Koch Surface Designer

Sam Sagan

December 18, 2014

Abstract

Koch curves are fractals formed with self-repeating alterations to line segments. Every line segment is replaced by some formation of smaller line segments, and then repeated. I extend this to three dimentions in what I call *Koch surfaces*. My Koch surface starts with an object with congruent, regular polygon sides. What I call a *fractal pattern* is then applied to each of the regular polygon faces. Users of my code define their fractal pattern in real time. Due to the recursive nature of these fractals, the fractal pattern built on n-gons must also consist of n-gons.

Code Overview: Built in c++ using OpenGL and GLUT, my code draws shapes and lines to the screen, illuminated by internal glut lighting sources. The fractal itself is stored in a tree structure, which is generated, edited, and read recursively. Every shape that is drawn to the screen is stored in an instance of my Polygon class, and all three-dimentional locations are stored in my Vector class.

Files and Classes: This provides a file/class specific overview of functionality.

- **main.cpp:** Initializes GLUT windows and rendering, registers display and event-handling callbacks, and starts the game-loop.
- **class ShapeTree:** Stores polygons in a tree structure used to build and draw fractals. Templated to work with any primitive, numerical data type and polygons of any number of sides. Also contains the fractal pattern's parameters and draws the gui accordingly.
- **class Vector:** Represents a three-dimentional vector; storing generic primitive, numerical data types as the coordinates. Useful algebraic vector functions are included in the implementation.

- **class Polygon:** Stores instances of Vector as vertices of a flat, yet three-dimentionally existing polygon.
- **class Singleton:** Uses a singleton pattern to maintain at most one instance of the class at any time. This allows for storage of useful variables and functions pertaining to event-handling, window indexing, and camera angle. A pointer to the sole instance of ShapeTree is stored here as well.

Breaking down the ShapeTree: The tree structure of the ShapeTree class is the backbone of this code. Here I explain this class in greater detail. The building block of the tree is the *ShapeNode* struct. Each ShapeNode contains a Polygon shape, a std::vector of child ShapeNode pointers, and a level. As the tree builds outward, ShapeNodes of the outer level have an empty vector of children.

As one would expect, the major public functions that act on the tree are addLevel(), removeLevel(), and draw(). The overall functionalities of these are clear from their names. Actual implementation is slightly less intuitive. Each has a private helper function that takes a pointer to a ShapeNode as its parameter. These helper functions are the recursive functions that act on the ShapeNode which was pointed to, and then call themselves on all of that ShapeNode's children.

addLevel() only call the recursive helper function to add on to an existing base object, however. This is generated with the private addBaseLevel() function which hard codes the tetrahedron and cube that you see upon execution of the tri_koch and quad_koch executables respectively. Based on the value of private member variable *max_level*, addLevel() calls the suitable helper function.

Many member functions and variables are tasked with managing the fractal pattern and accompanying gui. The fractal pattern is stored in two private arrays. Array *extrusion_edges* divides each edge into segments that define smaller sections of the face. These are to be extruded/stellated by an amount proportional to the square root of the section's area multiplied by the corresponding value in array *extrusion_heights*. Users may edit only one of the fractal pattern parameters at any time, and that parameter index is stored in private variable *uiParam*; toggled with public nextUIparam() and prevUIparam(). The previously described arrays come with public functions to set and get. These account for user error, preventing negative extrusions and out-of-bounds edge divisions. Public member function drawUI() creates the visualization of the fractal parameters in the lower-right corner.

The point of entry into the tree is through its root node; stored in private member ShapeNode pointer *root*. The root has a default Polygon for its shape, which has zero area. Its children are the ShapeNodes that make up the base tetrahedron or cube; this is handled in addBaseLevel(). The root's level is zero, making the base level one.

Other functions are the constructor, destructor, copy constructor, and assignment operator. These all operate as expected; some using recursive helper functions clear(ShapeNode * subroot) and copy(const ShapeNode * subroot). These are all necessary, as memory is dynamically allocated within the ShapeTree class.

Conclusions and Next Steps: The original plan included both robust fractal design capabilities and numerical convergence testing. The fractal design is user-friendly and powerful. I was able to repeat mentor Yuliya Semibratova's findings of simplest-form tetrahedral convergence to a cube. Yuliya also brought up the question of simplest-form cube to octohedron convergence. While viewing this case along any one Cartesian axis strongly suggests this convergence, looking at the full three-dimentional model does not make a visually compelling case for this supposed covergence.

Next steps in fractal design could be allowing for higher-order regular polyheda base figures, or even breaking out of the regular polyhedra mold entirely. Another option to explore would be negative (inward) extrusions.

Next steps in convergence testing would fit a test shape to a high-level koch surface and return a numerical convergence score based on average distance between the test shape and surface at random, discrete points on the surface.

Next steps in general program quality would allow for smooth rotation and enhanced user interface.

Overall, this project was fun to work on and excellent practice in generic programming, tree structures, memory management, makefiles, and the singleton pattern. Thank you.

References

[1] Semibratova, Yuliya. *Higher-Dimentional Koch Curves* (2006). new.math.uiuc.edu/math198/MA198-2013/semibra2/