



HAL
open science

3D Coating Self-Assembly for Modular Robotic Scaffolds

Pierre Thalamy, Benoit Piranda, Julien Bourgeois

► **To cite this version:**

Pierre Thalamy, Benoit Piranda, Julien Bourgeois. 3D Coating Self-Assembly for Modular Robotic Scaffolds. RSJ International Conference on Intelligent Robots and Systems, Oct 2020, Las Vegas, United States. 10.1109/IROS45743.2020.9341324 . hal-03186603

HAL Id: hal-03186603

<https://hal.science/hal-03186603>

Submitted on 31 Mar 2021

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

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

3D Coating Self-Assembly for Modular Robotic Scaffolds

Pierre Thalamy[†], Benoît Piranda[†], and Julien Bourgeois[†]

Abstract—This paper addresses the self-reconfiguration problem in large-scale modular robots for the purpose of shape formation for object representation. It aims to show that this process can be accelerated without compromising on the visual aspect of the final object, by creating an internal skeleton of the shape using the previously introduced sandboxing and scaffolding techniques, and then coating this skeleton with a layer of modules for higher visual fidelity.

We discuss the challenges of the coating problem, introduce a basic method for constructing the coating of a scaffold layer by layer, and show that even with a straightforward algorithm, our scaffolding and coating combo uses much fewer modules than dense shapes and offers attractive reconfiguration times. Finally, we show that it could be a strong alternative to the construction of dense shapes using traditional self-reconfiguration algorithms.

I. INTRODUCTION

Modular self-reconfigurable robots (MSR) [12] are robots consisting of an arrangement of interconnected modules with computational capabilities and that can generally move around the robot and communicate with other modules, allowing for unprecedented versatility thanks to their metamorphic properties. For this reason, MSR are a remarkably promising form of programmable matter (PM), usually defined as matter that is able to autonomously change its state based on sensed events. We will use the term *configuration* to refer to a given arrangement of modules in an MSR, and *self-reconfiguration* for the autonomous displacement of modules leading from one configuration to another.

In our work, we are interested in modular robotic systems consisting of thousands of micro-scale modules, whose purpose is to represent a particular object through self-reconfiguration, after its description is supplied to the system.

However, the *self-reconfiguration problem*, that finds a sequence of individual motions of the modules to transform an MSR from an initial configuration into a goal one is a notoriously hard problem due to the combinatorial explosion caused by the all the different ways the modules can be connected to each other. Furthermore, self-reconfiguration is usually painfully slow, as the constraints imposed on the motion of the modules makes it extremely tedious to reach a high number of modules moving around the robot in parallel without dramatically increasing the risk of collisions between modules or deadlocks.

Accordingly, we have shown in [13] and [15] that unprecedented self-reconfiguration speeds could be achieved thanks to two propositions we made. First, engineering the

reconfiguration environment such that the reconfiguration takes place over a reserve of modules, or *sandbox*, through which modules can be supplied from the ground of the reconfiguration scene or discarded from it. Second, by engineering the goal shape itself and building a porous skeleton, or *scaffold*, of the shape instead of the compact target object, and with a regular and predictable internal structure, the construction requires fewer modules and it is much easier to coordinate the flow of modules for maximum parallelism and efficiency.

Nonetheless, the scaffolding technique has a major drawback, which is that the external aspect of the built object is not preserved (as its surface is porous too), so the fidelity of the constructed object to the supplied model is lessened. For that reason, we propose in this article to cover the surface of these porous objects using a single layer of modules, through a process called *coating*, so as to recreate the correct external aspect of objects, and complement scaffolding. This scaffold and coating method can be seen as a special case of shape self-reconfiguration from a reserve of modules.

The objective of this paper is to introduce the coating problem and the challenges it poses in a face-centered cubic (FCC) lattice. It also addresses how a coating can be designed in this context for a large class of shapes, and provides a straightforward algorithmic solution to its construction. It aims to show that even with a relatively inefficient coating method, using a coated scaffold may be preferable to building a dense shape.

The next section will start by introducing related works. Then, Section III introduces our model and formulates the scaffold coating problem, while Section IV presents our assembly strategy. Finally, Section V presents results and simulations of our method performed on the VisibleSim [9] simulator for modular robots, before concluding the paper in Sections VI.

II. RELATED WORK

This work relates most closely to the literature on robotic self-assembly, whose aim is to produce a correct and deadlock-free assembly plan for constructing a shape, either made from passive materials brought by swarm robotic units [18, 3], or from modular robotic units themselves as in our case [16, 8, 19]. In the second case, self-assembly approaches usually rely on a set of construction rules that provide a feasible and deadlock-free assembly plan to construct a shape. This assembly plan is sometimes pre-computed prior to the construction itself, in a centralized manner. It can then be followed by the robotic units taking part in the construction of the target shape. This is usually

[†]All authors are with Univ. Bourgogne Franche-Comté, FEMTO-ST Institute, CNRS, 1 cours Leprince-Ringuet, 25200, Montbéliard, France. {first}. {last}@femto-st.fr

done through a virtual disassembly of the target shape, as is the case with the compiler for the TERMES swarm systems [18, 3], which has been recently applied generically to modular robotic self-assembly in [8]. In other instances, such as in [19], the exact order of the construction of the shape is pre-defined by a human operator.

Lastly, assembly decisions can be made by the distributed robotic units on the fly by relying on a set of local neighborhood rules that describe the order and constraints under which a module of the shape must attract neighbors and by resolving global constraints through communication [16, 15, 13]. Our present work belongs to the latter category.

While self-assembly is usually unconcerned with the actual dispatch of the modular robotic modules to their respective destination inside the shape, self-reconfiguration consider this extra step, and aims to *transform* an initial configuration of such interconnected modules into a goal configuration, through the motion of its components. This has been extensively studied in lattice-based modular robots [14] using both stochastic [21, 5] and deterministic approaches [11, 7], and was shown to be an NP-complete problem in [20].

This work is a follow-up of our previous work on the coordinated construction of the scaffold of objects by *3D Catom* modules mentioned in the introduction, which was demonstrated in [13] for the asynchronous construction of square pyramids of various sizes and later generalized to a large class of convex objects in [15].

Finally, the problem of coating a 2D shape has been studied theoretically in the context of *self-organizing particle systems (SOPS)*, more specifically under the *Amoebot* model. In this theoretical approach, it is shown in [4, 2] that the coating of a 2D object can be done using only local information in linear time with high probability. To the best of our knowledge, our work is the first work on the coating of a modular robotic structure by other modular robotic units in 3D.

III. PROBLEM FORMULATION AND MODEL

A. *3D Catom* model

The *3D Catom* [10] is a micro-scale lattice-based modular robot that is currently under development in the context of the Programmable Matter Project¹. It embeds the Michigan Micro-Mote (M^3)[6], a power driver and electrostatic soft actuators in a 3.6mm-diameter quasi-sphere.

It resides in a Face-Centered-Cubic (FCC) lattice with staggered horizontal layers, and its individual modules can assemble with up to 12 neighbor modules, connected through electrostatic connectors on their surface (in red in Figure 1). The connectors are not only used for latching onto neighbors and actuation between modules, but also for communicating with them in a peer-to-peer fashion.

3D Catom motions are rotations exclusively, and require the presence of a neighbor module acting as a *pivot* on which to perform the motion. Furthermore, as the modules

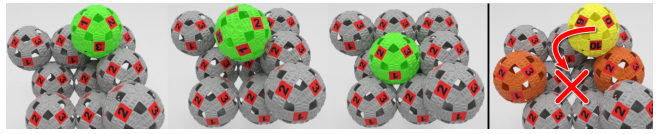


Fig. 1. *3D Catoms* and motion capabilities.

undergo no deformation during motions, there is another major constraint on the rotation of *3D Catoms*, which is that a module cannot leave or enter a lattice position that is surrounded by two opposing neighbor modules, as the module would either be blocked or the motion would result in a collision with one of the neighbors (see Figure 1). As a consequence, structures made of *3D Catoms* must assemble in such a way that no deadlock of this sort can occur.

Lastly, all modules are embedded with a microprocessor and all computation is performed locally on each module, as dictated by the principles of distributed computing.

B. The Coating Problem

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.

There are a few things to unpack from this problem definition. First, the scaffold structure corresponds to an arrangement of interconnected *3D Catom* modules built using a generalization of the method introduced in [13], where a shape is discretized into multi-module units named tiles, which consist of a root module and a number of branches both on the XY-plane and upward, with a length of 1 to b modules, with b a parameter of the scaffold (see Figure 2.a). These tiles would assemble by connecting the tips of the branches of a tile to the tile root of other tiles, filling the boundaries of the shape to be represented (see Figure 2.b and Figure 3.a). Note that in this context, a *3D Catom cube* has equal dimensions in number of modules composing it, but as the horizontal layers of modules are staggered in a FCC lattice, the resulting cube appears shorter than it is wide or deep.

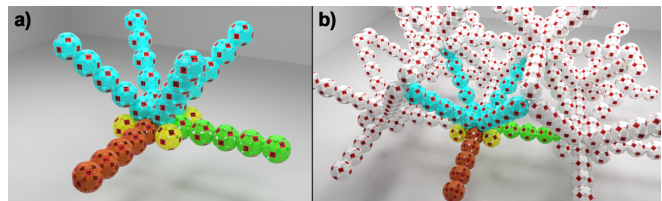


Fig. 2. (a) Anatomy of a tile with $b = 6$ (5 modules per branch + a root module at the center that is hidden). Colors: Horizontal branches along the \vec{x} and \vec{y} axes in red and green, respectively; the four vertical tile branches in blue; helper modules enabling vertical tile traversal in yellow. (b) Arrangement of tiles constituting an arbitrary scaffold.

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

¹<http://projects.femto-st.fr/programmable-matter/>

of being hollow, hence providing high fidelity to the object that is being represented.

Finally, 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 relates to the mechanical stability of the object, as the scaffold provides an internal structure to the object that grants its mechanical stability. This can be represented on a spectrum, with a *loose coating* on one end, and a *tight coating* on the other one. 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. On the other hand, 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 paper, 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.

C. Assumptions

This work relies on a number of assumptions, listed below:

- All modules hold a description of the target shape, and an additional simple lookup function for evaluating whether a position is in the target shape. This description is lightweight and vectorized based on *Constructive Solid Geometry (CSG)*, as first introduced in the context of lattice-based modular robotics in [17]. Consequently, all modules can evaluate if a position is a coating position or a support position.
- Computation is fully distributed across *3D Catom* modules and communication is also distributed and based on a message-passing scheme to physically connected neighbors.
- The reconfiguration occurs in a *sandbox environment*, as introduced in [13, 15], which means that there is a reserve of modules below the ground of the reconfiguration scene and that can supply modules at any time to various ground locations of the scene (cf. the gray modules and tile branches in Figure 3.a/b).
- Message sending and message propagation time are negligible against the rotation time of *3D Catoms*.
- All modules agree on a common coordinate system defined by the sandbox (with the origin at its front-bottom-left corner), where the FCC lattice is represented as a 3D grid.

D. Coating Definition

As mentioned in Subsection III-B, our coating is a thin (one module thick) layer of modules that covers every bit of the surface of the scaffold (besides the ground, as the sandbox prevents us from adding that coating without complex additional steps), with a small distance between the two so as to avoid the risk of deadlocks during construction as much as possible, but with added contact points that provide mechanical stability to the overall shape.

The definition of the scaffold and coating are both derived from a single CSG description of the target shape stored in the memory of modules. A position is considered to be inside the scaffold if its position verifies a set of geometrical rules determining if this position can be a scaffold component, and if that position is within the object described by the CSG, at least at a distance of two lattice cells from its border. Then, a module is in the coating if it is on the border of the CSG object, that is to say if it is inside the object but has a neighbor that is outside of it. This will result in a skeleton formed by the scaffold at the core of the object, surrounded by an empty envelope, and then the coating, thus leaving space between the two (see Figure 3.a).

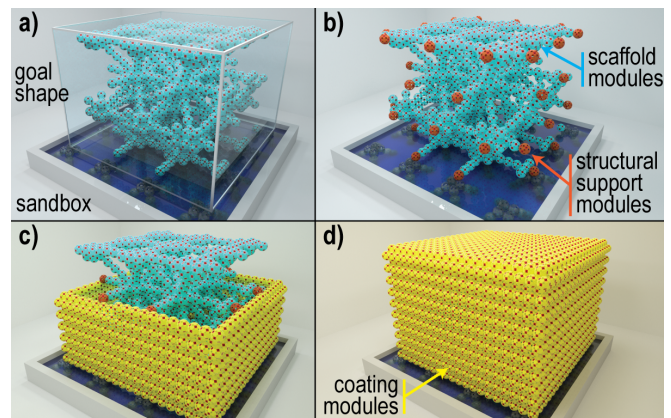


Fig. 3. a) Scaffold of a cube with length 20 module, with empty space around; (b) scaffold with its horizontal branches extended into structural supports; (c) scaffold being coated; (d) fully assembled coating of a cube.

The contact points (or *structural supports*) are closely linked to the scaffold itself and its b parameter, as the supports are modules resulting from lengthening the external horizontal branches of surface tiles by one module (see Figure 3.b, thus closing the gap between the coating and the scaffold at various points of horizontal layers every b modules in height.

IV. COATING SELF-ASSEMBLY

This section describes how the coating is **assembled** in such a way that no deadlocks occur, **without regard to the actual flow of the modules from the sandbox to their target location**, which is out of the scope of this paper. Our method can assemble the coating of all shapes where all portions of its coating layers are directly adjacent (or connected) to the layer below. This means that shapes with overhangs or that require inner coating (such as bowl-like shapes) are unsupported and will be dealt with in future

work. Our assembly method can be decomposed into three different components.

The first one is a high-level view of how the coating is assembled, by building the horizontal layers constituting it one at a time, from bottom to top. Then, the other components are two different strategies that determine the assembly order of the coating within a given layer, which is therefore a 2D assembly problem. The strategy to be used for a given layer depends on whether or not structural supports, potentially causing obstacles, have been introduced for that layer. We will use the term *attract*, to refer to the action of a module that advertises that a position next to itself is ready to be filled, causing another module to come and claim it, thus filling that position.

A. High-Level Assembly Strategy (Bottom-Up Layering)

The manner in which the coating is assembled at the scale of the object can be summed up as *Bottom-Up Layering*, which means that the coating is assembled one layer at a time, from the base of the object to its top. This may be suboptimal, but thanks to the space between the scaffold and the coating itself, this relaxes the constraints imposed on the construction of a given layer, turning it into a simpler 2D problem. Coating layer n thus has to wait for coating layer $n - 1$ to have finished building before starting its construction, with $n > 0$.

The assembly of a given layer always starts with the attraction of a module to a single position of that layer, and proceeds through the recursive attraction of neighbors by attracted modules, according to the rules detailed in the next subsections. The need for a single source and direction of growth of the shape is a consequence of the motion constraints of *3D Catoms* introduced in Subsection III-A. Let $seed_n$ be the first module that must be attracted for horizontal layer n of the coating. Please note, however, that for a layer made of multiple disjoint parts as it might happen in some shapes, these parts are built independently, from different seeds. This module will be attracted by a module from the previous layer, $attractor_n$, determined according to a method inspired by the Tucci Algorithm [16]:

Any module can test if it is an attractor module for the next plane by checking if it has a top neighbor position that is part of the border of the coating, and if that is the position with minimum y position and maximum x position across the border that has a bottom neighbor that is in layer $n - 1$. This can all be done through a virtual border following and an exploration of the next coating layer, as modules all hold the CSG description of the shape in memory. $seed_0$ is simply determined by the coordinate criterion as it has no matching attractor. A simple messaging is used to reach a consensus on when the construction of the current layer is over, and for notifying the next attractor (or attractors in the case of a splitting of the shape) that the construction of the next layer start.

From there on, two different methods are used to assemble a given layer, depending on whether this layer needs to attract support modules or not. Now, these methods need

to lead to a correct solution systematically, no matter what the morphology of the coating layer is like. Indeed, while the simplest coating layers are simply a one-module thick border around a section of the object, when the surface of the object has a steep slope, and which can be solved trivially by simply adding modules one at a time following the border, more complex cases exist. These cases includes cases with thicker borders when the surface of the object has a gentler slope (as the lattice forces an approximation of the shape, much like the approximation of lines using Bresenham’s algorithm in computer graphics [1]), when the layer is a fully horizontal plateau, or any combination of those cases. Systematically finding an assembly plan is hence not trivial in all cases.

B. Standard Layer Assembly Strategy (The Tucci Algorithm)

In the case where the layer does not need to attract support modules (when $z \bmod b \neq 0$, with b the branch length parameter of the scaffold), the assembly problem is a standard 2D assembly problem, without obstacles. Luckily, the assembly problem without obstacles has been previously solved in the exact context of FCC lattice modular robots using the Tucci Algorithm [16], hence we can rely on this method for this type of layers. The 2D version of this algorithm is briefly explained in the rest of this section.

As stated in the previous subsection, every layer starts with the attraction of a first module to the *seed* position. Then, each module M_i attracts neighbor modules to the cells among its 4 horizontal neighbor locations that are inside the target shape G (or coating in our case), according to local rules that enforce a diagonal growth direction of the plane and without deadlocks. The rules are built in such a way that:

- Modules having a local neighborhood in which the addition of a neighbor in a direction does not risk to cause another position to become unreachable, attract a module to that neighbor position right away.
- Otherwise, if an attraction might block a nearby position, they communicate with their neighbors to synchronize and ensure that the attraction is only performed when potential blocked positions are filled.
- If a target position might be a merge point between two parts of a plane growing concurrently around an internal hole, the neighbor seeking to attract a module to that position will send a probe message that will follow the border of the hole and will return when all the other positions of the border have been filled.

Please refer to [16] for more information on assembly rules and experiments showing that this method achieves a very high convergence rate into the goal shape, with only a number of messages linear in the number of modules.

C. Support Layer Assembly Strategy (Border Completion)

For all layers that need to attract support modules ($z \bmod b = 0$), first, all the structural supports are attracted by their neighbor modules from the previous layer, and then a different assembly algorithm is applied, taking into account the supports as obstacles.

1) *Supports and Segments Attraction*: Therefore, when the consensus on the completion of the $n - 1$ layer has been reached, all the modules check whether they have a support position as a top neighbor on the next plane, and attract another module to that position if that's the case. However, as mentioned, the support modules now create obstacles to the assembly process, which can cause neighbor coating positions to become unreachable for any growth direction other than one originating from the support itself (see Figure 4.b) because of the opposing module constraint of *3D Catoms*. This is not always the case, and modules can cause no potentially blocked cells (see Figure 4.a), as this depends on the location of the support module itself.

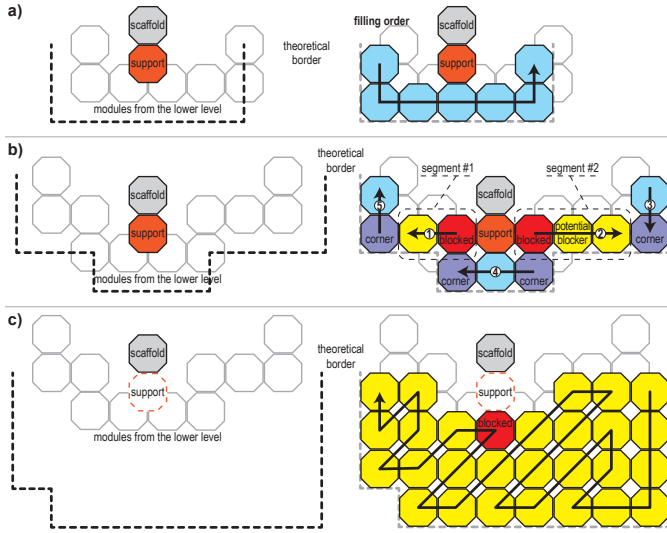


Fig. 4. (a) Structural support producing no blocked positions; (b) support producing blocked positions and corresponding support segments; (c) support position that cannot be filled in the current implementation.

In some cases, the support that is attracted might cause some coating positions to become blocked (Please refer to Figure 4.b throughout this paragraph). In that case, the support module (in Orange) would cause its East and West neighbor positions to become unreachable, blocked by the yellow modules, if the direction of growth of the border has any source other than the support. This is not the case of its South neighbor (blue, bottom), as the position opposite from the support is not part of the coating. In that case, the only way to avoid a deadlock is by actually growing this part of the coating from the support module. Fortunately, this does not require the whole coating layer to be grown from the support, which would cause intractable synchronization issues, but simply a number of segments, originating from the support, followed by the blocked position, and consisting of all modules in that direction until the next corner module (in purple, not part of segments). While it is possible that two support modules are adjacent to modules of the same segment, this segment will only be created by one of the two supports, as it is impossible that a coating border contains two supports that create blocked positions without a corner between the two, except on wide borders.

Indeed, in extreme cases, in which a support is adjacent

to a coating section that is larger than one module thick, the insertion of the support would force the construction of the whole plane from the support. This can be problematic if one or more other supports are adjacent to the same coating section, causing multiple starting points of this coating layer that are very hard to synchronize. In such case, where a support is adjacent to a border thicker than one module, its insertion is omitted (see Figure 4.c).

Segments are grown by recursively attracting a module to the next position of the segment and sending it a message when it arrives instructing it to continue the growth of the segment, until the next position is a corner. When that happens, an acknowledgment is returned to the support growing that segment, so that it can be known when it has finished growing its segments.

2) *Segment Detection*: Once all segments from all supports have been grown, the coating layer will be partially filled by all segments positions. As Tucci's algorithm does not support obstacles (otherwise simply defining support positions as part of the coating would have sufficed as a general solution to our coating problem), another method of assembling the remaining modules is required.

This algorithm can only start once all the segments are in place, so the first step is to detect when that is the case. For that purpose, the *attractor* module of the previous layer will send a *NextPlaneSupportReadyRequest* message across the external border of the coating, in a single direction. This message contains a single bit of data, which indicates whether a segment has been detected along the border above. For all modules that do not have a support neighbor in the layer above, the message will be directly forwarded to the next module along the border. For all the others, one of three things can happen:

- 1) Their support neighbor has not yet arrived: In that case, the module will wait for the support to arrive, and then forward the request to it.
- 2) The support is present and has not finished growing its segments: The module forwards the request to it and the support holds its response until its segments are complete.
- 3) The support is present and has finished growing its segments or does not have any: The module forwards the request to it and the support responds with *True* if it has grown a segment, *False* otherwise.

Once a module receives a response from a support, it sets the segment detection bit to $sd = sd \text{ OR } sd_{rcvd}$ so that once set to *True* it cannot be unset later along the path. The message is then forwarded with the newly set data bit to the next module along the path. This ends when the message reaches the *attractor* module again, from the other side of the border. Since its propagation cannot proceed as long as all the segments of the supports along the path are not complete, the purpose of this message propagation is twofold: (1) detecting the presence of segments in the layer above; (2) detecting when all supports have been attracted and have finished constructing their segments.

3) *Border Completion Algorithm*: At that point, either no segment has been detected and the *attractor* initiates the Tucci algorithm as before, since there are no obstacles, or, segments have been detected and it performs a border completion algorithm. It is quite straightforward: if the *seed* module of the next layer is not part of a segment, it first attracts it. Then, or if the *seed* is part of a segment, it sends it a *BorderCompletion* message. The border completion message is then propagated along the coating border in a single direction (or two, but this requires a synchronization corner somewhere along). When a module receives it, either the next coating position along the border is already filled, and it forwards the *BorderCompletion* message there, or first attracts a module to that position and then forwards it. The layer is over when the message returns to the *seed*.

When all layers of the coating have finished building, the coating algorithm terminates.

V. RESULTS

A. Preservation of message complexity

In this section, we show that the number of messages used by our coating assembly method is linear in the number of modules in the coating. It was shown in [16] that the number of messages to assemble a shape using the Tucci algorithm was linear in the number of modules in the shape. We thus aim to demonstrate that this result is preserved and that our method is asymptotically as efficient.

First, as explained in Section IV, all coating layers that are not at the level of scaffold tile roots (every b layers, with b the scaffold branch length parameter) and thus do not have supports, are simply executing the Tucci algorithm to assemble the coating, and thus do not require any additional message exchange. It is therefore sufficient to show that the assembly of a coating layer that has structural supports is also in $O(N_i)$, where N_i is the number of modules at layer i of the coating, and such that $\text{mod}(i, b) = 0$.

To that purpose, the list of all types of messages (and not part of Tucci's algorithm) we use is reviewed below:

- **Support segment attraction and completion messages**: A support can have a maximum of two segments growing from it according to our support selection criteria, and each segment will have a length that is a fraction of N_i . The growth of a segment of length l takes l messages for its construction and l messages to notify the parent support of completion. Thus it is in $O(N_i)$.
- **Support ready detection messages**: *NextPlaneSupportReadyRequest* messages are propagated along the border of the coating, reaching at most once every module of the coating that is not a neighbor to a structural support on the same plane. For all modules that have a support neighbor, it takes an additional message to request the state of the support, and another one for its response. This process thus also takes $O(N_i)$ messages.
- **Border completion messages**: This is propagated once from the attractor of the previous plane to the seed of the

current one, and then once per module of the coating, it is therefore also linear in the number of modules in the coating layer.

Since all of these messages are used in a number linear to the N_i , we can therefore deduce that the total additional number of messages used by our method compared to Tucci's algorithm is also linear in the number of modules in the shape, and thus the message complexity of the overall coating algorithm is in $O(n)$, with n the number of coating modules.

B. Simulations

The following simulations have been performed on VisibleSim [9], a lattice modular robot simulator. We have constructed the coating of objects of varying complexity to show that our method works on very diverse styles of shapes. A video of these simulations is provided² to better illustrate the method and results, which also contains additional explanations of the coating algorithm. The *Cube 100* is excluded from the video, as it is just like the *Cube 20*, but bigger. The results are presented on Table I and additional details regarding the specificities of each shape are provided below:

- *Cube* ($l = 20$ modules): Uses Tucci's algorithm only as all supports are non-blocking.
- *Cylinder* ($h = 20$, $d = 20$): Simple border completions.
- *Chair*: Concavities. Merge of disjoint components.
- *Sandcastle*: Large complex shape. Splitting components.
- *Cube* ($l = 100$ modules): Uses Tucci's algorithm only. Very high volume.

The number of modules in the coating and the coating formation time are represented by the *coating count* and *coating time* values on Table I, respectively. The coating time is expressed as a number of time steps, with a single time step corresponding to the insertion time for one module. We can thus deduce from the coating time and coating count the coating rate, the average number of modules attracted per time step. As we will see, the coating rate depends on the number of disjoint subparts of an object, and the portion of the coating that is on horizontal planes, which can be easily built in parallel.

Then, the *density ratio* expresses the ratio of modules in the coated scaffold object over the number of modules in the dense version of that object. It appears that the higher the volume of an object, the lower its density ratio, as the gain in modules is essentially the result of the scaffold.

Finally, the *coating ratio* expresses the ratio of coating modules over all modules in the final shape. For shapes big enough, the number of coating modules will become lower than the number of scaffold modules. Though not only the size matters as ultimately the coating ratio just reflects the size of the surface of an object relative to its volume. In our case, the actual volume of the shape in number of modules grows slower than for dense shapes because of the porous nature of the scaffold. The *Chair* for example, despite

²<https://youtu.be/5nQVQgAu3SQ>

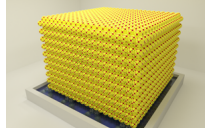
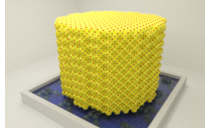
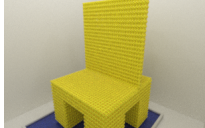
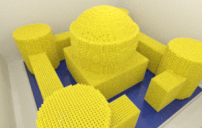
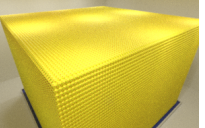
	Cube (l=20 modules)	Cylinder (h=20, d=20)	Chair	Sandcastle	Cube (l=100)
					
Coating Count	1769	1820	8430	39225	49404
Coating Time	773	1476	4155	23652	20336
Coating Rate	2.29	1.23	2.03	1.66	2.43
Density Ratio	31.53%	37.35%	38.89%	29.12%	19.87%
Coating Ratio	69%	76%	84%	60%	24.73%

TABLE I

SIMULATION RESULTS OF OUR COATING ALGORITHM ON SHAPES OF VARIOUS SIZES.

its relatively big size, has a very low volume compared to its surface, and this is reflected in its coating ratio. As our coating algorithm is inherently slower and less parallel than the scaffold construction algorithm, our scaffold coating method will therefore become even more efficient compared to the construction of dense objects for shapes with a large volume (e.g., *Cube of side 100*), where most of the modules belong to the scaffold.

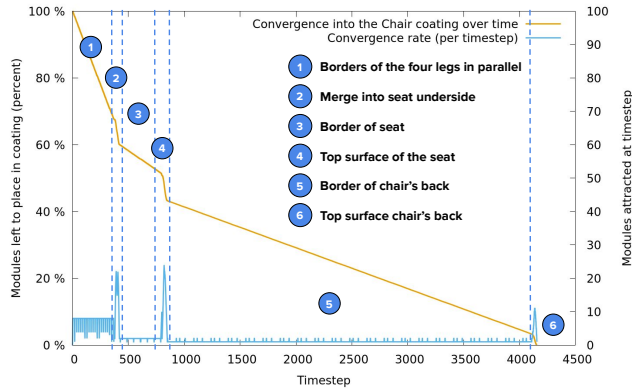


Fig. 5. Convergence of the coating algorithm into the coating of the *Chair* shape over time.

While it is always linear, the convergence rate of our method is very uneven depending on the portion of the coating being built, as can be seen in Figure 5 with the example of the *chair*. Whenever the current coating layer is a border, the convergence rate is rather low (1 or 2 modules per time step and disjoint subpart), whereas planar layers cause great accelerations in the convergence rate, as all modules along the diagonal of the plane can be attracted at the same time. We can therefore conclude that the speed of the construction of the coating of a shape depends on the percentage of the coating that is part of a horizontal plane, as well as the number of disjoint subparts of that shape.

C. Complexity

We have seen in the previous section that the time to assemble the coating of an object — regardless of the motion of the modules from the sandbox to their attraction position — was linear in the number of modules composing the coating. We now want to show that even when considering the motion of the modules, the construction time is still linear in the number of modules, as long as trains of modules

moving in parallel are formed, moving from a sandbox source location to the available positions of the coating. Indeed, let us consider a single source of sandbox modules at the base of the scaffold, or rather a single source per disjoint component of the object, and where modules are introduced at regular intervals leaving only one free position between them, forming a train of modules. Let t_i the time at which the coating position number i in the assembly order is filled, and assuming that it would take a constant number of time steps c (realistically 2 to 4) for the next module along the train to take its position, then:

$$t_0 = 0, t_1 = t_0 + c, t_2 = t_1 + c = t_0 + 2 \times c, \dots$$

Therefore, $t_{(n-1)} = t_0 + (n-1) \times c$, where n represents the number of modules in the coating and $t_{(n-1)}$ the time of arrival of the last coating module. The construction time is thus in $O(n)$.

Our self-reconfiguration algorithm for building the scaffold of a shape has been shown to have an $O(\sqrt[3]{N})$ reconfiguration time for all *semi-convex* shapes, i.e., having no vertical concavities, and with no hole between the sandbox and the shape [15]. This makes the total reconfiguration time $O(n_{\text{coating}}) + O(\sqrt[3]{n_{\text{scaffold}}})$, or $O(n_{\text{coating+scaffold}})$ for that class of shape. But more importantly, our method reduces the number of modules to reconfigure to a fraction of the number of modules in the dense version of the object for the same visual result, massively cutting down on reconfiguration time. Furthermore, the bigger the object, the bigger the gain in saved modules. Nonetheless, for the scaffold and coating method to reach its true potential, the current performance of the coating algorithm will need to be greatly improved.

D. Possible Improvements

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.

We are therefore investigating two advanced strategies to improve the parallelism of the method.

- 1) **Parallel Single Layer:** For all non-planar layers, we can segment the coating border into sections of the coating separated by corners, which can be formed in parallel. Once two adjacent segments are complete, the corner modules in between can be added.

- 2) **Parallel Multilayer:** Allow the attraction of a module to a coating position as soon as all its neighbor positions that are in the previous layer are filled. 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. This would be equivalent to extending the multilayer version of the Tucci algorithm to support the presence of obstacles.

With both strategies, numerous sources of modules are needed to dispatch them optimally, and the concurrent planning of the motion of the modules from their source to their destination becomes non-trivial.

VI. CONCLUSION

In this paper, we have introduced the coating problem of modular robotics, where a modular robot forming a scaffold of an object has to be covered with a thin layer of modules so that it appears dense to the eye. We have explained how the coating could be designed so that it can be built easily without having to sacrifice the mechanical structure of the object, thanks to the addition of special modules named a *structural support*. We have then introduced a method that provides an assembly order for constructing this coating from a reserve of modules in the form of a sandbox, using the Tucci algorithm and our Border Completion algorithm. Finally, we have provided simulation results with our coating method applied to various kinds of shapes, outlining its performance and current limitations, while showing that even in its current state it could be used to achieve the construction of a coating in time linear to the number of modules in the envelope for all shapes with no overhangs or bowl-like concavities. Finally, we have shown that together, the sandbox, scaffold construction, and the coating could be used to greatly speed up the construction of modular robotic objects compared to the regular construction of dense shapes, and without altering their resulting external aspect. Though this algorithm has been designed for creating a coating that envelops a scaffold, this method could in principle work for any shape inside an FCC lattice, though the location and definition of the *structural supports* might need to be adapted to the problem at hand.

As future work, different strategies for achieving a higher level of parallelism and generalizing our approach to all shapes will be studied. We will also investigate different methods for dispatching the modules from the sandbox to the coating with maximum efficiency.

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

- [1] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.
- [2] Joshua J. Daymude, Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. On the Runtime of Universal Coating for Programmable Matter. *Natural Computing*, 17(1):81–96, March 2018. arXiv: 1606.03642.
- [3] Yawen Deng, Yiwen Hua, Nils Napp, and Kirstin Petersen. Scalable Compiler for the TERMES Distributed Assembly System. In Nikolaus Correll, Mac Schwager, and Michael Otte, editors, *Distributed Autonomous Robotic Systems*, volume 9, pages 125–138, Boulder, CO, USA, 2019. Springer International Publishing.
- [4] Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Universal coating for programmable matter. *Theoretical Computer Science*, 671:56–68, April 2017.
- [5] Robert Fitch and Rowan McAllister. Hierarchical Planning for Self-reconfiguring Robots Using Module Kinematics. In *Distributed Autonomous Robotic Systems 10*, pages 477–490, 2013.
- [6] S. Jeong, Z. Foo, Y. Lee, J. Sim, D. Blaauw, and D. Sylvester. A fully-integrated 71 nm cmos temperature sensor for low power wireless sensor nodes. *IEEE Journal of Solid-State Circuits*, 49(8):1682–1693, Aug 2014.
- [7] Jakub Lengiewicz and Pawel Holobut. Efficient collective shape shifting and locomotion of massively-modular robotic structures. *Auton. Robots*, 43(1):97–122, 2019.
- [8] Florian Pescher, Nils Napp, Benoît Piranda, and Julien Bourgeois. GAPCoD: A Generic Assembly Planner by Constrained Disassembly. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, Auckland, New Zealand, 2020.
- [9] Benoît Piranda. VisibleSim: Your simulator for Programmable Matter. In Kay Römer, Christian Scheideler, Sándor P. Fekete, and Andréa W. Richa, editors, *Algorithmic Foundations of Programmable Matter (Dagstuhl Seminar 16271)*, volume 6 of *Dagstuhl Reports*, page 12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, May 2016. doi:10.4230/DagRep.6.7.1.
- [10] Benoît Piranda and Julien Bourgeois. Designing a quasi-spherical module for a huge modular robot to create programmable matter. *Autonomous Robots*, 42(8):1619–1633, December 2018.
- [11] Kasper Støy and Radhika Nagpal. Self-Reconfiguration Using Directed Growth. In *Distributed Autonomous Robotic Systems 6*, pages 3–12, 2007.
- [12] Ning Tan, Abdullah Aamir Hayat, Mohan Rajesh Elara, and Kristin L. Wood. A Framework for Taxonomy and Evaluation of Self-Reconfigurable Robotic Systems. *IEEE Access*, 8:13969–13986, 2020.
- [13] P. Thalamy, B. Piranda, F. Lassabe, and J. Bourgeois. Scaffold-Based Asynchronous Distributed Self-Reconfiguration By Continuous Module Flow. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4840–4846, November 2019.
- [14] Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. A survey of autonomous self-reconfiguration methods for robot-based programmable matter. *Robotics and Autonomous Systems*, 120:103242, October 2019.
- [15] Pierre Thalamy, Benoît Piranda, Frédéric Lassabe, and Julien Bourgeois. Deterministic scaffold assembly by self-reconfiguring micro-robotic swarms. *Swarm and Evolutionary Computation*, 58:100722, 2020.
- [16] Thadeu Tucci, Benoît Piranda, and Julien Bourgeois. A Distributed Self-Assembly Planning Algorithm for Modular Robots. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 550–558, Stockholm, Sweden, July 2018. Association for Computing Machinery (ACM).
- [17] Thadeu Tucci, Benoît Piranda, and Julien Bourgeois. Efficient Scene Encoding for Programmable Matter Self-reconfiguration Algorithms. In *Proceedings of the Symposium on Applied Computing*, pages 256–261, 2017.
- [18] J. Werfel, K. Petersen, and R. Nagpal. Designing Collective Behavior in a Termite-Inspired Robot Construction Team. *Science*, 343(6172):754–758, February 2014.
- [19] Paul White, Victor Zykov, Josh Bongard, and Hod Lipson. Three Dimensional Stochastic Reconfiguration of Modular Robots. In *Robotics: Science and Systems I*. Robotics: Science and Systems Foundation, June 2005.
- [20] Zipeng Ye, Minjing Yu, and Yong-Jin Liu. NP-completeness of optimal planning problem for modular robots. *Autonomous Robots*, July 2019.
- [21] Mark Yim, Ying Zhang, John Lamping, and Eric Mao. Distributed Control for 3D Metamorphosis. *Autonomous Robots*, 10(1):41–56, January 2001.