Andrew Barba
Computer Graphics Fall 2015
December 16th, 2015

Efficiently Animating with Core Animation

The main goal of the project was to dive deeper in the Core Animation API and push it to it's limits. CoreAnimation is the framework that powers much of the animations built into iOS; from smooth scrolling, to swiping, to basic physics. The API allows a programmer to specify animation logic in Swift or Objective-C (on the CPU), and the system will run the animations on the GPU (presumably in OpenGL).

In order to push the framework to it's limits I knew I would need to manage a lot of layers at one time. Because of this, a particle system seemed like the perfect thing to conduct experiments. My project involved creating 4 different particle systems and comparing their frame rates and CPU usage.

Processing

Similar to Homework 7, this system runs entirely on the CPU and updates layers in a loop every "tick"

Swift - Single Threaded

This system is essentially a port of the Processing version running in a single thread on the CPU.

Swift - Multi-Threaded

This system uses multiple threads to compute paths for particles and passes the paths to Core Animation for rendering on the GPU.

Swift - OpenGL

This system uses a built in class in UIKit. I built a wrapper around the class to keep an identical API to the other systems.

After gathering results it is clear that Core Animation has limits and was not meant for these types of graphics applications. The single threaded implementations capped out at about 5,000 layers. The Core Animation system capped out at 10,000 layers. That's not to say it is a bad framework, simply that the design goal of CoreAnimation is best suited elsewhere.