Convex Hull Formation for Programmable Matter

Joshua J. Daymude
Arizona State University
Computer Science, CIDSE
Tempe, AZ, USA
jdaymude@asu.edu

Robert Gmyr
University of Houston
Department of Computer Science
Houston, TX, USA
rgmyr@uh.edu

Kristian Hinnenthal
Paderborn University
Department of Computer Science
Paderborn, Germany
krijan@mail.upb.de

Irina Kostitsyna
TU Eindhoven
Department of Mathematics and Computer Science
Eindhoven, The Netherlands
i.kostitsyna@tue.nl

Christian Scheideler
Paderborn University
Department of Computer Science
Paderborn, Germany
scheidel@mail.upb.de

Andréa W. Richa
Arizona State University
Computer Science, CIDSE
Tempe, AZ, USA
aricha@asu.edu

ABSTRACT

We envision *programmable matter* as a system of nano-scale agents (called *particles*) with very limited computational capabilities that move and compute collectively to achieve a desired goal. Motivated by the problem of sealing an object using minimal resources, we show how a particle system can self-organize to form an object's convex hull. We give a distributed, local algorithm for convex hull formation and prove that it runs in $O(B)$ asynchronous rounds, where B is the length of the object's boundary. Within the same asymptotic runtime, this algorithm can be extended to also form the object's (weak) O-hull, which uses the same number of particles but minimizes the area enclosed by the hull. Our algorithms are the first to compute convex hulls with distributed entities that have *strictly local sensing, constant-size memory, and no shared sense of orientation or coordinates.* Ours is also the first distributed approach to computing restricted-orientation convex hulls. This approach involves coordinating particles as distributed memory; thus, as a supporting but independent result, we present and analyze an algorithm for organizing particles with constant-size memory as distributed binary counters that efficiently support increments, decrements, and zero-tests — even as the particles move.

CCS CONCEPTS

• **Theory of computation → Self-organization**; *Distributed algorithms*; *Computational geometry*.

KEYWORDS

programmable matter, self-organization, distributed algorithms, computational geometry, convex hull, restricted-orientation geometry

ICDCN 2020, January 4–7, 2020, Kolkata, India
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7751-5/20/01…$15.00
https://doi.org/10.1145/3369740.3372916

ACM Reference Format:
Joshua J. Daymude, Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Christian Scheideler, and Andréa W. Richa. 2020. Convex Hull Formation for Programmable Matter. In *21st International Conference on Distributed Computing and Networking (ICDCN 2020), January 4–7, 2020, Kolkata, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3369740.3372916

1 INTRODUCTION

The vision for *programmable matter* [25] is to realize a physical material that can dynamically alter its properties (shape, density, conductivity, etc.) in a programmable fashion, controlled either by user input or its own autonomous sensing of its environment. Such systems would have broad engineering and societal impact with applications such as reusable construction materials, self-repairing spacecraft components, and even nanoscale medical devices. While the form factor of each programmable matter system would vary widely depending on its intended application domain, a budding theoretical investigation has formed over the last decade into the algorithmic underpinnings common among these systems. In particular, the unifying inquiry is to better understand what *sophisticated, collective behaviors* are achievable by a programmable matter system composed of *simple, limited computational units.* Towards this goal, many theoretical works, complementary simulations, and even a recent experimental study [24] have been conducted using the *amoebot model* [8] for *self-organizing particle systems.*

In this paper, we give a local, distributed algorithm for *convex hull formation* (formally defined within our context in Section 1.2) under the amoebot model. Though this well-studied problem is usually considered from the perspectives of computational geometry and combinatorial optimization as an abstraction, we treat it as the task of forming a physical seal around a static object using as few particles as possible. This is an attractive behavior for programmable matter, as it would enable systems to, for example, isolate and contain oil spills [28], mimic the collective transport capabilities seen in ant colonies [17, 18], or surround and engulf malignant entities in the human body as phagocytes do [1]. Though our algorithm is certainly not the first distributed approach taken to computing convex hulls, to our knowledge it is the first to do so with distributed computational entities that have *no sense of global orientation nor of their coordinates* and are limited to only *local sensing and constant-size memory.* Moreover, to our knowledge ours is

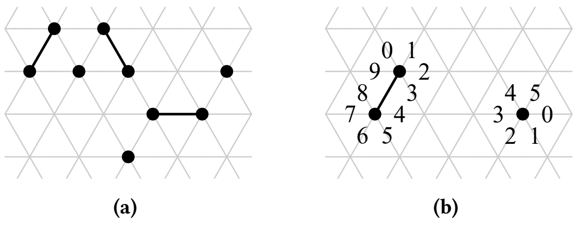


Figure 1: (a) Expanded and contracted particles (black circles) on the triangular lattice G_Δ (gray). Particles with a black line between their nodes are expanded. (b) Two particles with different offsets for their port labels.

the first distributed approach to computing restricted-orientation convex hulls, a generalization of usual convex hulls (see definitions in Section 1.1). Finally, our algorithm has a gracefully degrading property: when the number of particles is insufficient to form an object's convex hull, a maximal partial convex hull is still formed.

Due to space constraints, many details are omitted from this conference paper. A complete version of this paper with all proofs, extensions, and pseudocode can be found on arXiv [6].

1.1 The Amoebot Model

In the *amoebot model* [8],[1] programmable matter consists of individual, homogeneous computational elements called *particles*. Any structure that a particle system can form is represented as a subgraph of an infinite, undirected graph $G = (V, E)$ where V represents all positions a particle can occupy and E represents all atomic movements a particle can make. Each node can be occupied by at most one particle. The *geometric amoebot model* further assumes $G = G_\Delta$, the triangular lattice (Figure 1a).

Each particle occupies either a single node in V (i.e., it is *contracted*) or a pair of adjacent nodes in V (i.e., it is *expanded*), as in Figure 1a. Particles move via a series of *expansions* and *contractions*: a contracted particle can expand into an unoccupied adjacent node to become expanded, and completes its movement by contracting to once again occupy a single node. An expanded particle's *head* is the node it last expanded into and the other node it occupies is its *tail*; a contracted particle's head and tail are both the single node it occupies.

Two particles occupying adjacent nodes are said to be *neighbors*. Neighboring particles can coordinate their movements in one of two types of *handovers*. A contracted particle P can "push" an expanded neighbor Q by expanding into a node occupied by Q, forcing it to contract. Alternatively, an expanded particle Q can "pull" a contracted neighbor P by contracting, forcing P to expand into the node it is vacating.

Each particle keeps a collection of ports — one for each edge incident to the node(s) it occupies — that have unique labels from its own local perspective. Although each particle is *anonymous*, lacking a unique identifier, a particle can locally identify any given neighbor by its labeled port corresponding to the edge between them. Particles do not share a coordinate system or global compass and may have different offsets for their port labels, as in Figure 1b.

Each particle has a constant-size local memory that it and its neighbors can directly read from and write to for communication.[2] However, particles do not have any global information and — due to the limitation of constant-size memory — cannot locally count or estimate the total number of particles in the system.

The system progresses asynchronously through *atomic actions*. In the amoebot model, an atomic action corresponds to a single particle's activation, in which it can (*i*) perform a constant amount of local computation involving information it reads from its local memory and its neighbors' memories, (*ii*) directly write updates to at most one neighbor's memory, and (*iii*) perform at most one expansion or contraction. We assume these actions preserve *atomicity*, *isolation*, *fairness*, and *reliability*. Atomicity requires that if an action is aborted before its completion (e.g., due to a conflict), any progress made by the particle(s) involved in the action is completely undone. A set of concurrent actions preserves isolation if they do not interfere with each other; i.e., if their concurrent execution produces the same end result as if they were executed in any sequential order. Fairness requires that each particle successfully completes an action infinitely often. Finally, for this work, we assume reliability, meaning all particles are non-faulty.

While it is straightforward to ensure atomicity and isolation in each particle's immediate neighborhood (using a simple locking mechanism), particle writes and expansions can influence the 2-neighborhood and thus must be handled carefully.Conflicts of movement can occur when multiple particles attempt to expand into the same unoccupied node concurrently. These conflicts are resolved arbitrarily such that at most one particle expands into a given node at any point in time.

It is well known that if a distributed system's actions are atomic and isolated, any set of such actions can be *serialized* [4]; i.e., there exists a sequential ordering of the successful (non-aborted) actions that produces the same end result as their concurrent execution. Thus, while in reality many particles may be active concurrently, it suffices when analyzing amoebot algorithms to consider a sequence of activations where only one particle is active at a time. By our fairness assumption, if a particle P is inactive at time t in the activation sequence, P will be (successfully) activated again at some time $t' > t$. An *asynchronous round* is complete once every particle has been activated at least once.

Additional Terminology for Convex Hulls. We now define some terminology specific to our application of convex hull formation. An *object* $O \subset V$ is a static, finite, simply connected set of nodes. The *boundary* $B(O)$ of an object O is the set of all nodes in $V \setminus O$ that are adjacent to O. An object contains a *tunnel of width* 1 if its boundary is 1-connected. We assume particles can differentiate between object nodes and nodes occupied by other particles.

To generalize the notions of convexity and convex hulls to our discrete setting on the triangular lattice, we introduce the concepts of *restricted-orientation convexity* (also known as *O-convexity*) and *strong restricted-orientation convexity* (or *strong O-convexity*) which are well established in computational geometry [13, 23]. In the

[1]See [8] for a full description of the model including omitted details that are not necessary for convex hull formation.

[2]Here, we assume the *direct write communication* extension of the amoebot model as it enables a simpler description of our algorithms; see [8] for details.

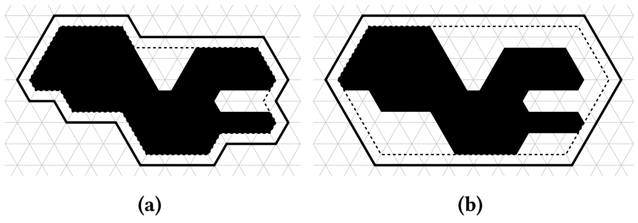


Figure 2: An object O (black) with a tunnel of width 1 on its right side and its (a) $\mathcal{O}$-hull (dashed line) and $\mathcal{O}_\Delta$-hull $H'(O)$ (solid black line), and (b) strong $\mathcal{O}$-hull (dashed line) and strong $\mathcal{O}_\Delta$-hull $H(O)$ (solid black line).

continuous setting, given a set of orientations $\mathcal{O}$ in $\mathbb{R}^2$, a geometric object is said to be *$\mathcal{O}$-convex* if its intersection with every line with an orientation from $\mathcal{O}$ is empty or connected. The *$\mathcal{O}$-hull* of an object A is the intersection of all $\mathcal{O}$-convex sets containing A, or, equivalently, the minimal $\mathcal{O}$-convex set containing A. An *$\mathcal{O}$-block* of two points in $\mathbb{R}^2$ is the intersection of all half-planes defined by lines with orientations in $\mathcal{O}$ that contain both points. The *strong $\mathcal{O}$-hull* of a geometric object A is the minimal $\mathcal{O}$-block containing A.

For our discrete setting, let $\mathcal{O}$ be the *orientation set* of G_Δ, i.e., the three orientations of axes of the triangular lattice. The *(weak) $\mathcal{O}_\Delta$-hull* of object O, denoted $H'(O)$, is the set of nodes in $V \setminus O$ adjacent to the $\mathcal{O}$-hull of O in $\mathbb{R}^2$ (Figure 2a). Analogously, the *strong $\mathcal{O}_\Delta$-hull* of object O, denoted $H(O)$, is the set of nodes in $V \setminus O$ adjacent to the strong $\mathcal{O}$-hull of O in $\mathbb{R}^2$ (Figure 2b). We offset the hulls from their traditional definitions by one layer of nodes since particles cannot occupy nodes of O. Unless there is a risk of ambiguity, we will use the terms “strong $\mathcal{O}_\Delta$-hull” and “convex hull” interchangeably throughout this work.

1.2 Our Results

An instance of the *strong $\mathcal{O}_\Delta$-hull (convex hull) formation problem* has the form $(\mathcal{P}, O)$ where $\mathcal{P}$ is a finite, connected system of initially contracted particles and $O \subset V$ is an object. Let $B = |B(O)|$ denote the length of the object's boundary and $H = |H(O)|$ denote the length of the object's convex hull. We assume that (*i*) $\mathcal{P}$ contains a unique leader particle ℓ initially adjacent to O,[3] (*ii*) there are at least $|\mathcal{P}| > \log_2(H)$ particles in the system, and (*iii*) O does not have any tunnels of width 1.[4] A local, distributed algorithm $\mathcal{A}$ solves an instance $(\mathcal{P}, O)$ of the convex hull formation problem if, when each particle executes $\mathcal{A}$ individually, $\mathcal{P}$ is reconfigured so that every node of $H(O)$ is occupied by a contracted particle. The *$\mathcal{O}_\Delta$-hull formation problem* can be stated analogously.

We present a **local, distributed algorithm** for the **strong $\mathcal{O}_\Delta$-hull formation problem** that runs in **$O(B)$ rounds** and later show how it can be extended to also solve the **$\mathcal{O}_\Delta$-hull formation problem** in an additional **$O(H)$ rounds**. Our algorithm is gracefully degrading: if there are insufficient particles to completely fill the convex hull with contracted particles (i.e., if $|\mathcal{P}| < H$) our algorithm will still form a maximal partial convex hull. To our knowledge, our algorithm is the first to address distributed convex hull formation using entities that have no sense of global orientation nor of their coordinates and are limited to only constant-size memory and local communication. It is also the first distributed algorithm for forming restricted-orientation convex hulls (see Section 1.3).

Our approach relies on the leader maintaining and updating the distances from its current position to each of the half-planes whose intersection composes the object's convex hull. However, these distances can far exceed the constant-size memory capacity of an individual particle. To address this problem, we give new results on coordinating a particle system as a distributed binary counter that supports increments and decrements by one as well as *zero-testing*, or testing the counter value's equality to zero. These results supplant existing work on increment-only distributed binary counters under the amoebot model [21]. Moreover, these results are agnostic of convex hull formation and can be used as a modular primitive for future applications.

1.3 Related Work

The convex hull problem is one of the best-studied problems in computational geometry. Many parallel algorithms have been proposed to solve it (e.g., [2, 12, 14]), as have several distributed algorithms [11, 19, 22]. However, conventional models of parallel and distributed computation assume that the computational and communication capabilities of the individual processors far exceed those of individual particles of programmable matter. Most commonly, processors are assumed to know their global coordinates and can communicate non-locally. Particles in the amoebot model have only constant-size memory and can communicate only with their immediate neighbors. Furthermore, the object's boundary may be much larger than the number of particles, making it impossible for the particle system to store all the geographic locations. Finally, to our knowledge, there only exist centralized algorithms to compute (strong) restricted-orientation convex hulls (see, e.g., [16] and the references therein); ours is the first to do so in a distributed setting.

The amoebot model for self-organizing particle systems is an *active* system of programmable matter — in which the computational units have control over their own movements and actions — as opposed to a *passive* system such as population protocols and models of molecular self-assembly (e.g., [3, 20]). Other active systems include modular self-reconfigurable robot systems (e.g., [27] and the references therein), the nubot model for molecular computing [26], and mobile robots (see [15] and the references therein) where robots abstracted as points in the real plane or on graphs solve problems such as pattern formation and gathering. A notable difference between the amoebot model and the mobile robots literature is in their treatment of progress and time: mobile robots progress according to fine-grained “look-compute-move” cycles where actions are comprised of exactly one look, move, or compute operation. In comparison — at the scale where particles can only perform a constant amount of computation and are restricted to immediate neighborhood sensing — the amoebot model assumes coarser atomic actions (as described in Section 1.1).

[3]One could use the leader election algorithm for the amoebot model in [7] to obtain such a leader in $O(|\mathcal{P}|)$ asynchronous rounds, with high probability. Removing this assumption would simply change all the deterministic guarantees given in this work to guarantees with high probability.

[4]We believe our algorithm could be extended to handle tunnels of width 1 in object O, but this would require technical details beyond the scope of this conference paper.

Lastly, we distinguish convex hull formation from the related problems of shape formation and object coating, both of which have been studied under the amoebot model. Like shape formation [9], convex hull formation is a task of reconfiguring a particle system's shape; however, the desired hull shape is based on the object and thus is not known to the particles ahead of time. Object coating [10] also depends on an object, but may not form a convex seal using the minimum number of particles.

1.4 Organization

Our convex hull formation algorithm has two phases: the particle system first explores the object to learn the convex hull's dimensions, and then uses this knowledge to form the convex hull. In Section 2, we introduce the main ideas behind the learning phase as a novel local algorithm run by a single particle with unbounded memory. We then give new results on organizing a system of particles each with $O(1)$ memory into binary counters in Section 3. Combining the results of these two sections, we present the full distributed algorithm for learning and forming the strong O_Δ-hull in Section 4. We conclude by presenting an extension of our algorithm to solve the O_Δ-hull formation problem in Section 5.

2 THE SINGLE-PARTICLE ALGORITHM

We first consider a particle system composed of a single particle P with unbounded memory and present a local algorithm for learning the strong O_Δ-hull of object O. As will be the case in the distributed algorithm, particle P does not know its global coordinates or orientation. We assume P is initially on $B(O)$, the boundary of O. The main idea of this algorithm is to let P perform a clockwise traversal of $B(O)$, updating its knowledge of the convex hull as it goes.

In particular, the convex hull can be represented as the intersection of six half-planes $\mathcal{H} = \{N, NE, SE, S, SW, NW\}$, which P can label using its local compass (see Figure 3). Particle P estimates the location of these half-planes by maintaining six counters $\{d_h : h \in \mathcal{H}\}$, where each counter d_h represents the L_1-distance from the position of P to half-plane h. If at least one of these counters is equal to 0, P is on its current estimate of the convex hull.

Each counter is initially set to 0, and P updates them as it moves. Let $[6] = \{0, \ldots, 5\}$ denote the six directions P can move in, corresponding to its contracted port labels. In each step, P first computes the direction $i \in [6]$ to move toward using the right-hand rule, yielding a clockwise traversal of $B(O)$. Since O was assumed to not have tunnels of width 1, direction i is unique. Particle P then updates its distance counters by setting $d_h \leftarrow \max\{0, d_h + \delta_{i,h}\}$ for all $h \in \mathcal{H}$, where $\delta_i = (\delta_{i,N}, \delta_{i,NE}, \delta_{i,SE}, \delta_{i,S}, \delta_{i,SW}, \delta_{i,NW})$ is defined as follows:

$$\delta_0 = (1, 1, 0, -1, -1, 0) \qquad \delta_1 = (0, 1, 1, 0, -1, -1)$$
$$\delta_2 = (-1, 0, 1, 1, 0, -1) \qquad \delta_3 = (-1, -1, 0, 1, 1, 0)$$
$$\delta_4 = (0, -1, -1, 0, 1, 1) \qquad \delta_5 = (1, 0, -1, -1, 0, 1)$$

Thus, every movement decrements the distance counters of the two half-planes to which P gets closer and increments the distance counters of the two half-planes from which P gets farther away. Whenever P moves toward a half-plane h for which $d_h = 0$, the distance stays 0, essentially "pushing" the estimation of the half-plane one step further (see Figure 4).

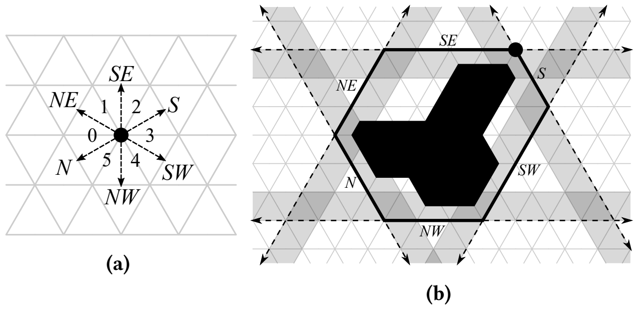


Figure 3: (a) A particle's local labeling of the six half-planes composing the convex hull: the half-plane between its local 0 and 5-labeled edges is N, and the remaining half-planes are labeled accordingly. (b) An object (black) and the six half-planes (dashed lines with shading on included side) whose intersection forms its convex hull (black line). As an example, the node depicted in the upper-right is distance 0 from the S and SE half-planes and distance 7 from N.

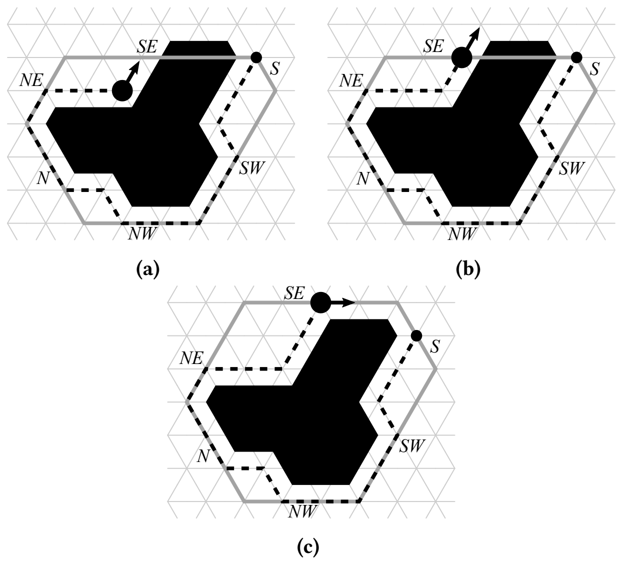


Figure 4: The particle P with its convex hull estimate (gray line) after traversing the path (dashed line) from its starting point (small black dot). (a) $d_h \geq 1$ for all $h \in \mathcal{H}$, so its next move does not push a half-plane. (b) Its next move is toward the SE half-plane and $d_{SE} = 0$, so (c) SE is pushed.

Finally, P needs to detect when it has learned the complete convex hull. To do so, it stores six terminating bits $\{b_h : h \in \mathcal{H}\}$, where b_h is equal to 1 if P has visited half-plane h (i.e., if d_h has been 0) since P last pushed any half-plane, and 0 otherwise. Whenever P moves without pushing a half-plane (e.g., Figure 4a–4b), it sets $b_h \leftarrow 1$ for all h such that $d_h = 0$ after the move. If its move pushed a half-plane (e.g., Figure 4b–4c), it resets all its terminating bits to 0. Once all six terminating bits are 1, P contracts and terminates.

Analysis. We now analyze the correctness and runtime of this single-particle algorithm. For a given round i, let $H_i(O) \subset V$ be the set of all nodes enclosed by P's estimate of the convex hull of O after round i, i.e., all nodes in the closed intersection of the six half-planes. We first show that P's estimate of the convex hull represents the correct convex hull $H(O)$ after at most one traversal of the object's boundary, and does not change afterwards.

Lemma 2.1. *If P completes its traversal of $B(O)$ in round i^*, then $H_i(O) = H(O)$ for all $i \geq i^*$.*

Proof. P exclusively traverses $B(O)$, so $H_i(O) \subseteq H(O)$ for all rounds i. Furthermore, $H_i(O) \subseteq H_{i+1}(O)$ for any round i. Once P has traversed the whole boundary, it has visited a node of each half-plane corresponding to $H(O)$, and thus $H_{i^*}(O) = H(O)$. □

We now show particle P terminates if and only if it has learned the complete convex hull.

Lemma 2.2. *If $H_i(O) \subset H(O)$ after some round i, then $b_h = 0$ for some half-plane $h \in \mathcal{H}$.*

Proof. Suppose to the contrary that after round i, $H_i(O) \subset H(O)$ but $b_h = 1$ for all $h \in \mathcal{H}$; let i be the first such round. Then after round $i - 1$, there was exactly one half-plane $h_1 \in \mathcal{H}$ such that $b_{h_1} = 0$; all other half-planes $h \in \mathcal{H} \setminus \{h_1\}$ have $b_h = 1$. Let $h_2, \ldots, h_6$ be the remaining half-planes in clockwise order, and let round $t_j < i - 1$ be the one in which b_{h_j} was most recently flipped from 0 to 1, for $2 \leq j \leq 6$. Particle P could only set $b_{h_j} = 1$ in round t_j if its move in round t_j did not push any half-planes and $d_{h_j} = 0$ after the move. There are two ways this could have occurred.

First, P may have already had $d_{h_j} = 0$ in round $t_j - 1$ and simply moved along h_j in round t_j, leaving $d_{h_j} = 0$. But for this to hold and for P to have had $b_{h_j} = 0$ after round $t_j - 1$, P must have just pushed h_j, resetting all its terminating bits to 0. Particle P could not have pushed any half-plane during rounds t_2 up to $i - 1$, since $b_{h_2} = \cdots = b_{h_6} = 1$, so this case only could have occurred with half-plane h_2.

For the remaining half-planes h_j, for $3 \leq j \leq 6$, P must have had $d_{h_j} = 1$ in round $t_j - 1$ and moved into h_j in round t_j. But this is only possible if P pushed h_j in some round prior to $t_j - 1$, implying that P has already visited $h_3, \ldots, h_6$. Therefore, P has completed at least one traversal of $B(O)$ by round i, but $H_i(O) \subset H(O)$, contradicting Lemma 2.1. □

Lemma 2.3. *Suppose $H_i(O) = H(O)$ for the first time after some round i. Then P terminates at some node of $H(O)$ after at most one additional traversal of $B(O)$.*

Proof. Since i is the first round in which $H_i(O) = H(O)$, particle P must have just pushed some half-plane h — resetting all its terminating bits to 0 — and now occupies a node u with distance 0 to h. Due to the geometry of the triangular lattice, the next node in a clockwise traversal of $B(O)$ from u must also have distance 0 to h, so P will set b_h to 1 after its next move. As P continues its traversal, it will no longer push any half-planes because its convex hull estimation is complete. Thus, P will visit every other half-plane h' without pushing it, causing P to set each $b_{h'}$ to 1 before reaching u again. Particle P sets its last terminating bit b_{h^*} to 1 when it next visits a node v with distance 0 to h^*. Therefore, P terminates at $v \in B(O) \cap H(O)$. □

The previous lemmas imply the following theorem.

Theorem 2.4. *The single-particle algorithm terminates after $t^* = \mathcal{O}(B)$ asynchronous rounds with particle P at a node $u \in B(O) \cap H(O)$ and $H_{t^*}(O) = H(O)$.*

3 A BINARY COUNTER OF PARTICLES

For a system of particles each with constant-size memory to emulate the single-particle algorithm of Section 2, the particles need a mechanism to distributively store the distances to each of the strong $\mathcal{O}_\Delta$-hull's six half-planes. To that end, we now describe a local, distributed algorithm for coordinating a particle system as a binary counter that supports increments and decrements by one as well as zero-testing, subsuming previous work on an increment-only binary counter under the amoebot model [21]. This algorithm uses *tokens*, or constant-size messages passed between particles [8].

Suppose that the participating particles are organized as a simple path with the leader particle at its start: $\ell = P_0, P_1, P_2, \ldots, P_k$. Each particle P_i stores a value P_i.bit $\in \{\emptyset, 0, 1\}$, where P_i.bit $= \emptyset$ implies P_i is not part of the counter; i.e., it is beyond the most significant bit. Each particle P_i also stores tokens in a queue P_i.tokens; the leader ℓ can only store one token, while all other particles can store up to two. These tokens can be increments c^+, decrements c^-, or the unique *final token* f that represents the end of the counter. If a particle P_i (for $0 < i \leq k$) holds f — i.e., P_i.tokens contains f — then the counter value is represented by the bits of each particle from the leader ℓ (storing the least significant bit) up to and including P_{i-1} (storing the most significant bit).

The leader ℓ is responsible for initiating counter operations, while the rest of the particles use only local information and communication to carry these operations out. To increment the counter, the leader ℓ generates an increment token c^+ (assuming it was not already holding a token). Now consider this operation from the perspective of any particle P_i holding a c^+ token, where $0 \leq i \leq k$. If P_i.bit $= 0$, P_i consumes c^+ and sets P_i.bit $\leftarrow 1$. Otherwise, if P_i.bit $= 1$, this increment needs to be carried over to the next most significant bit. As long as P_{i+1}.tokens is not full (i.e., P_{i+1} holds at most one token), P_i passes c^+ to P_{i+1} and sets P_i.bit $\leftarrow 0$. Finally, if P_i.bit $= \emptyset$, this increment has been carried over past the counter's end, so P_i must also be holding the final token f. In this case, P_i forwards f to P_{i+1}, consumes c^+, and sets P_i.bit $\leftarrow 1$.

To decrement the counter, the leader ℓ generates a decrement token c^- (if it was not holding a token). From the perspective of any particle P_i holding a c^- token, where $0 \leq i < k$, the cases for P_i.bit $\in \{0, 1\}$ are nearly anti-symmetric to those for the increment. If P_i.bit $= 0$ and P_{i+1}.tokens is not full, P_i carries this decrement over by passing c^- to P_{i+1} and setting P_i.bit $\leftarrow 1$. However, if P_i.bit $= 1$, we only allow P_i to consume c^- and set P_i.bit $\leftarrow 0$ if P_{i+1}.bit $\neq 1$ or P_{i+1} is not only holding a c^-. While not necessary for the correctness of the decrement operation, this will enable conclusive zero-testing. Additionally, if P_{i+1} is holding f, then P_i is the most significant bit. So this decrement shrinks the counter by one bit; thus, as long as $P_i \neq P_0$, P_i additionally takes f from P_{i+1}, consumes c^-, and sets P_i.bit $\leftarrow \emptyset$.

Finally, the zero-test operation: if P_1.bit $= 1$ and P_1 only holds a decrement token c^-, ℓ cannot perform the zero-test conclusively (i.e., zero-testing is "unavailable"). Otherwise, the counter value is 0 if and only if P_1 is only holding the final token f and (i) ℓ.bit $= 0$ and ℓ.tokens is empty or (ii) ℓ.bit $= 1$ and ℓ is only holding a decrement token c^-.

Analysis. Due to space constraints, we only summarize our rigorous analysis of the distributed binary counters; full proofs and accompanying pseudocode can be found in the arXiv version [6]. Correctness of the increment and decrement operations follows from the fact that the increment and decrement tokens remain in the order they were created, so order of operations is preserved. We then prove the correctness of zero-testing in two parts: first, we show that zero-testing is always eventually available; then we show that if the zero-test operation is available, it is always accurate.

To analyze the counter's runtime, we employ a *dominance argument* between asynchronous and parallel executions, building upon the analysis of [21] that bounded the running time of an increment-only distributed counter. The general idea of the argument is as follows. First, we prove that the counter operations are, in the worst case, at least as fast in an asynchronous execution as they are in a simplified parallel execution. We then give an upper bound on the number of parallel rounds required to process these operations; combining these two results also gives a worst case upper bound on the running time in terms of asynchronous rounds. This analysis culminates in the following result.

Theorem 3.1. *Given any nonnegative sequence of m operations and any fair asynchronous activation sequence, the distributed binary counter processes all operations in $O(m)$ asynchronous rounds.*

4 THE CONVEX HULL ALGORITHM

We now show how a system $\mathcal{P}$ of particles each with only constant-size memory can emulate the single-particle algorithm of Section 2. Recall that we assume $\mathcal{P}$ contains a unique leader particle ℓ initially adjacent to the object. This leader ℓ is primarily responsible for emulating the particle with unbounded memory in the single-particle algorithm. To do so, it organizes the other particles in the system as distributed memory, updating its distances d_h to half-plane h as it moves along the object's boundary. This is our algorithm's *learning phase.* In the *formation phase*, ℓ uses these complete measurements to lead the other particles in forming the convex hull. There is no synchronization among the various (sub)phases of our algorithm; for example, some particles may still be finishing the learning phase after the leader has begun the formation phase. Pseudocode for the entire algorithm can be found in the arXiv version [6].

4.1 Learning the Convex Hull

The *learning phase* combines the movement rules of the single-particle algorithm (Section 2) with the distributed binary counters (Section 3) to enable the leader to measure the convex hull $H(O)$. There are some nuances in adapting the general-purpose counters for use in our convex hull formation algorithm. These are omitted due to space constraints but can be found in the arXiv version [6].

In the learning phase, each particle P can be in one of three states, denoted P.state: *leader*, *follower*, or *idle.* All non-leader particles are assumed to be initially idle and contracted. To coordinate the system's movement, the leader ℓ orients the particle system as a spanning tree rooted at itself using the *spanning tree primitive* (see, e.g., [8]). If an idle particle P is activated and has a non-idle neighbor, then P becomes a follower and sets P.parent to this neighbor. This primitive continues until all idle particles become followers.

Imitating the single-particle algorithm of Section 2, ℓ performs a clockwise traversal of the boundary of the object O using the right-hand rule, updating its six distance counters d_h along the way. It terminates once it has visited all six half-planes without pushing any of them, which it detects using its terminating bits b_h. In this multi-particle setting, we need to carefully consider both how ℓ updates its counters and interacts with its followers as it moves.

Rules for Leader Computation and Movement. If ℓ is expanded and it has a contracted follower child P in the spanning tree that is keeping counter bits, ℓ pulls P in a handover. Otherwise, suppose ℓ is contracted. If all its terminating bits b_h are equal to 1, then ℓ has learned the convex hull, completing this phase. Otherwise, it must continue its traversal of the object's boundary. If the zero-test operation is unavailable or if it is holding increment/decrement tokens for any of its d_h counters, it will not be able to move. Otherwise, let $i \in [6]$ be its next move direction according to the right-hand rule, and let v be the node in direction i. There are two cases: either v is unoccupied, or ℓ is blocked by another particle occupying v.

In the case ℓ is blocked by a contracted particle P, ℓ can *role-swap* with P, exchanging its memory with the memory of P. In particular, ℓ gives P its counter bits, its counter tokens, and its terminating bits; promotes P to become the new leader by setting P.state $\leftarrow$ *leader* and clearing P.parent; and demotes itself by setting ℓ.state $\leftarrow$ *follower* and ℓ.parent $\leftarrow P$. This effectively advances the leader's position one node further along the object's boundary.

If either v is unoccupied or ℓ can perform a role-swap with the particle blocking it, ℓ first calculates whether the resulting move would push one or more half-planes using update vector δ_i. Let $\mathcal{H}' = \{h \in \mathcal{H} : \delta_{i,h} = -1 \text{ and } d_h = 0\}$ be the set of half-planes being pushed, and note that ℓ can locally check if $d_h = 0$ since zero-testing is currently available. It then generates the appropriate increment and decrement tokens according to δ_i. Next, it updates its terminating bits: if it is about to push a half-plane (i.e., $\mathcal{H}' \neq \emptyset$), then it sets $b_h \leftarrow 0$ for all $h \in \mathcal{H}$; otherwise, it can again use zero-testing to set $b_h \leftarrow 1$ for all $h \in \mathcal{H}$ such that $d_h + \delta_{i,h} = 0$. Finally, ℓ performs its move: if v is unoccupied, ℓ expands into v; otherwise, ℓ role-swaps with the particle blocking it.

Rules for Follower Movement. Consider any follower P. If P is expanded and has no children in the spanning tree nor any idle neighbor, it simply contracts. If P is contracted and following the tail of its expanded parent $Q = P$.parent, P can push Q in a handover. Similarly, if Q is expanded and has a contracted child P, Q can pull P in a handover. However, if P is not keeping counter bits but Q is, then a handover between P and Q could disconnect the counters (see Figure 5). So we only allow these handovers if either (i) both keep counter bits, (ii) neither keep counter bits, or (iii) one does not keep counter bits while the other holds the final token.

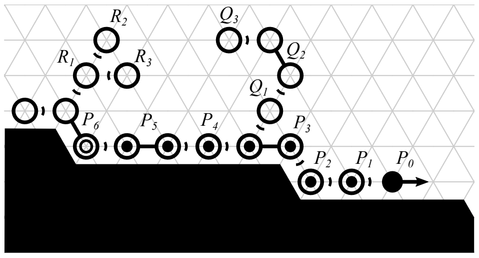


Figure 5: The leader P_0 (black dot) and its followers (black circles). Followers with dots keep counter bits, and P_6 holds the final token. Allowing Q_1 to handover with P_3 would disconnect the counter; all other potential handovers are safe.

4.2 Forming the Convex Hull

The *formation phase* brings as many particles as possible into the convex hull $H(O)$. It is divided into two subphases. In the *hull closing* subphase, the leader particle ℓ uses its binary counters to lead the rest of system $\mathcal{P}$ along a clockwise traversal of $H(O)$. If ℓ completes its traversal, leaving every node of the convex hull occupied by (possibly expanded) particles, the *hull filling* subphase fills the convex hull with as many contracted particles as possible.

4.2.1 The Hull Closing Subphase. When the learning phase ends, ℓ occupies some position $s \in H(O)$ (by Lemma 2.3) and its distributed binary counters d_h contain accurate distances to each of the six half-planes $h \in \mathcal{H}$. The leader's main role during the hull closing subphase is to lead the rest of the particle system along a clockwise traversal of $H(O)$. In particular, ℓ uses its counters to detect when it reaches one of the six vertices of $H(O)$, at which point it turns 60° clockwise to follow the next half-plane, and so on.

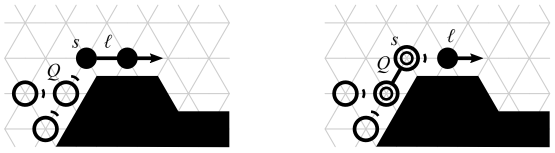
The particle system tracks the position s that ℓ started from by ensuring a unique *marker* particle occupies it. The marker can only contract out of s as part of a handover, at which point the marker role is transferred so that the marker always occupies s. Thus, when ℓ encounters the marker particle occupying the next node of the convex hull, it can locally determine that it has completed its traversal and this subphase.

However, there may not be enough particles to close the hull. Recall that $H = |H(O)|$ is the number of nodes in the convex hull. If $|\mathcal{P}| < \lceil H/2 \rceil$, eventually all particles enter $H(O)$ and follow ℓ as far as possible without disconnecting from the marker particle, which cannot leave position s. With every hull particle expanded and unable to move any farther, a token passing scheme is used to inform ℓ that there are insufficient particles for closing the hull and advancing to the next subphase. Upon receiving this message, ℓ terminates, with the rest of the particles following suit.

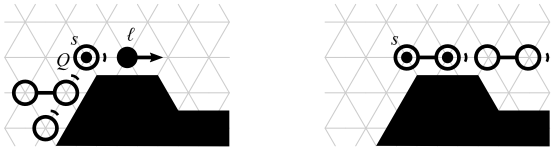
Rules for Leader Computation and Movement. If the leader ℓ is holding the "all expanded" token and does not have the marker particle in its neighborhood — indicating that there are insufficient particles to complete this subphase — it generates a "termination" token and passes it to its child in the spanning tree. It then terminates by setting ℓ.state $\leftarrow$ *finished.*

Otherwise, if ℓ is expanded, there are two cases. If ℓ has a contracted hull child Q (i.e., a child Q with Q.state = *hull*), ℓ performs a pull handover with Q. If ℓ does not have any hull children but does have a contracted follower child Q keeping counter bits, then this is its first expansion of the hull closing subphase and the marker should occupy its current tail position. So ℓ sets Q.state $\leftarrow$ *pre-marker* and performs a pull handover with Q (see Figure 6a–6b).

During its traversal of $H(O)$, ℓ keeps a variable ℓ.plane $\in \mathcal{H}$ indicating which half-plane boundary it is currently following. The leader updates ℓ.plane when it discovers (via zero-testing) that it has reached the next half-plane. It then inspects the next node of its traversal along ℓ.plane, say v. If v is occupied by the marker particle Q, then ℓ has completed the hull closing subphase; it updates Q.state $\leftarrow$ *finished* and then advances to the hull filling subphase (Section 4.2.2). Otherwise, if ℓ is contracted, it continues its traversal of the convex hull by either expanding into node v if v is unoccupied or by role-swapping with the contracted particle blocking it, just as it did in the learning phase.



(a) (b)



(c) (d)

Figure 6: (a) After expanding for the first time, the leader ℓ occupies the starting position s with its tail. (b) After performing a handover with ℓ, follower child Q becomes the pre-marker (inner circles). (c) When Q contracts, it becomes the marker (inner dot). (d) If there are insufficient particles to close the hull, the marker will eventually become expanded and unable to contract without vacating position s.

Rules for the Marker Particle Logic. The marker role must be passed between particles so that the marker particle always occupies the position at which the leader started its hull traversal. Whenever a contracted marker particle P expands in a handover with its parent, it remains a marker particle. When P subsequently contracts as a part of a handover with a contracted child Q, P sets P.state $\leftarrow$ *hull* and Q.state $\leftarrow$ *pre-marker.* Finally, when the pre-marker Q contracts — either on its own or as part of a handover with a contracted child — Q becomes the marker particle (see Figure 6c).

Importantly, the marker particle P never contracts outside of a handover, as this would vacate the leader's starting position (see Figure 6d). If P is ever expanded but has no children or idle neighbors, it generates the "all expanded" token and passes toward ℓ along expanded particles only. If this ultimately causes ℓ to generate and pass a "termination" token back to P, P consumes the termination token and becomes finished.

Rules for Follower and Hull Particle Behavior. Follower particles behave just as they did in the learning phase, with two modifications. First, if ever a follower performs a handover with the (pre-)marker particle, their states are updated as described above. Second, follower particles never perform handovers with hull particles.

Hull particles are simply follower particles that have joined the convex hull. They only perform handovers with the leader and other hull particles. Additionally, they pass the "all expanded" and "termination" tokens: if an expanded hull particle P holds the "all expanded" token and P.parent is also expanded, P passes this token to P.parent. If a hull particle P is holding the "termination" token, it terminates by passing this token to its hull or marker child and becoming finished.

4.2.2 *The Hull Filling Subphase.*

The hull filling subphase is the final phase of the algorithm. It begins when the leader ℓ encounters the marker particle in the hull closing subphase, completing its hull traversal. At this point, $H(O)$ is entirely filled with particles, though some may be expanded. The remaining followers are either outside the hull or are trapped between the hull and the object. The goal of this subphase is to allow trapped particles to escape outside the hull, and to use the followers outside the hull to "fill in" beside any expanded hull particles, filling the hull with as many contracted particles as possible.

At a high level, this subphase works as follows. The leader ℓ first becomes finished. Each hull particle also becomes finished when its parent is finished. A finished particle P labels a neighboring follower Q as either *trapped* or *filler* depending on whether Q is inside or outside the hull, respectively. This can be determined locally using the relative position of Q to the parent of P. A trapped particle performs a coordinated series of movements with a neighboring finished particle to effectively take its place, "pushing" the finished particle outside the hull as a filler particle. Filler particles perform a clockwise traversal over the finished particles on the hull, searching for an expanded finished particle to handover with, effectively replacing it with two contracted ones.

There are two ways the hull filling subphase can terminate. If $\lceil H/2 \rceil \leq |\mathcal{P}| < H$, there are enough particles to close the hull but not enough to fill it with contracted particles. In this case, all particles join the hull and become finished by following the rules above. If instead $|\mathcal{P}| \geq H$, the entire hull can be filled with contracted particles. This event is detected using a token passing scheme that, on completion, triggers a broadcast of termination tokens that cause all particles (including the extra ones outside the hull) to finish. In the following, we describe the local rules underlying the three important primitives for this subphase.

Freeing Trapped Particles. Suppose a finished particle P has labeled a neighboring contracted particle Q as trapped (see Figure 7a). In doing so, P updates Q.parent $\leftarrow P$. When Q is next activated, it sets P.state $\leftarrow$ *pre-filler* (see Figure 7b). This indicates to P that it should expand towards the outside of the hull as soon as possible (Figure 7c). Once P has expanded, P and Q perform a handover (Figure 7d). This effectively pushes P out of the hull, where it becomes a filler particle, and expands Q into the hull, where it becomes pre-finished. Finally, whenever Q contracts — either on its own or in a handover — it becomes finished, taking the original position and role of P (Figure 7e).

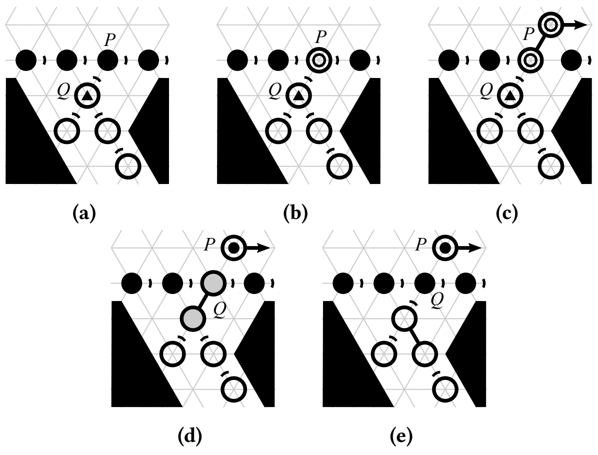


Figure 7: Freeing a trapped particle. (a) A finished particle P marks a neighboring follower Q on the interior of the hull as trapped (inner triangle). (b) Q marks its parent P as a pre-filler (inner circle). (c) P expands outside the hull. (d) In a handover between P and Q, P becomes a filler (inner dot) and Q becomes pre-finished (gray). (e) Q contracts and finishes.

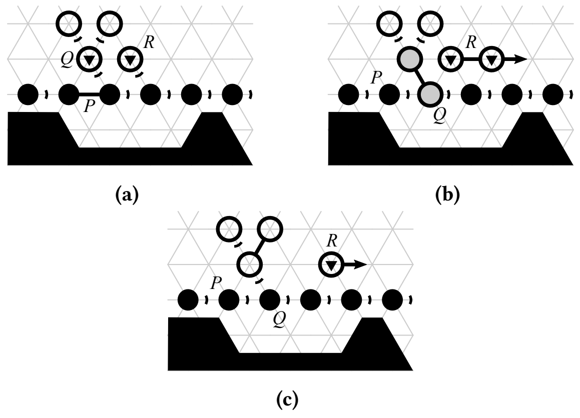


Figure 8: Some movements of filler particles. (a) A finished particle P marks neighboring followers Q and R on the exterior of the hull as fillers (inner triangle). (b) Q performs a handover with P to fill the hull, becoming pre-finished (gray), while R expands along a clockwise traversal of the hull. (c) Q contracts and becomes finished.

Filling the Hull. A particle P becomes a filler either by being labeled so by a neighboring finished particle or by being ejected from the hull while freeing a trapped particle, as described above. If P is expanded, it simply contracts if it has no children or idle neighbors, or performs a pull handover with a contracted follower child if it has one. If P is contracted, it finds the next node v on its clockwise traversal over the finished particles. P simply expands into v unless the first occupied node clockwise from v is occupied by the tail of an expanded finished particle Q. In this case, P performs a push handover with Q, sets Q to be its parent, and becomes pre-finished. Whenever P next contracts, either on its own or during a handover, it becomes finished. An example of some filler particle movements is depicted in Figure 8.

Detecting Termination. Before ℓ becomes finished at the start of this subphase, it generates an "all contracted" token containing a counter t initially set to 0. This token is passed backwards along the hull to contracted finished particles only. Whenever the token is passed through a vertex of the convex hull, the counter t is incremented. Thus, if a contracted finished particle is ever holding the "all contracted" token and its counter t is equal to 7, it terminates by consuming the "all contracted" token and broadcasting "termination" tokens to all its neighbors. Whenever a particle receives a termination token, it also terminates by becoming finished.

4.3 Analysis

Due to space constraints, we only give the highlights of our analysis here; the complete arguments can be found in the arXiv version [6]. Our correctness arguments begin with proving several safety properties of our distributed binary counters despite the adaptations required for convex hull formation. In particular, we use the assumption from Section 1.2 that our system contains $|\mathcal{P}| > \log_2 H$ particles in conjunction with the leader's traversal path to prove two useful results regarding the lengths of the counters.

Lemma 4.1. *Let L be the path of nodes traversed by leader ℓ from the start of the algorithm to its current position. Then there are at most $\lfloor \log_2 \min\{|L|, H\} \rfloor + 1$ particles holding bits of a distributed binary counter d_h.*

Lemma 4.2. *Let L be the path of nodes traversed by leader ℓ from the start of the algorithm to its current position. Then there are at least $\min\{|\mathcal{P}|, \lceil |L|/2 \rceil\}$ particles including ℓ along L.*

Using these lemmas, we can immediately conclude that the distributed binary counters never disconnect or intersect themselves and that there are always enough particles to maintain the counters. We then show the counters never impede the leader's ability to move forward indefinitely.

Lemma 4.3. *If ℓ only has one bit of a distributed binary counter d_h and is not holding the final token f_h at time t, thereby being prohibited from moving, then there exists a time $t' > t$ when ℓ either has two bits of d_h or is holding f_h.*

We then prove a series of correctness results analyzing the various (sub)phases of the algorithm. For the learning phase, it suffices to show that the leader ℓ traverses the boundary of object O and obtains an accurate measurement of the convex hull. Since we already argued about the correctness of the counters, it suffices to show the following.

Lemma 4.4. *If ℓ is contracted, it can always eventually expand or role-swap along its clockwise traversal of $B(O)$. If ℓ is expanded, it can always eventually perform a handover with a follower.*

For the hull formation phase, we analyze what level of hull completion is achieved based on the number of particles $|\mathcal{P}|$. We first show that the leader successfully closes the hull if and only if there are at least $\lceil H/2 \rceil$ particles in the system; otherwise, a maximal partial convex hull is formed. We then show that as long as the hull closing subphase completes, the hull will be filled with as many contracted particles as possible; however, if there are less than H particles in the system, not all nodes of $H(O)$ will be occupied by contracted particles. Synthesizing these lemmas yields the following correctness theorem.

Theorem 4.5. *The Convex Hull Algorithm correctly solves instance $(\mathcal{P}, O)$ of the convex hull formation problem if $|\mathcal{P}| \geq |H(O)|$, and otherwise forms a maximal partial strong O_Δ-hull of O.*

We then bound the worst-case number of asynchronous rounds for the leader ℓ to learn and form the convex hull. As in Section 3, we use dominance arguments to show that the worst-case number of parallel rounds required by a carefully defined parallel schedule is no less than the runtime of our algorithm. The first dominance argument shows that the counters bits are forwarded quickly enough to avoid blocking leader expansions and role-swaps. This argument culminates in the following lemma.

Lemma 4.6. *Suppose leader ℓ only has one bit of a counter d_h and is not holding the final token f_h in round $0 \leq t \leq T - 2$ of greedy parallel bit forwarding schedule $(C_0, \ldots, C_T)$. Then within the next two parallel rounds, ℓ will either have a second bit of d_h or will be holding f_h, allowing it to role-swap.*

The second dominance argument relates the time required for ℓ to traverse the object's boundary and convex hull to the running time of our algorithm. Both build upon previous work [5], which analyzed spanning trees of particles led by their root particles. Several nontrivial extensions are needed here to address the interactions between the counters and particle movements as well as traversal paths that can be temporarily blocked. The key lemma here is the following, which relates the time required for ℓ to traverse a path to the path's length.

Lemma 4.7. *If L is the (not necessarily simple) path of the leader's traversal, the leader traverses this path in $O(|L|)$ asynchronous rounds in the worst case.*

Using Lemma 4.7, we can directly relate the distance the leader ℓ has traversed to the system's progress towards learning and forming the convex hull. By Lemma 5 of [5], B particles self-organize as a spanning tree rooted at ℓ in at most $O(B)$ asynchronous rounds. By Lemmas 2.1 and 2.3, ℓ traverses $B(O)$ at most twice before completing the learning phase. Thus, by Lemma 4.7:

Lemma 4.8. *The learning phase completes in at most $O(B)$ asynchronous rounds.*

The hull closing and filling subphases can be analyzed similarly (with additional technicalities), ultimately yielding the following.

Lemma 4.9. *In at most $O(H)$ asynchronous rounds from when the leader ℓ completes the learning phase, either ℓ completes its traversal of $H(O)$ and closes the hull or every particle in the system terminates, expanded over two nodes of $H(O)$.*

Lemma 4.10. *In at most $O(H)$ asynchronous rounds from when the leader ℓ closes the hull, $\min\{|\mathcal{P}|, H\}$ nodes of $H(O)$ will be filled with contracted finished particles.*

Putting it all together, the algorithm is correct by Theorem 4.5, the learning phase terminates in $O(B)$ asynchronous rounds by Lemma 4.8, the hull closing subphase terminates in an additional

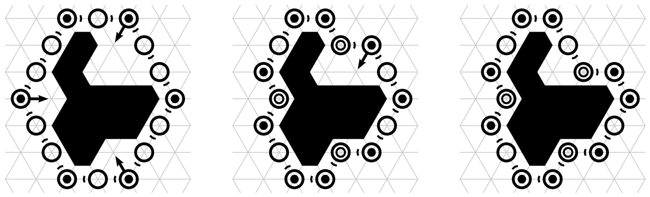


Figure 9: Transforming the cycle of tightening particles that initially form $H(O)$ into $H'(O)$ by moving convex particles (black dots) towards the object.

$O(H)$ asynchronous rounds by Lemma 4.9, and the hull filling subphase fills the convex hull with as many contracted particles as possible in another $O(H)$ asynchronous rounds by Lemma 4.10. Thus, since $B \geq H$, we have:

Theorem 4.11. *In at most $O(B)$ asynchronous rounds, the Convex Hull Algorithm either solves instance $(\mathcal{P}, O)$ of the convex hull formation problem if $|\mathcal{P}| \geq |H(O)|$ or forms a maximal partial strong O_Δ-hull of O otherwise.*

The time required for all particles in the system to terminate may be longer than the bound given in Theorem 4.11, depending on the number of particles. As termination is further broadcast to the rest of the system, we know that at least one non-finished particle receives a termination signal and becomes finished in each asynchronous round. So,

Corollary 4.12. *All particles in system $\mathcal{P}$ terminate the Convex Hull Algorithm in $O(|\mathcal{P}|)$ asynchronous rounds in the worst case.*

5 FORMING THE (WEAK) O_Δ-HULL

To conclude, we informally describe how the Convex Hull Algorithm can be extended to form the (weak) O_Δ-hull of object O, solving the O_Δ-hull formation problem. We refer the interested reader to the arXiv version [6] for the rigorous algorithm details and analysis. This *O_Δ-Hull Algorithm* extends the Convex Hull Algorithm at the point when a finished particle first receives the "all contracted" token with counter value 7, which usually triggers termination. Instead of terminating, this particle organizes the other contracted finished particles on $H(O)$ into a directed cycle of *tightening* particles. A tightening particle is *convex* if there is exactly one node in its neighborhood between its parent and child in the directed cycle, sweeping clockwise. The main idea of this algorithm is to repeatedly move convex tightening particles towards the object, progressively transforming the directed cycle into the object's O_Δ-hull (Figure 9). By arguments similar to those for the Convex Hull Algorithm, we have the following theorem.

Theorem 5.1. *In at most $O(H)$ asynchronous rounds, the O_Δ-Hull Algorithm solves instance $(\mathcal{P}, O)$ of the O_Δ-hull formation problem if $|\mathcal{P}| \geq H$. All particles terminate in an additional $O(|\mathcal{P}|)$ rounds.*

ACKNOWLEDGMENTS

Daymude and Richa are supported in part by the National Science Foundation under awards CCF-1422603, CCF-1637393, and CCF-1733680. Hinnenthal and Scheideler are supported by the German Research Foundation (DFG) under Project SCHE 1592/6-1.

REFERENCES

[1] Alan Aderem and David M. Underhill. 1999. Mechanisms of phagocytosis in macrophages. *Annual Review of Immunology* 17, 1 (1999), 593–623.

[2] Selim G. Akl and Kelly A. Lyons. 1993. *Parallel Computational Geometry*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[3] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. 2006. Computation in networks of passively mobile finite-state sensors. *Distributed Computing* 18, 4 (2006), 235–253.

[4] Philip Bernstein, Vassos Hadzilacos, and Nathan Goodman. 1987. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.

[5] Joshua J. Daymude, Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. 2018. On the Runtime of Universal Coating for Programmable Matter. *Natural Computing* 17, 1 (2018), 81–96.

[6] Joshua J. Daymude, Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Christian Scheideler, and Andréa W. Richa. 2019. Convex Hull Formation for Programmable Matter. (2019). Available online at https://arxiv.org/abs/1805.06149.

[7] Joshua J. Daymude, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. 2017. Improved Leader Election for Self-Organizing Programmable Matter. In *Algorithms for Sensor Systems (ALGOSENSORS '17)*. Springer, Cham, 127–140.

[8] Joshua J. Daymude, Kristian Hinnenthal, Andréa W. Richa, and Christian Scheideler. 2019. Computing by Programmable Particles. In *Distributed Computing by Mobile Entities*. Springer, Cham, 615–681.

[9] Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. 2016. Universal Shape Formation for Programmable Matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '16)*. ACM, New York, NY, USA, 289–299.

[10] Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. 2017. Universal Coating for Programmable Matter. *Theoretical Computer Science* 671 (2017), 56–68.

[11] Mohamadou Diallo, Afonso Ferreira, Andrew Rau-Chaplin, and Stéphane Ubéda. 1999. Scalable 2D Convex Hull and Triangulation Algorithms for Coarse Grained Multicomputers. *J. Parallel and Distrib. Comput.* 56, 1 (1999), 47–70.

[12] Patrick Dymond, Jieliang Zhou, and Xiaotie Deng. 2001. A 2-D parallel convex hull algorithm with optimal communication phases. *Parallel Comput.* 27, 3 (2001), 243–255.

[13] Eugene Fink and Derick Wood. 2004. *Restricted-Orientation Convexity*. Springer-Verlag Berlin Heidelberg, Berlin, Germany.

[14] Per-Olof Fjällström, Jyrki Katajainen, Christos Levcopoulos, and Ola Petersson. 1990. A sublogarithmic convex hull algorithm. *BIT Numerical Mathematics* 30, 3 (1990), 378–384.

[15] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro (Eds.). 2019. *Distributed Computing by Mobile Entities*. Springer International Publishing, Switzerland.

[16] Rolf G. Karlsson and Mark H. Overmars. 1988. Scanline algorithms on a grid. *BIT Numerical Mathematics* 28, 2 (1988), 227–241.

[17] C. Ronald Kube and Eric Bonabeau. 2000. Cooperative transport by ants and robots. *Robotics and Autonomous Systems* 30, 1 (2000), 85–101.

[18] Helen F. McCreery and Michael D. Breed. 2014. Cooperative transport in ants: a review of proximate mechanisms. *Insectes Sociaux* 61, 2 (2014), 99–110.

[19] Russ Miller and Quentin F. Stout. 1988. Efficient parallel convex hull algorithms. *IEEE Trans. Comput.* 37, 12 (1988), 1605–1618.

[20] Matthew J. Patitz. 2014. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing* 13, 2 (2014), 195–224.

[21] Alexandra Porter and Andréa W. Richa. 2018. Collaborative Computation in Self-organizing Particle Systems. In *Unconventional Computation and Natural Computation (UCNC '18)*. Springer, Cham, 188–203.

[22] Sergio Rajsbaum and Jorge Urrutia. 2011. Some problems in distributed computational geometry. *Theoretical Computer Science* 412, 41 (2011), 5760–5770.

[23] Gregory J. E. Rawlins. 1987. *Explorations in Restricted Orientation Geometry*. Ph.D. Dissertation. University of Waterloo, Ontario.

[24] William Savoie, Sarah Cannon, Joshua J. Daymude, Ross Warkentin, Shengkai Li, Andréa W. Richa, Dana Randall, and Daniel I. Goldman. 2018. Phototactic Supersmarticles. *Artificial Life and Robotics* 23, 4 (2018), 459–468.

[25] Tommaso Toffoli and Norman Margolus. 1991. Programmable matter: Concepts and realization. *Physica D: Nonlinear Phenomena* 47, 1 (1991), 263–272.

[26] Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. 2013. Active Self-assembly of Algorithmic Shapes and Patterns in Polylogarithmic Time. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science (ITCS '13)*. ACM, New York, NY, USA, 353–354.

[27] Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S. Chirikjian. 2007. Modular Self-Reconfigurable Robot Systems. *IEEE Robotics Automation Magazine* 14, 1 (2007), 43–52.

[28] Guoxian Zhang, Gregory K. Fricke, and Devendra P. Garg. 2013. Spill Detection and Perimeter Surveillance via Distributed Swarming Agents. *IEEE/ASME Transactions on Mechatronics* 18, 1 (2013), 121–129.