Visualisation de données spatio-temporelles: Étude sismologique du glacier d'Argentière

Spatio-Temporal Data Visualisation: Seismological Study of the Argentière Glacier

Renaud Blanch, Albanne Lecointre, Michael Ortega et Philippe Roux

1 Introduction

Depuis les prémisses de leur domaine, les chercheurs en Sciences de la Terre recueillent de grandes quantités de données pour explorer, modéliser et comprendre les phénomènes géophysiques. Aujourd'hui, l'outil informatique leur permet de traiter de plus grandes quantités de données, et plus rapidement. Cependant, leur panel d'outils est limité. Ils utilisent la plupart du temps un pipeline non-interactif de traitements successifs, qui aboutissent à des tableaux de valeurs numériques ou à une interprétation graphique figée. La mise en évidence d'un phénomène particulier passe ainsi par plusieurs essais-erreurs sur les paramètres du pipeline, méthode longue et fastidieuse.

Dans ce papier, nous présentons un exemple d'application plus efficace d'exploration des données, qui s'appuie sur les bénéfices de l'interactivité. Le développement de cette application a été mené en deux étapes distinctes, qui ont permis de la rendre accessible aux chercheurs, avec des contraintes matérielles minimales. L'application est aujourd'hui utilisée par plusieurs chercheurs des Sciences de la Terre.

2 Contexte

Contexte scientifique

En 2017, le projet RESOLVE[1], mené en particulier au laboratoire ISTerre[2], a positionné une centaine de capteurs sismographiques à la surface du glacier d'Argentière, dans les Alpes françaises, pour en étudier la dynamique. Ont ainsi été recueillis en continu une centaine de signaux sur une période de 33 jours. Le traitement de ces signaux permet de remonter, pour chaque seconde de cette période, à la localisation dans l'espace des évènements qui ont pu les générer, ainsi qu'à d'autres attributs permettant de caractériser ces évènements du point de vue des phénomènes physiques en jeu [3].

- *Renaud Blanch et Michael Ortega: Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France.*
- *Albanne Lecointre et Philippe Roux: Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, IRD, Univ. Gustave Eiffel, ISTerre, F-38000 Grenoble, France.*
  E-mails: prenom.nom@univ-grenoble-alpes.fr

1. <https://resolve.osug.fr/>
2. <https://www.isterre.fr/>

Nature des données

Les données générées par ces traitement sont très volumineuses et leur exploration est fastidieuse avec les outils dont disposent les chercheurs en Science de la Terre. Le traitement, dans sa version la plus simple, découpe en effet les 33 jours en tranches d'une seconde, soit un peu moins de 3 millions de dates, et effectue dans ces fenêtres temporelles des transformées dans une douzaine de gammes de fréquences. En sortie, des points 5D sont produits (X, Y et Z, position dans l'espace ; c, vitesse de propagation, et q un indicateur statistique de confiance). Ce jeu de données, utilisé pour développer les prototypes initiaux, pèse environ 500 MB, et les versions plus élaborées du traitement produisent des jeux de données pesant jusqu'à 50 GB.

3 Prototype client lourd

Suite à une mise en relation entre chercheurs en Science de la Terre et en Informatique, un premier prototype a été réalisé pour permettre d'explorer interactivement les données.

Ce prototype (voir Figure 1) est basé sur Python et ses bibliothèques scipy (pour accéder aux données) et numpy (pour ses tableaux efficaces) ; et sur les bibliothèques GLUT (interface entre le système de fenêtrage et l'accélération graphique) et surtout OpenGL et son pipeline programmable pour bénéficier de l'accélération matérielle du rendu graphique. Cette combinaison d'un language de haut niveau permettant d'abstraire le système hôte avec des bibliothèques permettant une utilisation des services offerts à bas niveau par ce même système est très efficace pour créer des prototypes et les partager avec des utilisateurs ayant des configurations hétérogènes.

Les données du jeu de prototypage n'étant pas trop volumineuses, elles sont chargées en mémoire vidéo telles quelles et le rendu (filtrage, gestion de la sélection, projection 3D, mise en couleur) est assuré par des programmes s'éxecutant sur la carte graphique.

La vue principale présente donc les points dans une vue 3D dont l'orientation est manipulable à l'aide d'un *arcball*, et le déplacement par un *drag*. Afin de mieux percevoir la 3D sans dispositif particulier, une animation fait varier le point de vue cycliquement de gauche à droite de quelques degrés, exploitant la sensibilité de notre système perceptif à la parallaxe de mouvement pour induire une perception 3D [6]. Une interaction de *brushing* permet la sélection d'un sous-ensemble de point, et la vue peut être recentrée sur ce sous-ensemble.
À côté de la vue 3D (à gauche sur la Figure 1) des vues affichent une représentation calendaire des points, un météograme et un *range slider* qui partagent tous trois la même échelle temporelle horizontale et permettent de focaliser la sélection sur un sous-ensemble de points. Deux histogrammes permettent enfin d'afficher les distributions et d'effectuer des sélections dans les domaines des variables c et q.

4 Application web

Malgré la portabilité du langage Python et de la librairie OpenGL, le premier prototype présenté nécessite une grande quantité de mémoire vive, et un processeur central et une carte graphique performants, afin d'assurer l'utilisabilité et la fluidité de l'interaction. De plus, la taille de certains jeux de données impose que ces dernières restent à proximité (sur le même ordinateur) de l'application. Cela est indispensable pour limiter les temps de transfert de données à l'application (mémoire vive, mémoire graphique).

Pour soustraire nos utilisateurs à ces contraintes matérielles, nous avons envisagé une version web de l'application. Le prototype a ainsi été découpé en deux parties distinctes : une dédiée au rendu, possédant l'accès direct aux données et les fonctionnalité d'affichage, que nous appellerons « serveur » ; et une dédié à l'interface de visualisation et de manipulation des données, que nous appellerons « client », communicant avec le serveur dès qu'une action utilisateur est réalisée.

4.0.1 Client-Serveur

Le client est une simple page web, qui se résume en une balise HTML <canvas>, prenant 100% de la fenêtre d'affichage du navigateur. Des widgets d'interaction (boutons, sliders, infos textes, etc.) survolent la page comme on peut le voir sur la Figure 2.

Dans la partie serveur, nous avons utilisé ModernGL [2], un wrapper d'OpenGL (en python) permettant de faire du rendu graphique *headless*, c'est à dire sans environnement d'affichage. Cette partie a été développée et installée sur une machine dédiée installée dans un *data center*, donc sans système d'affichage. Elle possède toutes les caractéristiques matérielles nécessaires à l'application.

La communication entre le client et le serveur est basée sur le protocole de communication WebSockets [5], canal de communication bi-directionnel. A chaque interaction, comme par exemple le survol de la souris sur le canvas du client, une websocket est ouverte pour envoyer l'information au Serveur et recevoir sa réponse le cas échéant.

4.0.2 Optimisations

Les informations envoyées sont très légères, au format JSON, e.g. une simple valeur avec le mot clé qui la désigne. La plupart du temps, la réponse est un nouveau rendu OpenGL, i.e., une image qui va remplir le canvas. Afin d'optimiser le transfert de ces images/rendus, elles sont compressées au format jpeg sur le serveur, en conservant un niveau de qualité élevé, juste après chaque rendu.

La fréquence de transfert n'est pas constante : un nouveau rendu n'est envoyé que lorsque cela est nécessaire, i.e., lorsqu'il est différent du précédent. Cependant, la fréquence de transfert peut devenir élevée dans certains cas, par exemple lorsque l'utilisateur manipule le point de vue de la scène. Ce cas est détecté et afin de conserver une fréquence suffisante, et préserver la fluidité de l'interaction, le poids des images transférées est réduit en baissant la qualité de compression JPEG. Cette perte de qualité est compensée par le mouvement du point de vue, et ne gêne pas l'interaction. Une fois l'interaction terminée, une image de qualité maximale est renvoyée pour permettre à l'utilisateur d'explorer efficacement son nouveau point de vue.

Concernant les données, lors du chargement d'un nouveau jeu nous stockons à la volée sur le serveur les objets calculés les plus lourds (e.g. données transformées/filtrées, résultats de clusterings, tableaux pour shaders graphiques, etc.) dans des fichiers, grâce au module pickle [1]. Cela permet le lancement rapide de l'application sur un jeu de données connu, i.e., 99% du temps.

5 Conclusion

L'application est actuellement hébergée sur un serveur du Laboratoire d'Informatique de Grenoble. Elle est utilisée par plusieurs chercheurs, pour l'instant uniquement sur les données du glacier d'Argentière. La collaboration avec l'institut des Sciences de la Terre nous a permis de concevoir une interface efficace et adaptée, ainsi que d'ajouter certaines fonctionnalités telles que le *clustering* d'évènements. Pour cette dernière, nous avons opté pour l'algorithme DBSCAN, et plus particulièrement sa version parallélisée [4]. Elle permet de regrouper les évènements sismiques proches dans l'espace et dans le temps. Grâce à cette classification, représentée par les différentes couleurs dans la Figure 2, les chercheurs d'ISTerre ont, en autre, vu apparaître des phénomènes liés aux crevasses du glacier.

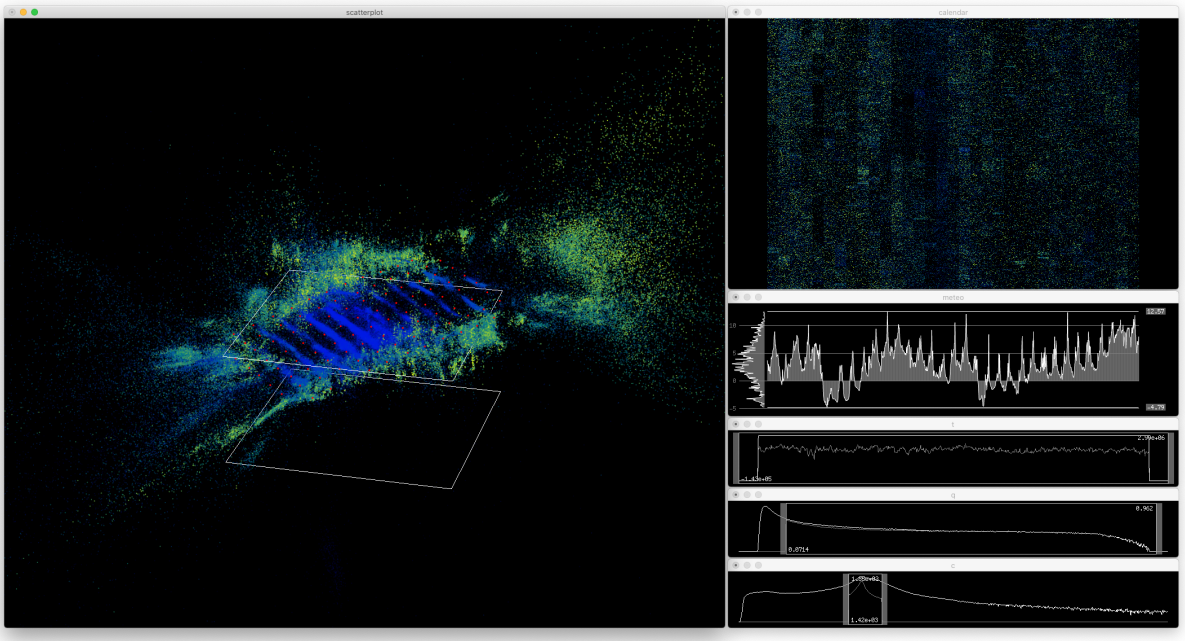

FIGURE 1. Prototype client lourd – À gauche : la vue 3D ; à droite de haut en bas : la vue calendrier, la frise des températures, la sélection temporelle, les sélections par q et c.

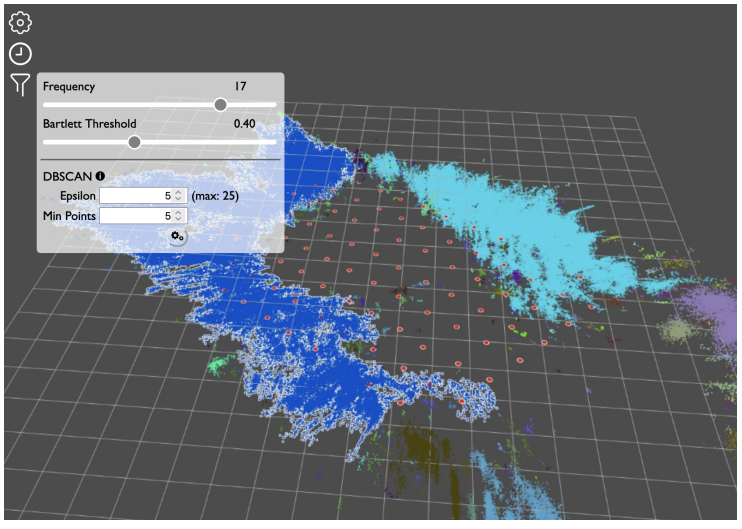

FIGURE 2. Prototype web – Les couleurs représentent différents clusters calculés par DBSCAN. Ici, le menu transparent en survol permet de filtrer les données et de paramétrer l'algorithme de *clustering*.

RÉFÉRENCES

[1] pickle — python object serialization. https://docs.python.org/3/library/pickle.html.

[2] S. Dombi. Moderngl, high performance python bindings for opengl 3.3+. https://github.com/moderngl/moderngl.

[3] F. Gimbert, U. Nanni, P. Roux, A. Helmstetter, S. Garambois, A. Lecointre, A. Walpersdorf, B. Jourdain, M. Langlais, O. Laarman, F. Lindner, A. Sergeant, C. Vincent, and F. Walter. The RESOLVE project : a multi-physics experiment with a temporary dense seismic array on the Argentière glacier, French Alps. *Seismological Research Letters*, 92(2A) :1185–1201, 2021.

[4] M. Götz, C. Bodenstein, and M. Riedel. Hpdbscan : highly parallel dbscan. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, page 2. ACM, 2015.

[5] A. Melnikov and I. Fette. The WebSocket Protocol. RFC 6455, Dec. 2011.

[6] M. Ortega and W. Stuerzlinger. Pointing at 3D wiggle displays. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 335–340, Reutlingen, 2018.