OPTION PF2 - Automne 2016
Jean-Paul Roy
L2-Info&Maths, Faculté des Sciences

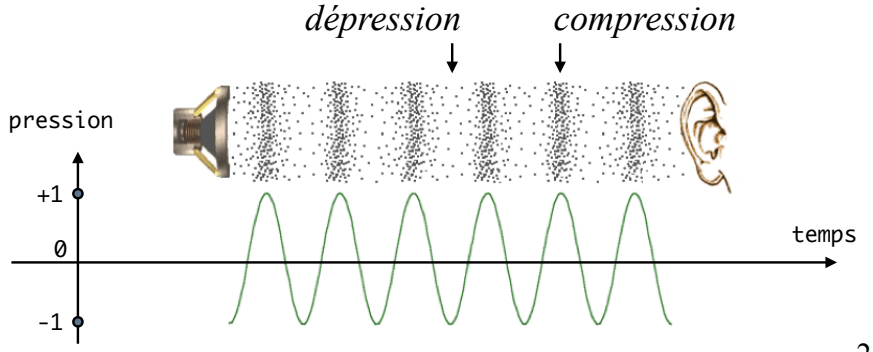
http://deptinfo.unice.fr/~roy

Cours 3

Les objets sonores de Racket

Qu'est-ce qu'un son ?

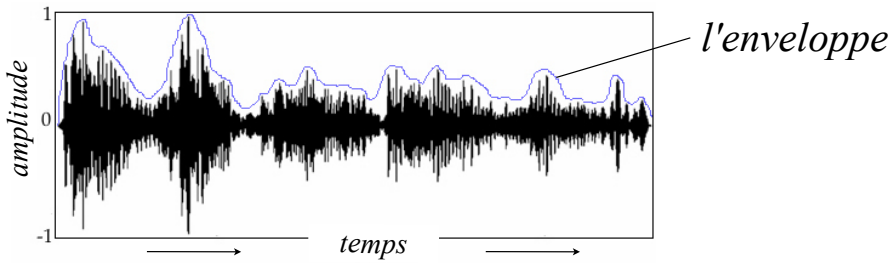
• Un **son** est un signal **ondulatoire** transportant une **variation de la pression d'air** en fonction du temps. L'oreille capture la variation de pression sur le tympan qui transmet cette information à l'oreille interne puis au cerveau, qui l'interprète.



Le son comme fonction du temps

• Un **son** (*sound*) est représenté par une fonction $p=p(t)$ où l'abscisse est le **temps** et l'ordonnée la **pression** d'air (ou **amplitude** du signal).

• La pression peut être positive (compression) ou négative (dépression) suivant le mouvement des particules d'air. La pression nulle correspond au silence.

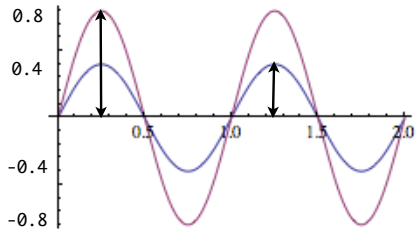


L'amplitude : intensité d'un son

• Le temps étant discrétisé, la figure précédente représente un grand nombre d'**échantillons** (*samples*) du son à intervalles réguliers. Un échantillon isolé ne renseigne pas beaucoup sur le son...

• L'amplitude d'un son gouverne son **intensité**. Le signal $0.8\ sin(\omega t)$ sera perçu deux fois plus fort que le signal $0.4\ sin(\omega t)$.

0.4 0.8



La fréquence : hauteur d'un son

• Un signal sonore $s = s(t)$ est **périodique** s'il se répète à intervalles réguliers, donc si la fonction s est périodique. Par exemple $s = sin(2\pi\omega t)$ est périodique de période $T = 1/\omega$. Sa fréquence est ω.

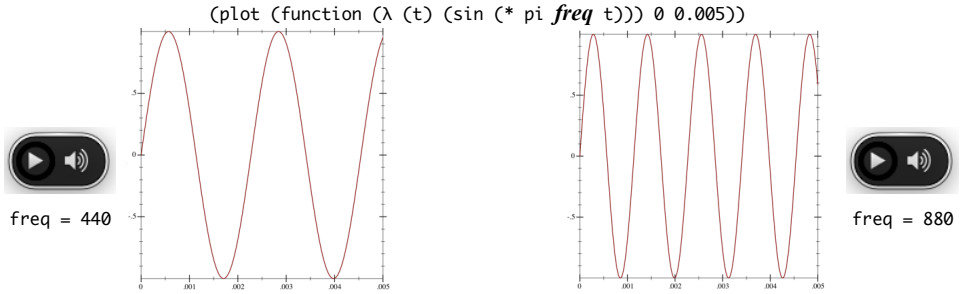
• La fréquence (*pitch*) mesure la **hauteur** du son (grave ou aigu). Le signal $sin(2\pi\ 440\ t)$ a une fréquence de 440 Hz. La fréquence est mesurée en Hertz (Hz = 1/sec).

• La plage de fréquences perceptibles par l'oreille humaine va du grave 20 Hz à l'aigu 20000 Hz (20 Kz).

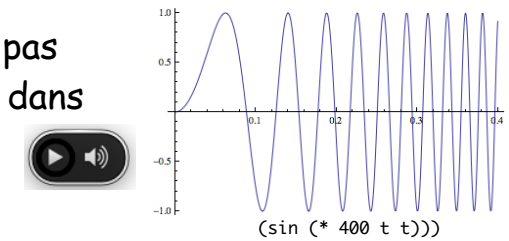
	Basse fréquence	Haute fréquence
	25.5 Hz	4186 Hz
	80 Hz	240 Hz
	140 Hz	500 Hz
	0 Hz	22000 Hz
	20 Hz	20000 Hz

d'où la fréquence d'échantillonage !

• La **fréquence** ω du signal $sin(2\pi\omega t)$ mesure l'écartement des vagues de la sinusoïde, alors que l'**intensité** en mesurait la hauteur.

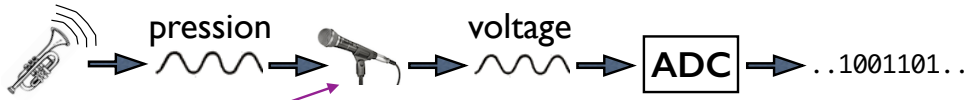


• La fréquence peut ne pas être constante, comme dans un gazouilli (*chirp*) :



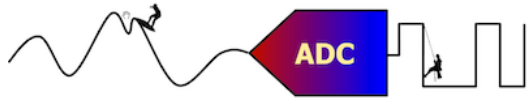
De l'Analogique au ~~Digital~~ Numérique

• Le monde **analogique** est **continu**, ses courbes mathématiques sont lisses. Le monde **numérique** est **discret**, une courbe est remplacée par une suite de points, les échantillons (*samples*).



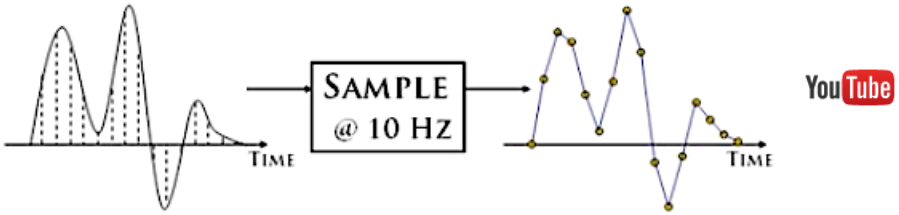
• Le **microphone** convertit une variation de pression d'air en un signal électrique continu : un voltage v = v(t).

• Le **convertisseur** ADC va discrétiser la courbe du voltage.



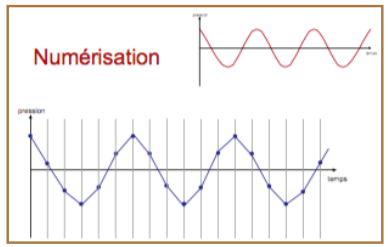
L'échantillonnage (*sampling*)

• Le **convertisseur** ADC va prélever des échantillons sur la courbe du voltage, à une certaine **fréquence d'échantillonnage** F_e, le plus souvent F_e = 44100 Hz.



• Plus la fréquence maximale F du signal à échantillonner est elevée, plus F_e doit être élevée, pour ne pas perdre trop d'information. Le **théorème de Nyquist** précise qu'il faut choisir $F_e \geq 2F_{max}$ pour reconstruire le signal.

• Le son stocké est une suite de nombres donnant l'amplitude du signal sonore. Ces nombres sont sur 8, **16**, 24 bits par ex. suivant la précision voulue.



• Les échantillons sont prélevés (sur un ou plusieurs canaux) à une certaine fréquence $F_e \geq 2\ F_{max}$

• **Occupation mémoire** : mem = nb-sec x nb-octets x F_e

• 10 sec de parole humaine, 11 Khz, 8 bits, mono : 110 Ko

• 1 sec de musique CD, 44.1 Khz, 16 bits, mono : 88 Ko

• *Skyfall* (4'50), 44.1 Khz, 16 bits, stéréo : 49 Mo

Le module rsound de Racket

• Nous utiliserons le module **rsound** de John Clements :

```
(require rsound)
```

ainsi qu'un morceau de musique sur disque dur comme 5th.wav (*Beethoven remixé techno*)…

• Nous travaillerons avec du **.wav 16 bits PCM**, **sans perte** (*lossless*) La production d'un .mp3 si besoin (avec perte, *lossy*) se fait ensuite avec un logiciel de conversion externe.

• L'API de rsound va nous permettre de construire ou transformer des objets sonores, puis les sauver sur disque en .wav.

Lecture d'un fichier-disque .wav

• Une valeur Scheme de type rsound est une structure :

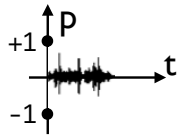
```
(define-struct rsound (data start stop sample-rate))
```

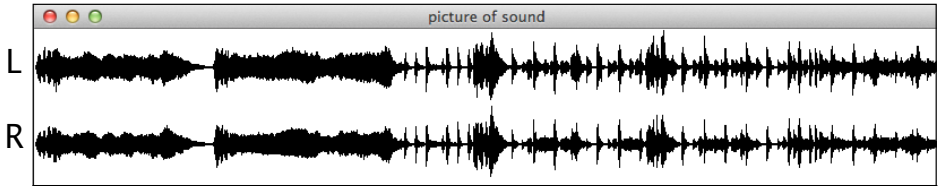
```
> (define foo (rs-read "5th.wav"))
> foo                          ; un son est une struct
#(struct:rsound #<s16vector> 0 1043006 44100)
> (play foo)                   ; the audience is listening
> (stop)                       ; play est asynchrone !
> (rsound-sample-rate foo)     ; fréquence Fe
44100
> (rs-frames foo)              ; nombre d'échantillons
1043006
> (duration foo)               ; durée en secondes, cf TP
23.650929705215418
```

Visualisation d'un son

• On peut visualiser la courbe d'amplitude pour chaque canal :

```
> (require rsound/draw)
> (rs-draw foo)
```





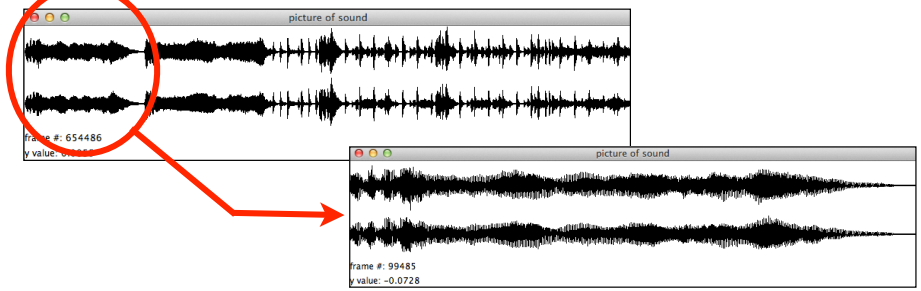
• Accès à l'amplitude dans [-1,1] d'un échantillon en O(1) :

```
> (rs-ith/right foo 100000)        ; l'échantillon
0.12253181554612873      ; n°100000 du canal droit
```


Extraction d'un clip

• On peut extraire un morceau (*clip*) d'un son, en donnant les numéros d'échantillons (*frame*) extrêmes :

```
> (define fooc (clip foo 3000 205000))
> fooc
#(struct:rsound #<s16vector> 3000 205000 44100)
> (rs-draw fooc)
```



Sauvegarde d'un son sur disque

• La fonction `rs-read` permettait de charger sous forme de son un fichier `.wav` 16 bits PCM stéréo.

• La fonction `rs-write` permet inversement de sauver sur disque un objet `rsound` construit en Scheme.

```
> (rs-write fooc "fooc.wav")
```

• L'objet de type `rsound` peut provenir du clip d'un son déjà existant, ou bien d'une suite d'opérations sur plusieurs sons, voire d'un son purement synthétisé. Le fichier produit sera au format WAVE 16 bits PCM.

http://fr.wikipedia.org/wiki/WAVEform_audio_format

Concaténation de plusieurs sons

• **Concaténation** de sons : `(rs-append` snd_1 snd_2`)`

```
(define one (rs-read "1.wav"))
(define two (rs-read "2.wav"))
(define three (rs-read "3.wav")
(define myclip
  (rs-append* (list one two three
                    (silence 44100)
                    fooc)))
```

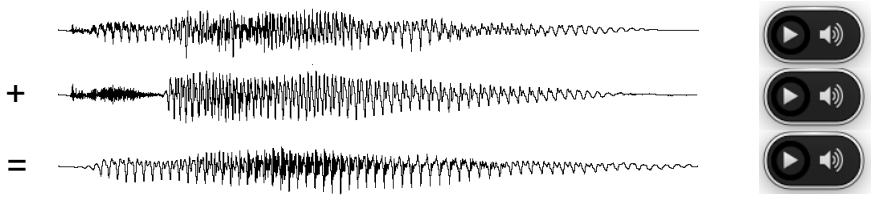
NB: `(rs-append one two)` ⇔ `(rs-append* (list one two))`

Superposition de plusieurs sons

• **Superposition** de sons : (rs-overlay snd_1 snd_2)

• ATTENTION : les amplitudes s'ajoutant, risquent de sortir de [-1,+1] !

```
> (define caco (rs-overlay* `(,one ,two ,three)))
> (play caco)
```

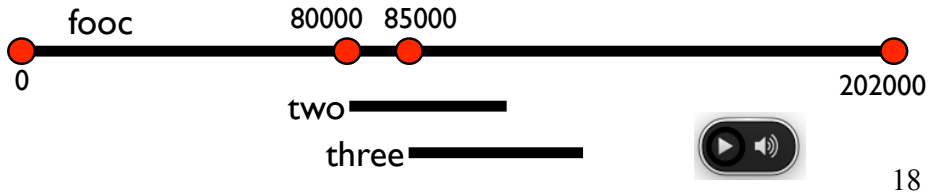


NB: (rs-overlay one two) ⇔ (rs-overlay* (list one two))

Assemblage de plusieurs sons

• Il s'agit d'une variante de rs-overlay dans laquelle les sons peuvent se chevaucher. Il suffit de préciser à quel moment (numéro de *frame*) installer un son.

```
> (define caco1
    (assemble `((,fooc 0)(,two 80000)(,three 85000))))
> (play caco1)
```



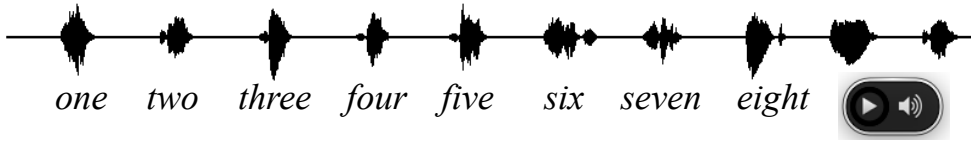
• Amplification d'un son (rs-scale k snd)

```
> (play (rs-append one (rs-scale 2 one)))
```

```
> (rs-equal? (rs-scale 2 one) (rs-overlay one one))
#t
```

• **Enregistrement** d'un son : (record-sound *nbframes*)

```
> (define voice (record-sound (* 44100 10)))
```



Le streaming avec les pstreams

• L'inconvénient de play est de demander au système de bas niveau (Portaudio) d'ouvrir un flot sonore (*stream*) pour chaque son. Pour y remédier, rsound propose d'ouvrir une **pstream** (un tube sonore) dans laquelle on peut **insérer un son à un moment donné (passé, présent ou futur)**. Avec *moment* = nombre de frames.

```
(define ps (make-pstream))      ; le temps 0 de ps
(pstream-queue ps (rs-scale 0.2 fooc) 0)
(pstream-queue ps two 100000)   ; à 100000 from now
(pstream-queue ps one 50000)    ; à 50000 from now
(printf "ps time is ~a\n" (pstream-current-frame ps))
```

• Il est possible à tout moment d'injecter un son dans la pstream pour lecture immédiate (superposée aux sons en train d'être joués) :

```
(pstream-play ps one)                ; tout de suite !
```

• Un mécanisme permet d'attacher à la pstream un son destiné à être joué à un moment donné (depuis t=0) :

```
(pstream-queue ps one
  (+ (pstream-current-frame ps) 88200)) ; dans 2 sec
```

• Ou bien une fonction d'arité 0 pour l'appeller plus tard.

```
(pstream-queue-callback ps
  (lambda () (stop))                 ; stop
  441000)                            ; à t = 10 sec
```