A User Interface for Procedurally Generated Images

Benjamin Eskildsen
Advisor: Holly Rushmeier
May 9, 2016

Abstract

This paper presents a pattern generator in the form of a web application that enables discovery of novel designs based on generative grammars. It serves as a discovery tool for users to create, visualize, and tune context-free generative grammars using an interface that intuitively maps parameters to the rendered output. This application can serve as a tool for artists who wish to generate complex and rule-based patterns, textures, or designs without painstakingly drawing every pixel by hand -- this application is especially useful in an instance where many similar variations on a design are desired. Similar programs require the user to write code themselves (limiting the size of the possible audience)--while this application uses a simple graphical user interface to create varied and complex designs. Emphasis on sliders for parameter input allows users to explore the space of possible designs, but exact numbers can also be entered if precision is required. Furthermore, users can draw their own symbols, allowing any possible mapping of a string generated according to the rules of a context-free grammar to a rendered image. Several pre-made designs have been selected for their utility in showcasing the capabilities of the application. These include the rules to generate a Sierpinski triangle. Creating this application required research into L-systems, generative grammars, procedural modeling, and Turtle graphics, as well as an iterative design approach that led to a user-friendly non-coding-based solution to the problem of visualizing rule-based systems with user-determined constraints.

Introduction and Related Work

While this project took many forms over the course of the semester, a single core idea prevailed throughout: that there should exist an interface for navigating the parameter space of procedurally generated images. Stanford PhD student Daniel Ritchie, who gave a talk at Yale in February, clearly explains the advantages of procedural modeling:

> *It facilitates efficient content creation at massive scale, such as procedural cities. It can generate fine detail that would require painstaking effort to create by hand, such as decorative floral patterns. It can even generate surprising or unexpected results, helping users to explore large or unintuitive design spaces.*[1]

[1] Ritchie, D., Thomas, A., Hanrahan, P., Goodman, N., "Neurally-Guided Procedural Models: Learning to Guide Procedural Models with Deep Neural Networks," 2016, http://arxiv.org/pdf/1603.06143v1.pdf

However, while procedural modeling succeeds in efficiency and discoverability, it loses in lack of precise control over the final output. Current methods for managing this disadvantage focus on one or both of two main schemes: using machine learning methods to cull the range of possible outputs to (hopefully) those that are desirable,[2,3,4] and by limiting the range of images than can be generated to very specific types of objects.[5,6] Given the popular leaning towards machine learning, I wanted to focus on using human preferences, rather than machine-learned classification for manipulating the parameters used to generate the images (furthermore, machine learning methods are often slow, taking at least several minutes to generate simple images). I also wanted to create an application that was not limited to only generating, for example, cities, flower patterns, or Jackson Pollock paintings, but rather could be easily configured to generate any of those things and more.

Process

My initial idea was inspired by Daniel Ritchie's probabilistic domain-specific language within javascript (that he presented at Yale on February 9) that used Bayesian methods to evaluate the generation rules: I created essentially a javascript "preprocessor" that would run through the code checking for certain indicated variables that the programmer intended to be tunable parameters. These variables, rather than being hardcoded, would have their values set by an html slider that was automatically added to the page that was rendering the image. This way, after the code was run, the user could easily adjust the value of that variable and see the change's effect on the image. Next, I wanted a way to navigate this parameter space at a distance, so I added the ability to assign any variable to an axis, rendering multiple sample images side-by-side.

---

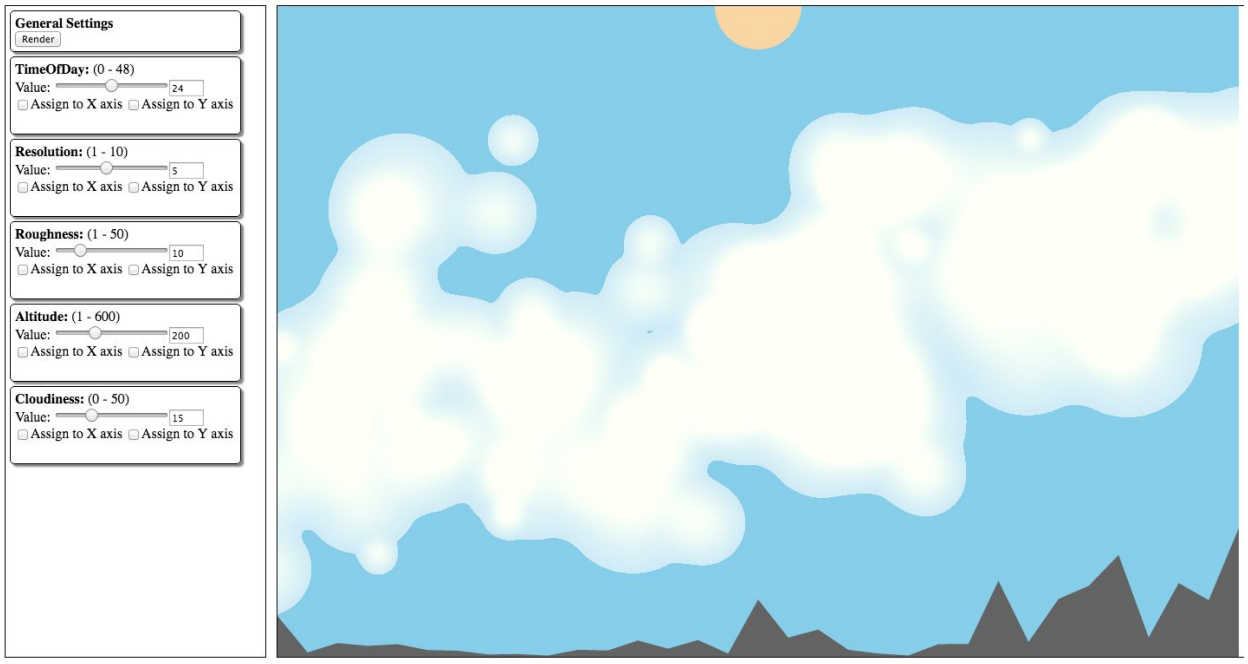
[2] *ibid*

[3] Talton, J., Lou, Y., Duke, J., Lesser, S., Mech, R., Koltun, V., "Metropolis Procedural Modeling," 2011 http://vladlen.info/publications/metropolis-procedural-modeling/

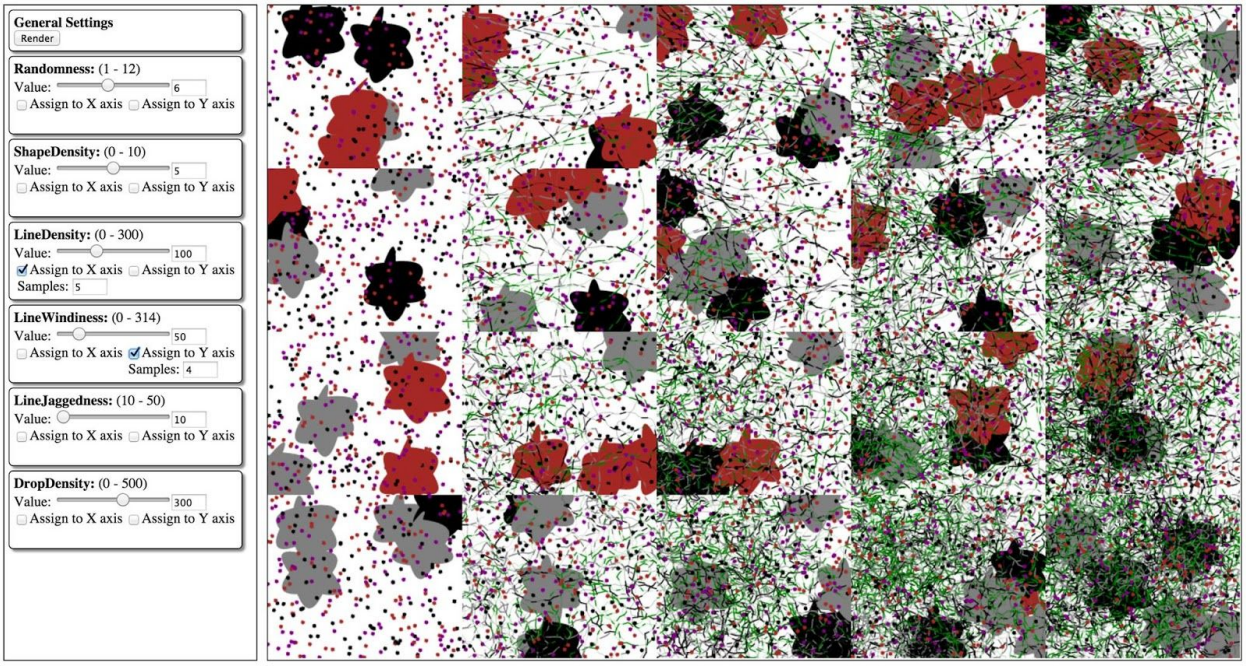
[4] Ritchie, D., Mildenhall, B., Goodman, N., Hanrahan, P., "Controlling Procedural Modeling Programs with Stochastically-Ordered Sequential Monte Carlo," 2015, http://stanford.edu/~dritchie/procmod-smc.pdf

[5] Taylor, R., Micolich, A., Jonas, D., "The Construction of Jackson Pollock's Drip Paintings, https://blogs.uoregon.edu/richardtaylor/files/2016/02/PollockLeonardo-2e2w0wh.pdf"

[6] Kerr, W., Pellacini, F., "Toward Evaluating Material Design Interface Paradigms for Novice Users," 2010, http://pellacini.di.uniroma1.it/publications/matstudy10/matstudy10-paper.pdf



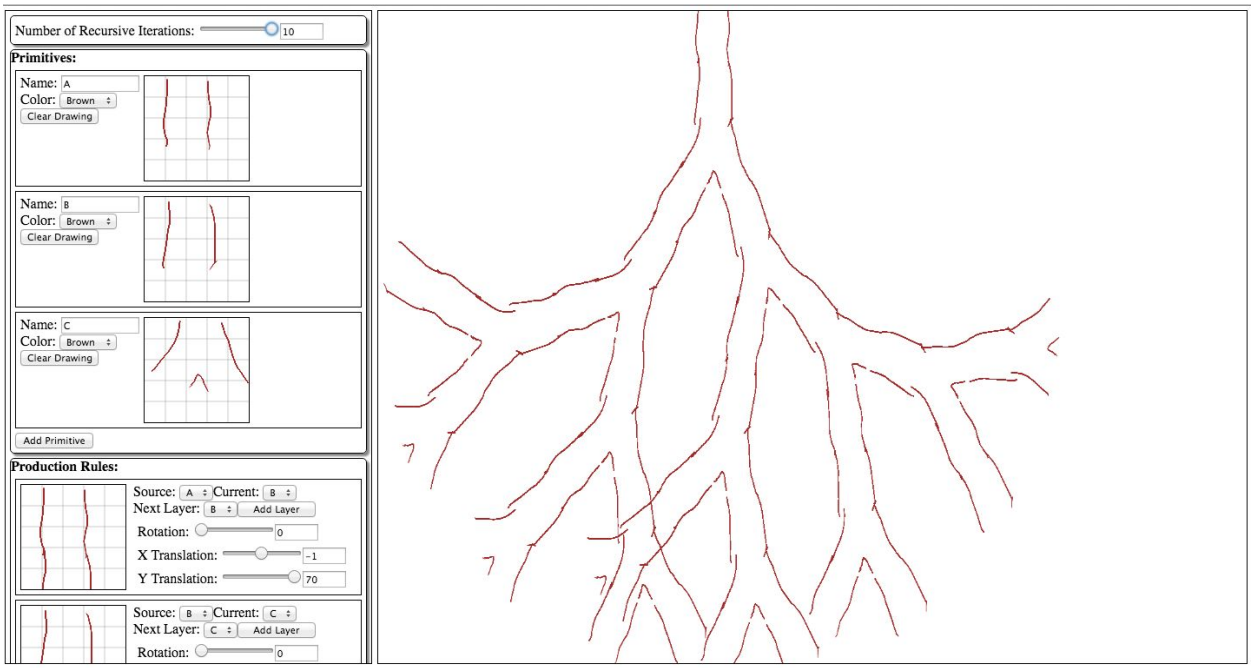
A screenshot showing how sliders could be used to control different parameters of the image. This image shows a mountain range generated using fractal Brownian motion with clouds adapted from a previous CPSC 472 Computer Graphics assignment.



A rendering example from an early iteration of the project of a Jackson Pollock painting using a procedural algorithm to generate the Pollock-style images.[7] The image shows line windiness plotted versus line density.

[7] Algorithm for generating Pollock-style images inspired by Zheng, Y., Nie, X., Meng, Z., Feng, W., Zhang, K., "Layered Modeling and Generation of Pollock's Drip Style," 2014, https://www.utdallas.edu/~kzhang/Publications/VisComp2014.pdf

Ultimately, I decided to move away from this slider-centric interface because significant coding was required behind the scenes to create each algorithm. The average user could not be expected to be able to code their own images and would therefore be stuck with whatever defaults I provided. But to allow the user to generate different kinds of images would require the user to be able to define the rules by which images were generated without writing any code at all. A rule-based approach like contextfreeart.org's interface would be required (though this application still requires the user to write code) since rules could be manipulated and composed rather than the other two extremes: individual pixels or lines of code. The result is similar to Turtle graphics applications, except instead of simply tracing the path of an imaginary turtle as it follows certain rules (limiting the output to those images which a virtual turtle could trace), instead the user draws symbols themselves and then adds "production rules" that specify how a given source symbol is transformed at each step of the recursive generation process.



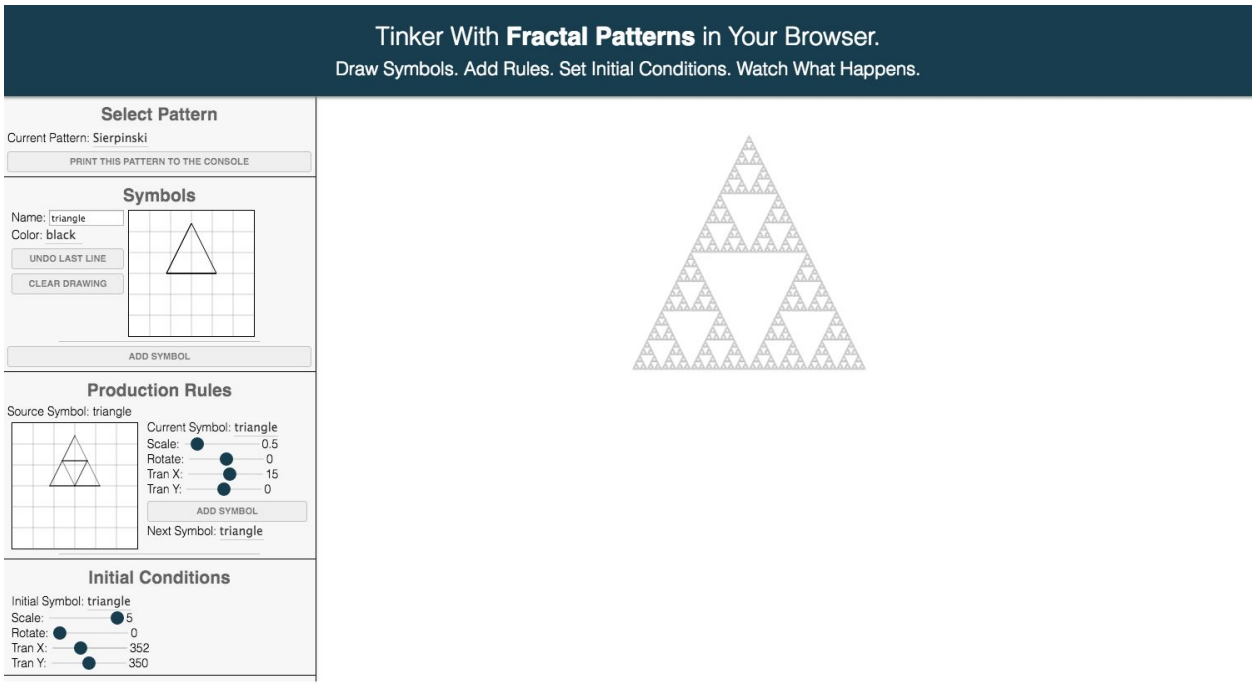
An early version of the context-free grammar image generator used to model the tunnels of an ant colony. The slider at the top left could be moved to change the number of iterations of the pattern, thus the 0th iteration is just a single segment of vertical tunnel and moving the slider causes the tunnel to grow into the pattern seen here at iteration 10.

Results and Deliverables

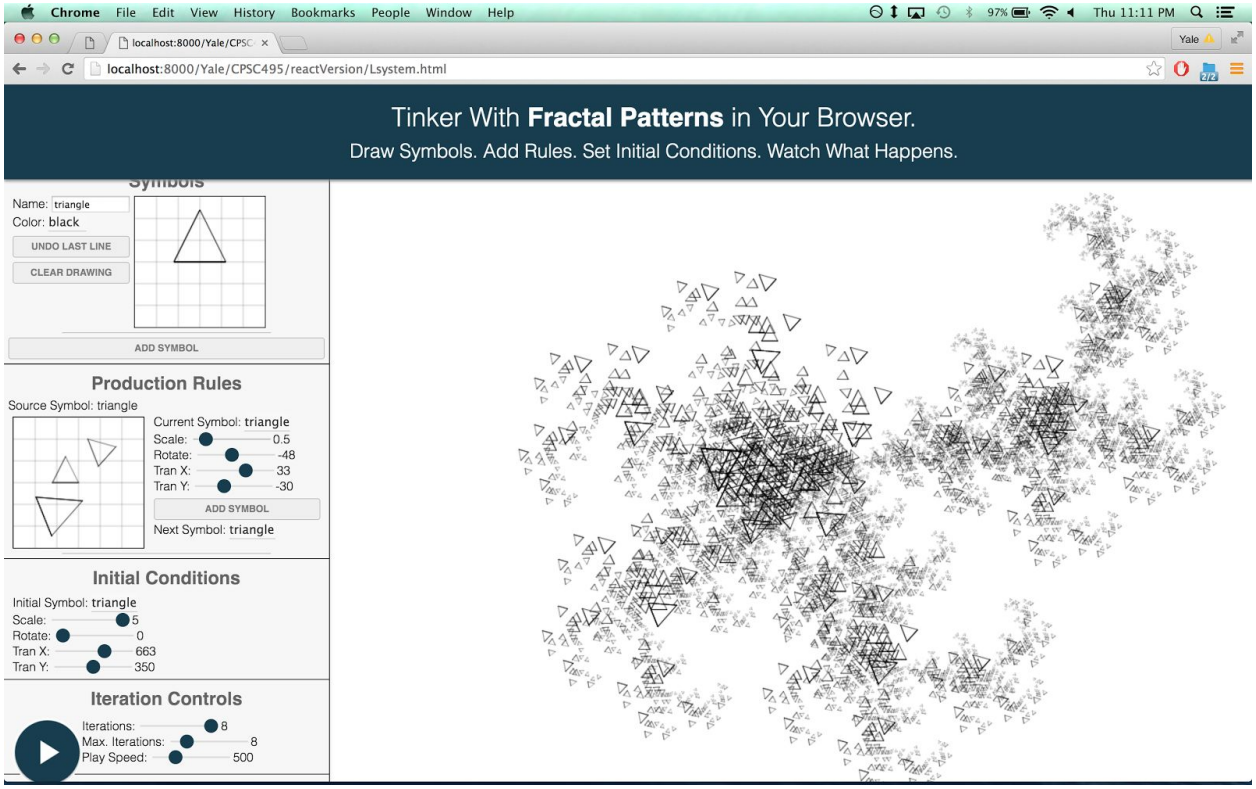
The final product is a graphical user interface for creating, manipulating, and visualizing context-free grammars. The basis of the program is an L-system consisting of: certain symbols that the user is free to draw, production rules that specify how to transform a symbol at each iteration, initial conditions, and then a play-button for watching the pattern generate itself (note that the play button is not initially visible because the side bar is scrollable). The sliders next to all the values, like in the earlier prototypes, let the user quickly see how changes in the input parameters affect the generated image.

To use the program, the user first selects a default pattern (including a blank pattern if they want to start from scratch). Then in the Symbols section the user can edit or add their own symbols that are used in the image. Each symbol has a corresponding Production Rule. That is, at each iteration, the algorithm goes through each symbol in the image and produces the configuration of symbols shown on the small canvas to the left of that rule (with relative scaling, rotations, and translations applied). The productions are manipulated by adding symbols on top of each other and manipulating the transformations of each one at a time. In the Sierpinski triangle example shown below, at each iteration, every triangle is replaced by three smaller triangles aligned with the corners of the previous, larger one. When this is iterated again and again, it creates the full Sierpinski triangle.

Behind the scenes the program, written with Facebook's React javascript framework, maintains an array of the outputs of each iteration (rather than simply storing all the lines, though, the program stores a list of symbols and their corresponding transformation matrices, and the lines are computed on the fly). When the user makes a change by, for example, moving a slider, the array of iterations is recomputed given the adjusted transformation and then re-rendered -- giving constant and immediate feedback. Each iteration is necessarily computed based on the output of the previous iteration (by multiplying the production rules' transformation matrices by the transformation matrices of the source symbols), so it makes sense to simply store all the iterations so that no more computation is required once the play button is pressed.



The Sierpinski triangle is one of the default patterns that the user can manipulate. This is the triangle after 8 recursive iterations.



By adjusting the position of the triangles in the production rule of the original Sierpinski triangle and running the pattern for 8 recursive iterations, this spiraling galaxy-like fractal was discovered.

Future Work

The goal of the project was to design and build an interface for creating and manipulating procedurally generated images without using machine learning or restricting the user to specific kinds of visual outputs. This interface succeeds in both areas, though it is not without its flaws. Handling and drawing the recursive iterations can quickly overwhelm the browser's memory, crashing the tab (so increase the maximum number of iterations at your own risk). Many performance optimizations are possible. For example, changing the initial conditions should not require recomputing the output of the iterations since transforming the original source symbol would be equivalent to just transforming the entire finished image (which could be done without recomputing the productions). The nature of the react framework is such that all state changes force a total re-render by default and I stuck with this behavior initially because it is easy to reason about, but moving forward on the project would necessitate as many speed increases as possible.

Another obvious addition would be stochastic rules, rather than purely deterministic rules. I had implemented a prototype that could allow for multiple possible outcomes, but the ways to use it were less clear than the deterministic version: should pressing "play" cause a different random variation to be rendered each time the recursive iterations started over? (And if it was done this way, how could it be done efficiently, since my current implementation pre-computes all the iterations for speed). How could the user navigate the non-deterministic elements otherwise? I decided to try to keep the project constrained for now, but the introduction of randomness is a definite addition once it is more fleshed out.

Finally, I think it would be a good idea to add a play button next to every slider, rather than have just the single play button at the bottom. This way instead of playing through just the iterations, the user could play through, eg. symbol rotations from -180 to +180 degrees or scaling from 0 to 5 times and watch how changes in each parameter change the outcome.

Acknowledgements

I greatly appreciate Professor Holly Rushmeier's advice and guidance throughout the project. Her involvement throughout the semester kept me productive and on track.