HUYGENS

an odd kind of sympathy

Huygens is a library of tools for manipulating and synthesizing sound. It depends on `PortAudio` to provide access to audio hardware and on `RtMidi` for processing MIDI messages.

Class Hierarchy

a. `Wave<T>`: a wavetable. Initialized by passing a lambda of a phase (by default in $[0, 1)$). Supports different forms of interpolated lookup.

b. `Oscillator<T>`: stores a frequency and a phase. The `tick()` method updates the phase according to the frequency. Also supports phase- and frequency-modulation, with an adjustable stiffness coefficient for smoothing sudden changes in these parameters.

c. `Synth<T>`: inherits from `Oscillator<T>`; also stores a pointer to a `Wave<T>`. A `Synth<T>` object has an `operator()` function which returns the wavetable evaluated at the current phase.

d. `Filter<T>`: initialized with a list of feedback and feedforward coefficients, $\{a_j\}_{j=0}^N, \{b_j\}_{j=1}^N$. Given a signal $\{x_n\}_{n\in\mathbf{Z}}$, an object of the `Filter<T>` class produces the signal $\{(Tx)_n\}_{n\in\mathbf{Z}}$ satisfying the recurrence

$$(Tx)_n = \sum_{j=0}^{N} a_j x_{n-j} - \sum_{j=1}^{N} b_j (Tx)_{n-j}.$$

Has an `operator(T sample)` function, which takes a new sample as input and returns a filtered sample. Needs a call to `tick()` at sample rate.

Also supports initialization by a list of poles and zeros.

e. `Buffer<T>`: circular buffer object; internally, stores an array of some length and an "origin" position which is advanced with `tick()`. Member functions `write` and `accum` modify the array at the origin; `operator(T position)` looks up the contents of the buffer `position` samples in the past.

f. `Sinusoids<T>`: a bank of sinusoids in a series over some fundamental, with harmonicity and decay parameters. Supports sample rate modulation of the decay, harmonicity, and fundamental frequency.

g. `Particle`, `Spring`, `Gravity`: classes for discrete Newtonian physics, currently in one dimension; a `Particle` has mass, position and velocity, and may accumulate forces with calls to `pull(double force)`. Objects of the class are initialized with a drag coefficient (the user supplies a time in seconds for relaxation from velocity 1). The `Spring` and `Gravity` classes couple two particles; in the case of the first, a call to `tick()` pulls on the particles with a force proportional to their distance; in that of the second, with a force proportional to the product of their masses and inversely proportional to their distance squared.

h. `Polyphon<T>`: an ensemble of particles whose positions set the frequencies of an additive synthesizer. A "note" is a series of particles (whose positions represent pitches, not frequencies), coupled with `Spring` objects to enforce some harmonicity. Particles from distinct notes are coupled by `Gravity` objects, so that when polyphony is present, overtones align with one another. Needs a call to `tick()` to update the phases of its oscillators, and occasional (not necessarly sample-rate) calls to `physics()` to update the particles.

i. `Polyres<T>`: identical to `Polyphon<T>`, except that particle positions are used to set the center frequencies of a bank of resonant biquad filters.

j. `Granulator<T>`: granular synthesizer. Initialized with a pointer to a `Wave<T>`, the windowing function, and a pointer to a `Buffer<T>`, the buffer from which to read grains. The member function `request` attempts to allocate a new grain with some parameters (delay from current readhead of the buffer, grain length, playback speed, gain). Needs a call to `tick()` at audiorate. The `operator()` member function gets the current sample for output.