Self-Timed Patterning

Micah Z. Brodsky
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA
Email: micahbro@csail.mit.edu

Abstract—**Timing and convergence constitute a serious and under-appreciated complication in developmental patterning. I demonstrate an algorithm design methodology that eliminates timing problems across a broad family of spatial patterning mechanisms, with discrete gradients and growing points as examples. Values are exchanged between modules using partially informative representations, which monotonically increase in precision until a definite answer is represented. As a result, the convergence of local outputs can be detected locally. Irreversible, disruptive operations such as cell division can then be orchestrated using inputs known to be stable and valid. I term this approach *self-timed patterning*.**

I. Introduction

In centralized systems, timing is often taken for granted. Clocks are global and consistent. Tasks block until they are completed. Combining tasks into sequences is straightforward, implicit in every imperative language.

In spatially distributed systems, however, time is slippery. Time is relative, due to communication delay [1]. Clocks are decentralized, conflicting, and often drifting. Inputs may be available in different places at different times. Tasks, distributed spatially, complete in some places sooner than in others. The speed of computation may not even be consistent.

In spatial computing, and in particular developmental patterning, the problem of time is ever-present. The complications due to timing are often sidestepped through a mixture of ad-hoc, optimistic assumptions about the timing properties of the substrate or by pursuing problems that naturally lend themselves to asymptotic convergence rather than particular results at definite times.

For example, the Growing Points Language [2] assumes that spatial gradients (e.g. Bellman-Ford distances) may be trusted once they are locally detectable. This is potentially race-prone if multiple sources appearing in different places must be considered, because their influences may arrive at different times, but even in the common case of a single advancing source, it assumes that spatial distance and propagation time are comparable, or that propagation time is completely negligible. Temporal irregularities such as a slow patch of cells will lead to transient errors in magnitude and direction as the gradient navigates around the irregularity. Even though the gradient eventually corrects itself, because it is hard for a growing point to correct a path already traversed, such transients may lead to permanent irregularities.

The Origami Shape Language [3], on the other hand, has many features that are naturally "feed-forward", in that transients will ripple through the calculations and eventually be replaced by the final, steady-state values, because the computation has no persistent memory in which to retain transient errors. In time, the outputs will be correct. However, the language does include some operations that require convergence to be achieved beforehand, such as the repurposing of reusable communication channels, and so at a certain point it must stop waiting and accept the answers. The amount of time to delay is calibrated empirically, based on propagation time measurements during start-up using test gradients. The wait is configured to be an overestimate based on the longest possible propagation distance. So long as timing is uniform and consistent, this is reasonable, albeit wasteful. However, it will fare poorly if timing properties vary unpredictably or if the domain size can enlarge.

When such ad-hoc assumptions are unjustified, and when the structure of the computation does not naturally forget transient perturbations, the problem of timing can no longer be ignored. In particular, when irreversible or difficult to reverse actions must be taken on the basis of the results of spatial computations, it is imperative that the results be accurate and stable beforehand. This sort of difficulty arises particularly often in developmental patterning, where disruptive and destructive primitives such as cell division, death, and topological rearrangement must be choreographed along complex, dynamically constructed spatial patterns. Moreover, development time is often precious, and so hard-coding adequately conservative convergence delays may be completely unreasonable. This problem has been acknowledged only rarely (e.g. [4]) and investigated even less.

To illustrate the magnitude of the problem with a natural example, consider the early development of the chick embryo. At the beginning of incubation, it consists of a mass of largely undifferentiated cells. Within about 24 hours, it has finished gastrulation, begun its neural folds, and assembled its first somite and the beginnings of a notochord [5]. Yet, given the embryo's size on the order of a few millimeters, the diffusion time to establish a typical morphogen field is expected to be on the order of an hour, and several hours more to reach a close approximation of steady state [6, ch. 3]. With multiple, sequential developmental steps crammed into such a tiny span, there's little time to waste. Furthermore, timing skew is acutely evident, with the head of the embryo maturing far in advance of the tail.

One can imagine two general approaches to the problem of timing in spatial patterning. One might design the distributed computations such that correct convergence can be detected by inspecting the output provided. This can be nontrivial, because the verification process must be significantly simpler and more temporally tractable than the original problem being solved or no progress has been made. Once correct values have been identified, they may be checkpointed and passed onto disruptive and irreversible actuation processes. This might be compared loosely with the manner in which common bilaterian animal body plans such as in drosophila develop, with successive cascades of gradients and compartment patterns preceding large-scale morphogenesis. Alternatively, one could design the overall algorithm such that even though individual steps are effectively irreversible, the large-scale dynamics follow a self-stabilizing trajectory, and so excursions due to acting on erroneous transient values will eventually die out like any other perturbation. This is perhaps reminiscent of the way such exemplary regenerating animals as the hydra are thought to maintain their body structure.

In this paper I demonstrate a simple methodology for designing spatial patterning algorithms whose local completion status is implicitly indicated by the output values themselves. Inspired by the self-timing methodology in clockless digital circuit design, I term this technique self-timed patterning. The key insight is the use of partially informative data representations, which monotonically increase in precision until the true answer is indicated. I show how they can be used to pattern disruptive, irreversible transformations both quickly and safely, completely robust to timing pathologies.

II. Self-Timing

When a system is decomposed into multiple, simultaneously executing modules, the modules must communicate in order to coordinate their behaviors, and their communication protocols must confront the problem of timing. A self-timed system, loosely speaking, is a system whose modules communicate through a type of asynchronous protocol such that no assumptions need to be made are made about the latencies of the modules themselves [7]. Self-timing naturally lends itself to race-free designs, and additional steps are often necessary to allow data races. The key insight is that that data signals must indicate, explicitly or implicitly, when they are ready for use, and once they have done so must remain fixed until acknowledged. General self-timed circuits must also treat the acknowledgement signals with similar care in order that circuit elements may be reused for multiple waves of data, but for our purposes, focusing on single-use developmental cascades, we can usually omit the acknowledgement pathways entirely.

The simplest approach to self-timed signaling is to bundle the data signals with a boolean completion signal that is asserted once the data outputs can be trusted. This is known as "bundled data" and it is used successfully, although care must be taken to avoid timing skew between the arrival of the data and completion signals. However, an alternative scheme, known as "dual rail", turns out to be more natural and insightful for our purposes. In classic dual rail signaling, two separate signals are provided for each bit, one indicating whether it is true and one indicating whether it is false. This scheme can articulate four possible status values per bit: true, false, unknown, and contradiction (generally unused). For multi-bit data values, the individual bits may become valid at different times, but once a bit becomes valid it will not become invalid (at least until the next acknowledgement cycle).

Many variations on dual-rail-style self-timed signaling are possible, but they share a common insight: glitch-free partial information about the answer accumulates monotonically over time, beginning with complete uncertainty and ending with an unambiguously specified value. This is the concept of monotonic partial information championed by Propagators [8], [9], and single-use self-timed systems can be represented most clearly as propagator networks.

When data values are not limited to raw digital signals but can include analog values or compound data structures, more complex partial information structures may be used, such as interval arithmetic. Modules need not wait until their inputs have reached complete convergence, either; they may compute partially informative answers based on partial input information. When an actuator determines its inputs are sufficiently precise, it may act on them even before final convergence. (If, however, such actuation may disrupt the upstream computations by triggering non-monotonic changes that may contradict the existing outputs, then the inputs must be checkpointed first and the computation disabled; this is discussed in Section IV-B.)

III. Simple Computations

The simplest sort of self-timed computations are those that depend only on local information, for example, taking local averages or detecting local maxima of a field. Such computations can simply wait for all their inputs to be available, locally and from immediate neighbors, and then produce an output. If a partial set of inputs or partially informative inputs are available, a partial output may also be computed early, if desired. For example, if the number of neighbors expressing a signal is to be counted, and two have responded with the signal, three without, while one response remains to be received, the output count can be expressed as the interval $[2, 3]$. When the final response is received, the output is narrowed to an exact answer. On the other hand, sometimes partial inputs are sufficient to produce a complete output. For example, the boolean OR of binary inputs is known to be true as soon as the first positive response is received.

Implicit in this formulation is that cells must be able to enumerate their set of neighbors or otherwise determine when a complete set of responses has been received. This is natural for cells in physical contact with one another but is a slight departure from the classical amorphous computing model [10].

A. Gradients

Most spatial patterning algorithms are not so simple, relying on long-range propagation of information from cell to cell. However, the same principle can be applied, since simple

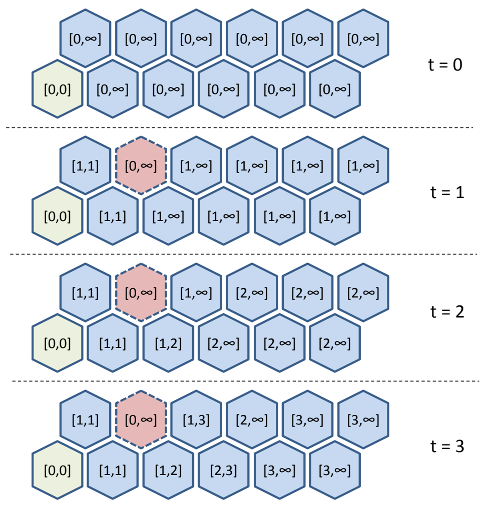

Fig. 1. Four iterations in the computation of a self-timed gradient, with source cell at the left (green) and one uncertain or stalled cell (red).

functions are the building blocks used to compute the outputs that are shared with cells' neighbors. To illustrate, let us construct a self-timed version of the Bellman-Ford hop count gradient algorithm, one of the most useful algorithms in discrete spatial patterning.

Ordinary Bellman-Ford works using a single communication channel, where each cell broadcasts its best known distance to a source, or zero if it itself is a source. Each cell iteratively replaces its best known distance with $\min_{n \in neighbors} distance(n) + 1$. Eventually, distances converge to steady-state values, but there is no local indicator of convergence.

To reformulate this as a self-timed computation, we must properly express what is known in the form of monotonic partial information. The best known distance is just that—an upper bound. If we represent distance as an interval, we can include a lower bound as well. With no information, a cell must report that its distance is in the interval $[0, \infty]$. At each local iteration, sources broadcast zero for their distance, while non-sources compute $\min_{n \in neighbors} distance(n) + [1, 1]$ and broadcast that. In interval arithmetic, the minimum value of a set of intervals is an interval spanning from the minimum over the lower bounds to the minimum over the upper bounds. As cells discover paths to sources, the interval upper bound always falls. As cells learn that successive rings of neighbors are not sources, the interval lower bound steadily rises. The interval thus brackets the actual distance between progressively narrowing bounds. When the two bounds meet, local convergence has been achieved. The time required relative to Bellman-Ford is essentially unchanged.

The same gradient algorithm works whether there is one source or many. If the input to the algorithm—the map of sources—is determined programmatically, it too must be represented as monotonic partial information. In this case, each cell may be source, a non-source, or undetermined. An undetermined cell always reports the lower bound of its distance as zero and hence will be surrounded by a "well" of cells with uncertain distance intervals, waiting to determine whether they're sitting right near a source (see Figure 1). Once the cell's source status is finally determined, the uncertainty disappears.

Cascading gradients and simple local computations, we have a fairly powerful patterning toolkit along the lines of [11], only now completely self-timed.

B. Growing Points

Another useful patterning tool is growing points [2]. Growing points are virtual excitations that move about a substrate according to programmable tropisms, tracing out paths. With self-timed gradients to direct their tropisms, we can construct self-timed growing points fairly easily, though some limitations will arise, affecting the tropisms that can be practically implemented.

The output of a growing points computation is a field of "secretions" left behind in the growing points' wake. For a complete self-timed formulation, we must know not only where the secretions are, but where they are not. Thus, we must be able to determine both the paths traversed by the growing points and the places that, at the conclusion of propagation, they will not have traversed. The presence of one or another determination indicates that, locally, the computation has completed.

The forward propagation of a growing point is straightforward. The cell hosting the growing tip waits until sufficient tropism-relevant information has been posted by all of its neighbors and then nominates the most favorable among them as a successor. Upon noticing the nomination, the designated successor changes its status from "unknown" to "in path" and performs its own nomination process. Secretions are then a local function computed on the path status.

In order to compute the complete secretion field, cells that will never be in the path must ultimately know to change their status from "unknown" to "not in path". The constraints used to generate this information necessarily relate to the global behavior of the growing point (and of all other indistinguishable growing points from the same family). For example, if the path has a definite termination condition and it is known that there will be only one trace of that family, then upon reaching the termination condition the growing point can broadcast a globally propagated completion message, signaling to all remaining "unknown" cells they are actually "not in path".

With more detailed information about the tropism, however, more prompt constraints can be used. For example, a lone, orthotropic growing point climbing a gradient can broadcast each level of the gradient passed; all "unknown" cells below that gradient level can conclude they are not in the path. More locally, any growing point family known not to self-intersect can back-propagate not-in-path status; any "unknown" cell whose preferred successor is in path must not be in path.

Additionally, any cell whose preferred successor is not in path must also be not in path. Together, these naturally cover many cases, with the exception of diatropism (propagation parallel to the contours of a gradient), where back-propagation must be bootstrapped e.g. by emitting perpendicular orthotropic growing points to turn off neighboring cells.

As a consequence of the fact that secretions are a function of partial information not complete until the growing point has exited the area, it is also generally not possible to use "auto-tropism", sensitivity to a gradient emitted by a trace's own secretions. Thus, other techniques must be used for purposes such as inertia and preventing diatropic traces from doubling back on themselves. Restricting successors to be non-neighbors of one's predecessor or favoring candidates farthest away from the predecessor are useful alternatives [12].

IV. Putting the Pieces Together

A. Composition

The composition of two self-timed computations is naturally another self-timed computation. In the simplest case, the downstream function merely waits for its inputs to arrive before generating an output; more generally, monotonic partial information propagates from one end to the other. As a result, local computations, gradients, growing points, and other self-timed computations can be cascaded seamlessly.

For example, a local computation based on the orientation of per-cell polarization vectors identifies which cells are on the left edge of the substrate and which are on the right (Figure 2a). Left edge cells are then used as the source for a left gradient; right edge cells are used as the source for a right gradient. The difference in value of the two gradients is used to divide the substrate into two compartments (2b). Within each compartment, similar computations can be nested recursively (2c). Note that this can converge even faster than the traditional feed-forward equivalent, because the gradient lower bounds allow compartment determination at the far ends well before a signal can arrive from the opposite end's source.

Since self-timed computations must know which neighbors to expect responses from, nested computations must be made aware of which cells belong to the same compartment and hence should be expected to participate. Like the input arguments, this information must be provided in a self-timed fashion.

B. Actuation and Checkpointing

Eventually, however, it comes time to act based upon the patterns established. Some operations, e.g. cell differentiation, are merely *irreversible*. These operations may be performed locally as soon as the necessary inputs are locally available.

Other operations, such as cell division and motility, are *disruptive*, causing non-monotonic changes to the properties and neighborhoods of cells. These must be treated with greater care, as they may cause the very inputs on which they depend to change, changing their own behavior and giving rise to race conditions in neighboring cells. To accommodate these in a self-timed computation, a checkpointing step must be employed.

Checkpointing is a process that serves to hand control from one stage of a patterning cascade to the next, where the stages would interfere if they ever overlapped in time. Checkpointing can be local to a cell or coordinated across a wider region, depending on the needs of the computation. Checkpointing entails halting the previous stage of computation (or at least preventing it from having further side effects), since subsequent results may no longer be trusted, and saving its outputs for future reference. This includes saving intermediate values shared with neighboring cells, because neighbors of a cell performing a checkpoint may not yet have completed the computation preceding the checkpoint and will need access to all their neighbors' broadcasts. Once prior computations have been halted and their outputs recorded, locally or over a sufficiently wide radius, the next stage of computation may be launched—e.g. a disruptive actuation process. Through checkpointing, disruptive actuation becomes a form of irreversible actuation.

For example, in Figure 2d, the cells in the third compartment are programmed to change their adhesive properties to reduce their affinity for the remaining cells and to undergo three rounds of oriented division. This leads to the formation of a crossbar. In the absence of checkpointing, the resulting rearrangement and increase in cell number leads to changes in gradient values and hence changes in the boundaries of the compartment, leading cells to reassess their fates. Some cells that have already divided will conclude that they do not belong in the crossbar after all, while other cells may be freshly recruited, dividing and changing their adhesive properties. The results of such churn in this example are shown in Figure 2e, where values were allowed to converge but checkpointing was disabled; the extent to which this matters varies, but here it yields a much sloppier result. Without an indication of a stage's completion such as provided by self-timing, checkpointing is impossible, on top of the hazard of actuation based on premature results.

When disruptive effects are merely local, checkpointing and actuation can be local as well. Neighboring cells may still be continuing to work on the prerequisite computations, relying on checkpointed copies of the cell's broadcasts. On the other hand, when the disruptive effects are non-local, for example, by exerting forces sufficient to cause neighboring cells to rearrange, a barrier computation may be needed to ensure that all cells within the radius of potential disruption have produced and checkpointed their outputs before the disruptive stage is allowed to begin. In the example above, a short-range barrier would probably be warranted (although in practice omitting it produces acceptable results). After actuation, barriers may be used again to verify completion of the disruptive steps, and then another wave of patterning can be initiated.

If only nearest neighbors are disrupted, the barrier simply waits for all neighboring cells to indicate completion. More generally, an arbitrarily ranged barrier can be constructed by the Bellman-Ford lower bound calculation: completed values

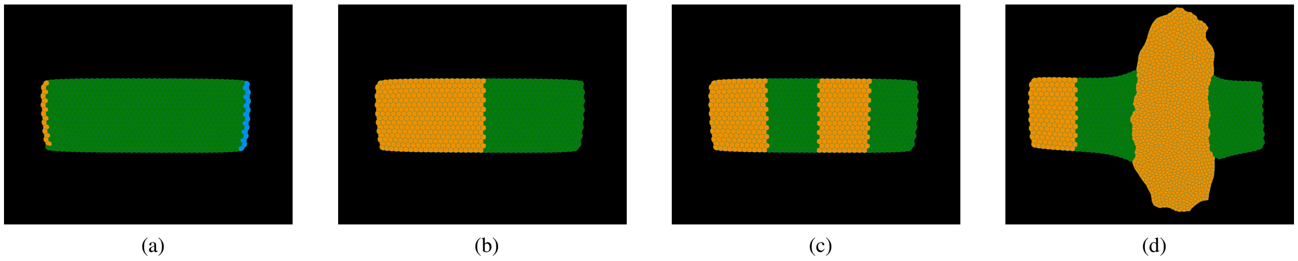
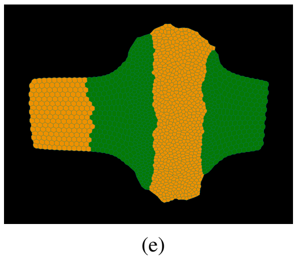

Fig. 2. 2a-2d: Steps in patterning a crossbar. 2e: Crossbar patterned without checkpointing inputs prior to cell division and adhesion changes. A custom, energy-minimization mechanical simulator modeling cells as soap bubbles is used to determine mechanical conformation and cell rearrangement [13].

are treated as non-sources, and incomplete values are treated as unknown. As nearby input values converge, the distance lower bound rises monotonically. When the distance to a source is known to be greater than the desired range, all cells within that range must have completed. The time required for such a barrier computation is proportional to its range.

When the effects of actuation are predictable and consistent, one can reasonably "prepattern" and checkpoint a map of actuations to be performed, then perform them—patterning by dead reckoning. One can expect to achieve a particular final configuration, albeit with some difficulty in specification given that the actuation map itself may be distorted by actuation. If, however, instability, noise, variation in cell layout, and other uncertainties lead to the actuation being not only disruptive but also *unpredictable*, such naive prepatterning can make only small changes before accumulated errors make a mess of both the structure and the prepattern. This sort of difficulty often arises, for example, when cells rearrange under stress or after division. Note how in Figure 2d the border of the crossbar is quite irregular.

In the presence of such unpredictability, a single round of patterning followed by actuation may be insufficient. Feedback measurements must then be used to correct the system towards its goal. A second set of barriers is necessary to verify the completion of the actuation, and then a round of re-measurement and corrective patterning can begin, driving a round of corrective actuation. This process may be repeated several times, if necessary. Unlike traditional closed-loop feedback, where error measurement and actuation take place simultaneously, self-timed feedback control must cycle through discrete stages (potentially in the form of traveling oscillations when timing skew is present).

V. Self-timed vs. Self-correcting

In the preceding, we have been exploring the technique of prepatterning—constructing a spatial template and then performing irreversible and disruptive operations based on that pattern, in order to achieve some desired product. Self-timed patterning has played a valuable role, facilitating aggressive actuation that neither jumps the gun nor trips over itself by destroying its own inputs. Multiple stages of prepatterning and actuation can be cascaded, facilitated by the barrier mechanism. Results can be used as soon as they are available, with no need to wait for a conservative estimate of worst-case convergence time. Self-timed patterning and partial information thus improve the speed, robustness, and composability of the patterning process, compared to mechanisms that rely on pre-programmed delays or ad-hoc assumptions about timing.

In prepatterning, the target pattern is not a stationary point of the control algorithm. Indeed, careful steps must be taken to prevent the control algorithm from recursing on its outputs. However, prepatterning runs into difficulties when faced with unpredictable operations. Unpredictability demands closed-loop feedback to correct errors, and such feedback is most easily phrased in terms of recursion seeking a steady state. Furthermore, in the face of *asynchronous* damage (e.g. unexpected rearrangement that may occur during a patterning stage rather than merely during disruptive actuation), self-timing becomes only approximate; answers may be forced to change non-monotonically or to remain stale. Ultimately, the entire notion of prepatterning breaks down if serious damage can occur at any time, independent of the state of the computation. It is possible in this case to create self-timed, self-correcting loops, but it is not obvious whether self-timing really aids in solving the problem anymore.

In separate work, I have been investigating the opposite solution to the timing problem, self-correcting patterning, particularly as applied to traction-driven cell rearrangement. In this approach, cells apply forces to their neighbors in order to incrementally rearrange themselves towards the desired pattern. A self-correcting template defines the layout of the pattern within the substrate [14], while each region of the pattern thus designated runs a closed-loop control algorithm to enforce desired local features. The results are similarly robust to timing pathologies, but in many other ways they are dual to self-timed prepatterning: convergence is slow and indeterminate; completion is difficult to ascertain, and it is not obvious how to compose sequential stages. Even if convergence can be reliably identified, sequential composition is likely to interfere with the self-corrective ability of earlier stages. On the other hand, unpredictability is assumed, not avoided, and damage naturally heals.

VI. Conclusion

Self-timed prepatterning and closed-loop self-correction represent two extremes, both useful, both with some precedent in nature. Prepatterning is fast and composable; self-correction is robust and regenerative. I have demonstrated how the monotonic propagation of partial information facilitates convergence detection and checkpointing, crucial for implementing prepatterning. On the other hand, unpredictable operations, noise, and asynchronous damage degrade prepatterning, to the point that it can be come unviable. Bridging the gap between prepatterning and self-correction remains the goal of ongoing work.

Acknowledgment

The author would like to thank Gerald Jay Sussman, who supervised this work, and Alexey Radul, for many valuable discussions on the subject of monotonic partial information and propagators, along with the workshop organizers and the anonymous reviewers. This material is based on work supported in part by the National Science Foundation under Grant No. 1116294 and in part by a grant from Google.

References

[1] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978. [Online]. Available: http://doi.acm.org/10.1145/359545.359563

[2] D. Coore, "Botanical computing: A developmental approach to generating inter connect topologies on an amorphous computer," Ph.D. dissertation, MIT, 1999.

[3] R. Nagpal, "Programmable self-assembly: Constructing global shape using biologically-inspired local interactions and origami mathematics," Ph.D. dissertation, MIT, 2001.

[4] R. Doursat, "Programmable architectures that are complex and self-organized: from morphogenesis to engineering," in *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, S. Bullock, J. Noble, R. Watson, and M. A. Bedau, Eds. MIT Press, Cambridge, MA, 2008, pp. 181–188. [Online]. Available: http://www.alifexi.org/papers/ALIFExi_pp181-188.pdf

[5] V. Hamburger and H. L. Hamilton, "A series of normal stages in the development of the chick embryo," *Journal of Morphology*, vol. 88, no. 1, pp. 49–92, 1951. [Online]. Available: http://dx.doi.org/10.1002/jmor.1050880104

[6] J. M. W. Slack, *From Egg to Embryo: Regional Specification in Early Development (Developmental and Cell Biology Series).* Cambridge University Press, 1991.

[7] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective*, ser. European low-power initiative for electronic system design. Springer, 2001.

[8] G. J. Sussman and A. Radul, "The art of the propagator," MIT CSAIL, Tech. Rep. MIT-CSAIL-TR-2009-002, January 2009.

[9] A. Radul, "Propagation networks: A flexible and expressive substrate for computation," Ph.D. dissertation, MIT, 2009.

[10] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman, and R. Weiss, "Amorphous computing," MIT, Tech. Rep. AIM-1665, 1999.

[11] R. Doursat, "The growing canvas of biological development: Multiscale pattern generation on an expanding lattice of gene regulatory networks," *InterJournal: Complex Systems*, vol. 1809, 2006.

[12] A. S. Morgan and D. N. Coore, "Modeling intertia in an amorphous computing medium," in *The 6th International Workshop on Spatial Computing (SCW 2013)*, May 2013.

[13] M. Brodsky, "Deformable amorphous computing with foam-inspired surface mechanics," 2014, MIT CSAIL Tech. Rep., in preparation.

[14] ——, "Patterning with the rule of normal neighbors," 2014, MIT CSAIL Tech. Rep., in preparation.