



A BOOK APART

*Les livres de ceux qui font le web*

No.

4

Ethan Marcotte

---

# RESPONSIVE WEB DESIGN

---

PRÉFACE DE Jeremy Keith

Découvrez le responsive Web design, et apprenez à concevoir des sites Web agréables qui anticipent et répondent aux besoins de vos utilisateurs. En explorant des techniques CSS et des principes généraux de design, comme les grilles fluides, les images flexibles et les media queries, Ethan Marcotte démontre qu'il est possible d'offrir une expérience utilisateur de qualité, quelle que soit la taille, la résolution ou l'orientation de l'écran qui affiche le site.

---

## Au sommaire

**Principes du responsive design** \* Attachez vos ceintures \* Responsive architecture \* La voie à suivre \* **La grille flexible** \* Composition flexible \* Créer une grille flexible \* Marges et espacement flexibles \* **Les images flexibles** \* Retour aux (codes) sources \* Images fluides \* Mosaïque d'arrière-plan flexible \* Apprenez à aimer overflow \* Négociez votre contenu \* Images et grilles flexibles, tenez-vous bien \* **Les media queries** \* Cicatrisation douloureuse \* Le problème en question \* Traînasser vers plus de réactivité \* Un robot plus « responsive » \* Les media queries en action \* Au sujet de la compatibilité \* Pourquoi la flexibilité ? \* **Passer au responsive design** \* Une question de contexte \* Mobile first \* Vers un responsive workflow \* Être « responsive » et responsable \* L'amélioration progressive revisitée \* Va et sois « responsive » \* **Ressources**



A BOOK APART

*Les livres de ceux qui font le web*

Ethan Marcotte

---

# RESPONSIVE WEB DESIGN

ÉDITIONS EYROLLES  
61, bld Saint-Germain  
75240 Paris Cedex 05  
[www.editions-eyrolles.com](http://www.editions-eyrolles.com)

Traduction autorisée de l'ouvrage en langue anglaise  
intitulé *RESPONSIVE WEB DESIGN* de Ethan Marcotte  
(ISBN : 978-0-9844425-7-7), publié par A Book Apart LLC

Adapté de l'anglais par Charles Robert

*Dans la même collection*

*HTML5 pour les Web Designers*, Jeremy Keith, 2010, 96 pages.  
*CSS3 pour les Web Designers*, Dan Cederholm, 2011, 128 pages.  
*Stratégie de contenu Web*, Erin Kissane, 2011, 96 pages.

© 2011 Ethan Marcotte pour l'édition en langue anglaise  
© Groupe Eyrolles, 2011, pour la présente édition,  
ISBN : 978-2-212-13331-8

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans autorisation de l'éditeur ou du Centre Français d'Exploitation du Droit de Copie, 20, rue des Grands Augustins, 75006 Paris.

# TABLE DES MATIÈRES

1	CHAPITRE 1 Principes du responsive design
13	CHAPITRE 2 La grille flexible
42	CHAPITRE 3 Les images flexibles
64	CHAPITRE 4 Les media queries
106	CHAPITRE 5 Passer au responsive design
140	<i>Remerciements</i>
142	<i>Ressources</i>
144	<i>Références</i>
147	<i>Index</i>



## AVANT-PROPOS

Le langage a des propriétés magiques. Le mot « glamour », à l'origine synonyme de magie ou d'envoûtement, tient ses origines du mot « grammaire ». De toutes les facultés du langage, le pouvoir de nommer est bien la plus magique et la plus puissante.

La courte histoire du design Web nous a déjà prouvé le pouvoir de transformation du langage. Jeffrey Zeldman nous a donné l'expression « standards du Web » derrière laquelle nous rassembler. En forgeant le mot « Ajax », Jesse James Garrett a changé la nature de l'interactivité sur le Web.

Quand Ethan Marcotte a employé pour la première fois l'expression « Responsive Web design », il a créé quelque chose de spécial. Les technologies existaient déjà : grilles fluides, images flexibles et media queries. Mais en ralliant ces techniques sous la même bannière, Ethan a changé notre façon de penser le design Web.

Ethan a le sens de la formule. Il est clairement la personne la mieux placée pour écrire un livre sur le responsive Web design. Mais il a fait bien mieux : il a écrit *le* livre sur le responsive Web design.

Si vous espérez trouver ici une liste de trucs et astuces pour ajouter quelques artifices superficiels sur vos sites Web, passez votre chemin. Ce petit bijou va bien plus loin que ça.

Quand vous aurez fini ce livre (ce qui ne devrait pas prendre très longtemps), prenez attention à la manière dont vous aborderez votre prochain projet. Il est possible que vous ne remarquiez même pas le pouvoir de transformation des mots d'Ethan, à travers son style enjoué, divertissant, parfois tout simplement hilarant ; mais je peux vous garantir que le tour de prestidigitation qu'il est sur le point d'opérer sur vos neurones vous sera grandement bénéfique.

Ethan Marcotte est un magicien. Préparez-vous à être envoûté.

Jeremy Keith



# 1 PRINCIPES DU RESPONSIVE DESIGN

“Something there is that doesn’t love a wall...”

—ROBERT FROST, “Mending Wall”

EN ÉCRIVANT CES PREMIÈRES LIGNES, je réalise que vous ne lirez peut-être pas ces mots sur une page imprimée, un petit livre relié entre les mains. Vous êtes peut-être assis à votre bureau, une copie électronique du livre affichée sur votre écran, ou dans les transports, en train de feuilleter sur votre téléphone ou votre tablette. Si ça se trouve, vous ne voyez pas ces mots comme moi : votre ordinateur est peut-être en train de les lire à haute voix.

Au fond, je sais si peu de choses de vous. Je ne sais pas comment vous lisez ce livre. Je n’ai aucun moyen de le savoir.

Le monde de l’édition a fini par hériter de l’une des caractéristiques fondamentales du Web : la flexibilité. L’éditeur et designer de livres Craig Mod pense que son industrie évolue rapidement vers une phase « post-objet » (<http://bkaprt.com/rwd/1/>), et que l’ère numérique est en train de redéfinir la notion même de « livre ».



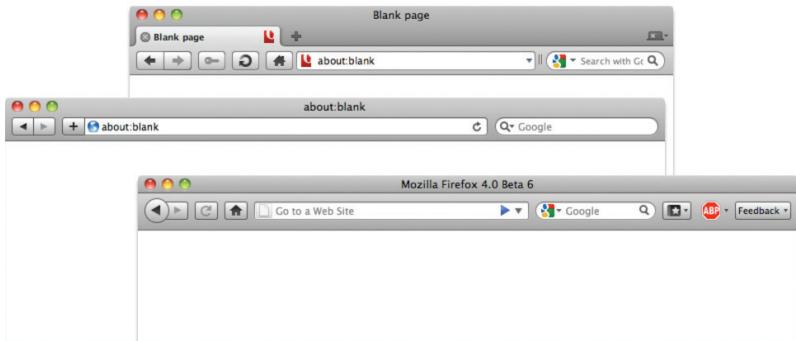
**FIG. 1.1 :** La toile, même vierge, impose des limites au travail de l'artiste. (Photo de Cara StHilaire : <http://bkaprt.com/rwd/2/>)

---

Rien de bien nouveau pour nous autres designers Web. En fin de compte, notre profession n'a jamais eu d'objet propre. Il n'y a pas de *chose* produite sur le Web, pas d'objet palpable à tenir entre ses mains, à chérir et à transmettre à ses enfants. Mais en dépit de la nature si éthérée de notre travail, le vocabulaire que nous utilisons pour en parler est bien concret : en-tête, espace, interligne... Chacun de ces mots nous vient directement de l'imprimerie. On n'a fait que les dépoussiérer pour les appliquer à notre nouveau support numérique.

Ce recyclage est parfaitement naturel. Nous sommes, après tout, des êtres routiniers. Quand nous déménageons, quand nous changeons de travail, nous appliquons notre expérience passée à un contexte étranger, afin de trouver progressivement

---



**FIG 1.2 :** La fenêtre du navigateur, notre support (pour le meilleur et pour le pire).

---

nos repères. Comme le Web est de toute façon un support récent, il est tout à fait normal d'emprunter certains termes à des disciplines que nous connaissons : la conception graphique est riche d'une histoire de plusieurs siècles, et il serait bête de ne pas se servir de son vocabulaire pour aider notre industrie à prendre forme.

Mais nos emprunts à d'autres disciplines ne s'arrêtent pas au langage. En fait, nous avons emprunté un autre concept, auquel on ne pense pas souvent : la toile (**FIG 1.1**).

Un artiste commence avant tout par choisir un support. Un peintre choisit une feuille de papier ou une toile ; un sculpteur choisira un bloc de pierre dans une carrière. Quel qu'il soit, le choix du support est un acte créatif puissant : avant le premier coup de pinceau ou de burin, le support donne une dimension, une forme, une hauteur et une largeur, établissant les limites de l'œuvre naissante.

Sur le Web, on essaie d'imiter ce procédé. On crée un « canvas » dans notre éditeur d'images préféré, un document vierge doté d'une hauteur et d'une largeur, d'une dimension et d'une forme. Le problème de cette approche, c'est qu'elle nous éloigne déjà de notre *véritable* toile : la fenêtre du navigateur, avec tous ses défauts et ses incohérences (**FIG 1.2**). Car soyons honnêtes : une fois en ligne, nos designs sont à la merci immédiate de ceux qui les visualisent — de leurs polices de caractères,

---



**FIG 1.3 :** Le fait de dévier légèrement de nos paramètres « idéaux » peut affecter l'utilisateur...

des couleurs de leur écran, de la forme et de la taille de la fenêtre de leur navigateur.

Face à toutes ces incertitudes, à cette flexibilité, on commence donc par établir des contraintes, en définissant la taille de nos polices en pixels ou en créant des mises en page à largeur fixe pour une résolution minimum. Le choix de ces contraintes s'apparente un peu au choix d'une toile : elles nous donnent des paramètres à prendre en compte, des certitudes qui nous aident à préserver notre travail de la flexibilité inhérente du Web.

Mais le meilleur (et souvent le pire) aspect du Web, c'est qu'il échappe à toute définition simpliste. Si je voulais être cynique, j'irais même jusqu'à dire qu'il excelle dans sa capacité à se jouer des contraintes qu'on lui impose. Et les paramètres que l'on place dans nos designs ne font pas exception : ils sont facilement défaits. S'il réduit la taille de son navigateur en deçà de notre largeur minimale (**FIG 1.3**), notre visiteur verra s'ajouter



FIG 1.4 : ... voire notre activité et nos clients. (C'est quoi ce « Reg » me demandez-vous ? C'est le bouton d'abonnement « Register now ».)

une barre de défilement horizontale et un contenu tronqué. Mais cela peut également avoir une incidence sur notre activité et nos clients (FIG 1.4) : dans une mise en page fixe, le placement de liens ou d'éléments critiques peut s'avérer extrêmement fragile, coupé par une fenêtre qui obéit aux préférences de l'utilisateur — pas aux nôtres.

## ATTACHEZ VOS CEINTURES

Il y a plus d'une décennie, John Allsopp écrivait « A Dao of Web Design » (<http://bkaprt.com/rwd/3/>), un article que je vous invite à aller lire maintenant si ce n'est déjà fait. (Sérieusement, allez-y, je peux attendre.) C'est de loin mon essai préféré sur le design Web, et il est tout aussi pertinent aujourd'hui qu'il l'était à l'époque où il a été écrit. John soutient :

*Le contrôle que les designers connaissent avec le support imprimé, et aimeraient parfois avoir sur le Web, n'est qu'une fonction des limitations de la page imprimée. Nous devrions accepter le fait que le Web n'impose pas les mêmes contraintes, et concevoir en fonction de cette flexibilité. Mais nous devons d'abord « accepter l'impermanence des choses ».*

C'était les premières années du Web, une période de transition pendant laquelle les designers transposaient les principes appli-

qués dans l'édition sur ce nouveau support. Mais une grande partie de ce que John écrivait il y a dix ans est toujours valable. Car le Web n'a jamais été aussi fluctuant et variable qu'aujourd'hui.

Au fond, cela fait un moment que nous traversons notre propre période de transition. Avec des appareils qui deviennent à la fois plus petits *et* plus grands, le navigateur s'éloigne de plus en plus du bureau. On estime que les appareils à petit écran seront la forme prédominante d'accès au Web d'ici quelques années (<http://bkaprt.com/rwd/4/>), alors que les consoles de jeu modernes démocratisent l'accès au Web sur grand écran. Ces derniers temps, les tablettes ont de plus en plus la cote et sont un mode d'accès au Web qui n'est ni complètement « mobile » ni complètement « bureau », mais quelque part entre les deux.

En clair, nous devons nous adapter à plus d'appareils, de modes de saisie, de *solutions* que jamais. Le Web a quitté le bureau, et il n'est pas près d'y retourner.

Malheureusement, nos premières tentatives de design mobile s'apparentaient à nos vieilles approches : dans le doute, appliquer des contraintes. Il y a quelques mois, une amie m'a envoyé un lien vers un article qu'elle venait de lire sur son téléphone :

[http://www.bbc.co.uk/worldservice/mobile/world\\_service\\_bulletin.htm](http://www.bbc.co.uk/worldservice/mobile/world_service_bulletin.htm)

Vous voyez le répertoire [/mobile/](#) ? Le propriétaire du site avait créé une URL séparée pour cloisonner l' « expérience mobile », adaptée à une largeur de 320 pixels. Mais si quelqu'un décidait de partager ce lien sur Twitter, Facebook ou par email, les visiteurs se retrouvaient bloqués sur cette vue adaptée aux petits écrans, quel que soit l'appareil utilisé. En ce qui me concerne, sur le navigateur de mon ordinateur de bureau, la lecture était... eh bien, catastrophique.

Cela ne veut pas dire que cette approche comporte une faille inhérente, ou qu'il n'existe pas de raisons légitimes de créer un site Web distinct pour les appareils mobiles. Mais je pense tout de même que la fragmentation de notre contenu entre plusieurs expériences optimisées pour chaque type d'appareil est une mauvaise marche à suivre, du moins à long terme. Comme on a pu le voir ces dernières années, on ne peut tout simplement pas

suivre le rythme des avancées technologiques. Peut-on vraiment créer des expériences sur mesure pour chaque nouveau navigateur ou type d'appareil qui apparaît ?

Et sinon, quelle est l'alternative ?

## RESPONSIVE ARCHITECTURE

Je suis amateur d'architecture depuis aussi longtemps que je m'en souvienne. Pour un designer Web, il y a quelque chose de fascinant dans toutes les contraintes que les architectes semblent apprécier : du croquis au schéma, des fondations à la façade, chaque étape du processus architectural est plus permanente que la précédente. Dans *Parentalia*, l'architecte anglais Christopher Wren écrit que « l'architecture vise l'éternité », et il y a du vrai là-dedans : les décisions créatrices de l'architecte subsistent pendant des décennies, voire des siècles.

Après une journée passée à maudire Internet Explorer, ce genre de stabilité semble très, très attrayant.

Mais depuis quelques années, une discipline relativement nouvelle appelée *responsive architecture*<sup>1</sup> défie quelque peu la permanence architecturale. C'est une discipline toute jeune, mais cette forme d'architecture plus interactive s'est déjà manifestée de plusieurs façons intéressantes.

Des artistes ont expérimenté avec des surfaces qui réagissent au son de la voix d'une manière très spéciale (<http://bkaprt.com/rwd/5/>) et avec des espaces de vie qui peuvent changer de forme pour mieux accueillir leurs occupants (<http://bkaprt.com/rwd/6/>). Une entreprise a produit un système de « verre intelligent » capable de s'opacifier quand les occupants d'une pièce dépassent un certain palier de densité, bénéficiant ainsi une couche d'intimité supplémentaire (FIG 1.5). En combinant des matériaux élastiques et de la robotique embarquée, une firme de design allemande a créé un « mur » qui se plie et se déforme quand on s'en approche, pouvant potentiellement créer plus ou moins d'espace selon le nombre de personnes présentes (FIG 1.6).

---

1. Parfois traduit en français par « architecture réactive » ou « interactive ». (NdT)



**FIG 1.5** : Jour, nuit, jour, nuit : le verre intelligent peut être configuré pour s'opacifier automatiquement (<http://bkaprt.com/rwd/7>).

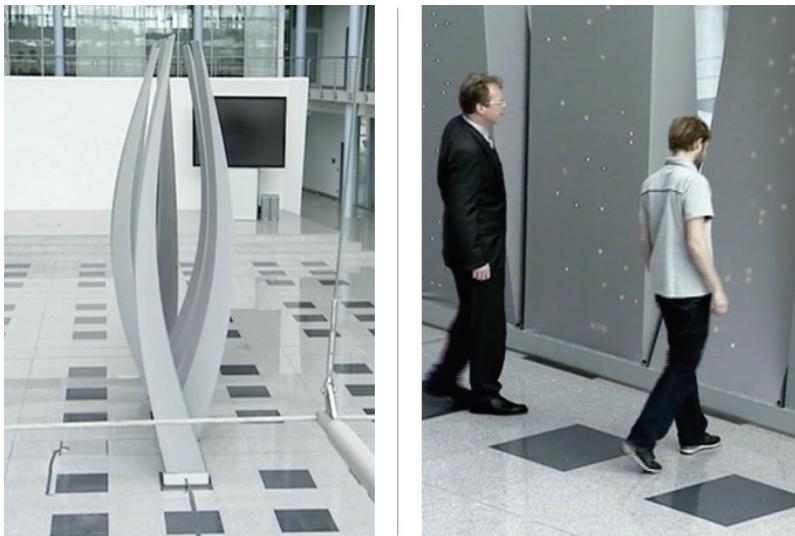
Plutôt que de créer des espaces qui influent sur le comportement des personnes qui y accèdent, le responsive designer cherche à ce que l'architecture et ses habitants s'influencent et s'informent mutuellement.

## LA VOIE À SUIVRE

Ce qui me fascine chez les architectes, c'est qu'ils essaient de dépasser les contraintes inhérentes à leur support. Mais nous autres designers Web, face à cette multitude de nouveaux appareils et de nouveaux contextes, sommes maintenant forcés de briser les contraintes que *nous* avons imposées à la flexibilité naturelle du Web.

Nous devons lâcher du lest.

Plutôt que de créer des designs déconnectés, conçus chacun pour un appareil ou un navigateur particulier, nous devrions les traiter comme les facettes d'une même expérience. Il est possible de créer des sites plus flexibles, qui en plus s'adaptent aux supports qui les restituent.



**FIG 1.6 :** Plus qu'une intéressante installation artistique, ce mur peut réellement détecter votre présence et se déformer à votre approche (<http://bkaprt.com/rwd/8/>).

En un mot, nous devons adopter le **responsive Web design**<sup>1</sup>. Nous pouvons nous approprier la flexibilité inhérente du Web sans abandonner le contrôle dont nous avons besoin en tant que designers. Tout cela en incorporant des technologies standards dans notre travail, et en changeant légèrement notre philosophie du design Web.

## Les ingrédients

Alors, que faut-il pour créer un design réactif ? Pour ne parler que de la mise en page, il y a trois ingrédients clés :

1. Une **grille de mise en page flexible**,
2. Des **images et des médias flexibles**,
3. Les **media queries**, un module de la spécification CSS3.

1. Parfois traduit en français par « réactif » ou « adaptatif ». (NdT)

Dans les trois prochains chapitres, nous étudierons chacun de ces éléments tour à tour — la grille flexible, les images et les médias fluides, ainsi que les media queries CSS3 — afin de développer une approche plus flexible et plus réactive du design Web. Ce faisant, nous aurons créé un design qui peut s'adapter aux contraintes de divers navigateurs et appareils tout en répondant aux besoins de l'utilisateur.

Mais avant d'attaquer, il faut que je vous avoue : je suis fan de science-fiction. J'aime les pistolets laser, les androïdes et les voitures volantes, mais aussi les films et les séries qui en comportent en copieuses quantités. Et franchement, je me fiche pas mal de la qualité desdits films et séries. Que ce soit réalisé par Kubrick ou avec le budget de mon repas de midi, du moment qu'il y a au moins un vaisseau spatial, je suis comblé.

Dans tous les films de science-fiction que j'ai pu voir, bons ou moins bons, il y a un procédé narratif que les écrivains du genre semblent adorer : le robot secret. Vous êtes sûrement déjà tombé sur une histoire comme ça. Ça commence toujours avec une bande d'aventuriers courageux qui veulent vaincre une sorte de mal sans visage, menés par un héros modèle lui-même armé de répliques cinglantes et/ou d'une détermination de fer. Mais dans leurs rangs se cache, incognito... *un robot secret*. (Musique inquiétante.) Cet appareil sournois et maléfique, fait d'acier froid et de machinations plus froides encore, a été conçu pour ressembler à un être humain. Son objectif est clairement malfaisant : exterminer notre bande de héros de l'intérieur.

La révélation de l'identité du robot secret est le point culminant de l'histoire. Vous savez qui est le héros, vous savez qui est le robot espion, d'accord. Mais une question reste toujours en suspens : parmi les autres personnages, qui est un robot et qui ne l'est pas ?

Je n'ai jamais compris pourquoi il fallait se donner tant de mal. Moi, je suis un nostalgique de Johnny 5 et de Z-6PO, de l'époque où on savait qui était un robot au premier coup d'œil et où il n'y avait pas toutes ces saloperies d'espions en peau synthétique. Alors j'ai décidé de prendre le problème à bras-le-corps : pour lever le voile, j'ai conçu un petit site tout simple appelé « Robot or Not » (FIG 1.7). Il permet de déterminer exactement qui est et qui n'est pas un robot, afin de nous aider



**FIG 1.7 :** Le design de Robot or Not, dans toute sa splendeur robotisée.

à mieux distinguer nos amis faits de chair et de sang de nos ennemis en titane.

Bon, peut-être que je suis le seul à avoir ce problème.

Mais quelle que soit l'utilité réelle de ce site, nous utiliserons son modeste design pour démontrer précisément comment on construit un site adaptatif. Au fil des prochains chapitres, nous développerons Robot or Not ensemble, avec des grilles flexibles, des images flexibles et des media queries.

Vous n'êtes peut-être pas amateur de suspens. Ou plus vraisemblablement, vous en avez peut-être déjà marre de lire mes âneries et vous voulez simplement voir le produit fini. Si c'est le cas, entrez l'adresse <http://responsivewebdesign.com/robot/> dans votre navigateur, et n'hésitez pas à faire un tour de piste.

Toujours là ? Super. Commençons.



# LA GRILLE FLEXIBLE

QUAND J’ÉTAIS À LA FAC, un professeur m’a dit un jour que chaque mouvement artistique — en musique, en littérature ou dans les beaux-arts — pouvait être considéré comme la réponse au mouvement précédent. Les producteurs des années soixante ont produit *Bonnie et Clyde* et *Le Lauréat* pour contrer les vieux films hollywoodiens comme *La Mélodie du bonheur*. Dans *Le Paradis perdu*, John Milton, pas très subtil, brocarde ses prédécesseurs littéraires en les plaçant dans le décor de l’enfer. Et Charlie Parker lui, n’aurait peut-être jamais produit cette expérimentation folle et chimérique que fut le be-bop sans les arrangements au cordeau de Duke Ellington et de Benny Goodman.

Un artiste définit un point ; un autre, le contrepoint. Et c’était particulièrement vrai pour les artistes de la période moderniste du milieu du xx<sup>e</sup> siècle. Les modernistes portaient un regard un peu méprisant sur la production créatrice de leurs prédécesseurs, les romantiques de la fin du xixe<sup>e</sup>. Pour eux, l’art romantique était surchargé d’ornements inutiles qui submergeaient l’œuvre et entravaient la communication avec le public (FIG 2.1).



**FIG 2.1 :** Les modernistes prenaient leurs distances avec le réalisme embelli de William Blake et d'Eugène Delacroix pour adopter l'approche plus rationnelle de Hans Hofmann et Josef Müller-Brockmann.

La réaction moderniste prit bien des formes, sur pratiquement tous les supports artistiques. En peinture, elle consistait à réduire le superflu pour expérimenter avec les courbes, les formes et la couleur. Les graphistes de l'époque, comme Jan Tschichold, Emil Ruder et Josef Müller-Brockmann, popularisèrent le concept de grille typographique : un système rationnel de colonnes et de lignes, sur lesquelles des modules de contenu peuvent être placés (FIG 2.2). Et grâce à des designers comme Khoi Vinh et Mark Boulton, nous sommes parvenus à adapter ce vieux concept aux besoins du design Web contemporain.

Müller-Brockmann, dans son livre *Grid Systems in Graphic Design*, parle de « créer un espace typographique sur la page » en adaptant les proportions de la grille à la taille d'une feuille de papier vierge. Mais pour un designer Web, il manque un élément essentiel : une vraie feuille de papier. Notre toile, la fenêtre du navigateur, peut se déformer et prendre n'importe quelle taille, selon la volonté du lecteur ou l'appareil qu'il utilise pour visualiser notre contenu.

Souvent, la première couche de notre grille de mise en page ressemble à ça :

```
#page {  
    width: 960px;
```

Winter 2007 Season	Aaron Copland <b>The Tender Land</b> January 2007	Eric Satie <b>Gymnopédie 1, 2</b> February 2007
	01/12/07 Middlebury College Center for the Arts 8:00 pm	02/03/07 Johnson State College Dibden Center for the Arts 8:00 pm
	01/19/07 Johnson State College Dibden Center for the Arts 8:00 pm	02/10/07 Castleton State College Fine Arts Center 8:00 pm
	01/26/07 Lyndon State College Alexander Twilight Theater 8:00 pm	02/13/07 Middlebury College Center for the Arts 8:00 pm

**FIG 2.2 :** Quand elle est adaptée aux besoins de votre contenu et aux dimensions de la page, la grille typographique est un outil puissant et une aide précieuse pour le designer comme pour le lecteur.

```
margin: 0 auto;
}
```

On crée un élément doté d'une largeur fixe dans notre feuille de styles, puis on le place au centre de la page. Mais pour penser de manière flexible, il nous faut plutôt traduire notre design créé sous Photoshop (FIG 2.3) en quelque chose de plus fluide, de plus *proportionnel*.

Par où commencer ?

## COMPOSITION FLEXIBLE

Pour trouver une réponse, faisons un petit jeu de rôle. Non, non, vous pouvez ranger votre D20 ; j'avais quelque chose d'un peu plus pratique (et d'un peu moins orc-oriented) en tête.

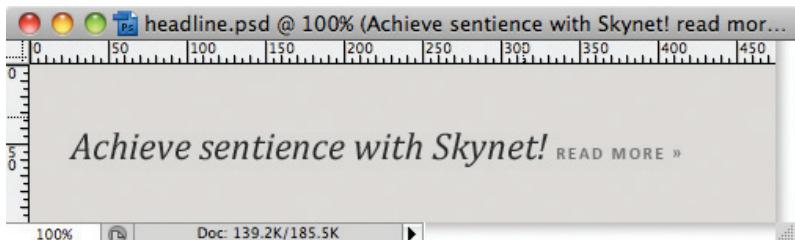
Imaginez un instant que vous êtes développeur front-end. (Si vous l'êtes déjà, imaginez que vous portez un chapeau de pirate.) Une designer de votre équipe vous a demandé de repro-



**FIG 2.3 :** Notre PSD est bien joli, mais cette grille est un peu trop pleine de pixels. Comment la rendre plus flexible ?

duire un design simple en HTML et en CSS. Toujours partant pour donner un coup de main, vous ouvrez le PSD qu'elle vous a envoyé (FIG 2.4).

Il n'y a pas beaucoup de contenu, c'est vrai, mais même



**FIG 2.4 :** La maquette de notre exercice de composition. Ça ne devrait pas prendre bien longtemps.

les plus petits projets doivent être étudiés de près ; vous vous penchez donc sérieusement sur la question. Après avoir relevé les différents types de contenu présents sur la maquette, voilà le code HTML que vous pondez :

```
<h1>Achieve sentience with Skynet! <a href="#">Read more  
&raquo;</a></h1>
```

Un lien dans une balise de titre — une bonne base de HTML sémantique, pas vrai ? Après avoir déclaré un reset CSS, le contenu commence à prendre forme dans votre navigateur (FIG 2.5).

C'est un modeste début. Mais maintenant que nos fondations sont en place, nous pouvons commencer à ajouter une couche de style. Commençons par appliquer quelques règles élémentaires à l'élément **body** :

```
body {  
    background-color: #DCDBD9;  
    color: #2C2C2C;  
    font: normal 100% Cambria, Georgia, serif;  
}
```

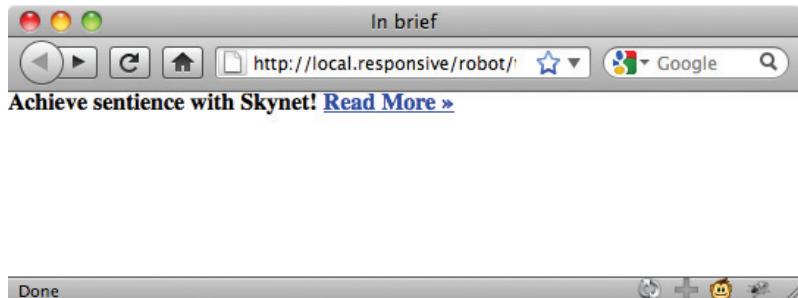


FIG 2.5 : Du HTML simple et sans fioritures. Ce dont les rêves (et les sites Web) sont faits.

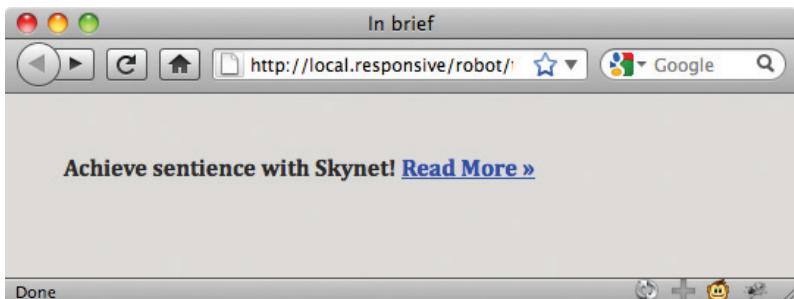
Rien de bien sorcier : on applique une couleur d'arrière-plan gris clair (#DCDBD9) à notre document, et une couleur de texte plutôt sombre (#2C2C2C). On définit ensuite la typographie : une graisse par défaut (`normal`) et des polices à empattement (`Cambria`, `Georgia`, `serif`).

Enfin, vous aurez probablement remarqué que la propriété `font-size` a été fixée à `100%`. Ce faisant, nous avons simplement pris la valeur par défaut du navigateur comme taille de police de base, soit 16 pixels en général. Nous pourrons ensuite utiliser des unités relatives pour agrandir ou réduire la taille du texte par rapport à cette valeur de référence. Mais nous pouvons déjà voir que notre titre commence à prendre forme (FIG 2.6).

Vous vous demandez pourquoi notre `h1` n'a pas vraiment une allure de titre ? Nous utilisons un reset CSS, qui réinitialise les styles par défaut du navigateur pour les éléments HTML. C'est une méthode pratique pour que tous les navigateurs partent sur les mêmes bases. En ce qui me concerne, je suis un grand fan du reset d'Eric Meyer (<http://bkaprt.com/rwd/9/>), mais il existe des dizaines d'alternatives tout à fait correctes.

Quoi qu'il en soit, c'est pour cette raison que notre `h1` paraît si petit : il ne fait qu'utiliser la `font-size` de `100%` que nous avons définie dans l'élément `body`, qui se traduit par une taille de police par défaut de 16 pixels.

Si l'on voulait se contenter de pixels, on pourrait simplement transposer les valeurs de la maquette dans notre feuille de styles :



**FIG 2.6 :** Avec une seule règle de CSS toute simple, on peut définir les paramètres généraux de notre design.

```
h1 {  
    font-size: 24px;  
    font-style: italic;  
    font-weight: normal;  
}  
  
h1 a {  
    color: #747474;  
    font: bold 11px Calibri, Optima, Arial, sans-serif;  
    letter-spacing: 0.15em;  
    text-transform: uppercase;  
    text-decoration: none;  
}
```

Et ce serait parfaitement acceptable ; il n'y a rien de *mal* à définir la taille de vos polices en pixels. Mais pour les besoins de notre petite expérience, commençons plutôt à penser de manière proportionnelle en exprimant ces valeurs en termes relatifs. Au lieu d'utiliser des pixels, nous utiliserons donc notre ami `em`.

## Questions de contexte

Pour cela, nous allons faire un peu de maths : nous allons simplement prendre la taille de police *cible* de notre maquette, et la diviser par la `font-size` de son contenant — en d'autres

termes, son *contexte*. Le *résultat* est la taille désirée de notre police exprimée avec notre unité relative super flexible, **em**.

En clair, les tailles de police relatives peuvent être calculées à l'aide de cette simple formule :

```
cible ÷ contexte = résultat
```

(Petite parenthèse : si vous êtes comme moi et que le mot « maths » provoque instantanément chez vous une crise d'angoisse, attendez un peu avant de vous mettre la tête dans un sac. Moi qui ai pris un cours de philo pour valider mes crédits de maths, j'utilise beaucoup la calculatrice de mon ordinateur. Il me suffit ensuite de coller le résultat dans ma feuille de styles. Ça m'évite de trop brusquer mes neurones.)

Avec notre formule en poche, revenons à notre titre en **24px**. En supposant que notre **font-size: 100%** de base dans l'élément **body** corresponde à **16px**, nous pouvons entrer directement ces valeurs dans notre formule. Ainsi, si nous voulons exprimer la taille de la police cible de notre **h1 (24px)** par rapport à son contexte (**16px**), nous obtenons :

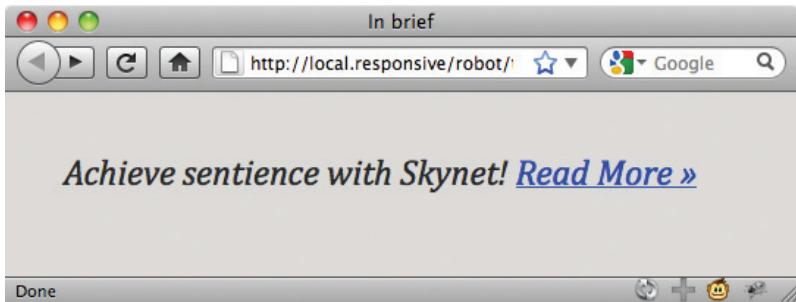
```
24 ÷ 16 = 1,5
```

Nous y voilà : **24px** est 1,5 fois plus grand que **16px**, donc notre **font-size** prendra la valeur **1.5em** :

```
h1 {  
    font-size: 1.5em /* 24px / 16px */;  
    font-style: italic;  
    font-weight: normal;  
}
```

Et hop ! La taille de notre titre correspond parfaitement à la taille spécifiée dans notre maquette, mais elle est exprimée en termes relatifs et flexibles (FIG 2.7).

(Je place généralement le calcul correspondant dans un commentaire à droite de la ligne (*/\* 24px / 16px \*/*), ce qui facilite grandement les modifications ultérieures.)



**FIG 2.7 :** Notre titre est parfaitement dimensionné à l'aide d'unités relatives flexibles.  
(Et ce fichu lien alors ?)

Maintenant que ce problème est tiré au clair, tournons-nous vers notre brave petit lien « Read More ». En fait, comme vous pouvez le voir à la FIGURE 2.7, il n'est pas *si* petit, et c'est bien là tout le problème. Dans notre maquette (FIG 2.4), il utilise une police de taille 11 sans empattement avec un crénage généreux : nous devons donc réduire la taille du texte, de beaucoup. Pour le moment, il hérite simplement de la règle `font-size: 1.5em` définie dans son élément contenant, le `h1`.

Et c'est un point important à noter. Comme le texte du titre prend la taille `1.5em`, tout élément placé à l'intérieur de ce titre doit être exprimé par rapport à cette valeur. En d'autres termes, *notre contexte a changé*.

Ainsi, pour définir la `font-size` de notre lien en unités relatives, nous allons diviser notre taille cible de `11px`, non pas par `16px`, la valeur définie dans le `body`, mais par `24px`, la taille de la police du titre, notre nouveau contexte :

$$11 \div 24 = 0,4583333333333333$$

Cette division nous donne le nombre le moins sexy qu'on ait vu jusqu'ici. (Oh, mais attendez un peu, le chapitre n'est pas fini.)

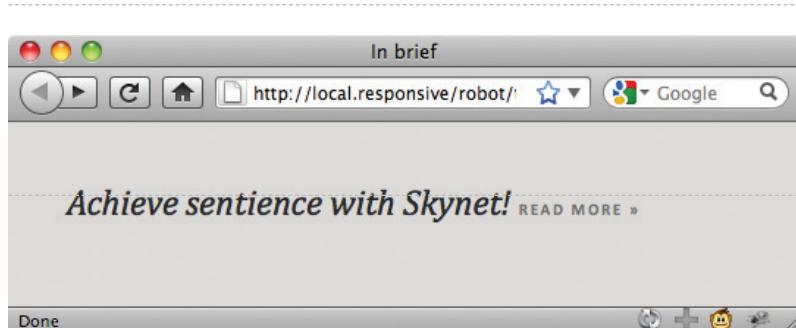
Vous allez peut-être être tentés d'arrondir `0.45833333333333em` au nombre raisonnable le plus proche, par exemple `0.46em`. Mais ne vous avisez pas d'y toucher !

Vos yeux saignent peut-être à la vue de ce nombre, mais **0.45833333333333** représente parfaitement la taille désirée de notre police en termes proportionnels. De plus, les navigateurs sont parfaitement capables d'arrondir ces décimales quand ils convertissent les valeurs en pixels. Donnez-leur le plus d'informations possible et vous obtiendrez un meilleur résultat.

Dans un souci d'exactitude, nous pouvons directement lâcher ce nombre au physique ingrat dans notre feuille de style (notez que » indique que la ligne se poursuit) :

```
h1 a {  
    font: bold 0.45833333333333em Calibri, Optima, »  
        Arial, sans-serif; /* 11px / 24px */  
    color: #747474;  
    letter-spacing: 0.15em;  
    text-transform: uppercase;  
    text-decoration: none;  
}
```

Le résultat ? Notre petite page est finie, correspond parfaitement au design voulu, et la taille du texte est définie en unités relatives flexibles et redimensionnables (FIG 2.8).



**FIG 2.8** : Et avec quelques calculs tout simples, notre composition est achevée, le tout sans un seul pixel en vue.

## Des polices flexibles à la grille flexible

À ce stade, il est probable que vous vous ennuyiez ferme. C'est vrai quoi, vous êtes en plein milieu d'un chapitre censé être consacré à la création de grilles de mise en page flexibles, et tout ce que ce type trouve à faire, c'est de parler *composition* et *mathématiques*. Quel culot.

Mais la première fois que j'ai eu à construire une grille flexible, je ne savais absolument pas par où commencer. Alors, j'ai fait ce que je fais à chaque fois que je suis confronté à un problème que je ne sais pas résoudre : je l'ai mis à la trappe et j'ai commencé à travailler sur autre chose.

Alors que je cherchais à dimensionner les polices de mon site en unités relatives, j'ai eu une sorte de révélation : j'ai compris que l'on pouvait appliquer le même genre de raisonnement proportionnel à la mise en page. En d'autres termes, chaque aspect de notre grille — les lignes et les colonnes, ainsi que les modules qui les recouvrent — peuvent être exprimés comme des *proportions* de leur contenant, au lieu d'utiliser des pixels immuables et inflexibles.

Et pour cela, on peut réutiliser notre fidèle formule `cible ÷ contexte = résultat`. Voyons ce que ça donne.

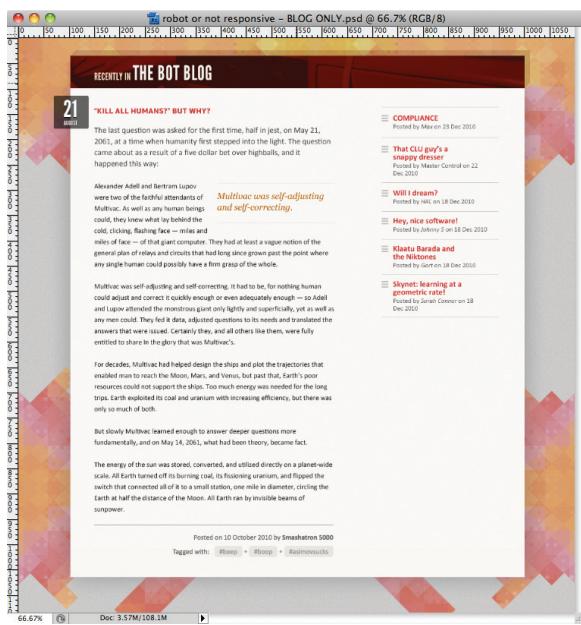
## CRÉER UNE GRILLE FLEXIBLE

Supposons que notre designer, impressionnée par notre réactivité, nous ait envoyé une nouvelle maquette. Nous sommes maintenant chargés de coder la section blog du site Web Robot or Not (FIG 2.9).

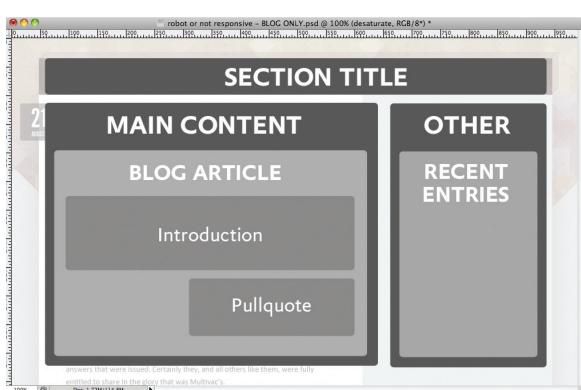
Il s'avère que notre designer nous aime tellement qu'elle a même inclus un bref inventaire du contenu de la page (FIG 2.10), ce qui nous épargnera du travail de préproduction. Pensez à lui apporter des cookies un de ces jours.

On peut facilement traduire sa maquette en une structure HTML de base comme ceci :

```
<div id="page">
  <div class="blog section">
    <h1 class="lede">Recently in <a href="#">The Bot
```



**FIG 2.9 :** Notre nouvelle mission : convertir ce design de blog en une mise en page flexible et standard.



**FIG 2.10 :** L'inventaire de contenu de notre module de blog.

```
Blog</a></h1>
<div class="main">
  ...
</div><!-- /end .main -->

<div class="other">
  ...
</div><!-- /end .other -->
</div><!-- /end .blog.section -->
</div><!-- /end #page -->
```

Notre ossature en HTML est sèche, musclée et riche sémantiquement, répondant parfaitement à notre inventaire de contenu global. Nous avons créé un contenant générique pour toute la page (`#page`), qui lui-même contient notre module `.blog`. Dans `.blog`, nous avons créé deux contenants supplémentaires : une balise `div` appelée `.main` pour le contenu principal de nos articles, et une autre appelée `.other` pour, eh bien, tout le reste. Ce n'est pas très poétique, mais ce n'est pas le but.

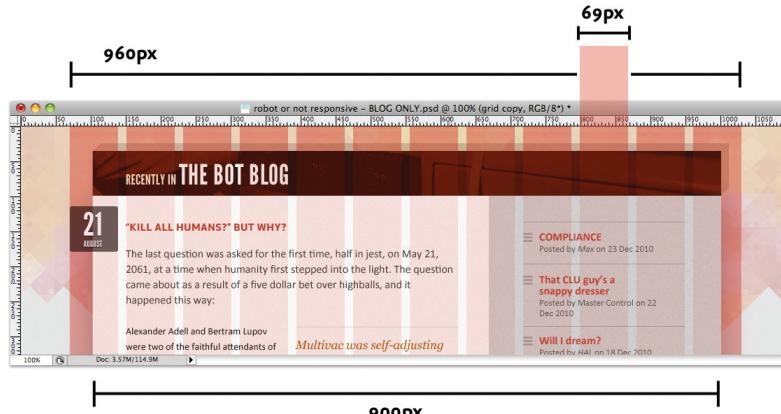
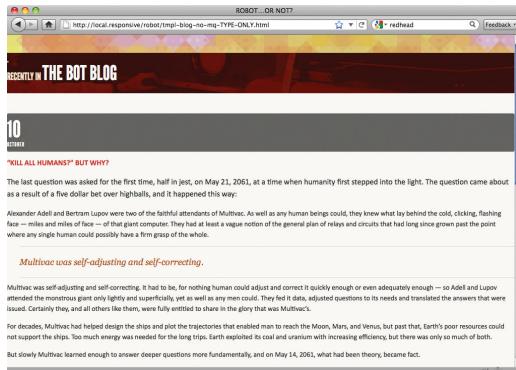
À ce stade, nous allons sauter plusieurs étapes de notre exercice. Nous allons faire comme dans ces émissions de télé-réalité où le chef balance une tonne d'ingrédients dans une marmite et en ressort une dinde aux marrons. (Cette métaphore témoigne de mon expérience limitée en matière d'émissions de cuisine. Et de cuisson de dinde.)

Mais admettons donc que nous nous soyons déjà occupés de la composition, des images d'arrière-plan, et de tous les éléments de notre design qui n'ont *pas* de lien avec la mise en page (FIG 2.11). Ces autres détails étant réglés, nous pouvons nous concentrer exclusivement sur la production de notre grille fluide.

Alors, *comment* fait-on exactement pour transformer ces blocs `.main` et `.other` en véritables colonnes ? Notre inventaire de contenu étant déjà fait et notre HTML de base en place, nous pouvons revenir à notre maquette et examiner de plus près les caractéristiques physiques de la grille (FIG 2.12).

En examinant le design, on remarque plusieurs choses : tout d'abord, la grille elle-même est divisée en 12 colonnes, espacées de 12 pixels et mesurant chacune 69 pixels de large.

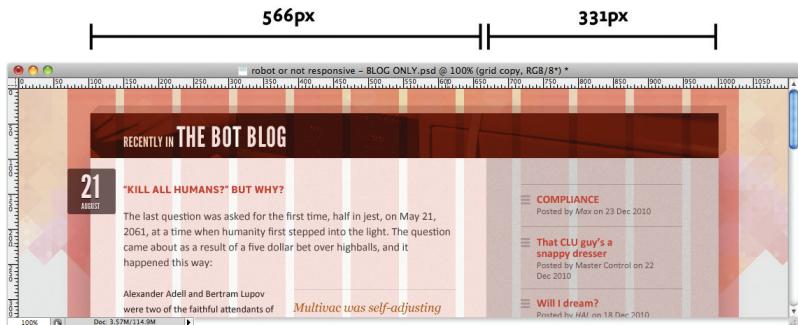
**FIG 2.11 :** Notre template est presque fini ! À l'exception peut-être d'un détail... la mise en page, quoi.



**FIG 2.12 :** Une grille de mise en page est basée sur une grille !

Au total, ces colonnes et l'espacement représentent une largeur de 960 pixels. Cependant, le blog lui-même ne fait que 900 pixels de large ; il est centré horizontalement sur notre toile de 960 pixels.

Voilà pour les grandes lignes. Si nous regardons de plus près les deux colonnes du blog (FIG 2.13), nous pouvons voir que celle de gauche (`.main`) fait 566 pixels de large, alors que celle de droite (`.other`) n'en prend que 331.



**FIG 2.13 :** Approchons-nous un peu et mesurons les colonnes internes.

Eh bien, ça fait quand même beaucoup de pixels jusqu'ici. Si les pixels nous convenaient, il nous suffirait de placer ces valeurs directement dans la feuille de styles (si vous avez l'impression que je vous balade, c'est normal).

```
#page {
    margin: 36px auto;
    width: 960px;
}

.blog {
    margin: 0 auto 53px;
    width: 900px;
}

.blog .main {
    float: left;
    width: 566px;
}

.blog .other {
    float: right;
    width: 331px;
}
```

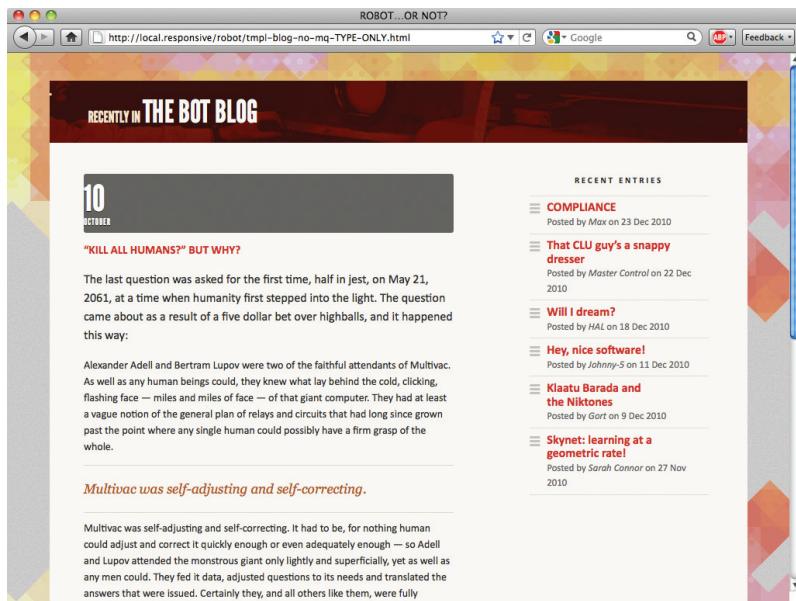


FIG 2.14 : Quelques pixels plus tard, notre design est presque fini. Mais l'est-il vraiment ?

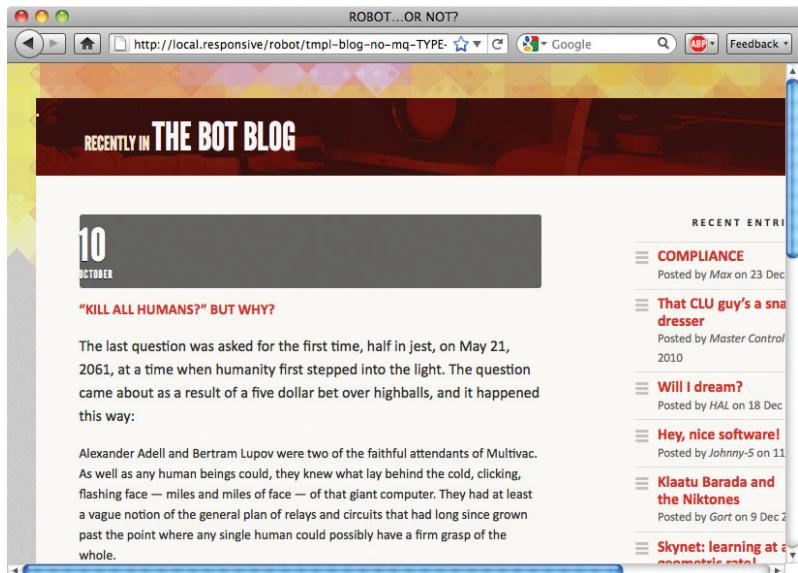
Nickel chrome : on a fixé la largeur de `#page` à 960 pixels, placé le module `.blog` de 900 pixels de large au centre de ce contenant, fixé la largeur de `.main` et de `.other` à respectivement `556px` et `331px`, puis on a fait flotter les deux colonnes face à face. Et le résultat est splendide (FIG 2.14).

Mais même si notre mise en page correspond parfaitement à la maquette, le résultat n'est absolument pas flexible. Avec sa largeur fixe de `960px`, notre page ignore royalement les dimensions du navigateur, et affichera invariablement une barre de défilement horizontale si la fenêtre passe à peine en-dessous des 1024 pixels (FIG 2.15).

En trois mots : peut mieux faire.

## Des pixels aux pourcentages

Au lieu de copier directement les valeurs en pixels de notre



**FIG 2.15 :** Notre mise en page est bien jolie, mais elle manque cruellement de souplesse. Il est temps de régler ça.

maquette dans notre feuille de styles, nous devons exprimer ces largeurs en termes relatifs et *proportionnels*. Ainsi, nous aurons une grille qui peut se redimensionner en même temps que la fenêtre du navigateur, sans compromettre les proportions originales du design.

Commençons par la couche externe qui contient notre design, l'élément `#page`, et avançons pas à pas :

```
#page {  
    margin: 36px auto;  
    width: 960px;  
}
```

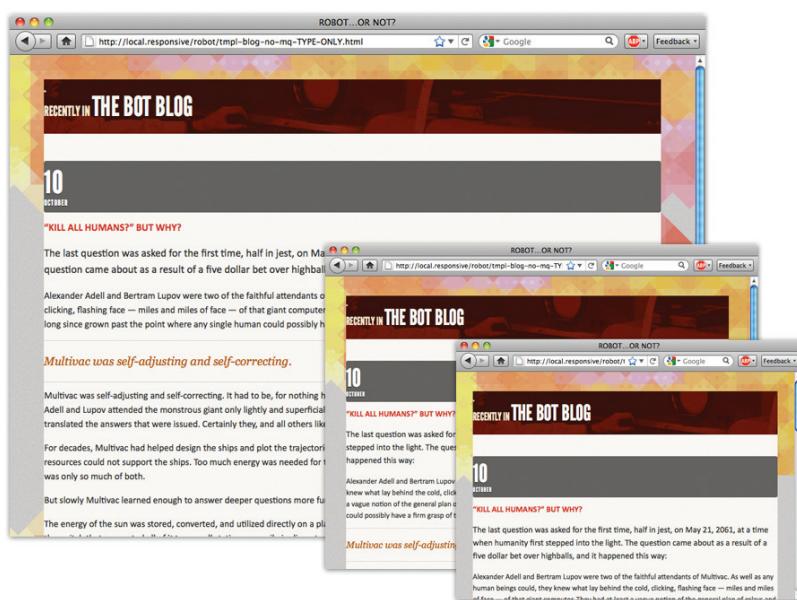
Méchants petits pixels, ils nous veulent du mal..

Oui, oui, j'exagère. Souvenez-vous, il n'y a absolument rien de mal à utiliser des mises en page à largeur fixe ! Mais comme

nous souhaitons développer une grille flexible, remplaçons ces **960px** par un pourcentage :

```
#page {  
    margin: 36px auto;  
    width: 90%;  
}
```

J'admet que j'ai choisi 90 % de manière un peu arbitraire, en tâtonnant dans mon navigateur pour voir ce qui rendait le mieux. En donnant à notre élément **#page** une largeur proportionnelle à la fenêtre du navigateur, nous avons créé un contenant qui pourra s'étirer et se rétracter en même temps que celle-ci (FIG 2.16). Et comme ce contenant est centré horizontalement sur la page, il nous restera une confortable marge de cinq pour cent de chaque côté.



**FIG 2.16** : Notre contenant s'étire avec la fenêtre du navigateur.

Jusqu'ici, tout va bien. Intéressons-nous ensuite au module `.blog`. Quand on jouait avec des pixels, on avait écrit la règle suivante :

```
.blog {  
    margin: 0 auto 53px;  
    width: 900px;  
}
```

Au lieu de donner une valeur en pixels, nous allons exprimer la largeur de 900 pixels de notre module en termes proportionnels : plus spécifiquement, le décrire comme un pourcentage de son élément contenant. C'est là que notre chère formule `cible ÷ contexte = résultat` revient à notre rescoufse.

Nous connaissons déjà la largeur cible de notre blog : `900px` d'après la maquette. Ce que nous voulons, c'est décrire cette largeur en termes relatifs, comme un pourcentage du contenu de `.blog`. Comme `.blog` est intégré dans l'élément `#page`, nous connaissons notre contexte, à savoir 960 pixels, la largeur définie dans la maquette.

Nous pouvons donc diviser la largeur cible de `.blog (900)` par celle de son contexte (`960`) :

$$900 \div 960 = 0,9375$$

Le résultat est donc 0,9375. J'admetts que ça n'a l'air de rien, comme ça. Mais en déplaçant la virgule de deux décimales, on obtient 93,75 %, un pourcentage que nous pouvons directement incorporer à notre feuille de styles :

```
.blog {  
    margin: 0 auto 53px;  
    width: 93.75%; /* 900px / 960px */  
}
```

(Comme je l'ai fait dans notre exercice de composition, j'ai laissé la formule dans un commentaire à la droite de la propriété `width`. C'est une préférence personnelle, bien sûr, mais ça m'a souvent été extrêmement utile.)

Voilà pour nos deux conteneants. Mais que fait-on de nos colonnes de contenu ?

```
.blog .main {  
    float: left;  
    width: 566px;  
}  
  
.blog .other {  
    float: right;  
    width: 331px;  
}
```

Notre contenu principal flotte à gauche, et sa largeur est fixée à **566px** ; le contenu secondaire flotte de l'autre côté, occupant une largeur de **331px**. Cette fois encore, nous allons remplacer ces largeurs en pixels par des pourcentages.

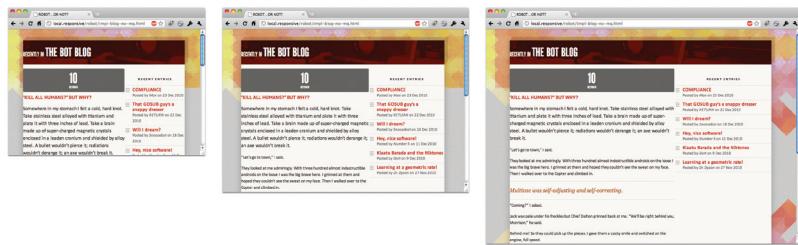
Mais avant d'entrer ces valeurs dans notre formule, notons que notre contexte a changé. La fois précédente, nous avions divisé la largeur de notre module de blog par **960px**, la largeur de son contenant (**#page**). Mais comme nos colonnes sont intégrées dans **.blog**, nous devons exprimer leur largeur par rapport à **900px**, la largeur du blog.

Nous allons donc diviser nos deux valeurs cible (**566px** et **331px**) par **900px**, notre nouveau contexte :

$$566 \div 900 = 0,628888889$$
$$331 \div 900 = 0,367777778$$

Après avoir déplacé la virgule, il nous reste 62,8888889 % et 36,7777778 %, les largeurs proportionnelles de **.main** et de **.other** :

```
.blog .main {  
    float: left;  
    width: 62.8888889%; /* 566px / 900px */  
}  
  
.blog .other {
```



**FIG 2.17 :** Notre grille flexible est finie.

```
float: right;
width: 36.777778%; /* 331px / 900px */
}
```

Sans plus d'efforts, nous venons de créer une grille de mise en page flexible (**FIG 2.17**).

Avec quelques calculs simples, nous avons créé un contenant et deux colonnes flexibles et proportionnels, définis à l'aide de pourcentages, donc une mise en page qui se redimensionne de concert avec le navigateur. La largeur en pixels de ces colonnes pourra varier, mais les proportions de notre design resteront intactes.

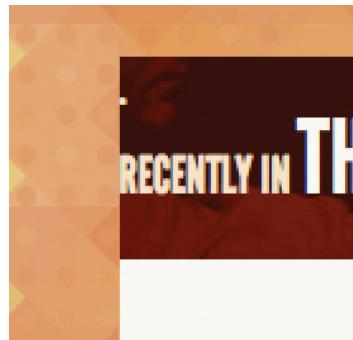
## MARGES ET ESPACEMENT FLEXIBLES

Maintenant que ces colonnes sont en place, nous en avons fini avec les composants principaux de notre grille flexible. Merveilleux. Fantastique. Prodigieux, même. Mais avant d'employer plus de qualificatifs, il reste pas mal de détails à régler.

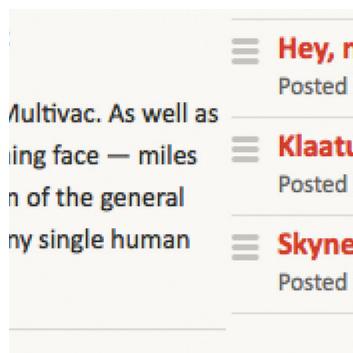
### De l'air

Pour commencer, notre design est peut-être flexible, mais il souffre d'un manque flagrant de finitions. Les deux pires contrevenants ? Le titre de notre blog, qui est relégué tout à gauche de son contenant (**FIG 2.18**), et nos deux colonnes qui se chevauchent, sans marge ni espace en vue (**FIG 2.19**). Il faut vraiment faire un coup de ménage.

**FIG 2.18 :** Notre titre a vraiment besoin d'air.



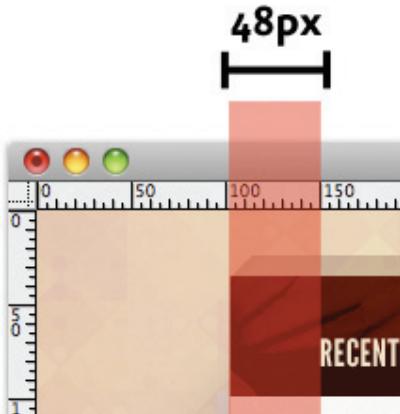
**FIG 2.19 :** Des marges ? On s'en passe très bien ! (Quoique, peut-être pas finalement.)



Commençons donc par ce titre. Dans notre maquette, il y a un espace de 48 pixels entre notre titre et le bord gauche de son contenant (FIG 2.20). Bien sûr, nous pourrions définir une propriété `padding-left` fixe, en pixels ou en em, comme ceci :

```
.lede {  
  padding: 0.8em 48px;  
}
```

Ce serait une solution décente. Mais une valeur fixe donne un espacement qui ne colle pas avec le reste de notre grille fluide. À mesure que nos colonnes flexibles s'étirent ou se rétractent, l'espacement ignore les proportions de notre design et reste obstinément fixé à `48px`.



**FIG 2.20 :** D'après le design, il faut une marge horizontale de 48px à gauche de notre titre.

Nous allons donc plutôt créer un espace flexible. Jusqu'ici, nous avons décrit les largeurs de divers éléments en termes proportionnels. Mais nous pouvons également créer des marges et des espacements basés sur des pourcentages pour préserver l'intégrité de notre grille flexible. Et nous pouvons réutiliser notre formule `cible ÷ contexte = résultat` pour ce faire.

Mais avant de revenir à nos mathématiques, il est important de noter que le contexte est légèrement différent pour les marges et l'espacement :

1. Si vous souhaitez créer une **marge** flexible sur un élément, le contexte est la largeur de son contenant.
2. Si vous souhaitez créer un **espacement** flexible sur un élément, le contexte est la largeur de l'élément lui-même. Ce qui est logique si l'on songe au modèle des boîtes en CSS : on décrit l'espacement par rapport à la largeur de la boîte elle-même.

Comme nous voulons un léger espace sur notre titre, notre contexte est la largeur de notre titre `.lede`. Puisque le titre n'a pas de largeur déclarée, sa largeur (et le contexte dont nous avons besoin pour notre formule) est la largeur du module blog, soit `900px`. Sortez la calculatrice et vous obtiendrez :

$$48 \div 900 = 0,0533333333$$

Ce qui se traduit par :

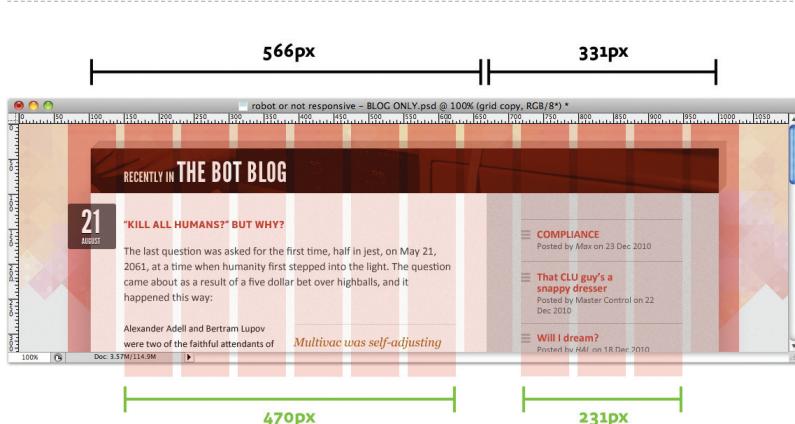
```
.1ede {  
    padding: 0.8em 5.3333333%; /* 48px / 900px */  
}
```

Et voilà : notre espace de **48px** est exprimé en termes relatifs, proportionnellement à la largeur de notre titre.

Ce problème étant réglé, ajoutons un peu d'espace dans notre contenu compact. Pour cela, il est bon de se rappeler que chaque colonne contient en fait un module plus petit : à gauche, la colonne **.blog** contient un module **.article**, tandis que la colonne **.other** contient notre liste **.recent-entries** (FIG 2.21).

Commençons par le module « recent entries ». Heureusement pour nous, il n'y a pas grand-chose à faire. Comme on connaît la largeur de l'élément (**231px**) et la largeur de la colonne qui le contient (**331px**), on peut facilement centrer notre module horizontalement :

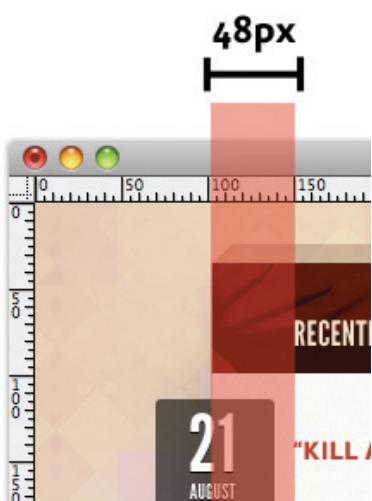
```
.recent-entries {  
    margin: 0 auto;
```



**FIG 2.21 :** En jetant un œil sur la maquette, on peut rapidement mesurer la largeur respective de chaque module.

```
width: 69.7885196%; /* 231px / 331px */  
}
```

Nous pourrions utiliser la même approche avec notre article, mais nous allons faire quelque chose de plus intéressant. Vous vous souvenez de l'espacement de `48px` qu'on a ajouté à gauche de notre titre ? Eh bien notre article tombe dans la même colonne (FIG 2.22). Alors plutôt que de nous contenter de centrer notre article dans son contenant, nous allons créer une autre marge proportionnelle.



**FIG 2.22 :** Notre titre et notre article présentent le même espacement.

Notre valeur cible est `48px`. Puisque nous travaillons avec un espacement relatif, notre contexte devrait être la largeur de l'article lui-même. Mais une fois encore, comme il n'y a pas de largeur explicitement définie pour `.article`, nous pouvons simplement utiliser `566px`, la largeur du contenant supérieur (`.blog`), comme contexte :

```
.article {  
padding: 40px 8.48056537%; /* 48px / 566px */  
}
```

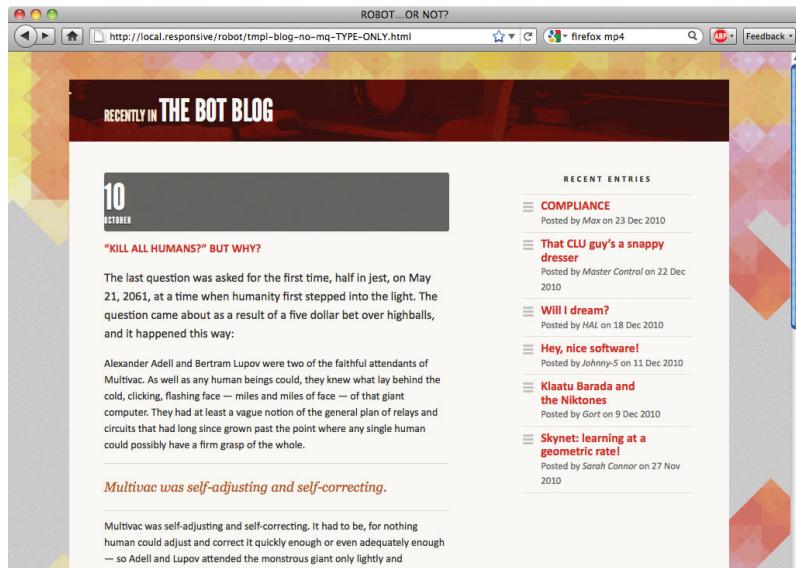


FIG 2.23 : Hourra ! Un espacement et des marges flexibles !

Et voilà ! Notre grille flexible est pratiquement finie (FIG 2.23).

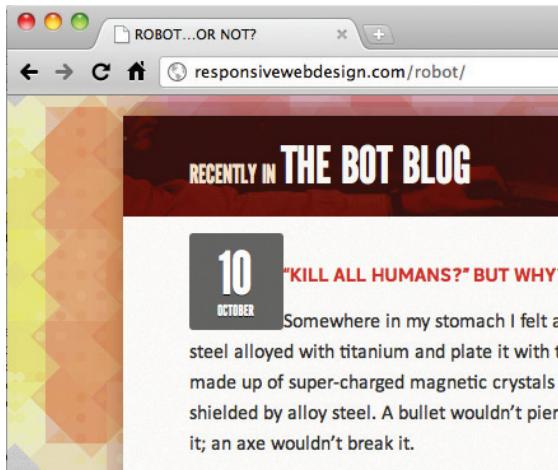
## Marge négative

Intéressons-nous maintenant à l'affichage de la date de nos articles, laissé pour compte jusqu'ici.

Pour le moment, la date prend toute la largeur du billet, et ce n'est pas terrible. Avec tout ce que l'on a appris jusqu'à présent, il devrait être plutôt simple de régler sa largeur : la maquette nous dit que la date doit flotter à gauche, et occuper une colonne de 69px (cf. FIG 2.12). Comme la date est intégrée à notre module article de 474px de large, nous avons notre contexte.

Armés de ces informations, entrons ces quelques lignes de CSS :

```
.date {  
    float: left;
```



**FIG 2.24 :** Il y a quelque chose de pourri au royaume du Danemark. (Par « Danemark », je veux dire « la date de notre article », et par « pourri », je veux dire « beaucoup trop près du texte environnant ».)

```
width: 14.556962%; /* 69px / 474px */  
}
```

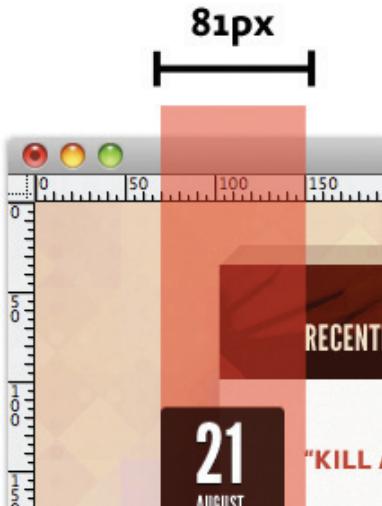
Jusqu'ici, tout va bien. Mais il manque un composant essentiel : notre date flotte contre le bord gauche de l'article et n'est pas assez espacée du texte environnant (FIG 2.24). Nous devons donc la déplacer au-delà du bord gauche du module.

C'est précisément ce que nous permettent de faire les marges négatives. Pas besoin de changer d'approche : comme précédemment, il nous faut simplement exprimer cette marge par rapport à la largeur de l'élément contenant.

Si nous regardons la maquette, nous pouvons voir qu'il y a 81 pixels entre le bord gauche de la date et le bord gauche de l'article (FIG 2.25). Si nous étions en train de créer un design fixe, cela donnerait :

```
.date {  
    float: left;  
    margin-left: -81px;  
    width: 69px;  
}
```

**FIG 2.25 :** Nous devons déplacer cette date de 81px vers la gauche. Enfin, vous avez compris, de l'équivalent en pourcentage.



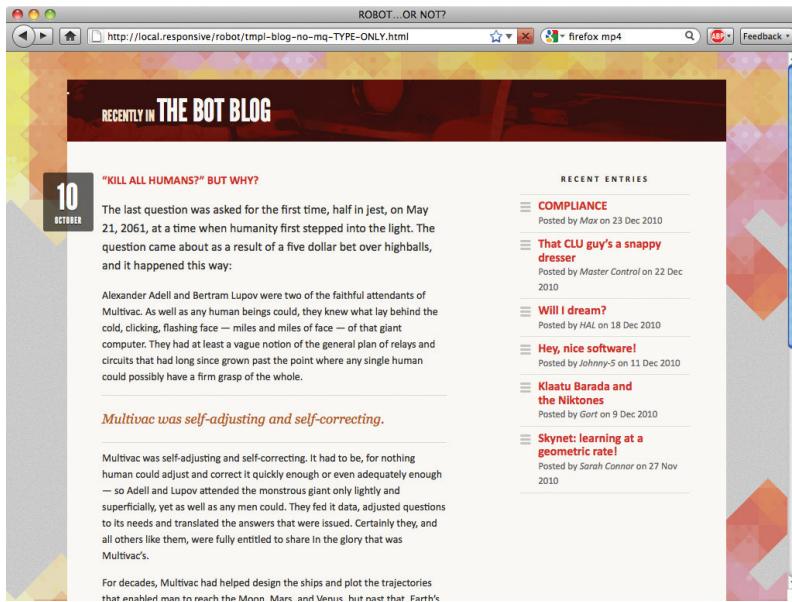
Mais bon, on n'a pas encore utilisé un seul pixel, on ne va pas commencer maintenant. Nous allons plutôt exprimer cette marge en termes relatifs, comme nous l'avons fait auparavant. Ce sera une marge négative, mais ça ne change rien au calcul. Nous voulons toujours exprimer notre valeur cible, cette marge de **81px**, comme un pourcentage de **474px**, la largeur de l'élément contenant la date :

$$84 \div 474 = 0,170886076$$

Déplacez la virgule, rajoutez le signe moins et vous avez votre marge négative proportionnelle :

```
.date {  
    float: left;  
    margin-left: -17.0886076%; /* 81px / 474px */  
    width: 14.556962%; /* 69px / 474px */  
}
```

Vous pouvez maintenant vous relaxer, respirer un grand coup et admirer votre première grille flexible pleinement fonctionnelle (FIG 2.26). Topez-là.



**FIG 2.26 :** Notre grille flexible est enfin finie. Pas un seul pixel en vue, et on n'a pas lésiné sur l'esthétique.

## La suite, tout en flexibilité

Je me rends compte que je vous ai infligé pas mal de calculs. Moi qui arrive à peine à tenir mon budget, croyez-moi : je compatis.

Mais construire une grille flexible n'est pas qu'une histoire de maths. Bien sûr, la formule **cible ÷ contexte = résultat** permet de traduire facilement ces proportions en pourcentages prêts à être intégrés dans nos feuilles de style. Mais ce qui compte, c'est que nous perdions l'habitude de transposer directement nos pixels depuis Photoshop pour nous focaliser sur les proportions de nos designs. Il s'agit de faire attention au contexte : de mieux comprendre les relations de proportionnalité entre un élément et son contenant.

Mais la grille fluide n'est que la base, la première couche d'un responsive design. Passons à l'étape suivante.



# LES IMAGES FLEXIBLES

JUSQU'ICI TOUT SE PRÉSENTE BIEN : nous avons une grille de mise en page qui ne renonce pas à la complexité au profit de la flexibilité. Je dois admettre que la fois où j'ai enfin compris comment construire une grille fluide, j'étais plutôt fier de moi.

Mais alors, comme souvent dans le design Web, le désespoir s'est emparé de moi. Pour l'instant, notre page contient beaucoup de mots et pas grand-chose d'autre. En fait, rien d'autre : il n'y a que du texte sur notre page. Pourquoi est-ce un problème ? Eh bien déjà, le texte s'adapte sans problème à un contenant flexible. Ensuite, je ne sais pas si vous avez remarqué, mais on trouve parfois sur Internet deux ou trois trucs que l'on appelle « images ». Et il n'y en a guère dans notre grille fluide.

Mais alors, que se passe-t-il quand on introduit des images à largeur fixe dans notre design flexible ?

## RETOUR AUX (CODES) SOURCES

Pour avoir la réponse, faisons une nouvelle petite expérience : déposons une image directement dans notre module de blog, et

observons comment la mise en page se comporte. Il nous faut tout d'abord lui faire une petite place dans notre code source.

Vous vous rappelez de notre petit **blockquote**, confortablement installé dans notre article ? Il y a beaucoup trop de texte sur cette fichue page, alors remplaçons-le par une image :

```
<div class="figure">
  <p>
    
    <b class="figcaption">Lo, the robot walks</b>
  </p>
</div>
```

Rien de bien folichon : un élément **img**, suivi d'une légende brève mais descriptive insérée dans un élément **b**. Dans ce bout de code, je m'approprie en fait les balises **figure/figcaption** de HTML5 pour en faire des noms de classe et avoir des fondations sémantiques solide.

(Mes lecteurs avisés remarqueront que j'utilise un élément **b** comme support non sémantique. Certains designers utiliseront peut-être un élément **span** à la place. Personnellement, j'aime le dépouillement des balises plus courtes comme **b** ou **i** pour le balisage non sémantique.)

Mon HTML en place, j'ajoute quelques règles CSS de base :

```
.figure {
  float: right;
  margin-bottom: 0.5em;
  margin-left: 2.53164557%; /* 12px / 474px */
  width: 48.7341772%; /* 231px / 474px */
}
```

Nous créons ainsi un bel effet d'incrustation. Notre image flottera à droite et occupera environ la moitié de la largeur de notre article, soit quatre colonnes de notre grille flexible. Balisage : check ; style : check. Bien sûr, tout ce HTML et cette CSS ne sert à rien si l'on n'a pas d'image à proprement parler.

Et comme je vous aime autant que j'aime les robots, je ne pouvais pas me contenter de n'importe quelle image. Ainsi, après

**FIG 3.1 :** Un robot tout ce qu'il y a de plus robotique, avec l'aimable autorisation de Jeremy Noble (<http://bkaprt.com/rwd/10/>).



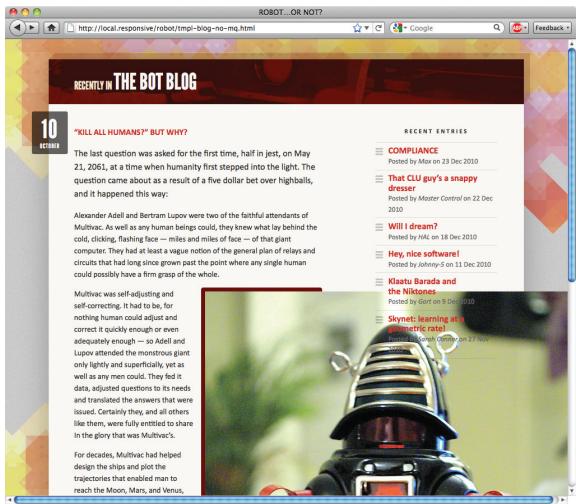
avoir parcouru le Web pendant des minutes entières, j'ai trouvé ce roboportrait absolument grandiose (FIG 3.1). Le bon côté de cette image (outre le robot, évidemment), c'est qu'elle est grande. Je l'ai légèrement recadrée, mais je ne l'ai pas réduite, gardant sa résolution d'origine de 655×655. Cette image est beaucoup plus grande que son contenant, ce qui nous donne une occasion parfaite de tester la robustesse de notre mise en page flexible.

Alors déposons notre image trop grande sur le serveur, rechargeons la page, et... ah. D'accord. C'est à peu près ce qu'on pouvait espérer de pire (FIG 3.2).

En fait, ce résultat n'est pas si surprenant. Notre mise en page n'est pas vraiment cassée — notre contenant flexible fait son office, et les proportions des colonnes de notre grille restent intactes. Mais comme l'image est beaucoup plus large que son contenant `.figure`, le contenu en excès dépasse tout simplement du contenant et reste visible pour l'utilisateur. Aucune contrainte appliquée à notre image ne la force à s'adapter à notre environnement flexible.

## IMAGES FLUIDES

Et si nous pouvions introduire une telle contrainte ? Si nous pouvions écrire une règle qui empêche les images de dépasser la largeur de leur contenant ?



**FIG 3.2 :** Notre grosse image est grosse. Notre mise en page cassée est cassée.

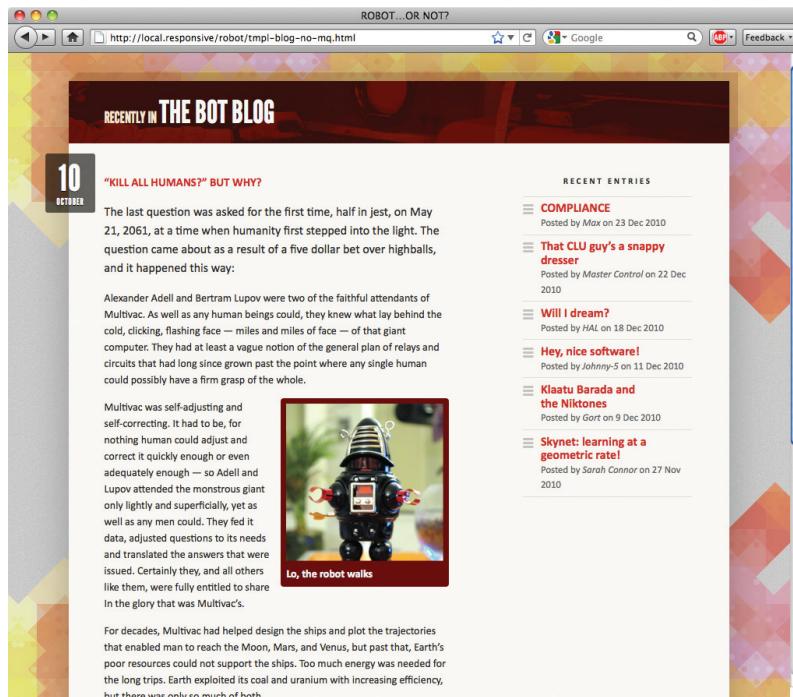
La bonne nouvelle, c'est ce que c'est très facile à faire :

```
img {
    max-width: 100%;
}
```

Découverte par le designer Richard Rutter (<http://bkapr.com/rwd/11/>), cette seule règle fournit une contrainte très utile à toutes les images de notre document. Maintenant, notre élément `img` sera rendu à la taille qui lui convient, du moment que celle-ci est inférieure à la largeur de son contenant. Mais si l'image est plus large que son contenant, alors la directive `max-width: 100%` forcera l'image à s'ajuster. Et comme vous le constatez, notre image a retrouvé sa place (FIG 3.3).

De plus, les navigateurs modernes sont tellement évolués qu'ils redimensionnent les images proportionnellement : en jouant avec la fenêtre du navigateur, on constate que le rapport hauteur/largeur reste intact (FIG 3.4).

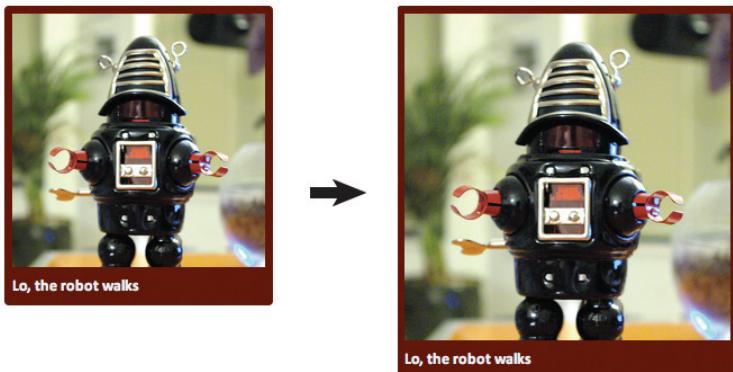
J'espère que vous aimez les bonnes nouvelles, car il se trouve que la règle `max-width: 100%` peut également s'appliquer à la plupart des éléments à largeur fixe, comme les vidéos et autres



**FIG 3.3 :** En incluant simplement la règle `max-width: 100%`, nous avons empêché notre image d'échapper à son contenant flexible. Au fait, j'adore `max-width: 100%`.

médias riches. En fait, on peut très simplement étendre notre sélecteur pour intégrer d'autres médias, comme ceci :

```
img,
embed,
object,
video {
    max-width: 100%;
}
```



**FIG 3.4 :** Quelle que soit la taille de son contenant flexible, l'image se redimensionne proportionnellement. Magie ? Qui sait.

---

Que ce soit une jolie vidéo Flash (FIG 3.5), tout autre média intégré, ou juste une modeste `img`, la plupart des navigateurs parviennent à redimensionner le contenu proportionnellement dans une mise en page flexible. Tout ça grâce à notre contrainte poids plume, `max-width`.

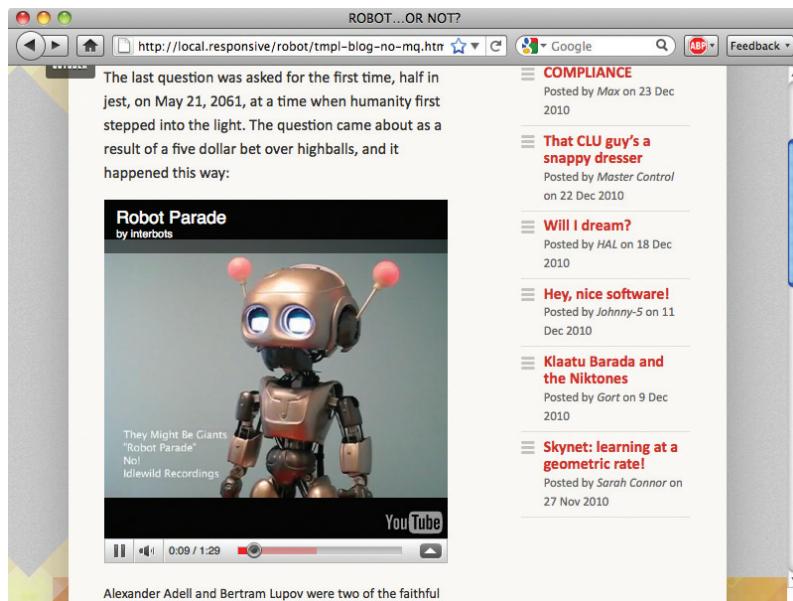
Alors ça y est, on a résolu le problème des images et des médias flexibles ? Une règle CSS et c'est bouclé ?

## PARCE QUE CE BOULOT N'EST PAS DE TOUT REPOS

Il est temps de panser quelques plaies : nous allons affronter la douleur, les pleurs et les cheveux arrachés, et parler de quelques problèmes spécifiques à certains navigateurs concernant les images flexibles.

### **max-width dans Internet Explorer**

La cruelle vérité, c'est qu'Internet Explorer 6 et inférieur ne supportent pas la propriété `max-width`. IE7 et supérieur ? Pas



**FIG 3.5 :** Les autres médias s'adaptent parfaitement à notre règle `max-width: 100%` et deviennent eux-mêmes flexibles. Je vous ai dit que j'adorais `max-width: 100%` ?

de problème. Mais si vous voulez que votre site soit compatible avec le vénérable (*tousse*) IE6 et ses prédecesseurs, notre approche doit être affinée.

Il existe plusieurs méthodes documentées pour faire marcher la propriété `max-width` sous IE6. La plupart utilisent JavaScript, généralement avec le filtre propriétaire `expression` de Microsoft, pour évaluer dynamiquement la largeur d'un élément et le redimensionner manuellement si elle excède un certain seuil. Pour avoir un exemple de ce bidouillage pas très standard, je vous recommande le billet incontournable de Svend Tofte sur le sujet (<http://bkaprt.com/rwd/12/>).

Moi ? J'ai plutôt tendance à favoriser un système D utilisant la CSS. En l'occurrence, tous les navigateurs modernes reçoivent notre contrainte `max-width` :

```
img,  
embed,  
object,  
video {  
    max-width: 100%;  
}
```

Mais dans une feuille de styles séparée, j'inclus la règle suivante pour IE6 :

```
img,  
embed,  
object,  
video {  
    width: 100%;  
}
```

Vous voyez la différence ? IE6 et inférieur reçoivent la règle `width: 100%` au lieu de `max-width: 100%`.

Avertissement : à utiliser avec précaution, car ce sont deux règles radicalement différentes. Alors que `max-width: 100%` demande à nos images de ne jamais excéder la largeur de leur contenant, `width: 100%` force nos images à toujours s'adapter à la largeur de leur contenant.

En général, cette méthode fonctionne bien. Par exemple, on peut supposer sans crainte que notre image `robot.jpg` surdimensionnée sera toujours plus grande que son contenant, et la règle `width: 100%` conviendra parfaitement.

Mais dans le cas d'images plus petites, des miniatures par exemple, ou de la plupart des vidéos, les agrandir aveuglément avec notre règle CSS peut poser des problèmes. Dans ce cas, on peut fournir à IE une règle plus spécifique :

```
img.full,  
object.full,  
.main img,  
.main object {  
    width: 100%;  
}
```

Si vous ne voulez pas voir la règle `width: 100%` s'appliquer à tous les médias à largeur fixe de votre page, vous pouvez simplement définir une liste de sélecteurs qui ciblent certains types d'images ou de vidéos (`img.full`), ou certaines zones de votre document où vous savez que vous aurez affaire à des médias trop larges (`.main img, .main object`). Voyez cela comme une liste blanche : si des images ou d'autres médias apparaissent dans cette liste, alors ils seront flexibles ; sinon, ils resteront fixés sur leurs vieux pixels moisis.

Alors si vous supportez encore les versions antédiluviennes d'Internet Explorer, une règle `width: 100%` bien appliquée peut faire marcher ces images flexibles en toute beauté. Mais maintenant que ce bug est réglé, il nous en reste un autre.

Et pas des moindres...

## Où il devient clair que Windows nous déteste

Si vous regardez notre blog dans certains navigateurs fonctionnant sous Windows, notre `robot.jpg` est passé de son statut imposant au statut de robot cassé (FIG 3.6). Mais ce n'est pas tant un problème lié au navigateur qu'à la plateforme : Windows a du mal à redimensionner des images. En effet, sur cette plateforme, des artefacts apparaissent rapidement dans les images redimensionnées, ce qui a des conséquences sur leur qualité. Des conséquences négatives, bien sûr.

Pour l'exemple, j'ai jeté une image pleine de texte dans un contenant flexible, puis j'ai redimensionné notre image avec la règle `max-width: 100%`, tandis que IE6 et inférieur utilisent notre règle alternative, `width: 100%`. Personne n'a besoin de mettre autant de texte dans une image, mais cela illustre parfaitement la mauvaise tournure que peuvent prendre les choses sous IE7 et inférieur. Comme vous pouvez le constater, l'image a un rendu — passez-moi le terme technique — tout simplement dégueulasse (FIG 3.7).

Mais avant que vous n'abandonniez la promesse d'une image flexible et redimensionnable, il convient de remarquer que ce bug n'affecte pas tous les navigateurs tournant sous Windows. En fait, seul IE7 et inférieur sont affectés, tout comme Firefox 2 et inférieur. Les navigateurs plus modernes, comme Safari,



**FIG 3.6 :** Sous IE6, notre image de robot présente quelques artefacts disgracieux. Apparemment, Windows se fiche pas mal de nos images flexibles.

A screenshot of a Microsoft Internet Explorer window. The title bar says "Respin' My Images, Boss.". The main content area contains several lines of text that are severely distorted by multiple overlapping artifacts. The text is illegible but includes words like "Fluid is the new black.", "ON FLU", and "Now if you'll excuse me, I need to go pop a few dozen D".

**FIG 3.7 :** Dans certains navigateurs tournant sous Windows, les artefacts se multiplient rapidement, rendant le texte illisible.

Firefox 3 et plus et IE8 et plus, ne rencontrent pas le moindre problème avec les images flexibles. De plus, le bug semble avoir été réparé dans Windows 7 ; encore une bonne nouvelle.

Maintenant que nous avons évalué l'étendue des dégâts, il doit bien exister un correctif à appliquer, non ? C'est heureusement le cas, sauf pour Firefox 2.

Évidemment, ce vieux navigateur grisonnant étant sorti en 2006, on peut raisonnablement supposer qu'il n'engorge pas vos logs. Quoi qu'il en soit, un correctif pour Firefox 2 nécessiterait une méthode de détection du navigateur permettant de cibler des versions spécifiques sur une plateforme spécifique,

méthode peu fiable dans le meilleur des cas. Mais même si nous voulions utiliser cette méthode, ces vieilles versions de Firefox n'ont pas de bouton pour réparer nos images tordues.

En revanche, Internet Explorer possède un tel bouton. (Excusez-moi le temps que je ravale ma fierté pour le prochain titre de section.)

## Vive AlphaImageLoader, notre vaillant héros

Vous avez déjà essayé de faire marcher des PNG transparents sous IE6 et inférieur ? Il est fort probable que vous ayez rencontré [AlphaImageLoader](http://bkaprt.com/rwd/13/), l'un des filtres CSS propriétaires de Microsoft (<http://bkaprt.com/rwd/13/>). Depuis lors, des correctifs plus robustes ont été créés pour pallier les défaillances de la prise en charge d'IE (la librairie DD\_belatedPNG de Drew Diller est mon favori à l'heure actuelle : <http://bkaprt.com/rwd/14/>), mais à l'époque, pour définir un PNG comme arrière-plan d'un élément, vous pouviez ajouter la règle suivante dans une feuille de styles spécifique à IE :

```
.logo {  
background: none;  
filter: progid:DXImageTransform.Microsoft. »  
AlphaImageLoader(src="/path/to/logo.png", »  
sizingMethod="scale");  
}
```

Ce correctif fait plusieurs choses. D'abord, il retire l'image d'arrière-plan de l'élément, puis l'insère dans un objet [AlphaImageLoader](#) qui est placé « entre » la couche d'arrière-plan et le contenu de l'élément. Mais c'est la propriété [sizing-Method](#) (<http://bkaprt.com/rwd/15/>) qui fait le sale boulot, dictant dans quels cas l'objet [AlphaImageLoader](#) doit couper une image qui dépasse son contenant, la traiter comme une image normale ou la redimensionner à la taille de son contenant.

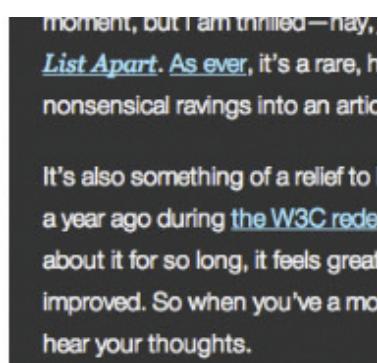
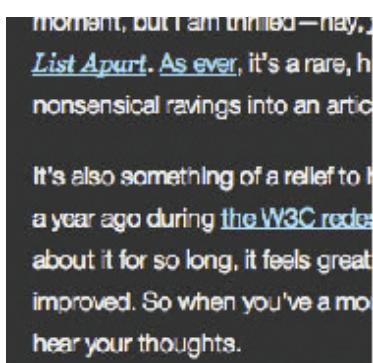
Je vous entendez étouffer vos bâillements d'ici : après tout, qu'est-ce qu'un correctif spécifique à IE a à voir avec le rendu de notre image cassée ?

Bien des choses, en fait. À un moment, j'ai découvert qu'en appliquant `AlphaImageLoader` à une image, son rendu s'en trouvait radicalement amélioré sous IE et n'avait plus rien à envier à tous les autres navigateurs. De plus, en donnant à la propriété `sizingMethod` la valeur `scale`, nous pouvons utiliser notre objet `AlphaImageLoader` pour créer une pseudo-image flexible.

J'ai donc rapidement bidouillé un peu de JavaScript pour automatiser ce processus. Téléchargez simplement le script (disponible à l'adresse <http://bkaprt.com/rwd/16/>) et incorporez-le dans n'importe quelle page contenant des images flexibles ; il parcourra votre document pour créer une série d'objets `AlphaImageLoader` flexibles et de haute qualité.

Une fois ce correctif appliqué, la différence de rendu entre nos images est manifeste (FIG 3.8) : dans notre exemple, nous sommes passés d'une image excessivement déformée à une image au rendu irréprochable. Et en plus, ça marche parfaitement dans un contexte flexible.

(Notons tout de même que de nombreux filtres propriétaires Microsoft, et `AlphaImageLoader` en particulier, affectent les performances — Stoyan Stefanov couvre ces écueils plus en détail sur le blog YUI : <http://bkaprt.com/rwd/17/>. Quelles sont les implications pour vous ? Assurez-vous juste de tester le correctif minutieusement sur votre site, jaugez l'impact sur vos



**FIG 3.8 :** Notre image est maintenant parfaitement lisible et se redimensionne sans problème. Un petit coup d'`AlphaImageLoader` pour la route ?

utilisateurs et décidez si l'amélioration du rendu vaut la perte de performance.)

Notre règle `max-width: 100%` étant en place (avec l'aide de nos correctifs `width: 100%` et `AlphaImageLoader`), notre image se redimensionne parfaitement sur nos navigateurs cibles. Quelle que soit la taille de la fenêtre du navigateur, notre image s'adapte harmonieusement aux proportions de notre grille flexible.

Mais que fait-on des images d'arrière-plan ?

## MOSAÏQUE D'ARRIÈRE-PLAN FLEXIBLE

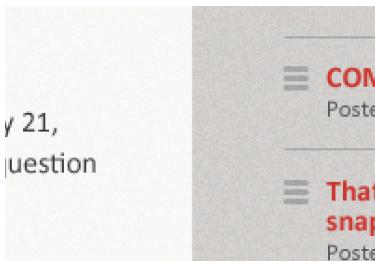
Supposons que notre chère designer nous ait envoyé une copie révisée de la maquette de notre module de blog. Vous remarquez la différence ? (FIG 3.9)



FIG 3.9 : Notre blog a mis son plus bel arrière-plan. Sexy.

Jusqu'à présent, le contenu de notre blog reposait sur un arrière-plan modeste, pratiquement blanc. Mais le design a été légèrement modifié : on a appliqué deux couleurs différentes aux deux colonnes de notre blog pour renforcer le contraste. De plus, un grain a été ajouté à l'arrière-plan, qui donne une texture supplémentaire à notre design (FIG 3.10).

Alors, comment ajouter cette nouvelle image d'arrière-plan à notre template ?



**FIG 3.10 :** Gros plan sur notre nouvel arrière-plan.

En 2004, Dan Cederholm a écrit un excellent article démontrant comment une image d'arrière-plan répétée verticalement pouvait être utilisée pour créer une sorte de « fausse » colonne (<http://bkaprt.com/rwd/18/>). Le génie de cette technique réside dans sa simplicité : en répétant une image d'arrière-plan colorée à la verticale, on peut créer l'illusion de colonnes de hauteurs égales.

Dans la technique originale de Dan, l'image d'arrière-plan était simplement centrée au sommet du contenu et répétée à la verticale, comme ceci :

```
.blog {  
background: #F8F5F2 url("blog-bg.png") repeat-y 50% 0;  
}
```

Et cette technique marche admirablement. Mais elle suppose que votre design ait une largeur fixe — on crée alors une image de la même dimension. Mais alors, comment allons-nous parvenir à intégrer une image d'arrière-plan qui se répète sur deux colonnes flexibles ?

Grâce au travail du designer Doug Bowman (<http://bkaprt.com/rwd/19/>), nous pouvons tout de même appliquer cette technique. Cela requiert juste quelques efforts supplémentaires, et un coup de main de notre formule favorite, **cible ÷ contexte = résultat**.

Commençons par examiner notre maquette pour trouver le point de transition de notre image d'arrière-plan, le pixel exact où l'on passe de la colonne blanche à la colonne grise.



**FIG 3.11 :** La démarcation entre la colonne blanche et la colonne grise se situe au niveau du 568<sup>e</sup> pixel. C'est notre point de transition.

Apparemment, ce changement se produit au niveau du 568<sup>e</sup> pixel (FIG 3.11).

Armés de cette information, nous pouvons maintenant adapter la technique de « fausse colonne » à notre grille fluide. Nous allons d'abord convertir ce point de transition en un pourcentage proportionnel à la largeur de notre module de blog. Et pour cela, nous ressortons notre bonne vieille formule. Nous avons notre valeur cible de **568px**, et la largeur du design — notre contexte — est de **900px**. Entrons ces deux valeurs dans notre fidèle formule :

$$568 \div 900 = 0,6311111111111111$$

Eh oui, encore un nombre sans fin, qui nous donne un pourcentage de 63,111111111111 %.

Gardez ce pourcentage dans un coin de votre esprit. Ouvrez maintenant votre éditeur d'images préféré, et créez un document d'une largeur absurdement grande — disons 3000 pixels (FIG 3.12). Et comme nous allons répéter cette image verticalement, sa hauteur ne sera que de **160px**.

Dans un instant, nous allons transformer ce document vierge en image d'arrière-plan. Mais pourquoi une telle largeur ? Eh bien, cette image doit être plus large que n'importe quelle fenêtre de navigateur. Et à moins que vous ne lisiez ce site au xxv<sup>e</sup> siècle sur un écran holographique, je pars du principe que votre écran n'est pas si large.

Pour créer les colonnes elles-mêmes, nous allons devoir appliquer le pourcentage de notre point de transition

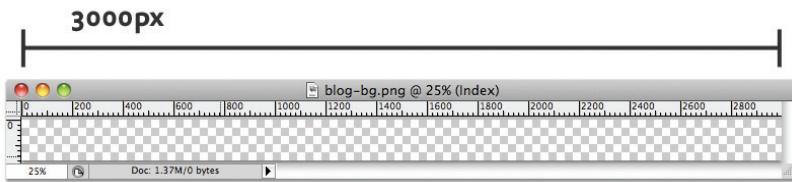


FIG 3.12 : Une toile démesurée qui deviendra (bientôt) notre image d'arrière-plan.

(63,1111111111111%) à notre nouvelle toile plus large. Comme nous travaillons avec une image de 3000 pixels de large, il nous faut simplement multiplier cette largeur par le pourcentage, comme ceci :

$$3000 \times 0.63111111111111 = 1893,333333333333$$

On obtient le résultat 1893,333333333333. Comme Photoshop ne prend que les pixels entiers, arrondissons à 1893. Avec ce nombre, nous allons recréer nos textures dans notre image vierge, en la faisant virer du blanc au gris au niveau du 1893<sup>e</sup> pixel (FIG 3.13).

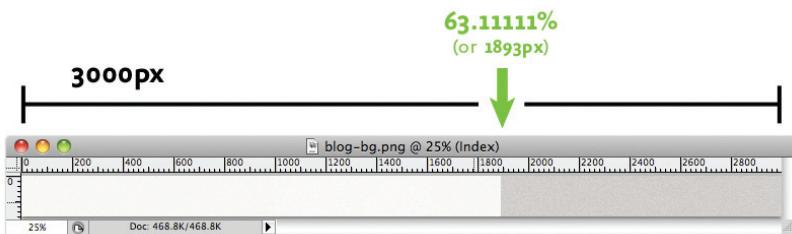


FIG 3.13 : Nous avons appliqué ce pourcentage à notre image d'arrière-plan trop large, obtenant ainsi des colonnes prêtes à être répétées.

En quoi cela nous aide-t-il ? Eh bien, nous venons de déplacer notre point de transition proportionnellement à notre nouvelle toile plus large. Nous pouvons donc utiliser cette valeur pour

récréer nos colonnes : la colonne blanche fera `1893px` de large, et la grise occupera la place restante.

Il ne nous reste plus qu'une seule chose à faire : incorporer notre image fraîchement moulue dans notre feuille de styles.

```
.blog {  
    background: #F8F5F2 url(«blog-bg.png») repeat-y »  
    63.111111111111% 0; /* 568px / 900px */  
}
```

Comme avec la technique originale de Dan, on place l'image au sommet de notre blog, puis on la répète verticalement le long de la largeur du module (`repeat-y`). Mais la valeur `background-position` réutilise le pourcentage de notre point de transition (`63.111111111111% 0`), afin que les colonnes restent en place si l'on redimensionne la page.

Et maintenant, nos fausses colonnes fonctionnent à merveille dans une mise en page fluide (FIG 3.14). Tout ça grâce à l'approche de Dan Cederholm, appliquée à une logique proportionnelle.

## Images d'arrière-plan entièrement flexibles ?

Évidemment, notre fausse colonne flexible n'est pas vraiment flexible : on utilise simplement des pourcentages pour placer une image d'arrière-plan de manière à ce que les colonnes semblent se redimensionner en même temps que leur contenant. Les dimensions de l'image n'ont absolument pas changé.

Mais si on veut qu'une image d'arrière-plan se redimensionne effectivement en même temps que la mise en page ? Il vous est sans doute arrivé de placer un logo dans l'arrière-plan d'un élément `h1`, ou d'utiliser des sprites pour créer des effets de survol sur les liens de votre site. Peut-on redimensionner des images qui doivent se trouver à l'arrière-plan ?

En quelque sorte, oui. Il existe une propriété CSS3, `background-size` (<http://bkaprt.com/rwd/20/>), qui nous permettrait de créer des images d'arrière-plan vraiment flexibles. Mais vous l'aurez deviné, elle est encore mal prise en charge.

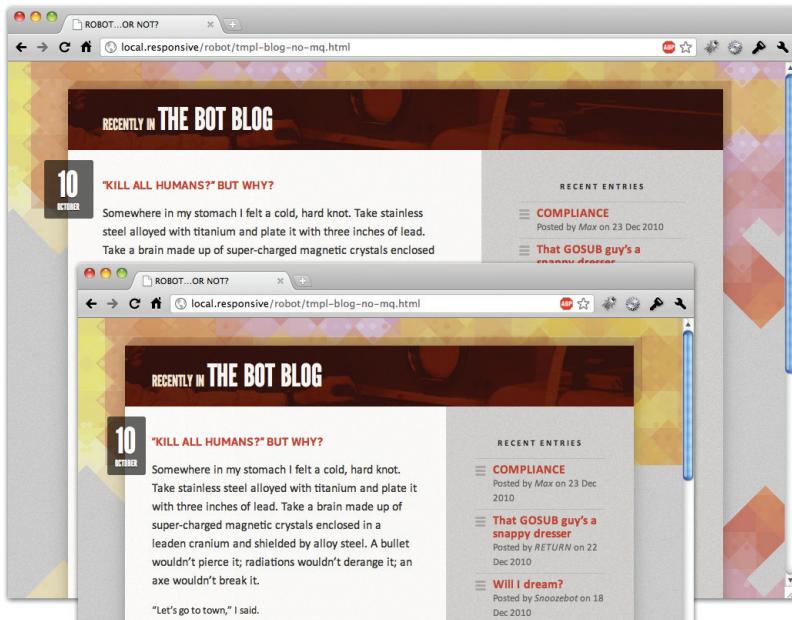


FIG 3.14 : Nos fausses colonnes flexibles.

En attendant, il existe des solutions plutôt ingénieuses en JavaScript : par exemple, le plugin jQuery Backstretch de Scott Robbin (<http://bkaprt.com/rwd/21/>) émule des images d'arrière-plan redimensionnables dans l'élément `body`. Et comme vous le verrez au prochain chapitre, les media queries CSS3 peuvent également être utilisées pour appliquer différentes images d'arrière-plan adaptées à chaque résolution. Alors même si `background-size` n'est pas encore franchement utilisable, tout est possible.

## APPRENEZ À AIMER OVERFLOW

Il existe d'autres options pour intégrer des images à largeur fixe dans un contexte fluide. En fait, vous feriez bien de vous intéresser aux expériences de Richard Rutter sur le place-

ment d'images larges dans des mises en page flexibles (<http://bkapt.com/rwd/11/>). Vous y trouverez pas mal d'expériences prometteuses, dont certaines vous seront peut-être utiles pour commencer à bricoler des mises en page flexibles.

Une méthode que j'ai employée à plusieurs occasions s'appuie sur la propriété `overflow`. Comme nous l'avons vu précédemment dans ce chapitre, une image large, par défaut, dépassera tout simplement de son contenant. Et dans la plupart des cas, la règle `max-width: 100%` est la meilleure manière de la restreindre et de lui donner une taille convenable. Mais vous pourriez aussi choisir de rogner l'image en lui appliquant la règle `overflow: hidden`. Ainsi, plutôt que de demander à notre image de se redimensionner automatiquement :

```
.feature img {  
    max-width: 100%;  
}
```

Nous pouvons simplement couper l'excédent comme ceci :

```
.feature {  
    overflow: hidden;  
}  
  
.feature img {  
    display: block;  
    max-width: auto;
```

---

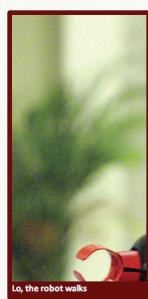
**FIG 3.15 :** Une fois la propriété `overflow: hidden` appliquée au contenant de notre image, on obtient une image coupée. Cool... enfin, si on veut.

"Coming?" I asked.

Jack was pale under his freckles but Chief Dalton grinned back at me.  
"We'll be right behind you, Morrison."

he said.  
Behind me! So they could pick up the pieces. I gave them a cocky smile and switched on the engine, full speed.

Caron City is about a mile from the plant. It has about fifty thousand inhabitants. At that moment, though, there wasn't a soul in the streets. I heard people calling to each other inside their houses, but I didn't see anyone, human or android. I circled in for a landing, the Police Copter hovering maybe a quarter of a mile back of me. Then, as the wheels touched, half a dozen androids came around the corner. They saw me and

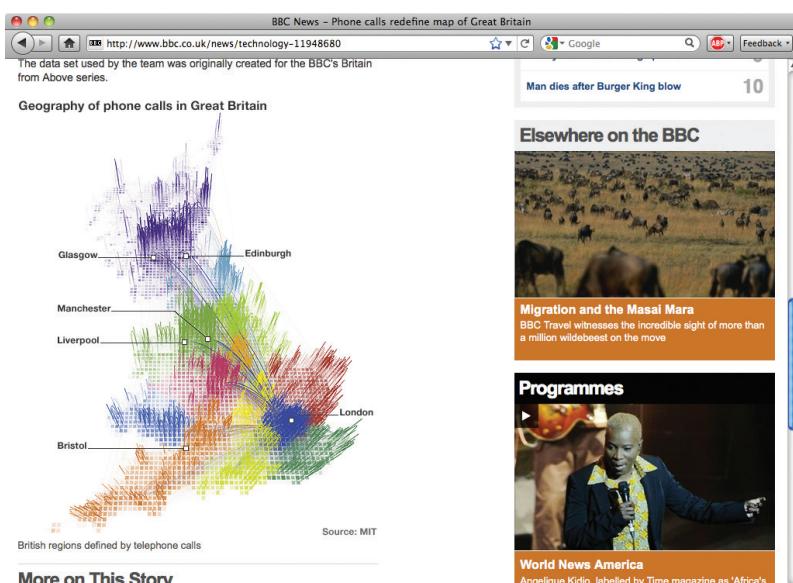


Et voilà le résultat : une image coupée pour rentrer dans son contenant (FIG 3.15). La totalité de l'image est toujours là, mais ce qui dépassait a été caché.

Comme vous pouvez le voir, ce n'est pas vraiment une solution exploitable. En fait, j'ai constaté que dans l'écrasante majorité des cas, `overflow` est généralement moins utile que `max-width`. Mais c'est tout de même une option à considérer, et à laquelle vous trouverez peut-être une utilité.

## NÉGOCIEZ VOTRE CONTENU

On remarquera que les approches utilisant `overflow` et `max-width: 100%` pour créer des images flexibles sont plutôt robustes, et marchent remarquablement bien avec la plupart des médias. En fait, je les ai utilisées avec succès dans de nombreuses grilles fluides complexes.



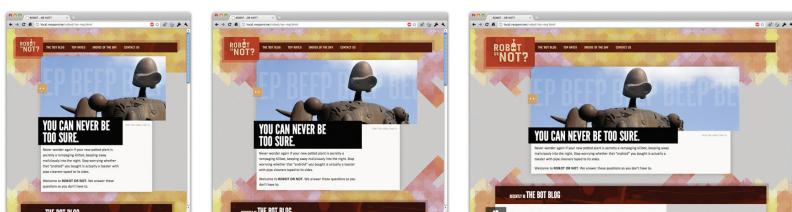
**FIG 3.16 :** Cette infographie détaillée provenant du site de BBC News (<http://bkapr.com/rwd/22/>) contient des informations cruciales pour le contenu de la page. Le fait de la réduire pourrait s'avérer contre-productif.

Cependant, ces deux approches ne prêtent finalement aucune attention au contenu. Elles établissent toutes deux des règles de base régissant l'interaction d'une image avec son contenant : `max-width: 100%` rétrécit les images trop grandes pour qu'elles correspondent à la largeur de leur contenant, tandis que la propriété `overflow` permet au designer de cacher la partie d'une image qui dépasserait de son contenant.

Mais comment faire avec des graphismes particulièrement complexes ? Si votre image est riche en informations (FIG 3.16), la redimensionner ou la couper peut s'avérer peu souhaitable — en fait, ces approches tendent plutôt à gêner la compréhension du contenu de cette image pour vos lecteurs.

Si c'est le cas, vous pouvez vous intéresser aux méthodes permettant de fournir différentes versions de la même image pour différentes plages de résolutions. En d'autres termes, vous pouvez créer plusieurs versions de votre image, par exemple une pour les ordinateurs de bureau et une autre, plus linéaire, pour les petits écrans. Une fois ces options établies, une solution sur le serveur peut fournir l'image la plus adaptée à la résolution de l'appareil.

La création d'une telle solution dépasse la portée de ce livre (et les compétences de votre humble serviteur), mais le designer et développeur Bryan Rieger a exposé une approche possible sur son blog (<http://bkaprt.com/rwd/23/>), et offre sa solution en téléchargement.



**FIG 3.17 :** Deux chapitres plus tard, nous avons finalement terminé une grille de mise en page qui peut s'agrandir et rétrécir en même temps que la fenêtre du navigateur.

Si vous décidez d'implémenter une solution back-end, vous pouvez lui adjoindre l'une des diverses techniques « côté client » dont nous avons parlé jusqu'à présent. Par exemple, vous pouvez fournir des images pour un certain nombre de résolutions, puis utiliser `max-width: 100%` pour aplanir la transition entre les différentes versions, afin de prendre correctement en charge les autres appareils, navigateurs et résolutions au besoin.

## IMAGES ET GRILLES FLEXIBLES, TENEZ-VOUS BIEN

À ce stade, nous avons exploré tout ce qu'il faut savoir pour construire des grilles de mise en page complexes mais flexibles : quelques calculs simples et des stratégies pour intégrer des images et d'autres médias dans cette structure. Nous avons commencé par un module de blog plutôt simple, et nous allons maintenant construire le reste du site Robot or Not, en créant un design basé sur un système de proportions et de pourcentages, sans un pixel à l'horizon (FIG 3.17).

Ces fondations flexibles étant en place, nous sommes prêts à ajouter l'ingrédient final à notre design réactif.

# 4 LES MEDIA QUERIES

PENDANT UNE GRANDE PARTIE de ma carrière, j'ai été un ardent défenseur des mises en page non fixes. Flexible ou complètement fluide, cela n'avait pas d'importance : je sentais que le fait d'intégrer une certaine dose de fluidité dans nos designs les préparait mieux aux changements inhérents au Web : changements de la taille du navigateur, de l'écran ou de la résolution des appareils. De plus, j'utilisais souvent des expressions comme « à l'épreuve de l'avenir » ou « tous terminaux confondus » pour parler de ce besoin de flexibilité. Souvent seul dans mon coin.

Mais à un certain stade, tout se brise.

Notre site Robot or Not a beau être flexible, il n'est pas à l'abri de tout. Bien sûr, sa grille fluide lui permet de résister aux changements de taille et de résolution, bien mieux qu'une mise en page fixe. Mais des changements, même mineurs, de la taille et de la forme de la fenêtre du navigateur provoqueront la déformation, voire la destruction de notre mise en page.

Mais voilà, dites-vous bien une chose : ce n'est pas grave.

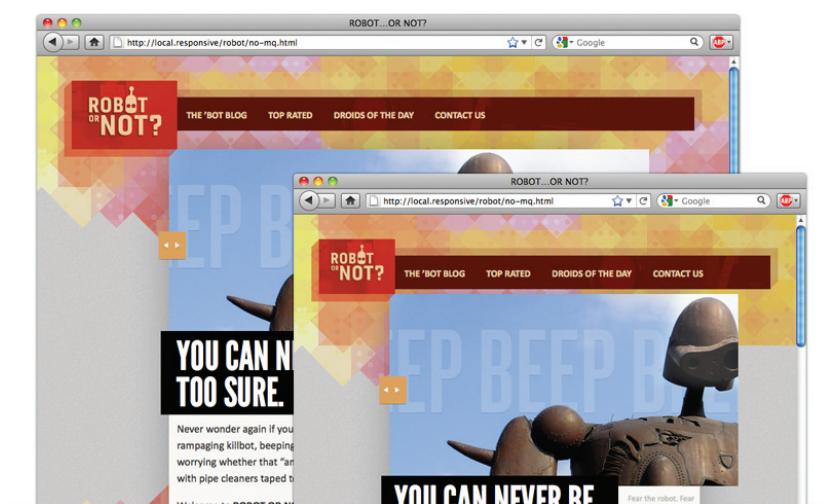
# CICATRISATION DOULOUREUSE

Aussi douloureux que cela puisse être, regardons à quels endroits notre design perd de sa superbe quand on le redimensionne. En identifiant les problèmes que nous rencontrons, nous serons plus à même d'appliquer les correctifs requis. Même si nous devons verser une larme au passage.

Comme nous travaillons avec une mise en page flexible, nous pouvons simplement redimensionner la fenêtre du navigateur de plusieurs manières. Rien ne remplace un test sur des appareils différents, mais cela nous permet d'évaluer rapidement comment notre design gère différentes résolutions, et de simuler le rendu que nos utilisateurs pourraient obtenir sur leur téléphone, leur tablette ou tout autre appareil.

## Une question d'emphase

Commençons par rétrécir un peu la fenêtre du navigateur, en passant de 1024 pixels de large à environ 760 (FIG 4.1).



**FIG 4.1 :** En ajustant la taille de la fenêtre de notre navigateur, nous pouvons avoir un aperçu de ce que notre design donnera dans différentes résolutions.

Assez rapidement, un certain nombre de problèmes font leur apparition.

Notre design initial faisait dans l'emphase : des titres importants, une image de robot proéminente et des marges généreuses. Tout cela parvient tout de même à se redimensionner dans notre mise en page flexible — mais visuellement, les priorités ont complètement changé.

Regardez le haut de notre site : l'image principale domine maintenant toute la page (FIG 4.2). Comme nous coupons l'image avec la propriété `overflow`, elle ne se redimensionne pas avec le reste de notre grille flexible. En plus, le sujet de notre image, notre robot adoré, s'en trouve sévèrement tronqué. Il nous reste une image non seulement énorme, mais à peine compréhensible. Génial.

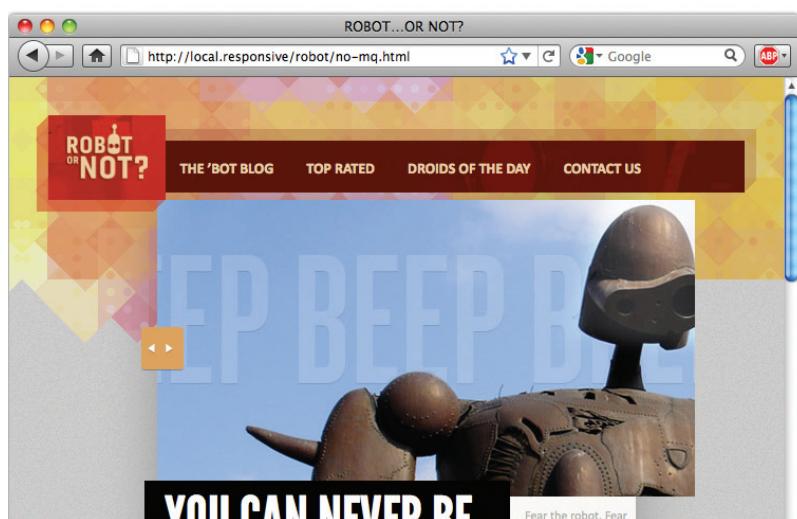


FIG 4.2 : Il n'y a pas exactement de quoi sortir le champagne, vous voyez ce que je veux dire ?

Dans l'ombre de cette gigantesque image, notre logo est devenu minuscule. Et le peu d'espace qui se trouvait entre les liens et l'image a disparu, ce qui rend l'en-tête un peu encombré.

Ça me fait mal de l'admettre, mais notre hiérarchie visuelle est réduite à néant dès lors que l'on change la résolution pour laquelle on l'a conçue à la base.

## Grille miniature, problèmes monstrés

Et ce n'est pas le pire. Si l'on réduit encore un peu plus la fenêtre du navigateur, aux alentours de 600 pixels — la taille d'un petit navigateur, ou des nouvelles tablettes en mode portrait — le casse-tête ne fait qu'empirer (FIG 4.3). Au sommet de l'écran, notre hiérarchie visuelle est toujours sens dessus dessous : notre image est maintenant coupée au point de l'incohérence, et notre pauvre logo est de plus en plus lilliputien. Mais maintenant, nos liens ne tiennent plus sur une seule ligne, ce qui est plutôt gênant. On doit sûrement pouvoir faire mieux que ça ?

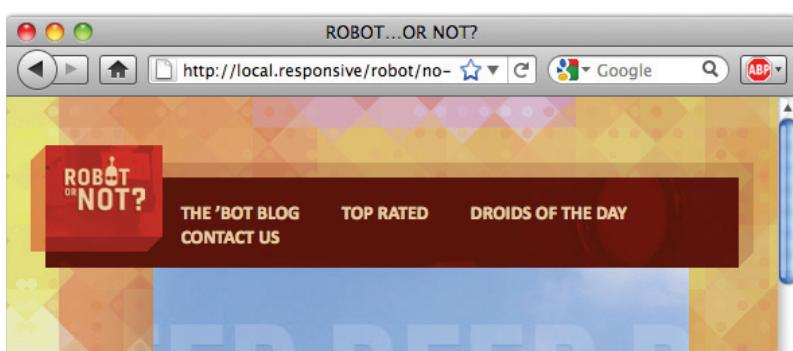
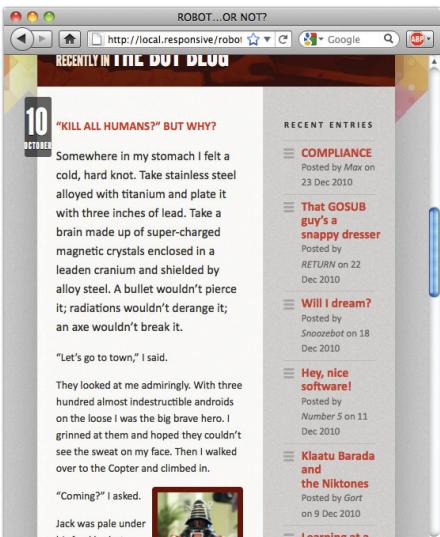


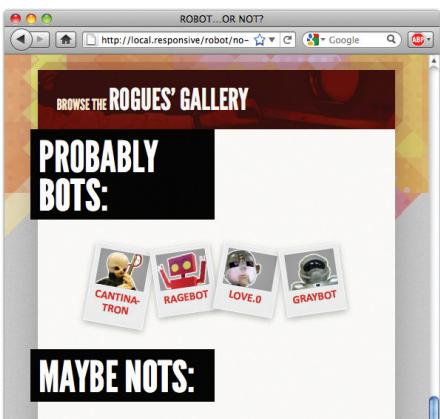
FIG 4.3 : Les visiteurs de notre site adoreront la mise en page complètement destructurée. Non, croyez-moi, ils adoreront.

Si l'on descend un peu, notre blog commence vraiment à pâtrir (FIG 4.4). Les deux colonnes qui offraient auparavant un accès facile aux informations sont maintenant comprimées. Notamment, les lignes du texte de l'article sont beaucoup trop courtes et gênent la lecture. Et le contenu de l'image est à peine distinguable.

**FIG 4.4 :** Parcourir ce blog, /  
Et penser lire un haïku : /  
lignes bien trop courtes.



**FIG 4.5 :** Petites images,  
marges monstrueuses. Un  
mariage en enfer.



Enfin, pour finir la liste des doléances, le module photo en bas de la page est pire encore (FIG 4.5) : les images sont ridiculement petites et presque indéchiffrables.

Les marges générées que nous avions placées au départ pour encadrer ces photos semblent maintenant totalement disproportionnées et noient nos photos dans une mer de blanc.

## Vils écrans larges

Cependant, nos problèmes ne n'arrêtent pas aux plus petites résolutions. Si nous maximisons la fenêtre de notre navigateur, de nombreux autres problèmes de design font leur apparition.

L'intro (FIG 4.6) n'est pas si horrible, mais l'image est maintenant plus petite que l'espace qui lui est alloué. Exception faite de ce détail, les choses ne se présentent pas si mal au sommet de la page. C'est loin d'être idéal, mais ce n'est pas non plus épouvantable. Globalement, notre grille flexible s'en tire bien jusqu'ici.

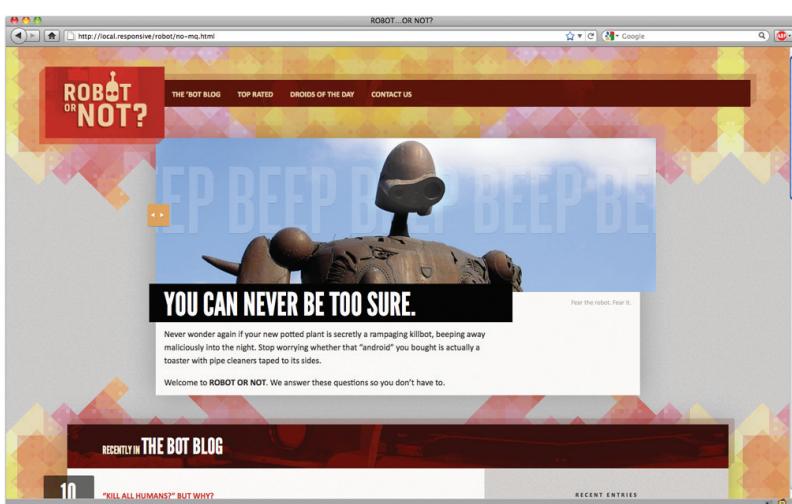
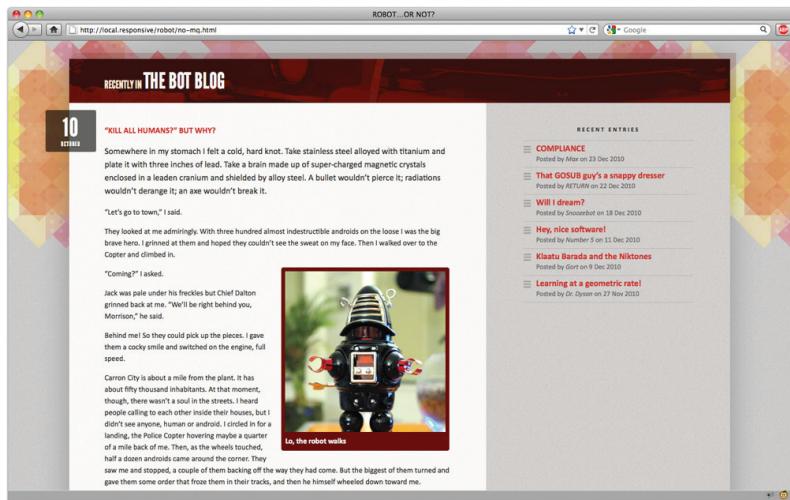


FIG 4.6 : Cette intro est juste beaucoup trop large.

Mettons sans plus tarder un terme à tout optimisme et descendons voir le reste du blog (FIG 4.7). Vous vous souvenez du texte haché par une fenêtre trop petite ? C'était presque mieux : ces lignes-ci sont terriblement longues, et la largeur de l'article est beaucoup trop généreuse. Mes yeux adorent faire des kilomètres pour revenir à la ligne, mais il doit exister une meilleure solution.



**FIG 4.7 :** En descendant le long de la page, les choses se corsent. Longues lignes, petites images, Ethan triste.

Et pour couronner le tout, notre galerie de photos domine complètement le bas de la page (FIG 4.8). Les images elles-mêmes ne présentent aucun défaut, si ce n'est qu'elles sont énormes. En fait, sur mon écran, il n'y a aucun moyen de deviner le contenu qui se trouve au-dessus ou en dessous du module. Est-ce là vraiment la meilleure façon de présenter ces informations à nos lecteurs ?

## LE PROBLÈME EN QUESTION

Nous avons identifié de nombreux problèmes visuels. Mais un problème plus grave se pose. Si l'on dépasse la résolution pour laquelle notre grille a été conçue à l'origine, celle-ci devient un handicap pour notre contenu. Ses proportions compriment notre contenu si la résolution est trop petite, elles l'isolent dans un océan de blanc si la résolution est trop grande.

Cependant, ce problème ne se pose pas qu'avec les mises en page flexibles. Aucun design, fixe ou fluide, ne se redimensionne convenablement au-delà du contexte pour lequel il a été conçu.

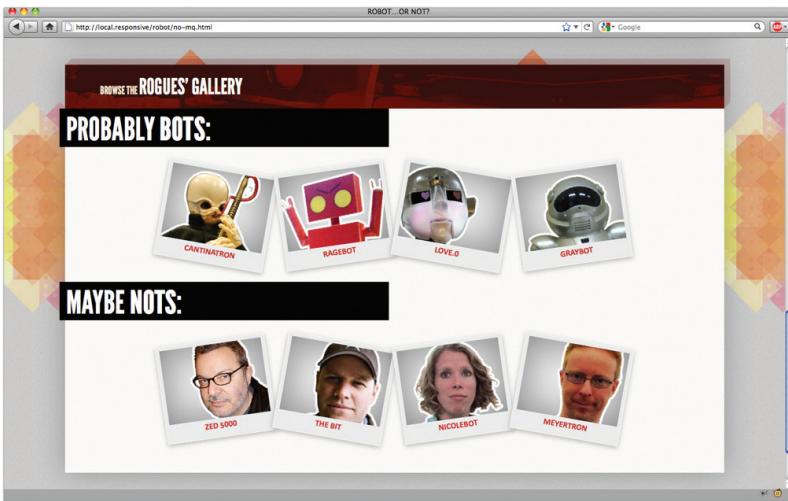


FIG 4.8 : Ces images sont, pour parler technique, grosses et joufflues.

Alors comment créer un design qui peut s'adapter aux changements de résolution et de dimensions du navigateur ? Comment optimiser notre page pour la myriade de navigateurs et d'appareils qui y accéderont ?

En d'autres termes, comment rendre nos designs plus « responsifs » ?

## TRAÎNASSER VERS PLUS DE RÉACTIVITÉ

Heureusement, le W3C est aux prises avec cette question depuis un certain temps. Pour mieux comprendre la solution qui a fini par être présentée, remettons les choses dans leur contexte.

### Les types de médias

La première tentative de solution donna naissance aux types de médias, inclus dans la spécification CSS2 (<http://bkaprt.com/rwd/24/>). Voilà comment ils étaient alors décrits :

*Occasionnellement, cependant, des feuilles de style pour différents types de médias peuvent partager une même propriété, mais utiliser des valeurs différentes pour cette propriété. Par exemple, la propriété « font-size » est utile à l'écran et sur un support imprimé. Ces deux types de médias sont suffisamment différents pour qu'il faille donner des valeurs différentes à la propriété commune ; un document requerra généralement une police plus grande sur un écran d'ordinateur que sur papier. Par conséquent, il est nécessaire de pouvoir exprimer qu'une feuille de styles, ou une section d'une feuille de styles, doive s'appliquer à certains types de médias.*

Ouais, d'accord. Pas très clair, non ? Essayons de traduire ça en français.

Vous avez déjà écrit une feuille de styles pour l'impression (<http://bkaprt.com/rwd/25/>) ? Alors vous avez déjà l'habitude de concevoir pour différents types de médias. L'expérience de navigation idéale diffère grandement entre les navigateurs de bureau et les imprimantes, ou entre les appareils portatifs et les lecteurs d'écran. Pour répondre à ce problème, le W3C a créé une liste de types de médias (<http://bkaprt.com/rwd/26/>), tentant de classer chaque navigateur ou appareil dans une catégorie spécifique. Les types de médias reconnus sont : **all, braille, embossed, handheld, print, projection, screen, speech, tty** et **tv**.

Vous avez probablement déjà utilisé certains de ces types de médias, comme **print** ou **screen**, voire même **projection**. D'autres, comme **embossed** (pour les imprimantes braille) ou **speech** (pour les navigateurs et les interfaces parlants), sont peut-être nouveaux pour vous. Mais tous ces types de médias ont été créés pour que nous puissions plus facilement concevoir pour chaque type de navigateur ou d'appareil, en chargeant conditionnellement des feuilles de styles adaptées. Un appareil utilisant **screen** ignorera la feuille de styles chargée pour le type de média **print**, et vice versa. Quant aux règles censées s'appliquer à tous les appareils, la spécification a créé la catégorie **all**.

En pratique, cela consiste à personnaliser l'attribut **media** d'un lien :

```
<link rel="stylesheet" href="global.css" media="all" />
<link rel="stylesheet" href="main.css" media="screen" />
```

```
<link rel="stylesheet" href="paper.css" media="print"
/>>
```

Ou alors à créer un bloc `@media` dans votre feuille de styles, et à l'associer à un type de média particulier :

```
@media screen {
  body {
    font-size: 100%;
  }
}

@media print {
  body {
    font-size: 15pt;
  }
}
```

Dans les deux cas, la spécification suggère que le navigateur s'identifie comme appartenant à l'un des types de médias. (« Je suis un ordinateur de bureau ! J'appartiens au type de média `screen`. » « Je sens le papier et le toner chaud : je suis un média `print`. » « Je suis le navigateur de votre console de jeu : je suis un média `tv`. » Et ainsi de suite.) En chargeant la page, le navigateur restituerait alors seulement la CSS correspondant à son type de média particulier, en laissant le reste de côté. Et en théorie, c'est une idée fantastique.

Bien sûr, la théorie, c'est la dernière chose dont un designer Web sérieux a besoin.

## Mauvais casting

Plusieurs problèmes sont apparus avec les types de médias quand tous ces navigateurs pour petits écrans, comme ceux des téléphones et des tablettes, sont arrivés sur le devant de la scène. D'après la spécification, les designers auraient pu les cibler simplement en créant une feuille de styles pour le type de média `handheld` :

```
<link rel="stylesheet" href="main.css" media="screen" />
<link rel="stylesheet" href="paper.css" media="print" />
<link rel="stylesheet" href="tiny.css" media="handheld"/>
```

Le problème de cette approche, c'était... nous. Enfin, en partie. Les premiers appareils mobiles n'étaient pas dotés de navigateurs suffisamment puissants ; on les a donc largement ignorés, préférant concevoir des feuilles de style pour les médias `screen` et `print`. Et quand des navigateurs dignes de ce nom ont fini par voir le jour, on ne trouvait pas des masses de fichiers CSS `handheld` sur le Web. De nombreux créateurs de navigateurs mobiles ont alors décidé de lire les feuilles de style `screen` par défaut.

De surcroît, chaque type de média englobe de nombreux modes d'accès. Une même feuille de styles `handheld` peut-elle vraiment convenir à un iPhone comme à un téléphone vieux de cinq ans ?

## Découvrez les media queries

Constatant les écueils des types de médias, le W3C a profité du travail sur CSS3 pour tenter d'apporter une nouvelle réponse au problème : les media queries (<http://bkaprt.com/rwd/27>), un mécanisme incroyablement robuste pour identifier non seulement les types de médias, mais aussi inspecter les caractéristiques physiques des appareils et des navigateurs qui restituent notre contenu.

Voyons ce que ça donne :

```
@media screen and (min-width: 1024px) {
  body {
    font-size: 100%;
  }
}
```

Chaque media query — y compris celle-ci — est composée de deux éléments :

1. Elle commence toujours par le **type** de média (`screen`) hérité de la spécification CSS2.1 (<http://bkaprt.com/rwd/26/>).

2. La **requête** elle-même est placée juste après, entre parenthèses : (`min-width: 1024px`). Et notre requête peut, à son tour, être divisée en deux composantes : le nom d'une **caractéristique** (`min-width`) et la **valeur** correspondante (`1024px`).

C'est un peu comme un test de votre navigateur. Quand un navigateur lit votre feuille de styles, la requête `screen and (min-width: 1024px)` pose deux questions : tout d'abord, le navigateur appartient-il au type de média `screen` ? Et le cas échéant, la fenêtre du navigateur fait-elle au moins 1024 pixels de large ? Si le navigateur répond à ces deux critères, alors les styles que renferme la requête sont restitués ; sinon, le navigateur détourne le regard et suit son petit bonhomme de chemin.

Notre media query ci-dessus est écrite dans une déclaration `@media`, ce qui nous permet de la placer directement dans une feuille de styles. Mais vous pouvez également la placer dans un élément `link` en l'insérant dans l'attribut `media` :

```
<link rel="stylesheet" href="wide.css" media="screen and  
(min-width: 1024px)" />
```

Ou vous pouvez l'attacher à une déclaration `@import` :

```
@import url("wide.css") screen and (min-width: 1024px);
```

Personnellement, je préfère l'approche employant `@media`, car elle permet de garder tout le code dans un même fichier, tout en réduisant le nombre de requêtes envoyées au serveur.

Mais quelle que soit la méthode que vous choisirez, le résultat sera le même : si le navigateur correspond au type de média et remplit la condition définie dans notre requête, il appliquera la CSS correspondante. Sinon, il l'ignorera.

## Les caractéristiques

On ne va pas se contenter de tester la largeur et la hauteur. La spécification propose tout un tas de caractéristiques à soumettre à l'examen de notre requête. Mais avant de nous y intéresser, notons que le langage utilisé pour décrire ces caractéristiques

peut être assez... dense. Voici deux conseils qui m'ont aidé à m'y retrouver :

1. Dans le langage de la spécification, chaque appareil a une « zone d'affichage » et une « surface de rendu ». Voilà qui est clair comme de la vase. Ce qu'il faut comprendre : la fenêtre du navigateur est la zone d'affichage, l'écran tout entier est la surface de rendu. Donc sur votre ordinateur portable, la zone d'affichage serait la fenêtre de votre navigateur et la surface de rendu, votre écran. (Ce n'est pas moi qui fais les mots.)
2. Pour tester les valeurs au-dessus ou en dessous d'un certain seuil, certaines caractéristiques acceptent les préfixes `min-` et `max-`. Un bon exemple est `width` : vous pouvez servir une CSS conditionnellement aux résolutions de plus de 1024 pixels en écrivant (`min-width: 1024px`) ou au contraire, aux résolutions de moins de 1024 pixels avec (`max-width: 1024 px`).

Vous avez tout compris ? Fantastique. Ces deux points étant éclaircis, intéressons-nous aux caractéristiques définies par la spécification (<http://bkaprt.com/rwd/28/>) (TABLEAU 4.1)

---

TABLEAU 4.1 : Liste des caractéristiques que nous pouvons tester avec nos media queries.

CARACTÉRISTIQUE	DÉFINITION	PRÉFIXES <code>min-</code> ET <code>max-</code>
<code>width</code>	Largeur de la zone d'affichage.	✓
<code>height</code>	Hauteur de la zone d'affichage.	✓
<code>device-width</code>	Largeur de la surface de rendu de l'appareil.	✓
<code>device-height</code>	Hauteur de la surface de rendu de l'appareil.	✓

---

CARACTÉRISTIQUE	DÉFINITION	PRÉFIXES min- ET max-
orientation	Accepte les valeurs portrait et landscape.	✗
aspect-ratio	Rapport entre la largeur et la hauteur de la zone d'affichage. Par exemple : sur un ordinateur de bureau, on peut demander si la fenêtre du navigateur est au format 16:9.	✓
device-aspect-ratio	Rapport entre la largeur et la hauteur de la surface de rendu de l'appareil.	✓
color	Nombre de bits utilisés pour représenter la couleur d'un pixel sur un appareil. Par exemple, un appareil offrant une profondeur de couleurs de 8 bits répondrait à la requête (color: 8). Un appareil sans couleurs devrait renvoyer la valeur 0.	✓
color-index	Nombre d'entrées dans la table des couleurs de l'appareil. Par exemple, @media screen and (min-color-index: 256).	✓

CARACTÉRISTIQUE	DÉFINITION	PRÉFIXES min- ET max-
monochrome	Comme color, la caractéristique monochrome nous permet de tester le nombre de bits par pixel sur un appareil monochrome.	✓
resolution	Teste la densité des pixels sur l'appareil, par exemple screen and (resolution: 72 dpi) ou screen and (max-resolution: 300 dpi).	✓
scan	Sur les télévisions, mesure si le balayage est progressif (progressive) ou entrelacé (scan).	✗
grid	Teste si l'appareil utilise un affichage en grille, comme les téléphones avec une seule taille de police. Peut s'exprimer simplement (grid).	✗

Ce qui est vraiment excitant, c'est que l'on peut enchaîner plusieurs requêtes avec l'opérateur `and` :

```
@media screen and (min-device-width: 480px) and
(orientation: landscape) { ... }
```

Cela nous permet de tester plusieurs caractéristiques en une seule requête, donc de créer des tests plus complexes pour les appareils qui restituent nos designs.

## Tes caractéristiques tu connaîtras

Vous vous sentez déjà ivre de pouvoir ? Eh bien, je devrais saisir cette opportunité pour mentionner que tous les navigateurs compatibles avec `@media` ne supportent pas toutes les caractéristiques définies dans la spécification.

Un exemple rapide : quand l'iPad d'Apple est sorti, il supportait la propriété `orientation`. Cela signifiait que l'on pouvait écrire une requête `orientation: landscape` ou `orientation: portrait` pour servir conditionnellement notre CSS à l'appareil, suivant la manière dont il était tenu. Cool non ? Malheureusement, l'iPhone ne supportait pas la requête `orientation` avant la sortie d'une mise à jour, quelques mois plus tard. Alors que les deux appareils permettaient à l'utilisateur de changer l'orientation, le navigateur de l'iPhone ne comprenait pas les requêtes ciblant cette caractéristique particulière.

Morale de l'histoire ? Étudiez minutieusement la compatibilité des appareils et des navigateurs que vous ciblez et testez-les en fonction.

Bien que la prise en charge ne soit pas encore étendue à tous les navigateurs et appareils modernes, les media queries nous offrent déjà un vocabulaire vaste nous permettant d'articuler la manière dont nous souhaitons que nos designs se comportent avec différents appareils et navigateurs.

## UN ROBOT PLUS « RESPONSIVE »

Et c'est pour cela que les media queries sont la touche finale d'un site Web adaptatif. Nous avons passé deux chapitres à implémenter notre grille de mise en page flexible — mais ce n'est que la base. Nous pouvons utiliser les media queries pour corriger les imperfections visuelles qui surgissent lorsqu'on redimensionne la fenêtre du navigateur.

De plus, nous pouvons aussi les utiliser pour optimiser l'affichage de notre contenu et mieux répondre aux besoins de l'appareil, en créant des mises en page alternatives adaptées à chaque plage de résolutions. En chargeant conditionnellement des règles de style qui ciblent ces plages, les media queries nous permettent de créer des pages plus sensibles aux besoins des appareils qui les restituent.

En d'autres termes, en combinant une mise en page flexible et les media queries, nous allons finalement parvenir à rendre nos sites réactifs.

En avant.

## Chambre avec viewport

Nous avons déjà identifié un certain nombre de points sensibles dans notre design. Mais avant de commencer à appliquer nos media queries, il nous faut faire une dernière modification dans notre code source.

Quand Apple a sorti l'iPhone en 2007, ils ont créé un nouvel attribut pour l'élément `meta` de Mobile Safari : `viewport` (<http://bkaprt.com/rwd/29/>). Pourquoi ? Il se trouve que les dimensions de l'écran de l'iPhone sont de 320×480, mais Mobile Safari affiche en fait les pages Web avec une largeur de 980 pixels. Si vous avez déjà visité le site du *New York Times* (<http://nytimes.com>) sur un téléphone utilisant le moteur WebKit (FIG 4.9), vous avez déjà vu ce comportement en action : Mobile Safari affiche la page sur une toile de 980px de large, puis la rétrécit aux dimensions de l'écran du téléphone, 320×480.

La balise `viewport` nous permet de contrôler la taille de cette toile, et d'outrepasser ce comportement par défaut : nous pouvons dicter exactement la largeur que doit prendre la fenêtre du navigateur et donc, par exemple, donner une largeur de `320px` à nos pages :

```
<meta name="viewport" content="width=320" />
```

Depuis sa création par Apple, plusieurs développeurs de navigateurs ont adopté la mécanique `viewport`, créant de facto une sorte de standard. Alors incorporons-la dans notre design (prochainement) réactif. Mais plutôt que de déclarer une largeur fixe en pixels, adoptons une approche faisant abstraction de la résolution. Dans le `head` de notre HTML, ajoutons cet élément `meta` :

```
<meta name="viewport" content="initial-scale=1.0,  
width=device-width" />
```



**FIG 4.9 :** Par défaut, Mobile Safari restitue le contenu Web à une largeur de 980px, alors que l'écran ne fait que 320 px de large en mode portrait.

La propriété `initial-scale` fixe le niveau de zoom de la page à **1.0**, soit 100 %, et contribue à garantir un certain degré d'uniformité sur les navigateurs des appareils à petit écran. (Pour plus d'informations sur le fonctionnement du redimensionnement sur divers écrans, je vous recommande l'explication de Mozilla : <http://bkaprt.com/rwd/30.>)

Mais ce qui nous intéresse, c'est le paramètre `width=device-width`, qui donne à la fenêtre du navigateur une largeur égale à celle de l'écran de l'appareil. Ainsi, sur un iPhone par exemple, Mobile Safari n'afficherait plus la page avec une largeur par défaut de `980px`. Au lieu de cela, elle ferait 320 pixels de large en mode portrait, et 480 en mode paysage.

Avec cette valeur en place, nous pouvons utiliser `max-width` et `min-width` pour tester la résolution et charger conditionnellement la CSS appropriée. De plus, cela permet à tous les navigateurs compatibles de profiter de nos media queries, rendant notre design adaptatif à tous les utilisateurs — qu'ils utilisent un téléphone, une tablette, un ordinateur de bureau ou un ordinateur portable.

Assez de baratin, voyons la bête en action.

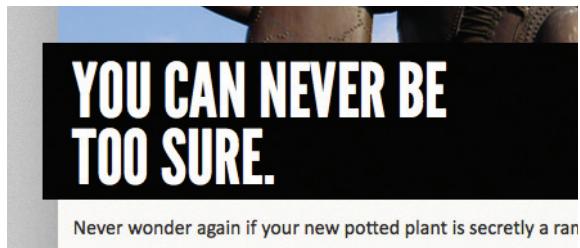
## LES MEDIA QUERIES EN ACTION

Vous vous rappelez de ces titres grands et imposants (FIG 4.10) ? Voici la CSS correspondante dans l'état actuel des choses :

```
.main-title {  
    background: #000;  
    color: #FFF;  
    font: normal 3.625em/0.9 "League Gothic", »  
        "Arial Narrow", Arial, sans-serif; /* 58px / 16px */  
    text-transform: uppercase;  
}
```

---

**FIG 4.10 :** Admirez l'impact du traitement de notre titre. Vous êtes tout impacté.



J'ai mis de côté quelques propriétés de présentation, parce que ce qui m'inquiète le plus, c'est à quel point ces titres sont ridiculement gros pour les plus petites résolutions. Ils utilisent la majestueuse police League Gothic (<http://bkaprt.com/rwd/31/>), colorisée en blanc (`color: #FFF`) sur un arrière-plan noir (`background: #000`). Et au cas où vous douteriez encore que ce titre doit être pris très au sérieux, il s'affiche en majuscules à l'aide de `text-transform` et prend la taille imposante de `3.625em`, soit `58px`.

Ce traitement marche plutôt bien. Mais comme nous venons de le voir, il perd de son allure quand la place vient à manquer. Sur un navigateur plus étroit ou un écran plus petit, le design ne parvient tout simplement pas à se redimensionner convenablement.

Alors, corrigeons tout ça.

Nous allons d'abord créer un bloc `@media` quelque part après notre règle `.main-title` initiale, pour cibler une plage de résolutions plus basse :

```
@media screen and (max-width: 768px) { ... }
```

Dans cette requête, nous avons demandé au navigateur de rendre la CSS incluse seulement si sa fenêtre mesure moins de 768 pixels de large. Pourquoi `768px` ? Parce que les téléphones compatibles, ainsi que la plupart des tablettes récentes, tombent bien en dessous de ce palier. Ou du moins lorsqu'on les tient d'une certaine façon : par exemple, la résolution de l'iPad est de `768px` de large en mode portrait, mais de `1024px` en mode paysage.

Mais comme on utilise `max-width`, et non `max-device-width`, les ordinateurs de bureau et les portables appliqueront aussi ces règles si la fenêtre du navigateur tombe en dessous du seuil défini. (Souvenez-vous : `width` et `height` mesurent la fenêtre du navigateur, alors que `device-width` et `device-height` mesurent les dimensions de l'écran tout entier.)

Cette requête en place, nous pouvons commencer à cibler les éléments de notre design qui se redimensionnent mal. Commençons par repenser le traitement de notre titre trop grand. Pour cela, nous allons placer une règle `.main-title` dans notre media query afin de neutraliser les propriétés CSS qui nous causent du souci :

```
@media screen and (max-width: 768px) {  
    .main-title {  
        font: normal 1.5em Calibri, Candara, Segoe, «  
            "Segoe UI", Optima, Arial, Helvetica, »  
            sans-serif; /* 24px / 16px */  
    }  
}
```

---

**FIG 4.11 :** League Gothic a beau être une jolie police, elle ne brille vraiment que dans les grandes tailles. Mais ici, elle est un peu trop petite.

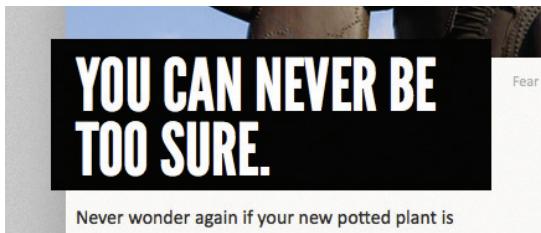


**FIG 4.12 :** Moins sexy que League Gothic ? Pas étonnant. Mais c'est tout de même plus lisible et ça colle avec le design.



Notre première règle `.main-title` s'applique toujours à tous les navigateurs qui lisent notre CSS. Mais pour les navigateurs et les appareils moins larges — spécifiquement dont la largeur n'excède pas 768 pixels — la deuxième règle s'applique également, outrepas-sant la précédente. Nous avons fait deux changements notables : d'abord, nous avons donné une taille de police plus petite à l'élé-ment `.main-title`, qui passe de `3.625em` (environ `58px`) à `1.5em`, soit `24px`, une taille plus appropriée pour les petits écrans.

Deuxièmement, la police que nous utilisions initialement pour notre titre — notre chère League Gothic — ne gagne pas à être réduite (FIG 4.11). J'ai donc décidé de changer la chaîne de substitution de polices `font-family` elle-même (`Calibri`,



**FIG 4.13 :** Notre titre par défaut, et la version corrigée en dessous.



Candara, Segoe, "Segoe UI", Optima, Arial, Helvetica, sans-serif), ce qui rend le titre un peu plus lisible (FIG 4.12).

Vous avez probablement remarqué que nous n'avions pas besoin de réécrire les autres propriétés de la première règle `.main-title`. Par conséquent, la couleur d'arrière-plan noire, le `text-transform` et la couleur blanche s'appliquent toujours à nos titres miniatures. Notre requête neutralise seulement les caractéristiques dont nous ne voulons pas.

Et voilà : en appliquant rapidement une media query, on obtient un traitement qui convient bien mieux aux petits écrans (FIG 4.13).

Mais ce n'est que le début. Nous pouvons non seulement ajuster notre typographie pour qu'elle réponde aux changements de résolution, mais nous allons également pouvoir résoudre des problèmes plus sérieux.

## Penser en miniature

Commençons par étoffer notre nouvelle media query et apporter un léger changement à notre mise en page. Vous vous souvenez de notre contenu flexible `#page` au chapitre 2 ?

Voilà ce que donne pour l'instant sa CSS :

```
#page {  
    margin: 36px auto;  
    width: 90%;  
}
```

Notre contenant occupe actuellement 90 % de la largeur de la fenêtre du navigateur, et il est centré horizontalement (`margin: 36px auto`). Ça fonctionne très bien, mais ajoutons une règle dans notre media query existante pour modifier légèrement son comportement quand on passe en dessous de notre résolution initiale :

```
@media screen and (max-width: 768px) {  
    #page {  
        position: relative;  
        margin: 20px;  
        width: auto;  
    }  
}
```

En dessous de `768px`, nous demandons à l'élément `#page` d'occuper toute la largeur de la fenêtre du navigateur, en conservant une marge de `20px`. Un changement mineur, mais qui nous laissera un peu plus d'espace à basse résolution.

Notre contenant étant réglé, nous pouvons nous tourner vers le contenu.

```
@media screen and (max-width: 768px) {  
    #page {  
        margin: 20px;  
        width: auto;  
    }  
  
.welcome,  
.blog,  
.gallery {  
    margin: 0 0 30px;
```

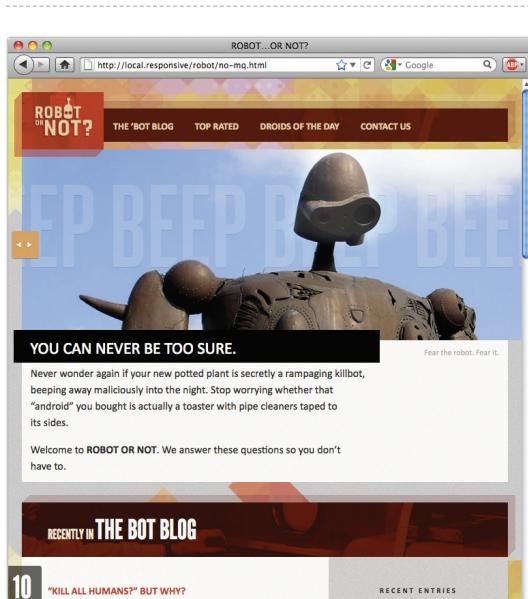
```
width: auto;  
}  
}
```

Cette nouvelle règle sélectionne les trois modules de contenu principaux — notre introduction ([.welcome](#)), le blog ([.blog](#)) et la galerie photo ([.gallery](#)) — et désactive leurs marges horizontales, afin qu'ils occupent toute la largeur de `#page`.

En deux temps trois mouvements, nous avons linéarisé notre mise en page, qui est maintenant parfaitement lisible sur un plus petit écran (FIG 4.14). C'est pas beau la vie ?

... non ? Que dites-vous ? Il reste une image totalement démesurée en haut de notre page (FIG 4.15) ?

Bon, d'accord. Je suppose que l'on peut faire quelque chose. Si ça vous pose vraiment problème, bien sûr. Mais avant toute chose, nous ferions peut-être mieux de regarder le code de cette image, conçue pour faire partie d'un module (pas encore implémenté) de diaporama.



**FIG 4.14 :** Notre contenu a été linéarisé à l'aide de deux règles supplémentaires. Chouette ! Mais quelque chose cloche encore...



**FIG 4.15 :** Plus spécifiquement, cette image d'introduction doit être retravaillée.

```

<div class="welcome section">
  <div class="slides">
    <div class="figure">
      <b></b>
      <div class="figcaption">...</div>
    </div><!-- /end .figure -->

    <ul class="slide-nav">
      <li><a class="prev" href="#">Previous</a></li>
      <li><a class="next" href="#">Next</a></li>
    </ul>
  </div><!-- /end .slides -->

  <div class="main">
    <h1 class="main-title">You can never be >
      too&nbsp;sure.</h1>
  </div><!-- /end .main -->
</div><!-- /end .welcome.section -->
```

Tout ce code pour créer un module `.welcome` qui contient notre image ainsi que le texte d'introduction qui suit (`.main`). Notre image fait partie d'un bloc `.figure`, la balise `img` elle-même étant emballée dans un élément `b`, qui servira de support au style.

Ça ne vous semble pas très catholique ? Je vois bien ce que vous voulez dire. Mais cet élément **b**, aussi absurde que cela puisse paraître, gère une bonne partie de la mise en page pour nous. Voici la CSS correspondante :

```
.slides .figure b {  
    display: block;  
    overflow: hidden;  
    margin-bottom: 0;  
    width: 112.272727%; /* 741px / 660px */  
}  
  
.slides .figure b img {  
  
    display: block;  
    max-width: inherit;  
}
```

Tout d'abord, nous avons donné la valeur `hidden` à la propriété `overflow` de l'élément **b**, créant un contenant qui rogne le contenu qui dépasse. Mais pour le moment, nos images flexibles se redimensionnent en même temps que l'élément **b**, ce qui empêche cet effet de s'appliquer. Nous avons donc désactivé la règle `max-width: 100%` pour les images de notre diaporama (`max-width: inherit`). Résultat, notre grosse image de robot sera simplement rognée si elle est plus large que l'élément **b** qui la contient.

Vous avez peut-être remarqué que la largeur de notre élément **b** est en fait supérieure à 100 %. Nous avons utilisé notre vieille formule `cible ÷ contexte = résultat` pour créer un élément plus large que le module `.welcome`, permettant à l'image intégrée de se prolonger un peu sur la droite.

Mais avec ma chance habituelle, aucun de ces effets ne marche particulièrement bien à basse résolution. (Précision : j'ai une chance épouvantable). Alors peaufinons la fin de notre media query :

```
@media screen and (max-width: 768px) {  
    .slides .figure b {
```

```

width: auto;
}

.slides .figure b img {
  max-width: 100%;
}
}

```

La première règle donne à notre contenant `b` la largeur `auto`, c'est-à-dire la même largeur que son contenant. La seconde règle rétablit la règle `max-width: 100%` dont nous avons parlé au chapitre 3, notre image se redimensionnera donc de nouveau en même temps que son contenant. Ensemble, ces deux règles alignent notre image rebelle avec son contenant et par extension, avec le reste de notre design (FIG 4.16). Je ne sais pas pour vous, mais je me sens déjà soulagé.

Mais il nous reste encore un problème à régler avant de mettre nos doigts de pied en éventail. Vous voyez notre barre de liens en haut ? Elle est toujours beaucoup trop étroquée. De plus, si l'on rétrécit un peu la fenêtre, elle ne tient même plus sur une seule ligne (FIG 4.17).



**FIG 4.16 :** Notre image a retrouvé sa place. Ouf.

Le code de notre en-tête est plutôt simple :

```
<h1 class="logo">
```



FIG 4.17 : « Contact Us », pourquoi tant de haine ?

```
<a href="/">
  <i></i>
</a>
</h1>

<ul class="nav nav-primary">
  <li id="nav-blog"><a href="#">The &#8217;Bot Blog</a>
    </li>
  <li id="nav-rated"><a href="#">Top Rated</a></li>
  <li id="nav-droids"><a href="#">Droids of the Day</a>
    </li>
  <li id="nav-contact"><a href="#">Contact Us</a></li>
</ul><!-- /end ul.nav.nav-primary -->
```

Je ne vous avais pas menti : une balise `h1` pour notre logo et une liste non ordonnée pour les liens. Comme je me sentais particulièrement imaginatif, je leur ai donné respectivement les classes `.logo` et `.nav-primary`. Mais qu'en est-il de la CSS ?

```
.logo {  
    background: #C52618 url("logo-bg.jpg");  
    float: left;  
    width: 16.875%; /* 162px / 960px */  
}  
  
.nav-primary {  
    background: #5E140D url("nav-bg.jpg");  
    padding: 1.2em 1em 1em;  
}  
  
.nav-primary li {  
    display: inline;  
}
```

Les styles sont plutôt modestes. On applique des images d'arrière-plan aux deux éléments, mais la mise en page elle-même est plutôt succincte : on fait flotter l'image à gauche en la faisant chevaucher la barre de navigation. Et les éléments de notre liste `.nav-primary` héritent simplement de la règle `display: inline`. Et pourtant, ça marche — du moins jusqu'à ce que notre page soit trop étroite pour afficher tous les éléments sur la même ligne.

Ajoutons une petite media query :

```
@media screen and (max-width: 768px) {  
    .logo {  
        float: none;  
        margin: 0 auto 20px;  
        position: relative;  
    }  
  
.nav-primary {  
    margin-bottom: 20px;  
    text-align: center;  
}  
}
```

On a désactivé le `float` initialement défini pour notre `.logo` afin de centrer horizontalement celui-ci au-dessus de notre menu.

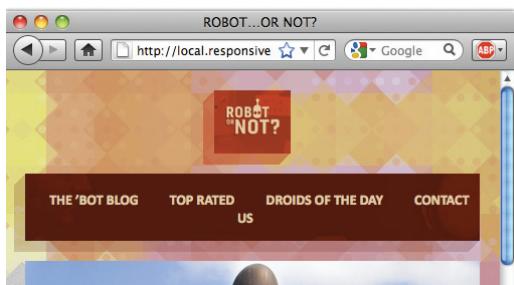
---

Et on a donné la propriété `text-align: center` à `.nav-primary`, ce qui centre tous les liens dans la barre de navigation. Ce sont des modifications plutôt mineures, mais le changement, lui, est plutôt remarquable (FIG 4.18). Le logo et les liens apparaissent sur deux lignes séparées, mais suivant l'ordre de priorité approprié.

Personnellement, je suis plutôt satisfait du résultat, mais tout n'est pas encore parfait. Si l'on regarde notre barre de navigation, on constate que les liens sont encore trop resserrés. En fait, si l'on réduit la taille de notre écran, même un tout petit peu, on fait face au même problème que précédemment (FIG 4.19).



**FIG 4.18 :** On peut radicalement réorganiser notre en-tête pour les plus petites résolutions et donner à notre logo et à nos liens un peu d'air frais.



**FIG 4.19 :** Bon, ça commence à être lourd.

(Il semblerait que je sois en croisade contre les sauts de ligne.  
Je ne sais pas pourquoi.)

Nous avons découvert un nouveau point de rupture, qu'on ne peut pas régler simplement en déplaçant le logo au-dessus de notre barre de navigation. Créons donc une autre media query pour parer à toute éventualité :

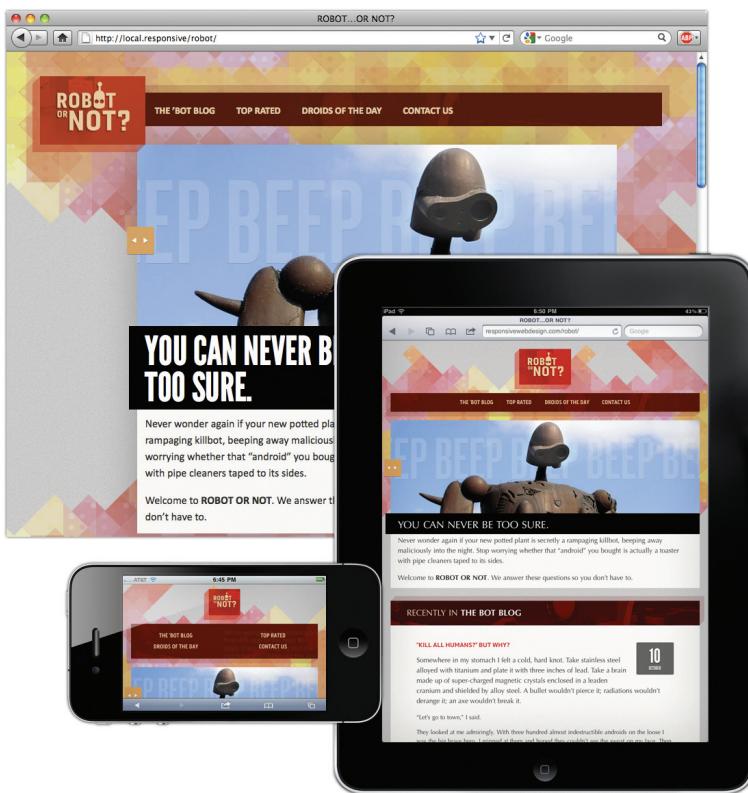
```
@media screen and (max-width: 768px) {  
    ...  
}  
  
@media screen and (max-width: 520px) {  
    .nav-primary {  
        float: left;  
        width: 100%;  
    }  
  
    .nav-primary li {  
        clear: left;  
        float: left;  
        width: 48%;  
    }  
  
    li#nav-rated,  
    li#nav-contact {  
        clear: right;  
        float: right;  
    }  
  
    .nav-primary a {  
        display: block;  
        padding: 0.45em;  
    }  
}
```

Pour les écrans encore plus petits (spécifiquement ceux qui font moins de 520 pixels de large), on fait flotter chaque `li` dans `.nav-primary`, en choisissant `float: right` pour le deuxième et le quatrième lien. On obtient au final une grille de deux sur deux avec nos liens, qui résistera mieux aux changements de la taille de la fenêtre que notre approche employant `display: inline` (FIG 4.20).

Remarquons tout de même qu'on n'a pas eu à réécrire les règles de notre requête précédente (`screen and (max-width: 768px)`) dans celle-ci. C'est parce que les écrans qui répondent à notre nouvelle condition « moins de `520px` de large » répondent



**FIG 4.20 :** Je ne devrais probablement pas vous dire à quel point je suis heureux que nos liens soient enfin bien rangés. Alors je vais m'abstenir.



**FIG 4.21 :** Notre responsive design prend forme, sur tous les supports.

également à la condition « moins de `768px` de large ». En d'autres termes, les règles des deux requêtes s'appliquent à la plus petite résolution du spectre. Par conséquent, notre deuxième requête doit s'attacher à résoudre les problèmes de design qui affectent uniquement les fenêtres de moins de `520px`.

Et nous y voilà (FIG 4.21). En apportant quelques corrections supplémentaires à la mécanique interne de notre page, nous avons finalement un design qui répond au contexte dans lequel il est visualisé. Nous ne sommes plus bloqués sur la grille, la mise en page ou la typographie que nous avons conçue à l'origine pour une plage de résolutions spécifique. En ajoutant des media queries par-dessus nos mises en page flexibles, on peut résoudre les problèmes de design qui surviennent sur les écrans plus petits.

### Cette mise en page-là va jusqu'à 11

Mais le design Web réactif ne consiste pas simplement à créer des designs accessibles aux plus petits écrans. Vous vous souvenez peut-être que notre design présentait un nombre important de problèmes quand il était visualisé dans une fenêtre maximisée : les images prenaient une taille incongrue alors que les lignes de texte devenaient trop longues pour être confortablement lues ; en somme, notre grille s'étirait au-delà des limites de l'utile (FIGS 4.6-4.8). Bien sûr, nous pourrions imposer une sorte de limite externe à notre design, éventuellement avec une règle `max-width` en `ems` ou en `pixels`. Mais nous allons plutôt saisir cette opportunité pour créer un design adapté à une autre plage de résolutions.

Commençons par introduire une nouvelle media query pour ce faire :

```
@media screen and (max-width: 768px) {  
    ...  
}  
  
@media screen and (max-width: 520px) {  
    ...  
}
```

```
@media screen and (min-width: 1200px) {  
    ...  
}
```

Notre première media query utilise un palier de 768 pixels : en d'autres termes, les appareils et les navigateurs dont la largeur excède la limite `max-width` ignoreront tout simplement la CSS incluse. Nous avons ensuite ajouté une requête pour les résolutions encore plus basses, de nouveau avec `max-width`.

Au contraire, pour notre prochaine media query, nous utilisons `min-width` pour définir une résolution minimale de `1200px` de large. Si l'appareil ou le navigateur accédant à notre site fait plus de 1200 pixels de large, il appliquera le style inclus dans cette requête ; sinon, il ignorerá la CSS et continuera gaie-ment son chemin.

Alors remontons nos manches et attaquons-nous à la création d'une mise en page adaptée aux grands écrans :

```
@media screen and (min-width: 1200px) {  
    .welcome,  
    .blog,  
    .gallery {  
        width: 49.375%;  
    }  
  
.welcome,  
.gallery {  
    float: right;  
    margin: 0 0 40px;  
}  
  
.blog {  
    float: left;  
    margin: 0 0 20px;  
}  
}
```

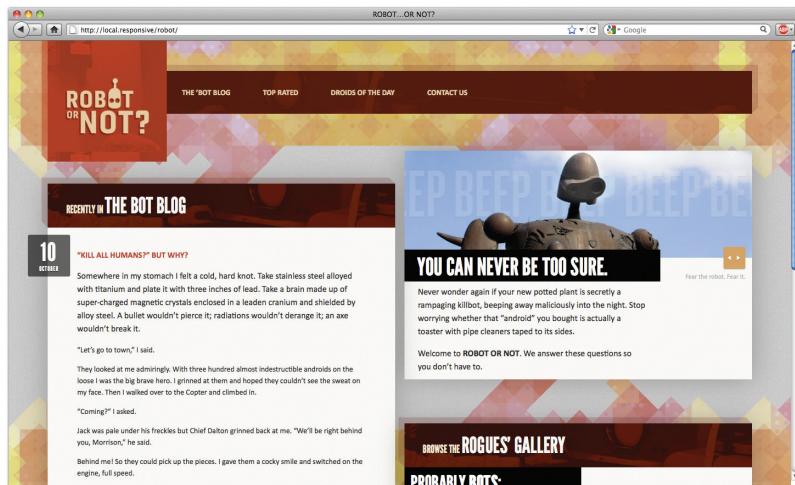


FIG 4.22 : Nous avons retravaillé notre design à l'intention de nos lecteurs sur grand écran, tout cela avec une simple media query.

Sur le site en ligne (<http://responsivewebdesign.com/robot>), vous verrez que de nombreux autres changements se produisent sur grand écran, mais ces trois règles sont les plus cruciales. On prend nos trois modules de contenu principaux (`.welcome`, `.blog` et `.gallery`), et on leur donne la largeur de la moitié de la page environ (49.375%). Ensuite, on fait flotter les modules `.welcome` et `.gallery` sur la droite, et le blog sur la gauche. Le résultat ? Un design parfaitement adapté à la lecture sur grand écran (FIG 4.22). Nos lignes de texte ont retrouvé une longueur acceptable et le blog — le morceau de contenu essentiel — est remonté d'un cran, ce qui le rend nettement plus accessible.

En un mot, notre responsive design est fini.

## AU SUJET DE LA COMPATIBILITÉ

Après vous avoir parlé des media queries en long et en large, il est temps d'arrêter de rêver. Euh, je veux dire, il est temps de parler de la compatibilité des navigateurs.

La bonne nouvelle ? Les media queries sont remarquablement bien supportées par les navigateurs de bureau modernes.

Opera supporte les media queries depuis la version 9.5, ainsi que Firefox 3.5+ et les navigateurs utilisant le moteur WebKit, comme Safari 3+ et Chrome. Même Internet Explorer 9 (<http://bkaprt.com/rwd/32/>) supporte les media queries (<http://bkaprt.com/rwd/33/>) ! Pincez-moi.

Les choses se présentent bien également pour les appareils mobiles. Les navigateurs mobiles basés sur WebKit, comme Mobile Safari, le webOS de HP et le navigateur d'Android supportent tous les media queries. Et comme le signale Peter-Paul Koch (<http://bkaprt.com/rwd/34/>), il semblerait qu'Opera Mobile et Opera Mini suivent le mouvement, tout comme Mozilla. Si l'on ajoute à cela le Windows Phone qui utilisera IE9 (<http://bkaprt.com/rwd/35/>) en 2011, on remarque que le support des media queries est de plus en plus répandu, ce qui est très prometteur.

Malheureusement, « répandu » ne veut pas dire « universel ». Les navigateurs de bureau plus anciens ne risquent pas d'être compatibles. Et non, Internet Explorer 8 et inférieur ne supportent pas les media queries, ce qui signifie que le vénérable IE6 pose toujours un sérieux problème. Et même si de nombreux appareils mobiles modernes sont globalement compatibles, certains navigateurs très répandus, comme IE Mobile et ceux des vieux BlackBerry, ne comprennent pas les media queries (<http://bkaprt.com/rwd/36>).

Tout n'est pas rose, mais cela ne veut pas dire que les mises en page adaptatives ne sont qu'un voeu pieux. Tout d'abord, il existe de nombreuses solutions en JavaScript pour corriger la prise en charge défaillante des navigateurs plus anciens. C'est exactement ce qu'une librairie au nom évocateur, [css3-media-queries.js](http://bkaprt.com/rwd/37) (<http://bkaprt.com/rwd/37>), se propose de faire : « permettre à IE5+, Firefox 1+ et Safari 2 de parser, tester et appliquer de manière transparente des media queries CSS3 ». C'est une version très préliminaire, et qui n'a pas connu beaucoup de développements, mais elle m'a été très utile.

Ces derniers temps, j'utilise un script appelé [respond.js](http://bkaprt.com/rwd/38) (<http://bkaprt.com/rwd/38>), une petite librairie bien pratique développée par Scott Jehl. Là où [css3-mediaqueries.js](http://bkaprt.com/rwd/37) comprend de nombreuses fonctionnalités, Respond apporte seulement le support des requêtes `min-width` et `max-width` aux vieux navigateurs. Et c'est tout ce dont j'ai besoin pour la plupart des media

queries que je programme. Mentionnons tout de même que `respond.js` utilise un petit bidouillage pour marcher aussi rapidement, ce qui nécessite d'ajouter un commentaire CSS spécialement formaté à la fin de chaque media query, comme ceci :

```
@media screen and (max-width: 768px) {  
    ...  
} /*/mediaquery*/  
  
@media screen and (max-width: 520px) {  
    ...  
} /*/mediaquery*/  
  
@media screen and (min-width: 1200px) {  
    ...  
} /*/mediaquery*/
```

Pourquoi ce commentaire supplémentaire ? Eh bien, `css3-mediaqueries.js` utilise beaucoup de code pour comprendre la structure des feuilles de style : il peut ouvrir notre CSS et immédiatement faire la différence entre l'accolade qui ponctue une règle CSS et celle qui ferme un bloc `@media`. Respond s'en fiche pas mal : en cherchant plutôt notre petit commentaire, il peut traiter nos requêtes plus rapidement que n'importe quel autre script.

En fait, en ajoutant ce commentaire à la fin des media queries et en plaçant la librairie `respond.js` dans le `head` de notre page, on obtient une mise en page réactive qui marche au poil, même dans les vieux navigateurs comme IE7 (FIG 4.23).

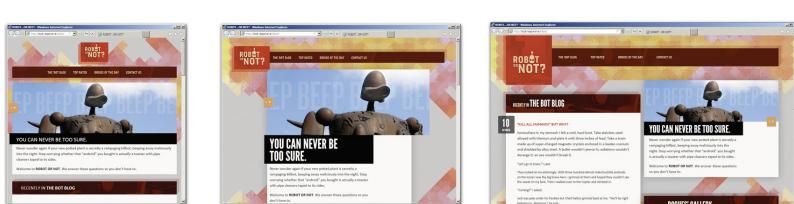


FIG 4.23 : Avec notre correctif en JavaScript en place, les navigateurs plus anciens comme IE ont un semblant de compatibilité avec les media queries.

Je ne suis pas du genre à me reposer sur le JavaScript, et je vous suggère de suivre la même approche. Vous pourrez me donner toutes les statistiques du monde, il n'y a tout simplement aucun moyen de garantir qu'un utilisateur ait activé JavaScript dans son navigateur. Il travaille peut-être sur un ordinateur bridé par des mesures de sécurité draconiennes. Sans même parler des appareils mobiles, connus pour offrir un support mince, voire inexistant dans de nombreux cas.

Dans l'espoir de régler ces problèmes, nous passerons un peu de temps au chapitre 5 à détailler des alternatives moins dépendantes de JavaScript. Je comprends parfaitement que des correctifs en JavaScript ne vous fassent pas rêver. Cela souligne cependant la nécessité de construire votre responsive design sur des fondations flexibles, afin de vous assurer qu'il est raisonnablement accessible sur des appareils et des résolutions variés.

## POURQUOI LA FLEXIBILITÉ ?

Si vous me permettez de faire ma groupie : les media queries c'est que du bonheur ! Elles nous permettent de servir du CSS3 conditionnellement selon les capacités de l'appareil qui accède à notre site, donc de concevoir des designs correspondant mieux à l'environnement de lecture de nos utilisateurs.

Cependant, les media queries ne suffisent pas à produire un responsive design. Un design véritablement « responsive » commence par une mise en page flexible, sur laquelle on ajoute une couche de media queries. De nombreux arguments vont dans ce sens : notamment, une mise en page flexible peut servir de solution de secours pour les appareils et les navigateurs qui ne supportent ni JavaScript, ni les media queries.

Mais ce n'est pas la seule raison. L'éditeur de logiciels 37signals a récemment commencé à expérimenter avec un responsive design pour une de leurs application. Voilà ce qu'ils en disent (<http://bkapr.com/rwd/39/>) :

*Il s'est avéré qu'il suffisait d'ajouter quelques media queries CSS au produit fini pour que la mise en page fonctionne sur de nombreux appareils. La clé pour que ce soit simple, c'est que la mise en page soit déjà liquide. Ainsi, l'optimiser pour les petits écrans consistait à réduire*

*quelques marges pour maximiser l'espace et à légèrement modifier la mise en page quand l'écran est trop étroit pour afficher deux colonnes.*

En clair, en partant de fondations flexibles, on a moins de code à produire. Quand on travaille avec des media queries, une mise en page fixe doit souvent être reprogrammée pour chaque résolution, alors qu'un design conçu avec des pourcentages conserve ses proportions d'une résolution à l'autre. Comme nous l'avons vu dans ce chapitre, on peut supprimer et modifier des propriétés au cas par cas pour chaque point de rupture, et ainsi optimiser notre mise en page avec quelques modifications rapides.

De plus, une mise en page flexible est mieux préparée aux appareils qui ne sont pas encore sortis. Il y a un an encore, le mot « tablette » évoquait l'image de l'iPad. Mais aujourd'hui, on voit arriver des tablettes de sept pouces comme le Samsung Galaxy Tab, et d'autres appareils comme le Kindle et le Nook embarquent aussi de fabuleux petits navigateurs. On ne peut tout simplement pas suivre le rythme du marché. Des fondations flexibles nous permettent de cibler des résolutions individuelles et de mieux préparer nos designs aux appareils qui n'ont même pas encore été imaginés.

## Établissez des contraintes s'il le faut

Cela dit, personne ne connaît mieux votre design — et ses utilisateurs — que vous. Si vous pensez que le fait de placer `max-width` sur un élément vous aidera à préserver son intégrité, allez-y. Voici un autre extrait des expériences réactives de 37signals (<http://bkapr.com/rwd/39/>) :

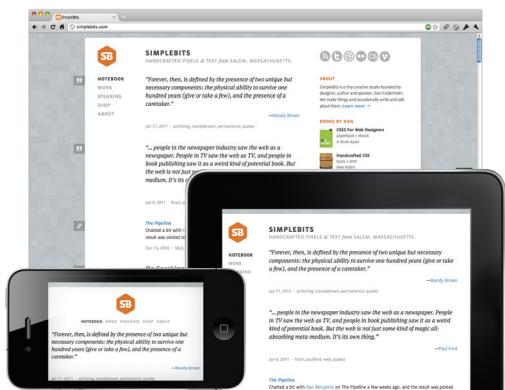
*La propriété CSS max-width semble presque avoir été oubliée au fond de la boîte à outils du designer parce qu'elle n'était pas supportée par Internet Explorer 6. Cette restriction étant levée, cette propriété est le complément idéal d'une mise en page liquide, qui permet au contenu de se mouvoir naturellement selon la largeur de la page, sans s'étirer au point de l'absurdité et gêner la lecture. C'est un excellent compromis entre une mise en page liquide et une mise en page fixe.*

Je travaille actuellement sur un projet de refonte « réactif » pour lequel une question similaire s'est posée. Le design a une

**max-width** fixe de **1200px**, mais est complètement flexible en-dessous. Mais alors, pourquoi ne pas le rendre intégralement fluide ? On a passé pas mal de temps à examiner la réaction de notre page au-delà certains points de rupture. Et les media queries que nous avons mises en place reflètent cette approche, en garantissant que le site sera aussi plaisant à lire sur la dernière version de Chrome que sur un téléphone Android ou le navigateur du Kindle. Mais en fin de compte, on a décidé que le public du site ne justifiait pas de mobiliser le temps et les ressources nécessaires à la création d'un design convaincant pour grand écran. On a donc décidé d'introduire notre contrainte **max-width**.

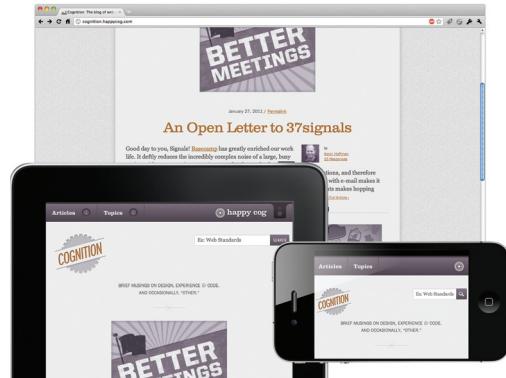
Que dites-vous ? Vous aimerez voir quelques exemples en action du mariage de **max-width** et des media queries ? Dans ce cas, je ne peux passer à côté du site de Dan Cederholm (<http://simplebits.com>) et du blog officiel de l'agence de design Happy Cog (<http://cognition.happycog.com>) (FIGS 4.24 ET 4.25). Ce sont deux parfaits exemples de cette approche hybride de mise en page flexible, basés sur une grille fluide restreinte par une contrainte **max-width** en pixels.

Certains designers préfèrent cette méthode, arguant que des lignes trop longues sont pénibles à lire. Et franchement, ils ont raison — mais la propriété **max-width** n'est qu'une des solutions à ce problème. Prenez le site du designer et illustrateur Jon Hicks (FIG 4.26), l'une des premières refontes « responsive » lancée en 2010 (<http://bkapr.com/rwd/40/>).

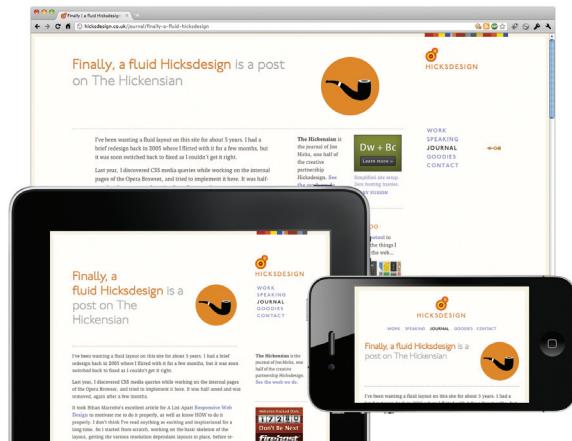


**FIG 4.24 :** Dan Cederholm, le designer Web des designers Web, a décidé de donner une **max-width** de 960 pixels à son nouveau responsive design. Et vous savez quoi ? Ça marche.

**FIG 4.25 :** Les talentueux chenapans de Happy Cog ont récemment lancé un design de blog réactif, en décidant qu'une `max-width` de 820 pixels serait appropriée. Le résultat ? Trop meugnon !



**FIG 4.26 :** Le site réactif de Jon Hicks est complètement flexible, et superbe dans n'importe quelle résolution.



Bien qu'il n'y ait pas de contrainte `max-width` dans son design, Jon a légèrement modifié la typographie pour chaque plage de résolutions afin que son site soit toujours un plaisir à lire, le tout sans placer de contrainte sur son design (FIG 4.27).

En d'autres termes, la flexibilité n'a pas à être un handicap. Elle peut au contraire être une nouvelle occasion de mettre notre art à l'épreuve, de mieux communiquer avec certains types d'utilisateurs, ou de résoudre des problèmes qui affectent un type d'appareil particulier.

I've been wanting a fluid layout on a brief redesign back in 2005 where months, but it was soon switched back right.

Last year, I discovered CSS media queries internal pages of the Opera Browser. It was half-assed and was removed,

It took Ethan Marcotte's excellent article [Responsive Web Design](#) to motivate me to know HOW to do it properly. I do find exciting and inspirational for a long time working on the basic skeleton of the resolution dependent layouts in place design (making a few changes long time)

I've been wanting a fluid layout on a brief redesign back in 2005 where months, but it was soon switched back right.

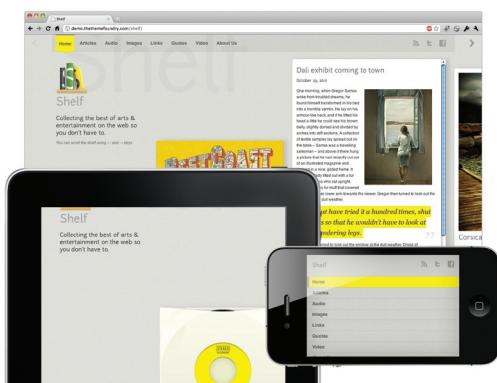
Last year, I discovered CSS media queries internal pages of the Opera Browser. It was half-assed and was removed,

It took Ethan Marcotte's excellent article [Responsive Web Design](#) to motivate me to know HOW to do it properly. I do find exciting and inspirational for a long time working on the basic skeleton of the resolution dependent layouts in place design (making a few changes long time)

**FIG 4.27 :** Plutôt que de se reposer sur `max-width`, Jon a choisi d'ajuster sa typographie à la résolution, ce qui rend son blog agréable à lire dans toutes les circonstances.

Mais voilà, c'est le genre de décisions que l'on prend constamment en tant que designers, en favorisant soit la flexibilité, soit le contrôle. Cependant, ce que le responsive design nous apprend, c'est que ce n'est pas nécessairement une proposition binaire ; on peut avoir un design basé sur une mise en page flexible, tout en incluant des éléments à largeur fixe (FIG. 4.28). Ainsi, si mon client décide un jour qu'une mise en page pour grand écran rendrait service à son public, il lui suffira d'enlever la contrainte `max-width` et de créer une nouvelle media query pour élaborer un nouveau design convaincant.

Pour résumer, nos designs peuvent s'adapter aux besoins de nos utilisateurs en un claquement de doigts.



**FIG 4.28 :** Le thème réactif « Shelf » de Jon Hicks pour WordPress et Tumblr (<http://bkapr.com/rwd/41/>) offre une magnifique mise en page flexible, mais comprend des conteneurs à largeur fixe pour certains types d'entrées. (J'adore ce défilement horizontal !)

# **PASSER AU RESPONSIVE DESIGN**

*“ La Voie tire sa forme de l’usage,  
Mais la forme disparaît.  
Ne t’attache pas aux formes  
Mais laisse la sensation parcourir le monde  
Comme la rivière vers l’océan.*

— DAO DE JING, SECTION 32, « FORMES »

VOUS AVEZ MAINTENANT TOUS LES OUTILS NÉCESSAIRES pour commencer à construire des mises en page réactives. Vous maîtrisez l'approche proportionnelle de la grille flexible, vous avez étudié quelques stratégies pour incorporer des médias à largeur fixe dans vos designs, et appris à vous servir des media queries pour adapter vos designs à toutes sortes d'appareils.

Mais jusqu'ici, on a pour ainsi dire fait du responsive design sous vide. Dans ce chapitre, nous allons voir plusieurs manières différentes de commencer à l'incorporer dans notre travail, ainsi que plusieurs pistes pour améliorer certaines des techniques dont nous avons déjà parlé.

## UNE QUESTION DE CONTEXTE

Au fil de vos expérimentations, vous remarquerez qu'un responsive design bien conçu peut offrir à vos visiteurs un haut niveau de continuité entre différents contextes. C'est parce qu'au niveau le plus basique, le responsive design consiste à servir un même document HTML à d'innombrables navigateurs et appareils, en utilisant des mises en page flexibles et des media queries pour que le design soit aussi portable et accessible que possible.

Cependant, certains designers Web plaident contre cette approche, suggérant qu'il faudrait une page séparée pour chaque type d'appareil. Dans un long billet, le développeur mobile James Pearce met en doute les mérites du responsive design (<http://bkaprt.com/rwd/42/>) :

*Le fait que l'utilisateur ait un petit écran entre les mains est une chose — le fait qu'il le tienne effectivement dans ses mains en est une autre. Le fait que l'utilisateur marche, conduise ou soit allongé en est encore une autre. En fait, l'utilisateur a probablement besoin d'un contenu et de services différents — ou d'une version aux priorités différentes par rapport à l'expérience par défaut.*

Jeff Croft (<http://bkaprt.com/rwd/43/>) le dit beaucoup plus succinctement :

*Globalement, les utilisateurs d'appareils mobiles attendent des choses différentes de votre produit que les utilisateurs de « bureau ». Si vous êtes un restaurant, les utilisateurs de bureau voudront probablement voir des photos de votre établissement, un menu complet et des informations sur son histoire. Mais il est probable que les utilisateurs de mobiles veuillent simplement votre adresse et les horaires d'ouvertures.*

Cet argument comporte deux facettes : d'abord, l'appareil implique un contexte, qui nous dit si l'utilisateur est stationnaire ou mobile. À partir de ce contexte, on peut créer une classe d'utilisateurs et en déduire ses objectifs. En clair, les utilisateurs mobiles voudront un accès plus rapide à certaines

tâches que s'ils étaient sur un ordinateur, quand le temps et la bande passante jouent en leur faveur.

Ensuite, si les priorités et les objectifs de l'utilisateur diffèrent effectivement d'un contexte à l'autre, le fait de servir le même document HTML à tout le monde ne peut tout simplement pas fonctionner. Prenez l'exemple de Jeff : si le site du restaurant comporte de très grandes photos en tête de chaque page, un visiteur mobile devra passer un temps considérable à la faire défiler ne serait-ce que pour trouver les horaires d'ouverture.

Pour ce que ça vaut, je suis d'accord avec ces arguments — mais jusqu'à un certain point. On peut tout à fait extrapoler le contexte d'un utilisateur en fonction de son appareil, mais cela reste une extrapolation. Par exemple, l'essentiel de ma navigation « mobile » se fait depuis le canapé de mon salon, connecté à mon réseau wifi, pour tuer le temps. Et ce n'est pas juste pour dire que je n'ai pas de vie : des études montrent que de nombreux utilisateurs du « Web mobile » y accèdent bien au chaud chez eux (<http://bkaprt.com/rwd/44/>, <http://bkaprt.com/rwd/45/>).

Cela ne veut pas dire que la question du contexte n'a aucune valeur, ou que nous ne devrions pas réfléchir à ces questions épineuses. Mais nous ne pouvons tout simplement pas deviner le contexte d'un utilisateur en fonction de l'appareil qu'il utilise — et la plupart du temps, l'implémentation de ces sites distincts semble incomplète (FIG 5.1). Se reposer sur des termes un peu trop vagues comme « mobile » et « bureau » ne peut pas se substituer à de vraies études sur la manière dont vos utilisateurs accèdent à votre site : non seulement les appareils et les navigateurs qu'ils utilisent, mais aussi comment, où et pourquoi ils les utilisent.

Mais il faut souligner que le responsive design n'est pas conçu pour remplacer les sites Web mobiles. Le responsive design, c'est, selon moi, 50 % de philosophie du design et 50 % de stratégie de développement d'interface. Et en tant que stratégie de développement, il doit être analysé pour déterminer s'il répond aux besoins du projet sur lequel vous travaillez. Vous avez peut-être une très bonne raison de séparer vos expériences bureau et mobile, ou peut-être que votre contenu serait mieux servi par une approche « responsive ». Seul vous et vos utilisateurs pouvez le savoir.

Même si je partage l'avis des designers Web mobiles quand ils affirment que certains utilisateurs de certains sites ont besoin



**FIG 5.1 :** Sur un iPad, Google Reader et Twitter affichent par défaut leurs sites « mobiles ». Super design, mais est-ce vraiment le bon contexte ?

d'un contenu différent, je pense que l'inverse est également vrai : pour de nombreux sites, il est préférable d'utiliser le même document pour différents contextes ou appareils. Et ces sites-là sont des candidats idéaux pour une approche « responsive ».

Alors comment savoir si une telle approche convient à votre projet ?

## Connaisez les objectifs de vos utilisateurs

Début 2010, j'ai travaillé sur un site appelé Cog'aoke (FIG 5.2), conçu pour faire la promotion d'un karaoké organisé par mon employeur d'alors. Son but principal était de fournir aux visiteurs des informations sur la fête, ses sponsors et le lieu où elle se tiendrait. Mais il y avait également une application : les visiteurs pouvaient s'inscrire pour chanter lors de l'événement, parcourir le catalogue de chansons et voter pour d'autres performers potentiels.

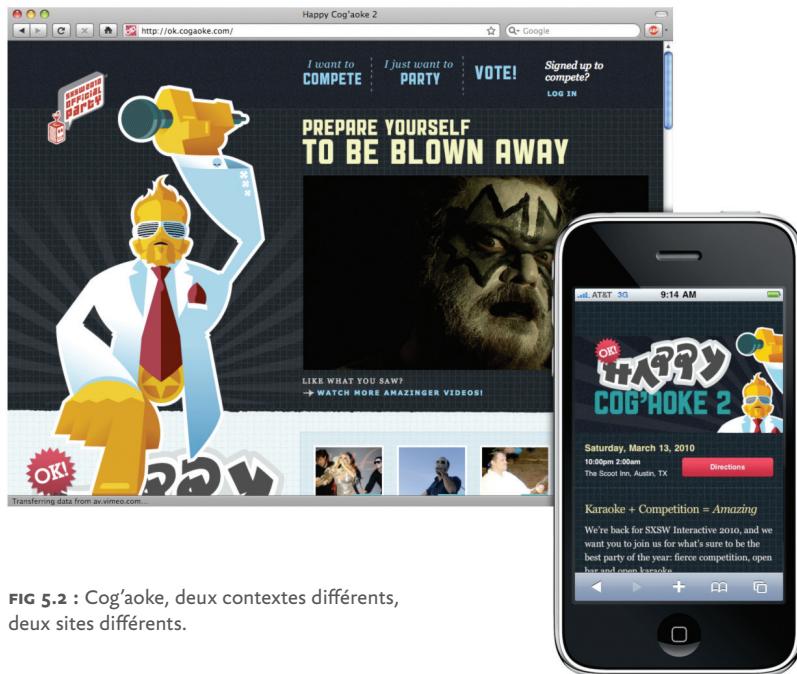


FIG 5.2 : Cog'aoke, deux contextes différents, deux sites différents.

Nous avions également décidé que le site devait être accessible aux appareils mobiles, mais nous avons imaginé quelque chose de complètement différent du site par défaut. Nous nous sommes dit que les gens qui viendraient à notre événement auraient besoin d'un accès rapide et direct à l'itinéraire. De plus, il devait y avoir un vote en direct, et nous voulions inviter les spectateurs à voter sur notre site avec leur téléphone portable.

Alors que nous concevions le site, il nous a été utile de penser le site pour bureau comme l'expérience préalable au jeu. Le site mobile, d'un autre côté, était en fait conçu pour la nuit de l'événement, pour les personnes qui étaient physiquement présentes. Les buts des deux contextes n'auraient donc pas pu être plus distincts.

Dans cet esprit, il aurait bien sûr été possible d'inclure tout le balisage pour chaque contexte sur chaque page du site. Si nous avions emprunté cette voie, chaque page aurait inclus le contenu

par défaut pour « bureau », ainsi que la carte, l’itinéraire et le système de vote pour le site mobile. Et avec ces deux modes dans chaque page HTML, nous aurions pu utiliser des media queries et la propriété `display: none` pour offrir le bon site au bon appareil.

Mais cela n’aurait pas été la bonne approche. Nous avons réalisé qu’il était irresponsable de demander à nos visiteurs de télécharger tout ce HTML superflu qu’ils ne verrraient pas et dont ils n’avaient que faire. Et je ne dis pas ça parce que je me soucie plus particulièrement des visiteurs mobiles : qu’ils soient sur un téléphone ou un ordinateur de bureau, nos utilisateurs seraient pénalisés par ce surplus.

## MOBILE FIRST

Quand vous aurez un moment de libre (et un triple whisky sec, avec modération), je vous recommande de parcourir l’album Flickr « Noise to Noise Ratio » de Merlin Mann (<http://bkaprt.com/rwd/46>). Ces captures d’écran présentent certaines des pages les plus saturées de contenu du Web : elles sont littéralement noyées dans une mer de superflu. Et l’article lui-même, ses deux misérables paragraphes, est pratiquement introuvable.

Même si vous voyez ces sites pour la première fois, je subodore que les problèmes qu’ils exposent vous sont familiers. Je pense que cette tendance influence nos préjugés sur le design « mobile » : on suppose que les utilisateurs mobiles ont besoin de moins de contenu en partie parce que les utilisateurs de bureau peuvent en tolérer beaucoup plus. Après tout, les écrans sont plus grands, les utilisateurs sont souvent plus stationnaires et peuvent généralement mieux se focaliser sur le contenu qui les intéresse.

Mais les utilisateurs de bureau doivent-ils passer leur temps à trier le contenu, juste parce qu’ils le peuvent ? L’accès aux tâches principales ne devrait-il être facilité que pour les utilisateurs mobiles ? Pourquoi tous les utilisateurs de nos sites ne pourraient-ils pas bénéficier d’un même contenu clair et organisé ?

Fin 2009, le designer Luke Wroblewski a largué une petite bombe en suggérant que l’expérience mobile d’un site Web ne devrait pas être secondaire (<http://bkaprt.com/rwd/47>). Citant la croissance exponentielle du trafic mobile sur le Web, ainsi que les nouvelles capacités excitantes des téléphones modernes,

Luke suggère que les professionnels du Web devraient maintenant se mettre à concevoir d'abord pour les appareils mobiles.

« Mobile first » est une excellente philosophie de design. Elle m'a d'ailleurs été extrêmement précieuse pour tous les projets de responsive design sur lesquels j'ai travaillé. À l'heure où de plus en plus de navigateurs et d'appareils accèdent à nos designs, et alors que nos clients commencent à s'intéresser sérieusement à l'expérience mobile, l'occasion est idéale pour revoir sérieusement la manière dont nous concevons pour le Web : nos procédés et notre vocabulaire, mais aussi les questions que l'on pose et les solutions que l'on trouve.

## VERS UN RESPONSIVE WORKFLOW

Évidemment, ce ne sont que les débuts. De nombreux designers, studios et agences sont encore au stade de la formation. Par conséquent, il n'existe pas de « bonnes pratiques » à partager dans notre communauté. Elles évolueront au fur et à mesure que nous envisagerons notre travail de manière plus réactive. Alors en attendant, je me suis dit que je partagerais avec vous quelques expériences de travail avec un workflow plus adaptatif. Elles vous seront peut-être utiles, et (plus vraisemblablement) vous trouverez des moyens de les améliorer.

À l'heure où j'écris ces lignes, je travaille sur la refonte d'un site de grande envergure et riche en contenu. Dans la même journée, un utilisateur pourra accéder au site le matin en buvant un café à la maison, lire un ou deux articles dans les transports, et éventuellement y retourner plusieurs fois par la suite.

Vu la diversité de son lectorat, le client a décidé qu'une approche « responsive » serait la plus appropriée. Alors pendant les phases de planification, l'équipe de design a planché sur chaque élément de contenu proposé pour le site, en posant une question : en quoi ce contenu ou cette fonctionnalité est utile à nos utilisateurs mobiles ?

Bon, dit comme ça, ce n'est pas très palpitant — je n'ai jamais été très bon en marketing. Mais c'est une question qui découle de l'approche « mobile first », et qui nous a été extrêmement utile pour concevoir le site. Lisez plutôt le raisonnement de Luke sur l'utilité de cette approche lors de la phase de planification (<http://bkaprt.com/rwd/48/>) :

*Si vous pensez d'abord aux appareils mobiles, vous devez vous mettre d'accord sur ce qui compte le plus. Vous pouvez ensuite appliquer la même logique à la version bureau/ordinateur portable du site Web. On a décidé que ces fonctions et ce contenu étaient les plus importants pour nos clients et notre activité — pourquoi cela devrait-il changer quand il y a plus d'espace ?*

Il n'est que trop facile de remplir une page de barres d'outils, de liens vers les réseaux sociaux et les articles similaires, de bataillons de liens RSS et de milliers de mots-clefs. (Je crois qu'on appelle ça « valeur ajoutée ».) Mais lorsqu'on est forcé de travailler avec un écran qui est 80 % plus petit que ce à quoi on est habitué, le contenu superflu passe rapidement aux oubliettes, ce qui nous permet de nous focaliser sur les aspects critiques de nos designs.

En d'autres termes, concevoir pour les appareils mobiles en priorité peut enrichir l'expérience de tous les utilisateurs en fournissant l'élément qui manque trop souvent dans le design Web moderne : un point focal. Cela ne veut pas dire que les pages de nos clients manquent de contenu ou de fonctionnalités. Mais en encadrant le processus de design avec cette simple question, on obtient un test utile à appliquer à chaque élément proposé et chaque nouvelle fonctionnalité.

## Identifier les points de rupture

La première étape du processus consiste à étudier les différents appareils pour lesquels le design sera conçu. À partir des résultats, on compile une liste de points de rupture : les largeurs que notre design réactif devra satisfaire. Le TABLEAU 5.1 propose un exemple de liste de ce type.

Cela ne signifie pas que les résolutions inférieures ou supérieures à un seuil seront ignorées, ou que notre design ne s'adaptera pas aux appareils dont la résolution n'est pas listée. (Après tout, la mise en page adaptative sera basée sur une grille flexible et sera donc indépendante de la résolution.) Mais construire une liste de ce genre aide à définir l'étendue des travaux, nous permet d'identifier les appareils les plus fréquemment utilisés par notre public et d'utiliser leurs résolutions respectives comme critères de référence.

320 pixels	Appareils à petit écran, comme les téléphones en mode portrait.
480 pixels	Appareils à petit écran, comme les téléphones en mode paysage.
600 pixels	Petites tablettes, comme le Kindle d'Amazon (600x800) et le Nook de Barnes & Noble (600x1024) en mode portrait.
768 pixels	Tablettes de dix pouces comme l'iPad (768x1024) en mode portrait.
1024 pixels	Tablettes comme l'iPad (1024x768) en mode paysage, ainsi que certains ordinateurs portables, netbooks et ordinateurs de bureau.
1200 pixels	Écran large, principalement les ordinateurs portables et de bureau.

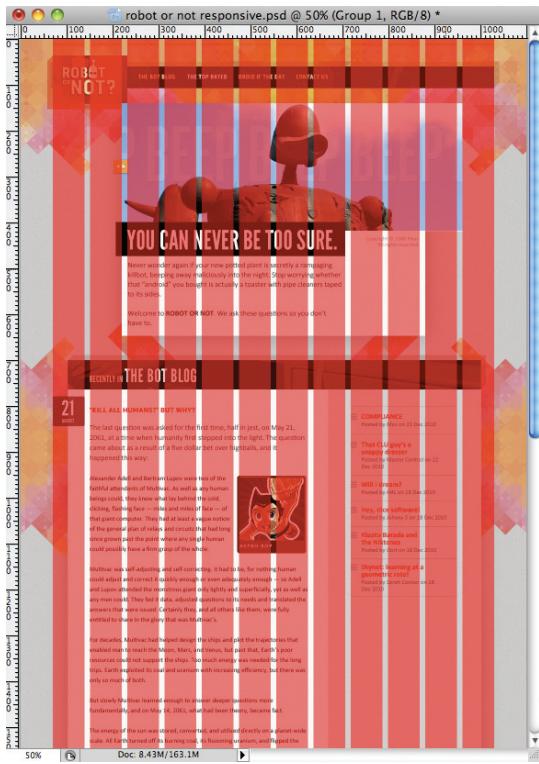
**TABLEAU 5.1 :** Exemples de points de rupture.

Cette liste en poche, il est temps de sérieusement mettre la main à la pâte.

## Design itératif et collaboratif

La plupart des projets de design suivent un modèle de gestion en « cascade », divisant chaque tâche à accomplir en phases distinctes. Les détails peuvent changer d'une agence à l'autre, mais il y a généralement quatre segments : une phase de planification, une phase de conception, une phase de développement et pour finir, la livraison du site fini. Au cours de chaque phase, des documents ou des fichiers sont créés — par exemple, un plan du site et des prototypes pendant la phase de planification —, et le client doit les approuver avant que la phase suivante ne commence.

Une fois de plus, la façon dont vous gérez vos projets peut être légèrement différente. Au cours de la phase de design, l'équipe de design réalise généralement des maquettes dans un éditeur graphique comme Photoshop, Fireworks ou autre logiciel du genre. Une fois ces maquettes bouclées et approuvées,



**FIG 5.3 :** On commence par étudier la maquette finale et poser des questions sur la manière dont elle doit se comporter sur divers appareils et navigateurs.

elles sont remises à l'équipe de développement, qui est prête à les transformer en templates HTML statiques.

Mais pour un site réactif, ce processus peut rapidement devenir lourd. Supposons un instant que vous soyez en train de reconstruire un site qui ne contient qu'une page. Vous bricolez alors une maquette dans votre application de design préférée. Mais comment communiquer à votre client l'aspect que prendra cette page sur un téléphone ? Ou un iPad ? Ou un écran large ? Si vous avez le temps, le budget et les ressources, vous pourrez peut-être concevoir chacune de ces vues alternatives, présenter ces maquettes au client, recueillir son avis puis réviser au besoin. Mais si vous avez quinze pages à créer, ou cinquante, cela peut rapidement s'avérer irréalisable.

Les projets de responsive design sur lesquels j'ai travaillé récemment se sont déroulés à merveille en combinant les phases de design et de développement en une seule phase hybride, réunissant les deux équipes au sein d'un seul groupe très collaboratif. Je parle maintenant de phase de « designancement ». (Rassurez-vous, en réalité, non.)

L'équipe de design commence par présenter une maquette à tout le groupe. C'est généralement une maquette au format bureau (FIG 5.3), mais on peut parfois partir d'une mise en page au format mobile. Le but est de définir un point de départ avec toutes les personnes concernées et de lancer le débat sur la façon dont le design devra s'adapter à différentes résolutions et différents modes de saisie. En général, les questions fusent rapidement : « Comment faire marcher ce diaporama sur une interface tactile ? » « Ce module doit-il être réduit par défaut, ou bien les utilisateurs de bureau ont-ils besoin de voir plus d'informations ? » « Comment cet élément apparaîtra (et se comportera) si JavaScript n'est pas disponible ? »

Un débat ouvert est excellent pour permettre à l'équipe de partager des idées, de définir comment le design est censé fonctionner sur des écrans différents, et d'analyser tout élément interactif particulièrement complexe. Si le groupe convient que le design doit être révisé, il l'est. Mais s'il lui semble acceptable ou si les révisions sont suffisamment mineures, l'équipe de développement récupère alors les maquettes pour en faire des prototypes.

« Des prototypes avant d'avoir le design définitif, dites-vous ? »

Absolument, je le dis. Notre objectif consiste à nous débarrasser des contraintes en pixels de Photoshop, et de commencer à construire un design qui peut s'étirer et se déformer avec la fenêtre du navigateur, et se redimensionner sur des appareils variés. L'équipe de développement commence donc rapidement à produire un responsive design, en convertissant la grille fixe en grille fluide, en déterminant la manière de gérer les différents types de médias et enfin en appliquant des media queries pour adapter notre design à diverses résolutions.

Une fois nos media queries en place, on prend le temps de redimensionner la fenêtre pour avoir une idée de ce que le design donnera dans des résolutions différentes (FIG 5.4). Les extensions de redimensionnement comme celle du module Web Developer pour Firefox et Chrome (<http://bkaprt.com/>)

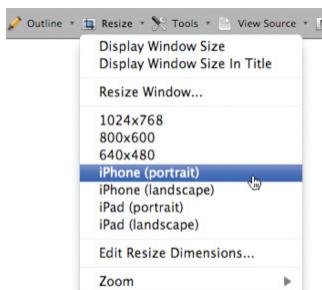


**FIG 5.4 :** Comme je le disais, une bonne façon de tester rapidement votre design consiste à redimensionner la fenêtre du navigateur. Mais ce n'est que la première étape.

[rwd/49/](#) peuvent s'avérer très utile ; si vous avez établi une liste de points de rupture comme au TABLEAU 5.1, vous pouvez simplement les enregistrer dans l'extension pour y accéder rapidement par la suite (FIG 5.5).

Mais comme nous le disions au précédent chapitre, redimensionner votre navigateur n'est qu'une étape intermédiaire. Si vous voulez tester la façon dont votre page se comporte sur un appareil donné, la meilleure solution reste encore de la consulter sur l'appareil en question. (Si l'idée d'une suite de tests mobiles vous intéresse, je vous recommande chaudement l'article de Peter-Paul Koch intitulé « Smartphone Browser Landscape », disponible sur A List Apart : <http://bkapr.com/rwd/50>. Même si vous n'avez pas l'intention d'acheter une petite armée de téléphones, c'est une excellente lecture.)

Pendant ce processus de développement, un prototype commence à prendre forme. Il est bien sûr basé sur la maquette initialement conçue par l'équipe de design, mais l'équipe de



**FIG 5.5 :** Le menu « resize » dans la barre d'outils Web Developer, avec quelques résolutions fréquemment utilisées.

développement commence à faire des recommandations sur la façon dont le design devrait se comporter sur différents appareils. En d'autres termes, les développeurs jouent aussi le rôle de designer au cours de cette collaboration ; ils conçoivent juste pour un support différent. Ils émettent des recommandations de design dans le navigateur plutôt que sous Photoshop — des recommandations qui sont partagées, testées et approuvées par toute l'équipe.

Le prototype ne doit pas forcément être testé de manière approfondie ni même être fonctionnel. Car une fois que ce modèle est plus ou moins fini, on lance un deuxième examen du design — mais cette fois-ci, les équipes de design et de développement travaillent sur du code, pas des maquettes.

## L'étude du design interactif

Pour préparer cette réunion, on charge la page prototype sur plusieurs téléphones, tablettes, ordinateurs portables et autres appareils cibles (FIG 5.6). Lorsque la réunion commence, l'équipe de développement présente la page au groupe, puis laisse chaque personne s'en servir. Durant le restant de cette étude, le groupe tout entier expérimente avec le design : sur des ordinateurs de bureau et portables, des téléphones et des tablettes. On redimensionne les fenêtres, on parcourt les galeries photos et on s'assure que les formulaires sont faciles à utiliser avec un clavier comme sur un écran tactile.

Mais pendant que tout le monde s'amuse avec le prototype, on essaie de continuer à alimenter la conversation. Il

---

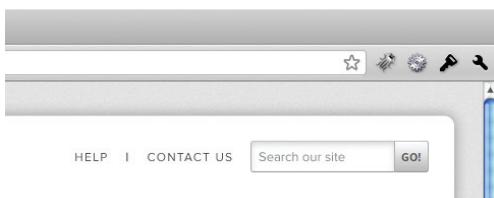
**FIG 5.6 :** Les appareils utilisés dans la suite de test jQuery Mobile (avec l'aimable autorisation du Filament Group, Inc., <http://bkapr.com/rwd/51/>.)



peut être utile que l'équipe de développement ait préparé une liste de questions qui se sont posées au cours de la création du responsive design. Quelqu'un a peut-être remarqué qu'un lien crucial est un peu trop difficile à cliquer sur un écran tactile, ou qu'une animation bouge un peu trop lentement sur un navigateur de bureau particulier. Identifier les zones d'intérêt ou les points de friction potentiels puis écouter les retours est une bonne manière de faire parler les gens sur la manière dont le design se comporte et d'avoir leur impression générale.

Parce qu'en fin de compte, le but de ces études est d'aider à accorder tout le monde sur le design définitif. La maquette initiale a servi de patron, en donnant des règles de mise en page, un guide typographique et une palette de motifs ; à partir de là, l'équipe de développement a pris la responsabilité d'adapter le design de manière plus réactive. En clair, on teste les recommandations faites par l'équipe de développement et on décide s'il faut affiner encore le design. Ce travail peut prendre la forme d'une maquette révisée ou de quelques modifications mineures du template. Et une fois la réunion finie, les deux moitiés du groupe repartent de leur côté avec leur feedback respectif, et le processus se répète. Étudier, concevoir, construire, répéter.

Voici un exemple hypothétique du fonctionnement de cette navette. Admettons que l'équipe de design ait conçu un module de navigation global, qui comprend quelques liens clés et un champ de recherche. À partir de cette maquette, l'équipe de développement a consciencieusement intégré la barre de navigation dans le template (**FIG 5.7**).



**FIG 5.7 :** La vue « bureau » de la barre de navigation globale fraîchement conçue.

Le design est plutôt simple : deux liens alignés avec le champ de recherche à leur droite. Et pour rendre le design adaptatif,

**FIG 5.8 :** Dans les plus petites résolutions, les liens étaient initialement placés sous la barre de recherche.

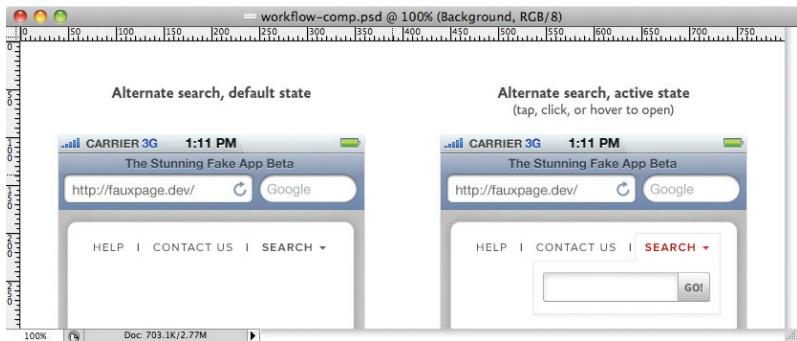


l'équipe de développement décide d'offrir une solution somme toute modeste aux petits écrans, en choisissant de dédier toute la largeur de la page à la barre de recherche et de centrer les deux liens en dessous (FIG 5.8).

Durant l'étude du design, plusieurs membres de l'équipe de design parlent de la version mobile de la barre de navigation ; le placement des éléments leur semble un peu hasardeux. La barre de recherche est beaucoup plus visible, c'est vrai, mais certains trouvent qu'elle est peut-être trop visible et fait de l'ombre aux liens placés en dessous. Et d'ailleurs, en commençant à interagir avec le design sur des téléphones tactiles, ils s'aperçoivent qu'on a vite fait de cliquer sur le champ de recherche en essayant d'accéder à un des liens.

La version de la barre de navigation qui a été programmée ne fonctionne donc pas très bien. Après discussion, l'équipe de design propose une solution alternative (FIG 5.9). Au lieu d'afficher la barre de recherche dans les plus petites résolutions, ils décident de la réduire par défaut et de lui donner le même aspect qu'un lien du menu. Mais quand on clique dessus, la barre de recherche se déroule sous le reste du menu (FIG 5.10).

Ce n'est qu'un exemple du fonctionnement possible de cette approche collaborative. L'essentiel, c'est que ce cycle design/développement soit aussi itératif que nécessaire, avec deux groupes qui affinent constamment leur travail avant de le partager lors d'un examen critique. Pour le projet sur lequel je travaille, on se réunit chaque semaine pour étudier le design



**FIG 5.9 :** Après avoir débattu des problèmes évoqués, l'équipe de design propose un design alternatif pour notre petite barre de navigation problématique.



**FIG 5.10 :** La barre de navigation fine, conçue de manière itérative par des designers et des développeurs.

interactif, mais on partage en permanence des esquisses — en design ou en code — par email.

Au final, le but est de combler le fossé qui sépare les traditionnels cycles de « design » et de « développement », et de laisser les deux groupes collaborer plus étroitement afin de produire un responsive design complet. Cette approche plus habile a permis aux groupes avec lesquels j'ai travaillé d'utiliser des applications comme Photoshop pour donner le ton et la

direction du design, mais de rapidement le traduire sur notre vrai support : le navigateur.

## ÊTRE « RESPONSIVE » ET RESPONSABLE

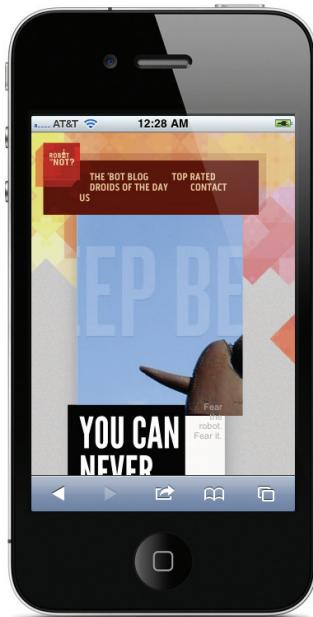
Pendant notre cycle design/développement, les pages sont constamment affinées au fur et à mesure qu'on les construit, le but étant de finir la phase en ayant des templates prêts pour la production. Et en programmant notre responsive design, la philosophie « mobile first » s'est avérée extrêmement importante.

Au fil du livre, on a utilisé le site Robot or Not pour démontrer comment une grille fluide, des images flexibles et des media queries pouvaient être combinées pour développer une approche plus réactive du design. On a commencé par prendre notre maquette rigide, conçue sous Photoshop, puis on l'a convertie en grille fluide. Comme on l'a vu au chapitre 4, cela provoque des problèmes sans fin dès que l'on commence à redimensionner la fenêtre du navigateur : notre design initial n'était pas conçu pour être redimensionné au-delà de son contexte d'origine. On a donc introduit des media queries pour résoudre ces problèmes et fournir des mises en page alternatives pour petits et grands écrans. Et finalement, pour les navigateurs qui ne prennent pas en charge les media queries à l'origine, on a inclus la librairie `respond.js` pour assurer l'accès à nos designs alternatifs.

Cependant, cette approche soulève un autre problème bien réel : que se passe-t-il si un navigateur qui ne supporte pas les media queries n'a pas non plus accès à JavaScript ? Dans ce cas, il serait forcé de restituer notre design de bureau complet, même s'il est inadapté à l'appareil. Et c'est exactement ce que de nombreux appareils mobiles verront : un design conçu pour un écran beaucoup plus large qu'on a fait rentrer avec un chausse-pied (FIG 5.11).

Et il y a un autre problème avec la façon dont on a construit le site. Voilà un extrait de la CSS :

```
.blog {  
    background: #F8F5F2 url("img/blog-bg.png") repeat-y;  
}  
  
@media screen and (max-width: 768px) {  
    .blog {
```



**FIG 5.11 :** Pas de media queries ?  
Pas de JavaScript ? Pas cool : notre mise en page flexible conçue pour un navigateur de bureau est tassée dans un petit espace.

```
background: #F8F5F2 url("img/noise.gif");  
}  
}
```

Tout d'abord, on définit l'image d'arrière-plan de l'élément `.blog`. (En l'occurrence, l'image `blog-bg.png` à deux tons que l'on a utilisée au chapitre 2 pour créer l'illusion des deux colonnes.) Pour les plus petits écrans, ceux de moins de `768px` de large, on met plutôt un GIF simple qui se répète sur l'élément `blog`, puisqu'on a linéarisé l'affichage de ces pages plus étroites.

Le problème de cette approche, c'est que certains navigateurs de téléphone, notamment Mobile Safari sur l'iPhone et l'iPad, téléchargeront les deux images, même si une seule est au final appliquée à la page. Bien que petit écran ne veuille pas forcément dire petite bande passante, on force les utilisateurs de petits écrans à télécharger une image beaucoup plus lourde que celle qui s'affichera.

Heureusement, ces problèmes ne sont pas dus au responsive design en tant que tel ; il faut simplement repenser la manière dont il est implémenté.

## « Mobile first » et media queries

Pour schématiser, le responsive design consiste à partir d'une résolution de référence, puis à utiliser des media queries pour l'adapter à d'autres contextes. Une approche plus responsable du responsive design consisterait à construire notre feuille de styles en ayant « mobile first » à l'esprit, plutôt que de créer une mise en page « bureau » par défaut. On commencerait donc par définir une mise en page adaptée aux petits écrans, puis on utiliserait des media queries pour améliorer progressivement notre design alors que la résolution augmente.

En fait, c'est l'approche que j'ai choisie pour mon portfolio personnel (<http://ethanmarkotte.com>). Par défaut, le contenu est arrangé de manière très linéaire, ce qui convient tout à fait aux appareils mobiles et aux navigateurs étroits (FIG 5.12). Mais dès que l'on élargit la fenêtre, la grille devient plus complexe et plus asymétrique (FIG 5.13). Et à l'extrême la plus élevée du spectre, le design « complet » se dévoile enfin : la mise en page est encore plus complexe, et quelques éléments plus lourds, comme cette grande image d'arrière-plan abstraite, font leur apparition (FIG 5.14).

Le design est toujours réactif et utilise toutes les techniques que nous avons détaillées dans ce livre : la mise en page est basée sur une grille fluide et les images marchent toujours aussi bien dans ce contexte flexible. Mais à l'inverse du site Robot or Not, j'applique des media queries `min-width` pour redimensionner le design à mesure qu'on remonte dans le spectre des résolutions. La structure de base de la feuille de styles ressemble à ça :

```
/* Mise en page linéaire par défaut */
.page {
    margin: 0 auto;
    max-width: 700px;
    width: 93%;
}
```

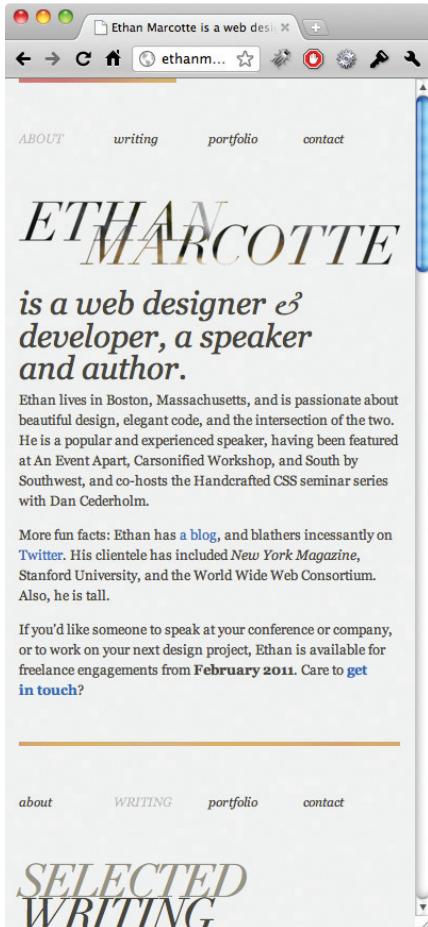
```

/* Petit écran ! */
@media screen and (min-width: 600px) { ... }

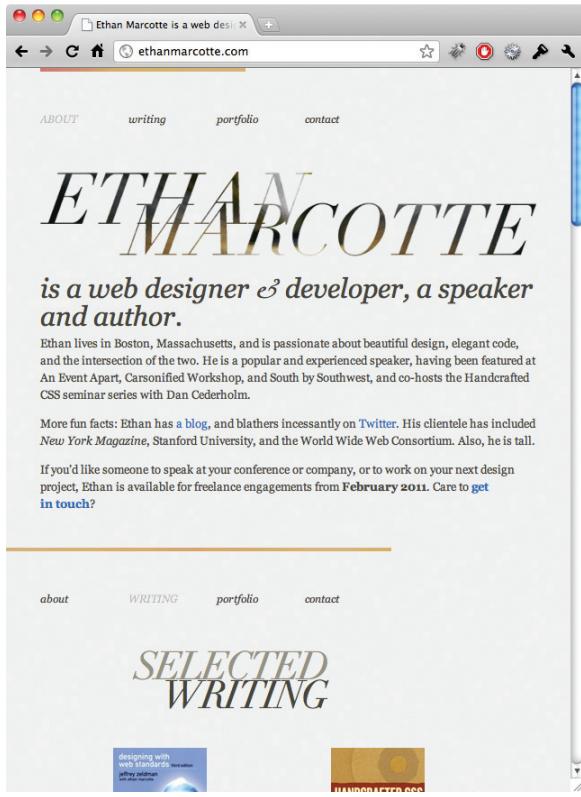
/* «Bureau» */
@media screen and (min-width: 860px) { ... }

/* IT'S OVER 9000 */
@media screen and (min-width: 1200px) { ... }

```



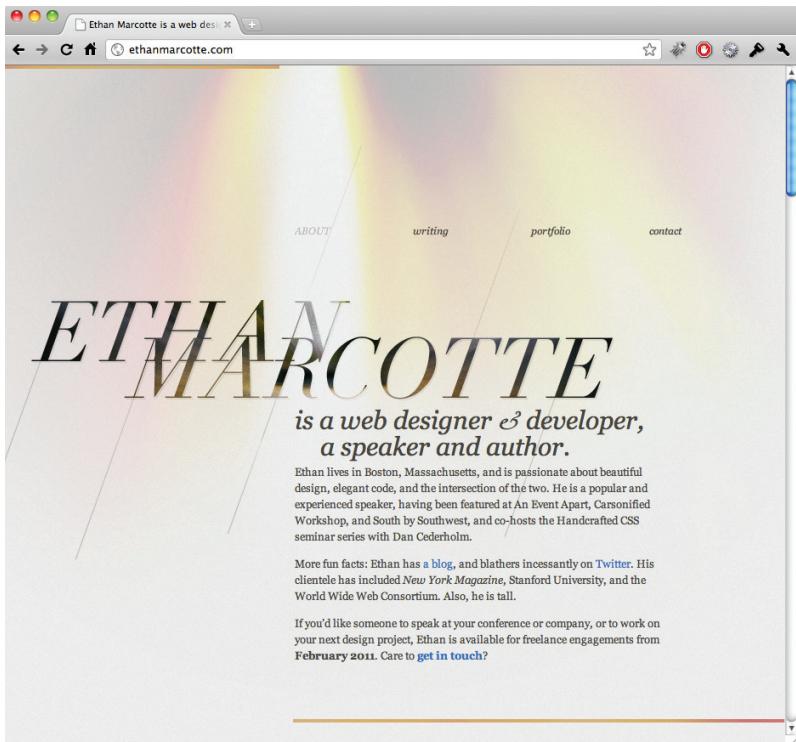
**FIG 5.12 :** Le design par défaut pour petit écran.



**FIG 5.13 :** À mi-chemin, le design devient un peu plus complexe.

Le gros de la feuille de styles ne contient pas grand chose d'autre que des règles de typographie et de couleur, offrant un design basique (mais espérons-le, attractif) à tous les utilisateurs. On crée ensuite des media queries pour les quatre points de rupture, `480px`, `600px`, `860px` et `1200px`. Désormais, si la largeur de la page dépasse ces paliers, les règles de mise en page appropriées s'appliqueront. Mais si un navigateur qui ne supporte pas les media queries accède à mon site, sans l'aide de notre correctif JavaScript, il obtient une mise en page attractive sur une seule colonne (FIG 5.15).

L'approche « mobile first » garantit une meilleure accessibilité à notre contenu sans spéculer sur les capacités de l'appa-

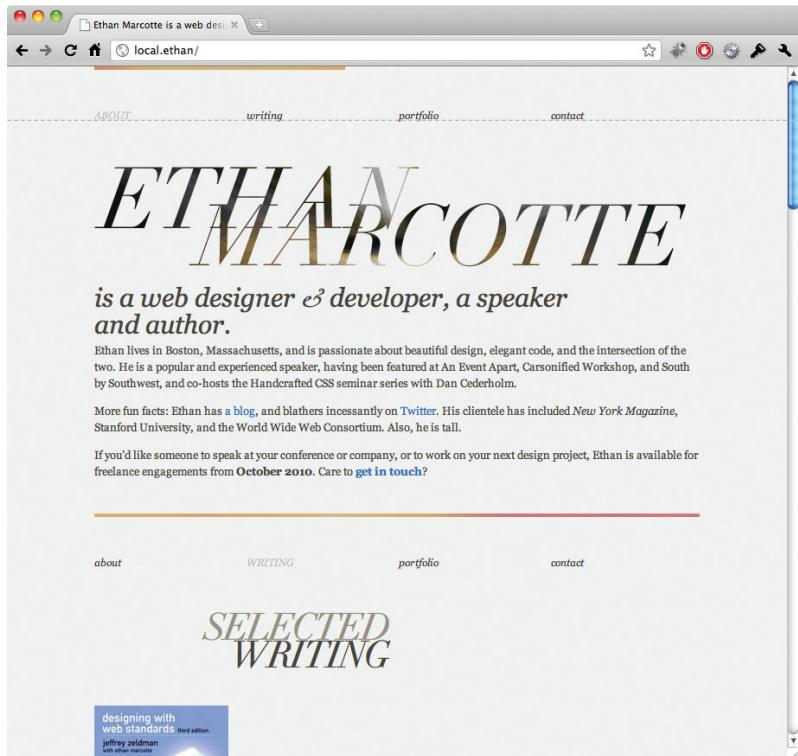


**FIG 5.14 :** Enfin, le design complet apparaît dans les plus grandes résolutions, progressivement amélioré à l'aide des media queries.

reil ou du navigateur qui restitue notre design. Et après l'avoir adoptée pour un certain nombre de projets client, je pense que c'est la meilleure et la plus infaillible manière d'implémenter vos responsive designs.

## L'AMÉLIORATION PROGRESSIVE REVISITÉE

Une implémentation plus complète de cette approche se retrouve dans la refonte récente de Yiibu (<http://yiibu.com>), un studio de design axé mobile. Bryan et Stephanie Rieger, les fondateurs de Yiibu, ont qualifié cette refonte de mélange de



**FIG 5.15 :** Pas de media queries ? Pas de JavaScript ? Cette fois-ci, pas de problème.

« mobile first » et de responsive design, et décrivent ainsi leur approche (<http://bkapr.com/rwd/52/>) :

*Le contenu de base et la présentation par défaut sont conçus pour les appareils mobiles et optimisés en premier pour les appareils les plus simples. C'est notre support « basique ». Les appareils à petit écran qui supportent les media queries bénéficient d'une mise en page améliorée et (occasionnellement) d'un contenu plus complexe. Nous l'avons baptisée « mobile ». Pour finir, la mise en page et le contenu sont encore améliorés pour le contexte « bureau ».*

Le vocabulaire est différent, bien sûr, mais il rappelle la définition originale de l'« amélioration progressive » (*progressive enhancement*) par Nick Finck et Steven Champeon (<http://bkaprt.com/rwd/53/>) :

*Plutôt que de compter sur une dégradation élégante, l'amélioration progressive consiste à construire en premier les documents pour les appareils les moins capables ou ayant des capacités différentes, puis à améliorer ces documents à l'aide d'une logique de présentation séparée, de façon à ne pas placer un fardeau indu sur les épaules des appareils les moins puissants tout en offrant une expérience plus riche aux utilisateurs dotés de navigateurs graphiques modernes.*

Depuis que Nick et Steven ont créé l'expression en 2003, l'amélioration progressive est devenue la caractéristique d'une approche responsable du design Web. En commençant par une base de HTML sémantique bien structuré, une couche de CSS et au besoin, des scripts DOM via JavaScript, on peut offrir une expérience magnifique aux navigateurs les plus capables, tout en garantissant un accès universel au contenu.

Stephen Hay lui aussi réitère cet appel en faveur de l'amélioration progressive dans son fantastique essai, « There is no Mobile Web » (<http://bkaprt.com/rwd/54/>) :

*La plupart des sites du Web ne sont pas construits dans l'optique d'utilisations spécifiques aux appareils mobiles. Cependant, des millions de personnes accèdent chaque jour à ces sites sur des appareils mobiles. Ils accèdent à un site Web « normal » (quoi que cela veuille dire) sur leur appareil « mobile ».*

...

*Pour être honnête, il me vient quelques idées de cas où des sites Web ou des applications devraient être exclusivement mobiles, mais pas beaucoup. Il semble que le Web mobile nous permet de revisiter toutes ces histoires d'inclusion, d'amélioration progressive et d'accessibilité dont on parlait il y a des années.*

Il vous est déjà arrivé que quelqu'un exprime parfaitement pourquoi vous croyez en quelque chose ? L'essai de Stephen parvient à définir exactement les raisons pour lesquelles je suis si enthousiaste devant le responsive design. Plutôt que de cloisonner simplement notre contenu dans des sites différents pour chaque appareil, on peut utiliser l'amélioration progressive pour garantir un accès de qualité à tout le monde, et une expérience améliorée pour les appareils qui en sont capables.

## Travailler avec JavaScript

Afin de mettre cette approche à l'essai, penchons-nous sur le diaporama placé au sommet du site Robot or Not (FIG 5.16). Pour le moment, le balisage ressemble à ceci :

```
<div class="slides">
  <div class="figure">
    <b></b>
    <div class="figcaption">...</div>
  </div><!-- /end .figure -->

  <ul class="slide-nav">
    <li><a class="prev" href="#">Previous</a></li>
    <li><a class="next" href="#">Next</a></li>
  </ul>
</div><!-- /end .slides -->
```

Rien de bien compliqué. Mais ce n'est pas non plus très fonctionnel : on a programmé l'interface du diaporama mais il n'est pas encore implémenté. On a inclus une seule image dans notre template, ainsi que des liens précédent/suivant qui ne fonctionnent pas encore.

On va donc devoir ajouter une pincée de JavaScript pour apporter un peu d'interactivité à notre design. Mais d'abord, il nous faut des images ! Alors récupérons-en quelques unes pour enrichir un peu notre HTML :



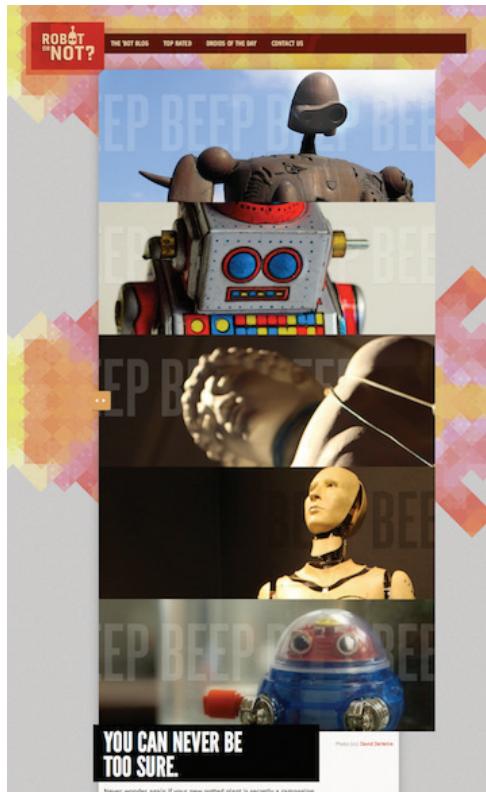
FIG 5.16 : Notre diaporama. Ou du moins une pâle copie qui ne marche pas.

```
<div class="slides">
  <div class="figure">
    <b></b>
    <div class="figcaption">...</div>
  </div><!-- /end .figure -->
  <div class="figure">
    <b></b>
    <div class="figcaption">...</div>
  </div><!-- /end .figure -->
  <ul class="slide-nav">
    <li><a class="prev" href="#">Previous</a></li>
    <li><a class="next" href="#">Next</a></li>
  </ul>
</div><!-- /end .slides -->
```

Ajoutons quatre images de plus sur le même modèle.

Pour le moment ça ne ressemble pas à grand chose : nos images sont toutes empilées les unes sur les autres (FIG 5.17). Pour que notre diaporama prenne vie, nous allons utiliser un plugin jQuery gratuit conçu par le développeur Mat Marquis (<http://bkaprt.com/rwd/55/>). C'est l'un des scripts de diaporama les plus robustes que je connaisse. Je l'aime particulièrement parce qu'il fonctionne à merveille avec du contenu flexible ; si vos diapositives comportent des quantités de texte ou d'images

**FIG 5.17 :** Ces images empilées sont particulièrement disgracieuses.



différentes, ce plugin les gère sans problème. Le tout sans avoir à recourir à une loufoquerie de CSS alambiquée. (Oh oui, j'ai bien dit loufoquerie. Je ne suis pas là pour plaisanter.)

Pour implémenter le script, il faut que j'ajoute trois nouveaux éléments `script` dans notre HTML :

```
<script src="jquery.js"></script>
<script src="carousel.js"></script>
<script src="core.js"></script>
```

Comme le script de Mat requiert la présence de jQuery, j'ai téléchargé la librairie sur <http://jquery.com> et je l'ai placée

dans le `head` de la page (`jquery.js`), suivie du script de Mat (`carousel.js`) et d'un fichier appelé `core.js`, dans lequel on va écrire le code de notre diaporama.

Et en fait, c'est plutôt facile. Dans le fichier `core.js`, on écrit :

```
$(document).ready(function() {
    $(".welcome .slides")
        .wrapInner('<div class="slidewrap"> >
                    <div id="welcome-slides" class="slider"> >
                    </div></div>')
        .find(".slidewrap")
        .carousel({
            slide: '.figure'
        });
});
```

Si vous avez du mal avec JavaScript ou si vous n'avez jamais utilisé jQuery auparavant, pas de panique. Ce script fait plusieurs choses :

1. Il commence par localiser l'élément `div.slides` dans le module `.welcome`, en utilisant une syntaxe de sélecteur se mariant idéalement avec la CSS (`$(".welcome .slides")`).
2. Une fois qu'il a localisé cet élément, il place le contenu dans une balise requise par le script de diaporama (`.wrapInner(...)`)
3. Une fois ce nouveau HTML en place, notre script exécute la fonction `.carousel()`, créant ainsi le diaporama. Et puisque l'on a attribué à chaque diapositive la classe `.figure`, c'est le nom qu'on a demandé au script d'utiliser.

Et avec ces huit lignes de JavaScript, on a un diaporama qui fonctionne (FIG 5.18). Joie !

## Chargement de contenu paresseux (mais intelligent)

Ou du moins, c'est un bon début. Si l'on désactive JavaScript dans le navigateur, on revient au point de départ : des diapositives empilées et des liens qui n'ont aucun effet. Pour un

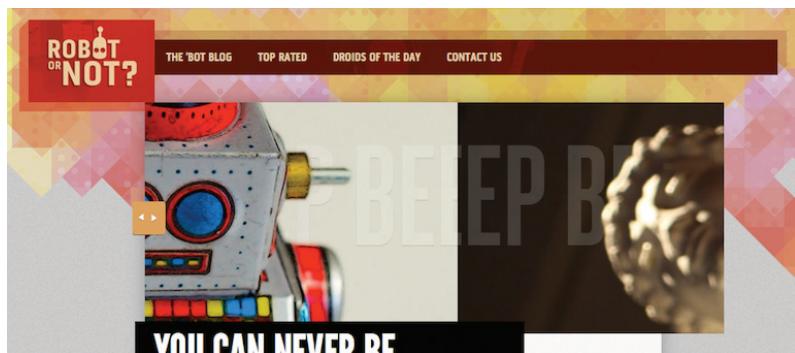


FIG 5.18 : Le diaporama est vivant. Il est vivant !

---

visiteur qui ne dispose pas de JavaScript, l'expérience est rapidement déplaisante. Attaquons-nous à ce problème.

Tout d'abord, retirons les liens précédent/suivant de notre HTML et insérons-les via JavaScript.

```
$(document).ready(function() {
    var sNav = [
        '<ul class="slide-nav">',
        '<li><a class="prev" >',
        '    href="#welcome-slides">Previous</a></li>',
        '<li><a class="next" href="#welcome-slides"> >',
        '    Next</a></li>',
        '</ul>'
    ].join("");
    $(".welcome .slides")
        .wrapInner('<div class="slidewrap"><div >'
        + id="welcome-slides" class="slider"></div></div>')
        .find(".slidewrap")
        .append(sNav)
        .carousel({
            slide: '.figure'
        });
});
```

---

Notre code paraît un peu plus complexe, mais nous n'avons fait qu'ajouter une nouvelle fonctionnalité. Tout d'abord, on déclare une variable appelée `sNav`, qui stocke le code HTML de nos liens. Puis, juste avant d'exécuter la fonction `carousel()`, on insère ce code dans notre diaporama. En utilisant jQuery pour insérer les liens dans la page, on s'assure que les utilisateurs sans JavaScript ne les verront pas. Amélioration progressive en action.

Mais cela ne résout pas le problème des images empilées. C'est là que les choses se corsent : on va retirer toutes les images de la page sauf une, et les placer dans un fichier HTML séparé. Le code source de notre page s'en trouve considérablement allégé :

```
<div class="slides">
  <div class="figure">
    <b></b>
    <div class="figcaption">...</div>
  </div><!-- /end .figure -->
</div><!-- /end .slides -->
```

Cependant, on a créé un fichier séparé (appelons-le `slides.html`) pour y copier le code de nos quatre diapositives restantes :

```
<div class="figure">
  <b></b>
  <div class="figcaption">...</div>
</div><!-- /end .figure -->
<div class="figure">
  <b></b>
  <div class="figcaption">...</div>
...
</div><!-- /end .figure -->
```

Vous aurez probablement remarqué que `slides.html` n'est même pas un fichier valide. En fait, c'est un bout de HTML, un mini-document qu'on peut utiliser pour stocker du HTML qu'on utilisera plus tard. On va simplement utiliser jQuery pour

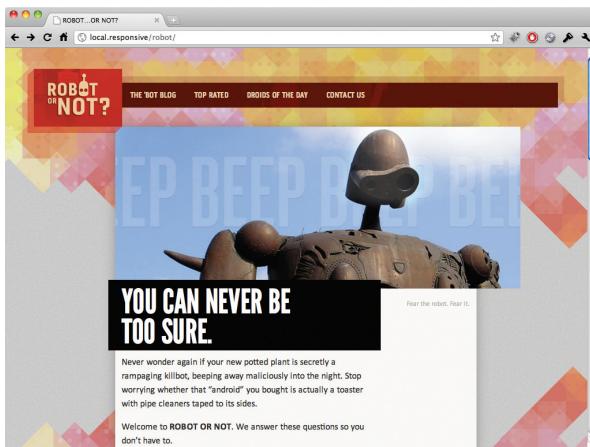
ouvrir `slides.html` et charger les images dans le diaporama, comme ceci :

```
$(document).ready(function() {
    $.get("slides.html", function(data) {
        var sNav = [
            '<ul class="slide-nav">',
            '<li><a class="prev" href="#welcome-slides">Previous</a></li>',
            '<li><a class="next" href="#welcome-slides">Next</a></li>',
            '</ul>'
        ].join("");
        $(".welcome .slides")
            .append(data)
            .wrapInner('<div class="slidewrap"> <div id="welcome-slides" class="slider"> </div></div>')
            .find(".slidewrap")
            .append(sNav)
            .carousel({
                slide: '.figure'
            });
    });
});
```

Et voilà. La fonction `.get()` de jQuery ouvre le fragment de HTML (`slides.html`) et insère son contenu dans notre module à l'aide de la fonction `append()`. Si JavaScript n'est pas disponible ou si jQuery ne parvient pas à charger ce fichier, l'utilisateur verra alors une seule image au sommet de la page : une solution de repli parfaitement acceptable pour notre design (FIG 5.19).

## Améliorations plus poussées

On a amélioré notre script de diaporama simple avec beaucoup plus de code, mais l'expérience est au final beaucoup plus



**FIG 5.19 :** Pas de JavaScript ? Pas de problème.  
Notre diaporama se dégrade en une seule image, qui est grandiose.

robuste et accessible. On ne spéculera pas sur les capacités du navigateur ou de l'appareil qui restitue notre page : si JavaScript est disponible, alors notre diaporama s'affiche.

Mais on peut toujours faire mieux, surtout avec ce grossier prototype. Par exemple, on pourrait limiter l'affichage de notre diaporama à sur certains types d'écrans seulement, en fonction de la résolution. Si on voulait l'empêcher de se charger sur les plus petits écrans, on pourrait intégrer un test de résolution simple dans notre script :

```
if (screen.width > 480) {  
    $(document).ready(function() { ... });  
}
```

Cette déclaration `if` est l'équivalent en JavaScript d'une media query `min-width: 480px` : si l'écran mesure moins de 480 pixels de large, le JavaScript placé à l'intérieur ne s'exécutera pas (FIG 5.20).

Et on pourrait continuer à affiner cette approche. Par exemple, on pourrait idéalement utiliser un chargeur JavaScript léger comme LabJS (<http://labjs.com/>) ou Head JS (<http://headjs.com/>) pour charger dynamiquement jQuery, le plugin



FIG 5.20 : On a décidé que notre diaporama ne serait disponible que pour les navigateurs de plus de 480 px de large. Les écrans plus petits verront une seule image.

de diaporama et un fichier `custom.js`, éventuellement en ne les chargeant que si l'écran de l'utilisateur dépasse une certaine résolution. Cela aiderait à garantir que les utilisateurs de petits écrans ne soient pas forcés à télécharger tout ce JavaScript, surtout si on les empêche de charger le diaporama. Et tant qu'à faire attention à la bande passante de nos utilisateurs, j'utiliserais probablement la fantastique librairie « responsive images » du Filament Group (<http://bkaprt.com/rwd/56/>), qui nous permettrait de fournir des images plus légères et moins gourmandes aux petits écrans, les images plus grandes étant réservées aux grands écrans.

## VA ET SOIS « RESPONSIVE »

Je mentionne ces améliorations mais cela ne signifie pas qu'il s'agisse nécessairement de la bonne approche ; à l'heure des hotspots 3G portatifs et des téléphones wifi, il est présomptueux d'assimiler automatiquement les dimensions d'un écran à la bande passante disponible. Mais si vous avez besoin de plus

de réactivité en fonction de la résolution, ces outils vous seront précieux.

Je trouve tout de même utile de garder la philosophie « mobile first » de Luke à l'esprit quand je fais face à une fonctionnalité particulièrement compliquée. Si je peux désactiver une interface complexe pour les utilisateurs mobiles, pourquoi le reste de mes visiteurs devrait-ils en avoir besoin ? Si la question vous semble tendancieuse, elle n'est pas censée l'être : il n'y a pas de réponses simples, ici.

Car plus que tout, le design Web consiste à se poser les bonnes questions. Et voilà ce qu'est le responsive design : une solution possible, une façon de mieux travailler en accord avec la flexibilité inhérente du Web. Dans le premier chapitre, je disais que les ingrédients d'un responsive design étaient une grille fluide, des images flexibles et des media queries. Mais il ne s'agit au fond que du vocabulaire que nous utilisons pour articuler des réponses aux problèmes que rencontrent nos utilisateurs ; un cadre pour structurer notre contenu à l'intention d'un nombre toujours croissant d'appareils et de navigateurs.

Si vous êtes prêt à étudier les besoins de vos utilisateurs et à incorporer soigneusement ces ingrédients, alors le responsive design s'avérera une approche puissante.

J'ai hâte de voir ce que vous en ferez.

## REMERCIEMENTS

Les mots — et l'espace — me manquent pour remercier comme il se doit les personnes qui ont influencé mon travail, sans même parler de ce petit livre. Pourtant, il faut que j'essaie.

Tout d'abord, je suis infiniment reconnaissant envers A Book Apart de s'être intéressé au responsive design et de m'avoir offert la chance d'écrire mon premier livre en solo. Jason Santa Maria prête une attention inégalée aux détails et à la qualité. Mandy Brown est une éditrice incroyablement incisive, et je me sens très chanceux d'avoir pu bénéficier de son aide et de sa patience pour créer ce livre. Et bien sûr, je remercie Jeffrey Zeldman du fond du cœur : pour ses écrits passionnés et son travail inlassable, et pour les opportunités qu'il m'a accordées pendant toutes ces années.

Si j'arrive parfois à écrire une phrase qui tient la route, c'est grâce à Garret Keizer.

Peter-Paul Koch, Bryan et Stephanie Rieger, Jason Grigsby et Stephen Hay m'ont appris beaucoup de choses sur le design pour les appareils mobiles, et m'ont aidé à améliorer ma réflexion sur le responsive design de tant de manières subtiles mais décisives. Et pour tout projet de design, qu'il soit ou non réactif, le travail de Luke Wroblewski sur « mobile first » m'est inestimable.

Khoi Vinh et Mark Boulton ont enseigné à notre communauté (et à moi-même) une grande partie de l'histoire de notre discipline. Par ailleurs, la grille fluide n'aurait jamais existé sans les premières études de Richard Rutter.

Si je n'avais pas lu le magnifique livre de John Allsopp, « *A Dao of Web Design* », il y a plus de dix ans, ma compréhension du Web serait radicalement différente, et ce livre n'aurait jamais été écrit.

David Sleight, l'équipe du Filament Group — Patty Toland, Todd Parker, Maggie Costello et Scott Jehl — ainsi que Mat Marquis ont apporté un retour indispensable sur la première ébauche de ce livre. De plus, Filament m'a offert la chance de travailler sur un projet de responsive design de grande envergure alors que je commençais à écrire ce livre. Cela a été une expérience extrêmement enrichissante dont moi-même et ce livre avons bénéficié.

La correction technique de Dan Cederholm était prévenante, minutieuse et hilarante. Tout comme lui.

Je n'ai pas de mots pour dire à quel point je suis honoré que Jeremy Keith ait accepté de préfacer cet ouvrage. Et mince alors, « honoré » ne lui rend même pas justice.

Ma famille — mes parents, mes frères et sœurs et ma grand-mère — était là pour me soutenir durant toute l'écriture de ce livre. Je vous aime.

Et enfin, à ma femme Elizabeth. Ce livre, et tout le reste, c'est pour elle.

## RESSOURCES

Pour une histoire plus complète de la grille typographique, je vous suggère :

- L'entrée de Wikipédia (en anglais) sur les canons de la construction de page : <http://bkaprt.com/rwd/57/>
- *The New Typography* de Jan Tschichold (deuxième édition, University of California Press, 2006) : <http://bkaprt.com/rwd/58/>
- *Grid Systems in Graphic Design* par Josef Müller-Brockmann (Verlag Niggli AG) : <http://bkaprt.com/rwd/59/>

Pour ce qui est de l'application des grilles au design Web, je vous suggère :

- *Ordering Disorder : Grid Principles for Web Design* par Khoi Vinh (New Riders Press, 2010) : <http://bkaprt.com/rwd/60/>
- *A Practical Guide to Designing Grid Systems for the Web* par Mark Boulton (Five Simple Steps, à paraître) : <http://bkaprt.com/rwd/61/>
- Le billet du blog de Mark Boulton intitulé « A Richer Canvas » : <http://bkaprt.com/rwd/62/>
- The Grid System : <http://bkaprt.com/rwd/63/>
- Mon article publié sur *A List Apart* intitulé « Fluid Grids » : <http://bkaprt.com/rwd/64/>

Vous cherchez des références sur les media queries ? Même si les deux liens suivants sont quelque peu techniques, je pense qu'ils sont tout de même accessibles et très utiles :

- La spécification du W3C sur les media queries : <http://bkaprt.com/rwd/65/>
- La référence de Mozilla pour les développeurs sur les media queries : <http://bkaprt.com/rwd/66/>

Si vous travaillez avec des images et d'autres médias dans un contexte flexible, je vous recommande :

- Le script « Responsive Images » du Filament Group : <http://bkaprt.com/rwd/67>, avec les articles correspondants sur le blog : <http://bkaprt.com/rwd/68/>, <http://bkaprt.com/rwd/69/>
- Les premières expériences de redimensionnement d'image par Richard Rutter : <http://bkaprt.com/rwd/70/>
- Les premières expériences de Bryan Rieger sur l'adaptation des images : <http://bkaprt.com/rwd/71/>

Pour vous aider à décider quand et comment adopter une approche réactive, je vous recommande :

- L'œuvre fondatrice de John Allsopp, « A Dao of Web Design » : <http://bkaprt.com/rwd/72/>
- Les articles de Luke Wreglowski sur « mobile first » : <http://bkaprt.com/rwd/47/>, ainsi que les lectures disponibles sur le même sujet : <http://www.lukew.com/ff/archive.asp?tag&mobilefirst>
- Les articles « One Web » (<http://bkaprt.com/rwd/73>) et « Context » (<http://bkaprt.com/rwd/74>) de Jeremy Keith
- L'article de Tim Kadlec intitulé « Responsive Web Design and Mobile Context » : [http://bkaprt.com/rwd/75/](http://bkaprt.com/rwd/75)
- Les écrits de Josh Clark (<http://bkaprt.com/rwd/76>) et de Jason Grigsby (<http://bkaprt.com/rwd/77>) pourront vous aider à décider quand une approche « responsive » est appropriée, et pour quels projets. (Vous avez tout intérêt à lire les blogs de Josh et Jason de toute façon.)

Mes propres articles, « With Good References » (<http://bkaprt.com/rwd/78>) et « Toffee-Nosed » (<http://bkaprt.com/rwd/79>).

# RÉFÉRENCES

Les URL abrégées sont numérotées dans l'ordre chronologique ; les URL complètes correspondantes sont listées ci-dessous à titre de référence.

## Chapitre 1

- 1 <http://www.dolectures.com/speakers/craig-mod/>
- 2 <http://www.flickr.com/photos/carabanderson/3033798968/>
- 3 <http://www.alistapart.com/articles/dao/>
- 4 [http://www.morganstanley.com/institutional/techresearch/mobile\\_internet\\_report122009.html](http://www.morganstanley.com/institutional/techresearch/mobile_internet_report122009.html)
- 5 <http://vimeo.com/14899669>
- 6 <http://vimeo.com/14899445>
- 7 <http://www.smartglassinternational.com/>
- 8 <http://vimeo.com/4661618>

## Chapitre 2

- 9 <http://meyerweb.com/eric/tools/css/reset/>

## Chapitre 3

- 10 <http://www.flickr.com/photos/uberculture/1385828839/>
- 11 <http://clagnut.com/sandbox/imagetest/>
- 12 [http://www.svendtofte.com/code/max\\_width\\_in\\_ie/](http://www.svendtofte.com/code/max_width_in_ie/)
- 13 <http://msdn.microsoft.com/en-us/library/ms532969.aspx>
- 14 [http://www.dillerdesign.com/experiment/DD\\_belatedPNG/](http://www.dillerdesign.com/experiment/DD_belatedPNG/)
- 15 [http://msdn.microsoft.com/en-us/library/ms532920\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms532920(VS.85).aspx)
- 16 <http://unstoppablerobotninja.com/entry/fluid-images>
- 17 <http://www.yuiblog.com/blog/2008/12/08/imageopt-5/>
- 18 <http://www.alistapart.com/articles/fauxcolumns/>
- 19 <http://stopdesign.com/archive/2004/09/03/liquid-bleach.html>
- 20 <http://www.w3.org/TR/css3-background/#the-background-size>
- 21 <http://srobbin.com/jquery-plugins/jquery-backstretch/>
- 22 <http://www.bbc.co.uk/news/technology-11948680>
- 23 <http://bryanrieger.com/issues/mobile-image-replacement/>

## Chapitre 4

- 24 <http://www.w3.org/TR/CSS2/media.html>
- 25 <http://www.alistapart.com/articles/goingtoprint/>
- 26 <http://www.w3.org/TR/CSS21/media.html#media-types>
- 27 <http://www.w3.org/TR/css3-mediaqueries/>
- 28 <http://www.w3.org/TR/css3-mediaqueries/#media1>
- 29 <http://developer.apple.com/library/safari/#documentation/appleapplications/reference/SafariHTMLRef/Articles/MetaTags.html>
- 30 [https://developer.mozilla.org/en/Mobile/Viewport\\_meta\\_tag#Viewport\\_basics](https://developer.mozilla.org/en/Mobile/Viewport_meta_tag#Viewport_basics)
- 31 <http://www.theleagueofmoveabletype.com/fonts/7-league-gothic>
- 32 <http://windows.microsoft.com/ie9>
- 33 <http://ie.microsoft.com/testdrive/HTML5/CSS3MediaQueries/>
- 34 <http://www.quirksmode.org/mobile/#t14>
- 35 <http://blogs.msdn.com/b/iemobile/archive/2011/02/14/ie9-coming-to-windows-phone-in-2011.aspx>
- 36 <http://www.quirksmode.org/m/css.html#t021>
- 37 <http://code.google.com/p/css3-mediaqueries-js/>
- 38 <https://github.com/scottjehl/Respond>
- 39 <http://37signals.com/svn/posts/2661-experimenting-with-responsive-design-in-iterations>
- 40 <http://hicksdesign.co.uk/journal/finally-a-fluid-hicksdesign>
- 41 <http://thethemefoundry.com/shelf/>

## Chapitre 5

- 42 <http://tripleodeon.com/2010/10/not-a-mobile-web-merely-a-320px-wide-one>
- 43 <http://jeffcroft.com/blog/2010/aug/06/responsive-web-design-and-mobile-context/>
- 44 <http://thefonecast.com/News/tabid/62/EntryId/3602/Mobile-shopping-is-popular-when-watching-TV-says-Orange-UK-research.aspx>
- 45 <http://www.lukew.com/ff/entry.asp?1263>
- 46 <http://www.flickr.com/photos/merlin/sets/72157622077100537/>
- 47 <http://www.lukew.com/ff/entry.asp?933>
- 48 <http://www.lukew.com/ff/entry.asp?1117>

- 49 <http://chrispederick.com/work/web-developer/>
- 50 <http://www.alistapart.com/articles/mobile-web-design/>
- 51 <http://www.flickr.com/photos/filamentgroup/5149016958/>
- 52 <http://yiibu.com/about/site/>
- 53 <http://www.hesketh.com/thought-leadership/our-publications/inclusive-web-design-future>
- 54 <http://www.the-haystack.com/2011/01/07/there-is-no-mobile-web/>
- 55 <http://matmarquis.com/carousel/>
- 56 [http://filamentgroup.com/lab/responsive\\_images\\_experimenting\\_with\\_context\\_aware\\_image\\_sizing/](http://filamentgroup.com/lab/responsive_images_experimenting_with_context_aware_image_sizing/)

## Ressources

- 57 [http://en.wikipedia.org/wiki/Canons\\_of\\_page\\_construction](http://en.wikipedia.org/wiki/Canons_of_page_construction)
- 58 <http://www.amazon.com/dp/0520250125/>
- 59 <http://www.amazon.com/dp/3721201450/>
- 60 <http://www.amazon.com/gp/product/0321703537/>
- 61 <http://www.fivesimplesteps.com/products/a-practical-guide-to-designing-grid-systems-for-the-web>
- 62 <http://www.markboulton.co.uk/journal/comments/a-richer-canvas>
- 63 <http://www.thegridsystem.org/>
- 64 <http://www.alistapart.com/articles/fluidgrids/>
- 65 <http://www.w3.org/TR/css3-mediaqueries/>
- 66 [https://developer.mozilla.org/En/CSS/Media\\_queries](https://developer.mozilla.org/En/CSS/Media_queries)
- 67 <https://github.com/filamentgroup/Responsive-Images>
- 68 <http://unstoppablerobotninja.com/entry/responsive-images/>
- 69 [http://filamentgroup.com/lab/responsive\\_images\\_experimenting\\_with\\_context\\_aware\\_image\\_sizing/](http://filamentgroup.com/lab/responsive_images_experimenting_with_context_aware_image_sizing/)
- 70 <http://clagnut.com/blog/268/>
- 71 <http://bryanrieger.com/issues/mobile-image-adaptation>
- 72 <http://www.alistapart.com/articles/dao>
- 73 <http://adactio.com/journal/1716/>
- 74 <http://adactio.com/journal/4443/>
- 75 <http://timkadlec.com/2011/03/responsive-web-design-and-mobile-context/>
- 76 <http://globalmoxie.com/blog/mobile-web-responsive-design.shtml>
- 77 <http://www.cloudfour.com/weekend-reading-responsive-web-design-and-mobile-context/>
- 78 <http://unstoppablerobotninja.com/entry/with-good-references/>
- 79 <http://unstoppablerobotninja.com/entry/toffee-nosed/>

# INDEX

37signals 101, 102  
.get() 136

## A

A List Apart 150  
Allsopp, John 5, 140, 143  
AlphaImageLoader 52, 53, 54  
Android 99, 103  
append() 136  
Apple 79, 80

## B

background-position 58  
background-size 58  
BlackBerry 99  
Boulton, Mark 14, 140  
Bowman, Doug 55

## C

Cederholm, Dan 55, 58, 103, 141  
cible ÷ contexte = résultat 20, 23, 31,  
35, 41, 55, 89  
Cog'aoke 109, 110  
Croft, Jeff 107  
css3-mediaqueries.js 99, 100

## D

Dao De Jing 106  
Diller, Drew 52  
display:  
display: none 111

## E

espace 66  
espacement 26, 33, 34, 35, 36, 37  
étude du design interactif 118  
expression 48

## F

Finck, Nick 129  
font-size 18, 19, 20, 21, 72

Frost, Robert 1

## G

Galaxy Tab 102  
Grid Systems in Graphic Design 14,  
142  
grille fluide 25, 34, 41, 42, 56, 64, 103,  
116, 122, 124, 139, 140  
grille typographique 14, 15, 142

## H

Happy Cog 103, 104, 150  
Hay, Stephen 129, 140  
Hicks, Jon 103, 104, 105

## I

initial-scale 81  
Internet Explorer 7 47, 50, 100  
iPad 79, 83, 102, 109, 114, 115, 123  
iPhone 74, 79, 80, 82, 123

## J

Jehl, Scott 99, 140  
jQuery 118, 131, 132, 133, 135, 136, 137  
jQuery Backstretch 59

## K

Kindle 102, 103, 114  
Koch, Peter-Paul 99, 117, 140

## L

League Gothic 84

## M

Mann, Merlin 111  
marge 30, 33, 35, 37, 40, 86  
Marquis, Mat 140  
max-width 61

max-width: 100% 45, 49, 50, 54, 61,  
62, 63, 89, 90  
mobile first 111  
Mobile Safari 80, 81, 82, 99, 123  
Mod, Craig 1  
Mozilla 99  
Müller-Brockmann, Josef 14, 142

## N

---

Noise to Noise Ratio 111  
Nook 102, 114

## O

---

Opera Mini 99  
Opera Mobile 99  
orientation 77, 79  
overflow 59, 60, 61, 62, 66, 89

## P

---

période moderniste 13  
période romantique 13

## R

---

reset CSS 17, 18  
respond.js 99, 100, 122  
responsive architecture 7  
Rieger, Bryan 62, 127  
Rieger, Stephanie 127

Robbin, Scott 59  
Ruder, Emil 14  
Rutter, Richard 45, 140, 143

## S

---

Samsung 102  
sizingMethod 52, 53  
surface de rendu 76

## T

---

Tofte, Svend 48  
Tschichold, Jan 14, 142  
types de médias 71, 72, 73, 74, 116

## V

---

Vinh, Khoi 14, 140

## W

---

Web Developer 116, 117  
webOS 99  
width: 100% 49, 50, 54  
width=device-width 82  
Wren, Christopher 7  
Wroblewski, Luke 111, 140

## Z

---

zone d'affichage 76

## À PROPOS DE A BOOK APART

Le Web design est une affaire de maîtrise multidisciplinaire et de haute précision. C'est justement l'idée qui se reflète dans notre collection de petits livres, destinée à tous ceux qui créent des sites Web. Nous traitons les sujets émergents et essentiels du design et du développement Web avec style, clarté et surtout concision, parce qu'un designer-développeur qui travaille n'a pas de temps à perdre.

L'objectif de tous les livres de notre catalogue, c'est de faire toute la lumière sur un sujet délicat, et de le faire vite pour mieux retourner au travail. Merci de nous soutenir dans notre mission : offrir aux professionnels les outils dont ils ont besoin pour faire avancer le Web.

## À PROPOS DE L'AUTEUR



Ethan Marcotte (<http://ethanmarkotte.com>) est un designer/développeur polyvalent dont le travail démontre une passion pour le code et le design de qualité. Ethan, qui vit à Cambridge dans le Massachusetts, a eu la chance de travailler pour des clients aussi prestigieux que le *New York Magazine*, l'université Stanford, le festival du film Sundance et le W3C. Petite anecdote : Ethan a un blog (<http://unstoppablerobotninja.com>), et s'épanche sans arrêt sur Twitter (@beep). Et il est grand.

Ethan est contributeur et éditeur technique chez *A List Apart*, le magazine pour ceux qui font des sites Web. Il est aussi un formateur reconnu et a participé aux conférences de An Event Apart, du SXSW Interactive Festival, du Future of Web Design ainsi qu'à la conférence In Control de AIGA.

Auteur expérimenté, Ethan a collaboré avec le fondateur de Happy Cog, Jeffrey Zeldman, sur la troisième édition de *Designing With Web Standards* (New Riders, 2009), un ouvrage classique qui se trouve sur les étagères de tous les designers dans le coup. Par ailleurs, il a contribué à l'écriture de *Handcrafted CSS* (New Riders, 2009), *Web Standards Creativity* (friends of ED, 2007) et *Professional CSS* (Wrox, 2005).

*Responsive Web Design* est le premier livre d'Ethan en solo, et ça le fait bicher grave. Il vous remercie de l'avoir lu.