# Comparing Model Synthesis and Wave Function Collapse

Paul Merrell
July 28, 2021

## 1 Summary

This document directly compares Model Synthesis to Wave Function Collapse ( WFC ) by analyzing the algorithms and through a series of experiments. Model synthesis was created by Merrell in 2007 [1]. WFC was created by Gumin in 2016. There are a few differences between them, but they are small. It is more appropriate to classify them as two versions of the same algorithm rather than two different algorithms. We are really comparing the original version of model synthesis to the WFC version of model synthesis. But for simplicity, we just call these model synthesis and WFC.

The results of this analysis show two things: (1) model synthesis and WFC produce similar results for small outputs. The results look very similar and the computation time is also similar. (2) WFC has great difficulty generating some large textures and models. It repeatedly fails. Model synthesis can generate in a few seconds what WFC fails to generate in over 20 minutes.

## 2 Differences

The two methods are very similar. Both contain a list of possible labels can be assigned to each cell in the grid. (In WFC, this is called the wave function.) In both cases, we start with every possible label in every cell. In both cases, we narrow down the possible labels by choosing a cell within the grid, picking one of the possible labels, and eliminating the rest. In both cases, we propagate this choice and eliminate labels that no longer agree with it. In both cases, we continue until all grid cells have one label.

There are two differences between them:

### 2.1 Order of Picking Cells

The first difference is in the step where we choose a cell and pick a label. The cells are chosen in a different order. Model synthesis sweeps through the grid in scanline order. WFC chooses the lowest entropy cell. Figures 2 - 10 show images generated with and without this change. This has little to no effect on the quality of the results.

The order can also affect the speed of the algorithm. Both methods can fail and the order may affect the failure rate. For small textures and model, the order has little impact. The failure rate is low in either case. But for some large textures, picking the lowest entropy cell greatly increases the failure rate. This is the main reason why WFC fails so much more in Table 1. The difference is huge for some inputs. It does not happen for every input, but model synthesis can generate in seconds what WFC fails in generate in over 20 minutes. I do not know how long WFC would take because it never finishes.

### 2.2 Modifying in Blocks

The second difference is that WFC does not implement an important part of the model synthesis algorithm. This is the part where we modify the model in smaller pieces instead of all at once. This is a complicated issue that is discussed in detail in my dissertation [Merrell 2009] (pages 41 - 65). In Theorem 3.3.4, I show that for some inputs this is an NP-hard problem. For some inputs, it is not NP-hard and the algorithm works flawlessly (Theorem 3.3 .6). So this issue only applies to some inputs. For those inputs, the problem gets exponentially more difficult as the size of the output increases. This means that if you try to generate the whole model all at once, the chance of success decreases rapidly once the output reaches a certain size. Model synthesis solves this problem by breaking the problem into smaller pieces. Because WFC does not implement this part it cannot generate large outputs in these cases. This is called modifying in blocks.

Modifying in blocks is essential for generate large complex models like the one in Figure 1. WFC cannot generate large models like that in a reasonable amount of time.

WFC has mostly focused on 2D textures rather than 3D model. (The original WFC code only works on 2D inputs, but

[1]To be completely accurate, model synthesis was first created in 2005, but not shared publicly until a paper published in 2007 [Merrell 2007].

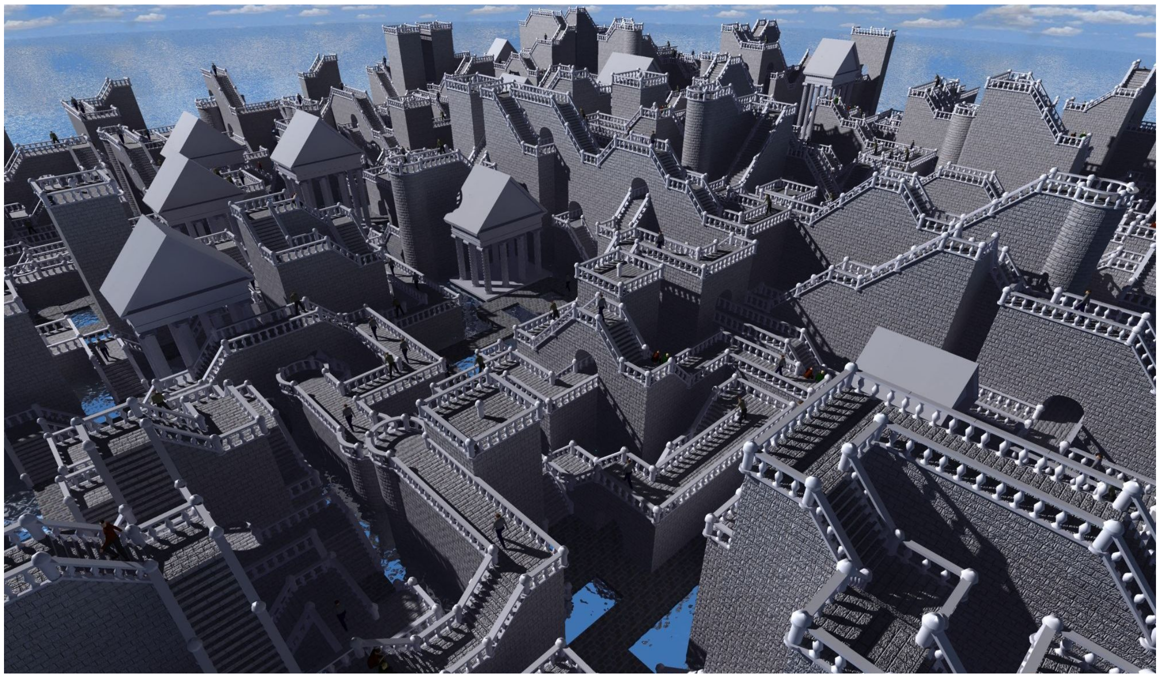


Figure 1: 3D Model generated by model synthesis.

other projects have extended it into 3D). 2 D inputs are less challenging. The tiles are not as interconnected and the outputs are smaller. The large textures in Table 1 are not especially large compared to many of the 3D models. The examples in [Merrell 2009] ranged from 9K to 72K voxels. Modifying in blocks is more important for generating 3D models because they are more difficult.

As Table 1 shows model synthesis is better at generating large textures. This is for two reasons: because of the order it picks the cells and because it modifies in blocks. Both contribute to its success in Table 1. The order of the cells is more important. Modifying in blocks is not even necessary for most of the examples.

## 3 Propagation AC-3 and AC-4

Both methods formulate synthesis as a constraint satisfaction problem (CSP) which is solved using an Arc Consistency algorithm. The AC-3 algorithm [Mackworth 1977] and the AC-4 algorithm [Mohr and Henderson 1986] both produce the same results meaning they both remove the same set of labels. But the AC-4 algorithm is faster especially when there are many labels. [Wallace 1993] says that AC-3 is faster for most CSPs because AC-4 requires a preprocessing step in which the support for each neighbor is calculated. However, model synthesis is structured in a way that makes this part easy. Every pair of neighbors is the same. AC-4 is faster.

## 4 Timing

Tables 2-3 show the timings for the two methods on my machine. The results show that model synthesis is not as fast as WFC. By far the slowest part of both methods is the propagation step. Both methods use the same AC-4 algorithm and the same set of labels and constraints. It is difficult to tell what is causing the difference. The different cell order (lowest entropy vs scanline) has little effect. The methods are implemented in different programming languages. Subtle difference in how the algorithm is implemented can affect memory access speeds and other factors.

## 5 Overlapping Tiles

Model synthesis generated 2D textures using image tiles from the beginning. But these tiles did not overlap each other. Gumin introduced the idea of using overlapping image tiles [Gumin 2016]. This has several benefits. The tiles can easily be computed from an image. And when you use overlapping tiles, the result is more tightly constraint to look like the input. The model synthesis code now support overlapping tiles. This requires a preprocessing step, but no other changes to the algorithm. The same algorithm works for overlapping tiles, non-overlapping tiles, and 3D model pieces.

It is also important to understand how model synthesis / WFC compares to other texture synthesis algorithms. This is discussed in greater detail in [Merrell 2009], but I will summarize the main points here. Many of the textures shown in Figures 2 - 10 could also be generated by other state-of-the -art texture synthesis algorithms. The main advantage that model synthesis / WFC has is it is better at forming closed loops. On the other hand, if you used a photograph as an input texture, many texture synthesis algorithms would work fine, but model synthesis / WFC would not. This is because they require the input image to be self-similar. In most photographs, the image patches all have slightly different colors. Model synthesis and WFC require pixel art style images that only use a few colors.

## 6 Extensions

The WFC code has been adapted and extended to many additional use cases and applications. I will not attempt to list them here. Note that these extensions could also be applied to model synthesis.

## 7 Conclusion

Model synthesis and WFC use nearly the same algorithm and produce similar results. WFC picks cells in a different order and does not modify in blocks. This causes the algorithm to fail more on some large models. Gumin introduced the idea of using overlapping tiles as an input which has several benefits.

## References

GUMIN, M., 2016. Wave function collapse, https://github.com/mxgmn/wavefunctioncollapse.

MACKWORTH, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence 8*, 1, 99–118.

MERRELL, P. 2007. Example-based model synthesis. In *I3D '07: Symposium on Interactive 3D graphics and games*, ACM Press, 105–112.

MERRELL, P. 2009. *Model Synthesis*. PhD thesis, University of North Carolina at Chapel Hill.

MOHR, R., AND HENDERSON, T. C. 1986. Arc and path consistency revisited. *Artificial Intelligence 28*, 2, 225–233.

WALLACE, R. J. 1993. Why ac-3 is almost always better than ac-4 for establishing arc consistency in csps. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, IJCAI'93, 239–245.

| Name | width x height | Model Syn Total (s) | WFC Total (s) | Model Syn Synthesis (s) | WFC Synthesis (s) | WFC Success Rate | WFC Trials |
|---|---|---|---|---|---|---|---|
| Summer | 100 x 100 | 3.741 | 116.3 | 0.969 | 114.5 | 0.2 % | 1,000 |
| Summer | 200 x 200 | 25.144 | – | 13.869 | $>$ 1,354 | 0 % | 1,000 |
| Castle | 100 x 100 | 0.628 | – | 0.557 | $>$ 1,365 | 0 % | 10,000 |
| Castle | 200 x 200 | 4.683 | – | 4.399 | $>$ 1,365 | – | – |
| Knot Dense | 100 x 100 | 0.310 | 16.04 | 0.156 | 15.94 | 2.7% | 1,000 |
| Knot Dense | 200 x 200 | 1.758 | – | 1.173 | $>$ 1,877 | 0 % | 1,000 |
| Knot TE | 100 x 100 | 0.210 | – | 0.098 | $>$ 882 | 0 % | 10,000 |
| Knot TE | 200 x 200 | 1.086 | – | 0.687 | $>$ 882 | – | – |
| Knot T | 100 x 100 | 0.180 | 2.60 | 0.071 | 2.56 | 10.4 % | 1,000 |
| Knot T | 200 x 200 | 0.909 | – | 0.532 | $>$ 1,253 | 0 % | 1,000 |
| Knot CE | 100 x 100 | 0.868 | 9.80 | 0.772 | 9.76 | 1.8 % | 1,000 |
| Knot CE | 200 x 200 | 1.055 | – | 0.673 | $>$ 787 | 0 % | 1,000 |
| Rooms | 100 x 100 | 0.557 | 1.30 | 0.544 | 1.29 | 45.9 % | 1,000 |
| Rooms | 200 x 200 | 4.136 | 200.8 | 4.049 | 200.7 | 1.8 % | 1,000 |
| Red Dot | 200 x 200 | 8.205 | – | 8.194 | $>$ 7,061 | 0 % | 10,000 |
| Shew1 | 200 x 200 | 15.603 | – | 15.593 | $>$ 1,757 | 0 % | 1,000 |
| Cat | 400 x 400 | 279.9 | – | 279.8 | $>$ 14,802 | 0 % | 1,000 |

Table 1: This shows the results for large textures. All times are in seconds. The total time is the average time to successfully compute one texture of size width x height. The synthesis time excludes the time for reading the input and encoding the image files. The success rate gives the fraction of attempts in which WFC was successful. Model synthesis is almost always successful. Whenever the failure rate is high, you can decrease the size of the block you are modifying and then it will succeed. Several entries for WFC are blank because no solution was found after waiting a long time. In these cases, the WFC Synthesis column shows the time that was spent. This is a lower bound as we do not know how long it would take. WFC trials is the number of attempts that were made.

| | Model Syn (s) | WFC (s) |
|---|---|---|
| Parse Input | 0.067 | 0.366 |
| Synthesis | 0.456 | 0.321 |
| Generate Output | 0.985 | 0.432 |
| Total | 1.508 | 1.119 |

Table 2: The time it took in seconds to generate all of the 36 small textures shown in Figure 3 - 6. The timings are separated into three parts. First, parsing the input files and settings up the input. Second, actually running the synthesis algorithm. Third, using the result to encode all the pixels into an image file.

| | Model Syn (s) | WFC (s) |
|---|---|---|
| Parse Input | 3.545 | 0.386 |
| Synthesis | 32.161 | 7.699 |
| Generate Output | 0.081 | 0.104 |
| Total | 35.787 | 8.186 |

Table 3: The time it took in seconds to generate all of the 54 small textures shown in Figure 2. The timings are divided into the same parts described in Table 2. These use overlapping tiles.

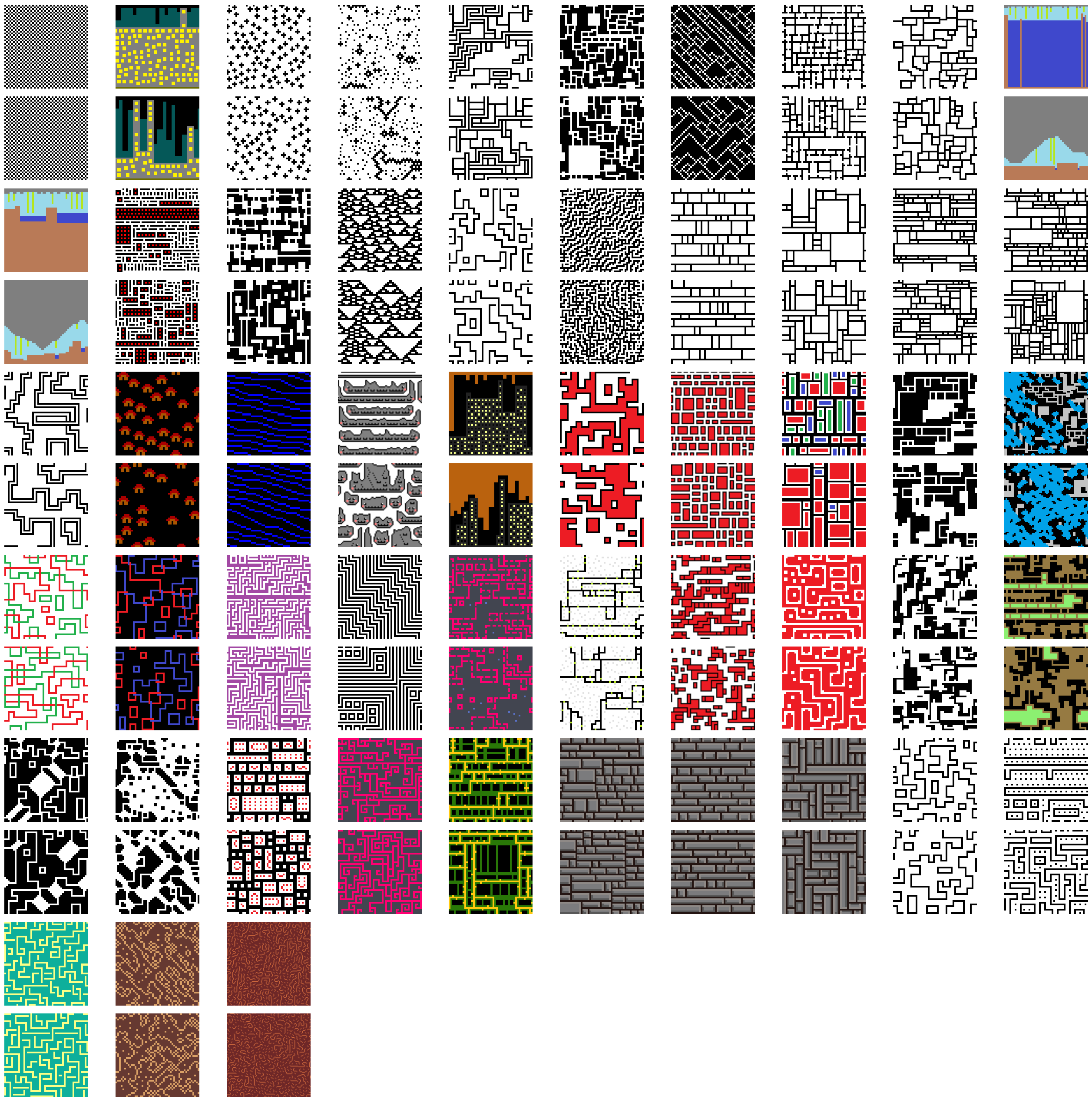


Figure 2: A comparison of images generated using model synthesis and WFC. The top row in each pair was generated by model synthesis. The bottom row by WFC. These all use overlapping image tiles. The results look similar.

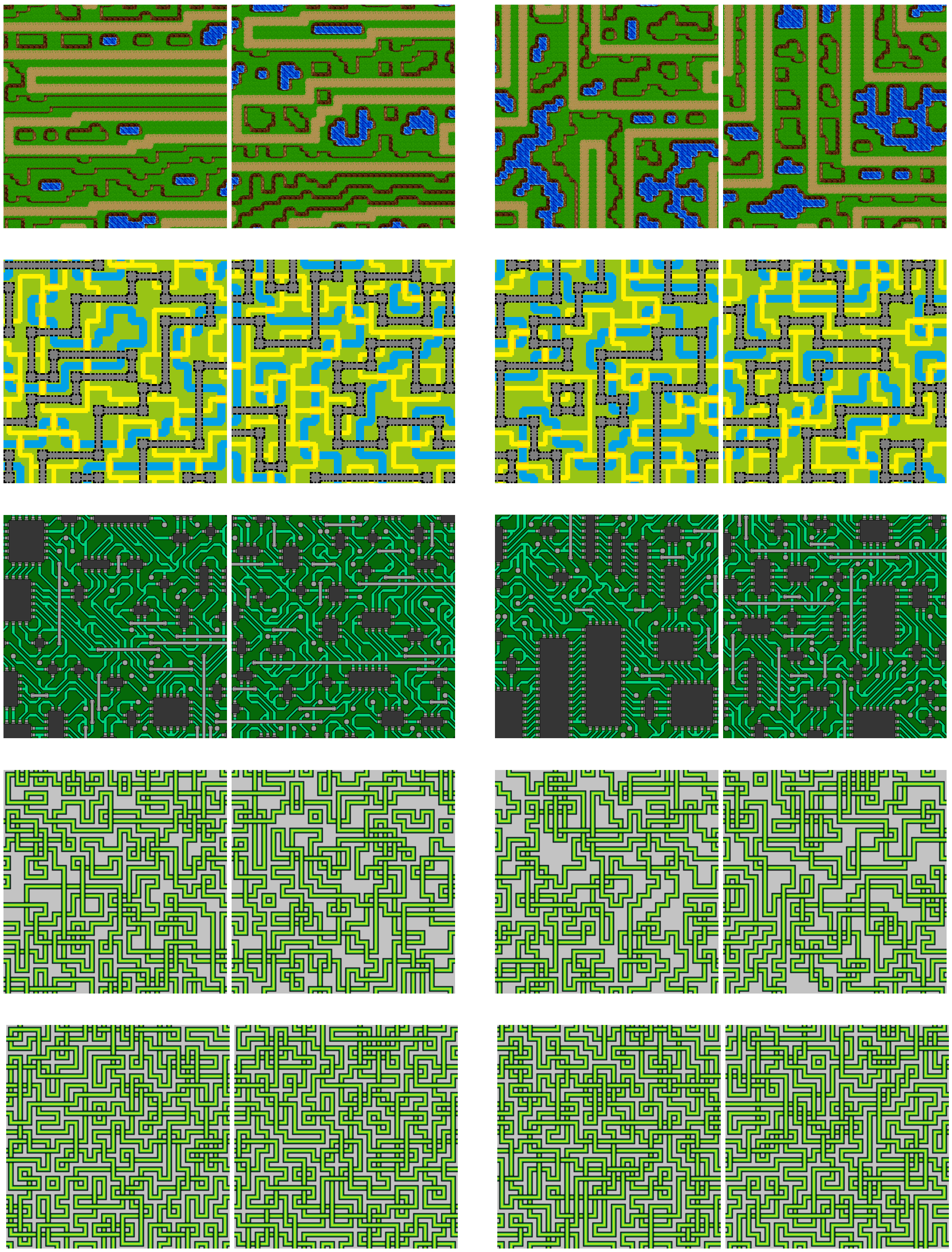


Figure 3: Textures on the left were generated by model synthesis. Textures on right by WFC. The results look similar.

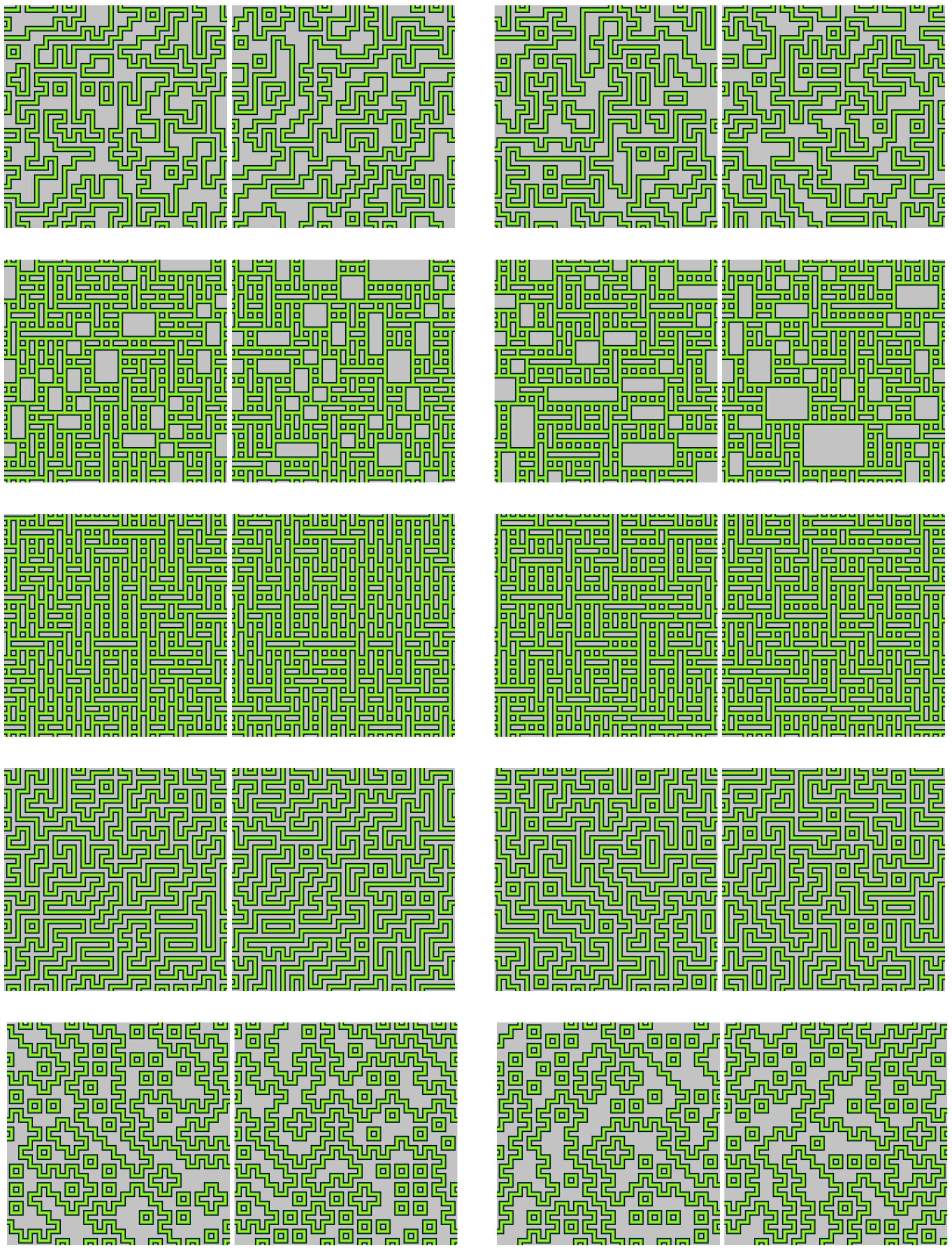


Figure 4: Textures on the left were generated by model synthesis. Textures on right by WFC. The results look similar.

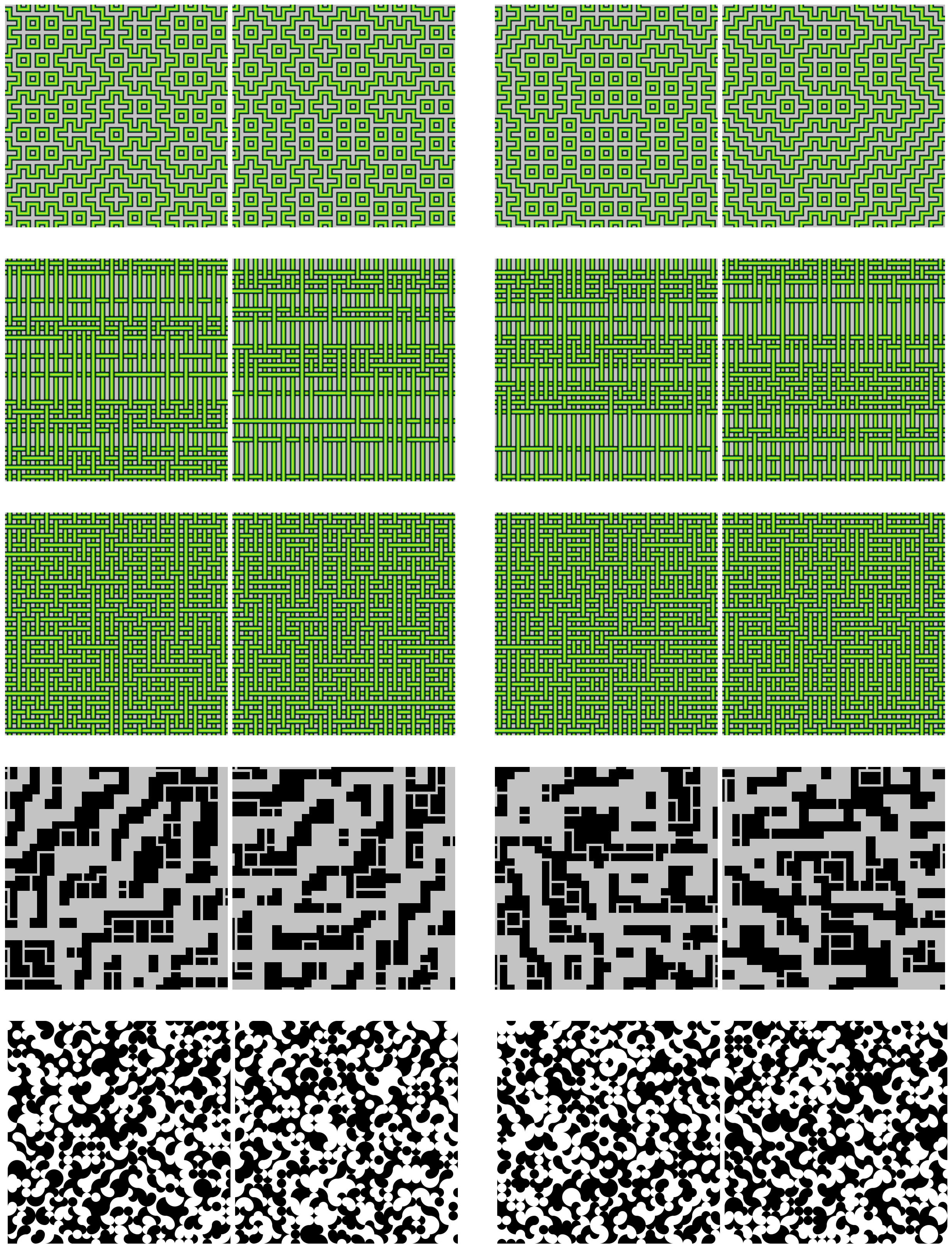


Figure 5: Textures on the left were generated by model synthesis. Textures on right by WFC. The results look similar.

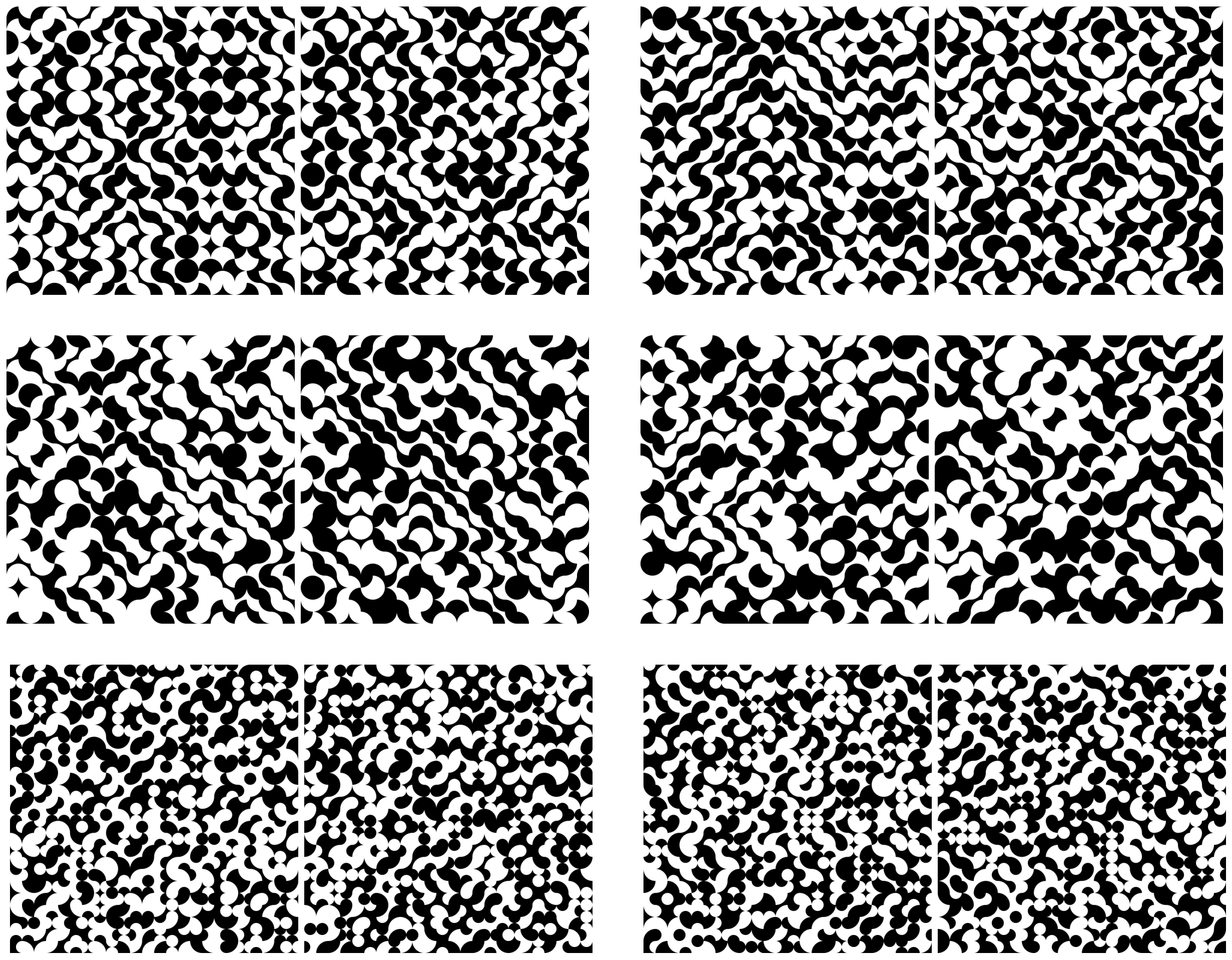
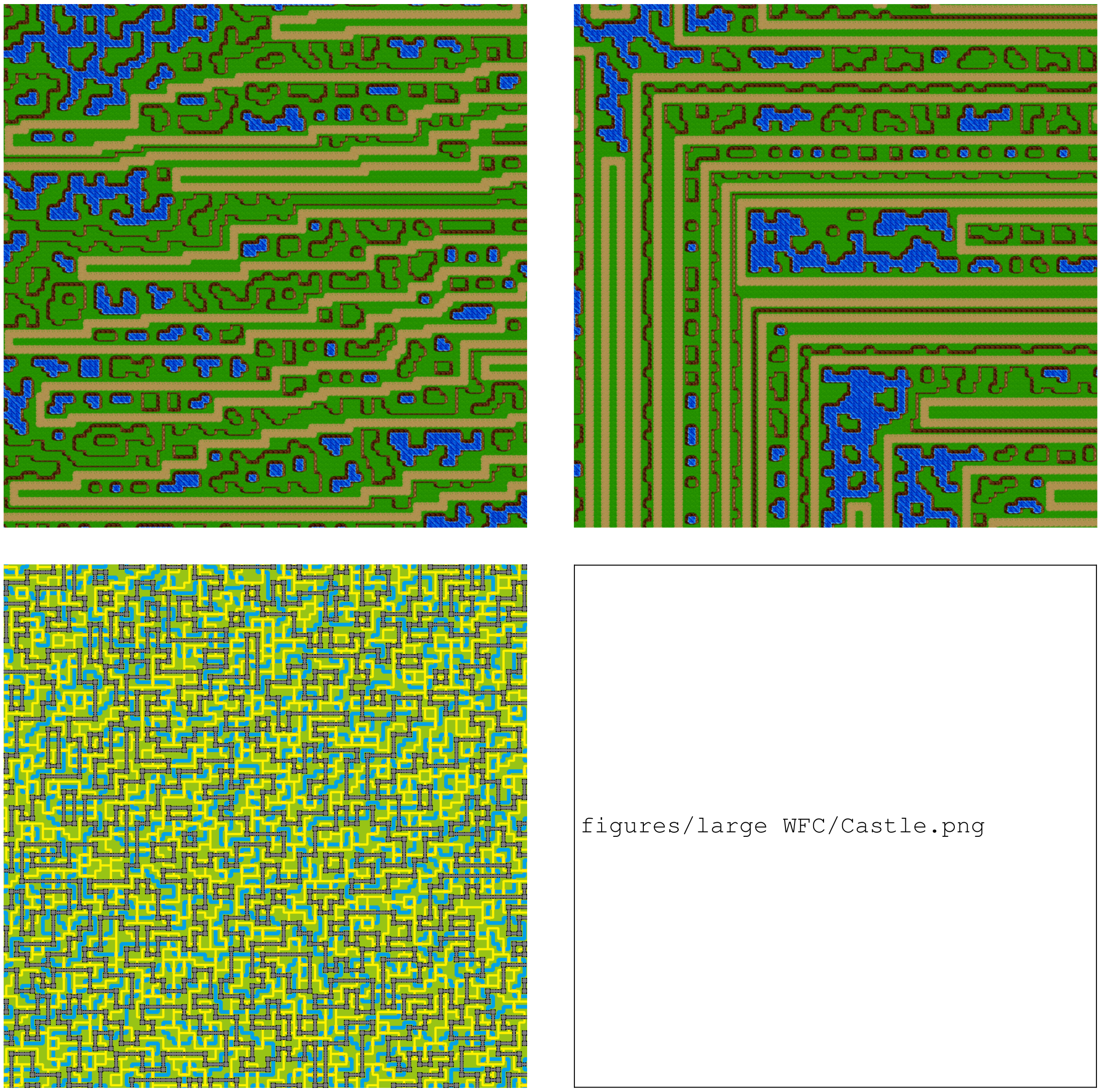


Figure 6: Textures on the left were generated by model synthesis. Textures on right by WFC. The results look similar.



figures/large WFC/Castle.png

Figure 7: Textures on the left were generated by model synthesis. Textures on right by WFC. The results look similar.

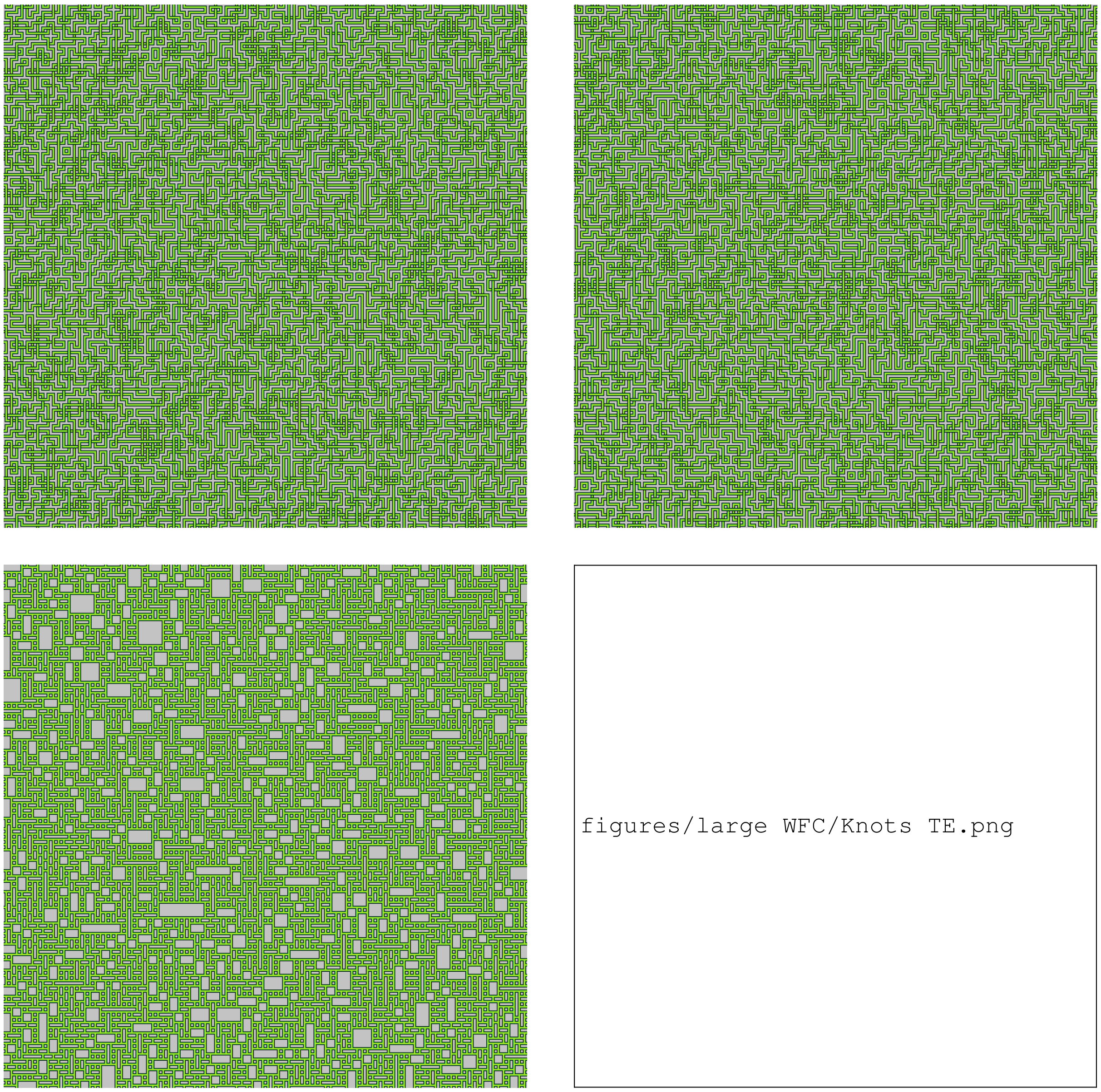


Figure 8: Textures on the left were generated by model synthesis. Textures on right by WFC. One box is empty because WFC was unable to generate that texture.

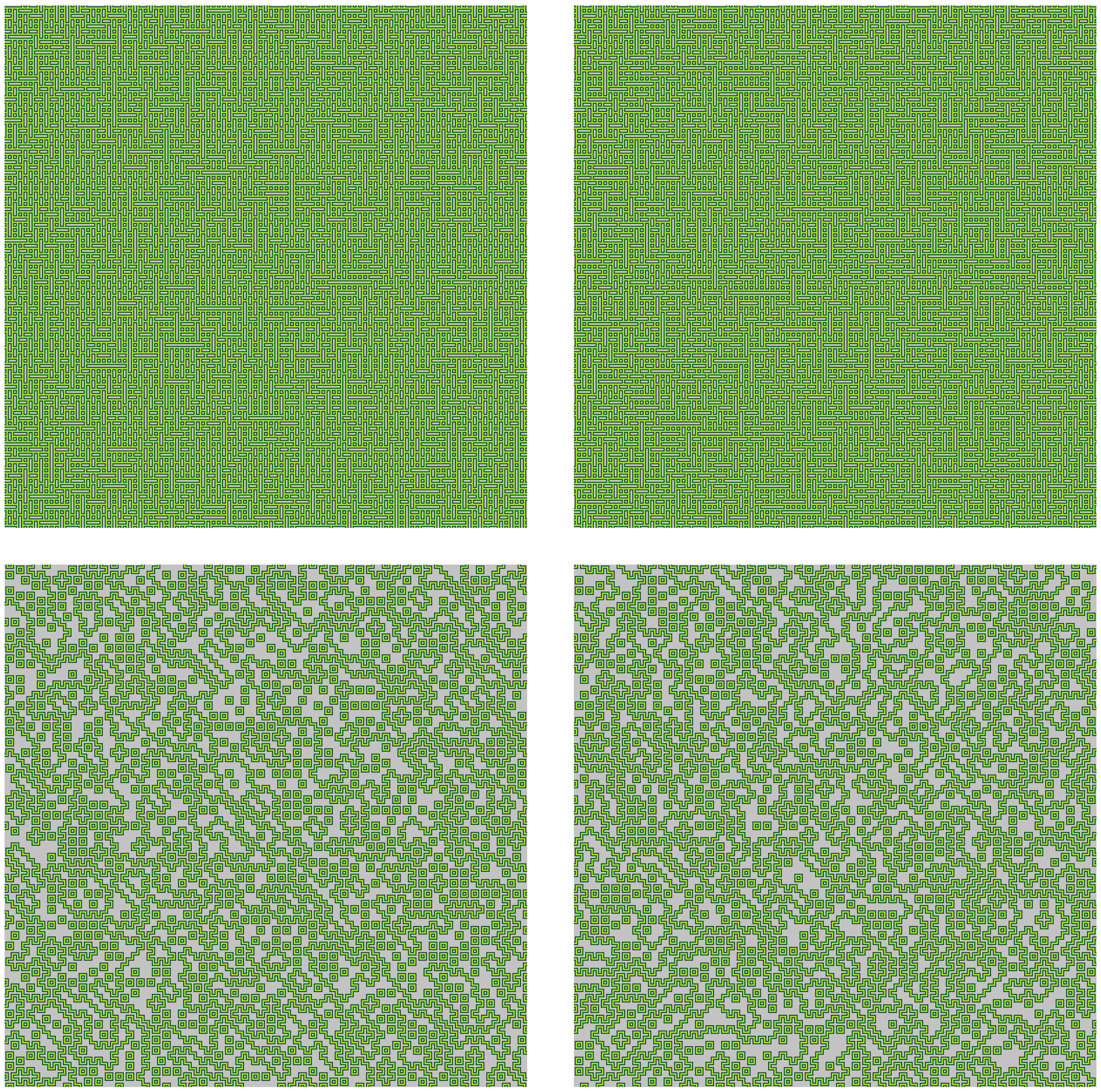


Figure 9: Textures on the left were generated by model synthesis. Textures on right by WFC.

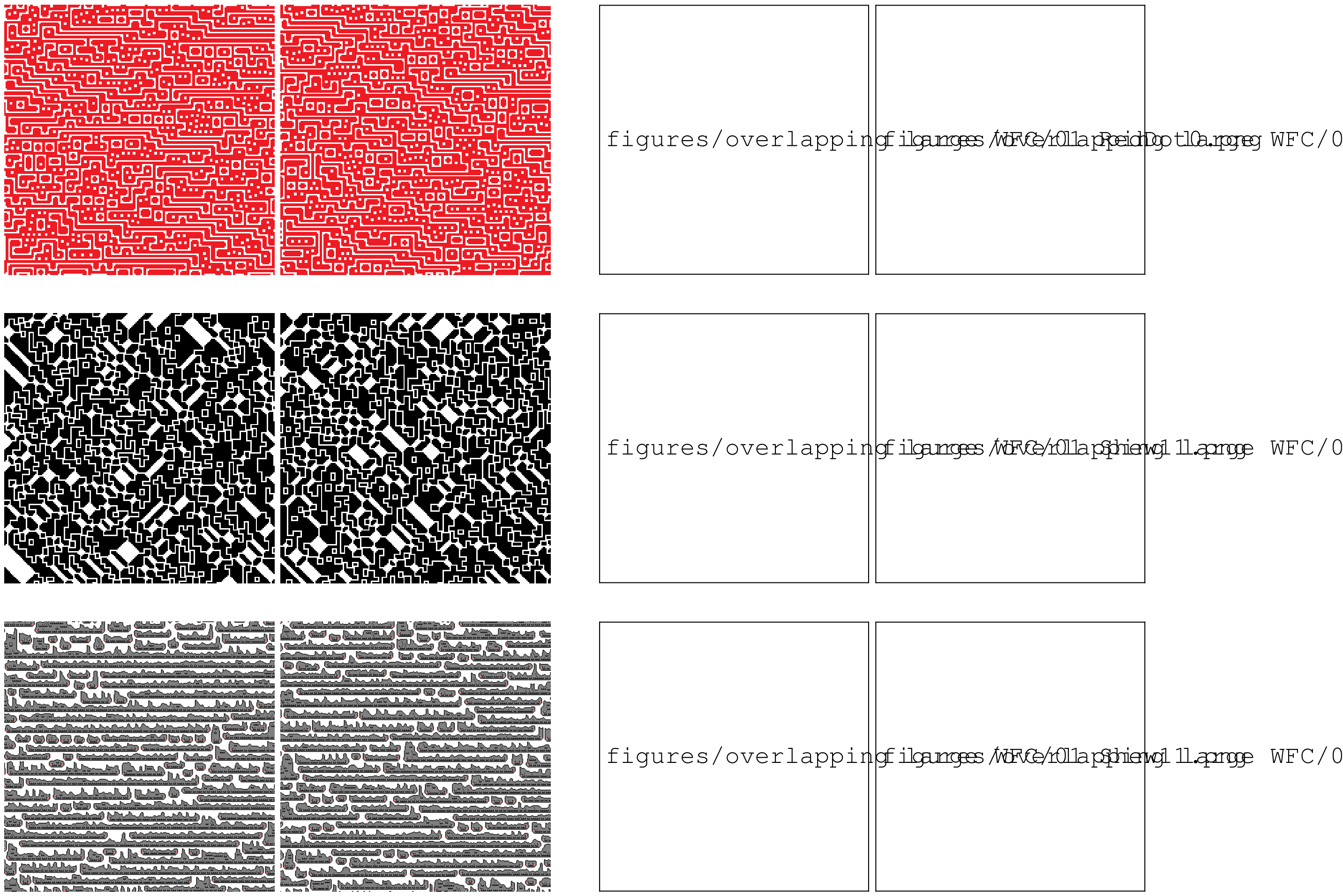


Figure 10: Textures on the left were generated by model synthesis. Textures on right are empty since WFC failed to generate them.