

# Creative Computing II

Christophe Rhodes  
c.rhodes@gold.ac.uk

Autumn 2009, Tuesdays, 10:00–15:00  
Winter 2010, tbc

# Sound

What is sound?

- ▶ *longitudinal pressure wave.*

# Sound

What is sound?

- ▶ *longitudinal pressure wave.*

**longitudinal:**

- ▶ displacement from equilibrium is in direction of wave propagation;
- ▶ distinct from **transverse** (perpendicular) waves:
  - ▶ water waves;
  - ▶ light waves.

# Sound

What is sound?

- ▶ *longitudinal pressure wave.*

**pressure:**

- ▶ generally, a force applied to a surface;
- ▶ in atmosphere, weight of air;
- ▶ measured in pascals;  $1 \text{ atm} \sim 100,000 \text{ Pa} = 100 \text{ kPa}$ .

# Sound

What is sound?

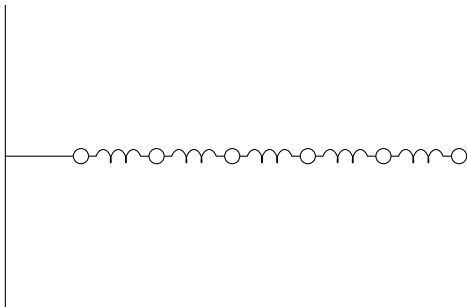
- ▶ *longitudinal pressure wave.*

Ear detects, amplifies and interprets incoming pressure waves.

# Sound

## Waves

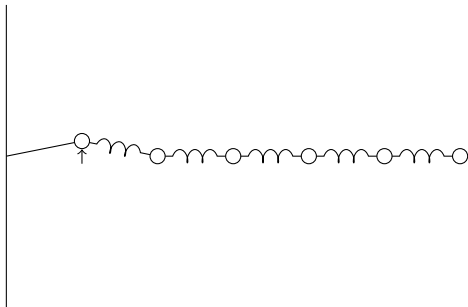
Balls and springs model of matter:



# Sound

## Waves

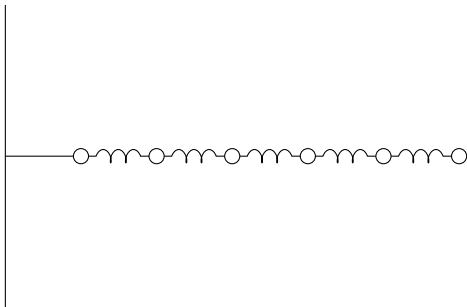
Balls and springs model of matter:



# Sound

## Waves

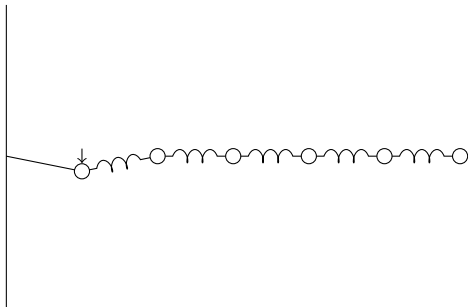
Balls and springs model of matter:



# Sound

## Waves

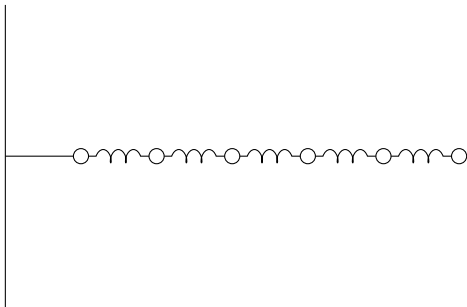
Balls and springs model of matter:



# Sound

## Waves

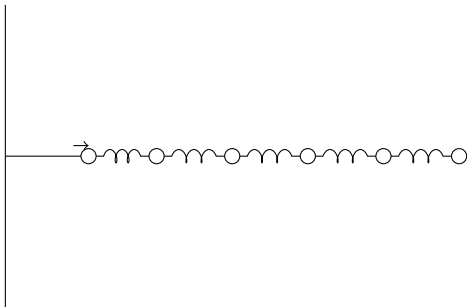
Balls and springs model of matter:



# Sound

## Waves

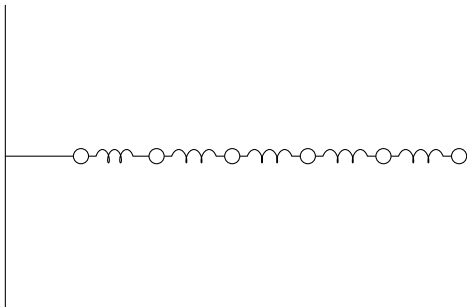
Balls and springs model of matter:



# Sound

## Waves

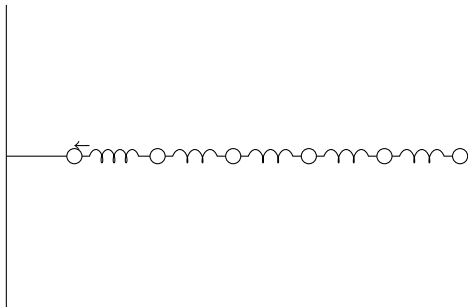
Balls and springs model of matter:



# Sound

## Waves

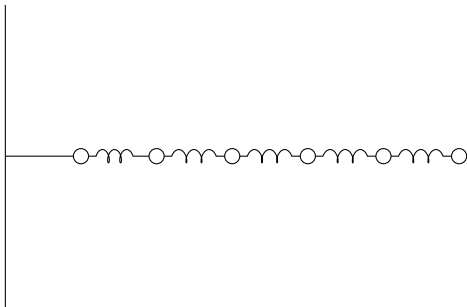
Balls and springs model of matter:



# Sound

## Waves

Balls and springs model of matter:



# Sound

## Wave signal properties

Wave signals have a number of properties:

- ▶ *frequency*;
- ▶ *amplitude*;
- ▶ *phase*.

# Sound

## Wave signal properties

Wave signals have a number of properties:

- ▶ *frequency*;
- ▶ *amplitude*;
- ▶ *phase*.

$$x(t) = A \sin(2\pi f t + p)$$

**frequency:**

- ▶ number of cycles per second (hertz);
- ▶ audible sound waves in range 20Hz – 20kHz;
- ▶ (compare light waves:  $\sim 5 \times 10^{12}$ Hz).

# Sound

## Wave signal properties

Wave signals have a number of properties:

- ▶ *frequency*;
- ▶ *amplitude*;
- ▶ *phase*.

$$x(t) = \mathbf{A} \sin(2\pi ft + p)$$

**amplitude:**

- ▶ how large is the displacement;
- ▶ (how much signal is present);

# Sound

## Wave signal properties

Wave signals have a number of properties:

- ▶ *frequency*;
- ▶ *amplitude*;
- ▶ *phase*.

$$x(t) = A \sin(2\pi ft + \mathbf{p})$$

**phase:**

- ▶ the initial ( $t = 0$ ) position of the wave.
- ▶ alternative to representation of the signal as a mixture of sin and cos:
  - ▶  $\sin(A + B) = \sin(A) \cos(B) + \cos(A) \sin(B)$
  - ▶  $\sin(2\pi ft + p) = \sin(2\pi ft) \cos(p) + \cos(2\pi ft) \sin(p)$

# Sound Perception

## Loudness

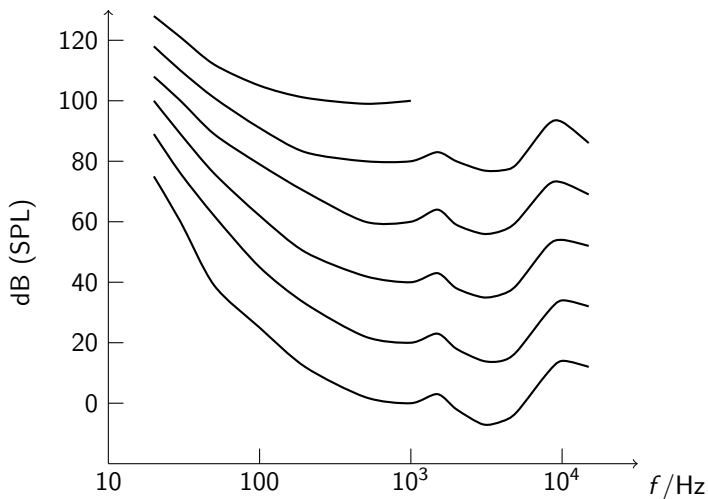
Related to amplitude, but not linearly:

- ▶ approximate logarithmic dependency
  - ▶ decibel scale: Sound Pressure Level;
  - ▶  $L_p = 20 \log_{10} \left( \frac{p}{p_0} \right)$ .
- ▶ perceived loudness also depends on frequency.
- ▶ (ref. CIE LAB colour space for light)

# Sound Perception

## Loudness

Equal-loudness curves (ISO 226:2003)



# Sound Perception

## Source Location

How do we know where a sound is coming from? Two principal mechanisms:

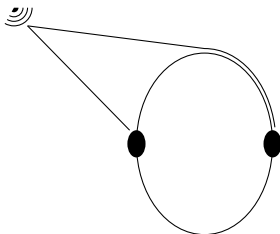
- ▶ time delay (low-frequency sounds);
- ▶ attenuation (high-frequency sounds).

# Sound Perception

## Source Location

How do we know where a sound is coming from? Two principal mechanisms:

- ▶ time delay (low-frequency sounds);
- ▶ attenuation (high-frequency sounds).



time delay:

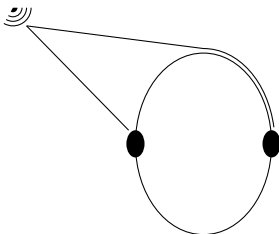
- ▶ difference in path length from source to each ear;
- ▶ gives rise to *phase difference* between ears.

# Sound Perception

## Source Location

How do we know where a sound is coming from? Two principal mechanisms:

- ▶ time delay (low-frequency sounds);
- ▶ attenuation (high-frequency sounds).



attenuation:

- ▶ diffraction around head causes energy to be lost;
- ▶ gives rise to *amplitude difference*.

# Octave

## Introduction

Octave: high-level language for numerical computations:

- ▶ part of the GNU project;
- ▶ dealing with *signals*:
  - ▶ construction;
  - ▶ manipulation;
  - ▶ visualization.
- ▶ interactive interface.

# Octave

## Introduction

Octave: high-level language for numerical computations:

- ▶ part of the GNU project;
- ▶ dealing with *signals*:
  - ▶ construction;
  - ▶ manipulation;
  - ▶ visualization.
- ▶ interactive interface.

For our purposes:

- ▶ visualisation of signals;
- ▶ rapid investigation of audio;
- ▶ (next term) linear systems and filters.

# Octave

## Scalars

Two types of scalar:

- ▶ numbers:

- ▶ 3

- ▶ -6

- ▶ 3.1416

- ▶ strings:

- ▶ 'a string'

# Octave

## Scalar Operations

Both *operators* and *functions* operate on scalars:

- ▶ basic mathematical operators:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $^$
- ▶ statistical functions: `min()`, `max()`, `mean()`
- ▶ trigonometric functions: `sin()`, `cos()`, `tan()`

# Octave

## Scalar Operations

Both *operators* and *functions* operate on scalars:

- ▶ basic mathematical operators:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $^$
- ▶ statistical functions: `min()`, `max()`, `mean()`
- ▶ trigonometric functions: `sin()`, `cos()`, `tan()`

Examples:

- ▶  $4.5 + 9.6 * 2$
- ▶  $(4.5 + 9.6) * 2$
- ▶  $(4.5+9.6) ^ 2 / 2$

# Octave

## Variables

Names for values:

- ▶ variables are untyped;
- ▶ values have type;
- ▶ = is assignment.

# Octave

## Variables

Names for values:

- ▶ variables are untyped;
- ▶ values have type;
- ▶ = is assignment.

Examples:

- ▶ `a = -100`
- ▶ `aLongVariableName = 10`

# Octave

## Variables

Names for values:

- ▶ variables are untyped;
- ▶ values have type;
- ▶ = is assignment.

Examples:

- ▶ `a = -100`
- ▶ `aLongVariableName = 10`

Some predefined constants:

- ▶ `pi`
- ▶ `e`

# Octave

## Other programming constructs

Relational operators:

- ▶  $<$ ,  $>$ ,  $\leq$ ,  $\geq$
- ▶  $==$ ,  $\neq$

# Octave

## Other programming constructs

Relational operators:

- ▶ `<`, `>`, `<=`, `>=`
- ▶ `==`, `!=`

Conditionals:

- ▶ `if(...)` ... `endif`
- ▶ `switch ... case {...}` ... `otherwise ... endswitch`

# Octave

## Other programming constructs

Relational operators:

- ▶ `<`, `>`, `<=`, `>=`
- ▶ `==`, `!=`

Conditionals:

- ▶ `if(...)` ... `endif`
- ▶ `switch ... case {...}` ... `otherwise ... endswitch`

Loops:

- ▶ `for(...)` ... `endfor`
- ▶ `while(...)` ... `endwhile`

# Octave

## Other programming constructs

Relational operators:

- ▶ `<`, `>`, `<=`, `>=`
- ▶ `==`, `!=`

Conditionals:

- ▶ `if(...)` ... `endif`
- ▶ `switch ... case {...}` ... `otherwise ... endswitch`

Loops:

- ▶ `for(...)` ... `endfor`
- ▶ `while(...)` ... `endwhile`

Look, ma, no braces!

# Octave

## Scripts and Functions

We can define *scripts* and *functions* to perform calculations:

- ▶ script: no arguments, no results: just side-effects;
- ▶ function: multiple arguments, multiple results, side-effects.

# Octave

## Scripts and Functions

We can define *scripts* and *functions* to perform calculations:

- ▶ script: no arguments, no results: just side-effects;
- ▶ function: multiple arguments, multiple results, side-effects.

```
function f = add(x, y)
    f = x + y;
endfunction
```

```
function [a, b] = fun(c, d)
    a = c + 1;
    b = sin(d);
endfunction
```

# Octave

## Vectors

Vectors are sequences of scalars:

- ▶ `[0 1 2 3 4 5]`
- ▶ `[0:5]`
- ▶ `[0:1:5]`

# Octave

## Vectors

Vectors are sequences of scalars:

- ▶ `[0 1 2 3 4 5]`
- ▶ `[0:5]`
- ▶ `[0:1:5]`

Many of the same operations work on vectors as scalars:

- ▶ `1 + [0:5]`
- ▶ `[0:5] - 2`
- ▶ `3 * [0:5]`
- ▶ `sin([0:5])`

# Octave

## Vectors

Vectors are sequences of scalars:

- ▶ `[0 1 2 3 4 5]`
- ▶ `[0:5]`
- ▶ `[0:1:5]`

Many of the same operations work on vectors as scalars:

- ▶ `1 + [0:5]`
- ▶ `[0:5] - 2`
- ▶ `3 * [0:5]`
- ▶ `sin([0:5])`

but not division (`/`), exponentiation (`^`) or multiplication of vectors by vectors.

# Octave

## Vector indexing

Elements of a vector can be retrieved by *indexing*:

▶ `[0:5](3)`

▶ `sin([0:5])(3)`

# Octave

## Vector indexing

Elements of a vector can be retrieved by *indexing*:

- ▶ `[0:5](3)`

- ▶ `sin([0:5])(3)`

**Note:** although Java and *Processing* arrays are indexed starting from 0, *Octave* arrays are indexed starting from 1.

# Octave

## Vector multiplication

Vectors can be multiplied in two ways:

- ▶ **element-by-element**: the two vectors must have the same dimensions:
  - ▶ `[0:5] .* [5:-1:0]`
  - ▶ error: `[0:5] .* [1:5]`

(also called the Hadamard product)

# Octave

## Vector multiplication

Vectors can be multiplied in two ways:

- ▶ **element-by-element**: the two vectors must have the same dimensions:

- ▶ `[0:5] .* [5:-1:0]`

- ▶ error: `[0:5] .* [1:5]`

(also called the Hadamard product)

- ▶ **matrix**: the two vectors must have the same length and one must be *transposed*:

- ▶ `[0:5] * [5:-1:0]'`

- ▶ `[0:5]'` \* `[5:-1:0]`

- ▶ error: `[0:5] * [5:-1:0]`