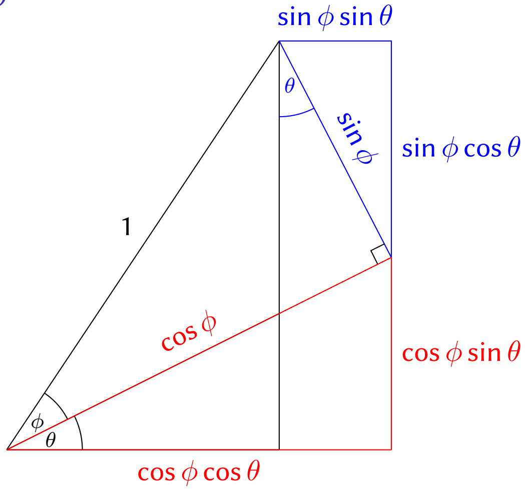
Audio-Visual Computing

Christophe Rhodes
c.rhodes@gold.ac.uk

Winter 2015

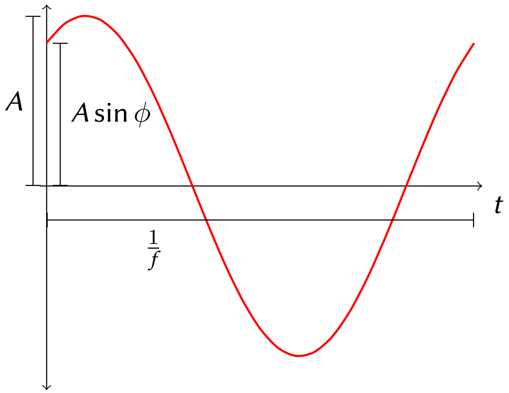
$\theta + \phi$



$$\sin(\theta+\phi) = \sin\theta\cos\phi + \cos\theta\sin\phi$$

$$\cos(\theta+\phi) = \cos\theta\cos\phi - \sin\theta\sin\phi$$

Sinusoidal oscillation



$$y = A \sin(2\pi f t + \phi)$$

Beats

$$\sin(\theta+\phi) = \sin\theta\cos\phi + \cos\theta\sin\phi$$
$$\sin(\theta-\phi) = \sin\theta\cos\phi - \cos\theta\sin\phi$$

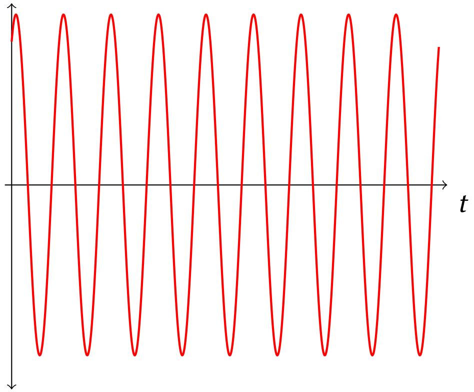
so

$$\sin(\theta+\phi) + \sin(\theta-\phi) = 2\sin\theta\cos\phi$$

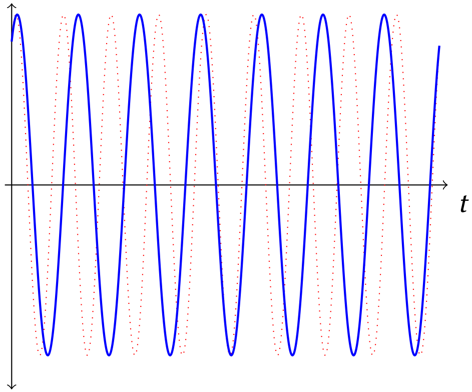
so

$$\sin(\alpha) + \sin(\beta) = 2\sin\left(\frac{\alpha+\beta}{2}\right)\cos\left(\frac{\alpha-\beta}{2}\right)$$

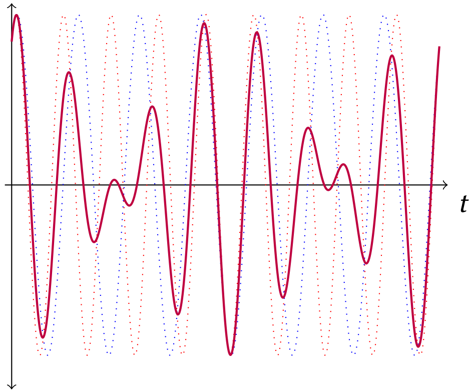
Beats



Beats



Beats



Octave equivalence

```
import ddf.minim.*;

Minim minim;
Shepard s;

void setup() {
  minim = new Minim(this);
  s = new Shepard(440);
  AudioOutput out = minim.getLineOut(Minim.MONO, 2048);
  out.addSignal(s);
}

void stop() {
  minim.stop();
  super.stop();
}

void draw() {
  s.sets(pow(2, (millis() % 5000)/5000.0));
}
```

Octave equivalence

```
abstract class MonoSignal {
  abstract void generate(float [] buf);
  void generate(float [] left, float [] right) {
    this.generate(left);
    for(int i = 0; i < right.length; i++) {
      right[i] = left[i];
    }
  }
}
```

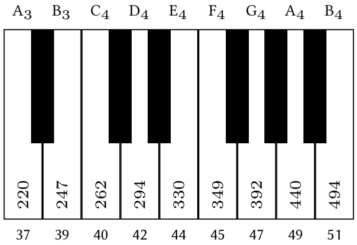
Octave equivalence

```
abstract class SineBuf extends MonoSignal {
  float p = 0;
  float sinebuf(float [] buf, float f, float a, float p) {
    return sinebuf(buf, f, a, p, true);
  }
  float sinebuf(float [] buf, float f, float a, float p, boolean zero) {
    for (int i = 0; i < buf.length; i++) {
      if (zero)
        buf[i] = 0;
      buf[i] += a*sin(2*PI*f*i / 44100 + p);
    }
    p += 2*PI*f*buf.length/44100;
    return p - 2*PI*floor(p/(2*PI));
  }
}
```

Octave equivalence

```
class Shepard extends SineBuf implements AudioSignal {
  float s = 1;
  float frequency;
  Shepard(float f) { frequency = f; }
  float amplitude(float f) {
    return 1.0/2000*pow(max(4-abs(log(f/1000.0)),0),4);
  }
  void generate(float [] buf) {
    float newp = sinebuf(buf, frequency*s, amplitude(frequency*s), p);
    for(int i = -5; i < 5; i++) {
      if(i == 0) continue;
      float f = frequency*s*pow(2.0,i);
      sinebuf(buf, f, amplitude(f), p*pow(2.0,i), false);
    }
    p = newp;
  }
  void sets(float _s) { if(_s<s) p = p/2; s = _s; }
}
```

12-tone Equal Temperament



Western tonal music:

- 12 “semitones” to the “octave”
- multiply or divide frequencies by $2^{\frac{1}{12}}$ (1.05946...)

Synthesis

Additive synthesis

- sine tones are boring;
- change tone quality (*timbre*) by adding other sine tones;

Synthesis

Additive synthesis

- sine tones are boring;
- change tone quality (*timbre*) by adding other sine tones;
- can make any shape of wave this way

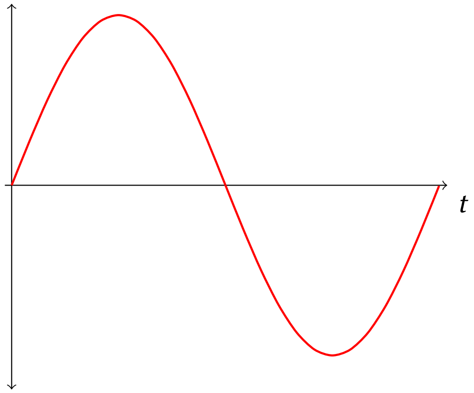
Synthesis

Additive synthesis

```
class AddSine extends SineBuf implements AudioSignal {
  float [] harmonics; float fundamental; float p = 0;
  AddSine() { }
  AddSine(float f, float [] hs) { fundamental = f; harmonics = hs; }
  void generate(float [] buf) {
    float newp = sinebuf(buf, fundamental, harmonics[0], p);
    for(int i = 1; i < harmonics.length; i++)
      sinebuf(buf, (i+1)*fundamental, harmonics[i], p*(i+1), false);
    p = newp;
  }
  float getf() { return fundamental; }
  void setf(float f) { fundamental = f; }
  float [] geth() { return harmonics; }
  void seth(float [] hs) { harmonics = hs; }
  void draw() {
    stroke(0); smooth();
    float [] b = new float[floor(88200/fundamental)];
    sinebuf(b, fundamental, harmonics[0], 0);
    for(int i = 1; i < harmonics.length; i++)
      sinebuf(b, (i+1)*fundamental, harmonics[i], 0, false);
    for(int i = 0; i < b.length; i++) {
      b[i] = (b[i] + 1)*height/2;
      if(i > 0)
        line((i-1)*width/(float)b.length, b[i-1], i*width/(float)b.length, b[i]);
    }
  }
}
```

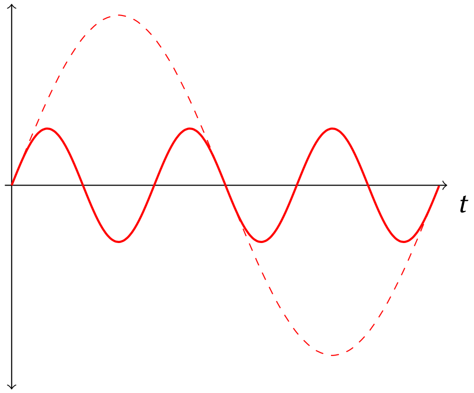
Synthesis

Additive synthesis



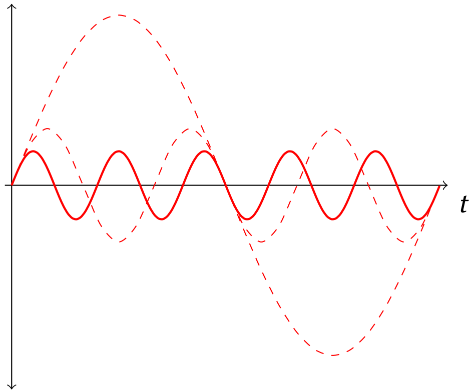
Synthesis

Additive synthesis



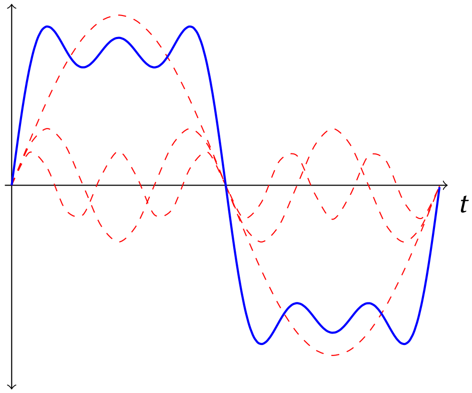
Synthesis

Additive synthesis



Synthesis

Additive synthesis



Square wave:

$$\sum_{k=0}^{\infty} \frac{1}{2k+1} \sin(2\pi(2k+1)ft)$$

Synthesis

Additive synthesis

```
class Square extends AddSine {
  Square(float f) { fundamental = f; setn(1); }
  void setn(int n) {
    float [] hs = new float[n];
    for(int i = 0; i < n; i++)
      hs[i] = (i % 2 == 0) ? 0.9/(i+1) : 0;
    seth(hs);
  }
}
```

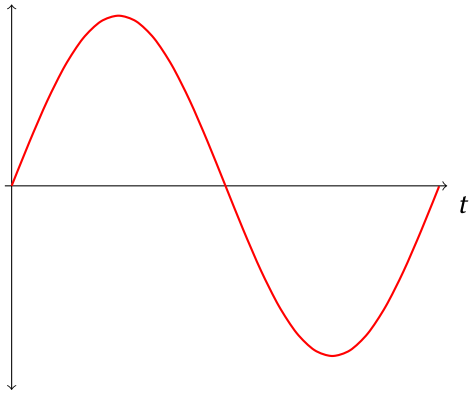
Synthesis

Additive synthesis

```
import ddf.minim.*;

Minim minim;
Square s;

void setup() {
  size(800,600); minim = new Minim(this);
  s = new Square(440);
  AudioOutput out = minim.getLineOut(Minim.MONO, 1024);
  out.addSignal(s);
}
void stop() {
  minim.stop(); super.stop();
}
void draw() {
  background(255); s.draw();
}

void keyPressed() {
  if(key == CODED) {
    switch(keyCode) {
    case UP: s.setf(s.getf() * pow(2,1.0/12)); break;
    case DOWN: s.setf(s.getf() / pow(2,1.0/12)); break;
    case LEFT: s.setn(max(1, s.harmonics.length-1)); break;
    case RIGHT: s.setn(min(90, s.harmonics.length+1)); break;
    }
  }
}
```

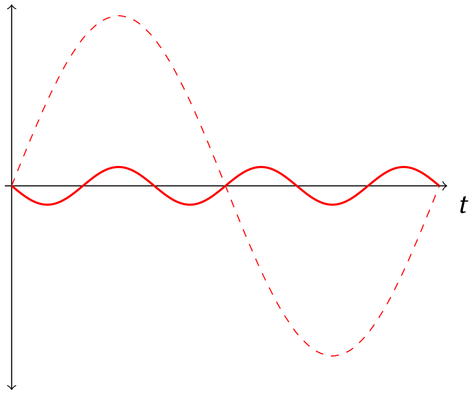
Synthesis

Additive synthesis



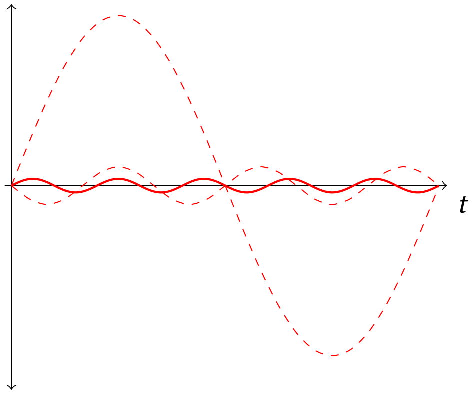
Synthesis

Additive synthesis



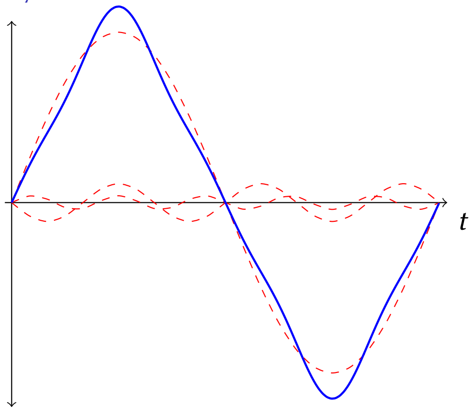
Synthesis

Additive synthesis



Synthesis

Additive synthesis



Triangular wave:

$$\sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)^2} \sin(2\pi(2k+1)ft)$$

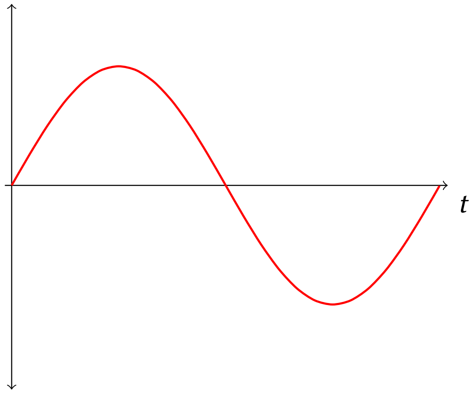
Synthesis

Additive synthesis

```
class Triangle extends AddSine {
  Triangle(float f) { fundamental = f; setn(1); }
  void setn(int n) {
    float [] hs = new float[n];
    for(int i = 0; i < n; i++)
      hs[i] = (i % 2 == 0) ? 0.7*pow(-1,i/2)/pow((i+1),2) : 0;
    seth(hs);
  }
}
```

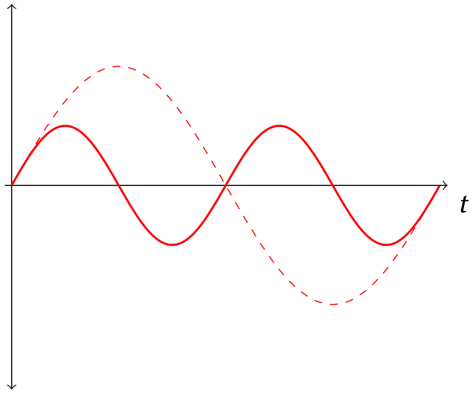
Synthesis

Additive synthesis



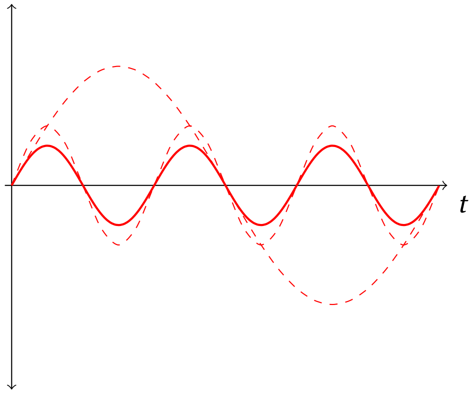
Synthesis

Additive synthesis



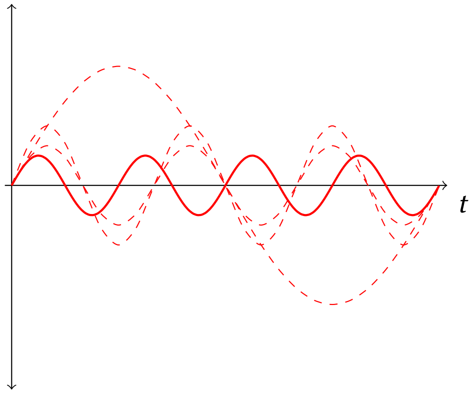
Synthesis

Additive synthesis



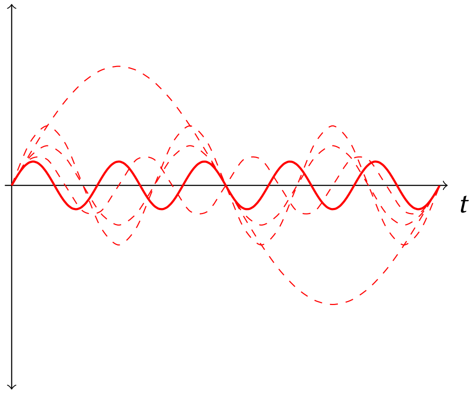
Synthesis

Additive synthesis



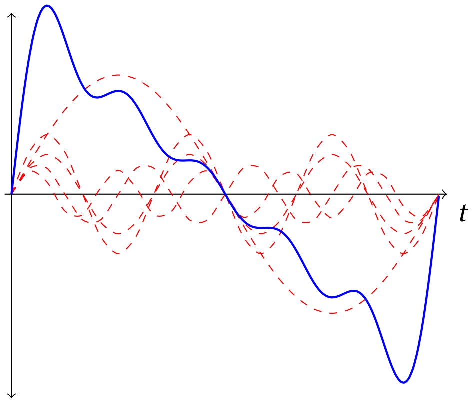
Synthesis

Additive synthesis



Synthesis

Additive synthesis



Sawtooth wave:

$$\sum_{k=0}^{\infty} \frac{1}{k} \sin(2\pi kft)$$

Synthesis

Additive synthesis

```
class Sawtooth extends AddSine {
  Sawtooth(float f) { fundamental = f; setn(1); }
  void setn(int n) {
    float [] hs = new float[n];
    for(int i = 0; i < n; i++)
      hs[i] = 0.5/(i+1);
    seth(hs);
  }
}
```

Synthesis

Phase modulation

Also known as Frequency Modulation (and hence “FM synthesis”)

$$y = A \sin\left(2\pi f_c t + A_m \sin(2\pi f_m t)\right)$$

Synthesis

Phase modulation

```
class FM extends MonoSignal implements AudioSignal {
  float fc; float fm; float am; int ii = 0;
  FM(float f, float m, float a) { fc = f; fm = m; am = a; }
  int fmbuf(float buf[]) {
    for (int i = 0; i < buf.length; i++)
      buf[i] = sin(2*PI*fc*(i+ii)/44100 + am*sin(2*PI*fm*(i+ii)/44100));
    return ii + buf.length;
  }
  void generate(float buf[]) {
    ii = fmbuf(buf);
  }
  float getfm() { return fm; } void setfm(float f) { fm = f; }
  float getam() { return am; } void setam(float f) { am = f; }
  void draw() {
    stroke(0); smooth(); fill(0,80,80); textSize(20);
    float [] b = new float[floor(20*44100/fc)];
    fmbuf(b);
    for(int i = 0; i < b.length; i++) {
      b[i] = (b[i] + 1)*height/2;
      if(i > 0)
        line((i-1)*width/(float)b.length, b[i-1], i*width/(float)b.length, b[i]);
    }
    text("Fc:␣" + floor(fc), 20, 20);
    text("Fm:␣" + floor(fm), 20, 50);
    text("Am:␣" + floor(am), 20, 80);
  }
}
```

Synthesis

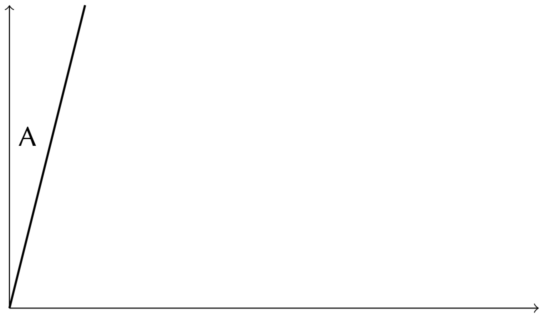
Phase modulation

```
import ddf.minim.*;
Minim minim; FM s; boolean drawNeeded;

void setup() {
  size(800,600); minim = new Minim(this);
  s = new FM(440,1,1);
  AudioOutput out = minim.getLineOut(Minim.MONO, 1024);
  out.addSignal(s); drawNeeded = true;
}
void stop() {
  minim.stop(); super.stop();
}
void draw() {
  if(drawNeeded) {
    background(255); s.draw(); drawNeeded = false;
  }
}

void keyPressed() {
  if(key == CODED) {
    switch(keyCode) {
    case UP: s.setam(s.getam() + 1); break;
    case DOWN: s.setam(s.getam() - 1); break;
    case LEFT: s.setfm(s.getfm() - 1); break;
    case RIGHT: s.setfm(s.getfm() + 1); break;
    }
    drawNeeded = true;
  }
}
```

Synthesis

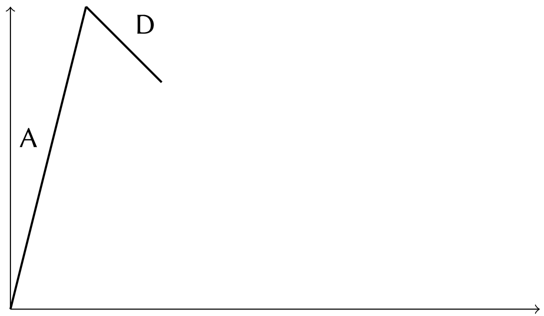
Amplitude envelopes

ADSR envelope:



Synthesis
Amplitude envelopes

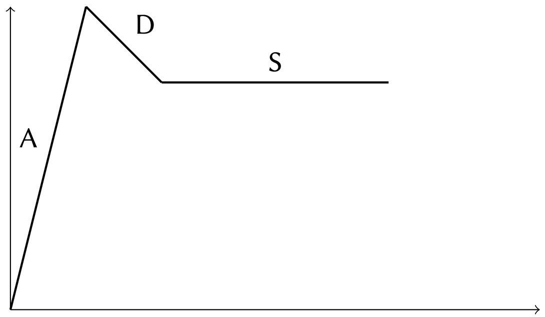
ADSR envelope:



Synthesis

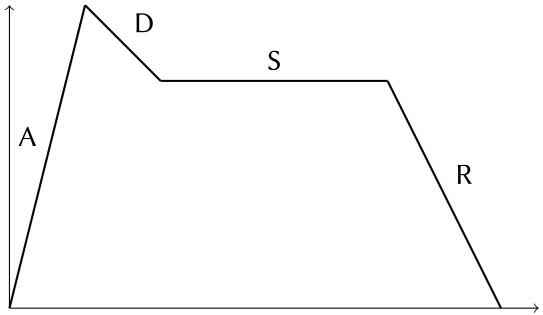
Amplitude envelopes

ADSR envelope:



Synthesis
Amplitude envelopes

ADSR envelope:



Synthesis

Amplitude envelopes

```
import ddf.minim.*;
import ddf.minim.ugens.*;

Minim minim;
ADSR adsr;

void setup() {
  minim = new Minim(this);
  AudioOutput out = minim.getLineOut(Minim.MONO, 1024);
  Oscil osc = new Oscil(440, 0.1, Waves.SQUARE);
  adsr = new ADSR(1, 0.04, 0.3, 0.4, 0.5);
  osc.patch(adsr).patch(out);
}

void draw () {
  if((millis() % 5000) / 1000 == 0) adsr.noteOn();
  if((millis() % 5000) / 1000 == 1) adsr.noteOff();
}
```

Effects

Systems

- general name
- takes a signal and transforms it, producing a new signal
  - microphone membrane
  - electrical circuit
  - loudspeaker

Effects
Delay

- basic building block of digital effects
- introduces latency

Effects

Delay

```
import ddf.minim.*;
import ddf.minim.ugens.*;

Minim minim;
ADSR adsr;

void setup() {
  minim = new Minim(this);
  AudioOutput out = minim.getLineOut(Minim.MONO, 1024);
  Oscil osc = new Oscil(440, 0.1, Waves.SQUARE);
  Delay delay = new Delay(0.4,0.6,false,false);
  adsr = new ADSR(1, 0.04, 0.3, 0.4, 0.5);
  osc.patch(adsr).patch(delay).patch(out);
}

void draw () {
  if((millis() % 5000) / 1000 == 0) adsr.noteOn();
  if((millis() % 5000) / 1000 == 1) adsr.noteOff();
}
```

Effects

Echo

```
import ddf.minim.*;
import ddf.minim.ugens.*;

Minim minim;
ADSR adsr;

void setup() {
  minim = new Minim(this);
  AudioOutput out = minim.getLineOut(Minim.MONO, 1024);
  Oscil osc = new Oscil(440, 0.1, Waves.SQUARE);
  Delay delay = new Delay(0.4,0.6,true,true);
  adsr = new ADSR(1, 0.04, 0.3, 0.4, 0.5);
  osc.patch(adsr).patch(delay).patch(out);
}

void draw () {
  if((millis() % 5000) / 1000 == 0) adsr.noteOn();
  if((millis() % 5000) / 1000 == 1) adsr.noteOff();
}
```