Princeton Psychedelic Simulator

COS 426: Computer Graphics, Spring 2021
Isabel Zaller (izaller)
Kasey McFadden (kaseym)

Abstract

Our project aims to create a psychedelic-like experience for our users. Essentially, we want to build something really interesting looking. Using Three.js, we intend to generate fractals and animate their motion. We want to capture the attention of our users and inspire them to think creatively. Working at the intersection of computer science and the natural art of fractal patterns, we hope to inspire others to think of their CS skills as not just applicable to tech, but as creative tools across all fields.

1. Introduction

1.1. Goal

The goal of this project is to create a psychedelic online interactive art exhibit, complete with fractals and dynamic geometric patterns. We want viewers to be impressed, engaged, and creatively inspired when they interact with the exhibit. By creating complex and original computer-based art and allowing the user to guide the direction of the art generated, we hope to design a fun and rewarding experience for anyone who tries it.

1.2. Previous Work

In brainstorming ideas for this project, we were initially very inspired by the beautiful fractal art we saw all over the internet. We saw incredible animations in both two and three dimensions [1, 2, 3, 4] and were excited by the idea of contributing to the world of fractal art. Many of these implementations use Three.js, which was an encouraging sign as to the feasibility of such a project. Most instrumental of the previous work is the description of an algorithm called the Chaos Game [5, 6], a method of fractal generation on which we based our project.

2. Methodology

2.1. Chaos game algorithm

We completed this project by using an algorithm called the Chaos Game. We chose to use this approach because it allows us to work within a framework similar to those used on past assignments (specifically assignment 5) and leverage several Three.js data types and methods that we have been exposed to throughout this course. The chaos game algorithm for pattern generation is as follows:

1. Choose a random point *p* in the polygon. This is your starting point.
2. Choose a random vertex *v* of the polygon.
3. Choose a new point *p*' that is some scaled distance *r* along the vector from *p* to the selected vertex *v*.
4. Repeat steps 2 through 4 until you have reached the maximum number of iterations, using *p*' from the current iteration as a starting point for the next iteration.

By adding each of these points to the scene as Three.js Points, we can visualize the creation of interesting fractal patterns. We allow users to toggle the number of vertices and side length of the polygon along with the *r* parameter, which ranges from 0 to 1.

2.2. First-pass: triangles

To begin, we set out to use our algorithm with the simplest possible polygon, a triangle. The file `triangle.js` serves as the first-pass implementation of our idea. This file was inspired by the use of `particle.js` and `cloth.js` from Assignment 5. Our triangle data type stores triangle side length and vertex positions, both of which we initially fixed as static properties so that we did not have to manage any additional complexity at the outset. This file also contains methods allowing for the selection of random points within the triangle.

Once we completed `triangle.js`, we began to edit the `sim.js` file in order to show our resulting fractal patterns. We started with a very bare-bones template of the methods from the Assignment 5 file and filled the functions with appropriate code. The function `Sim.chaos` does the work of the chaos algorithm.

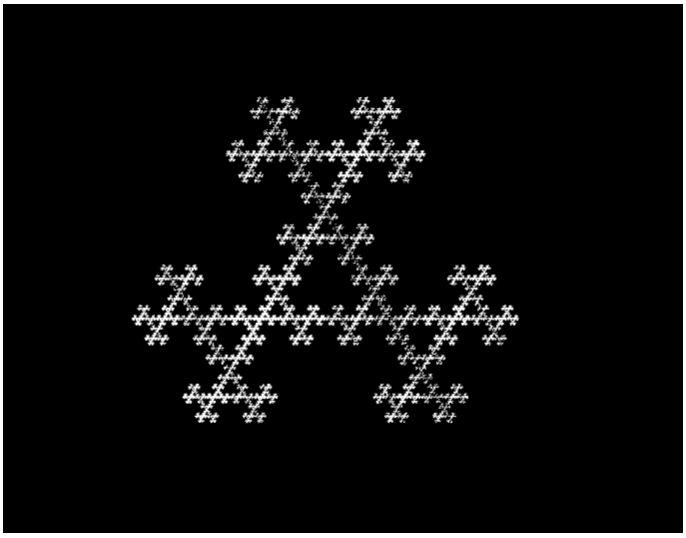

Figure 1. Triangle drawn with chaos algorithm.

Rendering was initially quite slow when we were drawing one point for every call to `Sim.chaos()`, so we introduced a loop in this function in order to display more points per function call. We kept adding more and more points until we had reached the maximum number

of iterations of our algorithm, which can be toggled by the user via GUI controls. This is quite useful when displaying bigger polygons in order to ensure pattern clarity.

We also allow the user to select whether they want the polygon to fade in and out or fade in and stay on the screen. We allow users to pause the fade in/out at any time so that they can see the progress of the algorithm up to that point in time.

Our implementation of triangle proved useful moving forward for testing new features on a small scale, particularly with the spin feature.

2.3. Generalizing to the n-gon

Once we successfully achieved triangle visualization, we began work towards polygons with more vertices. Our "n-gon" is the result of this work: a regular n-sided, n-vertex shape. In generalizing from triangle to n-gon, we now had to deal with a more complex constructor. Rather than having fixed positions for our vertices, we needed to be able to determine vertex positions based on the number of vertices. We used trigonometric calculations [7] to find vertex positions based on the calculated internal angle of the regular polygon, circumradius, and inradius. Our ngon constructor was more complex than our triangle constructor in its computations and stores two additional parameters, the number of vertices and the height of the ngon (the diameter of its bounding circle), beyond side length and vertex positions.

With our abstracted implementation of `sim.js`, it was a simple matter to change from using our triangle data type to our ngon data type when we were ready to test. There were some initial bugs to work out with the computation of vertex positions and scaling of side lengths, but once those kinks were worked out we were able to create some beautiful patterns.

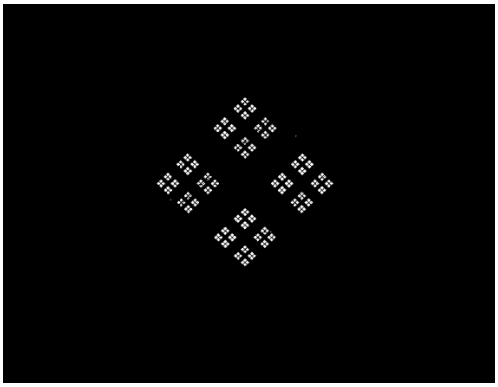

Figure 2: Squares.

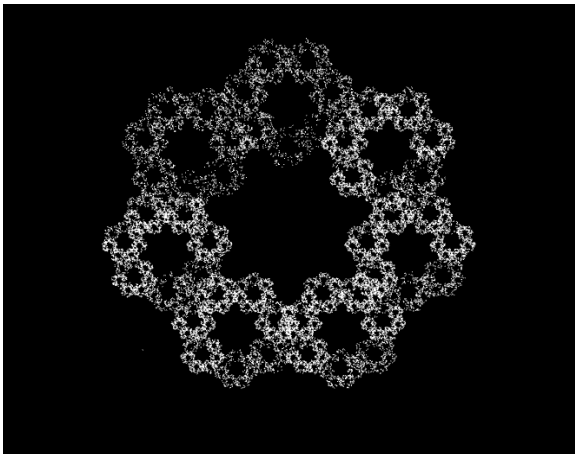

Figure 3: Heptagons.

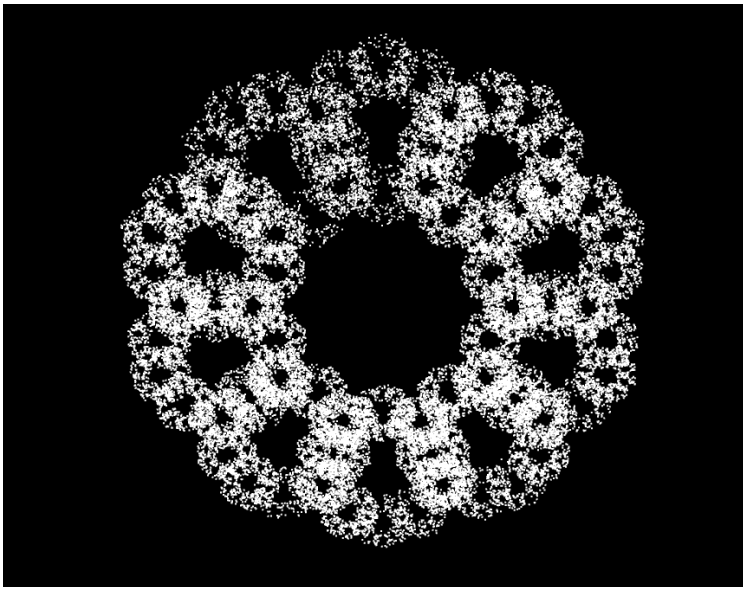

Figure 4: Decagons.

2.4. Adding more complexity: vertex restriction

Not all n-gons form interesting patterns on their own when the chaos algorithm is applied. For example, a square (4-gon) forms an unattractive block of points seemingly without order:

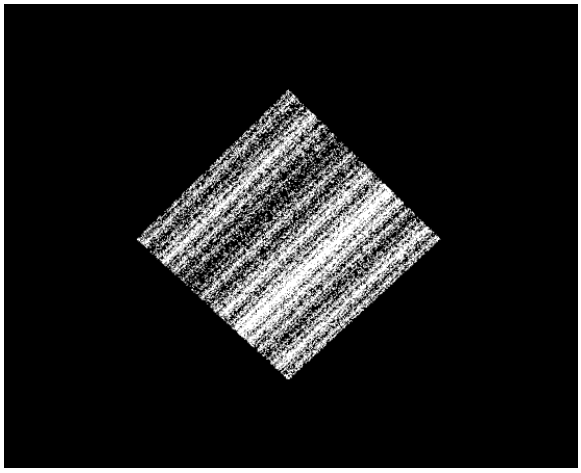

Figure 5: Square.

Enter the idea of vertex restriction. In step 2 of the described algorithm, instead of randomly selecting any vertex, we can set boundaries on which vertices may or may not be chosen. One possible restriction would be to select from all vertices except the one which was used in the last iteration. Another would be to not allow the selection of a vertex which is 2 away from the current vertex. By applying vertex restriction, we are able to create even more patterns.

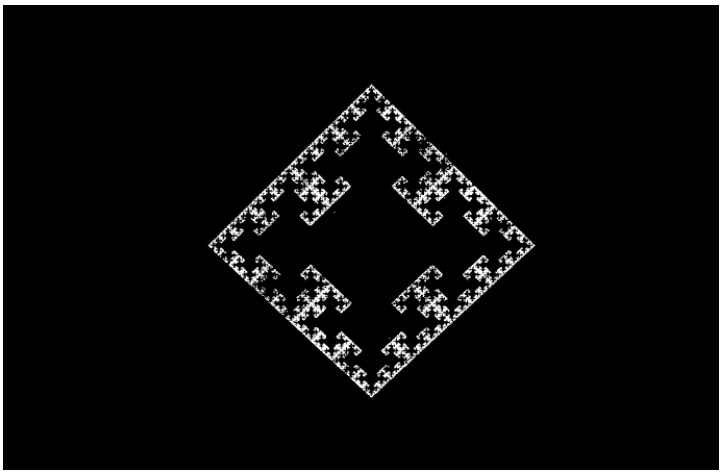

Figure 6: Square with vertex restriction.

2.5. Spin

Having established fractal patterns in place in the Three.js space, our next step was to find a way to scale and move these patterns to create an oscillating or kaleidoscopic effect. In order to generate this kind of motion with the fractal pattern, we call the function `ngon.spin()`, which spins the fractal clockwise by acting on its vertices. To do this, we calculate vectors spanning from each vertex to its neighboring vertex in order of increasing index. We use these vectors to offset the positions of each vertex in the direction of that vertex and of a scaled magnitude

determined by parameter “scale”. `ngon.spin()` is called between iterations of the `Sim.chaos()` function.

2.6. Animations

Most recently, we have been working on allowing the user to select from a variety of animations. So far, we have “rotate”, “bounce”, and “rave”. Each of these were implemented in the method `Renderer.render()` by modifying the perspective camera’s position. Bounce and rave were actually discovered as bugs during the implementation of rotate, but we liked the effect that they created and decided to keep them.

3. Results

Thus far, we have successfully achieved our MVP and then some: we have implemented 2D fractal visualization with animations. Additionally, we have added lots of opportunities for user interaction with the art, allowing our users to fully immerse themselves in our exhibit not just as passive observers but as active participants in the artistic process.

4. Discussion

We’ve accomplished a great deal in the time we’ve been working on this project, but there is still more that we hope to accomplish before we are ready to be done. Before the final deadline, we plan to work on adding color to the patterns, improving the spin animation with forward and reverse functionality (similar to fade), and putting in more time on new vertex restrictions. Additionally, we hope to create new animations for our users to watch. Ideally, we would be able to have an animation in which fractals fade from one pattern to the next, possibly with a zoom functionality; and perhaps a tile option, which uses lots of small fractals to fill the entire screen. These constitute far-reaching stretch goals and are dependent on our successful implementation of color, spin, and updated vertex restriction features. It may be more worth our time to perfect these features rather than pursuing new ones.

We both learned so much from this project. Perhaps the most important thing we learned is the difficulty of starting such a project. Once we had the framework of a viable MVP, we were able to really get rolling and split up the work. However, it was really quite difficult to get started. We tried using the download starter code from the final project assignment page but had limited successes, as neither of us were familiar with the setup of those files or with Node. We then transitioned to using the architecture from Assignment 5, which was a great starting point for us, but required a not insignificant amount of work to overhaul for use in this project.

An unexpected upside to using the Assignment 5 code as a foundation is that we have been able to do a deep dive into how all of our assignments have worked. Rather than just editing one or two files, as we did in assignments throughout this semester, we were making changes to many of the project files. It was really interesting to learn how the animations and visualizations that we’ve been using are generated.

5. Conclusion

We believe that we successfully achieved our goal of creating an interactive art exhibit which stimulates and engages our users. By taking the concrete steps discussed in the previous section, we will take our project to the next level. Beyond the scope of the time frame for this project, it would be awesome to add additional features such as different music tracks for each animation, more shading and three-dimensional effects, and better rendering performance. Overall, though, we are quite pleased with our project.

6. Contributions

This project was in large part a group effort done with both of us sharing a computer screen and figuring it all out together. Once we had completed the initial stage of being able to see a pattern generated using the modified Assignment 5 code, we split up the work a bit more. Izzy focused on generalizing from triangle to ngon, vertex restrictions, and animations. Kasey worked on the spin effect.

7. Honor Code

We pledge our honor that this project represents our own work in accordance with university policies.

/s/ Isabel Zaller
/s/ Kasey McFadden

8. Works Cited

[1] https://www.taoeffect.com/other/fractals/mandelbulb/
[2] http://algorithmicartmeetup.blogspot.com/2019/02/threejs-overview-and-visualising-games.html
[3] https://github.com/jmetterrothan/threejs-fractal
[4] https://medium.com/@SereneBiologist/rendering-escape-fractals-in-three-js-68c96b385a49
[5] https://en.wikipedia.org/wiki/Chaos_game#Restricted_chaos_game
[6] https://mathworld.wolfram.com/ChaosGame.html
[7] https://calcresource.com/geom-ngon.html