Visualisation de données : interactivité et mise à l'échelle

1 Introduction

La visualisation des données permet, dans le meilleur des cas, de faire apparaître les informations cachées et la structure d'un jeu de données. Plus l'accès aux jeux de données volumineux se répand en informatique, en mathématiques et dans les entreprises, plus le besoin de logiciels offrant des possibilités d'interaction et de mise à l'échelle grandit.

Plotly est un logiciel de visualisation de données en ligne conçu pour rendre des graphiques dans des navigateurs. Nos utilisateurs proviennent d'horizons divers (monde de l'entreprise ou laboratoires scientifiques). Au fur et à mesure que la taille des jeux de données de nos clients augmente, la demande de rendus de graphiques de grande taille en deux dimensions augmente elle aussi.

Notre but principal est de trouver un mécanisme de stockage des données et d'exécution des commandes qui nous permettra d'améliorer nos rendus et éventuellement de rendre des jeux incluant jusqu'à un million d'éléments, tout en préservant les possibilités d'interaction et d'exportation que nos clients tiennent pour acquises.

2 La technologie utilisée par Plotly

À l'heure actuelle Plotly utilise deux méthodologies pour rendre les graphiques. Pour les graphiques en deux dimensions, elle se sert de D3.js, une bibliothèque pour graphiques en JavaScript utilisant SVG (*Scalable Vector Graphics*). Nos graphiques en trois dimensions sont rendus grâce à WebGL, un logiciel utilisant directement les unités de traitement graphique ou UTG (*Graphics Processing Units*, en anglais) pour rendre les graphiques très rapidement.

Notre logiciel, lorsqu'il se sert de SVG, peut rendre jusqu'à 100 000 points dans le navigateur. Toutefois les commandes interactives sont exécutées plus lentement à partir de 30 000 points. Une liste complète de points de référence se trouve à l'adresse `https://plot.ly/benchmarks/`.

SVG est un langage de balisage permettant de rendre des graphiques. Les rendus sont effectués grâce à une liste d'éléments SVG, qui sont essentiellement des polygones ou des splines auxquels sont associées des commandes de dessin. Étant donné que chaque élément SVG est dessiné, il faut beaucoup de temps pour rendre des fichiers SVG compliqués. Pour rendre un nuage de points (*scatter plot*) à l'aide de SVG, il faut faire de chaque point un élément SVG.

SVG présente plusieurs avantages pour Plotly. Tout d'abord, le logiciel de Plotly profite de la bibliothèque D3.js, qui est très avancée et offre beaucoup de

commandes interactives. Deuxièmement, SVG effectue des rendus sur n'importe quel système, est très stable, s'adapte à toutes les résolutions et permet de créer des applications rapidement.

Nous avons vu ci-dessus que **WebGL** est un logiciel utilisant directement les UTG pour rendre les graphiques très rapidement. Il est basé sur OpenGL mais est conçu pour le web. Lorsqu'un rendu d'image est demandé à WebGL, des fonctions envoient les données à l'UTG pour qu'elles soient traitées et les images sont dessinées sur un canevas. Les formes sont créées par des scripts contenant des nuanceurs (qui affectent des couleurs aux pixels inclus dans la forme).

WebGL est rapide, réalise des rendus de manière très efficace, offre de bonnes possibilités d'interaction et donne un très bon contrôle à l'utilisateur. Par contre le développement de fonctionnalités prend plus de temps avec WebGL qu'avec SVG, et la génération de figures d'archives ne se fait pas aussi bien en WebGL qu'en SVG parce que les rendus de WebGL dépendent de la résolution.

3 Objectifs du projet et considérations importantes

Dans la solution que nous cherchons, il faut tenir compte de la problématique du dessin (avec SVG ou WebGL) et réfléchir à la manière dont Plotly stocke les graphiques et à la manière dont le navigateur télécharge les données d'archives. Plus précisément, nous sommes à la recherche d'une manière efficace de stocker des données sur le serveur et de les télécharger dans une fenêtre de visualisation.

Notre premier objectif est de trouver des solutions pour des nuages de points en deux dimensions et des graphiques linéaires illustrant des jeux de données incluant environ un million de points.

À court terme, nous aimerions implanter cette solution à l'aide de SVG mais à moyen terme WebGL deviendra une composante importante de notre bibliothèque de graphiques en deux dimensions. Voici quelques caractéristiques de la solution que nous cherchons.

1. Les données seront stockées sur le serveur et téléchargées de manière efficace.
2. La solution incluera une façon de décider quels points seront envoyés par le serveur au client, de telle sorte que le graphique ressemble au graphique qui aurait été obtenu si tous les points avaient été envoyés.
3. Le client pourra demander au serveur des zooms avant et arrière et le serveur mettra à jour les données.

Il est essentiel que la solution proposée conserve les caractéristiques actuelles, par exemple la possibilité de mettre en lumière un point donné et des caractéristiques telles que les glyphes, les couleurs, les frontières, etc. Le prétraitement

doit prendre un temps inférieur ou égal à un multiple de $n \cdot polylog(n)$, où n dénote le nombre de points du graphique. L'espace occupé par les points du graphique en mémoire doit être proportionnel à n. Les algorithmes doivent être suffisamment simples pour pouvoir être implantés en un temps raisonnable.

4 Quelques problèmes spécifiques

La mise au point de la solution requiert probablement la résolution de problèmes plus simples, que nous décrivons dans cette section. Commençons par définir ce qu'est un **nuage de points**. Étant donné n vecteurs en deux dimensions et une forme S (appelée le marqueur), un nuage de points est l'image consistant de la réunion de n copies de S, où chacune est obtenue par translation d'un des vecteurs. Voici une liste de problèmes.

1. Étant donné un nuage de points avec marqueur circulaire et devant être rendu en SVG, trouver la courbe représentant la frontière du nuage de points et consistant d'un nombre minimal d'arcs de cercle. Voici quelques variantes de ce problème.
   (a) Supposez que les marqueurs sont des cercles de rayon constant.
   (b) Permettez aux rayons des marqueurs d'être différents.
   (c) Permettez aux marqueurs de prendre des formes différentes.
   (d) Permettez aux couleurs des marqueurs d'être différentes.
   (e) Permettez aux frontières des marqueurs d'être de couleurs différentes.
   (f) Supposez que les marqueurs sont partiellement transparents. Est-il possible de calculer un ensemble minimal d'arcs qui « rendront » l'opacité de manière correcte ?
2. Pour les graphiques devant être rendus en WebGL, nous voulons étudier les questions suivantes.
   (a) Comment devons-nous soutenir le chargement avec priorités ?
   (b) Quelle est la meilleure façon d'implanter l'interactivité et les sélections ?
   (c) Comment devons-nous traiter les formes transparentes ?

Donnons maintenant une définition formelle d'un **graphique linéaire**. Un graphique linéaire est une courbe linéaire par morceaux. En ce moment nous utilisons un algorithme de décimation pour les graphiques linéaires ; il a une meilleure performance que celle que nous avons pour les nuages de points.

1. Pour les graphiques devant être rendus en SVG, nous voulons tenter de réduire le nombre d'éléments à rendre tout en préservant l'aspect du graphique.
   (a) Comment peut-on « filtrer » une courbe en enlevant certains points qui sont proches d'autres points ?

(b) Supposons qu'un graphique linéaire a des régions denses. Comment pouvons-nous remplacer des segments de droite par des polygones ?

(c) Lorsque nous utilisons des algorithmes de décimation, comment pouvons-nous soutenir des caractéristiques telles que les couleurs ou les largeurs de lignes ?

2. Pour les graphiques devant être rendus en WebGL, voici deux problèmes qui se posent.

(a) Comment pouvons-nous implanter les graphiques en continu ?

(b) Comment devons-nous soutenir le chargement avec priorités ?