

La boîte à outils InfoVis

Jean-Daniel Fekete

INRIA Futurs/LRI
Bât 490, Université Paris-Sud
91405 ORSAY, France
Jean-Daniel.Fekete@inria.fr

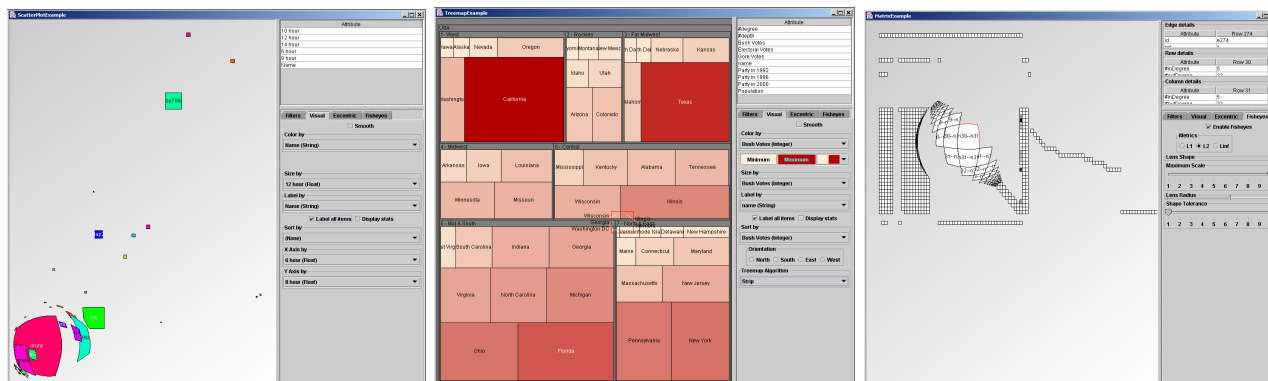


Figure 1 :Exemples de Scatter Plot, Treemap et Visualisation de graphe par matrice réalisés avec la boîte à outils InfoVis. Un Fisheye est appliqué à droite et à gauche tandis que des labels excentriques sont appliqués au milieu.

RESUME

Cet article décrit *InfoVis*, une boîte à outils conçue pour faciliter la conception et l'implémentation de nouvelles techniques de visualisation 2D et leur utilisation à partir d'applications Java. *InfoVis* fournit des structures de données optimisées pour les requêtes dynamiques et un ensemble de techniques de visualisations utilisables directement ou pouvant s'enrichir par dérivation de classe. Les structures de données définies actuellement sont les tables, arbres et graphes. Les visualisations fournies incluent les scatter plots, séries temporelles, coordonnées parallèles, diagrammes nœud lien pour les arbres et les graphes, treemaps et arbres à glaçons pour les arbres, matrices pour les graphes. Toutes ces visualisations, ainsi que d'autres réalisées à l'aide d'*InfoVis*, peuvent utiliser des Fisheyes ou des labels excentriques.

InfoVis peut utiliser *Agile2D*, une implémentation des objets graphiques de *Java2D* qui s'appuie sur les accélérations matérielles des cartes graphiques modernes offrant des facteurs d'accélération de 10 à 1000.

MOTS CLES : Visualisation d'information, boîte à outils, Interaction avancée, Graphique.

ABSTRACT

This article presents the *InfoVis* Toolkit, designed to

Copyright © 2004 by the Association for Computing Machinery, Inc. permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM, Inc., IHM 2004 Aug 30 – Sep 3, 2004, Namur, Belgium
Copyright 2004 ACM 1-58113-926-8 \$5.00

support the creation, extension and integration of advanced 2D Information Visualization components into interactive Java Swing applications. The *InfoVis* Toolkit provides specific data structures to achieve a fast action/feedback loop required by dynamic queries. It comes with a large set of components such as range sliders and tailored control panels required to control and configure the visualizations. These components are integrated into a coherent framework that simplifies the management of rich data structures and the design and extension of visualizations. Supported data structures currently include tables, trees and graphs. Supported visualizations include scatter plots, time series, Treemaps, node-link diagrams for trees and graphs and adjacency matrix for graphs. All visualizations can use fisheye lenses and dynamic labeling. The *InfoVis* Toolkit supports hardware acceleration when available through *Agile2D*, an implementation of the *Java Graphics* API based on *OpenGL*, achieving speedups of 10 to 1000 times.

CATEGORIES AND SUBJECT DESCRIPTORS: I.3.6 [Methodology and Techniques] Graphics data structures and data types; H.5 [User Interfaces] Graphical User Interface, Benchmarking; C.4 [Performance of Systems]: Design Studies;

GENERAL TERMS: Design, Performance

KEYWORDS: Information Visualization, Toolkit, Advanced Interaction, Graphics.

INTRODUCTION

La visualisation d'information est un domaine de recherche fécond et une industrie naissante. Chaque année depuis quinze ans, de nouvelles techniques de visualisation sont inventées, de nouvelles techniques d'interaction permettent d'interagir plus efficacement avec les visualisations, de nouveaux domaines d'applications exploitent avec succès la visualisation d'information et de nouvelles théories viennent enrichir notre compréhension du domaine.

Malgré toutes ces avancées, mettre en œuvre des techniques de visualisation d'information reste difficile. Chaque année, des projets d'étudiants reproduisent des techniques de visualisation existantes sans avoir le temps de les améliorer. Prises séparément, chaque technique utilisée en visualisation peut paraître simple : charger une structure de donnée en mémoire, afficher cette structure de donnée, la filtrer à l'aide de requête dynamique, interagir avec les données affichées, naviguer dans les données. Dans la réalité, chaque application de visualisation recrée un monde nouveau avec ses propres structures de données, ses propres formats d'entrées sorties, ses propres techniques de visualisation, etc. Les boîtes à outils graphiques actuelles ne permettent pas de réaliser rapidement une avancée dans une direction seule, il faut recréer un monde à chaque fois. Pour prendre un exemple simple, aucune boîte à outil graphique n'offre de « Range Slider », le composant le plus utilisé pour les requêtes dynamiques en visualisation d'information. Chaque projet commence donc par ré-implementer (plus ou moins bien) ce composant.

Dans cet article, nous décrivons la boîte à outils InfoVis qui propose un cadre cohérent pour utiliser des techniques de visualisation d'information connues ou créer des nouvelles techniques. InfoVis offre actuellement une dizaine de techniques de visualisation :

- pour les tables : scatter plots, séries temporelles, coordonnées parallèles ;
- pour les arbres : diagrammes nœuds liens cartésiens et polaires, treemaps (slice&dice, squarified, strip) et arbres à glaçons ;
- pour les graphes : diagrammes nœuds liens calculés par les programmes dot, neato et twopi de GraphViz [1], matrices d'adjacences.

Ces techniques reposent toutes sur des structures de données unifiées orientées vers la visualisation, ainsi que sur des mécanismes de filtrage et de requêtes dynamiques génériques. Enfin, des lentilles magiques peuvent être appliquées à toutes les visualisations, les deux fournies étant les fisheyes [4] et les labels excentriques [9].

Selon Alan Kay, les propriétés d'un bon outil sont : « Les choses simples doivent être simples [à réaliser]. Les choses complexes doivent être possibles ». InfoVis diminue le temps de mise en œuvre de techniques de visualisation en échange d'un temps d'apprentissage.

C'est le rapport entre les deux qui permet de juger en premier lieu de l'utilité de la boîte à outils. Un argument moins évident mais tout aussi essentiel pour le domaine de la visualisation d'information est la capacité à répliquer les recherches publiées et à les modifier en les améliorant. En 1991, Brian Gaines [11] a proposé un modèle du développement d'une science dans le temps qui distingue six phases :

1. La découverte : naissance d'une idée créative significative dans un domaine ;
2. La réplication : l'idée est répliquée et modifiée de manière créative pour expérimenter et améliorer l'idée originelle ;
3. Empirisme : des leçons empiriques sont tirées des expérimentations et formulées sous forme de règles empiriques utiles ;
4. Théorie : des théories sont formulées à partir des constatations expérimentales ;
5. Automatisation : les théories sont acceptées et utilisées de manières mécaniques pour prévoir les résultats d'expériences nouvelles ;
6. Maturité : les théories sont assimilées et utilisées journallement sans poser de questions.

Une boîte à outil comme InfoVis a aussi pour but de faciliter la phase de réplication des résultats de recherche en visualisation d'information pour faciliter la diffusion du domaine de recherche.

Nous présentons un état de l'art après la description d'InfoVis et les exemples pour faciliter la comparaison.

ARCHITECTURE DE LA BOITE A OUTILS INFOVIS

La boîte à outils InfoVis est organisée autour de cinq parties principales (Figure 2) :

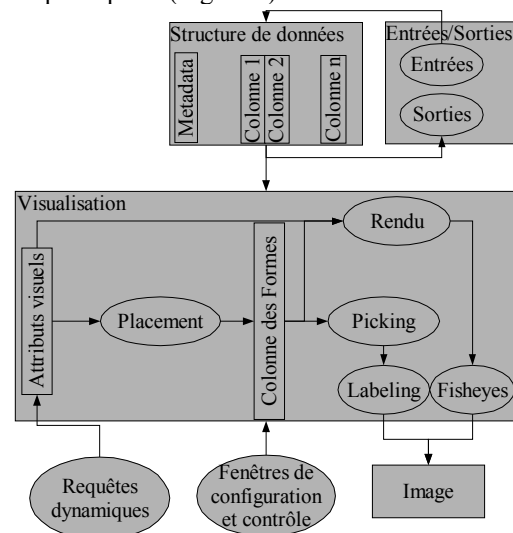


Figure 2: Structure interne de la boîte à outils InfoVis

1. les structures de données (table, arbre, graphe) ;
2. les visualisations ;
3. les requêtes dynamiques ;
4. les fenêtres de configuration et d'interaction ;
5. les entrées/sorties.

Dans cette section, nous décrivons les trois premières parties qui sont originales. Dans la section suivante,

nous décrivons quelques exemples intéressants d'utilisation d'InfoVis.

Structures de données

Les données à visualiser doivent être placées en mémoire vive pour permettre les requêtes dynamiques. InfoVis offre des structures de données très efficaces à la fois en place et en performance et originale dans leur conception.

La plupart des applications de visualisation d'information représentent les données sous forme de tuples attachés à une topologie. Par exemple, une table de données est implémentée par un tableau à une dimension dont chaque ligne contient les valeurs d'attributs (les tuples), stockées dans un autre tableau à une dimension. Des données arborescentes sont généralement représentées par une structure d'arbre dont chaque nœud contient un tuple d'attributs. C'est l'organisation des systèmes comme Polaris, Treemap, SpaceTree et bien d'autres encore.

InfoVis représente ses données sous forme de tables dont chaque attribut est stocké dans une colonne (et non une ligne). La clé est un indice et un enregistrement est l'ensemble des valeurs des colonnes à un indice donné. Cette organisation diffère donc à la fois des bases de données classiques et des préconisations orientées objet où chaque objet contient ses attributs.

Cette organisation est bien plus adaptée à la visualisation pour trois raisons : elle est plus économique en mémoire, elle est plus performante pour la visualisation et elle permet de créer de nouveaux attributs très facilement, opération qui est fondamentale en visualisation d'information.

Place mémoire. Les langages de programmation gèrent mal les données hétérogènes définies dynamiquement. La représentation d'un tuple de données hétérogènes en mémoire requiert des calculs de pointeurs complexes et risqués en C et doit passer par des doubles indirections et conversions de types en Java. Par exemple, supposons que l'on veuille stocker dynamiquement un entier I, une chaîne de caractère S et un nombre flottant en double précision F. En langage C, on devra allouer un bloc de mémoire de la bonne taille (`sizeof(int)+sizeof(char*)+sizeof(double)`) puis calculer les indexes dans le bloc avant d'accéder aux valeurs. Les architectures matérielles imposent des contraintes supplémentaires d'alignement qui rendent cette opération plus complexe et non portable. C'est ce que font les compilateurs C lorsqu'on définit une structure, mais cette définition ne peut être que statique. En Java, cette arithmétique de pointeurs est tout simplement interdite. Il faut créer un tableau de pointeurs et créer un objet pour chaque valeur : Integer pour I, Double pour F, S étant déjà un objet. Chaque objet prend une taille mémoire importante : au moins 16 octets supplémentaires par objet. Une analyse précise est disponible dans [6].

Performance. Accéder à une valeur d'un tuple est lent en C comme en Java car l'architecture actuelle des or-

dateurs favorise énormément l'accès à des données consécutives en mémoire en gardant des portions de mémoire en cache (un facteur de 1 à 4 [6]). En visualisation d'information, les attributs des objets sont utilisés comme attributs graphiques : couleur, taille, transparence, position, etc. Dans la majorité des jeux de données, les attributs des objets sont bien plus nombreux que les attributs graphiques. L'affichage d'une table de données implique donc le parcours de chaque objet avec extraction des valeurs d'attributs servant à l'affichage. Ces valeurs ne sont pas contiguës en mémoire puisque elles sont dispersées dans la table des tuples, ce qui provoque de fréquents défauts de cache et ralentit considérablement la vitesse de parcours. L'organisation des attributs en colonnes facilite la cohérence des caches.

Création de nouveaux attributs. La visualisation ne se contente pas d'utiliser des attributs, elle en crée aussi pour visualiser des données synthétiques (tel attribut multiplié par tel autre) ou gérer des propriétés comme la sélection ou le filtrage. La création de nouveaux attributs dans un tuple est très coûteuse contrairement à la création d'une colonne.

Bien que l'organisation interne d'InfoVis repose sur des tables remplies de colonnes, des interfaces rendent

```
Tree tree = new DefaultTree();
IntColumn date = new IntColumn («date»);
date.setFormat(new UTCDateFormat());
StringColumn name = new StringColumn («name»);
Tree.addColumn(date);
Tree.addColumn(name);
int n1 = tree.addNode(Tree.ROOT);
name.setValue(n1, «Root»);
date.setValue(n1, «13/Mar/2004 11:23:30»);
int n2 = tree.addNode(n1);
...
```

Figure 3: Création d'une structure d'arbre avec attributs dans InfoVis.

son utilisation aisée. La Figure 3 donne un exemple de construction d'un arbre de données attribuées. La topologie des données dans un arbre est représenté par des colonnes spéciales (parent, premier fils, etc.) qui peuvent être étendues si nécessaire (nombre de fils, profondeur etc.).

Un graphe est représenté sous la forme de deux tables : les sommets et les arêtes. Chacune des tables peut contenir un nombre arbitraire d'attributs. La table des sommets contient des colonnes spéciales qui lient les sommets à leurs arêtes. La table des arêtes contient des colonnes spéciales qui lient les arêtes à leurs sommets.

Les tables comme les colonnes peuvent aussi contenir des méta données qui permettent plusieurs niveaux d'annotations sur les données. Par exemple, une table chargée à partir d'un fichier peut contenir des informations relatives à ce fichier en méta données. Une colonne peut contenir des informations sur sa nature qui seront exploitées par les visualisations (données nominales, catégorielles, ordonnées ou différentielles par exemple). Enfin, les colonnes ne sont pas nécessaire-

ment denses : elles peuvent avoir des attributs manquants afin de modéliser des lacunes dans les données. Pour les données très lacunaires, on peut créer des colonnes « creuses », basées sur des tables de hachage et non sur des vecteurs. Enfin, les colonnes et les tables offrent un service de notification qui facilite la communication entre les structures de données et les visualisations, ainsi que les maintenances de dépendances pour les colonnes calculées.

Avec ces structures de données, InfoVis peut représenter des données aussi riches que nécessaire sans sacrifier aux performances ni à la compacité. InfoVis unifie aussi les attributs topologiques et les attributs utilisateurs, tous deux représentés sous forme de colonnes.

Les modules d'entrée et de sorties permettent de charger et de sauvegarder les structures de données à partir de formats bien connus (Excel, XML, DOT, etc.)

Visualisations

Dans InfoVis, un composant de visualisation gère une table et transforme chaque enregistrement en une marque visible sur l'écran – parfois nommée *item* – conformément à une technique de visualisation. En plus de l'affichage, la visualisation gère le filtrage, la sélection et l'interaction.

Concrètement, chaque technique de visualisation hérite de l'interface *Visualisation* et peut « connecter » des colonnes de sa table à des attributs graphiques. Toutes les transformations gèrent les attributs visuels suivants :

Couleur : colonne gérant la couleur des items ;

Taille : colonne gérant la taille de l'item ;

Alpha : colonne gérant la transparence des items ;

Label : colonne gérant le nom affiché des items.

La Figure 4 montre un exemple de création de visualisation pour l'arbre de la Figure 3.

```
Visualization visu =
    new TreemapVisualization(tree);
visu.setVisualColumn(«color», date);
visu.setVisualColumn(«size», date);
```

Figure 4: Création d'une visualisation d'arbre dans InfoVis.

Le mécanisme d'attributs visuel est aussi utilisé pour des attributs plus abstraits :

Sélection : une colonne de sélection peut être associée à chaque visualisation. Lorsque deux visualisations partagent la même colonne de sélection, leur sélection est effectivement partagée, permettant ainsi la technique de « brushing » ou d'exploration multidimensionnelle ou multi-représentation.

Filtrage : une colonne permet le filtrage des items dans chaque visualisation. Elle peut aussi être partagée pour synchroniser le filtrage inter visualisations.

Ordre : un ordre d'affichage peut être spécifié, contrôlé par une relation d'ordre, une permutation ou une colonne. Dans ce dernier cas, c'est l'ordre des éléments dans cette colonne qui sera respecté.

Chaque classe de visualisation peut ajouter des attributs visuels si elle en a besoin. Par exemple, les scatter plots utilisent une colonne pour l'axe des X et une autre pour l'axe des Y. Des valeurs par défauts sont utilisées et spécifiées lorsque les attributs visuels ne sont pas liés à des colonnes. Par exemple, la transparence par défaut est « opaque » pour la plupart des visualisations sauf les séries temporelles où elle vaut $\frac{1}{2}$ car les superpositions rendraient cette visualisation impossible à lire.

Les visualisations d'InfoVis s'organisent par structure de donnée (table, arbre, graphe). De plus, les visualisations peuvent se superposer et contrôler éventuellement d'autres visualisations. C'est le cas pour les labels excentriques qui forment une couche de visualisation au dessus d'une autre visualisation qu'elle observe. De manière similaire, les représentations nœud lien sont organisées en deux parties : la visualisation des nœuds et celle des liens. Cette organisation permet une grande souplesse et autorise des innovations intéressantes comme décrites dans la section suivante.

Grâce à ce découplage, il est possible de contrôler les attributs graphiques des nœuds indépendamment de ceux des liens. Il est aussi possible de construire des styles de liens qui pourront s'appliquer à n'importe quel type de diagramme nœud lien (liens droits, liens orthogonaux, liens courbes cubiques ou plus généraux, etc.).

Requêtes dynamiques

Les requêtes dynamiques reposent sur deux composants : les colonnes de filtrage liées aux visualisations et les interacteurs permettant à l'utilisateur d'appliquer le filtrage par manipulation directe. Les requêtes dynamiques doivent être très rapides pour permettre à la boucle action/affichage d'être fluide.

Plusieurs mécanismes compliqués ont été proposés dans la littérature de visualisation [19] pour gérer le filtrage de manière incrémentale. Ces algorithmes produisent des artefacts déplaisants lorsque des items se superposent : le filtrage d'un item efface alors toute la région sur laquelle il est affiché, y compris les items qu'il recouvre. Nous utilisons un mécanisme qui garantit la qualité de l'affichage au détriment de la vitesse : une colonne de filtrage contient un masque de bits à chaque ligne. Chaque composant de requête dynamique est associé à sa colonne de filtrage qui lui assigne un numéro de bit unique. Le composant se contente donc de mettre son bit à 1 pour chaque ligne filtrée. Une ligne est finalement filtrée si un de ses bits est à 1. Ce mécanisme est plus rapide, de plusieurs ordres de grandeur que l'affichage et ne nécessite pas d'optimisation.

Vitesse d'affichage Pour pallier au problème de l'affichage notoirement lent de Java, nous avons contribué au développement de la bibliothèque Agile2D qui implémente le modèle graphique de Java2D en utilisant la bibliothèque graphique OpenGL. Agile2D permet des accélérations très importantes sur

des machines disposant d'une carte graphique accélérée. Par exemple, les visualisations en coordonnées parallèles ou les séries temporelles sont 200 fois plus rapide avec Agile2D qu'avec Java2D natif sur notre ordinateur portable (Dell Latitude M50, 1,8GHz avec carte graphique Quadro4GL). Ce rapport a tendance à augmenter sur les machines plus récentes car Java n'utilise absolument pas les accélérations graphiques matérielles [6] qui pourtant s'améliorent rapidement.

InfoVis n'est pas dépendant d'Agile2D. Les visualisations n'utilisent que des fonctions graphiques standard de Java2D. Néanmoins, pour quelques opérations, nous utilisons des mécanismes qui sont particulièrement efficaces avec Agile2D, offrant des accélérations de l'ordre de 1000 par rapport à Java2D natif. Par exemple, nous avons introduit la notion de graphique « par lots » : plusieurs rectangles peuvent être dessinés en un seul appel de méthode. La version non-accelérée de cette méthode affiche chaque rectangle à l'aide des méthodes Java standard. La version accélérée utilise l'affichage par « Vertex Array » d'OpenGL qui bénéficie de toutes les accélérations possibles, soit entre 6 et 20 millions de rectangles par seconde sur les cartes actuelles.

Factories et comportement par défaut

Les mécanismes décrits jusqu'à présent sont flexibles et modulaires mais peuvent paraître complexes. Pour simplifier la construction d'applications, InfoVis utilise intensivement les « factories » [12], qui sont des objets facilitant la création d'objets complexes.

L'application la plus simple de visualisation consiste à charger un jeu de données et à le visualiser ensuite. Pour cela, InfoVis offre une « factory » permettant de charger les données simplement (Figure 5).

```
Tree tree = new DefaultTree();
AbstractReader reader =
    TreeReaderFactory.createReader(
        args[0], tree);
if (reader == null || !reader.load())
    Erreur !
...
```

Figure 5: Création d'une structure d'arbre et chargement de son contenu à partir d'un argument passé au programme principal et d'une factory de lecture.

Une fois la visualisation créée, une factory permet de créer le panneau de contrôle adapté, contenant les interacteurs de requêtes dynamiques et les sous panneaux de configuration pour cette visualisation. Ce mécanisme très général permet aux applications simples de rester simples tout en permettant aux applications plus complexes de contrôler finement ou d'étendre de nombreux aspects d'InfoVis.

EXEMPLES

L'utilisateur d'une boîte à outils est le programmeur. Nous lui présentons cinq exemples pour évaluer la simplicité, modularité et extensibilité d'InfoVis :

- la réalisation de coordonnées parallèles ;

- la modification de la visualisation cartésienne d'arbres nœud lien en visualisation radiale ;
- la visualisation de graphes par Treemap et liens superposés ;
- la technique d'animation de liens pour améliorer la lisibilité des diagrammes nœud lien ;
- l'affichage d'un répertoire d'images sous forme de treemaps.

```
public class ParallelCoordinatesVisualization
    extends TimeSeriesVisualization {
    public ParallelCoordinatesVisualization(Table table) {
        super(table);
    }
    public void paintBackground(Graphics2D graphics,
        Rectangle2D bounds) {
        super.paintBackground(graphics, bounds);
        double sx = bounds.getWidth()/(columns.size()-1);
        graphics.setColor(Color.BLACK);
        for (int i = 0; i < columns.size(); i++) {
            int x = (int)(sx * i + bounds.getX());
            graphics.drawLine(x, (int) bounds.getY(),
                x, (int) bounds.getHeight());
        }
    }
    public void computeShapes(Rectangle2D bounds) {
        double sx = bounds.getWidth()/(columns.size()-1);
        for (RowIterator iter = iterator(); iter.hasNext();) {
            int i = iter.nextRow();
            GeneralPath p = new GeneralPath();
            for (int col = 0; col < columns.size(); col++) {
                NumberColumn n = getNumberColumnAt(col);
                double min = n.getDoubleMin();
                double max = n.getDoubleMax();
                double diff = (max - min);
                double sy = bounds.getHeight() / diff;
                float x = (float) (sx * col + bounds.getX());
                float h = (float) (sy * (n.getDoubleAt(i) +
                    min));
                float y =
                    (float) (bounds.getY()+bounds.getHeight() -
                        h);
                if (col == 0) {
                    p.moveTo(x, y);
                } else {
                    p.lineTo(x, y);
                }
            }
            setShapeAt(i, p);
        }
    }
}
```

Figure 6: Implémentation de la visualisation par coordonnées parallèles avec InfoVis

Coordonnées parallèles

Nous avons demandé à des étudiants de DESS informatique d'implémenter la visualisation par coordonnées parallèles, certains à l'aide d'InfoVis et d'autres avec le système de leur choix. L'implémentation avec InfoVis requiert 96 lignes de Java (Figure 6). La plupart des groupes ont ajoutés des mécanismes d'interaction pour manipuler directement les axes car le travail leur semblait dérisoire. D'autres groupes ont choisi un autre langage que Java ou de ne pas utiliser InfoVis. Le nombre de lignes de code nécessaire a été bien supérieur (de 600 à 6000) pour des résultats honorables mais beaucoup moins de fonctionnalités. Il a fallu une journée de travail à un étudiant de maîtrise pour réaliser cette même visualisation. Le problème rencontré par les étudiants étant lié à la programmation objet : ils ont tous commencé par copier la classe de base pour modifier les méthodes avant de supprimer les méthodes non modifiées. Cette visualisation est maintenant disponible dans InfoVis.

Représentation radiale d'arbre

La transformation radiale d'une visualisation d'arbre nécessite 37 lignes de Java et a demandé une journée de travail à un étudiant de maîtrise, la plus grande partie de son temps étant consacré à réviser ses formules trigonométriques.

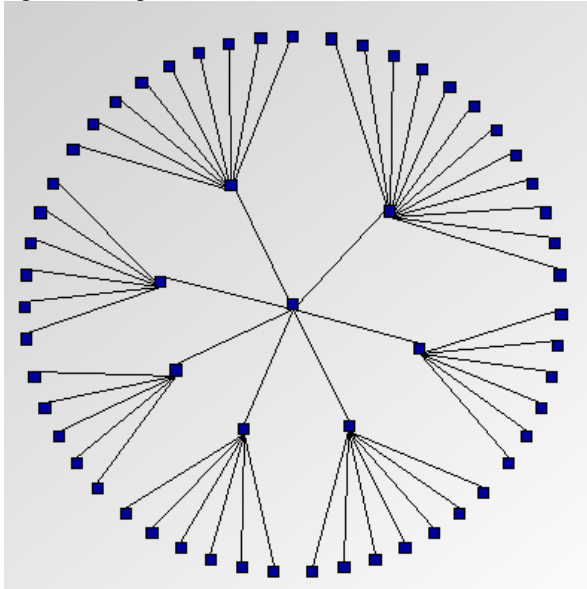


Figure 7: Représentation d'un arbre radial

Représentation de graphe par treemap + liens

Dans [7], nous avons présenté une technique pour représenter un graphe sous forme d'un treemap et de liens superposés. Pour réaliser cette technique dans InfoVis, nous avons dû construire une visualisation superposant un treemap et une représentation nœud lien. La superposition de visualisations étant prévue, cet aspect ne pose aucun problème particulier. Une seconde originalité consiste à représenter les liens sans flèches. Pour cela, nous utilisons un segment de courbe de Bézier quadratique dont la courbure est biaisée vers le point de départ (Figure 8), ce qui évite les superpositions des liens rectilignes et l'occlusion produite par le dessin des flèches. Dans InfoVis, le dessin des liens est contrôlé par un objet *LinkShaper* que nous avons dérivé pour dessiner cette courbe (50 lignes).

L'application de cette visualisation à un site Web a nécessité la lecture du graphe des pages Web, disponible dans InfoVis avec la classe *HTMLGraphReader* et la séparation du graphe en arbre et liens (50 lignes).

Plusieurs stratégies d'interaction ont pu être expérimentées à l'aide de quelques lignes de programme : visualisation de tous les liens, restriction aux liens partant ou arrivant à la sélection, sélection fixe ou suivant la souris. Les trois modes sont utiles donc le paramétrage du mode a nécessité la modification du panneau de contrôle lié à cette visualisation pour offrir le choix aux utilisateurs.

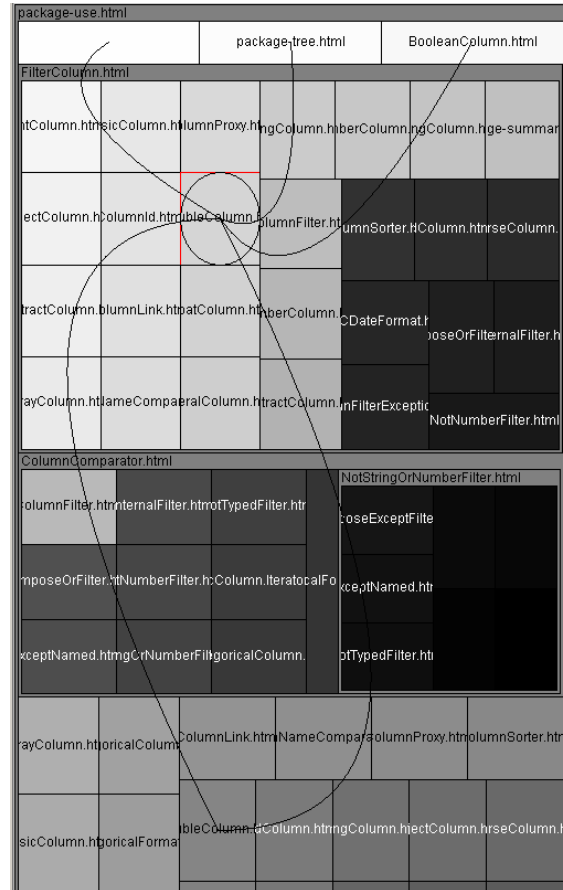


Figure 8: Affichage de liens sur un Treemap visualisant des pages de manuel d'InfoVis.

EdgeLens

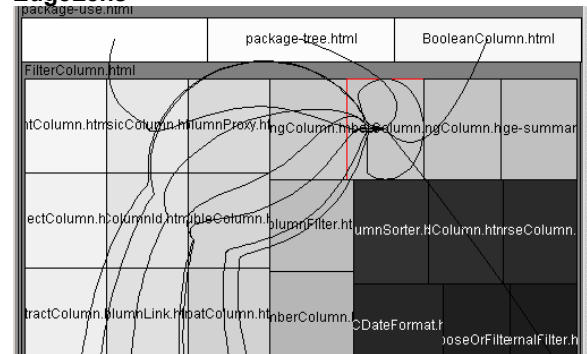


Figure 9: "EdgeLens" dans InfoVis

En 2003, Wong et Carpendale [20] ont proposé une visualisation dynamique pour améliorer la lisibilité des graphes en écartant les liens placés sous le curseur afin de diminuer la « congestion » des liens. Nous avons ajouté cette technique dans InfoVis simplement en restreignant l'application du Fisheye à la couche de visualisation des liens (Figure 9).

L'article original propose plusieurs méthodes pour déplacer les liens : localement et globalement. Le déplacement global nécessiterait une nouvelle implémentation de la classe *Fisheye* que nous n'avons pas réalisée mais qui serait beaucoup plus simple que la version actuelle.

Image dans les visualisations

Nous avons réalisé une visualisation de répertoires d'images sous forme de Treemap avec miniature (Figure 10). La principale difficulté réside dans la gestion des images, pas dans leur affichage. La classe de visualisation a nécessité 200 lignes de Java dont 40 pour la visualisation et 160 pour le chargement et le redimensionnement asynchrones d'images.

Cette représentation est similaire à celle de PhotoMesa [2], bien que nous n'ayons implémenté que les visualisations ordonnées, pas les « Quantum visualisations » ni les « Bubble maps » décrits dans l'article faute de temps.

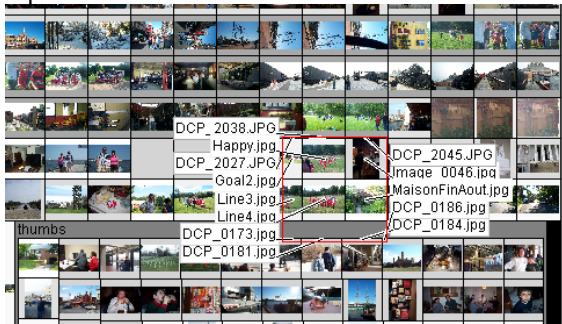


Figure 10: Visualisation de répertoires d'images dans un Treemap avec InfoVis

EVALUATION D'UNE BOÎTE À OUTILS

Comment juger de la qualité d'une boîte à outils ? Trop souvent, un étudiant, un chercheur ou un industriel construit une application reposant sur une bibliothèque qu'il s'empresse de qualifier de générique et réutilisable. Une première étape consiste à indiquer clairement les propriétés importantes d'InfoVis. Dans [8, 15], nous proposons six critères pour comparer les environnements de développements. Les voici appliqués à InfoVis :

Parties de l'application construite par l'environnement : les structures de données du noyau, la présentation et l'interaction.

Temps d'apprentissage : moyennement long (semaines), comme les squelettes d'application.

Temps de construction : très court (heures).

Méthode de construction préconisée ou imposée : structures de données du noyau d'abord, visualisations ensuite, interactions si nécessaire puis panneaux de contrôle spécifiques éventuellement.

Liaisons avec d'autres sous-systèmes : facilité de lecture/écriture de formats de fichiers spécifiques, utilisation des mécanismes de communication standard de Java/Swing (*Listener*, *Modèles*, etc.).

Extensibilité et modularité : comme tous les squelettes, très extensible et modulaire dans les limites des mécanismes prévus.

Il existe quelques boîtes à outils de visualisation d'information : GGobi [17], *XML Toolkit*[3], Polaris [16] et GeoVista studio [18]. GGobi et Polaris sont spécialisés pour les tables. Polaris est le système le plus proche d'InfoVis. Il est écrit en C++ et utilise OpenGL pour son affichage. Ses données sont aussi organisées

en mémoire mais sous forme de tuples et non de colonnes. Une particularité de Polaris est de pouvoir répartir la priorité d'affichage entre ses visualisations afin de fonctionner en temps réel. Polaris n'est pas encore distribué, il est donc difficile d'en savoir plus sur ses structures internes. XML Toolkit est plutôt un catalogue d'algorithmes qu'une boîte à outils. Il repose sur les structures internes de Java/Swing qui ne sont pas conçues pour la visualisation et font une utilisation lente et volumineuse de la mémoire. GeoVista est une bibliothèque de composants reposant sur les Java Beans [5] qui permet de configurer par programmation visuelle des visualisations. Pour nous, GeoVista est une couche supérieure à InfoVis qui en réutilise des composants : les labels excentriques et les Fisheyes en particulier.

En dehors de ces systèmes, plusieurs boîtes à outils existent pour la visualisation scientifique, mais leur organisation est un peu différente de celle d'InfoVis car les problèmes principaux sont la représentation ainsi que la navigation 3D efficaces. Dans InfoVis, nous ne proposons que des visualisations intrinsèquement 2D.

Enfin, il existe plusieurs systèmes de visualisation d'information qui sont extensibles jusqu'à un certain point : Spotfire (www.spotfire.com), Ilog Discovery (www.ilog.com/preview/Discovery/), Treemap4 (www.cs.umd.edu/hcil/treemap), etc. Ces systèmes sont très complets, mais ils sont fermés et dédiés à un ensemble prédéterminé de structures de données, techniques de visualisation et techniques d'interaction. Spotfire accepte de plugins programmés en C++ qui peuvent porter sur l'importation/exportation de données, la communication avec des applications et certains calculs, mais le contrôle des visualisations et des interactions reste limité. Ilog Discovery propose un langage d'extension pour calculer de nouvelles visualisations, mais ces visualisations sont limitées et ne peuvent pas s'étendre à des diagrammes nœud lien par exemple.

Enfin, InfoVis est principalement inspiré de plusieurs systèmes de visualisation développés par ou en collaboration avec l'auteur d'InfoVis comme MillionVis [10], VisADJ [13], Treemap4 et SpaceTree [14].

PERSPECTIVES ET CONCLUSIONS

La boîte à outils InfoVis fournit à la fois un cadre de développement de nouvelles visualisations et une riche bibliothèque de techniques de visualisation. Nous avons décrit son architecture afin de montrer qu'elle reposait sur quelques concepts et mécanismes simples et très flexibles : des tables organisées en colonnes, des visualisations liées à des tables et paramétrées par des colonnes. Nous avons montré quelques exemples de visualisations non-standards réalisées simplement à l'aide d'InfoVis. InfoVis nous sert actuellement à concevoir, expérimenter et développer de nouvelles visualisations et techniques d'interaction. Celles-ci à leur tour nourrissent InfoVis, ce qui rend l'évolution de la boîte à outils assez rapide tout en restant stable.

Parmi les utilisations d'InfoVis, notons la visualisation d'exécution de programmes par contraintes qui requiert la visualisation animée, temps réel et permettant un filtrage sur le temps, orthogonal au filtrage sur les valeurs. Nous comptons bientôt incorporer des mécanismes de gestion du temps, en particulier l'animation de visualisations.

Nous pensons qu'InfoVis est une bonne base pédagogique qui devrait permettre de transférer rapidement un savoir faire complexe à des étudiants, étape importante vers la diffusion de la visualisation d'information vers l'industrie.

Enfin, nous voudrions encourager les professionnels – chercheurs, enseignants ou industriels – à utiliser et critiquer InfoVis (www.lri.fr/~fekete/InfoVisToolkit) qui est distribuée avec une licence libre et à contribuer à l'enrichir par de nouveaux composants, de nouvelles visualisations ou de nouvelles interactions. Nous pensons que l'utilisation d'InfoVis facilite l'innovation en séparant les problèmes de visualisation, de structure de données et d'interaction tout en offrant une riche bibliothèque de techniques prêtes à l'emploi. La visualisation d'information gagnerait certainement à éviter le problème de fragmentation, très visible dans les boîtes à outils graphiques, qui – hors situation de rupture technologique – nous semble aller à l'encontre de l'innovation réelle

BIBLIOGRAPHIE

1. AT&T Labs Research. Graphviz - open source graph drawing software, 2004.
2. Bederson, B.B., PhotoMesa: A Zoomable Image Browser Using Quantum Treemaps and Bubblemaps. in *Proceedings of the 14th annual ACM symposium on User interface software and technology*, (Orlando, Florida, 2001), ACM Press, 71 - 80.
3. Börner, K. and Zhou, Y., A Software Repository for Education and Research in Information Visualization. in *Information Visualisation Conference*, (London, England, 2001), 257-262.
4. Carpendale, M.S.T. and Montagnese, C., A framework for unifying presentation space. in *Proceedings of the 14th annual ACM symposium on User interface software and technology*, (Orlando, Florida, 2001), ACM Press, 61-70.
5. Englander, R. *Developing Java Beans*. O'Reilly & Associates, 1997.
6. Fekete, J.-D. The InfoVis Toolkit. Report, I.F.R. ed., INRIA Futurs, Orsay, 2003, 15.
7. Fekete, J.-D., Dang, N., Plaisant, C. and Wang, D., Interactive Poster: Overlaying Graph Links on Treemaps. in *IEEE Symposium on Information Visualization*, (Seattle, WA, 2003).
8. Fekete, J.-D. and Girard, P. Environnements de développement de systèmes interactifs. in Kolski, C. ed. *Systèmes d'information et Interaction Homme-Machine*, Hermes, 2001.
9. Fekete, J.-D. and Plaisant, C., Excentric Labeling: Dynamic Neighborhood Labeling for Data Visualization. in *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems*, (1999), 512-519.
10. Fekete, J.-D. and Plaisant, C., Interactive Information Visualization of a Million Items. in *IEEE Symposium on Information Visualization (InfoVis'02)*, (Boston, MA, 2002), IEEE Press, 117-124.
11. Gaines, B.R. Modeling and forecasting the information sciences. *Information Sciences: an International Journal, Special issue on information sciences—past, present, and future*, 57-58. 3 - 22.
12. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publisher, Reading, MA, USA, 1994.
13. Ghoniem, M., Jussien, N. and Fekete, J.-D., VI-SEXP: visualizing constraint solver dynamics using explanations. in *FLAIRS'04: Seventeenth international Florida Artificial Intelligence Research Society conference*, (Miami Beach, FL, 2004), AAAI press.
14. Grosjean, J., Plaisant, C. and Bederson, B.B., SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation. in *IEEE Symposium on Information Visualization (InfoVis'02)*, (Boston, MA, 2002), IEEE Press, 57 - 64.
15. Shneiderman, B. and Plaisant, C. *Designing the User Interface*. Addison-Wesley Publisher, 2004.
16. Stolte, C., Tang, D. and Hanrahan, P. Polaris: A System for Query, Analysis and Visualization of Multi-dimensional Relational Databases. *IEEE Transactions on Visualization and Computer Graphics*, 8 (1). 52-65.
17. Symanzik, J., Swayne, D.F., Lang, D.T. and Cook, D., Software Integration for Multivariate Exploratory Spatial Data Analysis. in *Proceedings of the SCISS Specialist Meeting "New Tools for Spatial Data Analysis"*, (Santa Barbara, CA, 2002).
18. Takatsuka, M. and Gahegan, M. GeoVISTA Studio: A Codeless Visual Programming Environment For Geoscientific Data Analysis and Visualization. *The Journal of Computers & Geosciences*, 28 (10). 1131-1144.
19. Tanin, E., Beigel, R. and Shneiderman, B. Incremental data Structures and Algorithms for Dynamic Query Interfaces. *SIGMOD Record*, 25 (4). 21-24.
20. Wong, N., Carpendale, S. and Greenberg, S., EdgeLens: An Interactive Method for Managing Edge Congestion in Graphs. in *2003 IEEE Symposium on Information Visualization*, (Seattle, WA, 2003), IEEE Press, 52-58.