Visualiser un milliard (et au-delà) d'évènements en Python

Renaud Blanch <blanch@imag.fr>

Université Grenoble Alpes
Laboratoire d'Informatique de Grenoble

8 mars 2018

Enseignant-chercheur

J'enseigne à l'**UGA** (entre autre) :

- l'interaction Homme-machine ;
- la visualisation interactive d'information ;
- Python ;
- ...

Je cherche au **LIG** en :

- interaction Homme-machine ; et
- **visualisation** interactive d'information.

Utilisateur Python

- initié en 2001 (Python 2.0, pas convaincu)
- utilisation à partir de 2004 (Python 2.3)
- utilisation systématique depuis 2011
- quelques contributions

Cours

HowTo Python

Introduction à destination de programmeurs.

<http://iihm.imag.fr/blanch/howtos/Python.html>

Python par la pratique

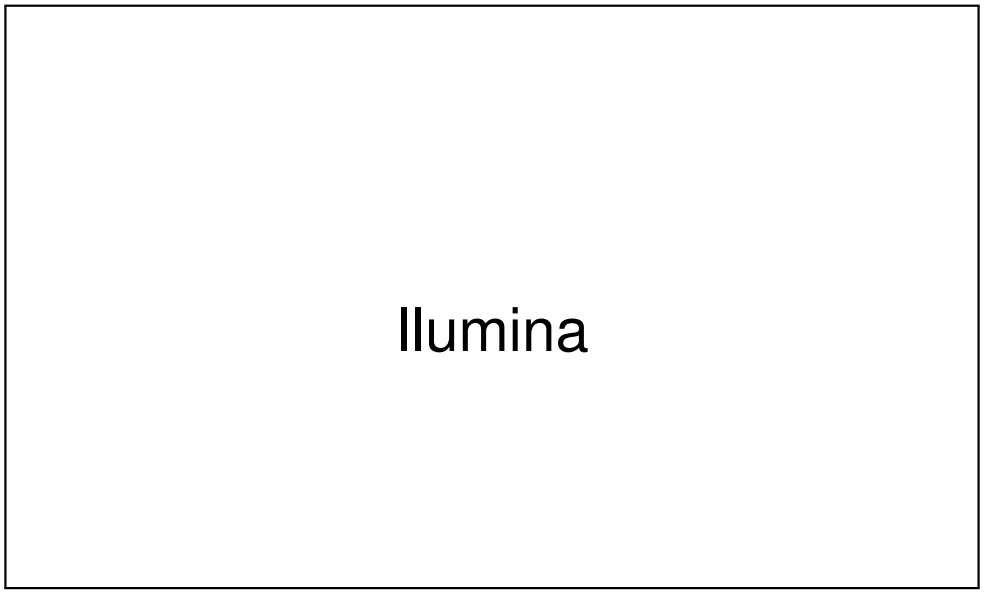
Version étendue avec cours et TPs.

<http://iihm.imag.fr/blanch/teaching/python3/>

Installation

Ilumina

Installation interactive



<http://iihm.imag.fr/blanch/projects/ilumina/>

Paquet

Seagull

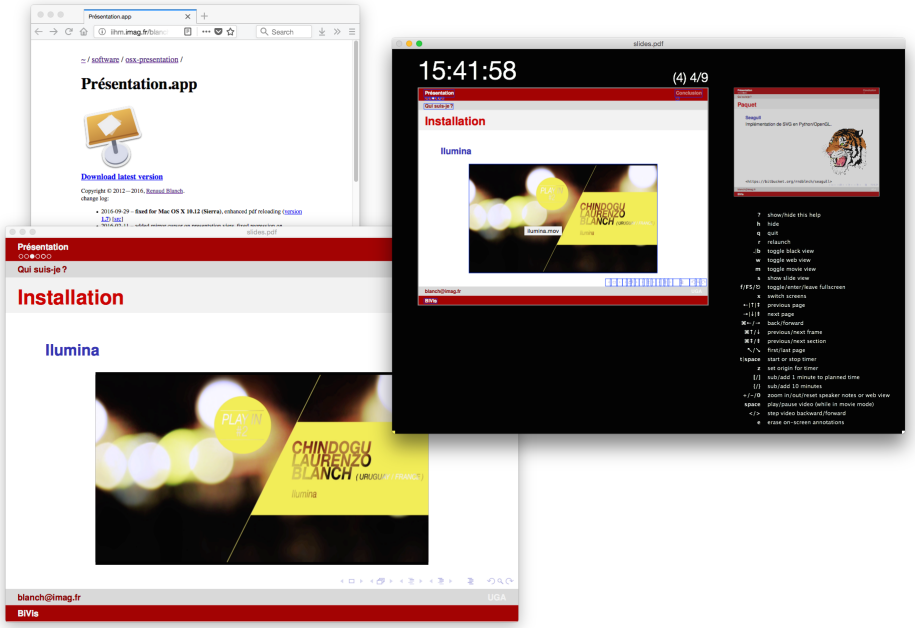
Implémentation de SVG en Python/OpenGL.

<https://bitbucket.org/rndblnch/seagull>

Application

Présentation.app

Logiciel de présentation de diapos PDF pour Mac OS X



<http://iihm.imag.fr/blanch/software/osx-presentation/>

Prototypes de recherche

Dendrogramix

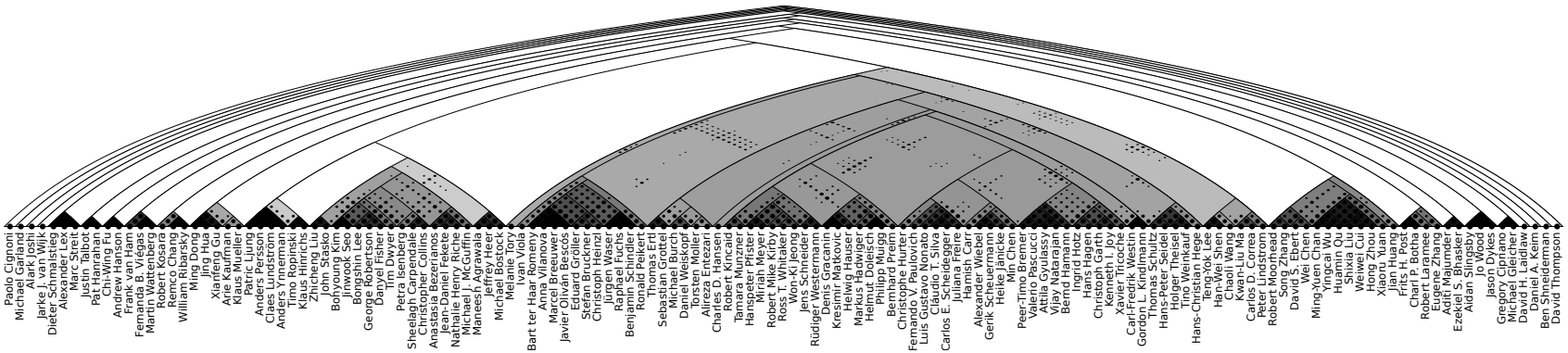
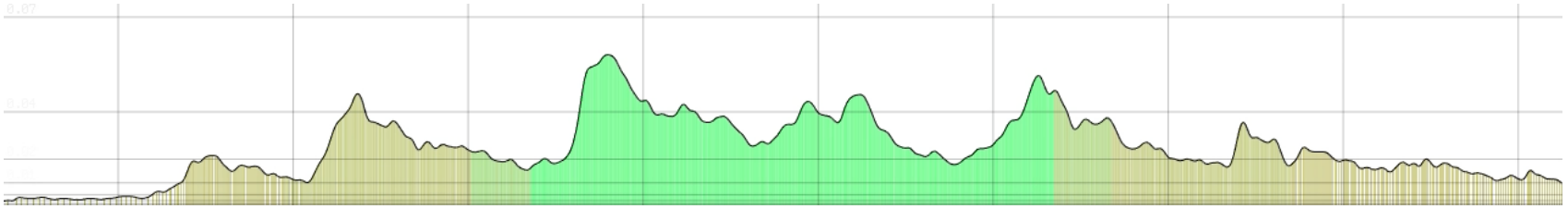


figure 1 – `<Dendrogramix> [mov]` [Blanch et al., 2015].

- R. Blanch, R. Dautriche and G. Bisson. Dendrogramix : a Hybrid Tree-Matrix Visualization Technique to Support Interactive Exploration of Dendrograms. In Proc. of PacificVis 2015, 31-38, 2015.

BiVis



démo !

Direct manipulation

Direct manipulation [Shneiderman, 1983] characterizes interaction with :

- continuous representation of the object of interest ;
- physical actions instead of complex syntax ; and
- **rapid, incremental, reversible** operations whose impact on the object of interest is immediately visible.

- B. Shneiderman. Direct Manipulation : A Step Beyond Programming Languages. In Computer, 16(8) :57–69, 1983.

The Visual Information-Seeking Mantra

The **Visual Information-Seeking Mantra** [Shneiderman, 1996] :

- **Overview first**,
- **Zoom and filter**, then
- **Details-on-demand**.

- Ben Shneiderman. The Eyes Have It : A Task by Data Type Taxonomy for Information Visualizations. In Proc. Visual Languages, 336–343, 1996.

The Information Visualization Pipeline

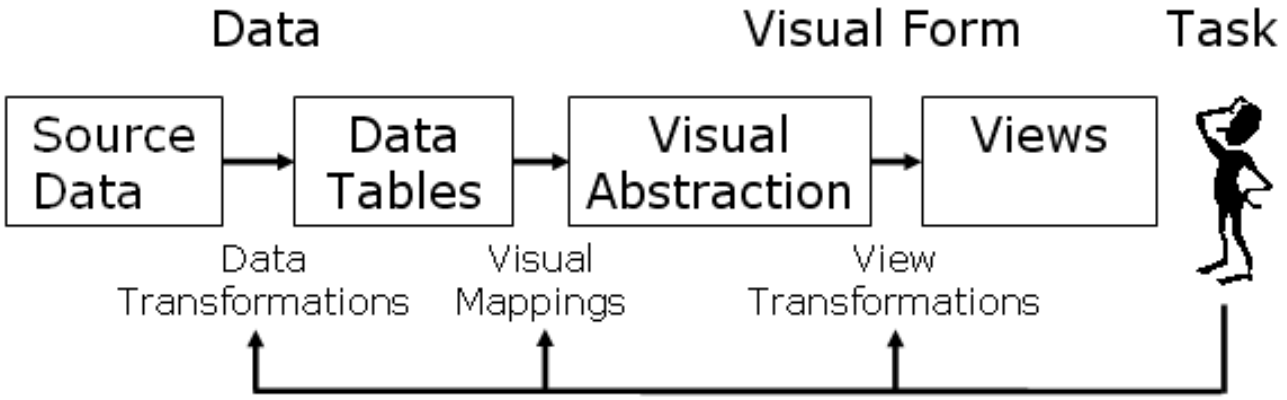


figure 2 – InfoVis pipeline [Chi & Riedl, 1998].

- E. Chi and J. Riedl. An Operator Interaction Framework for Visualization Systems. In proc. InfoVis'98, 63–70, 1998.

Visualization of Big Data

Challenge

Produce **interactive** visualizations of **big** (as in *“big data”*) time series.

How Big ?

Out of core

Billions of 64 bits timestamps, i.e., more than can fit in RAM.

Storing **4 billions** (2^{32}) events takes **32 GB**.
Pipping this file to /dev/null takes about **80 seconds** on this laptop.

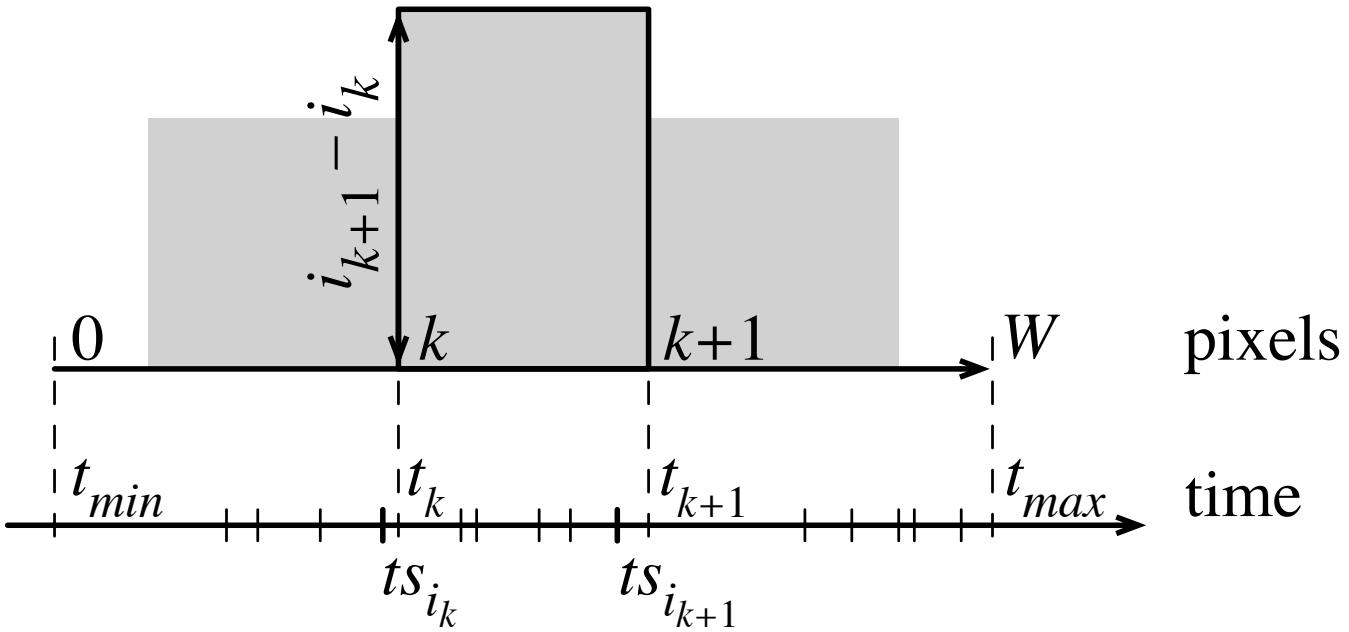
View-based Rendering

Main idea

BiVis performs a **look-up of the view pixels** into the data rather than a projection of the data onto the screen.

View-based Rendering (cont.)

Histogram



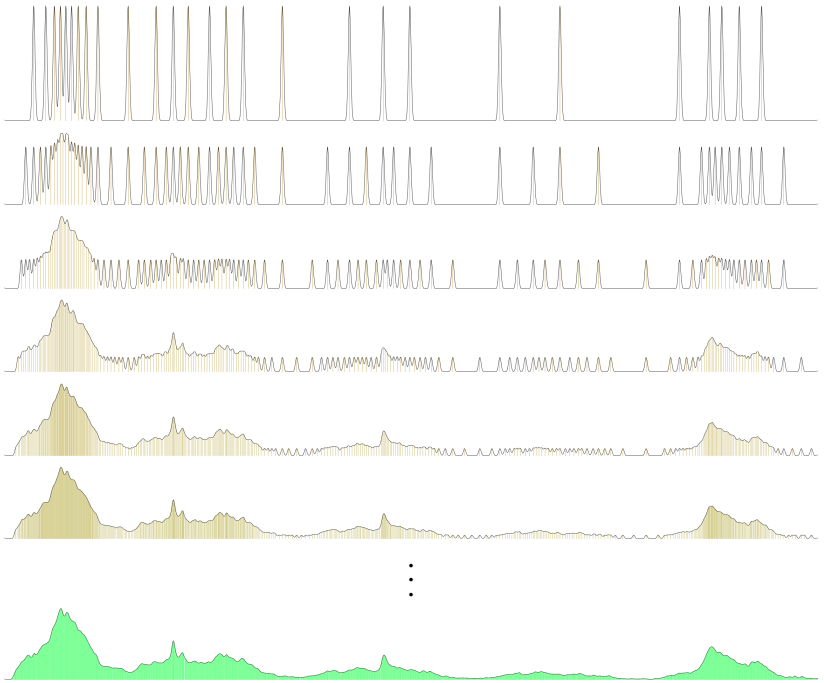
Anytime Rendering

Main idea

The binary search can be interrupted at **any time** and provide an approximated visualization.

Anytime Rendering (cont.)

Progressive refinement



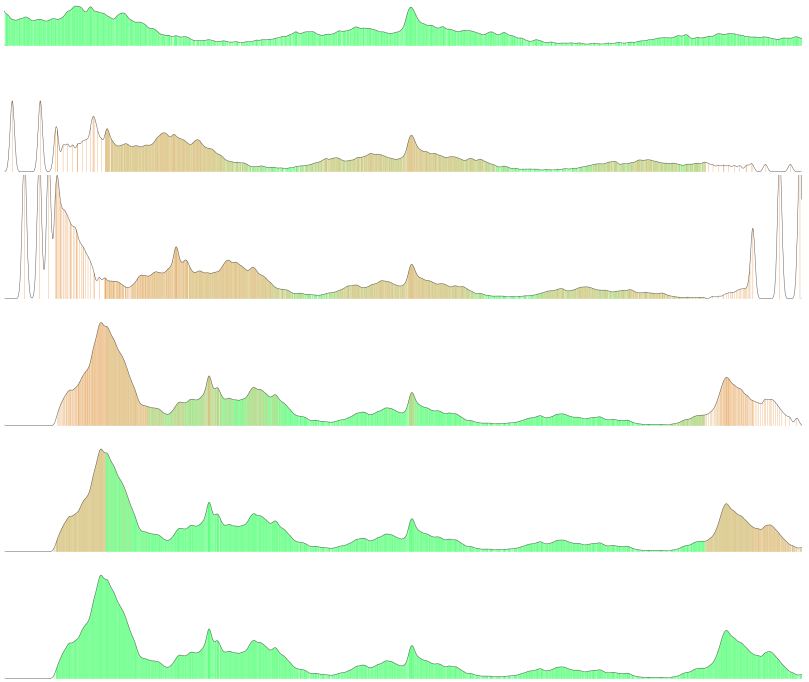
Interactive Rendering

Main idea

Reuse as much information from one frame to the other even if pixel borders have changed.

Interactive Rendering (cont.)

Information reuse



General idea

Time series often consist of couples **(timestamp, value)**.
The **partial sum** of the value provides an **aggregation** for any time interval.

Discrete values

partial sum

$s_i = \sum_{j \leq i} v_j, \quad i \in [0,\ N-1]$

aggregate at pixel border

$a_k = s_{i_k}, \quad k \in [0,\ W-1]$

histogram

$histogram_k = \frac{a_{k+1} - a_k}{t_{k+1} - t_k}, \quad k \in [0,\ W-1].$

Continuous values

partial sum

$$s_i = \sum_{j \leq i} \frac{v_j + v_{j+1}}{2} \times \left(ts_{j+1} - ts_j\right), \quad i \in [0,\ N-1]\ ; \text{or}$$
$$s_i = \sum_{j \leq i} v_j \times \left(ts_{j+1} - ts_j\right), \quad i \in [0,\ N-1]$$

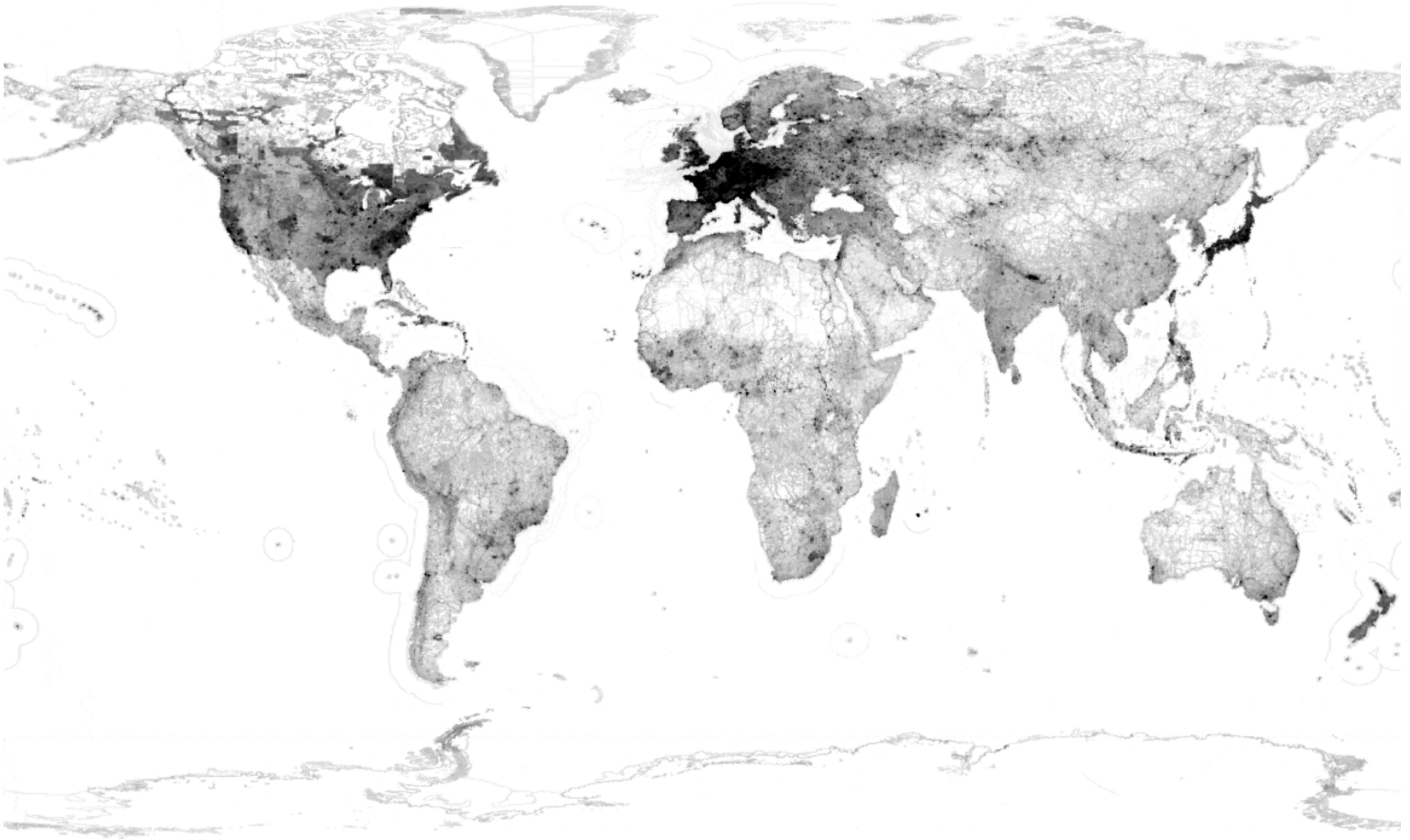
aggregate at pixel border

$$a_k = s_{i_k} + \left(s_{i_{k+1}} - s_{i_k}\right) \times \frac{t_k - ts_{i_k}}{ts_{i_{k+1}} - ts_{i_k}}, \quad k \in [0,\ W-1]$$

Geographical data

- no natural order in higher dimensions
- number of pixels grows quadratically

Geographical data (cont.)



Vectorisation

NumPy

```
def bisect(ts, t, lo, hi, runtime_ms):
    """anytime bisecting t in ts using known bounds"""
    start = time.time()
    while (time.time()-start)*1000. <= runtime_ms and \
          np.any(lo < hi):
        mid = (lo+hi)//2
        ts_mid = ts[mid]
        smaller = (t < ts_mid)
        higher  = (ts_mid <= t)
        hi[smaller] = mid[smaller]
        lo[higher] =  mid[higher]+1
    return (lo+hi)//2, hi-lo
```

Compilation

Cython

```
def bisect(object[np.float64_t] ts,
           np.ndarray[np.float64_t] t,
           np.ndarray[np.int64_t] lo, np.ndarray[np.int64_t] hi,
           runtime_ms):
    """anytime bisecting t in ts using known bounds"""

    cdef np.ndarray[np.int64_t] mid = np.empty(hi.size, dtype=np.int64
    cdef np.ndarray[np.float64_t] ts_mid = np.empty(t.size, dtype=np.f
    cdef np.ndarray smaller = np.empty(t.size, dtype=np.bool)
    cdef np.ndarray higher = np.empty(t.size, dtype=np.bool)

...
```

Compilation (cont.)

```
...
    cdef double start = time.time()
    while (time.time()-start)*1000. <= runtime_ms and \
          np.any(np.less(lo, hi)):
        np.add(lo, hi, mid)
        np.floor_divide(mid, 2, mid)
        ts_mid = np.array(ts[mid])
        np.less(t, ts_mid, smaller)
        np.less_equal(ts_mid, t, higher)
        np.copyto(hi, mid, where=smaller)
        np.copyto(lo, mid+1, where=higher)

    return np.floor_divide(np.add(lo, hi), 2), \
           np.subtract(hi, lo)
```

Parallélisation CPU

OpenMP

```
from cython.parallel import prange
cimport openmp

def bisect(object[np.float64_t] ts,
           np.ndarray[np.float64_t] t,
           np.ndarray[np.int64_t] lo, np.ndarray[np.int64_t] hi,
           double runtime_ms):
    """anytime bisecting t in ts using known bounds, using prange"""
    cdef np.int64_t mid
    cdef int i, stop = t.size
...
```

Parallélisation CPU (cont.)

```
...
    cdef double start = openmp.omp_get_wtime()
    cdef int not_finished = 1
    with nogil:
        while not_finished > 0 and \
              (openmp.omp_get_wtime()-start)*1000. <= runtime_ms:
            not_finished = 0
            for i in prange(stop):
                mid = (lo[i] + hi[i])//2
                if t[i] < ts[mid]: hi[i] = mid
                else:              lo[i] = mid+1
                if lo[i] < hi[i]:
                    not_finished += 1

    return np.floor_divide(np.add(lo, hi), 2), \
           np.subtract(hi, lo)
```

Interfaçage avec du C

```
cdef extern from "stdlib.h":
    int mergesort(void *base, size_t nel, size_t width,
                  int (*compar)(const void *, const void *)) nogil

from cython.parallel import prange, threadid
from cython.parallel cimport parallel
cimport openmp

def pargsort(np.ndarray[np.float64_t] u):
    """parallel argsort."""

    cdef size_t size = u.size
    cdef np.ndarray[np.int64_t] order = np.arange(size, dtype=np.int64
    cdef size_t width = order.itemsize
```

Interfaçage avec du C (cont.)

```
cdef int num_threads = openmp.omp_get_max_threads()
cdef size_t chunk_len = size/num_threads

cdef int thread_id
with nogil, parallel():
    thread_id = openmp.omp_get_thread_num()
    mergesort(&order[thread_id*chunk_len],
              chunk_len if thread_id < num_threads-1 else
              size-thread_id*chunk_len,
              width, compare)

mergesort(&order[0], size, width, compare)
return order
```

Plus loin

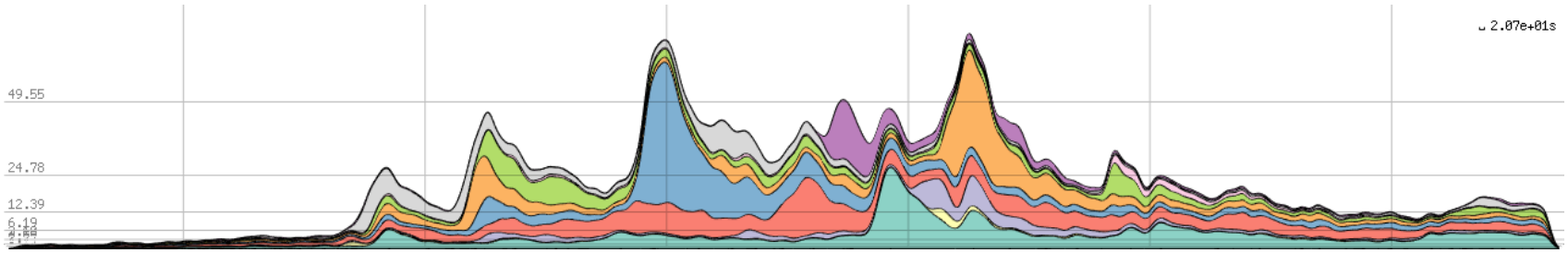
Parallélisation GPU
Utilisation d'OpenGL et OpenCL.

Entrées/Sorties
Utilisation de fichiers mappés en mémoire.

Vive Python !

Python, un langage aux **niveaux d'abstraction** allant du **pseudo-code** jusqu'au **bas niveau**, et permettant une **transition pas à pas**.

Merci !



Merci pour votre attention !

<http://iihm.imag.fr/blanch/>