Efficient Scene Encoding for Programmable Matter Self-Reconfiguration Algorithms

Thadeu Tucci, Benoît Piranda and Julien Bourgeois
FEMTO-ST Institute, UMR CNRS 6174
Univ. Bourgogne Franche-Comté (UBFC), France
University of Franche-Comté (UFC), Montbéliard, France
thadeu.tucci@femto-st.fr, benoit.piranda@femto-st.fr, julien.bourgeois@femto-st.fr

ABSTRACT

Programmable matter can be seen as a huge modular robot in which each module can communicate to its connected neighbors and work all together to achieve a common goal, more likely changing the shape of the whole robot. However, when the number of modules increases, the memory used in each module to store the target shape or the computation time to recreate this shape increases too. This article studies different approaches to describe the shape of any object for huge modular robots. The use of a good method for coding scene is a critical aspect that can reduce the memory, the time of transfer and the energy used in many distributed algorithms like self-reconfiguration. This paper proposes a method called Constructive Solid Geometry for Programmable Matter (CSG4PM), a compact description of an object and all the associated algorithms pre-processing and runtime. CSG4PM is compared to three existing solutions to describe a scene.

Keywords

programmable matter, large scale distributed robots, distributed algorithm, self-reconfiguration.

1. INTRODUCTION

An autonomous modular robot composes a distributed system in which micro-robots can communicate with their neighbors and work in a collaborative way to achieve common goals. The expected properties of modular robots [1] are: versatility, used to fulfill different tasks, robustness as a faulty module can be discarded, and affordable price as the mass production of identical modules is likely to reduce the overall cost.

Programmable matter is an interesting concept where computing entities exist to make decisions but also as a real physical instance. S.C. Goldstein and al. in [2] define Programmable Matter as a set of millions of millimeter robots with limited computer resources that can be reconfigured to another shape to create a synthetic reality.

This work is part of the Claytronics project [3] and uses the millimeter scale robots called Catoms (for Claytronics Atoms). These quasi-spherical robots can communicate with up to 12 neighbors, can move and change their color. The geometry of these robots and their abilities to communicate in a Face Centered Cubic Lattice are described in [4].

The most interesting capability of programmable matter is the ability to move each module in order to change the global shape or morphology of the whole, what is called self-reconfiguration. As a first step, the goal shape has to be transmitted to the system. There are actually 3 main solutions to deal with reconfiguration scenes on modular robots:

- Representation without coordinates [5]. This method does not need data transfer but can only describe some simple geometrical shapes.
- Shared model using a distributed shared memory system [6]. It can drastically reduce the size of data memorized in each module but needs communications.
- Compact representation is the solution we explore here. In a general way, the scene is first encoded and sent to a master module. Then, it is sent to all other modules of the network. The description is finally processed in each module in order to determine if the module is correctly placed or not. Compact representations are important in order to optimize memory space, reduce network bandwidth and to allow a faster broadcast of the scene.

Each robot is independent and must decide where and when to move using the knowledge of local neighborhood and distributed data only. They have the information of their position in the map, the current state of the configuration around their position and the final goal state.

For each step of a self-reconfiguration algorithm, each module needs to know if it is already in the goal map, or geometrical information that allows to know positions it must reach in order to reduce the global distance of the current configuration to the final one.

In this paper, we present Constructive Solid Geometry for Programmable Matter (CSG4PM) which is based on a CSG

SAC 2017,April 03-07, 2017, Marrakech, Morocco
Copyright 2017 ACM 978-1-4503-4486-9/17/04…$15.00
http://dx.doi.org/10.1145/3019612.3019706

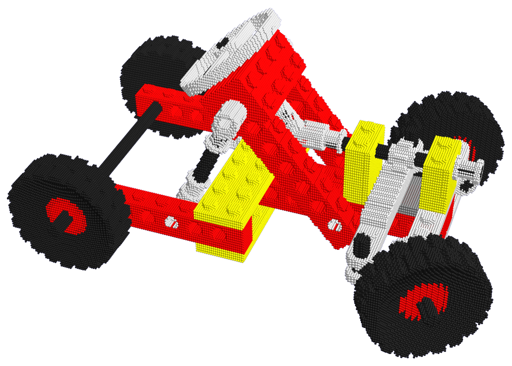
Figure 1: Huge set of 429,921 micro-robots defining a Toy Car.

representation and provides all the algorithms needed for accessing the data. For example, the Toy Car, constructed using CSG4PM and presented Figure 1 is defined by 427,921 3D Catoms.

2. RELATED WORKS

Our work falls into the field of self-reconfiguration of 3D lattice-based modular robots. Even though we focus on optimizing the scene description, lessons from previous work on self-reconfiguration can be learned.

In [7], Butler et al. published related work on mapping a configuration of modular robots. In their work, they use for the representation of the final configuration a binary matrix, with 0 corresponding to empty spaces and 1 to occupied spaces. It is a well-known way to represent the goal configuration, and, furthermore, it is easy to implement without any loss of details. In addition, a simple operation could tell when the module is inside the model. But, this representation depends on the number of modules and it will grow linearly with the size of the robots giving some restrictions on *scalability* and adaptability on object resize.

In [8], Park et al. propose an automatic configuration recognition in a centralized organization with chain-based modular robots, based in a graph where the nodes of the graph are the modules of the robots and the edges represent the connection between modules.

In the field of computer graphics, many solutions are proposed to describe 3D objects. Two different models are analyzed to be part of description scene for modular robots, Triangle Mesh and Constructive Solid Geometry.

Triangle Mesh is a very common representation of 3D objects, in literature we find it under the name of b-rep model (boundary representation) [9]. It consists in approximating the shape of an object by a set of small surfaces that define the border of the object, and then, interior and exterior spaces. The advantage of this description method is that we just have to describe a 2D surface in order to construct a 3D object, and it therefore needs less memory.

A wide number of 3D image software use Triangle Mesh despite the fact that this solution does not guarantee the final object to be a solid. The representation of the surface

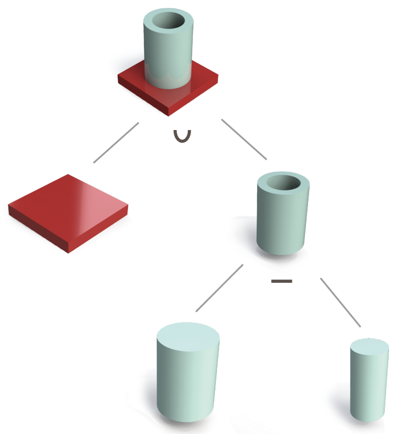

Figure 2: Mug CSG Tree.

is largely used to render objects on a screen but it may not be the best representation for our problem. Indeed, verifying if a module position is inside an object is complex as an object is described as a 2D surface.

An interesting idea was proposed by Stoy [10] et al. and followed by Fitch et al. [11], to transform a CAD model, that is a largely used 3D format in industry that includes Triangle Mesh, into a set of overlapping bricks. It is therefore easier for the modules to identify if their position is inside the model. Each brick can be represented by two coordinates which reduces the size of the final model. But, bricks does not produce a high quality representation of the object and to increase fidelity we have to work with smaller bricks which can increase dramatically the size of the model.

Constructive Solid Geometry (CSG) [12] is a classical method for describing scenes in image synthesis. It consists in defining a tree of objects that can be combined in order to model the final scene. Leaves of the tree contain geometric models and internal nodes are associated to geometrical transformations or combination operators. Geometrical transformations are useful to apply displacements, rotations or scales in a sub scene, they are placed in unary internal node of the tree. Three combination operators are usually used: union, intersection and difference.

The union of many objects is the volume filled by at least one of the objects, the intersection of many objects is defined by the common volume of all objects, and the difference $A - B$ is the volume of A that is not in B. These n-ary operators are detailed in Equation 1.

$$\begin{aligned} Union(B_1, B_2, ..., B_n) = & \quad B_1 \cup B_2 \cup ... \cup B_n \\ Inter(B_1, B_2, ..., B_n) = & \quad B_1 \cap B_2 \cap ... \cap B_n \\ Diff(B_1, B_2, ..., B_n) = & \quad B_1 \cap \neg (B_2 \cup ... \cup B_n) \end{aligned} \quad (1)$$

Figure 2 shows an example of CSG tree constructing a simple "mug" scene, this example uses 2 different operators (union and difference). Coding a scene using a CSG tree is very compact, because it consists in defining the volume occupied by the matter of the scene. Each object may be a simple geometric entity that can be described by some intrinsic parameters and placed using a homogeneous transformation

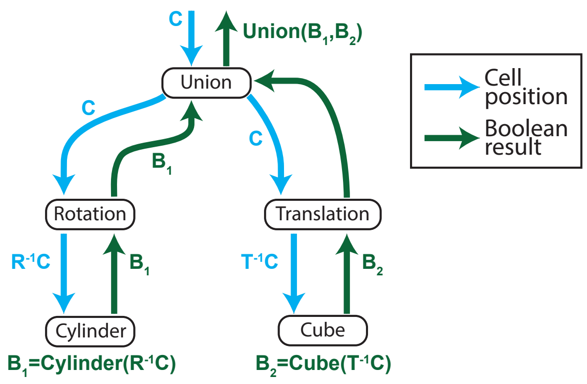

Figure 3: Example of traversing a CSG tree to determine if the point C is inside the described object

matrix. For example, a sphere is just defined by its radius, a cylinder needs a radius and a height. However, describing a complex scene using CSG tree becomes harder when it contains small details. In our case, the smallest size of a detail is the size of a module.

3. THE "CONSTRUCTIVE SOLID GEOMETRY FOR PROGRAMMABLE MATTER" METHOD

In order to build the goal configuration C_G, using a self-reconfiguration algorithm, each module of the set must know if it is already inside this goal configuration or not. The objective of Constructive Solid Geometry for Programmable Matter (CSG4PM) is to provide the best tradeoff between fidelity to the original shape, memory footprint and decoding processing time to fulfill this task. This a three-step process as it can be seen in Figure 4. The first step requires central computation to encode a 3D object into a set of modules. This implies a kind of discretization which will imply a loss of fidelity. The size and the number of modules are the two parameters that will influence the fidelity: the smaller the size and the higher the number of modules, the better the fidelity. But, the method chosen to represent the goal configuration also matters and have an impact on the fidelity.

In order to reduce memory used for the goal configuration while keeping a high level of detail, we choose to encode it in a vectorial compact format based on CSG Trees. Our CSG Tree is composed of four distinct nodes: primitive shapes, transformations, color and boolean operators.

- Primitive shapes are located on the leafs. Their parameters are the type of the shape (cube, sphere, cylinder, torus, etc.) and some associated intrinsic metrics, for example the ratio between small and large radius of the torus, the diameter of a sphere, etc.
- Geometrical transformations are placed in inner nodes. We consider translation, rotation, and scale transformations. Statistically in our test models, geometrical transformations are more frequently used directly on primitive shapes what lead us to make them accept only one child to reduce the representation size. Geometrical transformations are coded in a homogeneous matrix M.
- Color operators are unary inner nodes. They give the color of the subtree. The color of one node is given by the lowest color node.
- In order to reduce the height of our CSG trees, boolean operators are placed in inner nodes that can have $1 \dots n$ children.

We propose to transmit C_G model during the initial flooding algorithm that sends relative position of every module to the master module as shown in Figure 4. The flooding of information in a set of connected modules consists in sending step by step the data from the Master Module (MM) to all other modules and then waiting for an acknowledgment that every module has received the data. At the end of the flooding, MM receives the last acknowledgment. For positioning considerations, MM is empirically placed at the origin, it sends a relative position to each of its direct neighbors depending on the connector to which it is connected. Then these neighbors send a relative position to its other neighbors, and so on.

Considering a configuration coded by a CSG Tree, we define an algorithm that allows each module to solve in/out problem like knowing if the cell C is inside the model or not. The solution is obtained by a simple depth-first search algorithm:

- Traversing down, transmitted data are C coordinates.
- When visiting a geometrical transformation node, coordinates of C are converted into local coordinate of the subtree: $C_{subtree} = M^{-1}C$.
- When visiting a leaf node, simple geometrical calculation allows to defines if C is inside the geometrical shape coded in the leaf. For example, a simple distance calculation allows to know if C is in a sphere.
- During back tracking, the transmitted data is the boolean value indicating if the subtree is inside or out of the model. Arriving in a boolean operator node, the combination rule of Equation 1 is applied once every subtree has answered its intersection state B_i.

Figure 3 shows an example of crossing of a CSG Tree associating a rotated cylinder and a translated cube in order to deduce if a cell placed in C is in or out of the model. Blue arrows show cell coordinates transmitted when visiting a node for the first time, and green arrows are associated to boolean result (coding inside or not) after visiting a subtree.

In order to send C_G to all modules, we have to encode the tree structure into an array of bytes, send this array during flooding process, then each module must be able to decode the array of bytes to answer in/out problems. This array is obtained by a simple pre-order walk of the tree, each kind of node is coded in 1 byte followed by the intrinsics parameters (4 bytes float for real values). For n-airy nodes a "end of

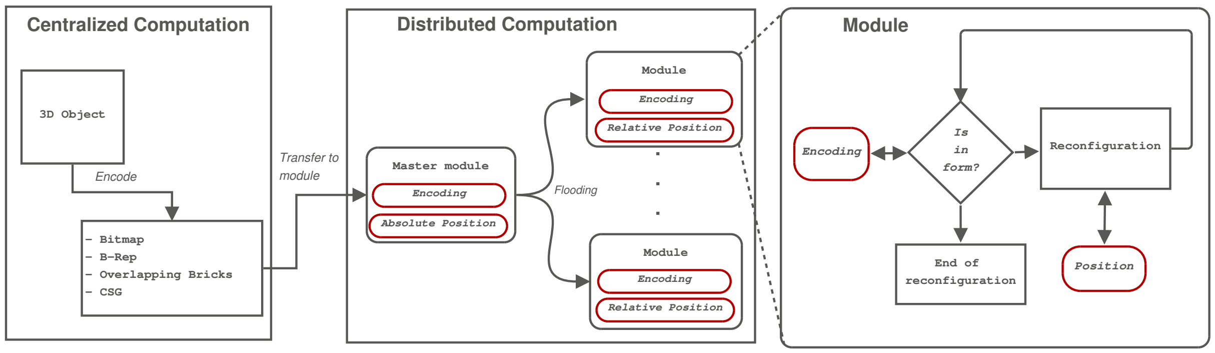

Figure 4: Global model flow used for reconfiguration.

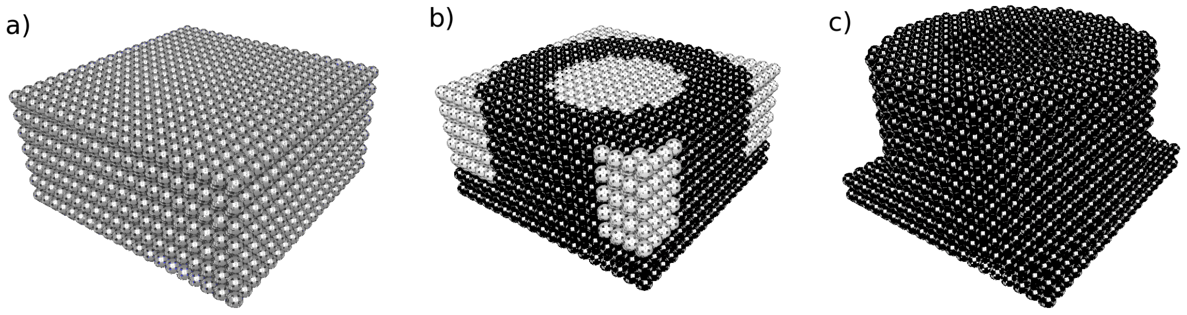

Figure 5: a) Initial configuration. b) Not well-placed modules are drawn in grey. c) After reconfiguration.

child code" (1 byte) is added in order to stop the crossing. The decoding process is obtained by a simple read of the array from the left to the right. This treatment is at least executed once during the flooding algorithm after reception of the map by each module, in order to know if the module is already well-placed in the map.

Figure 5 shows a simple model defined by a simple CSG Tree (detailed in Algorithm 1), the size of the array coding this CSG Tree is 65 bytes only. The first image shows an initial configuration (Figure 5.a) made of a block of $20 \times 20 \times 16$ catoms. Second image (5.b) shows catoms after receiving the map and evaluating locally if they are inside (black) or outside (light gray) the model. And finally after reconfiguration (Figure 5.c), i.e. displacement of light gray catoms of the previous image to a cell placed inside the model. This final object fills a larger volume than the initial one, because previous not well placed modules have moved to enlarge the parallelepipedic base and extend the cylindrical part to the top.

Toy Car presented in Figure 1 is our high definition test model. The associated CSG Tree is composed of 1757 nodes including 628 primitive shapes in its leafs and with a depth of 16. Using our coding method we obtain a 18757 bytes long array.

4. EXPERIMENTS

We compare our CSG based model with three classical methods: Bitmap model, Mesh model and Stoy overlapping bricks model. Bitmap model is the simplest solution consisting in storing the configuration bounding box of cells in an array. Each element of the array contains the color of the cell coded in 3 bytes, considering that the (0,0,0) color is reserved to code an empty cell. The advantage of this model is that it makes easy to deduce if a module is inside C_G or not. But it implies to transmit a huge memory from the master module to the others and do not allow to create objects with different scale without regenerating a new bitmap description.

Algorithm 1 Mug CSG Example

```
1: Color(0, 0, 0)
2:     Difference()
3:         Union()
4:             Cube([20, 20, 2.5]);
5:             Translate([10, 10, 2.5])
6:                 Cylinder(20, 10);
7:         Translate([10, 10, 2.5])
8:             Cylinder(20, 5);
```

Mesh model consists in describing an object by its border in the world coordinate system. It first describes a list of all vertex coordinates and then a list of all faces specifying vertices indices and face color. The complexity depends on the precision of the mesh to approximate the shape of the object. In this work we have to memorize 4×3 floats per vertex, 4×3 integers plus 3 bytes for color per face. This model implies a computational process in order to know if a module is inside or outside the mesh.

Definition 1. *A point is inside a closed object mesh if each ray starting from this point intersects borders an odd number of times.*

According to Definition 1, to calculate if a module is inside an object we define a ray from the center of the cell C, going in a random direction. Then we count the number of intersection between this ray and the list of faces of the mesh. We conclude that the cell C is inside the object if we obtain an odd number of intersections.

Experiments has been realized using our simulator, VisibleSim [13, 14]. They aim to compare the four models of coding in terms of memory, time of treatments and time of transfer.

We use two very different objects for this experimentation. The first one is a mug shape that admits low level of detail, surfaces are smooth and of large size. The second one is a toy car composed by complex elements with bumps and holes, presenting high level of detail.

These CSG models have been converted into overlapping bricks (with 3 different resolutions), triangle mesh and bitmap. Overlapping bricks in high resolution in our tests are defined with the smallest brick size having exactly the diameter of a catom. The medium resolution is defined by one and half catom diameter, and low resolution overlapping brick smallest size is twice the diameter of a catom. Bricks are defined by two vertices (using a float representations of 4 bytes per real number) and a color (coded in 1 byte). In total each brick uses 25 bytes to be stored.

4.1 Fidelity to the original format

The data structures used for comparisons are Bitmap, CSG, Overlapping bricks and Triangle mesh. The bitmap model is created according to the resolution of the lattice and CSG is the reference model, they can be considered as perfect. But triangle meshes approach curved surface by small planes faces, that, indeed small, can produce loss of fidelity. The smaller the faces are better is the quality, but higher is the size of coding. Overlapping bricks can have a significant loss of fidelity depending on the minimum brick size chosen.

Figure 6 shows the quality of models for several resolutions and the two experimented objects.

We can see that low resolution brick can generate important errors compared to CSG model and errors in Mesh model may be neglected. We consider that these models are enough similar to be used in the following experiments.

4.2 Comparison of code sizes

For bitmap model the code size S of 3 bytes for the colored position depends directly on the size ($C_n = n^3$) of the cubic lattice: $S = 3n^3\ bytes$. In the case of the Mug Model, coded with 61 *bytes* using CSG model, a cubic lattice C_3 of $n = 3$ cells per edge needs $S = 81\ bytes$, that is more than the corresponding CSG code size. For the second test model (the Toy Car, coded by $18,757$ *bytes*) a C_{19} cubic lattice needs more memory to be coded by a bitmap.

CSG, Mesh and overlapping bricks methods are vectorial, the size of the code is invariant. For overlapping bricks method we produce three descriptions corresponding to different level of subdivisions (called High, Medium and Low).

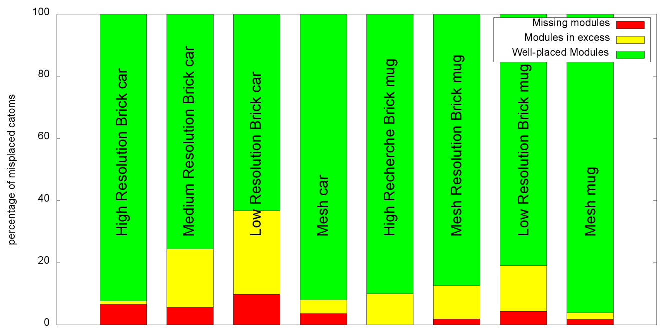

Figure 6: Fidelity of the Toy Car structure using overlapping bricks

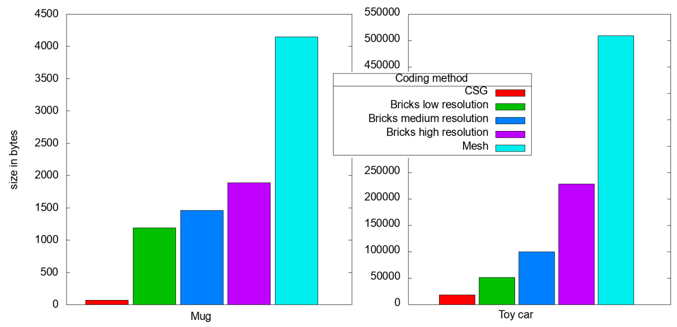

Figure 7: Size of codes using different vectorial encoding for 2 different structures

Figure 7 shows that CSG model gives very small size of code compared to other models. Comparing the small representation size with the fidelity proposed by the others data structures, CSG shows great results.

In order to be able to compare the bitmap storage we evaluate the memory used in each robot to represent the Toy car model presented Figure 1. The lattice used is $145 \times 229 \times 167$ cells large, that represents $5,545,235$ cells to memorize. Then bitmap needs 15.8 Mo to memorize the model. It is about 886 times more than the size of the code of the Toy Car using our CSG4PM method.

4.3 Decoding process time

Vectorial models are interesting in terms of size of coded data but they need a decoding process before using. This process may be important in term of decompression time. Figure 8 shows a comparison of average computation time of decoding task for the two studied models.

We can observe that computation time with bitmap model is a few nanoseconds, it simply consist in accessing to the cell of an 3D array. But CSG model drawn in red (that has the most compact encoding) gives a very good time of decoding compared to Bricks and Mesh models. The complexity of the decoding process is linear in the number of nodes in the CSG Tree. In practice, the decoding time of CSG Tree

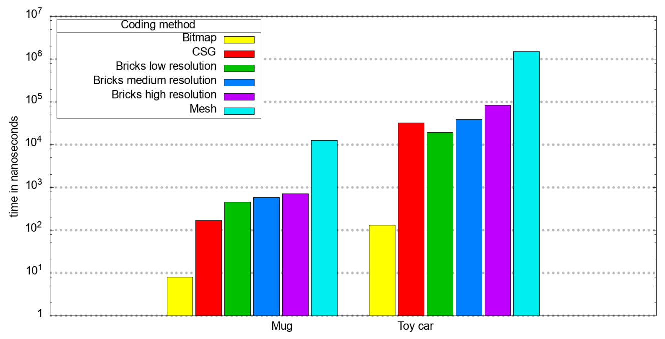

Figure 8: Time of decoding

may be neglected compared to communication time used to exchange data between modules. Latencies from 1 to 10 milliseconds are not unexpected in this type of network communication.

5. CONCLUSION

In this paper, we present an efficient method to reduce the memory used in each module for storing the goal map needed by reconfiguration processes of Programmable Matter. We compared our CSG4PM model with three existing methods for two very different objects both in size and in terms of complexity. We show that the gain is very significant compared to classical Bitmap models, and our model need less decoding time for better compression size compared to Mesh based method or overlapping bricks method.

As a future works, we think that CSG tree can be dynamical simplified for some area of the scene that are not concerned by a part of the geometrical model. We will work on a method that cut off the received CSG tree in each Catom before sending the different parts to its neighbors.

Acknowledgment

This work has been funded by CNPq, the Labex ACTION program (contract ANR-11-LABX-01-01).

6. REFERENCES

[1] M. Yim, Y. Zhang, and D. Duff, "Modular robots," *IEEE Spectrum*, vol. 39, no. 2, pp. 30–34, 2002.

[2] S. C. Goldstein, J. D. Campbell, and T. C. Mowry, "Programmable matter," *Computer*, vol. 38, no. 6, pp. 99–101, 2005.

[3] S. C. Goldstein and T. C. Mowry, "Claytronics: A scalable basis for future robots," 2004.

[4] B. Piranda and J. Bourgeois, "Geometrical study of a quasi-spherical module for building programmable matter," in *DARS 2016, 13th International Symposium on Distributed Autonomous Robotic Systems*, London, United Kingdom, Nov. 2016, pp. –.

[5] H. Lakhlef, H. Mabed, and J. Bourgeois, "Distributed and Dynamic Map-less Self-reconfiguration for Microrobot Networks," in *2013 12th IEEE International Symposium on Network Computing and Applications (NCA)*, Aug. 2013, pp. 55–60.

[6] J. Bourgeois, B. Piranda, A. Naz, H. Lakhlef, N. Boillot, H. Mabed, D. Douthaut, and T. Tucci, "Programmable matter as a cyber-physical conjugation," in *IEEE International Conference on Systems, Man and Cybernetics (SMC'16), Budapest, Hungary*, IEEE, Ed., October 2016.

[7] Z. Butler, R. Fitch, D. Rus, and Y. Wang, "Distributed goal recognition algorithms for modular robots," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 1. IEEE, 2002, pp. 110–116.

[8] M. Park, S. Chitta, A. Teichman, and M. Yim, "Automatic configuration recognition methods in modular robots," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 403–421, 2008.

[9] J. D. Foley, A. v. Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics (2nd edn in C): Principles and Practice*, Jan. 1996.

[10] K. Stoy and R. Nagpal, "Self-Reconfiguration Using Directed Growth," in *Distributed Autonomous Robotic Systems 6*, R. Alami, R. Chatila, and H. Asama, Eds. Springer Japan, 2007, pp. 3–12, dOI: 10.1007/978-4-431-35873-2_1.

[11] R. Fitch and Z. Butler, "Million Module March: Scalable Locomotion for Large Self-Reconfiguring Robots," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 331–343, Mar. 2008. [Online]. Available: http://ijr.sagepub.com/content/27/3-4/331

[12] A. G. Requicha, "Representations for Rigid Solids: Theory, Methods, and Systems," *ACM Comput. Surv.*, vol. 12, no. 4, pp. 437–464, Dec. 1980.

[13] D. Dhoutaut, B. Piranda, and J. Bourgeois, "Efficient simulation of distributed sensing and control environments," in *iThings 2013, IEEE Int. Conf. on Internet of Things*, Beijing, China, Aug. 2013, pp. 452–459.

[14] B. Piranda, "Visiblesim: Your simulator for programmable matter," in *Algorithmic Foundations of Programmable Matter (Dagstuhl Seminar 16271)*, S. Fekete, A. Richa, K. Römer, and C. Scheideler, Eds., Jul. 2016.