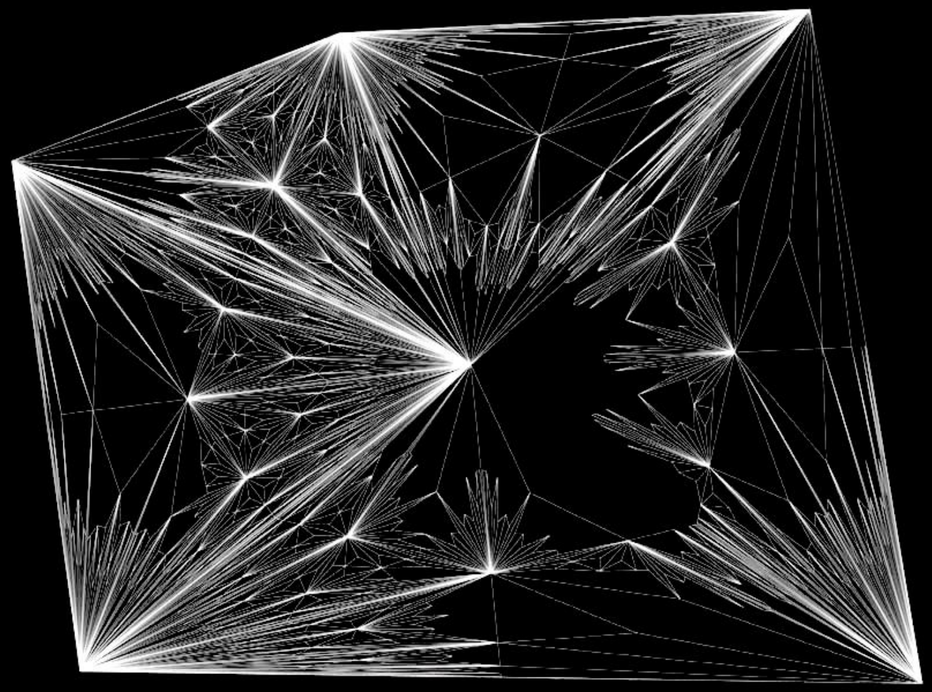
Assignment 6: Grasshopper 1 - Visual Coding and Algorithmic Manipulation



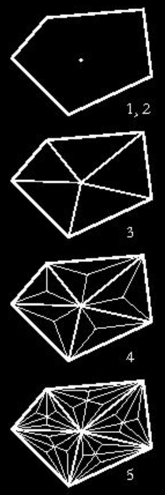
**Recipe for Cracking**

1. Choose a shape to be cracked.
2. Find its centroid.
3. Create subsidiary shapes by connecting the centroid to each end of one edge of the parent shape.
4. Repeat steps 2 and 3 for each new shape.*†
5. Continue until a limit is reached. Choose an iteration of the algorithm whose subsidiary shapes will be left whole.‡

* This algorithm produces a construction in which each edge is shared by exactly two shapes and each edge is continuous—connected to an edge which is connected to an edge, and so on—no matter how dense the mesh becomes.

† If one were to localize the cracking—crack more in one part of the structure than another—one could create patches made up of a higher number of shorter members.

‡ Each iteration contains an exponentially greater number of shapes than the one before it. Each iteration takes an exponentially longer time to process.



*"The term algorithm simply means a series of steps. Today, as modeling, representation and fabrication technologies shift from manual to automated process, this issue of algorithm is pressing precisely because it confronts the design of procedures themselves*." Aranda/Lasch (*Pamphlet Architecture 27: Tooling)*

Shapes are inherently driven by internal geometric properties; by manipulating these properties accessed through the algorithm, the architect is able to exploit certain formal predilections, allowing the liquefaction and manipulation of geometry.

The biggest challenge architecture faces when dealing with algorithms lies in the necessity to establish clear, concise and coherent rules that can be used to manipulate geometry. It is only once these rules are established that the task of designing can begin.

This assignment asks students to engage algorithms from both graphic and computational perspectives. By producing a graphic recipe that reduces complex geometry to a collection of *points, planes and vectors,* students will develop a diagrammatic pseudo-code. This pseudo-code will in turn be used to build a parameterized Grasshopper Component that will dynamically populate a fluid surface.

**This assignment is split into two parts,** with *Part A* being completed prior to Wednesday's tutorial. *Part B* will be handed out in tutorial and will have a final due date of 11:59 pm Sunday October 29, 2017.

**<u>Part A: Component Development</u>**

1. Design a graphic pseudo code for the production of a parametric component that can be manipulated through a minimum of three parameters. The pseudo code will illustrate how the geometry can be understood through hierarchical relationships between points, planes and vectors.

**Deliverables Part A:**

- Compile a graphic representation of your component, along with the graphic pseudo-code and diagrams that illustrate how parametric changes produce difference within the component into an 11” X 17” board. Post this board to your blog by 8:00 am Wednesday October 25, 2017.

**For the Tutorial:**

It is expected that each student will have Grasshopper 3D loaded and running on their computers for the tutorial on Wednesday October 25, 2017 (the latest release of Grasshopper can be downloaded at http://www.grasshopper3d.com/ ). It is also **strongly** encouraged that students familiarize themselves with the Grasshopper interface and the Grasshopper Primer before the tutorial (The Grasshopper Primer can be accessed at http://grasshopperprimer.com/en/index.html ).

The best way to learn Grasshopper is through practice. You will not be able to effectively utilize this extremely powerful and useful tool simply through tutorials and the instructions of others – it requires effort, patience and getting your hands dirty. To this, you should try to have very specific questions for your TA’s during tutorials. It is encouraged that all troubleshooting be undertaken through a process of sketching and understanding the problem through geometry (specifically points, planes and vectors). The importance of sketching to understand problems in algorithmic thinking cannot be over emphasized. Sketching leads to intention and intention is necessary to successfully execute any algorithm.