Computational Tectonics

Francis Bitonti
FADarch, New York City College of Technology

HARDWARE

ABSTRACT

The goal of this research is to define a methodology for the construction of complex non-repeating surfaces and structures that rely on the formulation of a singular tectonic mechanism. Computational systems like cellular automata seem to suggest that it might be possible for modular material systems to self-assemble into complex organizations. A single series of modular parts could be capable of producing not only complex behavior, but also, depending on initial conditions, simple periodic behavior. The research outlined in this paper uses simple geometric transformations to produce tectonic computers that can be applied to a variety of building systems. These "lock-and-key" mechanisms allow for the construction of complex surfaces and structures, with minimal instructional documents. The aim of this research is ultimately to redefine construction methodology as a series of localized operations that eliminate the need for centralized planning.

This paper outlines a methodology for encoding and decoding material assemblages as discrete computational systems. Exploiting the combinatorial nature of tectonic systems makes it possible to produce a population of "material algorithms" capable of exhibiting a wide range of behaviors. Encoding assemblages as discrete systems affords the designer the ability to enumerate and search all possible permutations of a tectonic system. In this paper, we will discuss the calculations and computational processes used to encode material assemblages as populations of discrete algorithms. The application of this methodology to the design and assemblage of a series of vertical stacking masonry blocks will also be discussed in detail.

1 INTRODUCTION

A Cellular Automaton is a discrete computational model studied in a variety of scientific disciplines. Cellular Automata are generally visualized as arrays of discrete elements, a one or a zero. As each discrete element looks at neighboring or adjacent cells once at each time step, its state evolves with time. For example, in elementary cellular automata, each cell can potentially hold one binary state, and it looks at the two nearest neighbors on the grid for instructions regarding its current state. For example, if given cells number one, two and three, cell number two will define its current state by looking at three distinct positions in the array. It will look at itself (c), cell number one (c-1), and cell number three (c+1). The number of neighbors it looks at is defined by what shall be referred to as the automaton's radius. For example, an elementary cellular automaton has a radius of 1. Three cells will be evaluated before calculating one cell in the following evolution. Imagine a row of ones and zeros and that we want to know what the center-most value will be after one evolution of the system. With a radius of 1, we would have to look at one cell to the left and one cell to the right in order to determine the next evolution of the center cell. In a ½-radius automaton, we would have to look at ½ a cell on the right and ½ a cell on the left; as a result, only one cell would be examined.

Knowing the radius and number of possible states, we can construct a rule table. The rule table refers to the results of all possible combinations of potential neighborhoods, and all the potential resulting states in the next evolution. This combinatorial property of discrete systems allows us to enumerate all the possible rules and resulting behaviors, thereby, producing a nonhierarchical, finite search space of potential behaviors.

Abstract models such as these translate nicely into modeling tectonic systems. These models assume that each new addition is dependent on some portion of the existing assemblage. Each element in the global assemblage has an associated neighborhood. These neighborhoods provide an entry point into modeling tectonic systems. For example, in most building systems, each successive addition to an assembly is made possible only by the preceding additions to the structure. Although, historically, most tectonic systems have produced linear and deterministic behavior, using the methodology outlined in this paper, it is possible to create systems of assembly that self-structure into a variety of organizations using the same system of connections.

2 ENUMERATING

Let us begin by building a system with two possible states and a ½ radius. State one shall be labeled as 0 and state two shall be labeled as 1. A ½-radius system will have to evaluate a neighborhood of two cells {0,1},{1,1},{0.0}, etc… and then produce a new cell of some resulting value, either a zero or a one. Table 1 outlines neighborhood configurations for both automatons, with a radius of ½ and of 1, respectively. Calculation of these sets will be discussed in succeeding paragraphs.

Table 1: Sample Neighborhood Enumerations for Binary Rules (0 or 1)

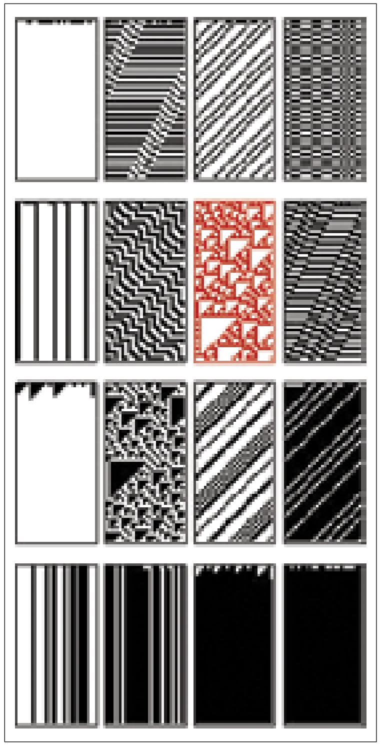
Radius	
½ = neighborhood of 2 cells	{0,0},{0,1},{1,0},{1,1}
1 = neighborhood of 3 cells	{0,0,0},{0,0,1},{0,1,0},{0,1,1},{1,0,0},{1,0,1},{1,1,0},{1,1,1}

We repeat this process to calculate all possible results for each neighborhood configuration. However, this time, we substitute the number of cells in each neighborhood for the length of the output from table 1 so that each neighborhood has one digit associated with it. For example, in the ½-radius rule, we want to enumerate all the possible combinations of 0 and 1 with a length of four; one possible output might be {1,1,0,1}. Table 2 contains some example output that illustrates this point.

Table 2: All Possible Outcomes for Potential Neighborhoods

Neighborhood Size	Possible Resulting States
½ radius = neighborhood size of 4	{0,0,0,0},{0,0,0,1},{0,0,1,0},{0,0,1,1},{0,1,0,0},{0,1,0,1},{0,1,1,0},{0,1,1,1}, {1,0,0,0},{1,0,0,1},{1,0,1,0},{1,0,1,1},{1,1,0,0},{1,1,0,1},{1,1,1,0},{1,1,1,1}
1 radius = neighborhood size of 8	{0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,1},{0,0,0,0,0,0,1,0},{0,0,0,0,0,0,1,1}, {0,0,0,0,0,1,0,0},{0,0,0,0,0,1,0,1},{0,0,0,0,0,1,1,0},{0,0,0,0,0,1,1,1}… Results are too long for publication total solutions equal 256

Figure 1 All ½ radius two color rules from random initial conditions



We then superimpose the two matrixes, constructing a table of all possible rules. The superimposition of the two arrays matches each neighborhood in table 1 with the results in table 2; the neighborhood for the ½-radius automaton looks like, {0,0},{0,1},{1,0},{1,1}. Then we sequentially match each of the possible resulting states from table 2 with each of the neighborhoods from Table 1. For example, if we take the second entry from table 2 for the ½-radius automaton {0,0,0,1} and match it with the potential neighborhoods {0,1},{1,0},{0,0},{1,1}, the resulting rule will be {0,0}=0, {0,1}=0, {1,0}=0, {1,1}=1. See table 4 for an example of a rule table for a ½-radius automaton. Sixteen rules are possible.

Table 4: All Possible Rules for a ½ Radius Automaton

{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=0},	{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=0},
{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=1},	{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=1},
{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=0},	{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=0},
{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=1},	{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=1},
{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=0},	{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=0},
{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=1},	{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=1},
{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=0},	{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=0},
{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=1},	{{0,0}=0,{0,1}=0,{1,0}=0,{1,1}=1},

2.1 RULE EXECUTION AND SEARCHING BEHAVIORS

One of the advantages of working with discrete systems, such as we have discussed above, is that it enables the designer to know the full range of possible behaviors that the system is capable of. The advantage of working through a design in this way, as a set of abstract parameters, is that it is now possible to run tests quickly on these rules and to perform statistical analysis on the output, helping us find rules that work best for our applications. Essentially, it affords you the opportunity to write algorithms that find other algorithms that exhibit a certain behavior. Although searching a space of algorithms is not intended to be the topic of this paper, it will be discussed briefly because, on large rule sets, it will be necessary to work this way.

For example, one desired property of a system might be to find rules that give drastically different behavior when they are run from different initial conditions. In the rule space shown below (fig. 1), we find that rule 6 exhibits this behavior. On simple systems such as these, we can locate this behavior visually; however, on complex systems containing thousands of rules, this is not always possible. In this case, a simple program can be written that runs each rule on a variety of initial conditions and that measures the entropy in each system. The rules that show the most varied output are selected for more detailed visual analysis. Figure 1 and figure 2 are both plots of the same ½-radius rule space: figure 1 is calculated from a random initial condition; figure 2, from a single black cell against a white background. Rule 6, highlighted in red, shows that it is capable of both periodic and random behavior.

It becomes possible to apply this methodology to construction when we construct a formal/geometric language for encoding these binary digits. Because they are symbolic and discrete systems, they can be encoded quite simply as a series of lock-and-key mechanisms. The following section will discuss a method for encoding these systems geometrically.

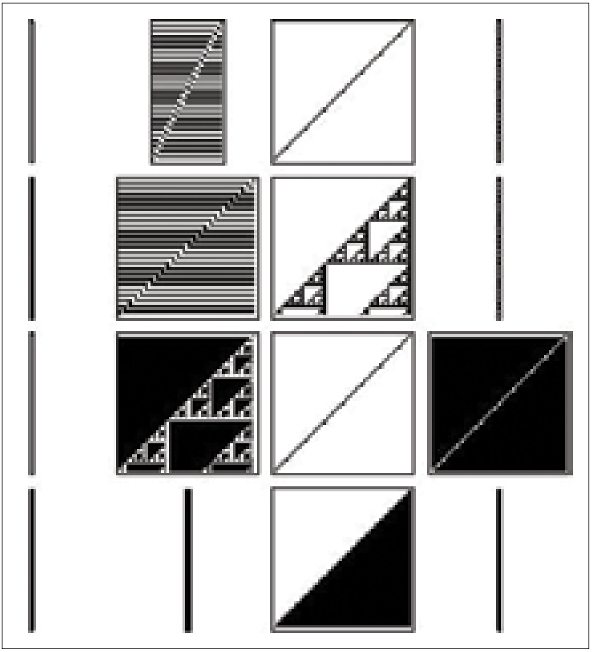


Figure 2 All ½ radius two color rules from an initial condition of one black cell

3 CASE STUDY: "LOKI-BLOCKS"

When designing "Loki-Blocks" the objective was to create a material system capable of producing a variety of complex organizations that could also be assembled to specification without complex instructions. In short, the goal was to create a connection that was self-structuring in the same way that a cellular automaton is self-organizing. In a typical approach to tectonics, one set of connections produces only one configuration. We wanted the same product to produce both periodic and random behaviors.

"Loki-Blocks" is a project that attempts to solve this problem in two dimensions. Running bond brick patterns produce the same neighborhoods we have seen in the ½-radius cellular automatons discussed above. Each brick on the successive lines rests on two bricks in the previous level. The top brick will only exist if we have two beneath it. Figure 3 shows how these two units can be assembled in a method similar to the running bond brick patterns discussed above.

Figures 4 and 5 show two sample units used in this material system. Note the similarity of interlocking connections at the top and bottom; this feature will be discussed in a following section.

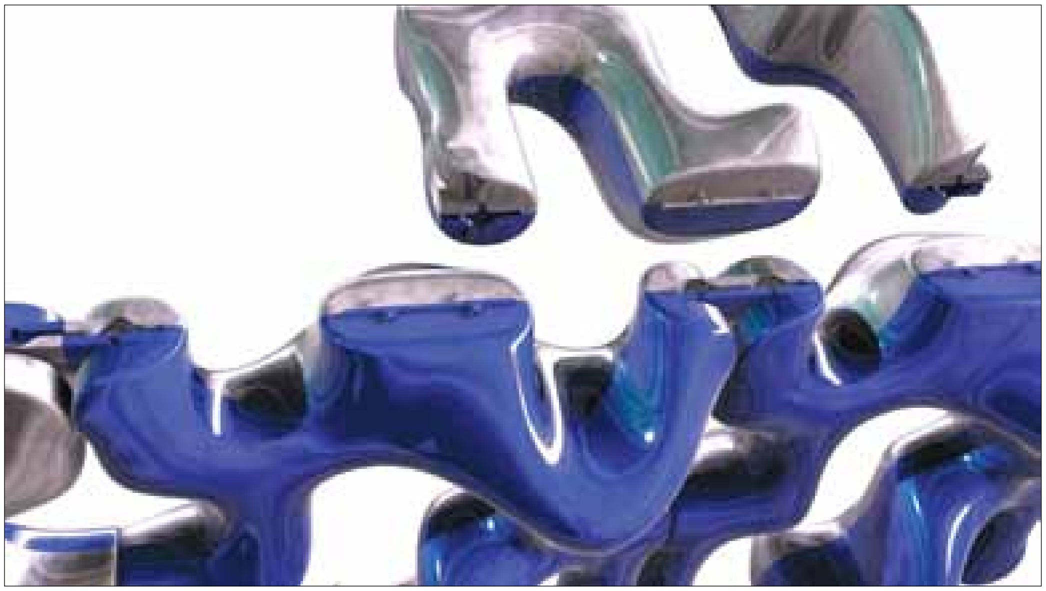


Figure 3 Rendering of "Loki-Block" Assemblage

In traditional masonry construction, these elements are simply stacked on top of one another. However, if a series of male/female interlocking connections were installed on the top and bottom, restricting the possibilities of the installer, we could evoke some of the same behaviors exhibited by a cellular automaton, including what is known as universality. Computational universality is the ability for a system to emulate the behavior of any other computational system. In theory, this research could lead to the development of a universal tectonic.

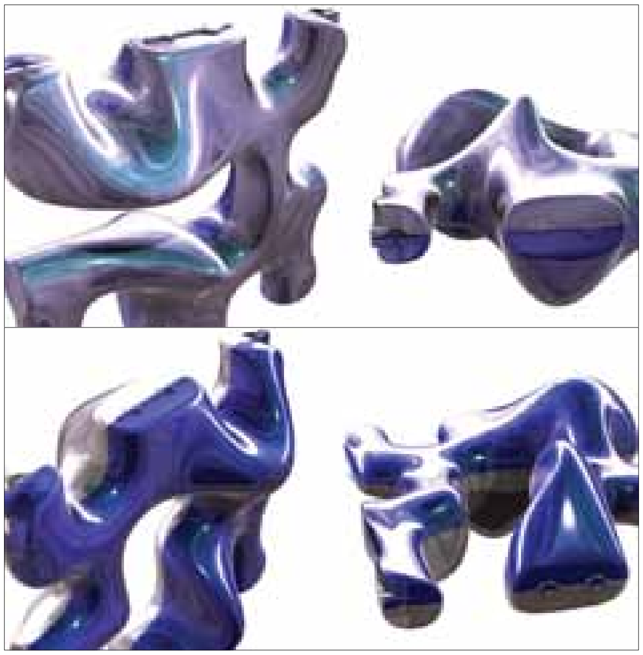


Figure 4 Unit one from “Loki-Block” Assemblage

Figure 5 Unit two from “Loki-Block” Assemblage

3.1 CONNECTION GEOMETRY

This portion of the paper will discuss a few design guidelines for determining the geometry of the interlocking connections. It is not the intention of this paper to define absolute principles for determining these geometries. However, in conducting this research, a number of guidelines have emerged that seem to be intrinsic to developing geometries capable of simulating the types of systems discussed in this paper.

The methodology outlined here revolves around the design of a single interlocking mechanism that can be reconfigured to produce a set of “states” similar to those used by a cellular automaton. This design methodology assumes the designer is working with traditional static building materials and not shape-memory alloys or plastics that can be programmed to perform deformations. As a result, we will use rotation, a simple geometric transformation, that can be carried out in physical space. If we rely on the rotation of the part in space to transform the geometry of the connection, we can design each connection with “x” number of symmetrical axis to produce the different states.

We can encode a binary system in the following manner, using rotation as a primary means of geometric transformation. The number of different states in a system equals the number of asymmetrical axes contained in the geometry. For example, a circle is symmetrical on all axes. A circle would be used for a system with one state; a half circle can be rotated 180 degrees and give us two possible states. Figure 6 shows a plan view of the connections generated for “Loki-Blocks.” The shape outlined in figure 6 utilizes a half circle with a line in the center of the flat portion. This shape is symmetrical about the Y-axis but not the X. Applying a 180-degree rotation transforms the geometry into the other state. The degree of rotation required to change into each state can be calculated by dividing 360 by the number of states. For example, in this rule with two states, we divide 360/2 and get 180.

We can then enumerate all the female connections as shown in figure 6; these are the elongated figures in the center of the block. These will be the connections where the two halves are already joined together. We end up with a total of four {1,1},{1,2},{2,1},{2,2}. This number tells us the maximum number of blocks that will be required to accommodate all the different connection types. The sizes of these elements are determined by the neighborhood size. With a radius of 1, we will aggregate the unit in arrays of three as opposed to the two shown in this example with a ½ radius. Essentially, you want to create a whole that is divisible by the number of parts defined by the neighborhood size so that the figure can be reconstructed when the other units are adjacent to each other.

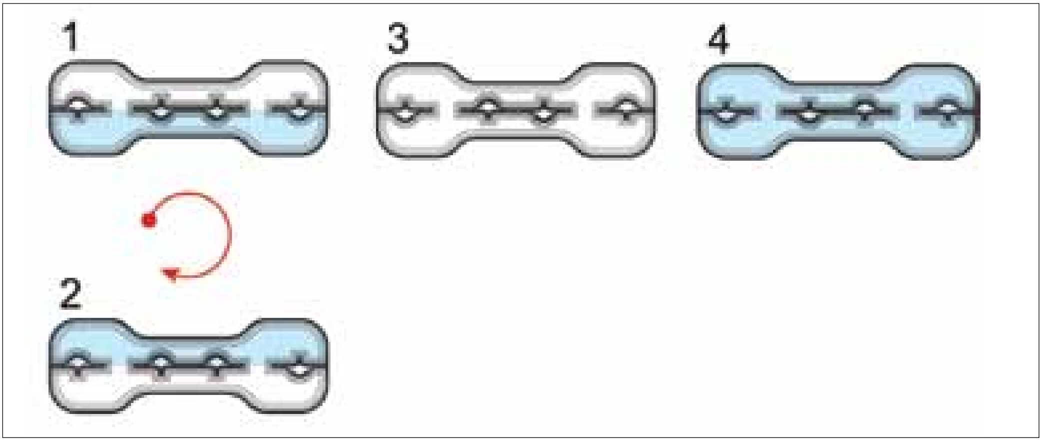


Figure 6 “Loki-Block” Plan drawings of connection types

The actual number of blocks required can be calculated by finding arrays of the same value, for example, {1,1} or {0,0}. In order to calculate the number of blocks that can be consolidated, you need to identify what shall be called transformable pairs. It is necessary to identify formations capable of transforming into other formations so that we can minimize the amount of material being used and reduce the ability for human error. As a result, we need to define transformation rules for each state in the system. For example, 0=1 and 1=0 when a rotation of 180 degrees is applied. {0,1}={1,0} is not a transformable pair because the geometry is asymmetrical on two axis, and it would have to be mirrored to take on the state of its sister geometry. However, {0,0}={1,1} is a transformable pair. For arrays longer than two, we need to search for symmetrical distributions of values. For example, in an automaton with a radius of 1, {1,0,1} or {0,0,0} could be coupled with {0,1,0} and {1,1,1}. Rotating the entire piece 180 degrees transforms these cell types into one another. The left-most connection type in figure 6 demonstrates this property. In this case study, we have one transformable pair and, therefore, will need three blocks.

The enumeration process will define the geometry at the ends of the blocks. In calculating all the possible rules, it is necessary that we calculate the distribution of geometry at the ends of the blocks. In section 4, this process will be discussed in detail. These elements are variable and are used to generate different types of behavior. The center geometry is fixed and is used in every case. The center geometry represents all the possible resulting neighborhood configurations. In the end, we can create a computer program or use parametric modeling applications to visualize the design of all the connections, and generate a diagrammatic representation of its behavior.

4 ENUMERATING "LOKI-BLOCKS"

"Loki-Blocks" was conceived as a vertically stacking wall assembly. When designing the connections for "Loki-Blocks," first the stacking method was determined. After establishing a method of assembly, it was possible to determine a neighborhood size by counting the number of adjacent cells. In the case of "Loki-Blocks," a method of assembly was chosen where one block would be placed on top of two other adjacent blocks. The center of one block would always be plugged into the edges of two others. It is necessary that our connection result in a rule that looks something like this: {i,j}=x. Using a rule like this, we could simulate the aggregation of blocks that would take place during the construction process. Given i and j, we can design two distinctive geometries that will be represented by these symbols. The logic behind the construction of this geometry has been outlined in section 3.1. We can then use the enumeration technique discussed in section 2 to construct all the possible configurations; as a result, we end up with four configurations {i,i},{i,j},{j,i},{j,j}.

Enumerating all the rules requires that we know both the number of modules in the physical system of parts and the number of neighbors that will be evaluated. We know from previous sections that our physical system works with a neighborhood of two cells; and we established the actual number of blocks by determining the number of transformable pairs in section 3. To summarize, our material system contains three blocks and builds on top of a neighborhood containing two cells, each with two possible states. In order to calculate all the necessary permutations for these end conditions, we will calculate all the possible combinations of the available neighborhoods (discussed in the following paragraph), using an array with three spaces because we have three blocks. For example, {i,i},{i,j},{j,i} or {i,j},{j,i},{j,j}, etc. As shown by tables 5 and 6, we have 64 combinations that can be used for the design of our male connections.

The software for this project was written using Mathematica from Wolfram Research as a development environment. Table 5 shows the code used to enumerate all the pairs. "Tuples" is a function that allows us to enumerate all the possible configurations of a particular set. Table 6 contains all the possible male connection types for the geometry that was calculated using the code in table 5.

Table 5: Enumeration Pseudo Code

```
X= Tuples[{1,2},2]
X= {{1,1},{1,2},{2,1},{2,2}}
Tuples[x,3] = All possible male connection types
```

Table 6: Enumeration of all possible male connection types (Tuples[x,3])

{{1,1},{1,1},{1,1}},	{{1,1},{1,1},{1,2}},	{{2,1},{1,1},{1,1}},	{{2,1},{1,1},{1,2}},
{{1,1},{1,1},{2,1}},	{{1,1},{1,1},{2,2}},	{{2,1},{1,1},{2,1}},	{{2,2},{2,1},{1,2}},
{{1,1},{1,2},{1,1}},	{{1,1},{1,2},{1,2}},	{{2,2},{2,1},{2,1}},	{{2,2},{2,1},{2,2}},
{{1,1},{1,2},{2,1}},	{{1,1},{1,2},{2,2}},	{{2,2},{2,2},{1,1}},	{{2,2},{2,2},{1,2}},
{{1,1},{2,1},{1,1}},	{{1,1},{2,1},{1,2}},	{{2,2},{2,2},{2,1}},	{{2,2},{2,2},{2,2}},
{{1,1},{2,1},{2,1}},	{{1,1},{2,1},{2,2}},	{{2,1},{1,1},{2,2}},	{{2,1},{1,2},{1,1}},
{{1,1},{2,2},{1,1}},	{{1,1},{2,2},{1,2}},	{{2,1},{1,2},{1,2}},	{{2,1},{1,2},{2,1}},
{{1,1},{2,2},{2,1}},	{{1,1},{2,2},{2,2}},	{{2,1},{1,2},{2,2}},	{{2,1},{2,1},{1,1}},
{{1,2},{1,1},{1,1}},	{{1,2},{1,1},{1,2}},	{{2,1},{2,1},{1,2}},	{{2,1},{2,1},{2,1}},
{{1,2},{1,1},{2,1}},	{{1,2},{1,1},{2,2}},	{{2,1},{2,1},{2,2}},	{{2,1},{2,2},{1,1}},
{{1,2},{1,2},{1,1}},	{{1,2},{1,2},{1,2}},	{{2,1},{2,2},{1,2}},	{{2,1},{2,2},{2,1}},
{{1,2},{1,2},{2,1}},	{{1,2},{1,2},{2,2}},	{{2,1},{2,2},{2,2}},	{{2,2},{1,1},{1,1}},
{{1,2},{2,1},{1,1}},	{{1,2},{2,1},{1,2}},	{{2,2},{1,1},{1,2}},	{{2,2},{1,1},{2,1}},
{{1,2},{2,1},{2,1}},	{{1,2},{2,1},{2,2}},	{{2,2},{1,1},{2,2}},	{{2,2},{1,2},{1,1}},
{{1,2},{2,2},{1,1}},	{{1,2},{2,2},{1,2}},	{{2,2},{1,2},{1,2}},	{{2,2},{1,2},{2,1}},
{{1,2},{2,2},{2,1}},	{{1,2},{2,2},{2,2}},	{{2,2},{1,2},{2,2}},	{{2,2},{2,1},{1,1}},

This calculation enumerates all the male connection types possible on each of the three blocks. However, one of these blocks contains a transformable pair, discussed in section 3.1. In order to construct a complete rule table, you need to know all the possible states at any given point in time. It will be necessary to write a function that can give us the alternate state of each transformable pair. The function in table 7 duplicates the transformable pair and replaces each binary state with its opposite, 1=0 and 0=1. Table 8 contains the output from this function.

Table 7: Enumeration of all possible male connection types (Tuples[x,3])

```
Table[Prepend[Tuples[Tuples[{1,2},2],3][[i]],Tuples[Tuples[{1,2},2],3][[i,1]]+a],{i,1,Length[Tuples[Tuples[{1,2},2],3]]}]/.
{{1+a,1+a}->{2,2},{2+a,2+a}->{1,1},{1+a,2+a}->{1,2},{2+a,1+a}->{2,1}}
```

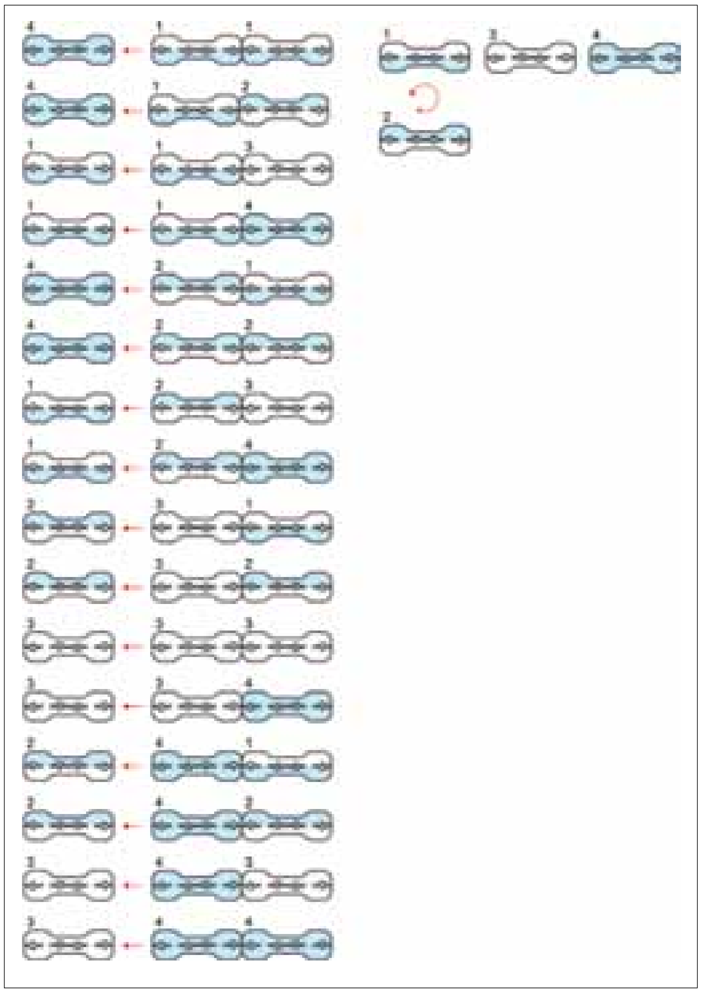


Figure 7 Visual rule table for "Loki-Blocks" Rule 17

Table 8: All possible male connections type with transformable pairs

{{2,2},{1,1},{1,1},{1,1}},	{{2,1},{2,1},{1,1},{1,1}},
{{2,2},{1,1},{1,1},{1,2}},	{{2,1},{2,1},{1,1},{1,2}},
{{2,2},{1,1},{1,1},{2,1}},	{{2,1},{2,1},{1,1},{2,1}},
{{2,2},{1,1},{1,1},{2,2}},	{{2,1},{2,1},{1,1},{2,2}},
{{2,2},{1,1},{1,2},{1,1}},	{{2,1},{2,1},{1,2},{1,1}},
{{2,2},{1,1},{1,2},{1,2}},	{{2,1},{2,1},{1,2},{1,2}},
{{2,2},{1,1},{1,2},{2,1}},	{{2,1},{2,1},{1,2},{2,1}},
{{2,2},{1,1},{1,2},{2,2}},	{{2,1},{2,1},{1,2},{2,2}},
{{2,2},{1,1},{2,1},{1,1}},	{{2,1},{2,1},{2,1},{1,1}},
{{2,2},{1,1},{2,1},{1,2}},	{{2,1},{2,1},{2,1},{1,2}},
{{2,2},{1,1},{2,1},{2,1}},	{{2,1},{2,1},{2,1},{2,1}},
{{2,2},{1,1},{2,1},{2,2}},	{{2,1},{2,1},{2,1},{2,2}},
{{2,2},{1,1},{2,2},{1,1}},	{{2,1},{2,1},{2,2},{1,1}},
{{2,2},{1,1},{2,2},{1,2}},	{{2,1},{2,1},{2,2},{1,2}},
{{2,2},{1,1},{2,2},{2,1}},	{{2,1},{2,1},{2,2},{2,1}},
{{2,2},{1,1},{2,2},{2,2}},	{{2,1},{2,1},{2,2},{2,2}},
{{1,2},{1,2},{1,1},{1,1}},	{{1,1},{2,2},{1,1},{1,1}},
{{1,2},{1,2},{1,1},{1,2}},	{{1,1},{2,2},{1,1},{1,2}},
{{1,2},{1,2},{1,1},{2,1}},	{{1,1},{2,2},{1,1},{2,1}},
{{1,2},{1,2},{1,1},{2,2}},	{{1,1},{2,2},{1,1},{2,2}},
{{1,2},{1,2},{1,2},{1,1}},	{{1,1},{2,2},{1,2},{1,1}},
{{1,2},{1,2},{1,2},{1,2}},	{{1,1},{2,2},{1,2},{1,2}},
{{1,2},{1,2},{1,2},{2,1}},	{{1,1},{2,2},{1,2},{2,1}},
{{1,2},{1,2},{1,2},{2,2}},	{{1,1},{2,2},{1,2},{2,2}},
{{1,2},{1,2},{2,1},{1,1}},	{{1,1},{2,2},{2,1},{1,1}},
{{1,2},{1,2},{2,1},{1,2}},	{{1,1},{2,2},{2,1},{1,2}},
{{1,2},{1,2},{2,1},{2,1}},	{{1,1},{2,2},{2,1},{2,1}},
{{1,2},{1,2},{2,1},{2,2}},	{{1,1},{2,2},{2,1},{2,2}},
{{1,2},{1,2},{2,2},{1,1}},	{{1,1},{2,2},{2,2},{1,1}},
{{1,2},{1,2},{2,2},{1,2}},	{{1,1},{2,2},{2,2},{1,2}},
{{1,2},{1,2},{2,2},{2,1}},	{{1,1},{2,2},{2,2},{2,1}},
{{1,2},{1,2},{2,2},{2,2}},	{{1,1},{2,2},{2,2},{2,2}}

The array variable "y" shown in table 9 enumerates all the potential adjacencies. For example, block one is adjacent to block four; see figure 6 in section 3.1 for the block numbering. The output from this calculation will be used to tell us what happens when two of these blocks are put side by side.

Table 9: Enumerating possible adjacencies

```
y= Tuples[{1,2,3,4},2]
y={{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4},{4,1},{4,2},{4,3},{4,4}}
```

The resulting array is then used with the output shown in table 8 to provide the resulting neighborhood configurations for each of the possible sixteen adjacencies described in table 9. For example, table 8 has four pairs of values per entry. These represent the left and right male connection for each block. If we take the second value in variable y, shown in table 9, {1,2}. We can use this value to figure out that when block one and block two are next to each other, and we are using rule 17, block number three will be placed on the next row. Rule 17 is the seventeenth entry in table 8 and requires the following set of male connections, {1,2},{1,2},{1,1},{1,1}. Block one and two are both {1,2}. To find the resulting connection type by placing these two blocks side by side, we take the right-most value from block one and the left-most value from block two, and we end up with a neighborhood configuration of {2,1}. Then, because all the female connections are static, we can use the following replacement rule to calculate which blocks will be used with which neighborhood configuration. Using this transformation logic, rule 17, which looks like this {1,2},{1,2},{1,1},{1,1}, becomes a sixteen-value array like this {2,1}, {2,1}, {2,1}, {2,1}, {2,1}, {2,1}, {2,1}, {2,1}, {1,1}, {1,1}, {1,1}, {1,1}, {1,1}, {1,1}, {1,1}, {1,1}.

The rule in this application looks like this: {1,1}=1,{1,2}=4,{2,1}=3,{2,2}=2. These arrays of numbers represent the two male connections that construct a neighborhood and the resulting female connection on the following row. The resulting neighborhood of {2,1} that we calculated in our above example will have block 3 on the next row above it {2,1}=3. We repeat this action for all the rules calculated in table 8 and store them in one array. Each entry in the array will look something like this: {2,1}=3, {2,1}=3, {2,1}=3, {2,1}=3, {2,1}=3, {2,1}=3, {2,1}=3, {2,1}=3, {1,1}=1, {1,1}=1, {1,1}=1, {1,1}=1, {1,1}=1, {1,1}=1, {1,1}=1, {1,1}=1. Table 10 outlines how these rules are constructed, and figure 8 shows the behavior of all 64 possible rules. This replacement rule is now an algorithm that we can use to digitally simulate the construction sequence, and to study the entire space of possibilities for configurations that might display interesting or even universal properties.

Table 10: Enumerating possible adjacencies

Male Connection Types (rule 17)	{1,2},{1,2},{1,1},{1,1}
Adjacencies	{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4},{4,1},{4,2},{4,3},{4,4}
Resulting Neighborhoods	{2,1},{2,1},{2,1},{2,1},{2,1},{2,1},{2,1},{2,1},{1,1},{1,1},{1,1},{1,1},{1,1},{1,1},{1,1},{1,1}
Female Connections Types and corresponding adjacency	{1,1}=1,{1,2}=4,{2,1}=3,{2,2}=2
Resulting Algorithm for digital construction simulation	{2,1}=3,{2,1}=3,{2,1}=3,{2,1}=3,{2,1}=3,{2,1}=3,{2,1}=3, {2,1}=3,{1,1}=1,{1,1}=1,{1,1}=1,{1,1}=1,{1,1}=1,{1,1}=1, {1,1}=1,{1,1}=1}

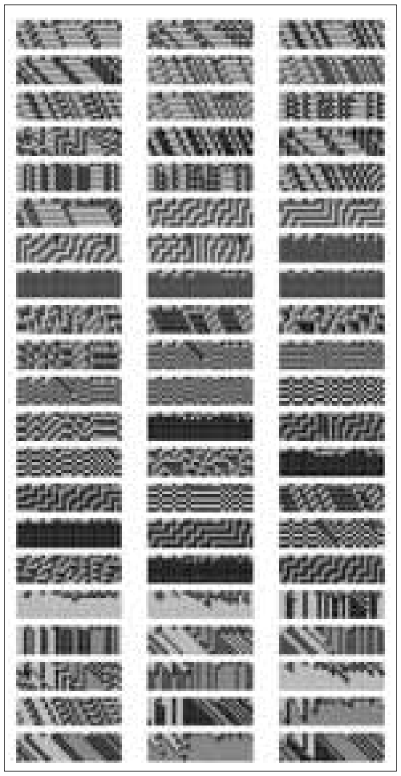


Figure 8 Visualization of all 64 possible rules for “Loki-Blocks”

We can then export these arrays into parametric modeling software to generate the following drawings of the necessary connections that will generate the desired material algorithm. Figure 7 shows one for the more interesting rules generated in this case study, and figure 8 shows the behavior for all 64 potential rules in this system.

We can also use these arrays of numerical data to produce three-dimensional visualizations of the output. Figure 9 shows a visualization of rule 55 from two different initial conditions, demonstrating how some rules can be capable of many different behaviors.

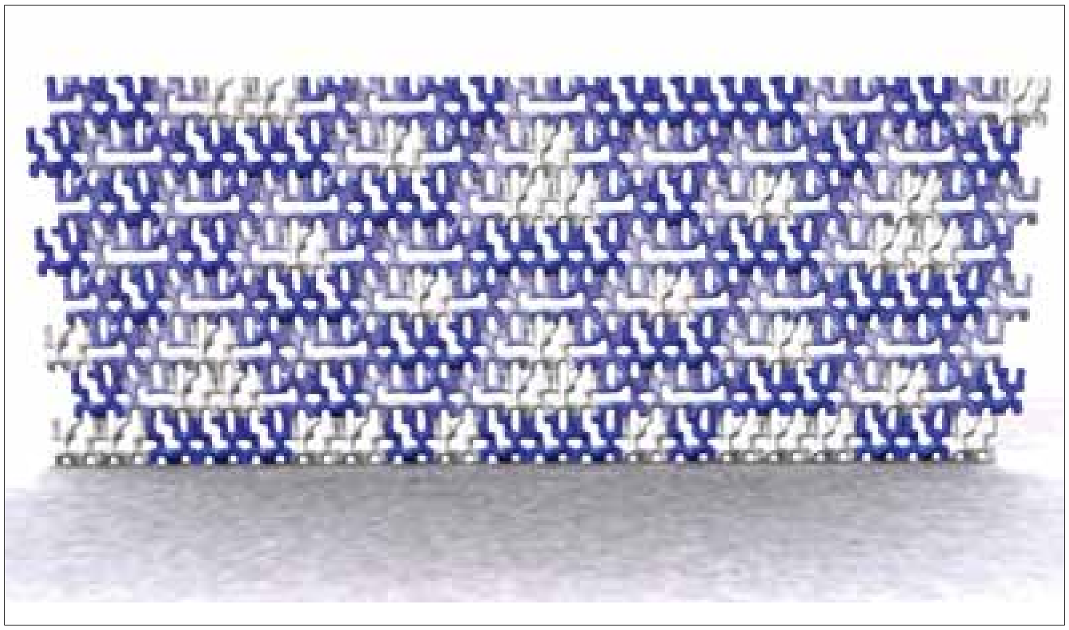


Figure 9 Visualization of rule 55 from two different random initial conditions

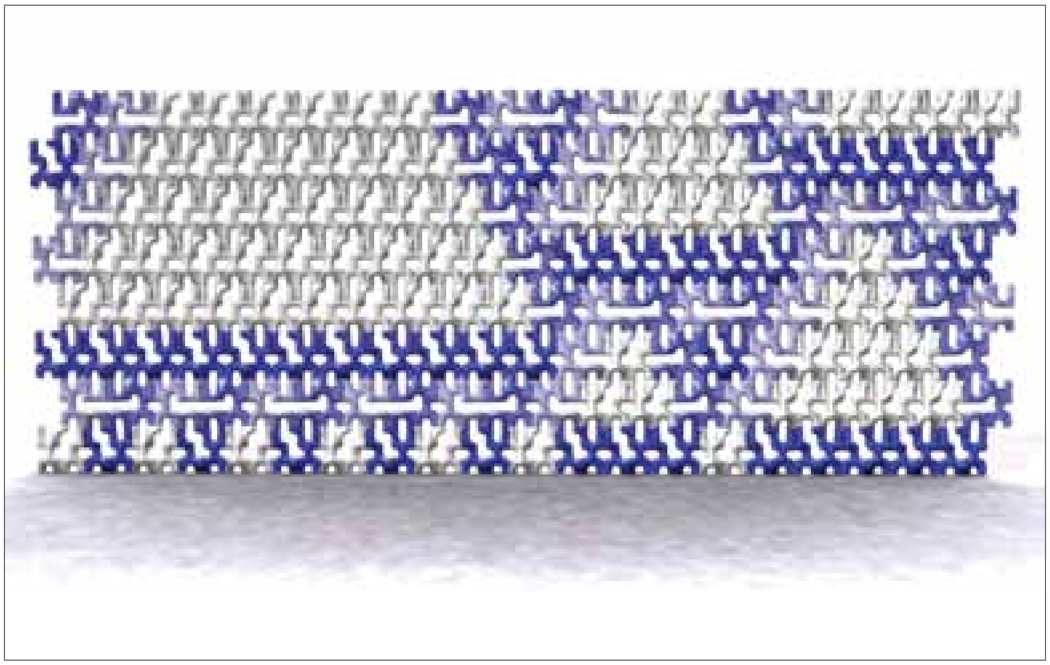


Figure 9 Visualization of rule 55 from two different random initial conditions

CONCLUSION

The above research demonstrates that it is possible to treat material assemblages as discrete computational systems. This affords the designer an extraordinary ability to develop and analyze a full range of complex behaviors in a digital environment. This research also shows us that it is possible to develop what has been termed a "tectonic computer" that is capable of transforming an assembly into a series of self-structuring accumulations, developed by a series of local decisions made by the builder. This process reduces the need for complex instructions about the assembly by applying a game-like methodology to construction.

Further developments of this research will include applications to three-dimensional structural systems, and will expand on two-dimensional systems. By transforming building components into bits of computer code, it is important that we consider what this means for the economy of building. The architecture becomes more about the product and the end user, rather then the whole composition.

Developing material systems such as these free architecture from the printed page by recording our design intent in the components of buildings themselves, and can begin to move architecture towards a more "open source" model of development. It might be possible to create high-level programming languages or a markup language for construction using the logics outlined in this paper, allowing architecture, as well as construction techniques, to evolve and be shared more freely in a digital environment.

REFERENCES

ANZALONE, PHILLIP, AND CORY CLARKE. "ARCHITECTURAL APPLICATIONS OF COMPLEX ADAPTIVE SYSTEMS." PROCEEDINGS OF THE 2003 ANNUAL CONFERENCE OF THE ASSOCIATION FOR COMPUTER AIDED DESIGN IN ARCHITECTURE (OCTOBER 27, 2003): 325–35.

GIPS, JAMES. SHAPE GRAMMARS AND THEIR USES ARTIFICIAL PERCEPTION, SHAPE GENERATION AND COMPUTER AESTHETICS. BASEL: BIRKHÄUSER, 1975.

"RADIUS - 1/2 CELLULAR AUTOMATA -." WOLFRAM DEMONSTRATIONS PROJECT. HTTP://DEMONSTR ATIONS.WOLFRAM.COM/RADIUS12 CELLULARAUTOMATA/ (ACCESSED APRIL 19, 2009).

SALEN, KATIE. RULES OF PLAY GAME DESIGN FUNDAMENTALS. CAMBRIDGE, MASS: MIT P, 2004.

TERZIDIS, KOSTAS. ALGORITHMIC ARCHITECTURE. OXFORD: ARCHITECTURAL, 2006.

WOLFRAM, STEPHEN. CELLULAR AUTOMATA AND COMPLEXITY. NEW YORK: WESTVIEW P, 2002.

WOLFRAM, STEPHEN. NEW KIND OF SCIENCE. S. L: WOLFRAM MEDIA INCORPORATED, WOLFRAM MEDIA, INCORPORATED.