RandScape: Complex Images from Simple Algorithms

Jeremy Smith
The Possomery
301 NE Byron Place
Corvallis, OR 97330-6233
jeremy@peak.org

Abstract This letter presents a range of complex patterns, some that seem eerily lifelike, that were produced from some simple algorithms. Some observations are offered that suggest the algorithms are modeling some natural processes.

Keywords
RandScape, random landscapes, self-organizing patterns, cellular automata

1 Introduction

In this letter, I present a number of images that were produced while exploring some data sorting techniques. The complexity and range of the images was unexpected, especially from such simple algorithms, and some of the images have intriguingly lifelike and natural qualities. Rather like a biologist trying to make sense of some field data, I offer some observations on these qualities in the discussion, and how they may arise from certain details in the algorithms. However, no definitive analysis is attempted. I first describe how the algorithms came about, and their inner workings.

The idea behind the data sorting technique was to obtain two- or three-dimensional data and have it arrange itself into a smooth landscape of hills and valleys, and then render it visually as such. The preliminary study used a stream of random data, and the results were plotted as a continuous array of cells, each colored a shade of gray to represent its value. The purpose was to use any successful technique to examine real data, and the study of fitness landscapes was the original inspiration [5].

2 The Algorithms

The images are representations of random data that has been arranged using a specific class of algorithms. Each piece of data has a value between 0.0 and 1.0 and is represented by a pixel with a corresponding gray color scaling from black to white. The width of the image is arbitrary, and the data is laid down, starting at the bottom, one row of pixels at a time.

Without any processing, the picture that emerges is an array of randomly shaded gray dots. The bottom of Figures 1 and 2 shows this, Figure 2 being a magnified region of Figure 1 (as indicated by the white rectangle). Once processing is invoked, however, the next row of random data is rearranged based on the values of the previous row. As subsequent rows are processed, patterns emerge, as seen in the main part of Figure 1, and in the top half of the inset, Figure 2.

Five algorithms are described below, and it is these that do the rearranging. There are also three further techniques described, but these are hybrids or composites of the five primary algorithms.

Some general calibrations are available. A new row of data is initially random, but can be sorted before being distributed on the previous row. The difference can be anything from negligible to very significant, depending on the arranging algorithm in

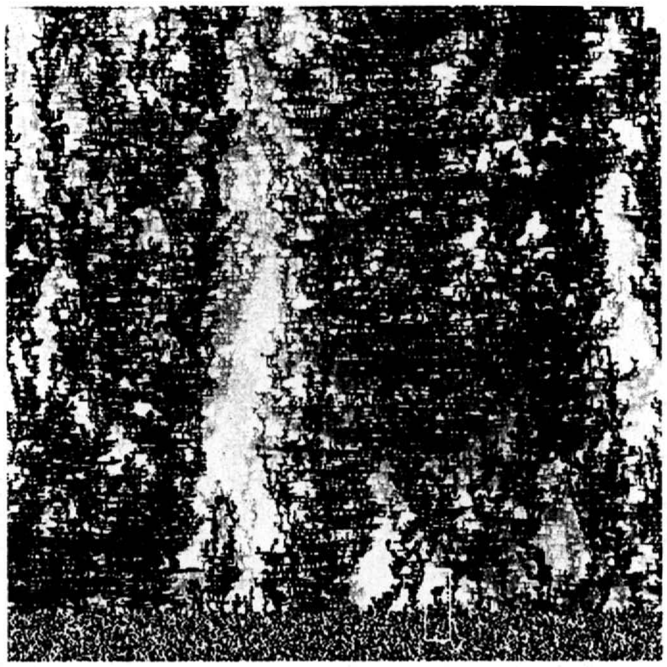

Figure 1. Next available slot (sorted).

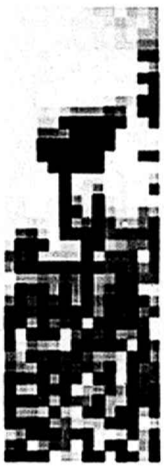

Figure 2. Magnified rectangle from Figure 1.

use. Data can be arranged based upon a value in the previous row, or it can be based on an average of a number of values in the previous row (lateral averaging), or on an average of a number of values from a number of previous rows (vertical averaging), or a combination. Combining the algorithms and calibrations in various ways occasionally produce random or featureless pictures, but many produce a surprising array of patterns, as displayed here.

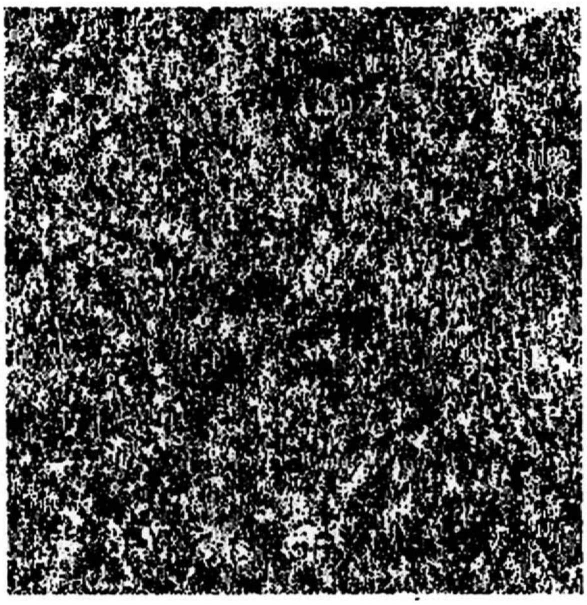

Figure 3. Next available slot (random).

2.1 Next-Available-Slot Algorithm

This algorithm places each new block over the block in the previous row with the closest matching shade of gray. However, if the closest shade is already taken, then the block is moved to the next available space to the left or right, arbitrarily, and dropped there. This is repeated with each block, until the whole new row is positioned. Subsequent rows are processed similarly. The algorithm's name comes from the positioning procedure, and it's possible that the majority of blocks are positioned this way.

If the new row is sorted prior to the blocks' being positioned, the resulting pattern looks like Figure 1. If the new row is left unsorted during positioning, then the pattern looks like Figure 3. (Figure 15, at the end of the paper, was produced by an early experimental version of this algorithm.)

2.2 Perfect Algorithm

In this algorithm, the darkest block from the new row is positioned against the closest shade from the previous row. The next-darkest block is positioned against the closest one of the remaining blocks, and so on, until all the blocks from the new row are positioned.

It seems perfect (hence the algorithm's name), but consider three old blocks very close in shade. The new one goes to the darkest, but if the next new one is closer to the third old one, it leaves the second one unmatched. When it finally gets a match, it will in general be much lighter—certainly lighter than the two on each side. Hence the resulting pattern, shown in Figures 4 and 12, and (with lateral averaging) in Figures 13 and 14.

2.3 Smoothing Algorithm

The smoothing algorithm puts the darkest new block against the darkest old block, the next-darkest new block against the next-darkest old block, and so on, all the way to putting the lightest new block against the lightest old block. This is done without

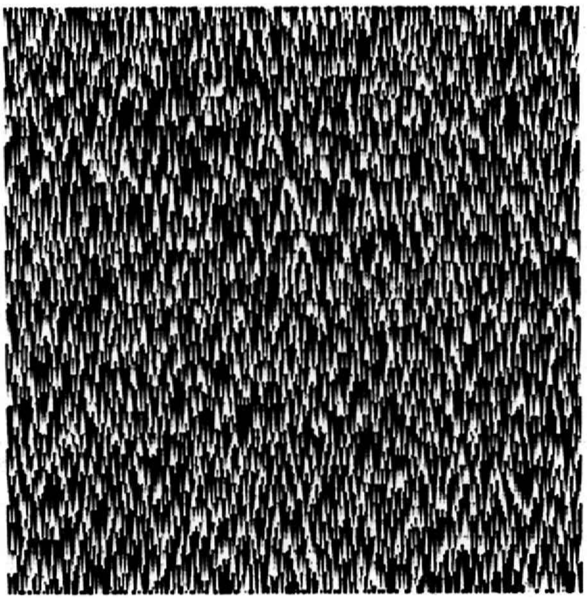
Figure 4. Perfect (sorted).

regard to the actual comparative shades of the blocks (as in the above two algorithms) but is arranged by shade order. The result is a continuous set of stripes.

However, with a little lateral averaging it produces what looks like the folds of a curtain, which, with progression, slowly smooth out (hence the algorithm's name). The rate of smoothing increases with the number of values that are averaged (3 in Figure 5; 11 in Figure 6).

2.4 Peaks Algorithm

In this algorithm, the highest values from the old row (about half of them) are designated as peaks. Blocks from the new row are placed on random peaks, and, as in the next-available-slot algorithm, if a peak already has a block, the new one moves sideways until an available slot is found and is dropped there.

As might be expected, the pattern (Figure 7) has a resemblance to that of the next-available-slot algorithm.

2.5 Settle and Seek Algorithms

The settle algorithm looks at blocks to the left and right of the previous row, and if there is a block closer in value to one side than the current position, then the block and its adjacent block are swapped. If the current position is closest in value, then no swapping is done.

This process is repeated for each block in the new row. Note that this process is dynamic in that, if a block has been swapped to the left (and processing is going leftward), it is then processed again, and could be shunted along a number of times.

The lateral averaging setting indicates the scope of blocks to each side to be compared with the current block. If lateral averaging is 7, then three blocks to each side, as well as the current position, are compared to find the closest. In addition, the program will process the new row as described that many times (three), which allows the blocks to get even more shunted along to values that are closer (it is in this sense that they "settle").

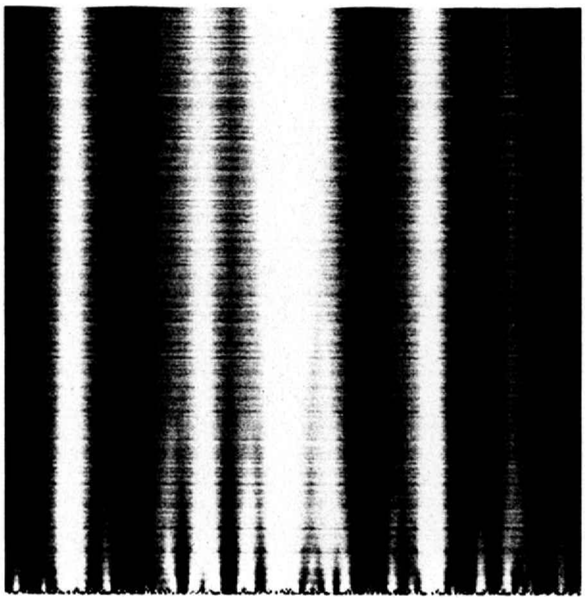

Figure 5. Smoothing (sorted, lateral 3).

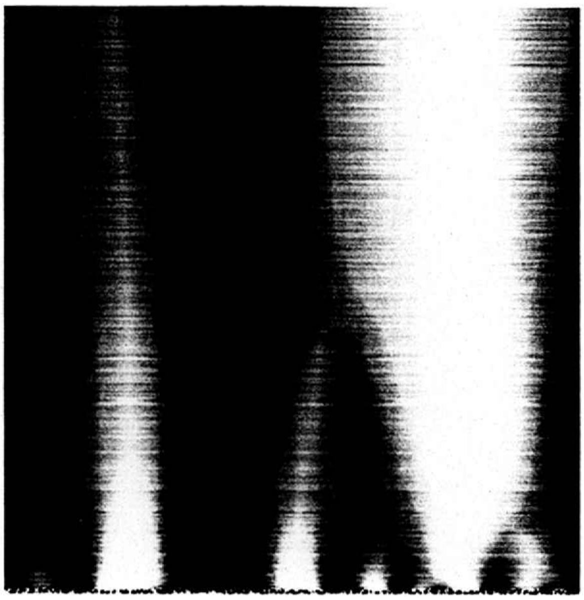

Figure 6. Smoothing (sorted, lateral 11).

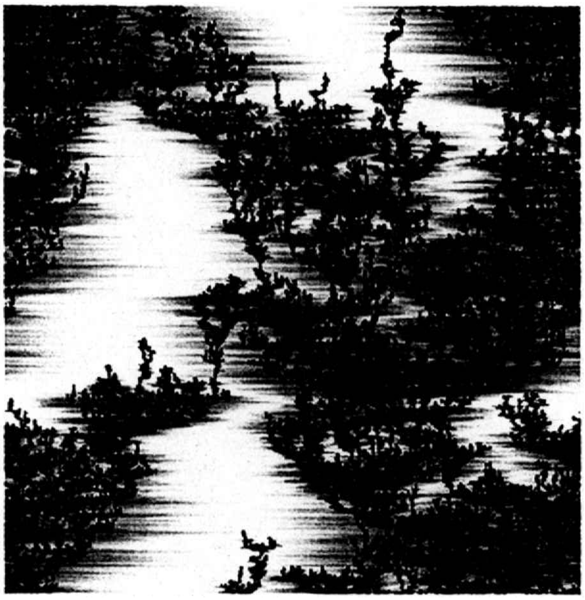

Figure 7. Peaks (sorted).

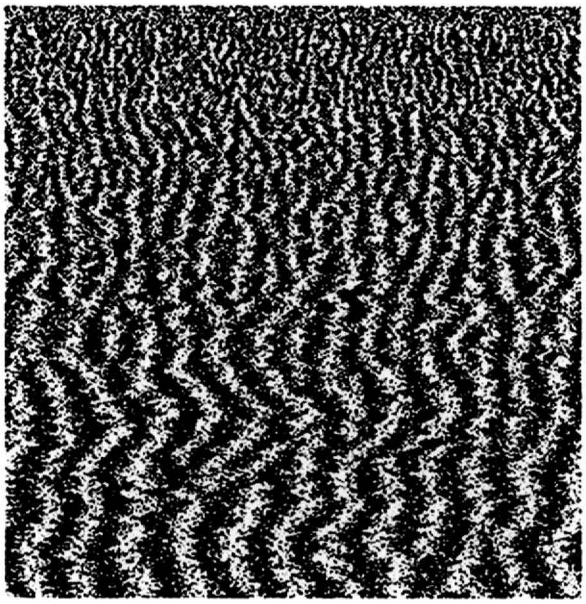

Figure 8. Seek (random, lateral 13-3).

The seek algorithm is similar to the settle algorithm except that, rather than search to the side for the closest block, the average of the left and right is calculated, and the block is swapped to the side that is closest to it, or not swapped if, as before, the current position is closest.

The settle and seek algorithms produce patterns that are virtually indistinguishable (Figure 8).

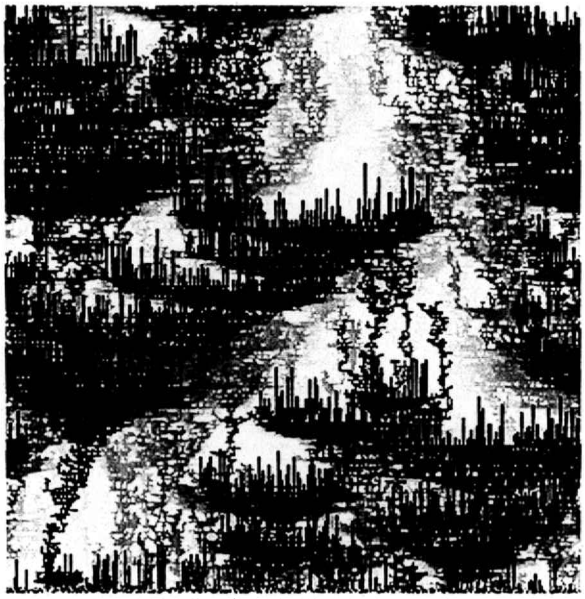

Figure 9. Mix (sorted).

2.6 Mixed Algorithms

This and the following two algorithms are contrived relative to the above algorithms, in that tricks are used to create special effects, rather than allowing the data to joggle itself based on its values and the values around it.

Any of the above algorithms could be mixed in various ways. One mix that produces interesting patterns is to arrange the first third of a new row of blocks by the perfect algorithm, and the remainder by the next-available-slot algorithm (Figure 9).

2.7 Icing Algorithm

The icing algorithm is basically the smoothing algorithm, but with two twists: the lateral value undulates back and forth (e.g., from 7 to 21 and back), and when averaging is done, the values are inverted before they are compared. This last calibration surprisingly does not create a jumbled mess, but, at high levels of averaging, creates repeating gridlike forms (Figure 10), which in some runs looked like drawn icing (frosting) on a cake.

2.8 Second-Layer Algorithm

The second-layer algorithm is basically the smoothing algorithm, but with two new twists: the vertical value is used as the basis for an undulating cycle. In the up cycle, a row from a second layer is increasingly used in place of the previous row; in the down cycle, decreasingly used. The second layer is actually a row from the existing picture but halfway down (Figure 11).

3 Calibrations

3.1 Sorting

Most of the descriptions of the algorithms describe positioning blocks of the new row from darkest to lightest, which implies that the blocks are sorted before being arranged.

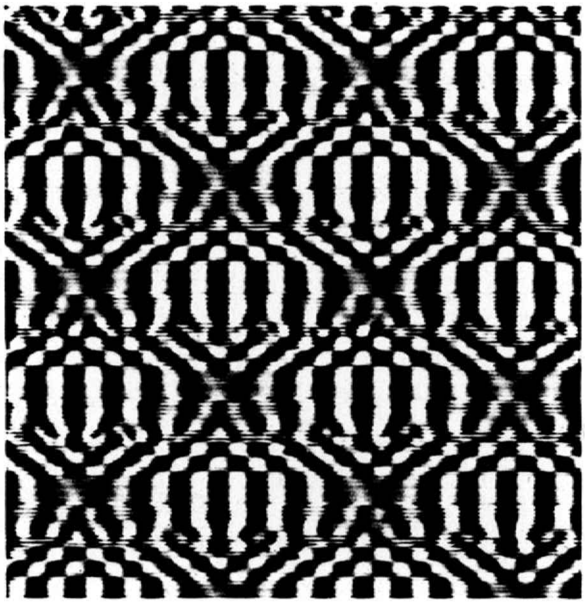

Figure 10. Icing (sorted, lateral 9, vertical 5).

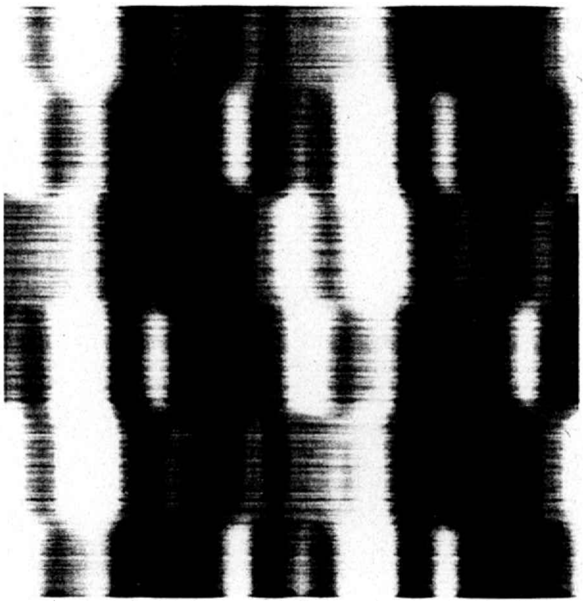

Figure 11. Second layer (sorted, lateral 9, vertical 7).

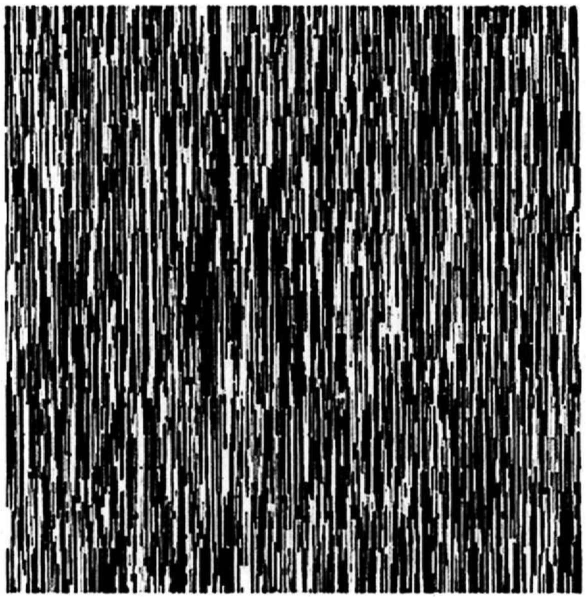

Figure 12. Perfect (random).

If they are left unsorted, which means random as initially generated, then two of the algorithms, peaks and smoothing, result in random patterns, but the others sometimes produce interesting effects. The figure captions indicate whether the new rows are presorted or left random.

3.2 Lateral Average

Instead of arranging each new block by comparing its shade against that of each old block, each new block can be compared against an average of, say, three or five blocks from the old row. (Even numbers cause the pictures to slant diagonally, since it is skewed.) Particularly interesting effects arise using this calibration on the perfect algorithm. Compare Figures 3 and 12 with Figures 13 and 14. Applying it to the smoothing algorithm demonstrates that it effectively speeds up any cumulative lateral effect. For instance, the larger the lateral averaging, the shorter are the domelike constructs (Figures 5 and 6).

3.3 Vertical Average

A new row can be arranged against the average of a number of previous rows rather than just one. Where this feature has an effect, it mostly seems to just fragment a normal pattern, but can occasionally produce some interesting results or enhance existing ones (Figures 10 and 11).

4 Discussion

Some of the pictures have lifelike qualities, like seaweed (Figure 1), sand dune weeds (Figure 7), or even rushes in a wetland (Figure 9). Others have inorganic qualities, like clouds (Figure 14), cascading water (Figure 13), or ripples in mud or sand (Figure 8). And some are in between, like bark (Figure 3). Are the algorithms embodying natural or lifelike processes?

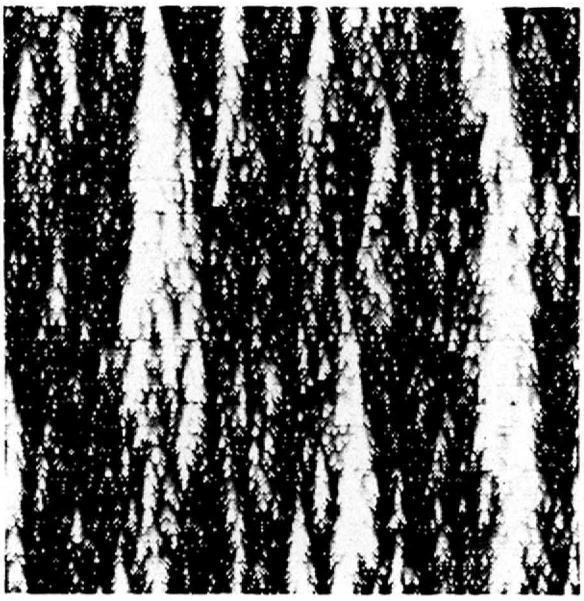

Figure 13. Perfect (sorted, lateral 3).

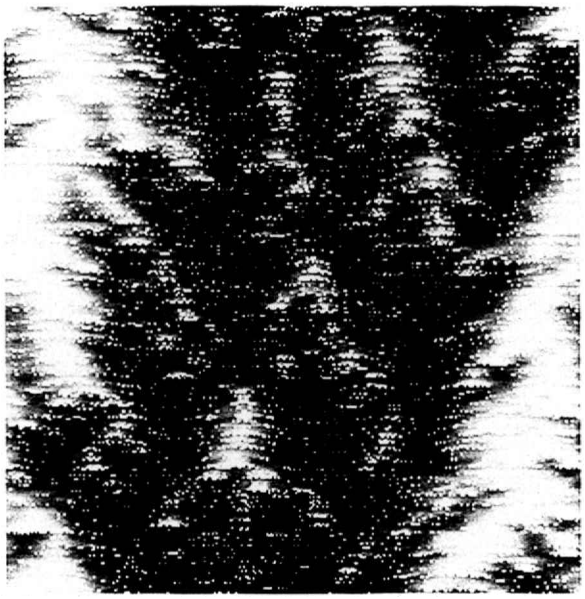

Figure 14. Perfect (sorted, lateral 15).

The basic algorithm operates like a continuous one-dimensional cellular automaton [3, 4, 7, 8]. It's continuous because each block has any value between 0.0 and 1.0, rather than discrete (0 or 1 only). It's one-dimensional because it's processed a row at a time, with each new value being influenced by a single value from the previous row, and not affected by values around it (two and more dimensions). However, because of lateral and vertical averaging, it might technically be considered slightly-more-than-one-dimensional. Also, unlike cellular automata, new values coming in are already set, and aren't set by previous values, just influenced by them.

The settle and seek algorithms are easily identified as modeling a natural process. The blocks in the new row gravitate towards ones of a similar shade, darker towards darker, lighter towards lighter, but they are constrained to move, one position at a time, only as far as the lateral averaging setting allows. This might compare to a smooth field of sand being blown on by the wind, where lighter grains are more easily picked up and carried further, and heavier ones more likely to roll back down the ensuing slopes. In fact, the settle and seek samples (Figure 8) bear a striking resemblance to the self-organized pattern of ripples in sand formed by the action of wind or water. The other pictures that resemble natural processes are less obviously so identifiable from the algorithms, but are still essentially particles aggregating in various ways, which is partially the process behind many natural systems.

All the lifelike pictures have a common element—dendritic structures. These are clearly due to a process like diffusion-limited aggregation (DLA) [6]. Ball [1] points out that real mineral dendrites, crystals that look just like DLA models, are easily mistaken for fossil ferns. On the other hand, many living systems, such as bacteria growing in a Petri dish, also exhibit dendritic structures almost identical to the DLA models. Ball says that whether the resemblance is anything more than coincidental is still an open question.

Finally, we must ask why patterns form at all. A recent article [2] reports on some physicists (Banavar et al.) who model populations of agents with no attributes, and find that if each agent goes off on a random walk, a population diffuses according to the laws of physics, like for instance a drop of dye diffusing in a bucket of water. However, if there is any significant peer pressure, where agents affect each other's behavior, then patterns emerge (aggregation) in the distribution (arrangement) of the agents that constitute the population. The algorithms described in this note rearrange data based upon values of surrounded data, which is essentially modeling peer pressure, a powerful property behind the emergence of patterns.

5 Conclusion

The algorithms, though simple, nonetheless perhaps model some quite complex natural processes, and the images reflect this. Wolfram delves deeply into the intrinsic complexity of some simple algorithms [8], and is a keen proponent of algorithms as an exploratory or analytical tool to augment other approaches.

The Java program used to generate these pictures can be found here:
www.peak.org/~jeremy/random/RandScape.jar
The jar file contains instructions on use in the readme.txt file, as well as source and program. A simple invocation would be:

```
java -jar RandScape.jar
```

References

1. Ball, P. (1999). *The self-made tapestry: Pattern formation in nature*. New York: Oxford University Press.

2. Cobb, K. (2002). Peer pressure in numbers. *Science News, 162*, 116.

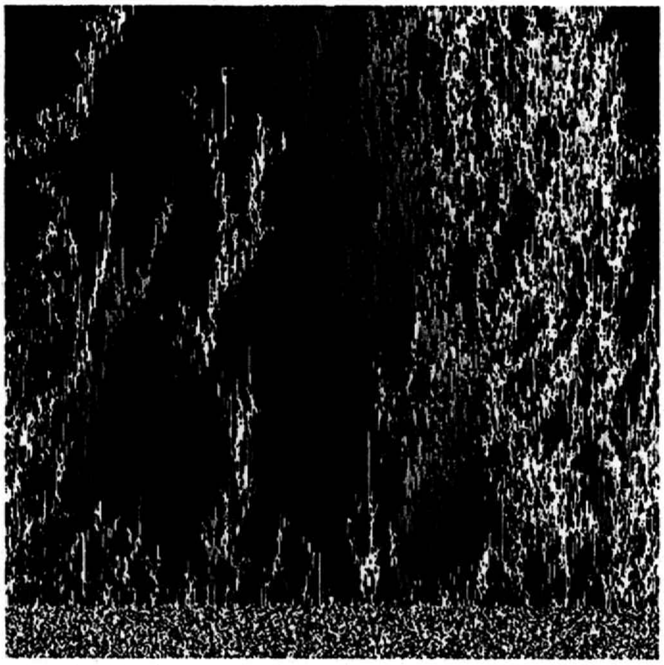

Figure 15. Next available slot (sorted, inverted, broken).

3. Langton, C. (1984). Self-reproduction in cellular automata. *Physica, 10D*, 135–144.

4. Langton, C. (1986). Studying artificial life with cellular automata. *Physica, 22D*, 120–149.

5. Pohlheim, H. (1999). Visualization of evolutionary algorithms—Set of standard techniques and multidimensional visualization. In W. Banzhaf et al. (Eds.), *Proceedings of the 1999 Genetic and Evolutionary Computation Conference* (pp. 533–540). San Francisco: Kaufmann.

6. Witten, T., & Sander, L. (1983). Diffusion-limited aggregation. *Physical Review B, 27*, 5686–5697.

7. Wolfram, S. (1983). Statistical mechanics of cellular automata. *Reviews of Modern Physics, 55*, 601–644.

8. Wolfram, S. (2002). *A new kind of science*. Champaign, IL: Wolfram Media.