BRITTLE BITS

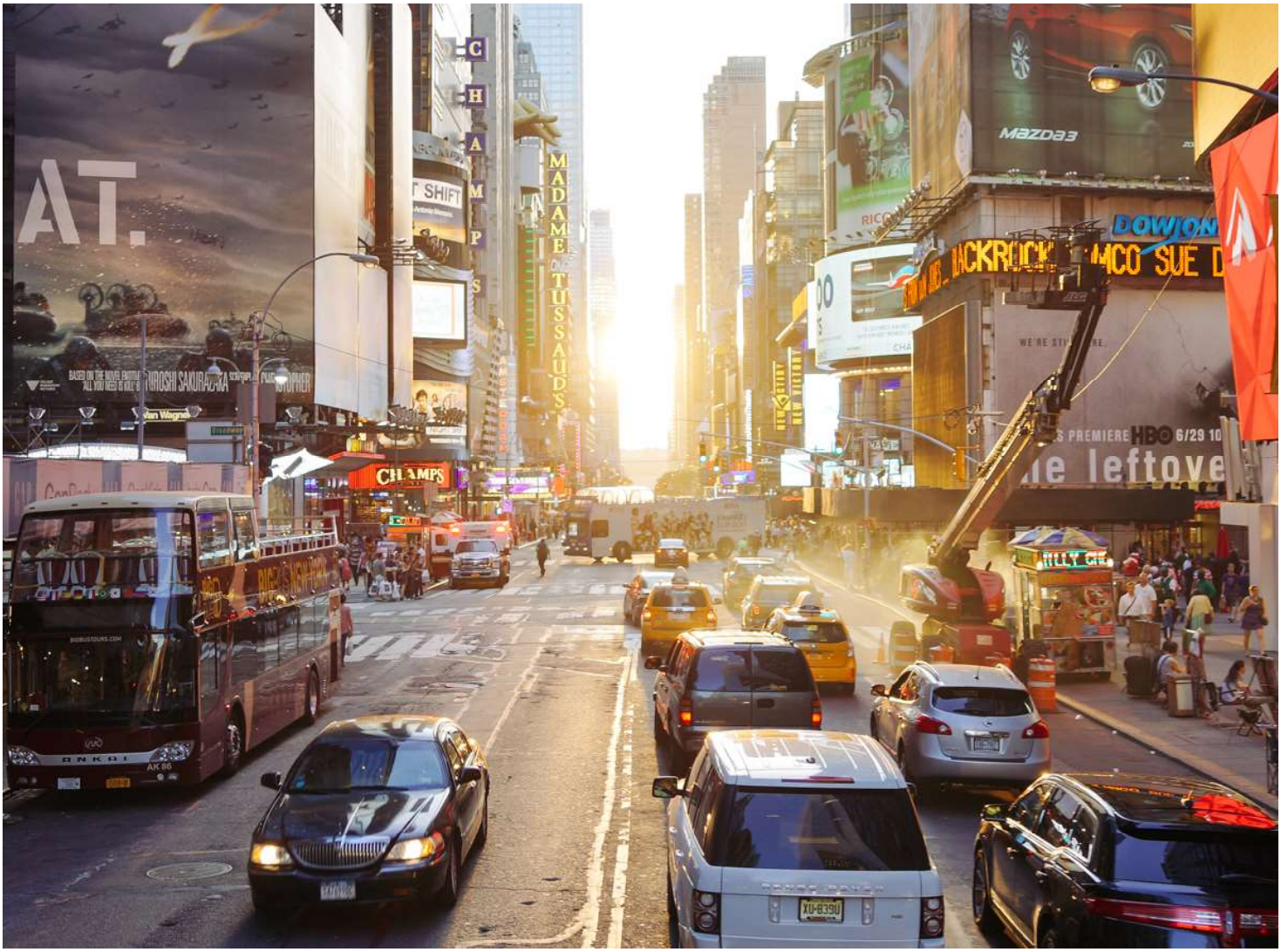
A visualization of code behavior by Ben Fry [1].

William Wu | **Brittle Bits**

Enveloped by the intricate electronic fabric of modern life, we often take for granted the myriad of digital systems that keep us safe, informed, and connected. From fire alarms to internet servers to the national power grid, we trust that these nearly-invisible systems are robust enough to uphold our civic infrastructure.

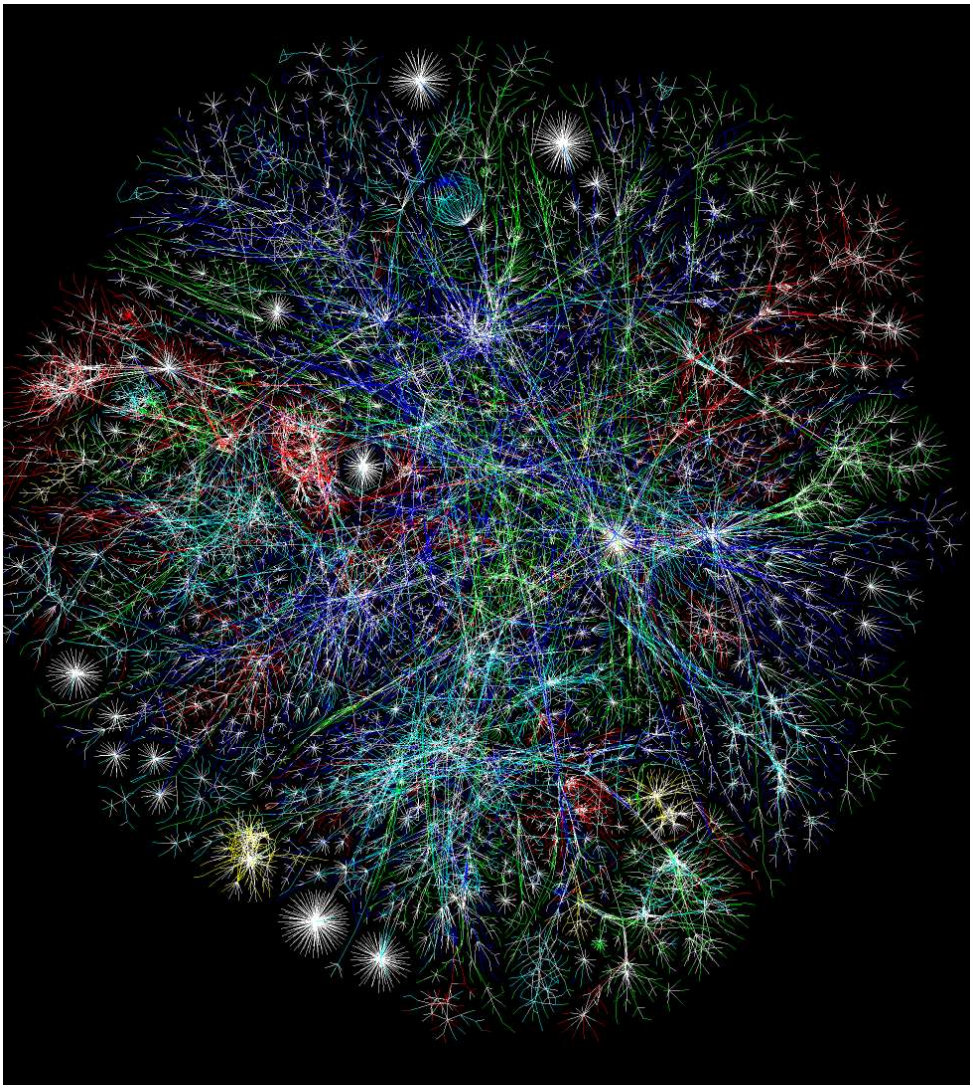
That is, until they break.

Unfortunately, many systems that we come to rely on are fragile and error-prone. An entire computer system can crash at the fault of a single component. More commonly, errors arise out of the opaque, often mysterious workings of code.



A bustling downtown New York City sunrise [2].

New York city durng the Northeast Blackout of 2003 [3].



A visualization of the Internet [5].

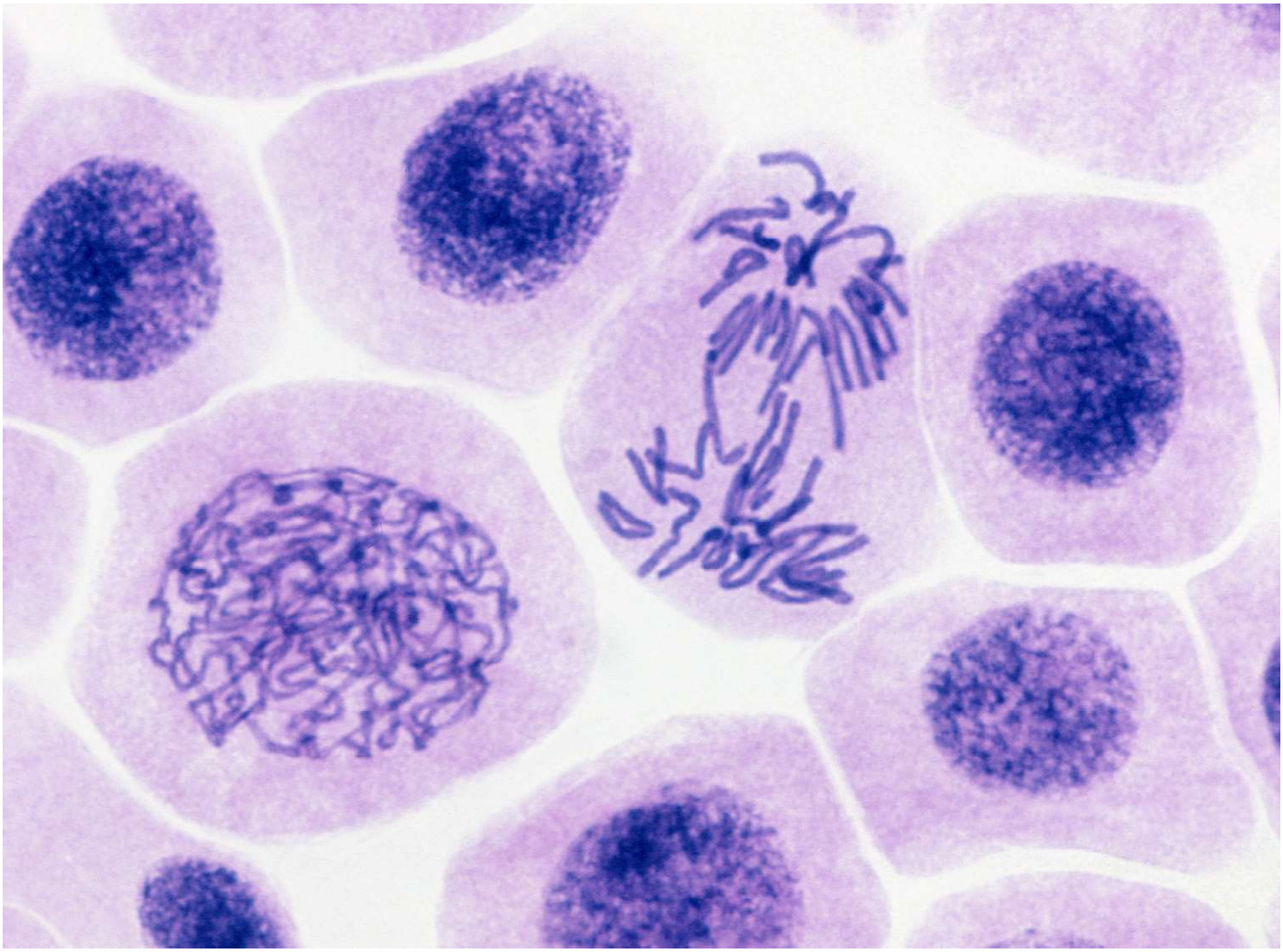
To anyone who chronically mistypes webpage URLs, the 404 “Page not found” error message is a common sight. Code is unforgiving of the ambiguity, variation, and randomness of real-world conditions. In unforeseen situations, code can unexpectedly misbehave and fail. One particularly well-known example is the Northeast Blackout of 2003, in which an energy management software error eventually caused a cascade of power line failures across the entire US Northeast.

The fragility of code and software-powered systems offers a stark contrast to the systems found in nature, which despite being exceedingly complex, are robust, distributed, and self-repairing. “There’s a distinction between Complex and Complicated,” says systems researcher Scott E. Page, “[We can build] very complicated things, but [they’re] not complex.” [4] Whether it be biomolecules or biomes, *E. coli* or ecosystems, the natural systems that surround us adapt to changing conditions, through growing, healing, or failing gracefully. Indeed, the Internet is one such human-designed system that also seems to exhibit these properties, but it is by far the exception, not the rule.

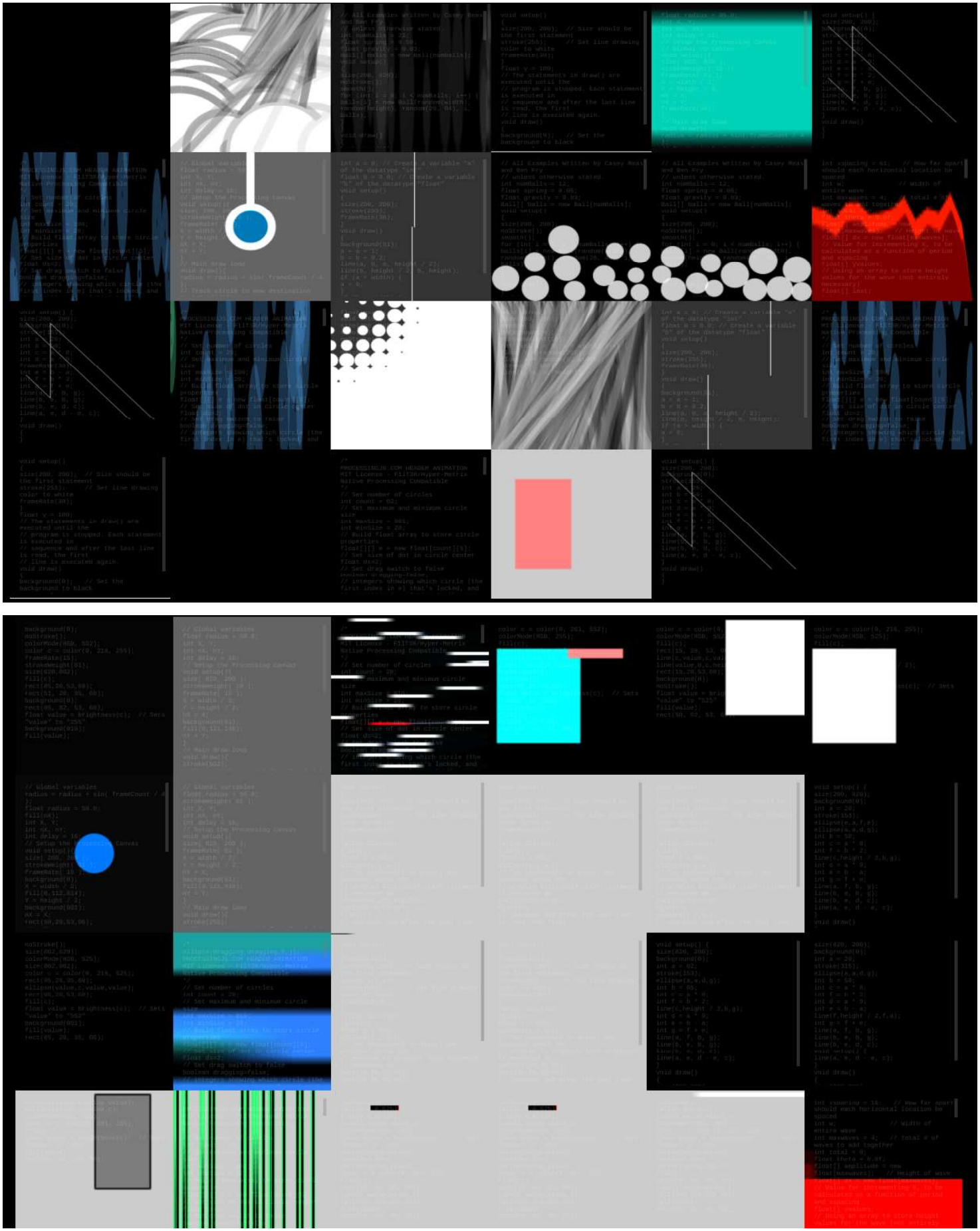
Brittle Bits seeks to represent software misbehavior in the context of biology. The cell is a convenient way to think about software systems - as self contained instructions, programs, or DNA. By creating an interdependent system of cells, cascading error can be propagated and visualized across the entire system.

I began with a grid of squares, each representing a “cell”. In my initial experiments, each cell had it’s own processing code and a countdown timer. When the timer ran out, the cell would either swap lines in its own code, randomly mix numbers in it’s own code, or select a random line to send to a randomly selected neighbor. The results are shown on the facing page.

Sometimes, interesting and novel cells would arise. But more commonly, broken or otherwise nonfunctional code would prevail. As evidenced by the gray and black boxes on the facing page, Processing code ended up too brittle to withstand the turbulent forces of random variation and evolution.



A cell’s DNA is in plain sight under a microscope [6].



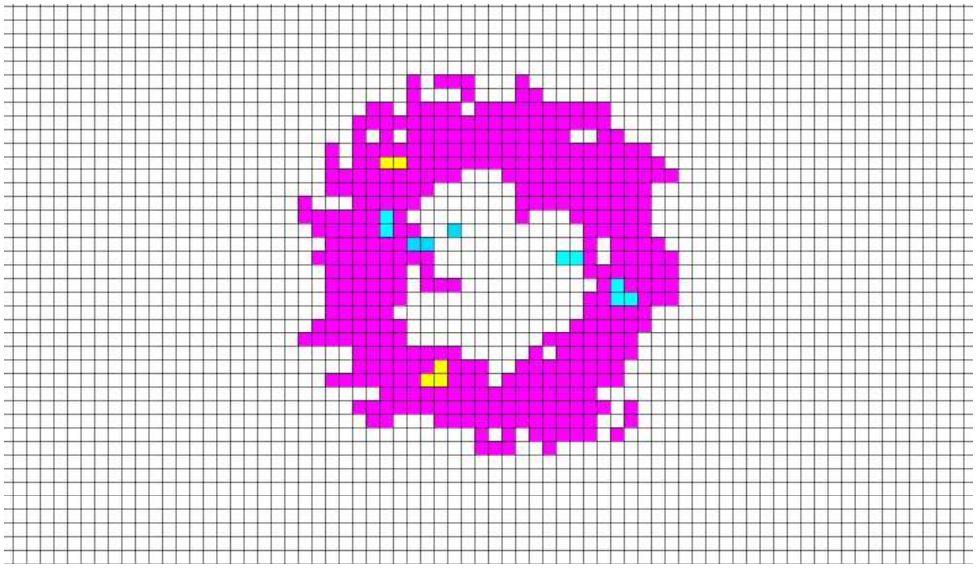
Code that starts out orderly (top) would mutate, swap code, and eventually break (bottom)

After additional exploration and experimentation, I decided to ditch the code swapping in favor of a new, color-based system, that I hoped would be more robust than the previous.

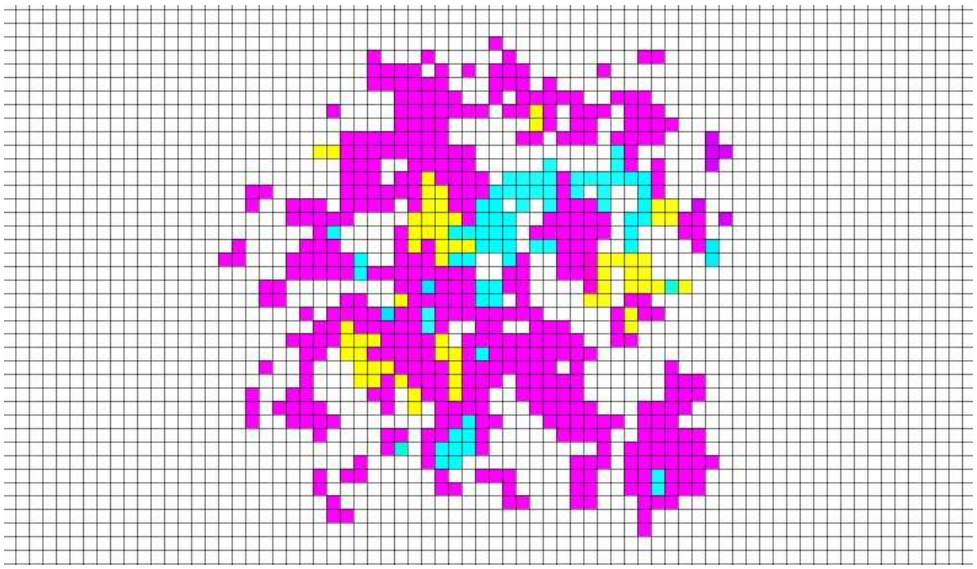
In the color system, each cell is colored by it's three number "genetic code", corresponding to the quantities of red, green, and blue. Like before, each cell also has countdown timer, reminiscent of a "cell cycle". When the timer expires, the cell "divides", creating a replica of itself in a neighboring cell's space and resetting its own timer to a randomized value. However, prior to division, the cell has a low chance of mutating, which involves swapping two of it's genomic numbers, one of which is also randomly varied. The grid is initially populated with entirely white cells, and user-placed cells begin as magenta (red: 255, blue: 255). The full process is represented visually on the facing page.

In a sense, the simulation hints towards how a genes would spread and mutate through a population, be it viruses, bacteria, or other genetic variation. On a large scale, these processes can be quite beautiful, as can be seen on the immediate right.

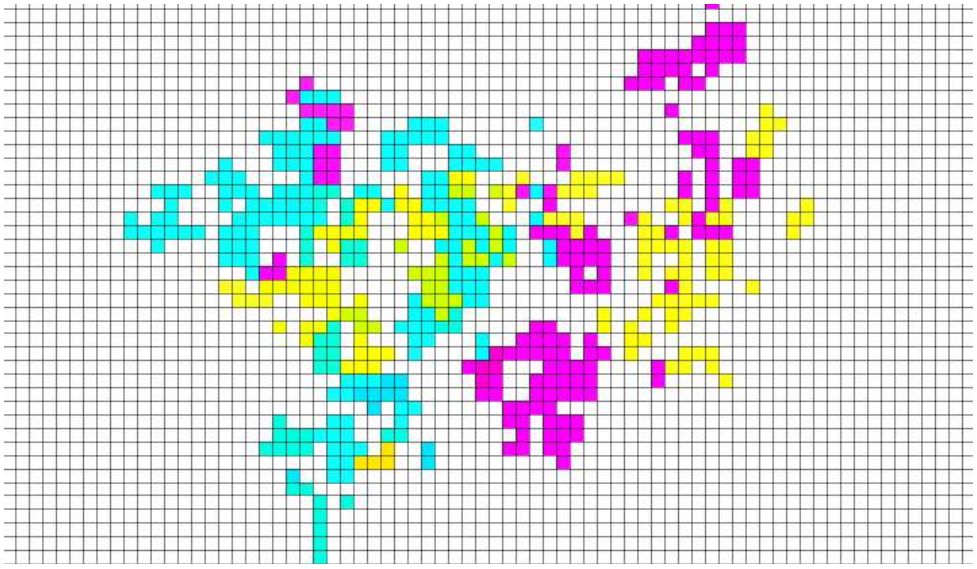
As a next step, I implemented the same color algorithm in 3D, with cubic cells. Again, similar patterns to the 2D version appeared. The results are shown on the following spread.



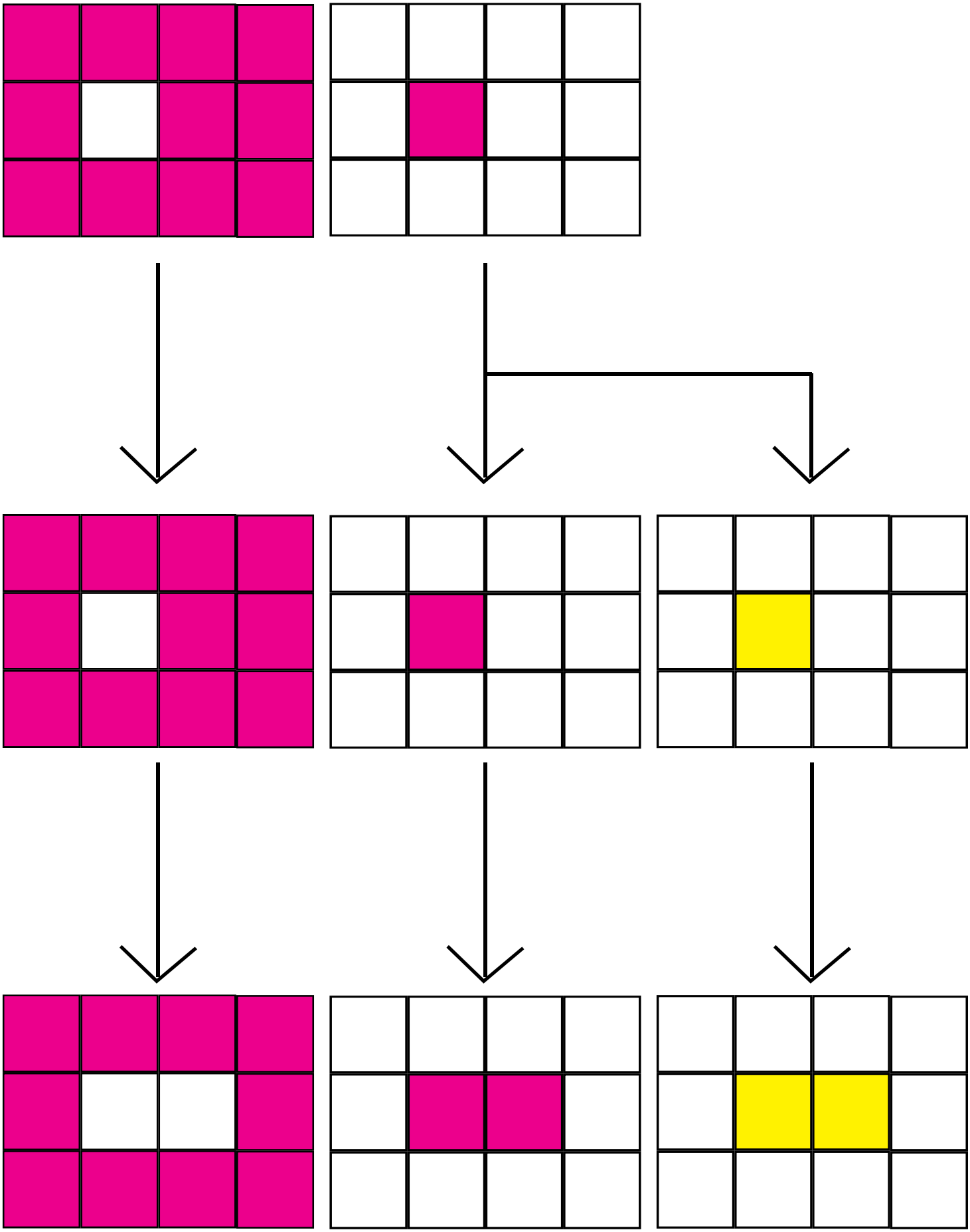
A user has just placed a ring of magenta cells.



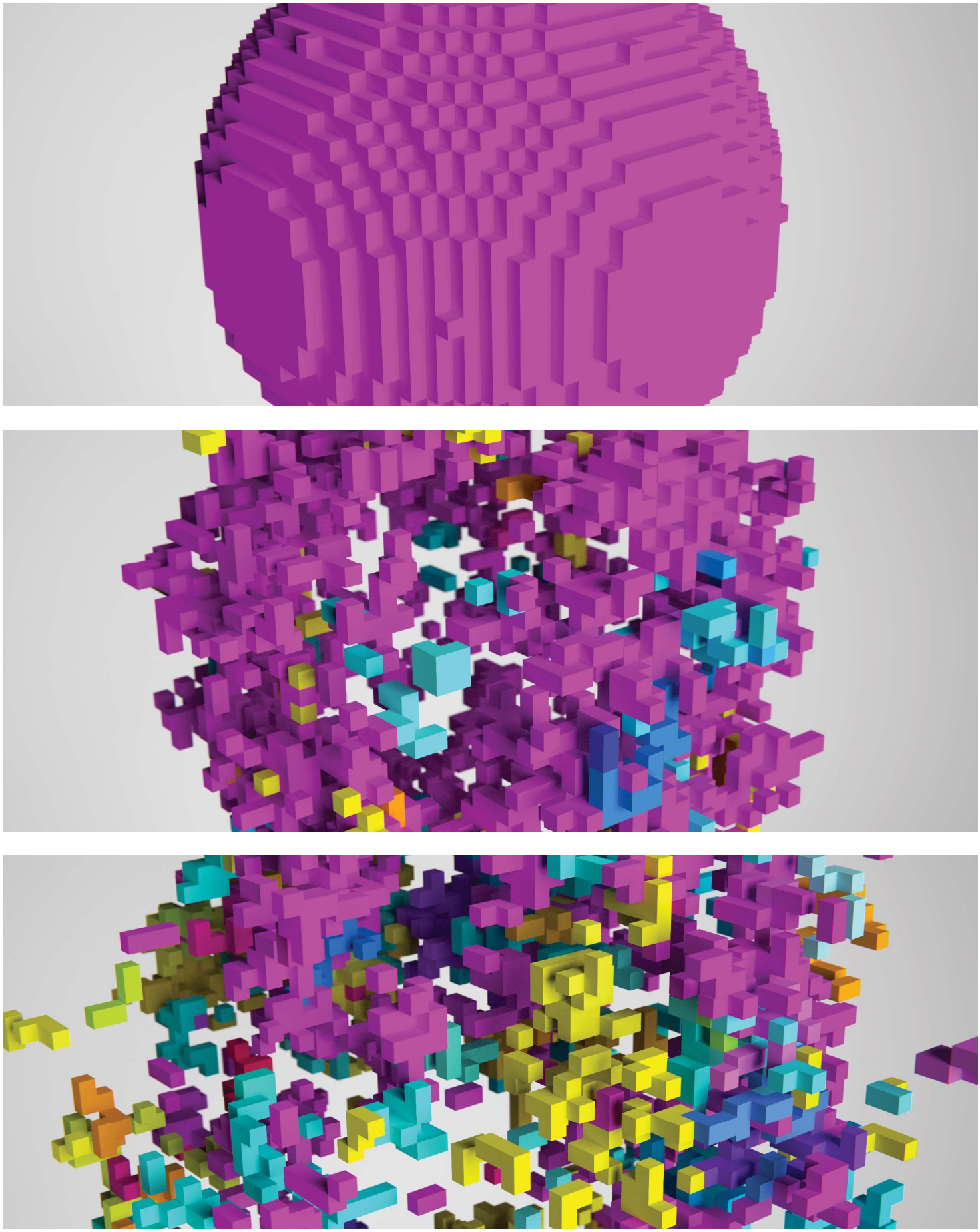
After a short period of time, the ring is hardly visible. Clusters of cells begin to form.



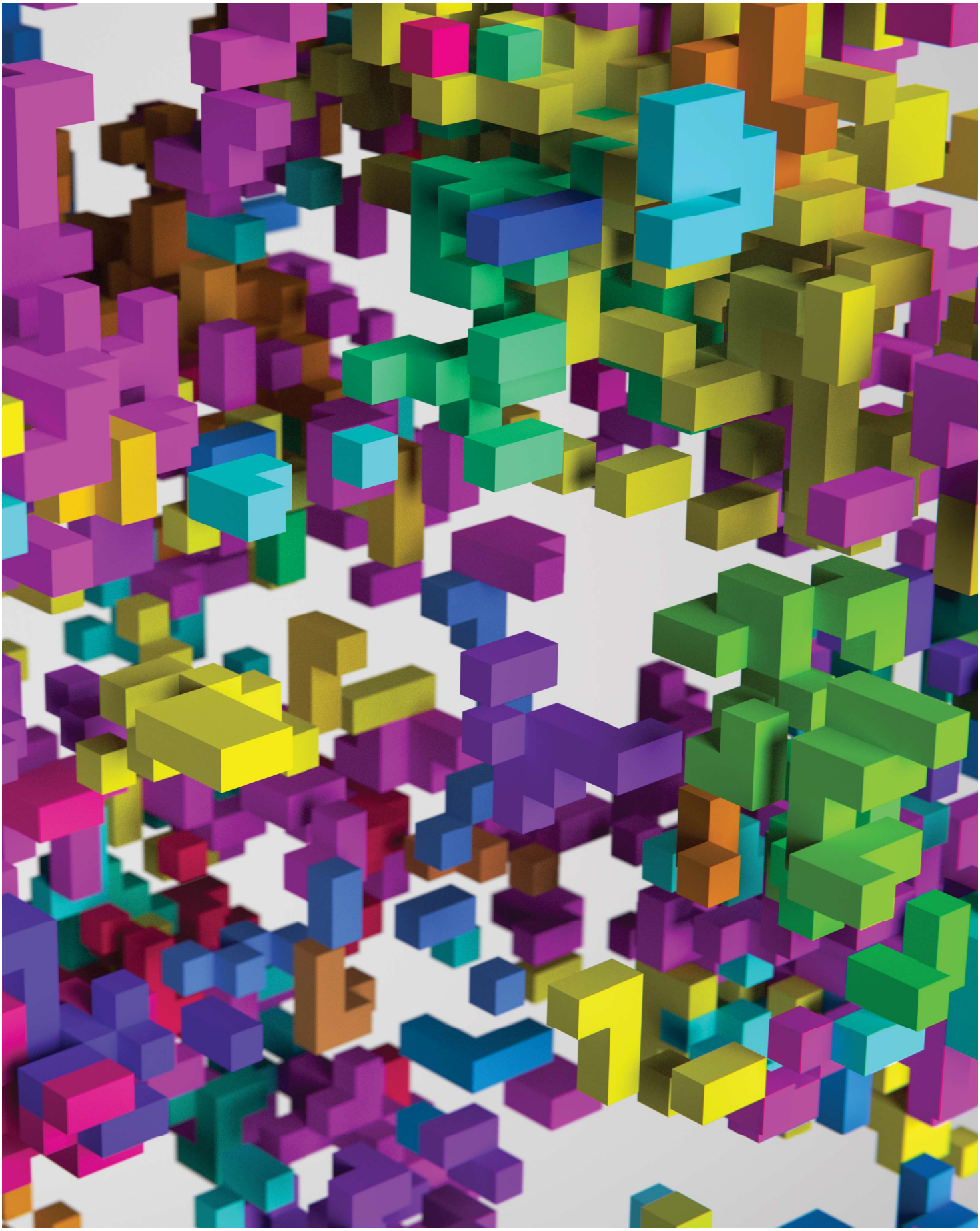
Over a longer period of time, additional colors and clusters emerge.



The algorithm governing cell evolution: when a cell's timer runs out (top), it conditionally mutates (middle), then replicates (bottom)



The turbulent processes of growth and mutation break the initial sphere apart.



A close-up on the colorful variation that emerges.

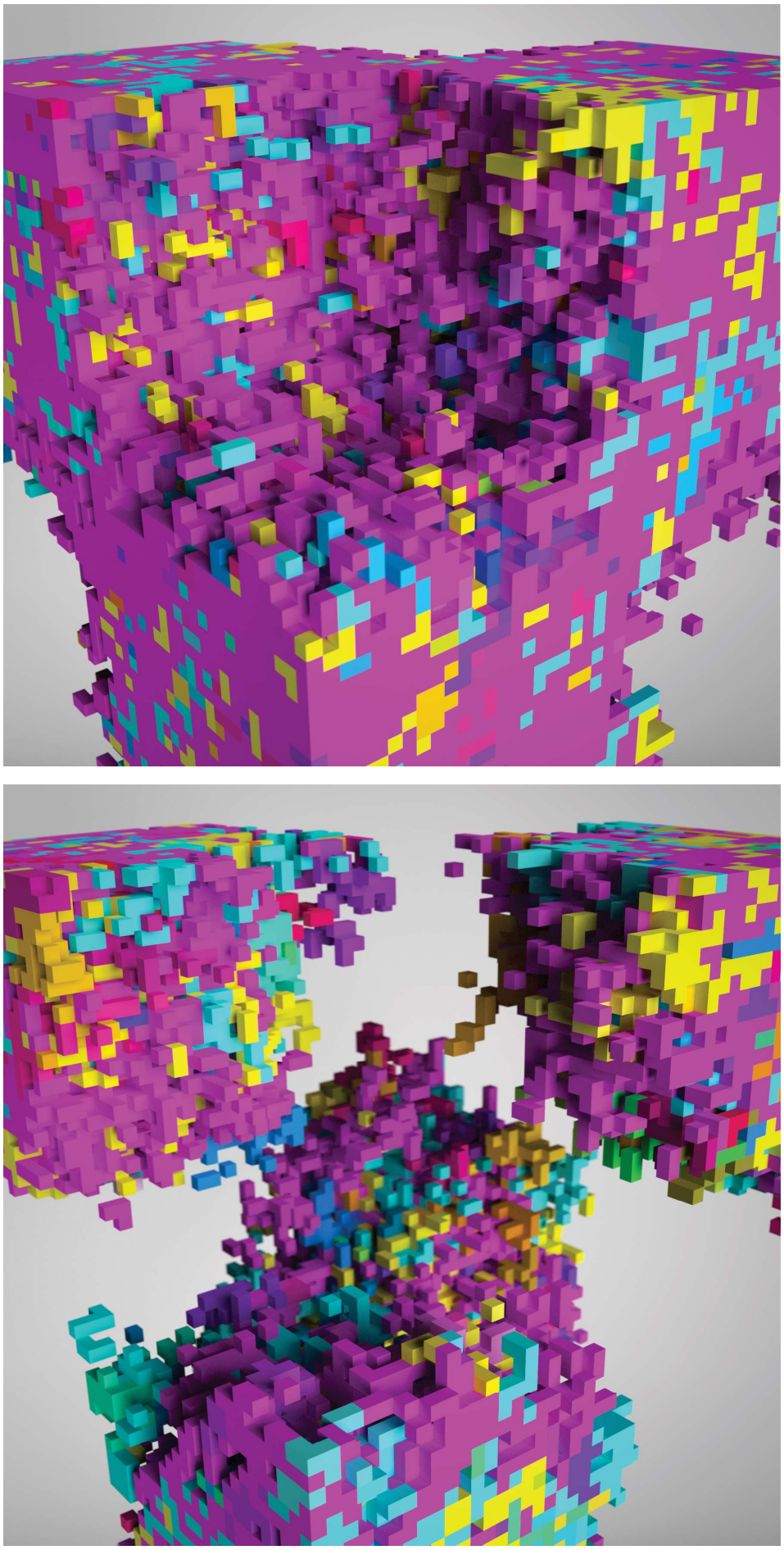
In the 3D version, the “white” cells are made transparent to aid viewing of the colored cells inside. It may seem that the colored cells are “alive”, or “grow”, and the transparent cells are “dead”. However, the algorithm treats all of these cells symmetrically. In reality, the transparent cells would also have colors and would also mutate, but the algorithm is designed to keep the white cells white to focus on the user-propagated error, rather than naturally occurring error.

As evidenced by the diagrams on the immediate right, the symmetry between growth and death is striking. From the same initial conditions, the top simulation was tweaked to accelerate user-propagated growth while the bottom one was tweaked to accelerate user-propagated death. But considered differently, the transparent cells had undergone death in the top simulation and growth in the bottom simulation.

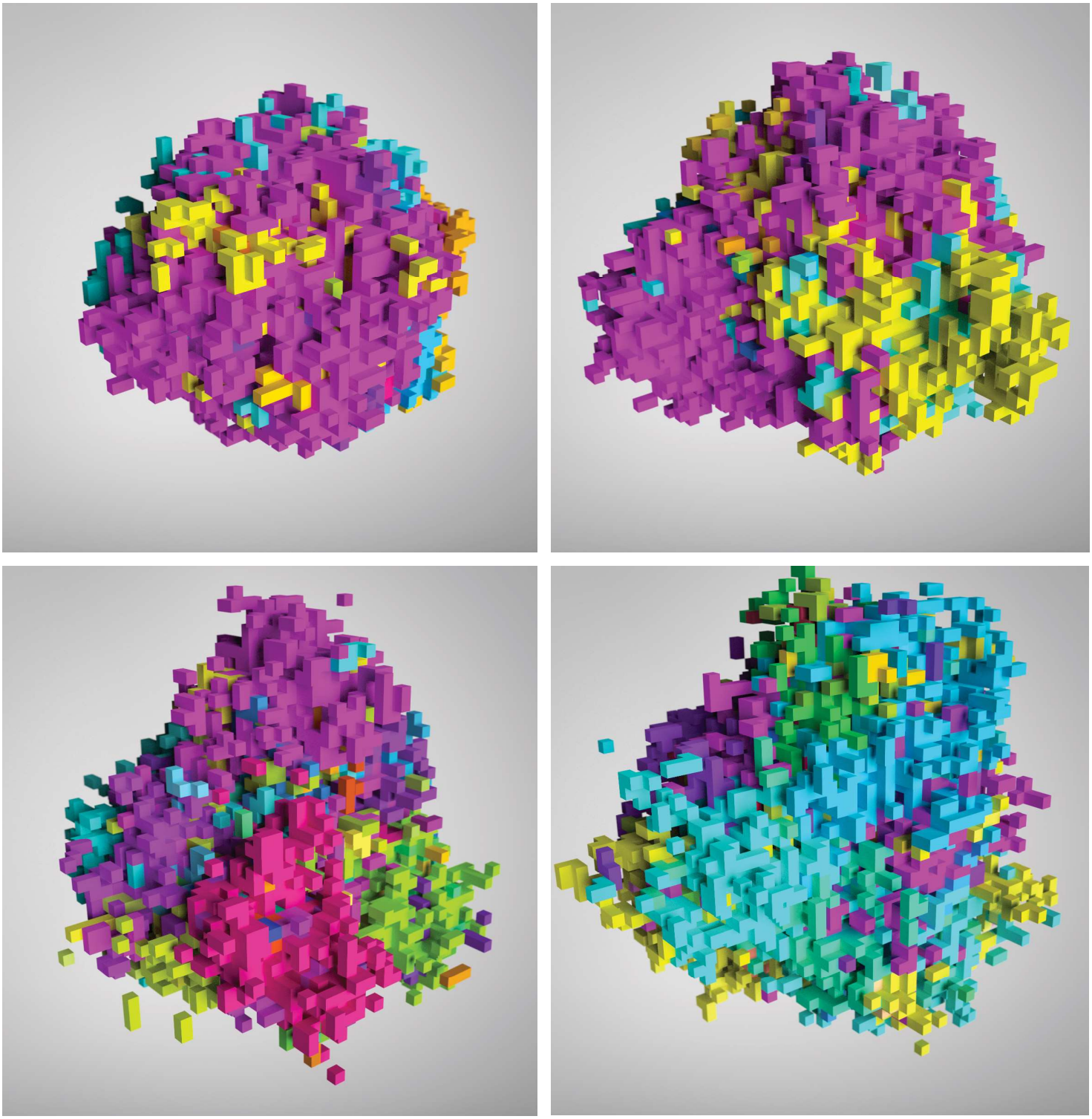
The equal and opposite behavior of transparent vs. non-transparent cells calls into question the what is proper behavior or erroroneous behavior, what is intentional or unintentional. Is there a difference between productivity and productive errors, between making and making mistakes? Is it possible for an outside observer to tell?

I continued to explore this dichotomy by varying the rates of growth and death. By repeatedly alternating between user-propagated growth and death, populations would alternatiedly blossom to hundreds of cells and shrink to a mere few. I’ve shown a few blossomed forms on the facing page, each successive bloom introducing unique features.

Besides mimicing a form of bottleneck natural selection, repeated growth and death also resembles the design process itself, as an iterative cycle of exploration and implementation, constantly growing, mutating, selecting, and evolving.



User-propagated growth (top) and death (bottom) look strikingly symmetrical



Four blossomed forms in sequence.

All in all, Brittle Bits strives to show how information propagates through, and sometimes breaks interconnected systems. Inspired by biology and made from code, its straightforward algorithms begin to yield varied and complex behavior, creating a microcosmic utopia that escapes the brittle nature of traditional code systems. The emergent symmetries call into question the difference between data flow and error flow, between productivity and productive error.

References

- [1] http://benfry.com/dismap/

- [2] http://www.minimallyminimal.com/blog/nyc-01

- [3] http://www.lloyds.com/cityriskindex/threats/power_outage/case-study

- [4] Page, Scott E. (2015, July 15). Knotty Objects: Design and Complexity. Video: from http://www.media.mit.edu/video/view/knotty-objects-2015-07-16-09

- [5] http://research.blogs.lincoln.ac.uk/files/2011/02/map-of-internet.png

- [6] https://commons.wikimedia.org/wiki/File:Mitosis_(261_14)_Pressed;_root_meristem_of_onion_(cells_in_prophase,_anaphase).jpg