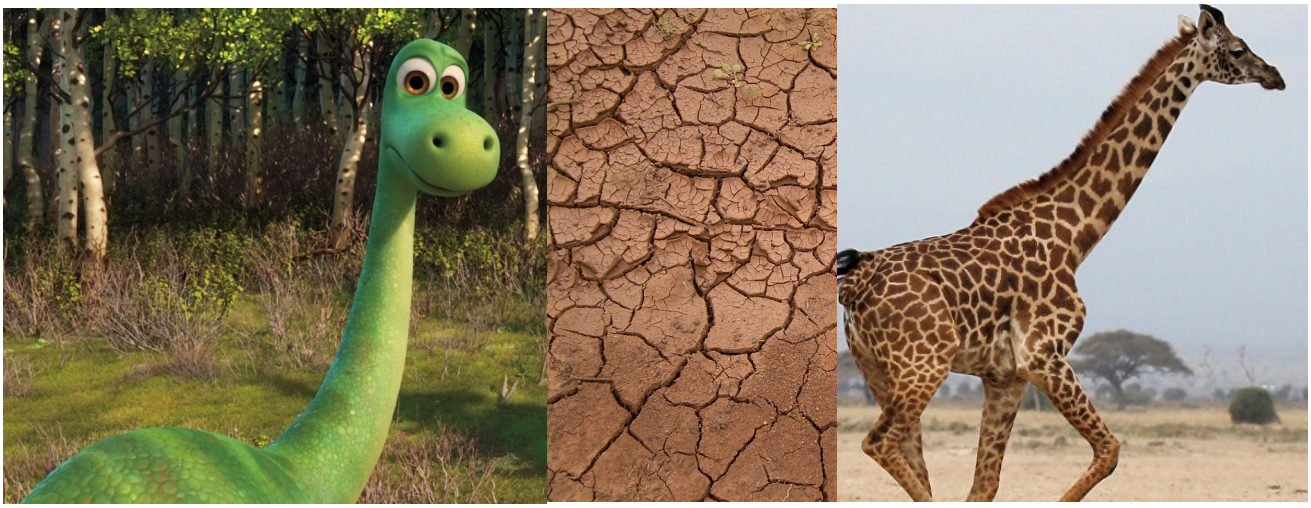
Procedural Texturing Using Voronoi Diagrams
CS 354 – Computer Graphics
Ramsey Hashem

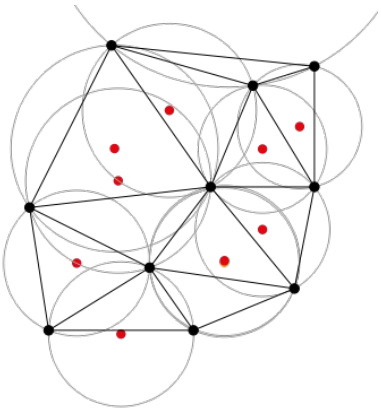
Introduction

For this project, I was primarily inspired by the Pixar movie *The Good Dinosaur.* More specifically, I was fascinated by the Pixar in a Box videos provided by Khan Academy, a portion of which are dedicated to patterns and the way the dinosaur's scales were achieved. The dinosaur's scales, and many other textures used in Pixar films, were based off of Voronoi diagrams. This form of procedural texture generation provides an interesting method for creating natural looking textures that mimic something like scales, giraffe patterns, mud, plants, and more. My goal with this project was the explore this concept and to see if I was capable of generating my own procedural generated textures that could be applied to a 3D model.

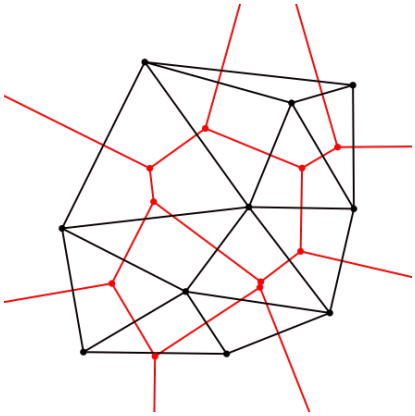


Methodology

There are many ways to generate a Voronoi diagram, some more complicated than others. The path I ended up pursuing was in which includes creating a Delaunay graph from a set of points and taking the dual of this graph to create a Voronoi diagram. I made heavy use of GCAL, more specifically the 2D Triangulation package, to accomplish this task.



A Delaunay triangulation with all the circumcircles and their centers (in red).



Connecting the centers of the circumcircles produces the Voronoi diagram (in red).

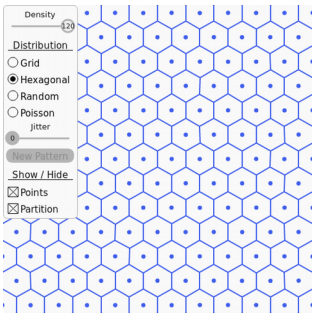
Implementation

The basic generation of the Voronoi diagram for this project involves 3 major steps:

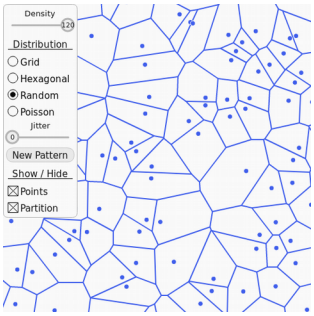
1. Generation of points
2. Creation of the Delaunay graph from these points
3. Creation of the Voronoi diagram by taking the dual of the Delaunay graph.

Task 1 was accomplished from a small library I found online, which I credited in the README. Points that are too uniform look fake and unnatural. Points that are too random look messy and unnatural. That is why I generated my points using a Poisson distribution, meaning that each new point was at least a minimum distance away from all the other points. My project generates these points and outputs them to Poisson.txt as well as Poisson.bmp, the latter of which can be found in **Examples**.

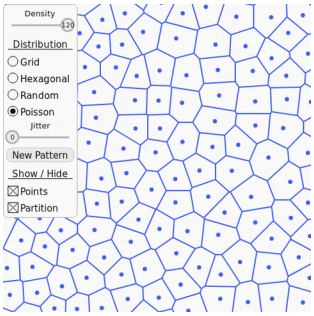
Uniform Points yield an uninteresting grid



Offsetting Points yields a honeycomb grid



Random Points yields a messy grid



Poisson Distributed Points yields a more natural looking grid

Tasks 2 and 3 were simple with the help of GCAL. The 2D Triangulation package allowed me to easily create a Delaunay graph from the Poisson points, and then take the dual of that graph to get the Voronoi diagram. My project creates the Voronoi diagram and outputs its points to voronoi_diagram.txt as well as Voronoi.bmp, the latter of which can be found in **Examples.** The points in voronoi_diagram.txt are written as segments, each of which has the x and y coordinate for that line segment's two vertices.

After I had the points for the Voronoi diagram, I used OpenGL, GLEW, and glfw to create the actual diagram and then output it to voronoi_texture.bmp, which can be used in the Project 1 code base for texturing.

I also added options allowing the user to change various settings, such as the size of the diagram cells, the resolution of the output files, the width of the diagram's edges, and the colors of the graph. Many examples with various settings can be found in **Examples.**

Discussion/Conclusion

I am happy with the work I accomplished. I was able to successfully generate the Voronoi diagrams, offer customization, and generate actual image files that can be used as textures. The most challenging aspect of the project, once I got comfortable with all of the libraries and figured out exactly how I was going to generate the Voronoi diagram, was doing the actual drawing of the diagram and including the various aspects of customization.

I also came across a few issues during this project. In some instances, I will seemingly randomly get a memory allocation error when trying to initialize the glfw library in line 421 of Voronoi.cpp in the display() function. Additionally, there seems to be a maximum line width that I'm able to use for my lines. From what I gathered online, the maximum width is dependent on your system.

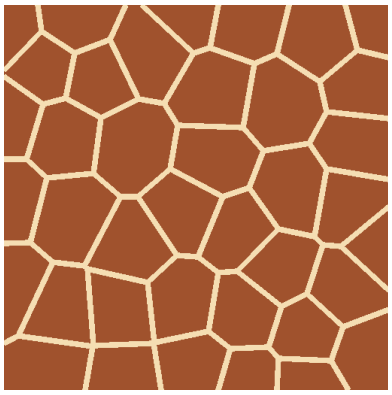
Going forward, there are much more robust things I can do with something like this, mainly in terms of more detail and subtle variations for the textures. In an ideal scenario, I would be able to generate Voronoi textures in which each cell can have a different color, and the lines can have varying widths, in order to add a bit of randomness to more closely mimic nature. Being able to apply a palette of colors to a diagram instead of just two would make for a much more convincing looking texture.

I would also be interested in using the lines of the diagram for bump/displacement mapping on a 3D model, to show depth and create a more realistic effect.

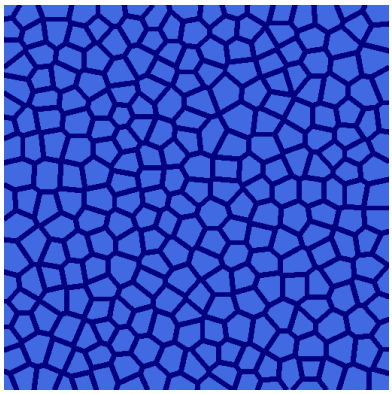
Additionally, there are some closely related ideas for procedural texture generation that I think could be worth exploring. For example, there are different variations of Voronoi diagrams that could be implemented. Also, the Poisson distributed points provides a rather pleasant arrangement that could be used to generate something like polka dots.

I think this project was a good starting point for what I wanted to accomplish. I learned a lot about procedural texture generation, Delaunay/Voronoi diagrams, and OpenGL as well as CGAL. Most importantly, I have something that creates some cool images!

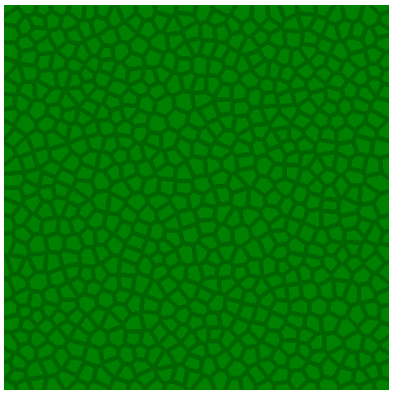
Examples



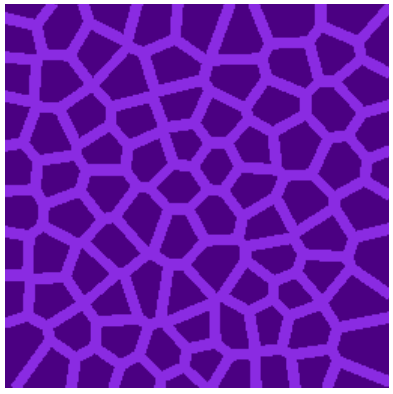
"Giraffe" texture. Resolution: 512x512; Cell Size: 900; Line Width: 10; Cell Color: sienna; Line Color: wheat



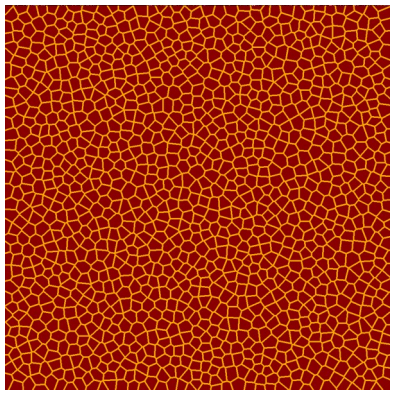
Resolution: 512x512px; Cell Size: 600, Line Width: 7; Cell Color: royalBlue; Line Color: navy



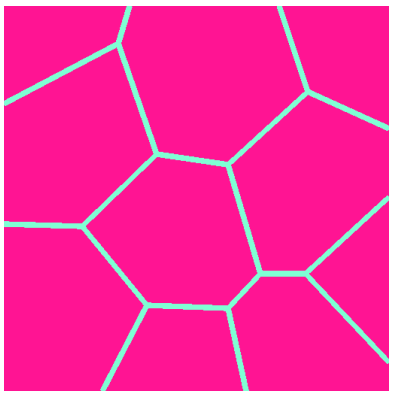
"Scales" texture. Resolution: 512x512; Cell Size: 30; Line Width: 6; Cell Color: green; Line Color: darkGreen



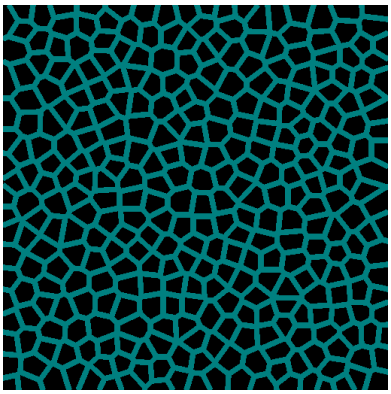
Resolution: 256x256px; Cell Size: 760, Line Width: 8; Cell Color: indigo; Line Color: blueViolet



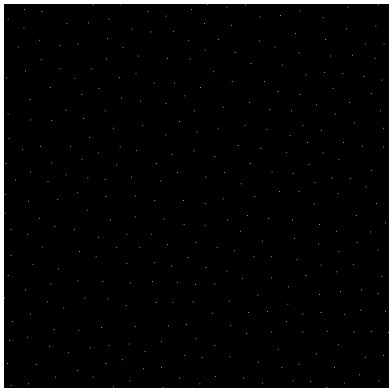
Resolution: 1024x1024px; Cell Size: 3, Line Width: 4; Cell Color: darkRed; Line Color: orange



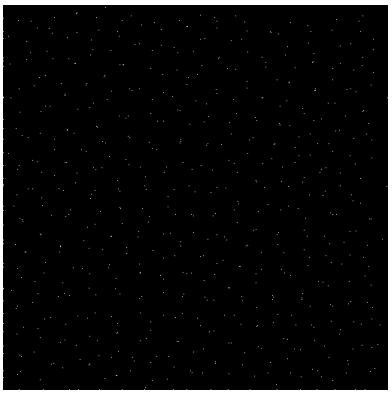
Resolution: 512x512px; Cell Size: 990, Line Width: 10; Cell Color: hotPink; Line Color: aquamarine



Resolution: 512x512px; Cell Size: 500, Line Width: 10; Cell Color: black; Line Color: teal



Poisson.bmp for the black/teal texture



Voronoi.bmp for the black/teal texture