

# Visualisation d'arbres de grandes tailles<sup>1</sup>

## Présentation de PSTL

Érika Baëna  
erika.baena@etu.upmc.fr

Diana Malabard  
diana.malabard@etu.upmc.fr

Antoine Genitrini (encadrant)  
antoine.genitrini@lip6.fr

Université Pierre et Marie Curie

13 mai 2014

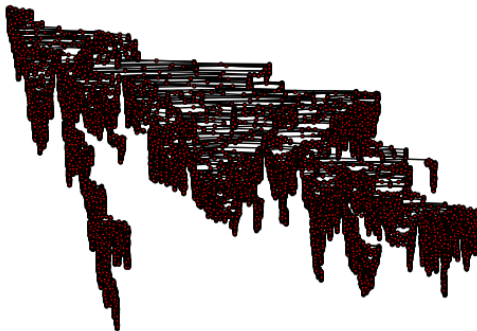
---

1. Disponible sur: [https://github.com/BErika/PSTL\\_TreeDisplay](https://github.com/BErika/PSTL_TreeDisplay)

## Plan de la présentation

- 1 Introduction
- 2 État des lieux
- 3 Implémentation
- 4 Étude de performances
- 5 Comparaison des rendus
- 6 Conclusion

## Pourquoi ?



- Pourquoi des arbres ? Structure primordiale en informatique.
- Pourquoi afficher des arbres de grande taille ? Pour observer des tendances.

# Objectif

## Objectif du projet

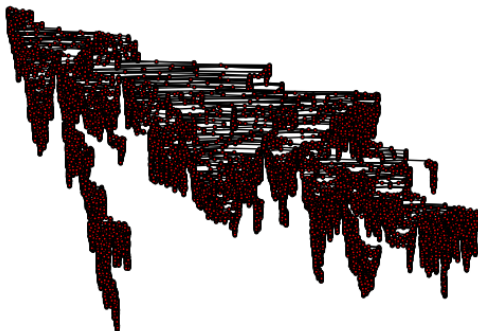
Affichage élégant et efficace de tout type d'arbre

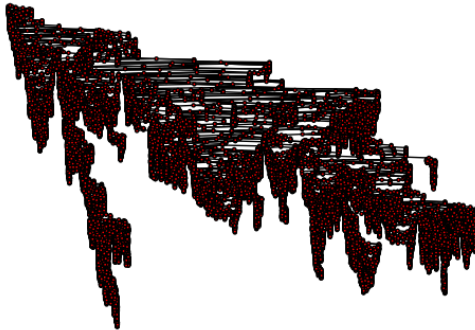
## Problèmes

- Qu'est-ce qu'un affichage élégant ?
- Comment optimiser le calcul de la mise en page ?

## Plan de la présentation

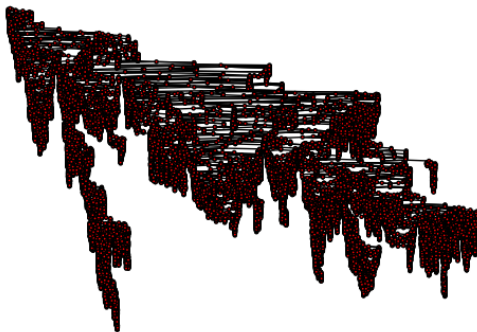
- 1 Introduction
- 2 État des lieux
  - Principes à respecter pour un affichage élégant
  - Algorithmes existants
- 3 Implémentation
- 4 Étude de performances
- 5 Comparaison des rendus
- 6 Conclusion



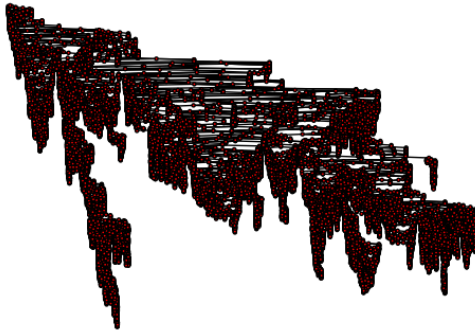


## Principe 1

Les arêtes de l'arbre ne doivent pas s'intersecter.

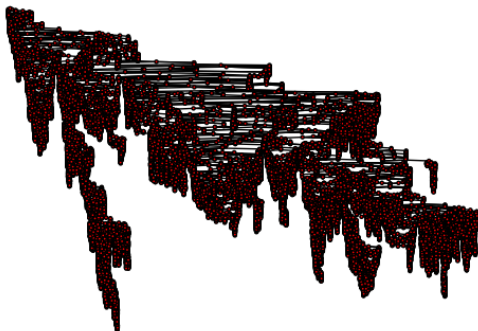


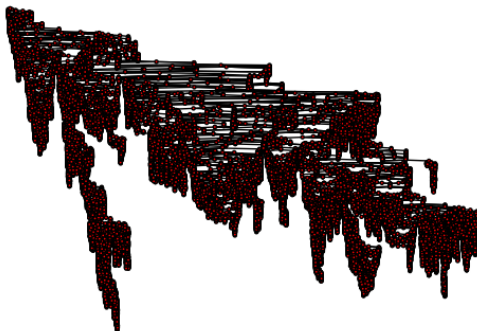




## Principe 2

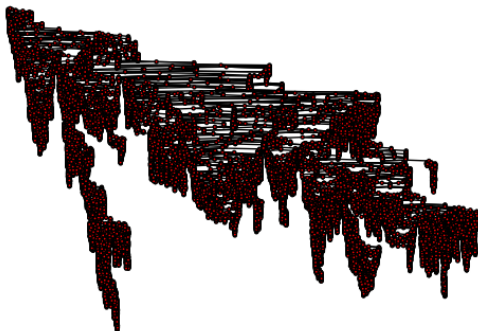
Les nœuds de même profondeur doivent être dessinés sur la même ligne horizontale.

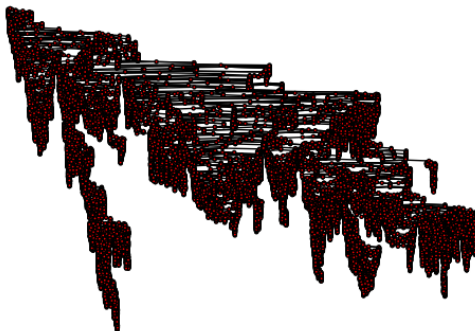




### Principe 3

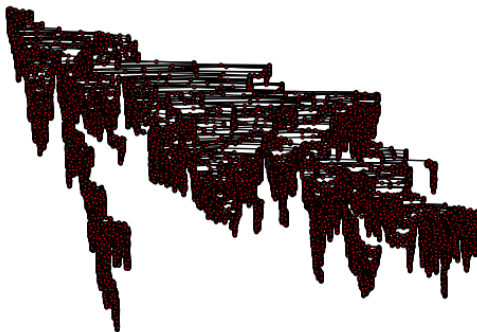
Les arbres doivent être dessinés de la manière la plus compacte possible.

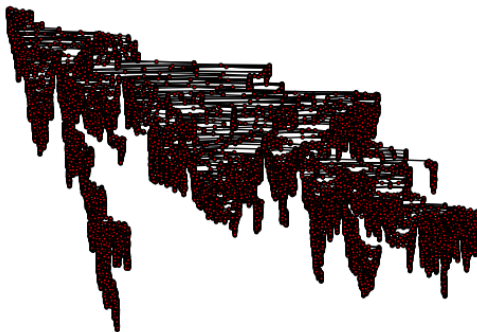




#### Principe 4

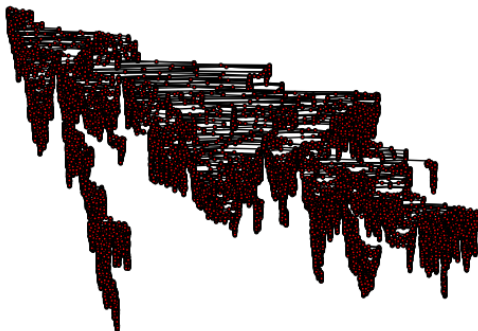
Un nœud parent doit être centré par rapport à ses fils.



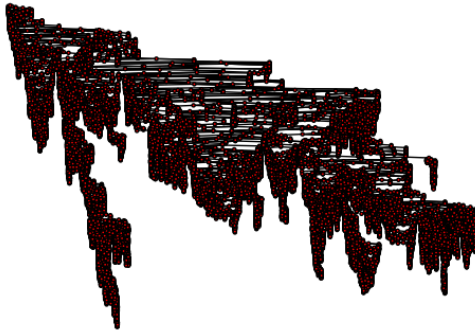


### Principe 5

Un sous-arbre doit être dessiné de la même façon, peu importe où il est placé dans l'arbre.



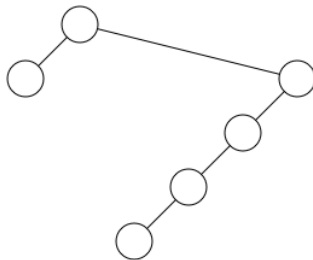




## Principe 6

Les nœuds fils d'un nœud père doivent être espacés de manière homogène.

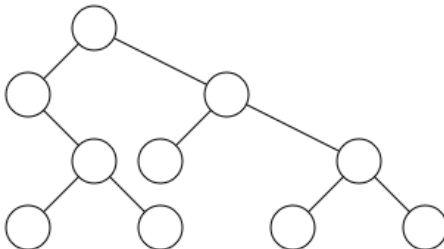
# Knuth



Décrit une idée de "slot disponible"

Inconvénient :

- Ne respecte que les principes 1 et 2



- Ne respecte pas les principes 4 et 5

## The Mods and the Rockers

Traitement en deux passes

Avantage :

- Respecte tous les principes

Inconvénient

- Ne concerne que les arbres binaires

# Plan de la présentation

- 1 Introduction
- 2 État des lieux
- 3 Implémentation
  - Fonctionnement général
  - Parsing
  - Calcul des coordonnées
  - Génération de la sortie
- 4 Étude de performances
- 5 Comparaison des rendus
- 6 Conclusion

## Paramètres

Entrées prises en compte :

- Mots bien parenthésés
- XML
- DOT

Sorties prises en compte :

- TikZ
- Asymptote
- NetworkX + Matplotlib

# Mots bien parenthésés

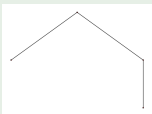
## Grammaire respectée

ARBRE : '(' LABEL NOEUDS ')'

NOEUDS : ARBRE NOEUDS | e

LABEL : [a-zA-Z1-9]\* | e

## Exemple



L'arbre est représenté par  $((()()))$ .

## DOT I

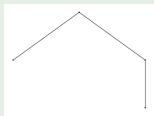
## Grammaire respectée

```
DOT : STRICT GRAPH ID '{' SEQINST '}'  
STRICT : strict | e  
GRAPH = digraph | graph  
SEQINST : INST ';' SEQINST | e  
INST : ID '[' label = "LABEL" ']' | ID LINK ID  
LINK : -- | ->  
ID : [0-9]*  
LABEL : [a-zA-Z1-9]* | e
```



## DOT II

## Exemple



L'arbre est représenté par :

```
digraph {  
  1 [label=""];  
  2 [label=""];  
  3 [label=""];  
  4 [label=""];  
  1 -> 2;  
  2 -> 3;  
  3 -> 4;  
}
```

## XML I

## Grammaire respectée

```
XML : <?xml version="1.0"?><tree> NOEUDS </tree>
```

```
NOEUDS : NOEUD NOEUDS | ε
```

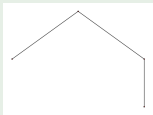
```
NOEUD : <node type=TAG id=ID> NOEUDS </noeud>  
      | <leaf type=TAG id=ID />
```

```
TAG : " [a-zA-Z1-9]* "
```

```
ID : [0-9]*
```

## XML II

## Exemple



L'arbre est représenté par :

```
<?xml version="1.0"?>
<tree>
  <node type="" id=1>
    <leaf type="" id=2 />
    <node type="" id=3>
      <leaf type="" id=4 />
    </node>
  </node>
</tree>
```

# Structure de données

```
1 class Tree(object):  
    "Local representation of tree"  
3  
    def __init__(self, x = -1, depth=0, label="", children=None,  
        offset = 0, isRoot = False):  
5        self.x = x  
        self.y = depth  
7        self.label = label  
        self.offset = offset  
9        self.height = None  
        self.width = None  
11       if children is None:  
            self.children = list()  
13       else:  
            self.children = children
```

## Algorithme 1

## Passe 1

```
1  def setup (self, depth=0, nexts=None, offset=None):
2      if nexts is None:
3          nexts = defaultdict(lambda:0)
4          if offset is None:
5              offset = defaultdict(lambda:0)
6
7      # L'ordonnée est triviale, c'est la profondeur.
8      self.y = depth
9
10     # On calcule d'abord les coordonnées des enfants.
11     for c in self.children:
12         c.setup(depth+1, nexts, offset)
13
14     # On centre le noeud au milieu de ses enfants.
15     nbChildren = len(self.children)
16     if (nbChildren == 0):
17         place = nexts[depth]
18         self.x = place
19     else:
20         place = (self.children[0].x +
21                 self.children[nbChildren-1].x) / 2
22
23     # On calcule l'éventuel décalage engendré.
24     offset[depth] = max(offset[depth], nexts[depth]-place)
```

## Algorithme II

Passe 1

```
25 # On applique le décalage de la profondeur.  
   if (nbChildren != 0):  
27     self.x = place + offset[depth]  
  
29 # On met à jour la prochaine place disponible à cette  
   profondeur.  
   nexts[depth] = self.x + 1
```

## Algorithme 1

Passe 2

```
def addOffsets (self , offsum=0):  
2   self.x = self.x + offsum  
   offsum = offsum + self.offset  
4  
   self.height = self.y  
6   self.width = self.x  
8  
   for c in self.children:  
       c.addOffsets(offsum)  
10  self.height = max (self.height , c.height)  
    self.width = max (self.width , c.width)
```

## Fonctionnement

Points communs :

- Parcours en profondeur
- Génération de la sortie au fur et à mesure

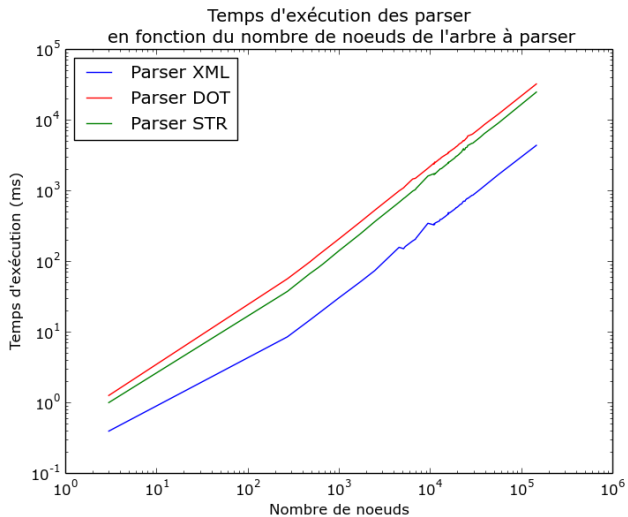
Différence :

- TikZ et Asymptote  $\Rightarrow$  Fichier .tex
- NetworkX  $\Rightarrow$  Image ou PDF

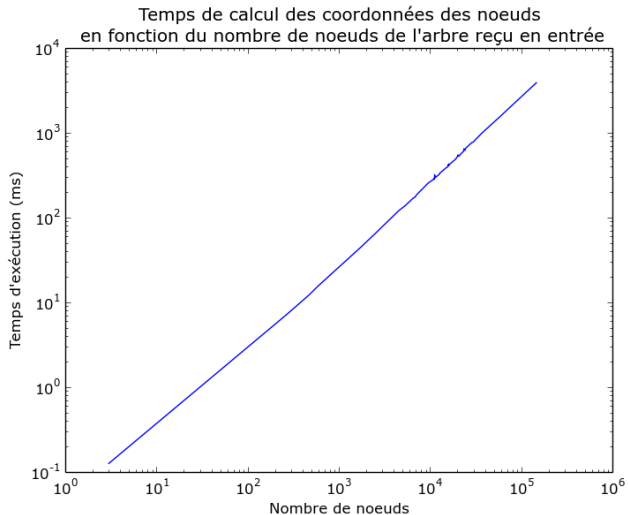


## Plan de la présentation

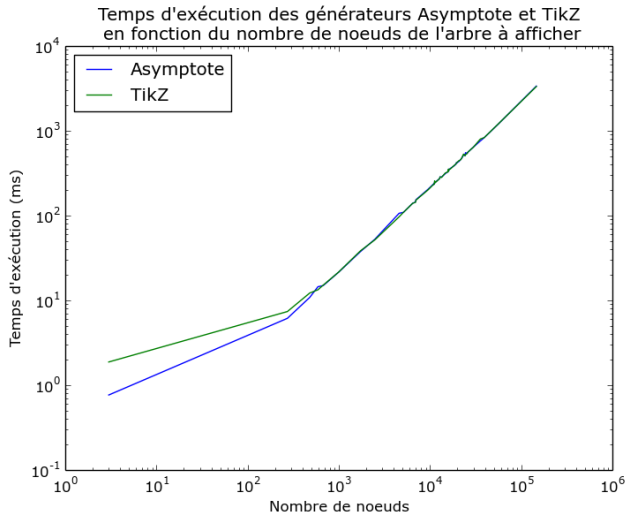
- 1 Introduction
- 2 État des lieux
- 3 Implémentation
- 4 Étude de performances**
- 5 Comparaison des rendus
- 6 Conclusion



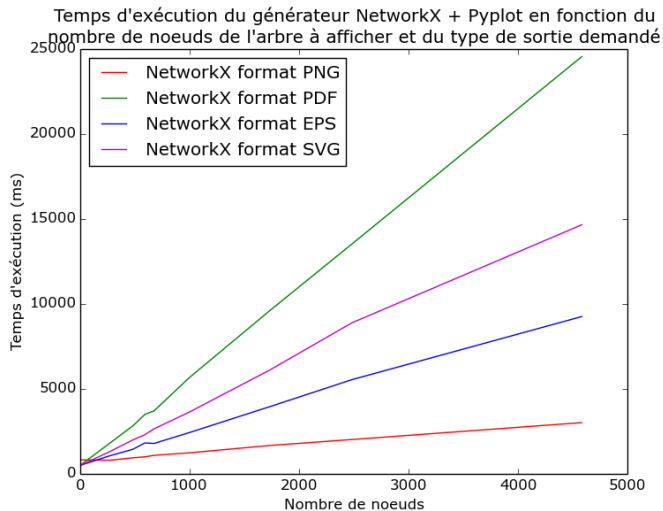
## Calcul des coordonnées



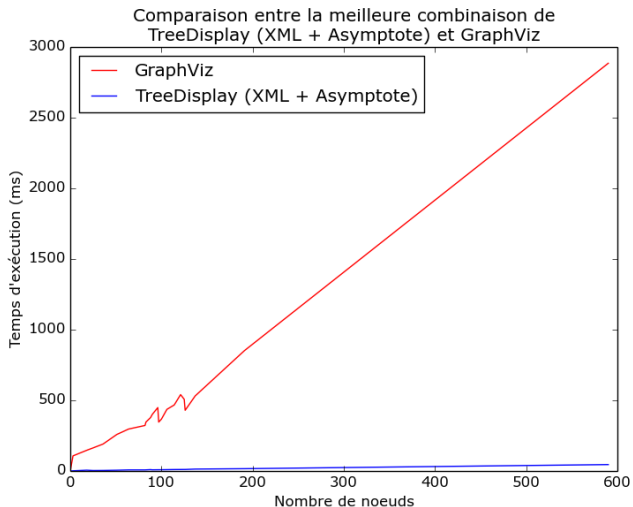
## Générateurs I



## Générateurs II

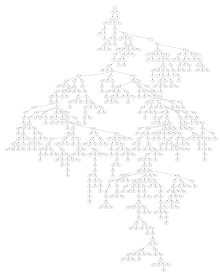


## GraphViz vs TreeDisplay

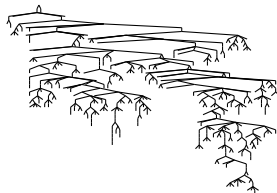


## Plan de la présentation

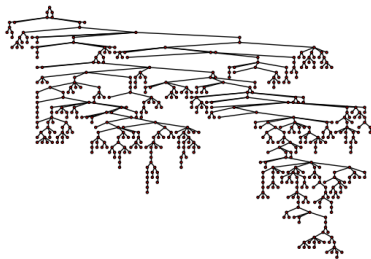
- 1 Introduction
- 2 État des lieux
- 3 Implémentation
- 4 Étude de performances
- 5 **Comparaison des rendus**
  - Arbre de grande taille sans labels
  - Arbre de petite taille avec labels
- 6 Conclusion



GraphViz

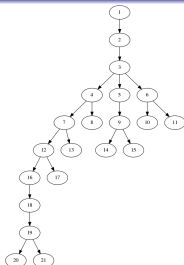


TikZ

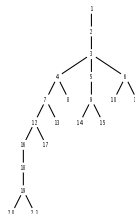


NetworkX

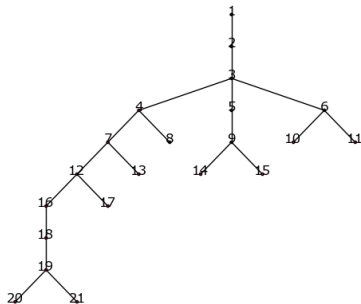




## GraphViz



TikZ



NetworkX

## Plan de la présentation

- 1 Introduction
- 2 État des lieux
- 3 Implémentation
- 4 Étude de performances
- 5 Comparaison des rendus
- 6 Conclusion

## Bilan

- Étude d'articles scientifiques
- Complexité linéaire
- Modules réutilisables

## Pour la suite

- Extension aux graphes
- Optimisation mémoire
- Prise en charge du format ARB
- Ajout de critères de représentation