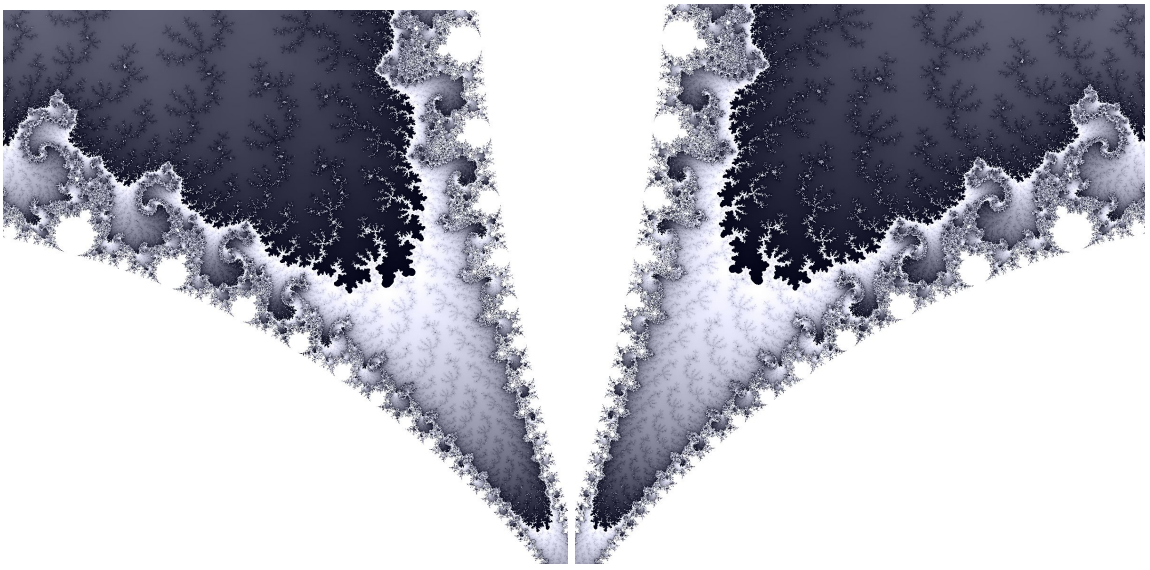
Construire un moteur d'animation 2D en JS



Version 0.1

Construire un moteur d'animation 2D en JS

Table des matières

Notes de l'auteur

Je m'appelle Grégory Jarrige.

Je suis développeur professionnel depuis 1991. Après avoir longtemps travaillé sur le développement d'applications de gestion, sur gros systèmes, avec des langages et technos propriétaires, j'ai fait le pari de me former aux technos et langage open source vers 2006. J'ai commencé à développer des applications webs professionnelles à partir de 2007, avant d'en faire mon activité principale à partir de 2010. L'arrivée du HTML5 dans la même période a été pour moi une véritable bénédiction, et surtout un formidable terrain d'expérimentation (avec des API comme Canvas, WebAudio, etc...).

Aujourd'hui, je développe des interfaces utilisateurs en Javascript sur des applications de type intranet, des tableaux de bord basés sur des solutions de dataviz comme D3.js, et des serveurs d'API basés sur NodeJS. Si j'ai longtemps dû faire le grand écart entre des technologies professionnelles et des technos plus ludiques, ce n'est plus le cas aujourd'hui : car j'utilise dans mon travail le même outil de base que dans mes projets de coding créatif, le Javascript (dont je suis fan). En clair, je suis un développeur heureux.

Ce document n'est pas exhaustif, il est forcément partiel et certainement aussi un peu partial. C'est un "work in progress", réalisé un peu dans l'urgence, pour une présentation faite au meetup CreativeCodeParis, le 20 décembre 2018.

Introduction

Dans le présent document, je souhaite expliquer les raisons qui m'ont amené à vouloir créer un petit moteur d'animation 2D en Javascrip, et je souhaite présenter un certain nombre d'auteurs qui m'ont inspiré et guidé dans mon développement.

Il faut souligner que, si le sujet me passionne, je ne suis en aucune manière un expert du développement d'animation. Mon expertise se situe plutôt dans le développement d'applications d'entreprise, aussi j'ai véritablement tout à apprendre dans le domaine, et ça tombe bien, j'adore apprendre des choses nouvelles, alors... "suivez le guide !".

Un élément important à prendre en compte, c'est que je m'efforce de construire en utilisant le meilleur des techniques disponible dans les versions ES5 et surtout ES6 du langage Javascript. Si vous n'êtes pas familiarisé avec ce que sont ES5 et ES6, lisez ce qui suit, sinon vous pouvez passer directement au prochain chapitre.

Le langage Javascript n'existe pas en tant que tel, c'est un nom d'usage, adopté par le plus grand nombre, pour désigner l'implémentation dans chaque navigateur de la norme ECMAScript 262 (souvent abrégée en EMCA-262, bon il est vrai qu'avec un nom pareil...).

Cette norme est régulièrement mise à jour et est publiée sur ce site :

http://www.ecma-international.org/publications/standards/Ecma-262.htm

Chaque navigateur est donc équipé d'un interpréteur de code Javascript, et il faut souligner que les éditeurs de navigateurs font aujourd'hui de gros efforts pour coller au plus près de la norme, et pour implémenter ses évolutions rapidement.

Si la norme EMCA-262 en est aujourd'hui à la version 9, publiée en juin 2018, il est intéressant de s'arrêter sur deux périodes clés de son évolution :

- L'édition 5 publiée en 2009, généralement appelée ECMAScript 5, constituait une avancée majeure. Elle apportait beaucoup de nouveautés qui auparavant étaient dispersées dans divers frameworks Javascript. Elle est arrivée en même temps que la norme HTML5 qui se son côté apportait un large panel d'API, toutes programmables en Javascript. Bref, l'année 2009 est véritablement une année charnière dans le monde du web. J'aime à penser qu'une petite révolution pacifique et bienveillante s'est produite, en 2009, dans nos navigateurs.
- L'édition 6 publiée en 2015, généralement appelée ECMAScript 6, ou quelquefois ECMAScript 2015 (ce qui a parfois entraîné des confusions dans les esprits). Plus discrète, mais tout aussi importante, cette édition a consolidé les bases posées par l'édition précédente, en apportant tout un tas de nouveautés très pratiques, notamment pour la manipulation de données. Nous en découvrirons certaines au fil de l'eau.

Ce qui est intéressant à souligner ici, c'est que nous sommes à la fin de l'année 2018 (au moment où j'écris ces lignes), que la norme ECMAScript 6 est là depuis 3 ans, et qu'elle est pleinement supportée par les navigateurs actuels. Si vous êtes un ou une « creative coder », vous pouvez donc vous lâcher. Si vous êtes un développeur professionnel, obligé de surveiller la compatibilité avec des navigateurs plus anciens ne supportant pas ES6, vous pouvez utiliser des solutions de type « transpilage » comme

le projet Babel (qui permet une conversion du code ES6 vers ES5), donc vous pouvez vous lâcher aussi sur ES6. Et puis si vous développez sous NodeJS, vous savez sans doute que l'interpréteur Javascript de NodeJS est à la pointe en ce qui concerne le support de la norme ECMASCript 262, donc tout ce que nous allons évoquer ici devrait vous intéresser (même si nous allons plus particulièrement nous intéresser à l'utilisateur de Javascript côté navigateur).

A qui est destiné ce support ?

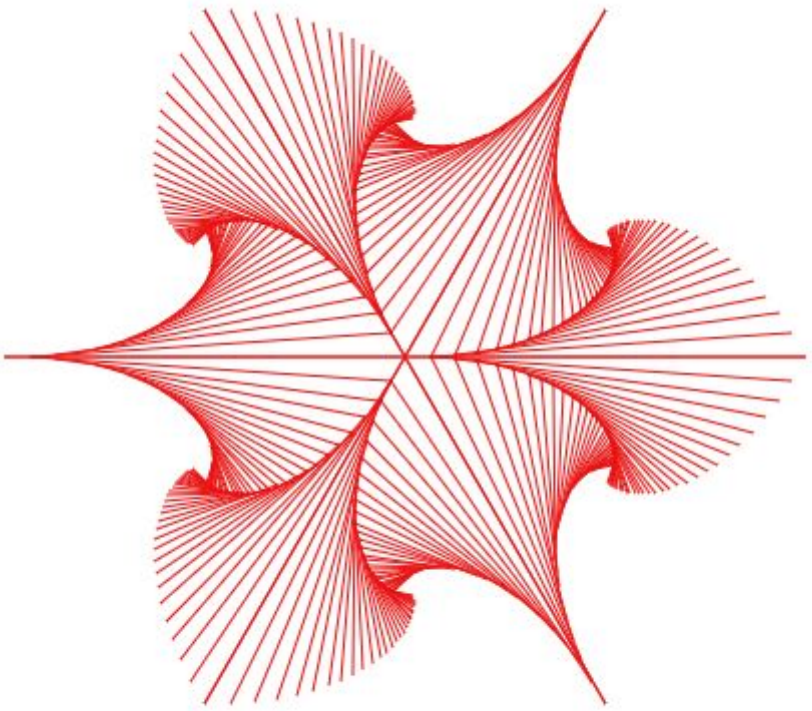
A un large public, à commencer par les membres du meetup CreativeCodeParis, qui sont de plus en plus nombreux à se lancer dans des projets créatifs exploitant l'API Canvas du HTML5.

Attention, ce document n'est pas destiné à de grands débutants en programmation (en tout cas dans pas dans cette première version). Aux débutants, je recommanderais plutôt le document PDF qui se trouve dans ce dépôt :

https://github.com/gregja/p5Corner

Canvas, une API géniale, mais...

Quand j'ai commencé à jouer avec l'API Canvas du HTML5 (c'était aux alentours de 2010 ou peut être 2011), mon premier réflexe a été de sortir mes vieux bouquins de programmation des années 80 et 90, et de m'essayer à la programmation de graphiques sympas, comme celui-ci :

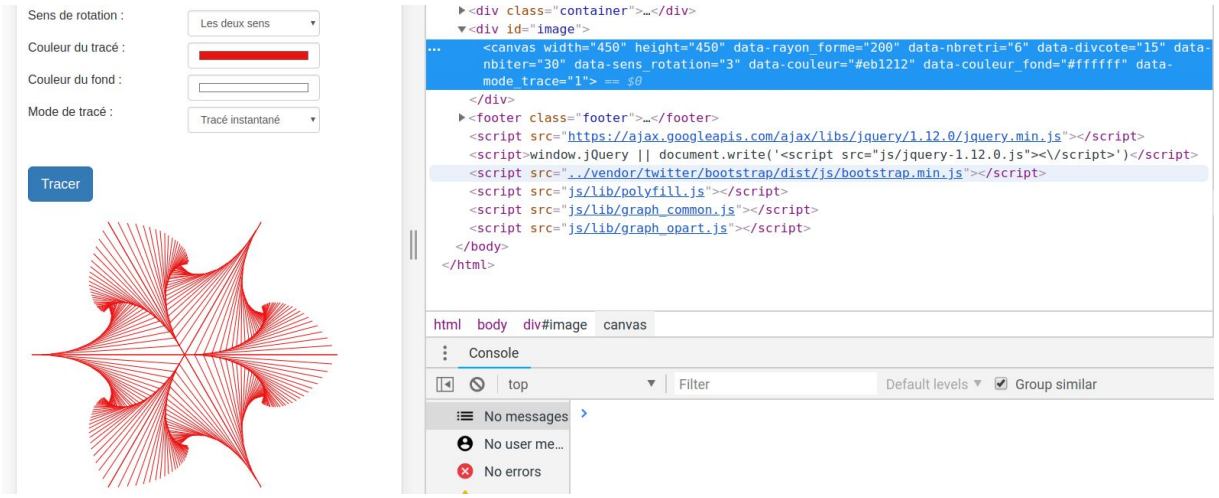


Ce graphe est généré à partir d'un algorithme pris dans un livre de Robert Dony qui s'intitule "graphisme dans le plan et dans l'espace avec Turbo Pascal" (éditions Masson, 1991).

C'était sympa de me replonger là dedans, j'étais content, alors j'ai poursuivi par l'étude des techniques d'animation disponibles avec l'API Canvas. Ca marchait plutôt bien, mais très vite je me suis heurté à un problème.

Pour dessiner des formes dans un canvas, on utilise des fonctions Javascript qui s'apparentent à du graphisme vectoriel, on obtient d'ailleurs un excellent rendu (cf. ci-dessus), mais une fois la forme dessinée dans le canvas, elle n'existe plus en tant qu'objet vectoriel. On obtient juste un objet de type "bitmap" (une matrice de pixels). C'est tout.

Quand on utilise l'inspecteur d'élément du navigateur, on voit très bien que le graphique à gauche est représenté par une unique balise "canvas" :



Or, j'aimerais bien lui tirer les oreilles à cette forme, saisir à la souris les fils, les tordre, les étirer, ça serait fun non ? Oui je sais, je suis un geek, et les geeks ont de drôles d'amusement… c'est comme ça, on ne se refait pas ;).

Bon, vous pourriez m'opposer qu'il y a une autre API, qui s'appelle SVG. Elle est dédiée au graphisme vectoriel justement, et elle est peut être un meilleur candidat pour ce que je viens de décrire (vous savez… tirer les fils et tout ça…). Oui mais moi je veux faire la même chose avec Canvas… je vous l'ai dit je suis un geek, et les geeks sont généralement têtus.

La solution à mon problème m'est apparue quelques temps plus tard, avec la découverte d'un cours proposé sur Paris par un développeur du nom de Nicolas Legrand.

Cours sur la construction d'un moteur d'animation 2D

Ce devait être en 2011 (ou peut être 2012), je découvre donc un cours sur Paris proposé par Nicolas Legrand. Je m'inscris et suis ce cours passionnant.

Je découvrirai à cette occasion que Nicolas a publié une partie importante de son support de cours sur ce site, où il est en téléchargement libre :

https://javascript.developpez.com/tutoriels/html5/creer-moteur-affichage-2d-html5/

Le cours de Nicolas est bien construit, il donne les bases pour bien démarrer avec l'API Canvas, mais surtout il donne les éléments permettant de dépasser les limites de l'API Canvas.

Voici la table des matières du support, pour vous faire une idée :

X-A. Créer une classe MovieClip permettant de jouer une animation.

XI. Les filtres

Le cours de Nicolas Legrand m'a beaucoup apporté. Il explique comment construire un moteur similaire à Tomahawk, un moteur de niveau professionnel qu'il a construit sur le même modèle.

Code source de Tomahawk :
https://github.com/thetinyspark/tomahawk/

Site web de l'agence de Nicolas Legrand :
http://the-tiny-spark.com/

A l'issue de la formation, je n'ai pas trouvé le temps nécessaire pour approfondir tout ce que Nicolas m'avait appris (trop de boulot, sur d'autres sujets trop éloignés de celui-ci).
Entre temps, j'ai découvert le projet P5.js, du coup j'ai eu envie de jouer avec. Mais après la phase de découverte de P5, je me suis rendu compte que je ne parvenais pas à aller beaucoup plus loin, avec P5, qu'avec l'API Canvas. Car P5 est lui aussi dépourvu de ce moteur d'animation que Nicolas m'avait appris à construire.

A noter qu'il existe dans le projet Processing (le grand frère de P5), un objet PShape, qui pourrait combler partiellement les lacunes de P5, mais aux dernières nouvelles, il n'a pas été reporté dans P5 :
https://www.processing.org/reference/PShape.html

Bon alors, t'en es où mec ?

De l'eau a coulé sous les ponts, la norme ES6 (de 2015) est arrivée avec plein de nouveautés sympas (comme les classes et l'héritage), dont j'ai très vite compris qu'elles allaient me permettre de revenir au sujet étudié avec Nicolas.

Il faut savoir que le moteur de Nicolas est fondé sur ES5 (version 2009). Or avec ES5, il fallait jongler (truander ?) pas mal pour contourner certaines limitations, notamment l'absence d'un mécanisme d'héritage natif. Nicolas proposait dans son cours une solution de contournement assez élégante, mais quand même très spécifique. Avec ES6, certaines contraintes sont levées, et cela devient plus simple de construire certains des designs patterns nécessaires au moteur d'animation de Nicolas.

Donc après avoir longtemps tergiversé, en cette fin d'année 2018, j'ai décidé de m'attaquer au problème, avec un double objectif :

- reconstruire un moteur sur le modèle de celui de Nicolas, mais en profitant du meilleur de la norme ES6 (classes et héritage inclus)
- essayer de construire un moteur qui permette de pallier la principale carence de P5 (son absence de moteur d'animation justement), et peut être construire un moteur qui puisse s'utiliser à la fois avec P5 et indépendamment de P5

Et je suis heureux de pouvoir vous dire que mon moteur existe, il est loin d'être terminé, il y manque par exemple une gestion de collision plus avancée (pour des formes autres que le parallélogramme), mais ça avance doucement. Et il manque surtout une documentation (là y'a du boulot !).

Le code source se trouve sur ce dépôt Github :

https://github.com/gregja/automatosjs

Vous pouvez le télécharger et vous amuser avec. Vous pouvez aussi me proposer des améliorations, toute remarque constructive est bienvenue.

Et j'espère que ceux des lecteurs qui n'ont pu être présents lors du meetup CreativeCodeParis du 20 décembre 2018 voudront bien me pardonner, mais il est temps de passer à la démo "live".

Bibliographie

Je dois souligner, qu'en plus du support de Nicolas Legrand, plusieurs livres m'ont apporté des éclairages intéressants, je vais les présenter brièvement dans ce chapitre.
Pour rappel, le lien vers le support de cours qui a constitué le point de départ de mon projet :

Comment créer un moteur d'animation 2D, de Nicolas Legrand
https://javascript.developpez.com/tutoriels/html5/creer-moteur-affichage-2d-html5/

Plusieurs ouvrages importants pour mieux maîtriser certains concepts graphiques, mathématiques et physique de l'animation 2D. Ces livres sont de véritables références. Certains sont consacrés à Actionscript, une version de Javascript bâtie par les ingénieurs d'Adobe pour les besoins de Flash. Actionscript était plus avancé que Javascript, il était notamment équipé de ce moteur d'animation 2D qui fait défaut sur Canvas.

Programming Macromedia Flash MX, par Robert Penner's, ed Mc Graw Hill (2002)

Si je ne dois emmener qu'un bouquin sur une île déserte, c'est celui là que je choisis. Robert Penner, excellent pédagogue, avait rédigé un formidable travail de synthèse des bonnes pratiques du développement Actionscript. Si certaines techniques qu'il présente ne sont plus adaptées au Javascript d'aujourd'hui, l'essentiel de ce qu'il faut savoir pour coder de l'animation 2D se trouve dans ce livre. Attention, c'est un livre destiné à des développeurs aguerris, je ne le recommande pas à un développeur débutant en JS.

A noter que le livre de Robert Penner a connu un certain succès auprès des développeurs JS, en particulier pour les fonctions "easing" qu'il décrit de manière détaillée dans son livre (fonctions qui permettent de gérer des effets d'accélération et de décélération). Ces fonctions ont été largement reprises dans des frameworks comme D3.js et jQuery. Le chapitre sur les fonctions "easing" est disponible sur cette page : http://robertpenner.com/easing/

Foundation ActionScript 3.0 Animation, by Keith Peters, ed FriendsofED / Apress (2007)

Très bon livre de Keith Peters, qui est l'animateur de la chaîne Youtube "the codin math", très complémentaire de celui de Robert Penner. Il existe une version JS de ce livre, c'est celle que je décris ci-dessus.

Foundation HTML5 Animation with Javascript, Billy Lamberta, ed FriendsofED / Apress (2011)

Excellente adaptation au JS, par Billy Lamberta, du livre de Keith Peters. Tous les exemples du livre de Keith Peters sont ici traduits en pur JS, et on peut dire que ce livre constitue une sorte de pierre de Rosette pour tout développeur désireux de comprendre les différences entre Actionscript et Javascript.

Flash Math Creativity, 2nd edition, ed FriendsofED / Apress (2004)

Plein de belles idées à explorer dans ce beau livre, qui démontre qu'en terme de créativité, les développeurs Actionscript étaient véritablement au top, durant l'âge d'or de Flash (soit les années 2000).