Immersive and Collaborative Data Visualization Using Virtual Reality Platforms

2014 IEEE International Conference on Big Data
http://arxiv.org/pdf/1410.7670

Cet article a été soumis très récemment et propose d'utiliser les derniers outils de réalité virtuelle développés pour l'industrie du jeux vidéo dans les mains des chercheurs et analystes, leur permettant ainsi d'interprêter de grandes quantitées de données complexes (Big Data) de façon plus intuitives pour un moindre coût. Une dimension collaborative est également abordée pour permettre le travail sur un même jeu de données simultanément depuis n'importe quel lieux grâce à l'Internet.

La problématique est réelle, nos systèmes d'information véhiculent toujours plus d'informations qui sont contrôlées et analysées dans le but faire évoluer la science et la technologie. c'est cette analyse qui, sur un jeu de données complexe, demande à un humain des capacités cognitives extrêmes. Lorsque trop de dimensions son associées à un même concept, il est très dur de le conceptualiser car toutes les dimensions ne peuvent être représentées dans un seul graphique planaire. Si bien que plusieurs graphiques sont alors nécessaires, ce qui ajoute une gymnastique mentale pour aggréger ces représentations qui formeront un modèle mental unifié. C'est justement cette étape que l'équipe cherche à annuler en proposant une représentation dynamique tirant parti au maximum des macro dimensions que sont l'espace tri-dimensionnel, les couleurs et le temps.

Ainsi ils ont choisi comme support de rendu visuel le moteur 3D Unity. Outre sa license communautaire qui permet à tout un chacun de développer sans payer de redevance pour toute application non-commerciale, ce moteur est dans la tête de course avec ses concurrents comme l'Unreal Engine. Il dispose nottamment de couches applicatives modulaires lui permettant de tourner aussi bien nativement sur mobile et bureau, tous système d'exploitation confondus, que dans les navigateurs web via un plug-in (Firefox et Chrome). L'application se trouve donc être multi-plateforme de par son choix technologique.

Les données, une fois extraites et formatées en amont, sont alors représentées dans un espace 3D sous la forme d'objets. En faisant varier leur positionnement (x,y,z), forme, couleurs, taille, opacité, texture, clignotement (mouvement), on atteint déjà 9 dimensions représentables dans une seule visualisation. C'est déjà beaucoup plus riche qu'un diagramme en batton ou qu'un repère orthonormé et ses courbes aussi complexes soient-elles.

La visualisation de ces dernières est elle aussi particulière, un masque, l'Occulus Rift, affichant une image différente dans chaque oeil selon le principe de la stéréoscopie, couplé avec un capteur d'orientation pour la direction du regard et une caméra Kinect pour la position du corps, vise à immerger l'utilisateur à l'intégrieur de cette représentation. Le but est de maximiser l'analyse intuitive associée à nos sens pour augmenter encore plus la compréhension des données.

L'interaction avec l'environnement virtuel, outre par le déplacement, est aussi possible, dans une position statique cette fois, avec le Leap Motion, dispositif permettant de suivre la position des doigts d'une main dans l'espace.

Enfin l'aspect collaboratif est concrétisé par la présence de l'avatar du/des collaborateur(s) modélisé(s) en 3D au sein même de l'espace de rendu des données et est mis à jour en fonction des déplacements de ce dernier.

Le dispositif global n'est pas voué à être utilisé dans n'importe quel environnement, en effet les périphériques support à la réalité virtuelle ne sont rattachable qu'à un ordinateur fixe et de par leur nature même, coupent l'utilisateur du monde réel. Par conséquent un lieux calme et sécurisé est de mise. On pourrait néanmois envisager une version mobile avec l'orientation du téléphone et son écran pour remplacer l'Occulus Rift, l'écran tactile à la place du Leap Motion, puisque Unity supporte les mobiles et que ceux-ci on accès à internet.

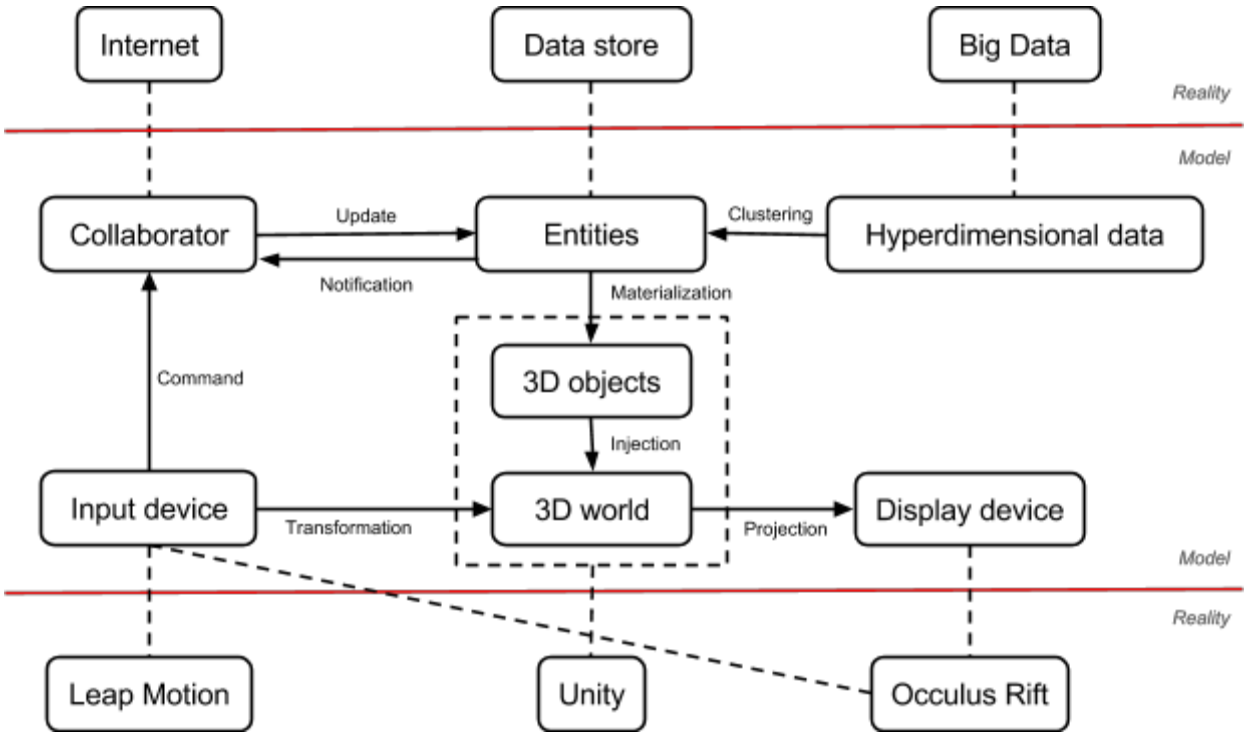


Fig. 1 : Relation entre un modèle possible et les réalités technologiques

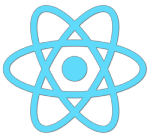
Lorsque l'on s'affranchi du sentiment d'avoir fait un pas dans le futur, on peut se demander quels sont les concepts qui étayent cette approche. A travers un possible modèle (Fig. 1), on distingue un agencement modulaire. Le fait que l'affichage soit stéréoscopique n'empêche pas son rendu sur un simple écran car l'image provient de la **projection** d'un environnement 3D géré par Unity. Qu'importe les entrées également, du moment que l'on peut apporter les **transformations** nécessaires à la visualisation des données et se déplacer dans l'environnement. Grace aux entrées utilisateur on va donc pouvoir **commander** notre avatar virtuel, ceux des collaborateurs le seront via le réseau (Internet). C'est ensuite chaque collaborateur qui va **mettre à jour** l'environnement en changeant des attributs sur les entitées qui représentent une **sous-partie** d'un ensemble hyper-dimensionnel de données issues du Big Data. ces même entitées sont **matérialisées** dans Unity par des objets 3D qui sont ensuite **injectés** dans un graphe de scène où ils se verront appliqué des contraintes physiques telles que l'éclairage ou les collisions.

Cet accomplissement technologique s'annonce donc prometteur car il laisse entrevoir énormément de champs d'application. Un exemple en est fait avec l'analyse de clichés martien. En mappant des éléments de la géologie à des coordonnées arbitraires, la réalité virtuelle permet de détecter les erreurs d'approximation de distance immédiatement grace à la capacité du cerveau à estimer les distances. Ainsi on peut corriger ce mappage instantanément de façon très précise. un autre exemple est celui permettant d'explorer l'univers pour en visualiser les différents aspects physiques. Voir les relations dans l'espace entre les étoiles, leurs différentes émissions de lumière, mouvements, masses…

En revanche le prototype actuel propose une UI basique et construite avec les éléments graphiques fournis par Unity. C'est une interface dite imédiate (IMGUI), par conséquent elle est basique et requiert des talents de programmeur pour la concevoir et la faire évoluer (design et fonctionnalités liées). Une approche avec un environnement graphique extèrne intégrant une vue sur la scène 3D serait à préférer. Et enfin le nombre d'accessoires nécessaires à l'immersion virtuelle rend son application limité à des salles spécialement équipées puisqu'il est peu probable qu'un chercheur, astronaute ou analyse ait chez lui tout cet attirail de joueur.

Dans le cadre de la plasticité, on se place ici au niveau utilisateur, FUI, de la référence CAMELEON. En effet les modalités d'interaction sont donc fixées à la conception, le code source étant néanmoins indépendant de la plate-forme.

Web  Components

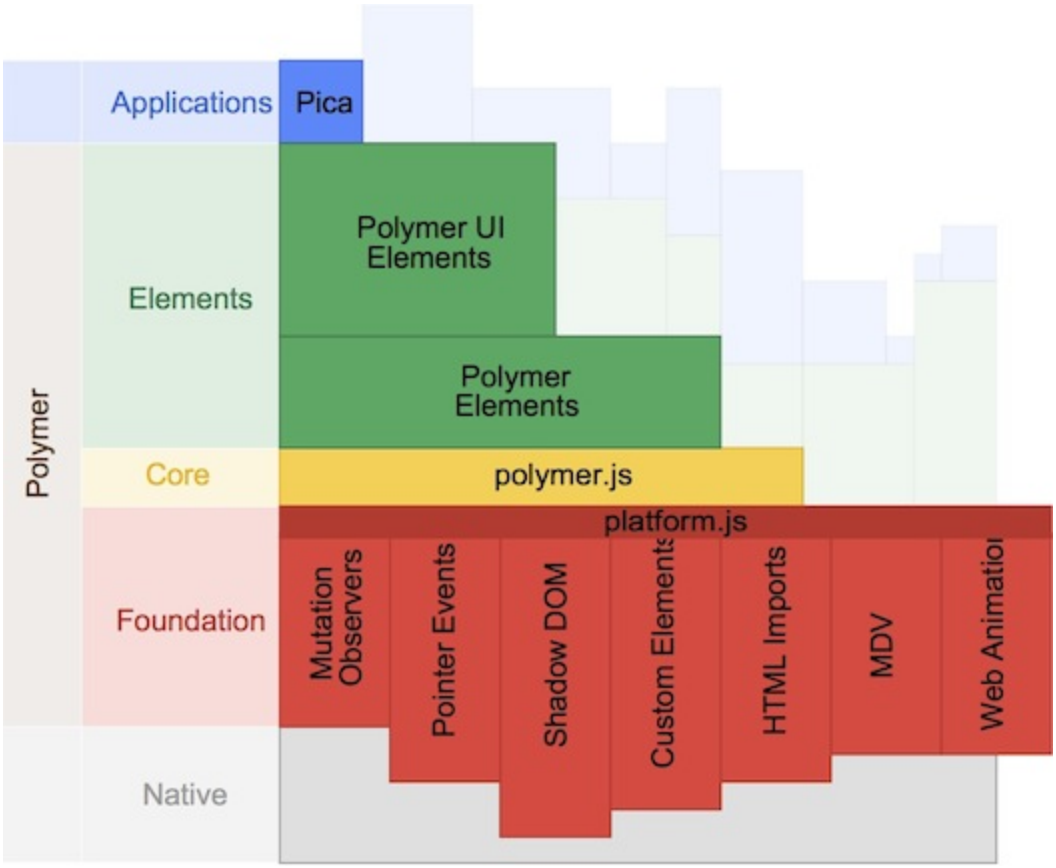
Polymer  VS  React

---

Révolutionner la manière de concevoir une interface graphique web est l'objectif de la spécification Web Components en cours de standardisation au sein du W3C. Elle introduit la notion de tags HTML réutilisables.
Templates, décorateurs (faisant appel à des sélecteurs CSS), éléments personnalisés, Shadow DOM... "Les Web Components permettent de combiner plusieurs éléments pour créer des widget réutilisables avec un niveau de richesse et d'interactivité sans commune mesure avec ce qui est aujourd'hui possible en se limitant aux CSS", explique W3C.
Même si le processus de standardisation des Web Components est loin d'être abouti au sein du consortium, les éditeurs de navigateurs commencent à s'engager en faveur de cette nouvelle technologie plus que prometteuse. Google et Mozilla ont notamment levé le voile chacun sur une infrastructure visant à faciliter le développement de ce nouveau type de composant.
Les deux acteurs entendent bien parallèlement assurer la prise en charge de cette nouvelle spécification par leur navigateur respectif. L'idée est d'œuvrer à un socle commun : une API de bas niveau (baptisée polyfills) pour faire en sorte que les Web Components s'exécutent de manière équivalente sur Firefox et Chrome.

Polyfill, Polymer et Paper Elements, l'écosystème de Google

Les Web Components sont un ensemble d'APIs (shadow DOM, templates, custom elements, etc.) qui vont permettre la réalisation de widgets et d'éléments graphiques pour le web. l'objectif principal du projet Polymer est d'aider dans l'utilisation de ces APIs à plusieurs niveaux. D'abord il est mis à disposition leurs propre version des polyfills, appelée Platform, ensuite un ensemble de Web Components assurant des fonctions variées et souvent nécessaires sont proposés, c'est la bibliothèque Polymer en elle-même. Enfin un dernier lot de components est proposé, uniquement orienté interface utilisateurs, ce sont les Paper Elements qui véhiculent les nouveaux choix en matière de design de Google. Tout dernièrement un éditeur graphique, du nom de code Pica, a été proposé pour rapidement prototyper un component, jusqu'à une page complète puisque Google propose des layouts sous la forme de Components.



Architecture de Polymer

Nous avons donc trois modules clairement séparés :

- platform.js: un ensemble de *shims*, codes permettant de proposer les concepts décris dans la norme sur tous les navigateurs en émulant, via javascript uniquement, les

fonctionnalités natives manquantes. Dans le futur cette couche sera ammenée à disparaître.
- polymer.js: C'est une couche d'abstraction arbitraire définissant une certaine façon d'utilisation des fondations.
- Enfin les éléments en vert sont la partie émergée de Polymer, ces web components recquièrent polymer.js. Ils n'affichent pas tous à un rendu sur la page. Par exemple, l'élément polymer-ajax, desitné à réaliser des appels AJAX. Le but des Paper elements (Polymer UI) est de répandre les notions de design propulsé dans les nouvelles versions des applications Google à toute la communauté.

Voici la philosophie décrite par Google :
- "Embrace HTML", Tout est composant, même ce qui ne touche pas à l'UI.
- "Leverage the evolving web platform", du code de meilleure qualité au cours du temps.
- "Minimize boilerplate code":
    - Promouvoir la création de webapps natives, sans bibliothèque supplémentaire.
    - Utiliser directement les standards du web et ainsi y contribuer.
- "Salt to taste", Polymer est conçu pour une utilisation à la carte. Chaque environnement qui l'intègre choisi ce qu'il veut utiliser et uniquement. c'est un écosystème agnostique d'où cette architecture en couches.

React, l'alternative de Facebook

A peut près au même moment où Google annonçait Polymer, Facebook a introduit react, qui se place comme alternative et concurrent féroce aux Web Components puisqu'il vise à résoudre les mêmes problématiques mais par une approche opposée. Avec React on cherche aussi à créer des components mais en s'abstrayant totalement de l'environnement. Le parrallèle avec les Web Components s'arrête donc là puisque l'implémentation est uniquement réalisée en javascript, aucun pré-requis sur une API native au navigateur n'est fait. Ce qui enlève déjà la nécessité de polyfills.

Deux principes fondamentaux guident les choix de React :
- Le DOM est un handicap en terme de performances, il faut y toucher le moins possible
- Réutiliser les principes de programmation réactive pour rendre les erreurs prédictibles

Pour s'abstraire du DOM les gars de Facebook ont tout bonnement recréé ce dernier en interne en pur javascript, c'est le *Virtual DOM*. L'avantage premier est qu'il contient uniquement les informations **nécessaires** à l'utilisation qui en est faite. Le DOM actuellement défini par le W3C impose pour n'importe quel élément un nombre astronomique d'attributs et fonctionnalités, c'est ce qui rend principalement le DOM natif plus lourd à manipuler, bien qu'écrit en C++, par rapport au Virtual DOM. Mais ça a aussi une autre nécessité qui va de pair avec la notion de *reactive programming.* Lorsqu'un changement intervient dans l'état d'un component, ce dernier est alors re-rendu et c'est tout le *virtual DOM* qui est mis à jour. cMais

'est carrément une nouvelle image de cet arbre qui est créée. Elle est ensuite comparée à l'ancienne version et, via une opération de *diff* que l'on doit aux gestionnaires de version et au jeu Doom dont s'est officiellement inspiré Pete Hunt l'un des initiateurs du projet, un nombre minimales de modifications du DOM sont alors inférées. React sonne le glas de JQuery puisqu'il abstrait totalement la manipulation du DOM, c'est lui qui sait le mieux comment le manipuler, nous nous contentons juste de modifier l'état de nos components pour voir ceux-ci être mis à jour automatiquement et d'une façon optimale.

Un exemple pour bien comprendre l'efficacité de cette approche. imaginons que nous avons une liste d'éléments rendus dans un élément html <li>, si nous avons l'élément "banana" puis à l'état n+1 la liste contient à la place "cuillère" et "champignon", avec une approche "à la JQuery", nous suprimerons d'abord l'élément présent pour insérer les deux nouveaux éléments. React est plus fort, il va constater qu'il existe déjà un élément qu'il va **altérer** pour qu'il affiche "cuillère" et seulement insérer un nouvel élément pour afficher "champignon". On a gagné deux opérations coûteuses pour insérer et retirer du DOM un élément.

Techniquement React se présente comme une bibliothèque javascript donc. La déclaration des templates et des styles est elle aussi réalisée en javascript. Mais une alternative pour incorporer des templates HTML est proposée. En nommant l'extension de la source ".jsx" et via un interprêteur statique ou embarqué à l'execution, ces sources sont ensuite traduites en javascript pur. Une approche orienté flot de données et aussi adoptée, ce qui enlève toute notion de *data binding* (bien que recréés en interne pour la liaison avec le vrai DOM), c'est une mode actuelle qui a l'avantage de mieux détecter les sources d'erreurs et qui combinée à des états immuables rend une component React robuste et concis.

Comparaison des deux approches

DSLs

Avec Polymer et les Web Components les styles sont encapsulés nativement dans le component, c'est le *shadow DOM*. React, lui, fournis ses propres mécanismes de style, en pur javascript. On peut alors trouver que cette approche n'est pas belle car elle casse la séparation des DSLs (Domain Specific Languages). Mais javascript est un langage riche et très souple. Définir un style en javascript nous donne beaucoup plus d'expressivité et de puissance en ayant accès directement au contexte de l'application qui traditionnellement est séparé de celui du style et n'autorisant que la modification depuis javascript du CSS. De plus plusieurs DSL signifie des règles différentes et une incertitude sur les données manipulées. Si tout est javascript, on a alors un contrôle énorme sur la façon de décorer son interface, on a des classes, des types, des valeurs par défaut, de la récursion, des closures… bref, passé la question de goût, l'approche tout javascript, ou mixte avec JSX, semble apporter un gain d'expressivité.

Des projets annexes se sont néanmoins développés pour permettre de continuer à développer en CSS et HTML séparément les components, mais ils perdent de la vitesse au fur et à mesure que la communauté prend ces nouvelles habitudes.

Natif vs VM

A une époque pas si lointaine, où l'assembleur venait d'apparaître, beaucoup d'amoureux de leurs machines préféraient continuer à coder en binaire pour réaliser leurs propres optimisations. De même avec javascript beaucoup ne font pas confiance à ce langage interprêté. Or les dernières versions des compilateurs javascript déploient des stratégies d'optimisation au runtime extrêmement poussées qui seraient impossible à automatiser dans un langage compilé. React choisi le tout javascript à l'encontre de Polymer qui se base sur des fonctionnalités natives des navigateurs.

il n'y a pas de place à l'opinion entre un DOM ultra légé en javascript avec 4-5 attributs par noeuds et un DOM natif qui contient des références cycliques et qui doit implémenter 4 niveau de fonctionnalités conformément au DOMTR.

De plus en s'éloignant de la plate-forme, React n'est plus dépendant du bon vouloir des navigateurs à implémenter la norme. On a donc un gain de portabilité et de stabilité puisqu'on ne dépend plus que du moteur javascript.

Pour quels besoins ?

React apparait donc comme un outsider rafraîchissant. Le projet est open-source maintenant et compte-tenu qu'il est partie intégrante de l'application Facebook, sa viabilité semble bonne. En revanche Polymer possède tout un aspect orienté design et animations que React supporte mais ne met pas en avant. Avec polymer on a à notre disposition les Paper Elements pour prototyper une interface qui répond au standards graphiques du moment très rapidement.

En revanche React permet d'obtenir des performances incroyables puisque dans tous les cas il effectura moins de manipulations du DOM que n'importe quel autre code issu de l'esprit humain. L'éditeur Atom de Github par exemple a tout récemment changé son moteur de rendu pour un conteneur web exploitant React. c'est pourtant une application C++ qui embarque le moteur web chromium dont les sources sont accessibles et donc modifiables pour gagner en performances pour ce cas particulier. Pourtant l'équipe a préféré React, ce qui prouve au moins que cette approche est suppérieure en terme de rapport effort/performances.

C'est vraiment sur le terrain des performances que les Web Components on du chemin à faire. Actuellement la majorité des components reposent sur Polymer, ce qui nécessite le téléchargement de toute la pile technologique qui pèse deux fois plus que Jquery dont beaucoup se plaignent déjà du poids. Ajouter à ça une requête HTTP pour télécharger chaque web component, Polymer est loin de se soucier des performances, même si avec le projet Vulcanize un pas est fait pour la minification et compression des sources.

Positionnement

Ces deux solutions se destinent donc à un panel très large de développeurs web désireux de mieux structurer leurs applications via des composants en jouant sur deux dimensions supplémentaires qui sont le design et les performances.

Tous les supports sont visés puisque le web est maintenant présent partout. Réutilisant les technologies telles que le CSS et le HTML, leur adaptation au contexte reste donc à la charge du développeur.

Polymer sera plus utilisé dans un environnement où prime design et respect des standards du web. React conviendra aux aficionados de la programmation fonctionnelle et ayant des problématiques de performances notamment pour l'affichage de gros jeux de données.