem and the challenges it poses in a face-centered cubic (FCC) lattice. It shows how a coating can be designed in this context, before we provide a straightforward algorithmic solution in Thalamy et al. (2020a).

While we have found interesting solutions for efficiently constructing the interior of objects, we have yet to implement a coating algorithm that reaches the same level of parallelism as our scaffolding algorithms — the coating is hence the current limiting factor of our method, requiring further research. However, we are interested in this paper in showing that even with a relatively inefficient coating method, using a coated scaffold may well be preferable to building the equivalent dense shape.

Given a prebuilt scaffold structure made of *3D Catom* modules in a 3D lattice environment and a description of it, coating consists in covering the surface of the shape with *3D Catom* modules such that the object appears solid while taking advantage of the mechanical stability provided by the scaffold itself.

Then, given the geometry of the *3D Catoms*, covering the surface of this scaffold using a single layer of modules would suffice to make the object appear solid. In that context, *solid* means that it would appear to be filled with matter instead of being hollow, hence providing high fidelity to the object that is being represented.

The scaffold provides an internal structure to an object for better mechanical stability. There are several ways that a coating can be devised for a given scaffold, which relates to the amount of contact between the modules of the surface of the scaffold and those of the coating layer. This can be represented on a spectrum, with a *loose coating* on one end, and a *tight coating* on the other. In that case, a tight coating means that the coating is made such that it fits to the scaffold as closely as possible, and thus provides the highest number of contact points between the surface of the scaffold and the coating layer, which yields to a maximal structural strength. A tight coating is, however, dramatically more difficult to achieve than the alternatives (intractable even), as it is essentially a case of reconfiguration among obstacles, which greatly constrains the possible assembly order of the coating, as numerous deadlocks could be created by unreachable cells between the growing coating and the scaffold structure itself in the case of the FCC lattice. On the other end of this spectrum, a completely loose coating is always at a distance from the scaffolding surface and thus provides no contact points and structural benefits (indeed, the scaffold itself adds no value at all in such case), but greatly relaxes the constraints imposed upon the construction of the coating, as it can be done in isolation from the scaffold. In this work, we propose a middle ground between these two options, based on a loose coating, but with added contact points between the scaffold and the coating layer.

The contact points (or *structural supports*, not to be confused with the *Support* modules in scaffold tiles) are closely linked to the scaffold itself and its b parameter, as the supports are obtained by extending the external horizontal branches of border tiles by one module (see Figure 13.b, thus closing the gap between the coating and the scaffold at various points of horizontal layers every b modules in height. These modules not added during scaffold construction but right before starting the coating for their layer.

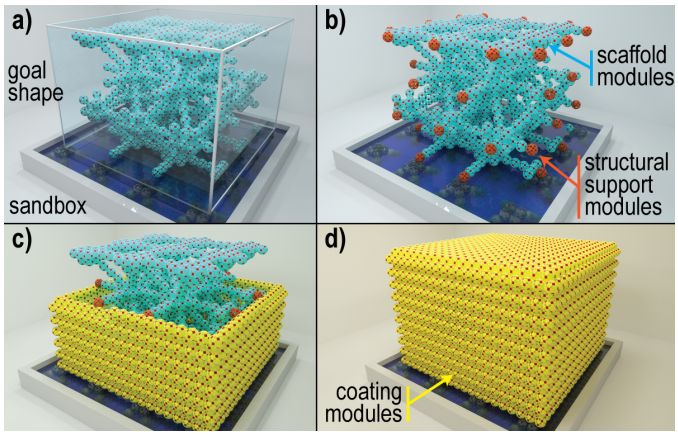

Figure 13: **(a)** Scaffold of a cube of size $20 \times 20 \times 20$ modules, with highlighted target volume; **(b)** scaffold with horizontal branches extended into structural supports; **(c)** snapshot of the coating phase; **(d)** fully assembled coating of a cube, with scaffold inside.

4. Construction Algorithms and Evaluation

In this section, we report on the various scaffold and coating construction algorithms that we have designed and published, in order to give an account of the performance of current implementations of our approach.

We will sometimes be referring to a class of shapes named *semi-convex* shapes. These shapes are a subclass of convex shapes where all branches of a scaffold are vertically connected to the sandbox below, without any hole between them and the sandbox. In other words, the union of the scaffold and the sandbox must be convex. This is a simplification that yields the highest possible throughput of modules through the scaffold, as modules are instructed to always flow vertically in the structures as trains of modules and without never intersecting the path of another. This can be seen as the optimal *pipelining*.

4.1. Scaffold Construction Algorithm

Our scaffold construction method (Please refer to Thalamy et al. (2020b) for an in-depth description of the algorithm) aims to construct a goal scaffold from an empty sandbox. In that sense, this is not yet self-reconfiguration from one scaffold to another, but the scaffold construction algorithm is a necessary primitive that can be used as part of scaffold-to-scaffold self-reconfiguration. Our scaffold construction algorithm uses simple geometric rules to determine an assembly order for the tiles of the scaffold so that no deadlock can occur during construction. This requires all tiles to act as synchronization points for the construction process, by ensuring that the construction of a tile is not started until its local dependencies with adjacent tiles are met. Then, at a lower level, this distributed algorithm coordinates the modules arriving from the sandbox by using message exchanges between modules acting as local *coordinators* and incoming modules, to route them to where they are needed, and ensure that construction of the tiles themselves is done according to a feasible construction plan. Finally, modules navigate the scaffold using local motion rules that describe how to explore a scaffold tile without running into obstacles, and a message-based motion coordination protocol ensures that modules to not collide into each other when moving by leaving free space between modules along the same path. This motion coordination protocol also provides robustness to stochastic variations in the rotation time of modules, which also makes the overall algorithm asynchronous.

This scaffold construction algorithm was first introduced in Thalamy et al. (2019a) and consisted in a synchronous version of the algorithm lacking the motion coordination protocol. It could only build pyramidal scaffolds, and had to request modules from the sandbox every time modules were needed, which produced a lag during construction, resulting in an $O(N^{\frac{2}{3}})$ reconfiguration time and using $O(N^{\frac{4}{3}})$ message exchanges, with N the number of modules in the pyramid.

We later introduced the motion coordination protocol in Thalamy et al. (2019c), making the construction asynchronous and more robust. We also changed the request-based strategy from the first version of the algorithm to a *continuous flow* of modules from the sandbox and until the scaffold, and that could be interrupted using messaging when modules were not needed anymore — much like closing an open faucet when the bottle is full. This resulted in an improvement of the reconfiguration time which is now $O(N^{\frac{1}{3}}) = O(\sqrt[3]{N})$ for pyramidal scaffolds.

Then, we generalized the method to all *semi-convex* shapes in Thalamy et al. (2020b). We have analysed the scaffold construction algorithm through the case study of the construction of a cube of dimension h scaffold tiles. We have showed that for any

scaffold, there exists a critical path in the scaffold, made of all tile branches that are last to arrive to a synchronization, on which the reconfiguration time depends. Then we demonstrated that the length of this path is linear in the dimension of the cube in the dimensions of the cube, that is to say that its length is $O(h)$, and we able to conclude that the reconfiguration time for the cube in number of timesteps (where a timestep is the average duration of a module motion), we proved that the reconfiguration time of the cube of dimension h was also $O(h)$.

Then, by analysing the number of modules contained in a cube of dimensions h, we have showed that the number of modules in a h-cube was $O(h^3)$. From there, we could therefore deduce that the reconfiguration time of the construction of the scaffold was $O(\sqrt[3]{N})$, where N is the number of modules in a cube of dimensions h.

Finally, this reconfiguration time can be generalized to all *semi-convex* shapes by showing that any semi-convex scaffold can be enclosed in an h-cube of higher dimension, thus yielding an $O(\sqrt[3]{N})$ reconfiguration for all shapes of this class. This means that scaffold construction can run in **sublinear** reconfiguration time, which is immensely faster than what has been achieved until now in traditional self-reconfiguration. These results were also confirmed in the simulations of the construction of many different scaffolds with up to 32,000 modules.

By analysing the parallelism and convergence rate of simulated scaffold constructions, we have showed that scaffold construction consists in three successive phases:

- **Load** phase: This is the start of the construction of the scaffold, from one or several seed tiles placed in ground corners of the shape. Here, the construction propagates diagonally through the surface of the base of the object, connecting sandbox entry points to existing tiles of the scaffold, until the entire ground surface of the scaffold is connected to the sandbox.
- **Cruise** phase: This is the main phase of the construction, where modules can vertically flow through the volume of the scaffold from the sandbox, and during which the construction of the shapes takes place across the entire volume in parallel, thus producing a constant reconfiguration speed.
- **Finish** phase: This is a decreasing speed phase that comes at the end of the construction, and during which parts of the structure are being gradually completed, hence stopping the flow of modules from the sandbox in parts of the shape, until the whole scaffold is finished.

This same three-phase convergence rate can be observed on all semi-convex shapes. We hint in Thalamy et al. (2020b) to how the current scaffold construction algorithm could be generalized to all shapes (not just *semi-convex* ones) by enriching the model of a tile with downward vertical branches and additional entry point locations around the tile root of the tile. The newly added vertical branches would allow for tiles to supply modules to children tiles below them that are not vertically connected to the sandbox, and the entry point locations would enable the exchange of modules between horizontally connected tiles from the same tile layer. It is unlikely that the current level of performance will remain unaffected however, as module flows will now have to be split, thus reducing the module throughput on parts of the structure.

4.2. *Coating Self-Assembly Algorithm*

Regarding the coating of constructed scaffold, the main problem that we have addressed in Thalamy et al. (2020a), is the one of finding a correct self-assembly plan in order to construct the coating around a scaffold without regard to the motion of the modules from the sandbox to the positions that need to be filled. Self-assembly planning is a notoriously complex problem due to the difficulty of deadlock avoidance and necessary coordination between dependent modules that are sometimes wide apart from a networking point a view. Tucci et al. (2018) proposed an algorithm using simple rules and communication to assemble 2D or 3D objects made from *3D Catoms* in an environment with no obstacles. Our current self-assembly approach relies heavily on this *Tucci algorithm*, assembling the coating of scaffolds in a bottom-up approach, layer-by-layer. On each layer a module on the border of the coating is chosen as

the *seed* for the current layer according to its coordinates, and the Tucci algorithm is then applied to attract this layer. This works well for all layers do not have *structural supports* presents (cf. Section 3.8), but as structural supports can create obstacles in the construction of the coating, the Tucci algorithm cannot always be applied on *support layers*. In that case we use an algorithm of our own, named *border completion algorithm*, to attract the support and modules whose position depend on it, before filling the remaining positions of the border in a circular fashion around the object.

The Tucci algorithm and our border completion algorithm are able to assemble the coating in linear time in the number of modules in the coating. This is a good result but the construction of scaffolds is currently much more efficient than their coating, due to the low parallelism of the current layer-by-layer method on vertical borders. Furthermore, the current assembly method is not suitable with shapes with overhangs or bowl-like concavities. Together, these elements hint that better self-assembly need to be designed with parallelism in mind. This is further discussed in Section 5.

4.3. Overall Approach

Regardless of the coating assembly method being used, by comparing the number of modules in dense shapes and the coated scaffold version of these same shapes (thus resulting in the exact same external aspect of the shapes), we find that total number of modules saved using our method is between **60% to 80% saved modules** Thalamy et al. (2020a). The bigger the volume of the shape that one is trying to represent, the bigger the gain in saved modules, so extremely large shapes could in theory even use only 10% of the number of modules in their dense counterpart. This is a massive gain that is not just about saving the amount of matter user, but also time itself, as the reconfiguration time is a function of the number of modules in the goal shape. Fewer modules that need to be displaced mean a faster reconfiguration.

Accordingly, our approach to reconfiguration not only is asymptotically more efficient than traditional self-reconfiguration, thanks to its **sublinear reconfiguration time**, but the speed-up is even greater thanks to its **dramatically reduced module usage**.

A video presenting the main steps of the construction of the scaffold and the coating is available online[3]. This video also shows the construction of many final 3D shapes, especially a large set made of more than 35,000 *3D Catoms*.

5. Discussion

5.1. Improving Coating

Discarded Coating Strategies. A number of alternative coating strategies were investigated while researching this work, and we explain why they have not made the cut below.

We have not mentioned a strict top-down coating approach in the previous section, because it does not appear like a good strategy in the general case, since modules would have to climb through the scaffold and then escape it to reach the coating, but as the coating would assemble, the coating itself could then prevent modules from escaping the scaffold. This is why we had to resort to using the coating itself as a path for dispatching modules.

An open question that remains is whether the construction of the coating can be done in parallel with the construction of the scaffold. This also has the effect of blocking many scaffold paths for the same reason, thus decreasing the parallelism of the method, and hindering the geometric separation of concern enabled by pipelining, thus making planning more complex or perhaps even impossible.

Limits of the Current Method. The current simplistic bottom-up coating method is well suited to build the coating of scaffolds that belong to the *semi-convex* geometric class. Furthermore, while we have not stated it explicitly until now, the class of object geometries supported by our coating method is in fact larger than *semi-convex* shapes alone. Indeed, as shown in Thalamy et al. (2020a) our algorithm was able to build the coating of the scaffold of a chair, which does not belong to the *semi-convex* class as the seat of the chair creates a concavity with regard to the sandbox.

As the current coating method relies on a *bottom-up layering* approach, it can build all shapes where

[3] On line video presentation of the algorithms: https://youtu.be/jrtuaBLXkQI.

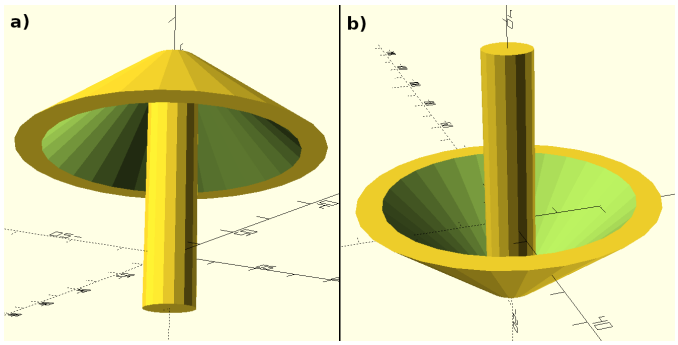

Figure 14: Two problematic shapes given a *bottom-up* coating strategy: **(a)** Mushroom-like shape with overhang; **(b)** Spinning-top shape with a bowl-like concavity.

any disjoint portion of a coating layer is directly adjacent (or connected) to the layer below. Otherwise this would be akin to the construction of an unconnected overhang in 3D printing. In our chair example, even though the four legs are disjoint coating parts, they all merge into the underside of the seat, which is connected to them, so our method operates with no hurdle. Shapes with overhangs or bowl-like concavities such as the ones shown in Figure 14 are therefore limit cases.

Replacing the strict *bottom-up* layer-by-layer approach by a construction following the surface of the object appears promising, as any coating portion will be supported by the one built before itself. Another strategy could be to build temporary scaffolds to supply modules to hard-to-reach coating locations.

Towards More Efficient Coating Methods. The current version of the algorithm might be too slow at reconfiguration time for some minor changes into the configuration, as would be required to first disassemble the coating, modify the scaffold, and then reassemble the coating. Otherwise, minor adjustments could be made simply by adding coating modules, and only when surpassing a certain threshold could the scaffold be also altered, thus saving precious time.

While our current method of coating is quite straightforward and by itself leads to the construction of a coating in linear time, it is obviously unsatisfactory that only wide or planar borders can achieve high parallelism. An ideal coating solution would build **any** layer with some degree of parallelism, truly leveraging the benefits of the sandbox and multiple sources of modules.

A promising parallel and multilayer solution could be to attract a module to a coating position as soon as all its neighbor positions that are in the previous layer are filled (and, of course, its horizontal dependencies too). In that way, the construction of the coating can also proceed in a vertical diagonal manner, building multiple planes at once. However, special rules would have to be designed regarding the introduction of support modules so that they do not hamper the construction. In a sense, this would be equivalent to extending the multilayer version of the Tucci algorithm to support the presence of obstacles.

Numerous sources of modules would however be needed to dispatch the modules optimally, and the concurrent planning of the motion of the modules from their source to their destination becomes non-trivial if there are fewer sources than coating sections that can be built in parallel.

Module Dispatch From the Sandbox. The coating algorithm introduced in this paper only addressed the assembly strategy of the coating, that is to say the scheduling of the construction, without regard to the actual dispatch of modules from the sandbox. Therefore, we touch on how modules can be routed from the sandbox to open positions within the shape using a gradient system below.

We have started studying a self-reconfiguration planning method based on attraction gradients emitted by modules adjacent to coating positions that is ready to be filled. This gradient is propagated through all the modules from the layer. Assuming there is a single source of modules from the sandbox to the coating, modules then move in a train-like fashion until reaching the coating and then simply follow the existing coating from the current layer being built. Every time they plan a motion, they probe their neighbors that are already inside the coating layer for the attraction gradient values (essentially a pair of positions and Manhattan distances), and pick the destination with the smallest distance. The gradients are invalidated when a module reaches an attraction position.

While this gradient is not very important on simple borders with a thickness of only one module, as on these borders modules could have just followed the borders until a free and ready spot is reached, it is a lot more important on more complex borders. For

large borders or entire planes of coating, the assembly order of the coating is non-trivial and many positions can be ready to receive a module at the same time. This thus requires some way of prioritizing the construction and dispatching the modules to the closest available coating positions — this is where the attraction gradient and gradient descent come in.

The major challenge of this approach, however, is to deal with sudden changes in the gradients that could make modules abruptly change direction, which might trigger collisions and other coordination issues. This problem will become even more salient when considering multiple sources of modules. This motion aspect of the coating problem will require more research, which existing work on self-reconfiguration and distributed motion planning can certainly guide.

5.2. *Reflection on the 3D Catom Model and Constraints*

We have seen in Section 2 that the *3D Catom* model had a geometry that produces *3D Catom* ensembles with an FCC lattice structure. As it turns out, this is one of the densest possible arrangements of matter. Coupled with the quasi-spherical geometry of *3D Catom*, this can create a very detailed representation of objects compared to other geometries, with smooth curves — recall Figure 1.

The high density of this FCC lattice geometry combined with the non-deformable nature of *3D Catoms* pose nevertheless serious complications with regard to the motion constraints of modules as introduced in Section 2.2. The *bridging constraint* may be the most infamous of these constraints due to its numerous and far-reaching effects.

We will see below that such constraint can be a serious impediment to both parallelism, robustness, and emergence in self-reconfiguration applications, limiting the range of practical solutions to self-reconfiguration problems.

We had the intuition that concave shapes were a problem early on as their construction would limit pipelining when building from a sandbox. For that reason, we investigated using a temporary scaffolding that would fill the concavities of the target shape during construction, and that would then be deconstructed at the end of self-reconfiguration. This required more modules than just building the scaffold (though less than the dense shape in most cases), but provided the highest reconfiguration speed throughout the structure and, we believe, supported the mechanical stability of intermediate configurations during self-reconfiguration. This is illustrated with the yellow temporary scaffold in Figure 15. Sadly, because of the bridging constraint, the temporary scaffold could never be deconstructed, and this promising idea had to be thrown away.

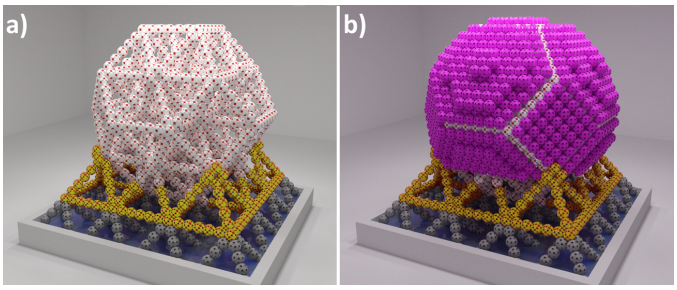

Figure 15: Original self-reconfiguration pipeline idea (artistic impression): using a temporary scaffold (yellow modules) both for mechanically supporting the growing structure and removing concavities during construction.

Furthermore, as the bridging constraint imposes a single direction of construction, this prevents us from starting the construction process from multiple opposite directions, which could further speed up self-reconfiguration, even though it would be harder to coordinate the construction.

The bridging constraint also limits the robustness of *3D Catom* ensembles, as it would prevent the extraction of a faulty module on the scaffold without further deconstruction, or the later insertion a replacement module. Fortunately robustness mechanisms can be added to our method to delegate the functions of the faulty modules, or restore communication links of the scaffold by attracting additional modules.

A deformable catom has in fact been introduced in the form of the *Datom* Piranda and Bourgeois (2020), and whether our self-reconfiguration method can be further improved without the bridging constraint or not is an interesting question.

We have also seen that the bridging constraint is not the only limit of the *3D Catom* hardware, as two problems, the *motion coordination challenge* and *remote blocking conundrum* were a consequence of the limited range of communications of *3D Catoms*. We

have already discussed in Section 2.2 the potentially huge communication overhead that would come with every module motion to satisfy these constraints with only communication between neighbor modules. Building modules with a wider range of communication or at least some sort of extended neighborhood sensing would produce a much more powerful model for self-reconfiguration.

Deformability and communication capabilities are, therefore, important variables that roboticists must take into account when designing modular robotic systems, and this process of requirement feedback between roboticists and computer scientists is crucial for achieving more practical modular robotic applications, as discussed in Thalamy et al. (2019b).

5.3. The Elephants in the Sandbox

Ground and Underside Coating. One particularly problematic aspect of object representation in a sandbox (and under our current model constrained under the bridging constraint) that has remained unaddressed up to this point is the fact that the underside of the object cannot be coated — and in fact remains tethered to the sandbox which might prevent manipulation by the user. While the sandbox could provide a way to disable the connection between the sandbox and the scaffold on demand so that the object might be manipulated (which would require an alternative power source if the matter is to remain active), adding the coating to the underside of the object seems harder to perform. The coating of the underside of the object cannot be performed according to the same process as the rest of the shape for two reasons: (i) because of the bridging constraint, it would not be possible to add all modules between the horizontal branches of the ground layer of the object (or between the sandbox branches if coating one layer below the scaffold instead); (ii) even if there was a way to do it, this would prevent any additional module from entering the scaffold from the sandbox, so this would have to be the last process of the construction to take place, or the first of the deconstruction. The only realistic way to perform that coating of the underside of the object that we have been able to conjure would be to manually rotate the object on the sandbox and then perform the coating of the underside. Needless to say, this is highly undesirable as this requires an external intervention, which breaks autonomy. Changing the model to get rid of the bridging constraint could be a lot more desirable if the functional trade-off is not too bad.

Nonetheless, for the purpose of a 3D multi-sensory display system based on programmable matter, it might not be necessary to allow the represented scene to be detached from its substrate and manipulated by hand. Such a display without the detachability features would already be quite sufficient for providing unprecedented sensory experiences to users.

Object Fidelity and Special Cases. On the topic of fidelity to the target object, and besides the absence of an underside coating, objects built using our method will in most case appear identical to their dense counterpart. When using a scaffold, however, the base unit of the shape changes from a 1-voxel module to a multi-voxel scaffold tile. This means that the scaffold is at best an approximation of the target shape or a lower-resolution version. In contrast, the coating preserves the original resolution of the object and is thus a great addition to the scaffold as it restores the original apparent resolution of the target shape. Nonetheless, there are several caveats of our method that need to be kept in mind when building objects in this way:

- Very small objects (smaller or in the range of the dimensions of a scaffold tile) will have little to no internal structure, only coating, as there would not be enough internal space to fit a scaffold. The resulting object would be even closer in density to its dense version. Furthermore, the downsides of our method might be greater than the benefits in this context, and it is possible that a classic self-reconfiguration would be more appropriate.
- Very detailed objects, or at least objects with very small details will have many substructures with low internal structure (a partial absence of scaffold) and only coating. Much like in the previous item, the scaffold will approximate the shape, ignoring details smaller than a tile (though in most cases partial tiles will be built, but this is highly contextual), and the coating will then restore these details.

- In both previous cases, it remains to be determined what kind of mechanical stress the partial or total absence of scaffold would put on the overall structure. Regarding the former scenario, it is likely that with such small objects do not really require an internal scaffold since it can be expected to be much more mechanically stable due to its size. A mechanical validation would be necessary to define the presence of a scaffold and if so to which extend.

Mechanical Validation of our Approach. The most critical aspect of our approach that is missing from this research is a mechanical validation of the scaffold and coating. It is not sufficient that the use of scaffolding appears mechanically equivalent, or at least mechanically stable enough for *3D Catom*-based programmable matter, this has to be mathematically and physically proven. This depends, however on the expected weight of future hardware *3D Catoms* and the size of their electrostatic actuators, but fortunately, the parameterizable nature of the scaffold with its tile branch length b could give us flexibility in designing the most stable and compact scaffold possible. We showed it is possible to use a distributed algorithm to assess if the configuration is mechanically sound regarding stability and breakage conditions Piranda et al. (2020). This information can also be part of the assembly constraints as it is showed in Pescher et al. (2020). As soon as the *3D Catoms* will be finalized, measurements will provide the data to feed these algorithms, answering the question possible and impossible structures. However, for big configurations, a simplification or an approximation will need to be done in order to reduce the computation time.

5.4. *From Scaffold Construction to Scaffold Self-Reconfiguration*

Finally, while this work has laid the foundations for a new approach to selfreconfiguration, no actual shape-to-shape reconfiguration have has been produced yet. Therefore, we discuss in this section how this scaffold construction and coating framework can be extended to shape-to-shape reconfiguration.

It must be first noted that there is in principle no reason that would prevent our scaffold construction method, from being reversible — the exact inverse process of the construction of the scaffold can be used to deconstruct it, within the same algorithmic bounds. Similarly, while the exact reverse of our coating algorithm would require new primitives to replace the Tucci assembly routine, an equivalent top-down disassembly strategy in linear time should also in principle exist. Entirely deconstructing the scaffold during reconfiguration seems like a redundant and inefficient approach. An advantage of the scaffold is that internal structure of any object we could build is similar. Thanks to that, efficiently self-reconfiguring from a shape $\mathcal{I}$ to a shape $\mathcal{G}$ could be the following process:

1. **Coating removal from $\mathcal{I}$**
2. **Computing an optimal overlap of $\mathcal{I}$ and $\mathcal{G}$**
3. **Scaffold disassembly of non-overlapping portions of $\mathcal{I}$**
4. **Scaffold assembly of non-overlapping portions of $\mathcal{G}$**
5. **Coating of the scaffold of $\mathcal{G}$**

In terms of complexity, this would be in the worst-case equivalent to running our entire construction pipeline twice in a row (once for disassembly, once for reassembly), and thus would preserve of the $O(\sqrt[3]{N})$ + $O(N)$ complexity of the construction phase alone.

The scaffold-to-scaffold self-reconfiguration problem can be thought of as a *resource allocation problem*, i.e., finding the optimal flow of modules between the sandbox, areas that must be grown, and areas that must be discarded. Resource allocation is likely to proceed both by exchanges between the initial and goal shapes as well as between the sandbox and the goal shape. It would also allow for reconfiguration between shapes with different cardinalities, another previously unstudied aspect of the self-reconfiguration problem.

6. Perspectives

In this final section, we compile a list of perspectives for improving our work and pushing the current limits of modular robotics systems and self-reconfiguration even further.

6.1. Our Approach

Regarding our self-reconfiguration based on sandboxing and scaffolding, here is what must to be done for our approach to be able to fully materialize:

Generalization, Improvements, and Self-Reconfiguration.

- Regarding the current results, there are a number of optimizations that could be explored, including prioritizing the construction of branches that are connected to children tiles in the construction scheduling of the tile, as well as investigating the impact of displacing the seed tiles at the center of the shape, which would lead to a centrifugal growth and might help increase the parallelism of the method even further, though this is unlikely to improve on the current cubic root worst-case reconfiguration time.
- Implementing the generalized version of our scaffold-construction algorithm, as hinted in Thalamy et al. (2020b), and creating a taxonomy of scaffold geometries with a mathematical analysis of the expected worst-case reconfiguration time for each class of shapes.
- Generalization is likely to require improvements on the current local-rule-based local motion planning solution, with the rules required for generalization becoming too numerous, potentially filling the scarce memory of modules, and rendering the hand design of rules laborious and troublesome. A more systematic and robust approach is thus needed. This could either be replaced by an alternative and better-suited motion planning method or benefit from improvements in design and compactness. It would be good to also prove the convergence and universality of the method once this is done.
- Design a coating assembly method that can coat any shape or scaffold, preferably achieving a high degree of parallelism in the construction of the scaffold. Accordingly, researching an adequate motion planning and coordination algorithm for implementing this assembly plan using sandbox-fed *3D Catoms*.
- Extending our method to shape-to-shape (or rather scaffold-to-scaffold) self-reconfiguration, with the goal to minimize the amount of matter that has to be displaced. Once this is done, perform a thorough comparison of the results of this method to state-of-the-art self-reconfiguration algorithms, studying metrics such as reconfiguration time, number of individual module motions, communication volume, raw matter usage (number of modules), and robustness.
- It would also be useful to study how this approach can be adapted to other models and in systems that reside in different lattices. This would maximize the usefulness of this approach, and perhaps show that it is preferable with some models but not others. Accordingly, it would be interesting to research whether the method can be made even more parallel and efficient when considering FCC lattice modular robots that do not have a bridging constraint such as the *Datom* Piranda and Bourgeois (2020), and how a coating can be built under these assumptions.

Feasibility.

- A mechanical validation of our scaffold model and a study of the stability of a scaffold-based object made of *3D Catom* compared to its dense counterpart.
- Accordingly, determining the range of realistic values for the parameter b of our scaffold.
- Engineering research on the feasibility and design of a sandbox system has described in our work.

6.2. General Perspectives

Regarding the general self-reconfiguration problem, researching the *overlap problem* between the initial and goal shape, and studying the impact of these hyper-parameters and others on the ensuing self-reconfiguration are essential. We have also provided in Thalamy et al. (2019b) a number of guidelines and perspectives for the field of 3D self-reconfiguration algorithms that we hope will be helpful to guide future research. Nonetheless, we would like to reiterate here that in order to accelerate progress

the field will need a real benchmark to test out and compare self-reconfiguration algorithms, as it is still extremely tricky to produce meaningful comparisons between published research works on the topic.

Acknowledgment

This work was partially supported by the ANR (ANR-16-CE33-0022-02), the French Investissements d'Avenir program, the ISITE-BFC project (ANR-15-IDEX-03), and the EIPHI Graduate School (contract ANR-17-EURE-0002).

References

Ahmadzadeh, H., Masehian, E., Asadpour, M., 2016. Modular Robotic Systems: Characteristics and Applications. Journal of Intelligent & Robotic Systems 81, 317–357. URL: https://doi.org/10.1007/s10846-015-0237-8, doi:DOI: 10.1007/s10846-015-0237-8.

Barraquand, J., Latombe, J.C., 1991. Robot Motion Planning: A Distributed Representation Approach. The International Journal of Robotics Research 10, 628–649. URL: https://doi.org/10.1177/027836499101000604, doi:DOI: 10.1177/027836499101000604.

Bie, D., Wang, Y., Zhang, Y., Liu, C., zhao, J., Zhu, Y., 2018. Parametric L-systems-based modeling self-reconfiguration of modular robots in obstacle environments. International Journal of Advanced Robotic Systems 15, 1729881418754477. URL: https://doi.org/10.1177/1729881418754477, doi:DOI: 10.1177/1729881418754477.

Bourgeois, J., Goldstein, S., 2012. Distributed intelligent mems: Progresses and perspectives. ICT Innovations 2011 , 15–25URL: http://link.springer.com/content/pdf/10.1007/978-3-642-28664-3.pdf#page=27.

Bourgeois, J., Piranda, B., Naz, A., Boillot, N., Mabed, H., Dhoutaut, D., Tucci, T., Lakhlef, H., 2016. Programmable matter as a cyber-physical conjugation, in: Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on, IEEE. pp. 002942–002947. URL: http://ieeexplore.ieee.org/document/7844687/, doi:DOI: 10.1109/SMC.2016.7844687.

Butler, Z., Kotay, K., Rus, D., Tomita, K., 2004. Generic Decentralized Control for Lattice-Based Self-Reconfigurable Robots. The International Journal of Robotics Research 23, 919–937. URL: http://journals.sagepub.com/doi/10.1177/0278364904044409, doi:DOI: 10.1177/0278364904044409.

Daymude, J.J., Richa, A.W., Weber, J.W., 2020. Bio-Inspired Energy Distribution for Programmable Matter. arXiv:2007.04377 [cs] URL: http://arxiv.org/abs/2007.04377. arXiv: 2007.04377.

Dewey, D.J., Ashley-Rollman, M.P., Rosa, M.D., Goldstein, S.C., Mowry, T.C., Srinivasa, S.S., Pillai, P., Campbell, J., 2008. Generalizing metamodules to simplify planning in modular robotic systems, in: Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pp. 1338–1345. doi:DOI: 10.1109/IROS.2008.4651094.

Fitch, R., Butler, Z., Rus, D., 2003. Reconfiguration planning for heterogeneous self-reconfiguring robots, in: Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, pp. 2460–2467. doi:DOI: 10.1109/IROS.2003.1249239.

Fitch, R., McAllister, R., 2013. Hierarchical Planning for Self-reconfiguring Robots Using Module Kinematics, in: Distributed Autonomous Robotic Systems 10, pp. 477–490. doi:DOI: 10.1007/978-3-642-32723-0_34.

Gorbenko, A.A., Popov, V.Y., 2012. Programming for modular reconfigurable robots. Programming and Computer Software 38, 13–23. URL: https://doi.org/10.1134/S0361768812010033, doi:DOI: 10.1134/S0361768812010033.

Hou, F., Shen, W.M., 2010. On the complexity of optimal reconfiguration planning for modular reconfigurable robots, in: Robotics and Automation (ICRA), 2010 IEEE International Conference on. doi:DOI: 10.1109/ROBOT.2010.5509642.

Kawano, H., 2015. Complete reconfiguration algorithm for sliding cube-shaped modular robots with only sliding motion primitive, in: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, pp. 3276–3283. doi:DOI: 10.1109/IROS.2015.7353832.

Kotay, K.D., Rus, D.L., 2000. Algorithms for self-reconfiguring molecule motion planning, in: Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on, pp. 2184–2193. doi:DOI: 10.1109/IROS.2000.895294.

Kurokawa, H., Murata, S., Yoshida, E., Tomita, K., Kokaji, S., 1998. A 3-D Self-Reconfigurable Structure and Experiments, in: Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications, p. 6.

Lengiewicz, J., Holobut, P., 2019. Efficient collective shape shifting and locomotion of massively-modular robotic structures. Auton. Robots 43, 97–122. URL: https://doi.org/10.1007/s10514-018-9709-6, doi:DOI: 10.1007/s10514-018-9709-6.

Michail, O., Skretas, G., Spirakis, P.G., 2017. On the Transformation Capability of Feasible Mechanisms for Programmable Matter. arXiv:1703.04381 [cs] URL: http://arxiv.org/abs/1703.04381. arXiv: 1703.04381.

Pannuto, P., Lee, Y., Foo, Z., Blaauw, D., Dutta, P., 2013. M3: a mm-scale wireless energy harvesting sensor platform, in: Proceedings of the 1st International Workshop on Energy Neutral Sensing Systems, ACM. p. 17. URL: http://dl.acm.org/citation.cfm?id=2534225.

Park, M., Chitta, S., Teichman, A., Yim, M., 2008. Au-

tomatic Configuration Recognition Methods in Modular Robots. The International Journal of Robotics Research 27, 403–421. URL: http://journals.sagepub.com/doi/10.1177/0278364907089350, doi:DOI: 10.1177/0278364907089350.

Pescher, F., Napp, N., Piranda, B., Bourgeois, J., 2020. GAP-CoD: A Generic Assembly Planner by Constrained Disassembly, in: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, Auckland, New Zealand.

Piranda, B., Bourgeois, J., 2016. Geometrical study of a quasi-spherical module for building programmable matter, in: DARS 2016, 13th International Symposium on Distributed Autonomous Robotic Systems. URL: https://www.researchgate.net/profile/Benoit_Piranda/publication/307930970_Geometrical_study_of_a_quasi-spherical_modules_for_building_programmable_matter/links/582438d308ae7ea5be7233a7.pdf.

Piranda, B., Bourgeois, J., 2018. Designing a quasi-spherical module for a huge modular robot to create programmable matter. Autonomous Robots 42, 1619–1633. URL: https://doi.org/10.1007/s10514-018-9710-0, doi:DOI: 10.1007/s10514-018-9710-0.

Piranda, B., Bourgeois, J., 2020. Datom: A Deformable modular robot for building self-reconfigurable programmable matter. arXiv:2005.03402 [cs] URL: http://arxiv.org/abs/2005.03402. arXiv: 2005.03402.

Piranda, B., Chodkiewicz, P., Holobut, P., Bordas, S.P.A., Bourgeois, J., Lengiewicz, J., 2020. Distributed prediction of unsafe reconfiguration scenarios of modular-robotic programmable matter. CoRR abs/2006.11071. URL: https://arxiv.org/abs/2006.11071, arXiv:2006.11071.

Piranda, B., Laurent, G.J., Bourgeois, J., Clévy, C., Möbes, S., Fort-Piat, N.L., 2013. A new concept of planar self-reconfigurable modular robot for conveying microparts. Mechatronics 23, 906–915. URL: http://linkinghub.elsevier.com/retrieve/pii/S0957415813001633, doi:DOI: 10.1016/j.mechatronics.2013.08.009.

Rubenstein, M., Ahler, C., Nagpal, R., 2012. Kilobot: A low cost scalable robot system for collective behaviors, in: 2012 IEEE International Conference on Robotics and Automation, IEEE, St Paul, MN, USA. pp. 3293–3298. URL: http://ieeexplore.ieee.org/document/6224638/, doi:DOI: 10.1109/ICRA.2012.6224638.

Spröwitz, A., Moeckel, R., Vespignani, M., Bonardi, S., Ijspeert, A., 2014. Roombots: A hardware perspective on 3D self-reconfiguration and locomotion with a homogeneous modular robot. Robotics and Autonomous Systems 62, 1016–1033. URL: http://linkinghub.elsevier.com/retrieve/pii/S0921889013001632, doi:DOI: 10.1016/j.robot.2013.08.011.

Støy, K., 2004. Emergent control of self-reconfigurable robots. Ph.D. thesis. University of Southern Denmark.

Støy, K., 2006. Using cellular automata and gradients to control self-reconfiguration. Robotics and Autonomous Systems 54, 135 – 141. URL: http://www.sciencedirect.com/science/article/pii/S0921889005001521, doi:DOI: https://doi.org/10.1016/j.robot.2005.09.017.

Støy, K., Nagpal, R., 2007. Self-Reconfiguration Using Directed Growth, in: Distributed Autonomous Robotic Systems 6, pp. 3–12. URL: https://doi.org/10.1007/978-4-431-35873-2_1, doi:DOI: 10.1007/978-4-431-35873-2_1.

Thalamy, P., Piranda, B., Bourgeois, J., 2019a. Distributed Self-Reconfiguration using a Deterministic Autonomous Scaffolding Structure, in: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, Montreal QC, Canada. pp. 140–148. doi:DOI: 10.5555/3306127.3331685.

Thalamy, P., Piranda, B., Bourgeois, J., 2019b. A survey of autonomous self-reconfiguration methods for robot-based programmable matter. Robotics and Autonomous Systems 120, 103242. URL: https://linkinghub.elsevier.com/retrieve/pii/S0921889019301459, doi:DOI: 10.1016/j.robot.2019.07.012.

Thalamy, P., Piranda, B., Bourgeois, J., 2020a. 3D Coating Self-assembly for modular robotic scaffolds, in: In Proceedings 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, NV, USA.

Thalamy, P., Piranda, B., Lassabe, F., Bourgeois, J., 2019c. Scaffold-Based Asynchronous Distributed Self-Reconfiguration By Continuous Module Flow, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4840–4846. doi:DOI: 10.1109/IROS40897.2019.8967775.

Thalamy, P., Piranda, B., Lassabe, F., Bourgeois, J., 2020b. Deterministic scaffold assembly by self-reconfiguring micro-robotic swarms. Swarm and Evolutionary Computation 58, 100722. URL: http://www.sciencedirect.com/science/article/pii/S2210650220303758, doi:DOI: https://doi.org/10.1016/j.swevo.2020.100722.

Tucci, T., Piranda, B., Bourgeois, J., 2017. Efficient Scene Encoding for Programmable Matter Self-reconfiguration Algorithms, in: Proceedings of the Symposium on Applied Computing, pp. 256–261. URL: http://doi.acm.org/10.1145/3019612.3019706, doi:DOI: 10.1145/3019612.3019706.

Tucci, T., Piranda, B., Bourgeois, J., 2018. A Distributed Self-Assembly Planning Algorithm for Modular Robots, in: International Conference on Autonomous Agents and Multiagent Systems) (AAMAS), Association for Computing Machinery (ACM), Stockholm, Sweden. pp. 550–558.

Yoshida, E., Murata, S., Kurokawa, H., Tomita, K., Kokaji, S., 1998. A distributed method for reconfiguration of a three-dimensional homogeneous structure. Advanced Robotics 13. doi:DOI: 10.1163/156855399X00234.

Zhu, L., El Baz, D., 2019. A programmable actuator for combined motion and connection and its application to modular robot. Mechatronics 58, 9–19. URL: https://linkinghub.elsevier.com/retrieve/pii/S0957415819300029, doi:DOI:

10.1016/j.mechatronics.2019.01.002.

Zhu, Y., Bie, D., Wang, X., Zhang, Y., Jin, H., Zhao, J., 2017. A distributed and parallel control mechanism for self-reconfiguration of modular robots using L-systems and cellular automata. Journal of Parallel and Distributed Computing 102, 80 – 90. URL: http://www.sciencedirect.com/science/article/pii/S0743731516301824, doi:DOI: https://doi.org/10.1016/j.jpdc.2016.11.016.

Ünsal, C., Khosla, P.K., 2001. A multi-layered planner for self-reconfiguration of a uniform group of I-Cube modules, in: Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on, pp. 598–605. doi:DOI: 10.1109/IROS.2001.973421.

Ünsal, C., kilivççöte, H., Patton, M.E., Khosla, P.K., 2000. Motion Planning for a Modular Self-Reconfiguring Robotic System, in: Parker, L.E., Bekey, G., Barhen, J. (Eds.), Distributed Autonomous Robotic Systems 4, Springer Japan, Tokyo. pp. 165–175. URL: https://doi.org/10.1007/978-4-431-67919-6_16, doi:DOI: 10.1007/978-4-431-67919-6_16.