CS 480 - Project Paper

Michael Whalen

April 18, 2018

Introduction

When I started this project, I really didn't have a good idea of its scope or what exactly I'd encounter along the way. I chose to do this project because I was vaguely familliar with hyperbolic tilings and thought they looked great, and that they'd be a neat playground for cellular automata. I ended up spending the first half of the semester largely just reading textbooks and math journals to learn how to actually construct the geometric objects I was after. Throughout that whole time I was unsure that I'd actually be able to complete the project and even had another project idea as backup. A couple months in, though, everything seemed to finally click and the code started flowing like water.

I'll begin by describing the gist of the mathematics behind hyperbolic space, then move into the algorithmic aspect of implementing what I needed. Finally, I'll discuss how I implemented cellular automata and the

discoveries made as a result.

Hyperbolic Geometry

At the most basic level, **Hyperbolic Geometry** is exactly like the normal Euclidean geometry we're used to, with one twist: with Euclidean geometry, given a line and a point, there is only one other line through that point that is parallel to the given line. In hyperbolic space, we let there be an infinite amount. This one allowance gives way to a ton of interesting behaviors and can be modeled in a multitude of ways. For my purposes, I chose to use what's known as the Poincare model as it has a significant amount of research in it and was relatively simple to learn more about. In essence, the Poincare model is represented by a disk, wherein 'lines' are defined to be arcs of Euclidean circles orthogonal to the disk.

Like Euclidean geometry, in hyperbolic geometry we can examine polygons. Moreover, we can create a **tiling** of the space using polygons of many kinds, i.e., the space is filled by interlocking, identical shapes. This project concerns itself only with regular polygons, but irregular polygons could be an interesting thing to explore as well. **Regular polygons** are shapes whose angle measures are uniform, like squares and pentagons in Euclidean space. The real restriction of Euclidean space however is that there are only three tilings of regular polygons: equilateral triangles, squares, and hexagons. Nothing else works as there will either be overlap of polygons or

empty space. This is where the value of hyperbolic space comes into play: there exists an infinte number of regular polygons that tile the hyperbolic plane.

Describing Tilings

Since there are so many tilings, I needed a simple and programmatic way of referring to individual tilings. This was done using a **Schläfli Symbol**, notated as **{p, q}**, where p denotes the number of sides on the polygon and q denotes the number of polygons adjacent to any given vertex. For example, in Euclidean space we can represent the square tiling as $\{4, 4\}$ since squares (four sides) meet four at each vertex. This symbol is in fact enough to determine whether a given tiling is Euclidean or hyperbolic. With the following formula:

$$1/p + 1/q = \theta$$

When $\theta = 1/2$, the tiling is Euclidean. When $\theta > 1/2$, it's hyperbolic. Using these pairs of numbers makes it much easier to implement them in software as a two-value representation of a complex concept is simpler to deal with than an alternative.

Constructing Tilings

So we know what a tiling is, but how do we make one? It's actually a question with a suprisingly simple solution, yet it took me a couple of months to realize it through extensive reading. We can construct hyperbolic tilings in exactly the same way that we might in Euclidean space: start with a polygon, reflect it about each of its sides, and repeat for these new polygons. The only real difference with hyperbolic space is the method of reflection. Because the sides of the polygon are sections of circles, reflecting a point about a side is the same as performing a Euclidean inversion about the corresponding circle.

Software Implementation

Thus, the following breadth-first algorithm arose:

1. Generate vertices for the polygon centered on the origin of the Poincare disk
2. Generate arcs between these points to form the sides of the polygon
3. Reflect vertices about each of the sides
4. Generate

Cellular Automata

Conclusion

Completed Points

Feature	**Point Value**
Hyperbolic tilings of a single polygon	20
Support arbitrarily many polygon types	20
Color in the tilings according to an automaton	20
Animate changes in state	10
Supports multiple sets of rules	5
Supports an arbitrary amount of rulesets	10
Supports more than two states	5
Users can interactively change the state of cell by clicking	10
User selects cell dimensions	5
User inputs rulesets	5
User inputs possible states	3
User selects color of states	3
User selects speed of animation	1
Start/stop animation	1
Step one generation at a time	1
Can save/load states and rules	5
Can zoom in	5
Zoom without losing resolution	10
Can rotate view	10
Total:	**149**

Point Range	**Grade**
100+	A
85 - 99	B
70 - 84	C
50 - 69	D
Below 50	F