Rapport de TER : Visualisations Responsives

Juliana LECLAIRE

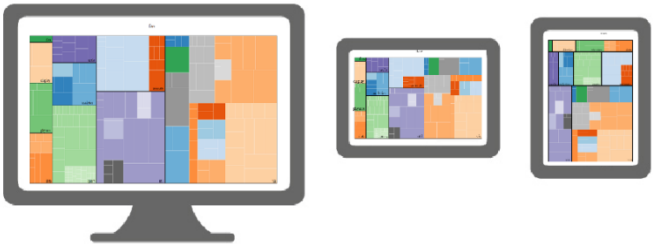
25-02-2015

Résumé

Ce document présente l'étude des visualisations responsives. Ce projet a pour but d'explorer comment des visualisations Web pourraient être adaptées aux dispositifs sur lesquelles elles s'affichent en s'adaptant aux propriétés du dispositif (taille de l'écran, modalité d'interaction).

Page web: http://juliana23.github.io/responsiveVisualisations/

Mots-clés : visualisations responsives, responsive design, D3.js, visualisation d'information, mobile.



1) Introduction

L'accroissement des données disponibles en lignes (open data, big data, web sémantique, géo localisation, ou produites par les réseaux sociaux, bases de données, etc.) a fait naître des outils de « Data Visualization » afin de rendre leurs analyses plus explicites et d'en saisir les informations principales en un seul coup d'œil.

Dans le cadre du Master 1ère année, j'ai réalisé un Travail d'Etude et de Recherche d'une durée de six semaines concernant les visualisations de données sur le Web. J'ai effectué ce projet sous la tutelle d'Aurélien Tabard, maître de conférences et membre du Laboratoire d' InfoRmatique en Image et Système d'informations.

Ce travail de recherche consiste à étudier différentes visualisations de données afin qu'elles s'adaptent à différents supports. Cette étude est ciblée principalement sur les périphériques de type smartphones, tablettes et ordinateurs. Les visualisations doivent pouvoir prendre en compte différents facteurs car tous n'ont pas les mêmes capacités. Au cours de ce projet je me suis penchée plus précisément sur les trois points suivants : les possibilités d'interaction, la taille et la résolution des écrans. Ma réflexion a porté sur les possibilités d'extension de la librairie d3.js pour que des objets génériques permettant la construction de visualisations, communs aux différents types de visualisations, s'adaptent simplement au support. Afin de respecter au mieux les adaptations j'ai tenu à veiller aux points suivants :

- la quantité d'informations permettant d'améliorer la lisibilité
- les interactions utilisateurs
- le passage des évènements souris au tactile et inversement

Ces critères sont nécessaires dans la gestion du responsive design. Ces objets doivent principalement permettre au développeur de ne pas vérifier à chaque redimensionnement la taille de l'écran, le type de périphérique utilisé, etc.

2) Etat de l'art

2.1. Les visualisations pour mobiles et tablettes

Ces dernières années plusieurs projets ont porté sur la visualisation pour mobiles. Dominikus Baur travaille sur les visualisations de données et les interactions mobiles, et recherche comment tenir au mieux des interactions sur ces écrans. Ses recherches sont évoquées dans une conférence vidéo[1]. De plus Baur, Lee et Carpendale, [BLC12], ont réalisé un projet TouchWave mettant en avant une multitude d'interactions que nous pouvons faire sur un support tactile pour un graphique empilé. D'après eux, tout contrôle avec les doigts au lieu d'un seul curseur de la souris change radicalement les exigences et les capacités des interfaces. Ils montrent comment amener un graphe empilé, pas vraiment compréhensible sur ordinateur avec seulement la souris et le clavier comme interactions, à des interactions multi-touch pour faciliter la compréhension. En outre, ils montrent comment aborder les questions de graphique empilé avec un jeu simple et complet d'interactions tactiles.

Un projet sur les mêmes principes est Kinetica réalisé par M. Rzeszotarski et Kittur, [RK14], de l'équipe HCI de Carnegie Mellon. L'idée est de faire réagir chaque point de données comme un point réel sur l'écran et de donner aux utilisateurs plusieurs outils interactifs inspirés de la physique, pour explorer les jeux de données.

D'autres chercheurs comme Sadowski et Heidmann, [SH02], se sont penchés sur la question « quel contenu et sous quelle forme devrais-je visualiser sur les mobiles? ». Leur objectif est de savoir comment adapter les données au support et ne pas seulement reproduire la visualisation en plus petit ou plus grand. Pour cela ils ont fait la distinction entre les appareils mobiles et les ordinateurs de bureau sur la base de quatre propriétés (la taille de l'écran, les interactions, la portabilité, et les capteurs).

Un problème récurrent sur les mobiles est le problème du « gros doigt ». Il a conduit à une recherche entreprise par Sebastian Boring, David Ledo, Xiang 'Anthony' Chen, Nicolai Marquardt, Anthony Tang et Saul Greenberg, [BLCMTG12]. Les utilisateurs de smartphones ont généralement tendance à utiliser le pouce pour manipuler leur appareil car il est souvent le seul doigt qui reste disponible. Dans leur recherche ils intègrent une nouvelle dimension à prendre en compte : la taille du pouce pour compléter la position x, y.

Ces études m'ont permis de comprendre plusieurs difficultés existantes pour les visualisations sur les supports tactiles. Différents dispositifs disposent de différentes tailles d'écran. Les ordinateurs de bureau ont des tailles d'écran de 19" à 27" diagonale. Les ordinateurs portables disposent généralement de tailles d'écran entre 13" et 17", les smartphones de 5" et 10" de diagonale. Outre la taille de l'écran, le rapport largeur-hauteur sur smartphone est différent des rapports habituels, les smartphones sont généralement maintenus verticalement dans la main. En raison de cette distinction, les visualisations développées pour les ordinateurs de bureau n'évoluent pas bien sur les appareils mobiles. Les smartphones offrent différentes possibilités d'interaction telles que le multi-doigt et la manipulation directe. Cependant il faut veiller au problème du « gros doigt » car les objets tactiles cibles sont plus petits sur les écrans tactiles. De plus du fait de la portabilité des smartphones l'attention de l'utilisateur n'est pas pleinement focalisée sur le support. Il faut donc que l'information soit compréhensible en un coup d'œil.

La difficulté est de savoir comment tenir au mieux les interactions sur les supports, notamment sur des supports de petites tailles.

[1] http://do.minik.us/blog/jsconf_talk

2.2 Les visualisations responsive

Quelques acteurs ont travaillé sur les visualisations responsive ayant pour but de créer des visualisations pour différents supports en utilisant le même code source. Gabriel Florit a créé des visualisations de données au Boston Globe et parle des difficultés d'adapter les visualisations de données aux différents supports. Celles-ci sont décrites dans une conférence vidéo[2]. Un des principaux problèmes est d'adapter la visualisation pour des tailles d'écrans réduites. Il travaillait sur une visualisation de nuage de points et se demandait comment naviguer entre les points du fait que celles-ci étaient des cibles minuscules. Il a évoqué plusieurs solutions en utilisant des interactions utilisateurs.

Mis à part ce projet, je n'ai pas réellement trouvé de recherche sur cette approche. Le principal problème est que les visualisations créées sont souvent destinées à un seul type de support. Les visualisations qui sont parfaitement lisibles sur un écran d'ordinateur peuvent finir par être inutilisable sur un écran mobile. Cette situation peut être aggravée par l'interactivité. Des modes bien établis d'interaction comme le zoom ont parfois des modes très différents d'interactions sur le toucher par rapport à une souris ou un clavier.

Une des visualisations que je souhaite étendre est la visualisation Treemap. Renaud Blanch et Éric Lecolinet, [BL07], ont proposé des visualisations arborescentes pour de grands ensembles de données. Ils sont partis du constat que les Treemaps ne sont pas pratiques pour explorer de grands ensembles de données surtout lors de l'accès aux détails. Ils ont pensé être nécessaires d'introduire différentes interactions utilisateurs permettant de naviguer en profondeur. D'après eux l'interaction primordiale pour ce problème est le système de zoom. Ils se sont donc focalisés sur cet aspect et ont déterminé différents points clés :

- Utiliser des couleurs pour déterminer la hiérarchie.
- Augmenter la lisibilité en entourant les lettres d'un contour blanc.
- Ne pas afficher le texte d'un nœud si le rendu est trop petit.
- Ne pas afficher toute la branche au-delà d'un certain seuil.
- Veiller à ce qu'il y ait toujours un nœud courant.
- Ne pas afficher les nœuds de la même manière pour optimiser le rendu.

Le zoom et ces points clés sont une base pour adapter la visualisation à une classe plus large de dispositifs.

2.3 Les extensions de d3.js

D3.js est une bibliothèque graphique JavaScript. Elle offre des capacités à lier de manière simplifiée des données d'un fichier au DOM éléments. Michael Bostock, Vadim Ogievetsky et Jeffrey Heer, [BOH11], montrent les performances de d3.js[3] comparé aux approches antérieures. D3.js permet notamment la manipulation efficace des documents fondés sur des données et les lient à des éléments arbitraires du DOM. Il existe plusieurs librairies qui étendent d3.js comme c3.js[4]. C3 permet de générer des graphiques à partir de d3 en enveloppant le code nécessaire pour construire l'ensemble du graphique. L'inconvénient de c3.js est qu'il ne propose pas nativement d'objets responsive.

[2] https://www.youtube.com/watch?v=BrmwjVdaxMM

[3] http://d3js.org/

[4] http://c3js.org/

2.4 Interfaces plastiques

Les visualisations responsives ont émergé depuis peu. Cependant il existait déjà les interfaces dites plastiques, une notion également liée à l'adaptation des interfaces par rapport à leur support. Gaëlle Calvary et Joëlle Coutaz, [CC02], ont travaillé sur ce point et ont ciblé plusieurs problématiques. La principale est que l'IHM d'un système interactif ne peut pas être la même sur tous les supports du fait que les ressources matérielles (taille de l'écran, absence de clavier, etc.) ne sont pas identiques. Depuis l'utilisation des appareils électroniques équipés d'interfaces, la problématique d'adapter au support est présente. Leur fil conducteur pour établir des interfaces adaptatives est de respecter continuellement son utilité et son utilisabilité. En effet, les différentes capacités du support (taille de l'écran, dispositifs d'interaction, capacités de calcul, etc.) doivent influencer le type d'interaction mise en place.

3) Problématique

3.1 Comment passer d'un support à un autre

a. Problème de précision

Lorsque nous utilisons un appareil avec une souris, le clic de la souris est précis pour naviguer. En revanche sur les supports tactiles le problème du « gros doigts » est récurrent car les cibles sont souvent plus petites. La navigation, en touchant des cibles minuscules, est pratiquement impossible.

b. Problème des évènements souris/tactiles

Selon la fonctionnalité et le support les équivalences des interactions souris et tactiles ne peuvent pas forcément être respectées. Prenons l'exemple d'une fonctionnalité déclenchée sur un évènement « hover ». Lorsque nous sommes sur des écrans équipés d'une souris cet évènement est pris en compte. En revanche il est physiquement impossible à réaliser sur des écrans tactiles.

c. Taille / Résolution

La taille et la résolution sont des aspects à prendre en compte pour passer d'un support à un autre. En effet les supports n'ont pas tous la même résolution d'écran. Nous ne pouvons donc pas afficher l'information de manière identique sinon elle serait illisible sur des écrans à résolution faible. De plus, plus la taille de l'écran est petite pour une même résolution, plus la taille des éléments est petite. Il faudra donc veiller à ne pas surcharger la visualisation.

d. Exemple

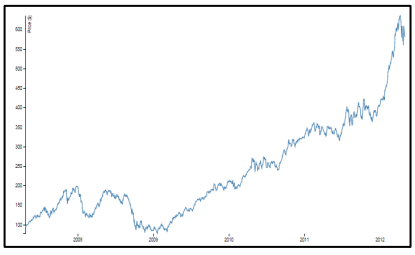


Figure 1: Timeline d3.js

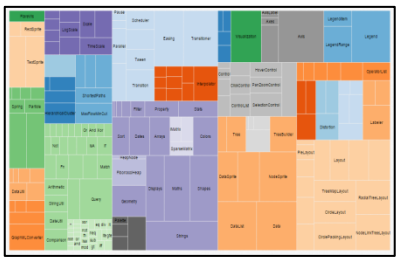


Figure 2: Treemap d3.js

La Figure 1 est un exemple de visualisation basique proposée par la librairie d3.js. Celle-ci présente les différents problèmes suivants :

- La taille de la courbe et des axes ne sont pas mis à jour lors d'un redimensionnement de la fenêtre.
- La quantité d'information affichée n'est pas variable en fonction de la taille et de la résolution de l'écran. Une courbe qui présente beaucoup de variations ne sera pas gérée en diminuant la quantité d'information par pixel.

- Il est difficile d'observer des zones ou des points précis de la courbe.
- En cas d'ajout d'interaction sur la courbe, le problème du « gros doigt » est rapidement présent.

La Figure 2 est aussi une visualisation proposée par la librairie d3.js. Plusieurs inconvénients sont présents sur cette représentation si nous la visualisons sur différents supports :

- Les labels sont affichés dès le chargement de la visualisation et s'il n'y a pas suffisamment de place dans le rectangle alors ils ne sont pas affichés. Cela ne pose aucun problème sur de grands écrans car tous les labels seront affichés et l'information sera complète mais qu'en est-il sur des écrans où la plupart des labels ne peuvent pas s'afficher ?
- Il n'y a pas d'interaction utilisateurs, une des solutions de remédiation à la surcharge d'information. En effet les interactions permettent d'ajouter de l'information avec parcimonie, évitant donc d'afficher toutes l'information dès le chargement de la visualisation.
- La disposition des rectangles n'est pas adaptée selon la taille de l'écran.

Cette recherche consiste dans un premier temps à trouver quels outils pourraient être ajoutés à ces visualisations dans le but d'améliorer la lisibilité quel que soit le support.

3.2 Avantages suivant le support

a. Les ordinateurs

En règle générale les ordinateurs offrent une meilleure précision des évènements avec la souris, de navigation ainsi qu'une taille d'écran plus confortable pour la lecture de données.

b. Les smartphones / tablettes

Les smartphones offrent différentes possibilités d'interaction telles que le multi-doigt et la manipulation directe. Du fait de la portabilité des smartphones l'attention de l'utilisateur n'est pas pleinement focalisée sur le support. Il faut donc que l'information soit compréhensible en un coup d'œil. Les smartphones contiennent des capteurs tels que le GPS, l'accéléromètre ou un capteur de lumière. Ces capteurs pourraient être utilisés pour cibler le contexte de l'utilisateur et proposer une visualisation mieux adaptée. Les visualisations pourraient être au courant de la situation géographique (par exemple filtrer les données en fonction de la position de l'utilisateur) ou des conditions d'éclairage (par exemple, adapter la luminosité ou le contraste).

Du fait qu'il faille optimiser l'espace sur un smartphone on aura plus tendance à essayer, dans un premier temps, de cibler les informations avec des graphiques simples. Par la suite l'utilisateur pourra avoir une visualisation plus détaillée, grâce à des interactions utilisateurs adaptées. Cet aspect contribuera davantage à la compréhension de la visualisation.

4) Méthodes

J'ai réalisé un travail préliminaire sur les visualisations Timeline et Treemap afin de chercher comment remédier aux problématiques liées aux visualisations responsives, évoquées dans les parties précédentes. Après avoir établi une liste des outils et fonctionnalités clés et communes permettant d'adapter les visualisations sur différents supports, j'ai implémenté la première version d'une librairie dans l'objectif de proposer des objets génériques pouvant être utilisés par des développeurs souhaitant créer des visualisations responsives à partir de d3.js.

5) Travail préliminaire

5.1. Visualisation Timeline

Figure 3: Timeline responsive

Afin de pallier au problème de lisibilité sur la visualisation sur différentes tailles d'écran j'ai utilisé des médias queries css3. Elles permettent de mettre à jour de manière dynamique les labels des axes et la taille des étiquettes en fonction de la taille de l'écran. La Figure 3 montre la capacité de la visualisation Timeline à s'adapter à différents supports.

La visualisation gère la quantité d'information sur les axes en fonction de la taille du support. Cette option a pour but d'améliorer la lisibilité en évitant la surcharge des axes. (Cf 7)Annexe 1.a.i. Figure 6/7)

Timeline de d3.js n'avait aucune interaction utilisateur. De plus majoritairement lorsque nous regardons une Timeline nous voulons voir l'aspect général mais aussi connaitre la valeur spécifique en un point choisi, la valeur d'un sommet par exemple. Un système d'étiquette serait pertinent pour ce problème. Cependant pour que cette fonctionnalité soit adaptée sur différents supports, il a fallu veillez au problème du « gros doigt » sur le tactile. En effet si les étiquettes s'affichaient en sélectionnant un point précis de la courbe, cette fonctionnalité serait inutilisable sur de petits écrans tactiles. Pour pallier à ces éventualités l'étiquettes s'affiche dès qu'on interagit sur la visualisation et va s'afficher pour le point correspondant au mieux de l'endroit où nous avons fait l'interaction. (Cf 7)Annexe 1.a.ii. Figure 8)

Un système de zoom a été mis en place pour bonifier les observations de zones précises. L'utilisateur a la possibilité de sélectionner une partie de la courbe qu'il souhaite zoomer. Cette fonctionnalité applique un changement d'échelle. (Cf 7)Annexe 1.a.iii. Figure 9/10)

Fonctionnalité	Description	Evènement Mouse / Touch
Etiquette	Informations de l'abscisse et de l'ordonnée correspondant à l'emplacement du curseur	Mousemove / Touchmove
Zoom	Système pour zoomer sur une partie de la courbe en changeant l'échelle	X / X
Responsive labels, étiquette	Utilisation des media queries	X / X

Tableau 1: Fonctionnalités sur Timeline

5.2 Visualisation Treemap

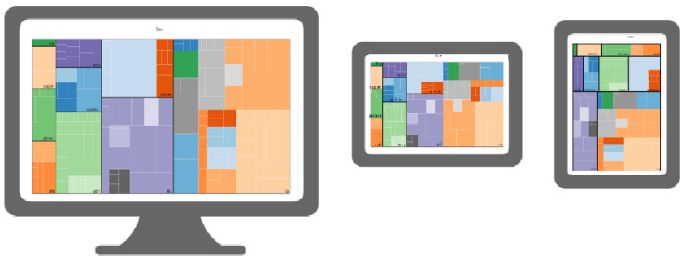


Figure 4: Treemap responsive

Afin de pallier au problème de lisibilité sur la visualisation sur différentes tailles d'écran j'ai utilisé des médias queries css3, les mêmes que pour Timeline. La Figure 4 montre la capacité de la visualisation Treemap à s'adapter à différents supports.

Pour éviter la surcharge d'information, j'ai fait le choix de ne plus afficher les labels des feuilles de l'arborescence lorsqu'on arrive sur la visualisation. Cependant afin de donner une direction de recherche à l'utilisateur, l'affichage des premiers parents de chaque « parent racine » est effectué. (Cf 7)Annexe 2.a.i. Figure 11)

Pour cette visualisation, une fonctionnalité de highlight (surbrillance) a été ajoutée. Celle-ci offre la possibilité d'identifier rapidement une hiérarchie de premier niveau en affichant en surbrillance le nœud parent sélectionné. (Cf 7)Annexe 2.a.ii. Figure 12)

Associées à la fonctionnalité de highlight, les étiquettes ont permis d'obtenir les noms des feuilles de l'arborescence sans surcharger la visualisation. En effet l'étiquette affiche le nom de tous les « enfants directs » du nœud sélectionné et est associé à un numéro qui se trouve dans chaque rectangle des enfants. La notion de « nœud sélectionné » est établie de la manière suivante :

- Seuls les rectangles feuilles des données de la visualisation sont affichés.
- Lorsque l'utilisateur interagit sur une feuille, je cherche son premier parent. Le parent trouvé, je cherche les « enfants directs » de ce parent pour les mettre dans le même cadre.
- Le parent trouvé est appelé « nœud sélectionné ».

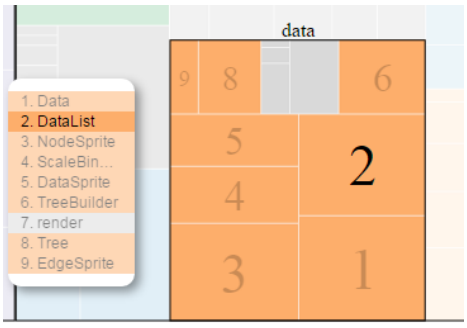


Figure 5: Treemap - Exemple Etiquette

Si nous prenons l'exemple ci-dessus le « nœud sélectionné » est « data » lorsque l'utilisateur interagit sur le nœud feuille numéro 2 « DataList ». Le premier parent de « DataList » est « data » et le nœud « data » n'a pas que des enfants feuille, il a aussi le nœud parent « render » (en gris) qui est un « enfant direct » de « data ». (Cf 7)Annexe 2.a.iii. Figure 13)

Les fonctionnalités précédentes permettent de ne pas surcharger la visualisation pour qu'elle

reste claire et lisible sur différents supports. Cependant sur le premier niveau nous ne pouvons pas obtenir toutes les informations d'un seul coup d'oeil. Un système de zoom permet donc d'atteindre différents niveaux de la hiérarchie et donc de voir les informations nécessaires.

Afin de ne pas faire de sauts en profondeur et par conséquent se perdre dans la hiérarchie, le zoom n'amène que sur une profondeur -1 par rapport au « nœud sélectionné », c'est-à-dire que le zoom s'effectue sur le parent du « nœud sélectionné ».

Chaque zoom remplace le parent racine par le parent du « nœud sélectionné » et le « nœud sélectionné » est mis en avant avec le highlight. (Cf 7)Annexe 2.a.i. Figure 14 / 15 / 16)

Fonctionnalité	Description	Evènement Mouse / Touch
Tooltip (étiquette)	Informations sur les « enfants feuilles » de l'arbre	Mouseover / Touchend
Zoom In / Out	Naviguation en profondeur dans la visualisation	Double Click / Double Tap
Affichage noms parents	Nom des premiers parents du nœud courant	X / X
Affichage de numéros	Numéro associé entre les noms des « enfants feuilles » du tooltip et les rectangles deTreemap	Mouseover / Touchend
HighLight	Luminosité du nœud sélectionné plus élevée	Mouseover / Touchend
Responsive labels, tooltip	Utilisation des media queries	X / X

Tableau 2: Fonctionnalité sur Treemap

6) Travail sur l'aspect responsive générique

L'application des principes du responsive sur les visualisations Timeline et Treemap m'a permis de ressortir différentes fonctionnalités, caractéristiques communes. L'objectif de mes recherches et des développements est de pouvoir créer une API minimale offrant au développeur un panel d'outils lui permettant de gérer facilement les visualisations implémentées quel que soit le périphérique d'affichage utilisé (aussi bien au niveau des interactions qu'au niveau de la taille d'écran). Dans cette partie, j'ai établi une liste des différents objets communs aux visualisations afin d'évoquer la démarche de conception et leurs avantages.

6.1 Objet tooltip (étiquette)

La conception de l'objet tooltip a évoqué plusieurs méthodes d'instanciation. La première est de créer une seule étiquette pour la visualisation et laisser le développeur effectuer des changements dynamiques de son contenu et de sa forme. La deuxième méthode est d'associer un objet tooltip à chaque conteneur. Toutes les étiquettes seront initialisées avec leur contenu et cachées ou non suivant le choix du développeur. La première méthode n'instancie qu'un seul objet pour toute la visualisation. La deuxième en revanche évite de recalculer à chaque fois le contenu du tooltip si le développeur ne souhaite pas changer. L'inconvénient principal de la seconde méthode est que si

nous avons x nœuds parents dans la visualisation alors il nous faudra x étiquettes. La création des x contenus peut entraîner une surcharge évidente du DOM. De plus si les nœuds changent de place sur un redimensionnement, il faut prendre aussi en compte la position de tous les objets attribués initialement ce qui augmente la quantité et donc le temps de calcul. Afin de respecter les principes du responsive un redimensionnement ne doit pas dépasser la demi-seconde. En réfléchissant de manière générique, afin de créer une visualisation, Timeline ou Treemap, avec le même objet tooltip, j'ai pensé que la première solution était mieux adaptée. La principale raison est la suivante : utiliser la deuxième solution signifiait instancier tous les objets pour tous les points de la courbe, ce qui allait réellement surcharger le DOM.

6.2 Objet axis

L'objet axis permet d'afficher les axes de la visualisation de manière responsive. A chaque redimensionnement l'objet se charge de se redessiner sans attendre l'appel du développeur. Pour éviter au développeur de donner une taille à ses axes à chaque fois que l'objet se redessine, un système de ratio a été mis en place. Le développeur fournit une taille initiale à son conteneur. L'objet axis va calculer le ratio initial et le garder en mémoire durant toute sa durée de vie.

Le ratio est calculé de la manière suivante :

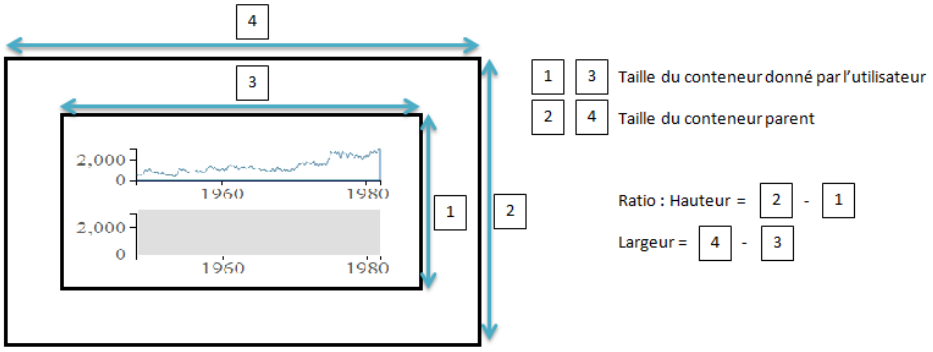


Schéma 1 : Ratio entre la taille du conteneur parent et la taille du conteneur de la visualisation

6.3 Objet brush

Il permet de zoomer par sélection sur une zone précise d'un graphe. Le développeur a la possibilité de l'instancier pour son graphe sans avoir à se soucier des redimensionnements. En effet, l'objet brush réutilise les objets axis.

6.4 Objet event

L'objet event permet au développeur de créer des évènements sur les objets qu'il souhaite en associant une fonction callback à l'évènement à écouter. Lors de la création de son évènement il peut choisir d'étendre ou non son évènement. L'extension permet de créer automatiquement l'objet évènement équivalent pour le support qu'il n'a pas précisé. L'objet event propose des équivalences par défaut entre les évènements souris et tactiles (Cf 7) Annexe 4. Tableau 4). Ces équivalences ont été difficiles à établir car l'évènement correspondant ne dépend pas uniquement de l'évènement d'origine mais aussi de la fonction appelée lors de son exécution. Selon les cas d'utilisation il sera donc préférable d'utiliser certains événements plutôt que d'autres (par exemple l'événement mouseover peut être utilisé comme un touchenter dans certains cas, touchmove dans d'autres cas ou encore comme un simple tap suivant l'action réalisée sur l'événement). Pour ne pas bloquer le développeur, l'API réalisée lui laisse la possibilité de définir ses propres équivalences.

6.5 Utilisation des media queries

De nos jours le responsive design est un atout majeur pour les sites Web. Ces sites Web adaptatifs utilisent les media queries pour adapter leur mise en page via des feuilles de style css3. Les media queries permettent de modifier le style dynamiquement en fonction de la taille du media. Une feuille de style par media ou caractéristiques d'un media peut être établie. Les media queries évitent à l'utilisateur de devoir zoomer/dézoomer pour voir correctement le contenu de la page.

a. Breakpoints

Certains sites Web présentent des media queries standard pour chaque type de media. Dans un premier temps j'ai utilisé un standard, mais en testant les visualisations sur différents type de dispositif (ordinateur, tablette, smartphone) j'ai trouvé que certains breakpoints manquaient. Notamment les breakpoints concernant les supports de petites tailles. J'ai donc modifié pour ces supports afin qu'ils prennent en compte l'orientation portrait ou paysage. Je pense ainsi avoir balayé quasiment l'ensemble des possibilités d'affichage pour les périphériques actuels (Cf 7)Annexe 4. Tableau 4).

b. Media queries configurables avec Less

Less est un langage qui étend du CSS et permet d'utiliser des variables, fonctions, faire de l'héritage avec le CSS. Afin de configurer rapidement les différents breakpoints des media queries, les différentes valeurs CSS pour les labels, la taille des étiquettes, la mise en place d'un fichier .less a été réalisée.

6.6 Objet less

De manière à pouvoir configurer facilement les variables du fichier less, j'ai implémenté une classe JavaScript utilitaire destinée uniquement à la mise à jour des variables less. Celle-ci génère l'ensemble des getters et setters des variables définies dans le fichier less. Le développeur peut ainsi modifier dynamiquement le style des visualisations en faisant à ces méthodes.

7) Conclusion

Au cours de ce travail d'étude et de recherche, j'ai travaillé sur différents aspects dans le domaine du responsive dans l'objectif d'adapter différents types de visualisations sur plusieurs périphériques. Dans un premier temps, l'état de l'art m'a permis de ressortir les principales difficultés pouvant survenir lors de la création de visualisations responsives et m'a apporté un premier niveau de solution. Dans un second temps, j'ai effectué des recherches sur deux visualisations proposés par la librairie d3.js afin de définir comment celles-ci pouvaient être adaptées au mieux sur les différents supports sur lesquels elles s'affichent. Ce premier aspect m'a permis de lister un ensemble de critères à éviter :

- Ne pas submerger d'information la visualisation.
- L'utilisateur ne doit pas utiliser le zoom pour améliorer la lisibilité. On en conclut que la visualisation n'est pas adaptée.
- Ne pas faire de cibles trop petites dues au problème du « gros doigt ».
- Ne pas négliger les différents types d'interaction selon le support.

Le second aspect m'a permis de lister des fonctionnalités pour y remédier :

- Ajouter des interactions utilisateurs. Un système d'étiquette pour afficher les informations concernant la position de l'interaction réalisée. Cette fonctionnalité permettra de réguler le flux d'information.
- Prendre en considération les caractéristiques du périphérique (taille d'écran, résolution)

pour adapter la quantité d'information.
- Utiliser des outils existants permettant de faire du responsive design (media queries).
- Faire une gestion des évènements en utilisant une librairie spécifique comme Hammer.js.

J'ai donc cherché comment utiliser tous ces éléments pour ces deux visualisations. J'ai pu observer que plusieurs interactions n'étaient pas utilisables sur de petites tailles d'écran. Ces interactions irréalisables étaient principalement dues au problème du « gros doigt » mais aussi au type d'évènement par défaut qui n'était pas forcément compatible avec la fonctionnalité proposée.

Une fois avoir extrait les différents problèmes et identifié, implémenté des solutions, j'ai pu dans la seconde partie de mon travail définir différents objets génériques pouvant s'adapter à différents supports. Ces objets dits génériques impliquent leur indépendance à toutes visualisations. Ils ont pour but de simplifier la création d'une visualisation responsive via la librairie d3.js. Ces objets JavaScript sont les prémisses d'une librairie[5] responsive qui offre des solutions aux problèmes liés au responsive.

L'extension à la librairie d3.js réalisé met en avant la possibilité et les avantages de proposer des objets résolvant les problèmes du responsive indépendamment de la visualisation. Cette dernière n'étant qu'un début, beaucoup d'améliorations et d'évolutions peuvent être apportées. Je n'ai pas eu l'opportunité de la soumettre à plusieurs développeurs pour savoir si son utilisation est suffisamment simple ainsi que pour faire ressortir les principaux, je pense que dans un premier temps ces critères doivent être validés. Ensuite l'objet brush a été réalisé en utilisant un objet natif de d3.js. Cet objet utilise ses propres évènements adaptatifs selon le périphérique et n'ont donc pas été traités. Pour une meilleure flexibilité, il faudrait peut-être voir comment insérer l'objet event pour les évènements de l'objet brush. Enfin, ayant étudié uniquement deux visualisation alors qu'il en existe une multitude sur d3.js, il faudrait réaliser différentes études complémentaires afin de compléter la librairie.

8) Références

[BLC12] Baur, D., Lee, B., & Carpendale, S. (2012, November). TouchWave: kinetic multi-touch manipulation for hierarchical stacked graphs. In Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces (pp. 255-264). ACM.
[RK14] Rzeszotarski, J. M., & Kittur, A. (2014, April). Kinetica: Naturalistic multi-touch data visualization. In Proceedings of the 32nd annual ACM conference on Human factors in computing systems (pp. 897-906). ACM.
[BL07] Blanch, R., & Lecolinet, E. (2007). Browsing zoomable treemaps: Structure-aware multi-scale navigation techniques. Visualization and Computer Graphics, IEEE Transactions on, 13(6), 1248-1253.
[SH02] Sadowski, S., & Heidmann, F. A Visual Survey of Information Visualizations on Smartphones.
[CC02] Calvary, G., & Coutaz, J. (2002). Plasticité des Interfaces: une nécessité. information-interaction-intelligence, Actes des deuxièmes Assises nationales du GDR I, 3, 247-261.
[BOH11] Bostock, M., Ogievetsky, V., & Heer, J. (2011). D³ data-driven documents. Visualization and Computer Graphics, IEEE Transactions on, 17(12), 2301-2309.
[BLCMTG12] Boring, S., Ledo, D., Chen, X. A., Marquardt, N., Tang, A., & Greenberg, S. (2012, September). The fat thumb: using the thumb's contact size for single-handed mobile interaction. In Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services (pp. 39-48). ACM.

[5] Lien API Javascript Documentation : http://juliana23.github.io/responsiveVisualisations/doc/

9) Annexes

1. Visualisation Timeline

a. Fonctionnalités ajoutées

i. Axes

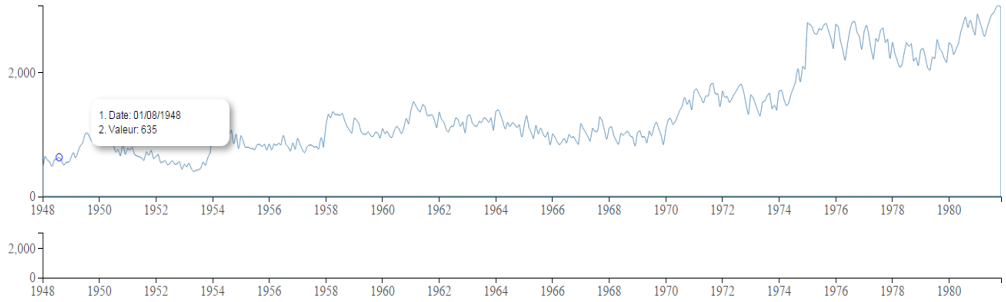


Figure 6: Timeline - Axes

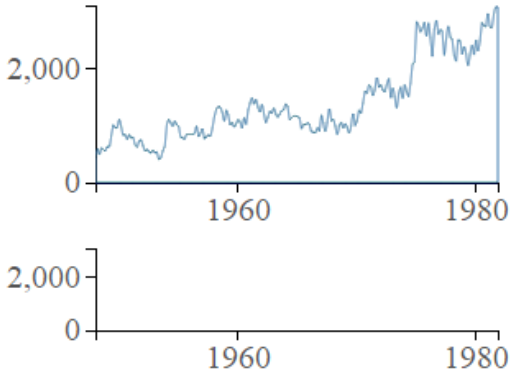


Figure 7: Timeline - Axes

ii. Etiquette

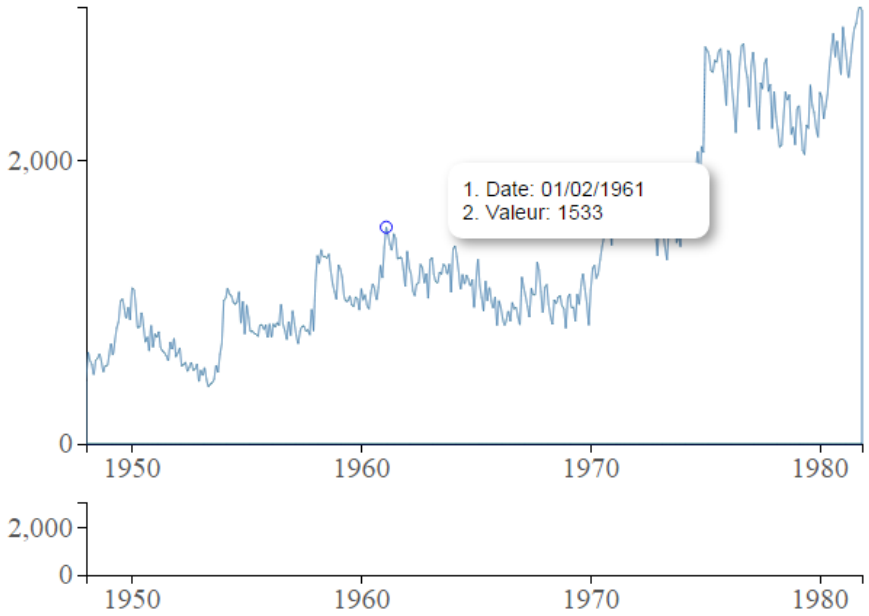


Figure 8: Timeline – Etiquette

iii. Zoom In / Out – Changement d'échelle

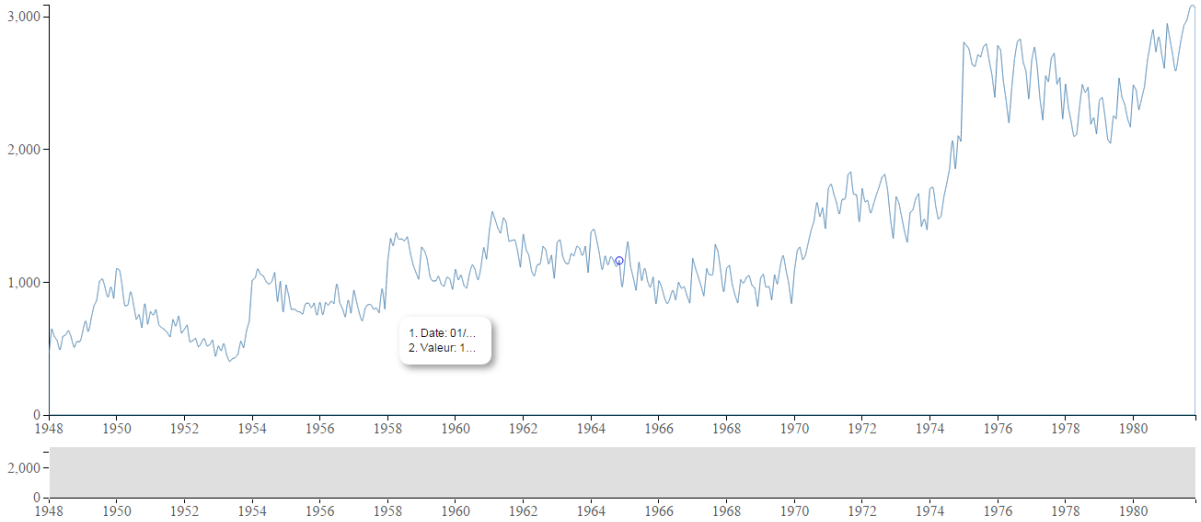


Figure 9: Timeline - Avant Zoom

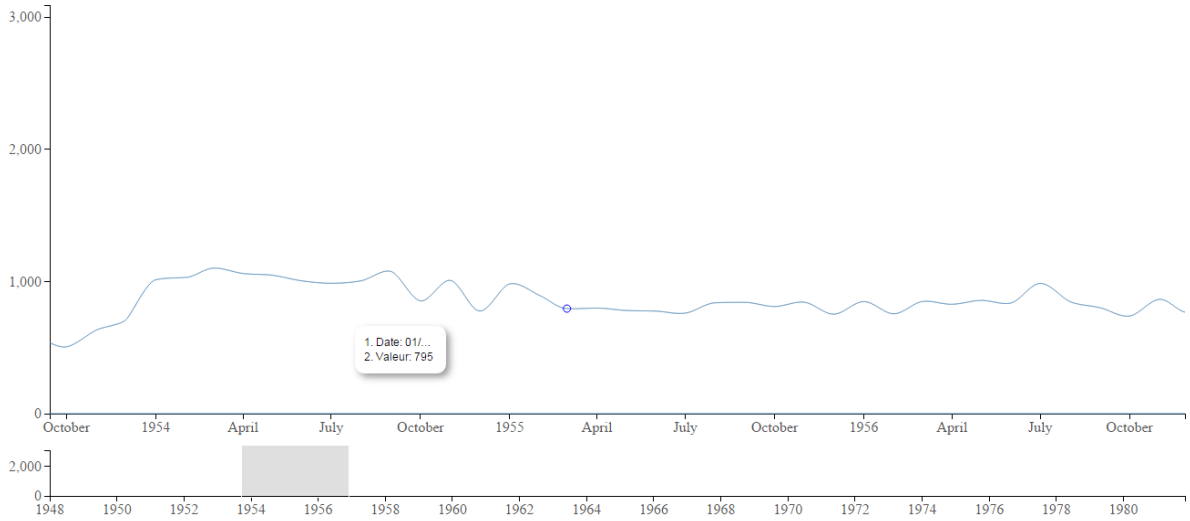


Figure 10: Timeline - Après Zoom

2. Visualisation Treemap

a. Fonctionnalités ajoutées

i. Affichage premiers parents

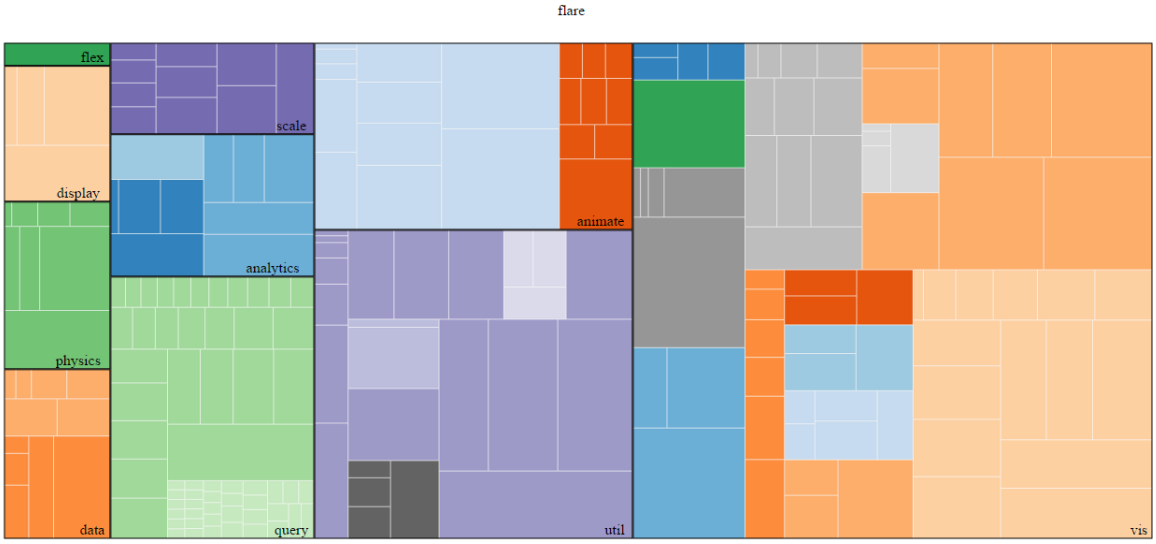


Figure 11: Treemap - Premiers parents

ii. Highlight

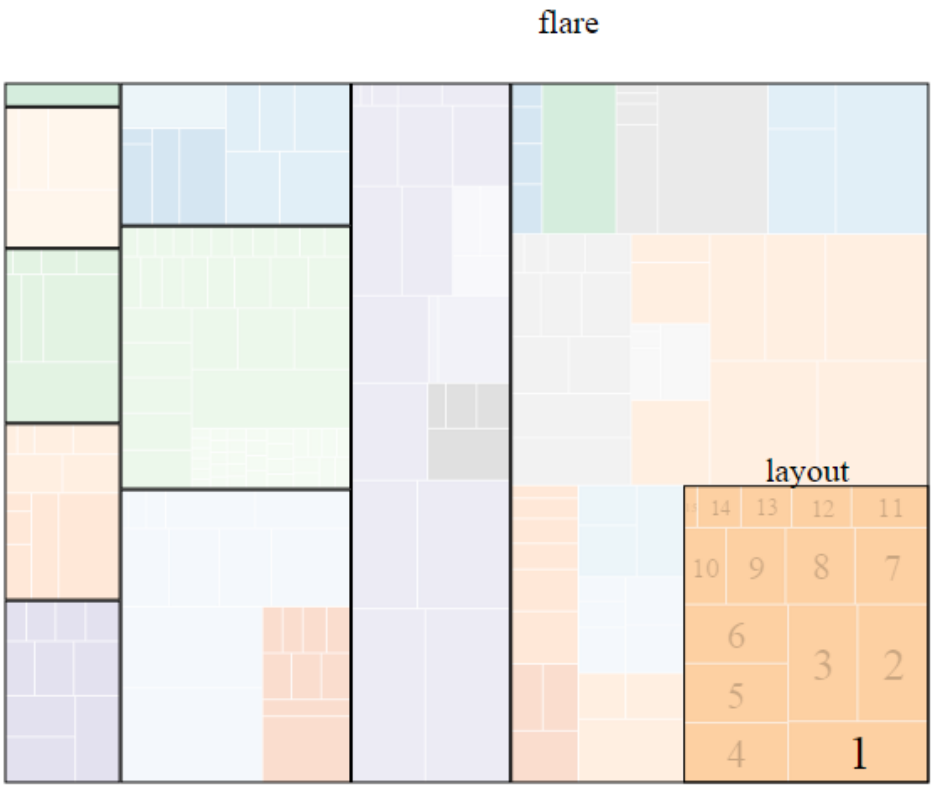


Figure 12: Treemap-HighLight

iii. Etiquette

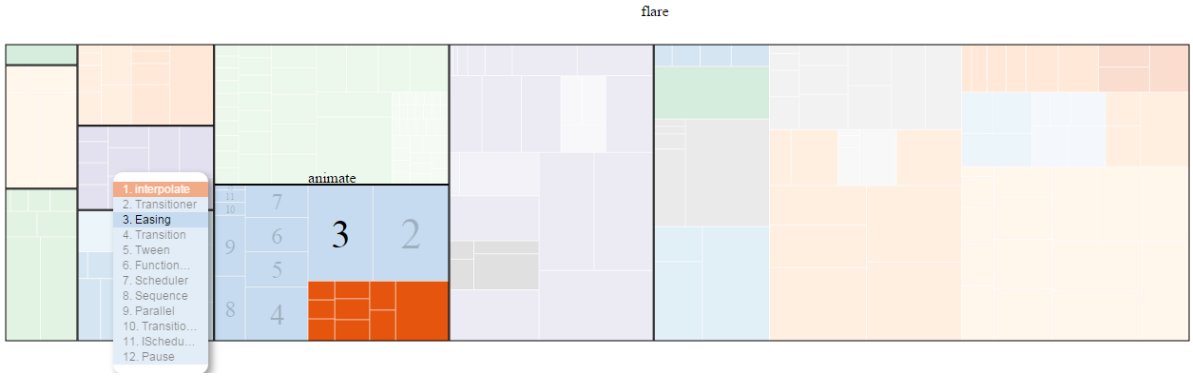


Figure 13: Treemap – Etiquette

iv. Zoom In / Out

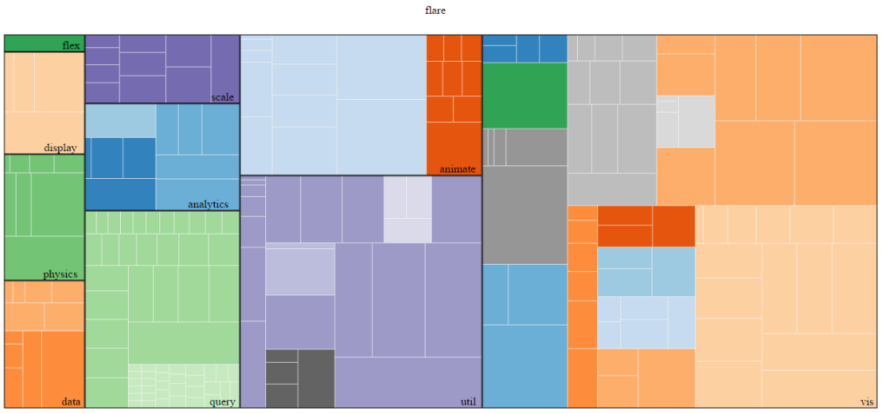


Figure 14: Treemap – Avant Zoom

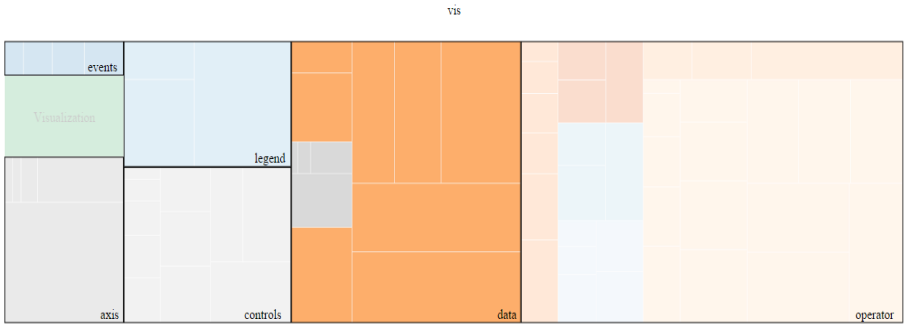


Figure 15: Treemap - Après Zoom et highlight sur le “noeud sélectionné”

Figure 16: Treemap – Après zoom et highlight sur le “noeud sélectionné”

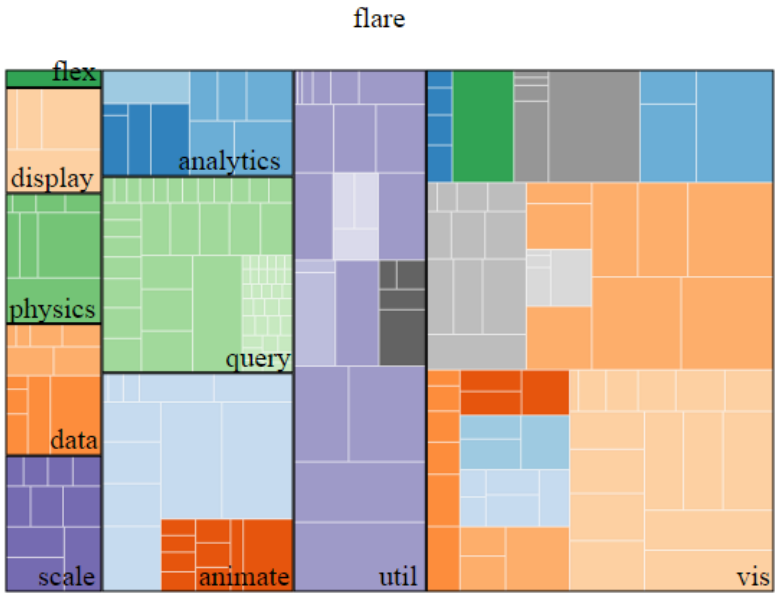


Figure 17: Treemap sur support de 600px / 590px

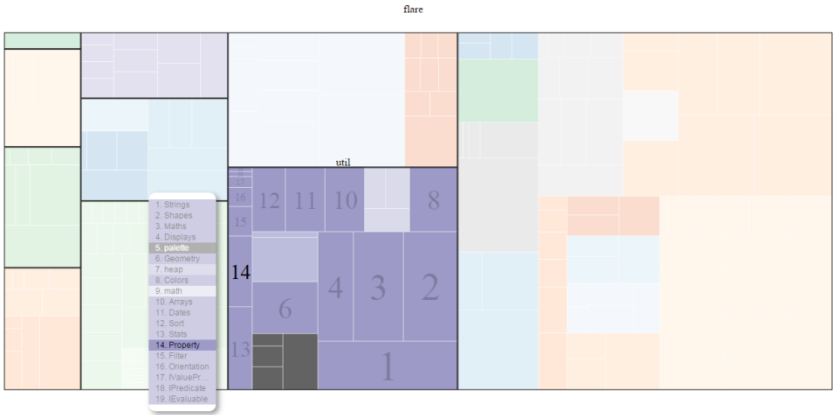


Figure 18: Treemap avec tooltip et highlight

3. Equivalences évènements souris / tactile

Evènement souris	Evènement tactile
Mousedown	Touchstart
Mouseup	Touchend
Mouseover	Touchenter
Mouseout	Touchleave
Click	Tap
Dblclick	DoubleTap
Mousecancel	Touchcancel

Tableau 3: Evènements souris / tactile

4. Media queries standards

Min-width	Max-width	Orientation	Device
X	320	X	Phone
321	768	X	Phone-landscape
1024	X	X	Desktop
1824	X	X	Large screen
768	1024	X	Tablet
321	768	landscape	Phone-landscape-strict
321	768	portrait	Phone-portrait-strict
768	1024	landscape	Tablet-landscape-strict
768	1024	portrait	Tablet-portrait-strict

Tableau 4: Media queries standard