`import CoreAnimation`

...

A look at efficiently animating in Core Animation

Research Goal

1. How to efficiently animate layer-backed views on iOS and Mac OS X.
2. Understand threads on the CPU vs. GPU. Threading Best practices.
3. How the CPU and GPU work together. Efficient scrolling and cell reuse.
4. Building a Core Animation backed Particle system. Stress testing.

Project Goal

1. Explore CPU bound particle systems.
2. Build 3 particle systems in: Processing, Swift (manual), Swift (CoreAnimation)
3. Stress test each particle system. Max out particles with acceptable frame rate.
4. Compare to a raw GPU particle system (there really is no comparison...)

What's been done?

1. Basic Particle System in Processing. Configurable.
2. Basic Particle System in Swift. Manually emit and render particles.
3. Threaded Particle System in Swift. Manually emit, render in Core Animation.

What's left?

1. ~~Stress test particle systems.~~
2. ~~Gather and analyze results.~~
3. Write my research paper.

Demo Recap

Most of my presentation consisted of demoing my project in Xcode. In my demo I showed implementations of 3 different particle systems. Each demo produced similar looking systems but were implemented with different threading models.

Basic Particle System: Does all operations on the main thread. Loops through all particles and moves them every “tick”.

Threaded Particle System: Uses background threads to pre-compute particle paths and then passes those paths to CoreAnimation on the main thread

OpenGL Particle System: built into iOS, best performance.