Möglichkeiten zur Optimierung von Bildern im Web im Hinblick auf Performance

Alexander Schuster <it201516@fhstp.ac.at>

17. Februar 2021

Zusammenfassung

Einen beliebten Bereich zur Performance-Optimierung moderner Websites und Web-Apps stellen Bilder dar. In diesem Artikel sollen Strategien zur Verringerung der Ladezeit von Bildern in HTML5-Dokumenten erläutert werden. Dies umfasst konkret den Einsatz effizienter Bild-Dateiformate und das Zurverfügungstellen mehrerer alternativer Versionen desselben Bilds, sodass der Client aus dem Markup die für ihn passende Auswahl treffen und von den damit einhergehenden Daten-Einsparungen profitieren kann.

1 Einleitung

Zwar ist Bandbreite in Zeiten von 5G längst nicht mehr solch ein großes Problem wie dies noch vor einigen Jahren der Fall war, jedoch kann das Vernachlässigen der Optimierung einer Website nicht zuletzt für NutzerInnen mobiler Geräte nach wie vor zu einer schlechten User-Experience führen. So kamen in der jüngeren Vergangenheit beispielsweise neue Bild-Dateiformate für das Web auf, welche eine effizientere Codierung der Bilddaten als dies bei den bereits seit Jahrzehnten etablierten Standards der Fall ist, zum Ziel haben [1], [2]. Auch wurden Standards geschaffen, um mehrere für unterschiedliche Bildschirmgrößen geeignete Bild-Dateien gleichzeitig anzugeben und zur Verfügung zu stellen, sodass der Browser eigenständig die passende Version lädt [3]. Dies ist spätestens seit dem Erscheinen von Retina-Displays mit höherer Auflösung relevant, welche nun mit den herkömmlichen Displays koexistieren.

In diesem Artikel soll ein Überblick über moderne Ansätze zur Optimierung von Bilddateien im Web gegeben und darauf eingegangen werden wie diese möglichst effizient vom Client angefordert werden können. Es werden mehrere prominente Konzepte aus diesem Themenfeld vorgestellt.

2 Moderne Dateiformate

Noch bevor ein Bild vom Client angefordert wird, erscheint es relevant, die Bild-Datei an sich effizient zu kodieren, um auf diese Weise die Dateigröße zu verringern. In diesem Kontext sind die Formate WebP, JPEG 2000, JPEG XR – und seit kurzem auch AVIF – zu nennen, deren Browser-Unterstützung unterschiedlich ausfällt, wodurch eine einander ergänzende Einsatz-Strategie vorstellbar ist [2].

Um bestehendes Bildmaterial zu konvertieren, können beispielsweise Web-Apps, wie etwa *Squoosh*, genutzt werden. Diese bietet die Möglichkeit, die Qualität zu vergleichen und zeigt die prozentuelle Reduzierung der Dateigröße. Es werden jedoch nicht alle Formate unterstützt [4].

2.1 WebP

Das von Google im Jahr 2010 ins Leben gerufene Format WebP unterstützt sowohl verlustfreie als auch verlustbehaftete Datenkompression. Nach eigenen Angaben sind verlustfreie WebP-Dateien etwa um 26% kleiner als vergleichbare PNG-Dateien. Für die verlustbehaftete Variante ist sogar von von bis zu 34% Ersparnis im Vergleich zu gewöhnlichem JPEG die Rede [1], [5]. Von den hier behandelten Formaten erfährt WebP zurzeit die umfassendste Unter-

stützung durch gängige Web-Browser und kann durchaus als etabliert betrachtet werden [6].

2.2 AVIF

Ein neuer Standard ist AVIF, welcher auf der AV1-Intra-Frame-Codierung basiert. Dieser wurde in [7] bereits evaluiert und erzielt in zahlreichen die Qualität bzw. Effizienz betreffenden Parametern bessere Ergebnisse als WebP und JPEG. Der Test umfasste drei Datensätze bestehend aus natürlichen (fotografischen), synthetischen sowie Gaming-Bildern.

Das Format wird bereits standardmäßig von Google Chrome unterstützt [8] und auch in Mozilla Firefox kann dieses Feature bereits genutzt werden, wenngleich auch erst in einem experimentellen Stadium [9]. Es zeichnet sich gegenwärtig ein deutlicher Trend hin zur Annahme dieses Formats ab. Einen Überblick diesbezüglich bietet [10].

2.3 JPEG 2000 und JPEG XR

Zur Vollständigkeit seien hier auch JPEG 2000 und JPEG XR erwähnt. Die Browser-Unterstützung für diese Formate ist sehr rudimentär, weswegen sie sich vor allem als Fallback-Optionen für wenige spezifische Clients eignen [11], [12]. Neben der geringeren Dateigröße gegenüber herkömmlichem JPEG besteht bei JPEG 2000 auch die Möglichkeit, aus demselben Bit-Stream verschiedene Auflösungen und Bildausschnitte zu extrahieren. Durch eine solche progressive Übertragung ist es dem Browser möglich, die für den Anwendungsfall nützlichen Daten zu selektieren [13]. Der später entwickelte JPEG-XR-Standard sollte eine Alternative zu JPEG 2000 mit geringem Rechenaufwand beim Kodieren und Dekodieren bieten, Bilder mit hohem Dynamikumfang (HDR) unterstützen sowie Möglichkeiten zur Interaktion mit dem Bild im Web-Browser schaffen [14].

3 Die Attribute „srcset“ und „sizes“

Die auf `<img>`-Elemente anwendbaren HTML-Attribute `srcset` und `sizes` bieten dem Client die Möglichkeit, aus mehreren verschiedenen Bildquellen (URLs) die gewünschte Version auszuwählen. Während `srcset` der Angabe einer URL-Liste und der originalen Bildgröße (oder der Pixel-Dichte, für die eine Quelle vorgesehen ist) dient, enthält `sizes` die nötige Information über die letztendliche Darstellungsgröße am Bildschirm und etwaige diesbezügliche Unterschiede zwischen verschiedenen Bildschirmgrößen. So kann beim Ändern der Fenstergröße bei Bedarf dynamisch eine weitere Version aus der Liste geladen werden. Wird `sizes` weggelassen, so geht der Browser von einer Darstellung über die gesamte Bildschirmbreite aus [15].

Ein Bild, welches stets die gesamte Breite des Bildschirms belegt, aber für Ausgabegeräte mit einfacher und doppelter Pixel-Dichte optimiert ist, könnte also wie in Code-Beispiel 1 in das HTML-Dokument eingebunden werden.

Code-Beispiel 1: Auszeichnung mehrerer alternativer Bild-Versionen

```
<img
    srcset="einfach.jpg 1x, doppelt.jpg 2x"
    src="einfach.jpg"
    alt="Ein Beispiel"
/>
```

Zu beachten ist, dass das Attribut `src` auch hier verpflichtend anzuführen ist [3]. Es stellt die Kompatibilität mit älteren Browsern sicher, welche `srcset` nicht unterstützen. In Code-Beispiel 1 dient diesen die URL zur Version mit einfacher Pixel-Dichte als Fallback.

Nun wird ein `sizes`-Attribut hinzugefügt, welches im konkreten Fall die dargestellte Breite des Bilds für Viewport-Breiten von maximal 800 Pixel mit der vollen, andernfalls mit der halben Breite beziffert. Gemeint sind hier nicht etwa Hardware-Pixel, sondern das in CSS-Media-Querys verwendete Maß. Geräte mit höherer Pixel-Dichte rechnen die für sie gültige Entsprechung dieses Werts automatisch in Hardware-Pixel um. Ferner kann die originale Breite des Bilds in Pixel mit `500w` bzw. `1000w` angegeben werden. Dadurch kann der Browser schließlich – wiederum unter Berücksichtigung der Pixel-Dichte des Geräts auf welchem er läuft – berechnen, welche Quelle angefordert werden soll. In diesem Fall könnte das `<img>`-Element im Dokument wie in Code-Beispiel 2 angeführt werden.

Code-Beispiel 2: Beispiel mit Angabe der Darstellungsgröße eines Bilds in Abhängigkeit von der Bildschirmgröße

```
<img
    sizes="(max-width: 800px) 100vw, 50vw"
    srcset="einfach.jpg 500w, doppelt.jpg
        1000w"
    src="einfach.jpg"
    alt="Ein Beispiel"
/>
```

4 Das Picture-Tag

Die in Code-Beispiel 1 und Code-Beispiel 2 veranschaulichte Strategie stößt an ihre Grenzen, sobald die einzelnen aufgelisteten URLs nicht nur auf verschiedene Versionen desselben Bilds mit unterschiedlicher Auflösung verweisen, sondern darüber hinaus auch abhängig von der Bildschirmgröße unterschiedlicher Bildinhalt gezeigt werden soll. Dies ist beispielsweise in einem Szenario denkbar, wenn ein sehr breites Bild in der Darstellung auf Mobiltelefonen speziell zugeschnitten werden muss, um das zentrale Bildelement in einer vorteilhaften Position zu behalten. Würde man dies in Code-Beispiel 2 so handhaben, würden Geräte mit unterschiedlicher Pixel-Dichte bei gleicher Bildschirmgröße womöglich unterschiedliche Bilder bzw. voneinander abweichende Bildkompositionen zeigen.

Um dem entgegenzuwirken, steht das `<picture>`-Element zur Verfügung, welches wiederum eine beliebige Anzahl an `<source>`-Elementen (und genau ein verpflichtendes `<img>`-Element als Fallback) enthält. Jedes `<source>`-Element erhält nun, analog zu den Beispielen im vorherigen Abschnitt, ein `srcset`- bzw. wo nötig auch ein `sizes`-Attribut. Optional kann durch das `media`-Attribut ein Überspringen des gesamten `<source>`-Elements erwirkt werden, wenn sich das darin spezifizierte Media-Query als unzutreffend erweist. Ebenso wird ein solches Element außer Acht gelassen, wenn ein `type`-Attribut einen vom Browser nicht unterstützten Typ (Internet Media Type) enthält [3]. Code-Beispiel 2 umgesetzt mittels `<picture>` und Dateien im WebP-Format könnte also lauten wie in Code-Beispiel 3.

Code-Beispiel 3: Ein Beispiel-Anwendungsfall für das Picture-Tag

```
<picture>
    <source
        type="image/webp"
        media="(max-width: 800px)"
        sizes="100vw"
        srcset="mobile-einfach.webp 500w,
            mobile-doppelt.webp 1000w"
    />
    <source
        type="image/webp"
        sizes="50vw"
        srcset="desktop-einfach.webp 1000w,
            desktop-doppelt.webp 2000w"
    />
    <img
        src="einfach.jpg"
        alt="Ein Beispiel"
    />
</picture>
```

Unterstützt der Browser den Internet-Media-Type `image/webp` nicht, so werden beide `<source>`-Elemente ignoriert. Beträgt die Breite des Viewports maximal 800 Pixel, so kommt das erste `<source>`-Element in Betracht und entsprechend der Pixel-Dichte wird die passende Quelle aus dem `srcset`-Attribut geladen. Bei einer größeren Viewport-Breite fällt die Wahl hingegen auf das zweite `<source>`-Element.

5 Fazit

Im Zuge der Recherche zum vorliegenden Artikel wurde der Autor auf mehrere ihm bis dahin noch nicht geläufige Bild-Formate aufmerksam. Dies führt vor Augen, wie groß der Fortschritt hinsichtlich immer effizienterer Algorithmen für Bild-Kodierung ist. Diese können dazu beitragen, die User-Experience in modernen Web-Applikationen stetig zu verbessern und die benötigte Datenmenge gleichzeitig zu reduzieren. Der Autor ermutigt daher dazu, sich mit den damit einhergehenden Möglichkeiten vertraut zu machen und von den in diesem Artikel vorgestellten Konzepten Gebrauch zu machen. Gleichzeitig scheinen ebendiese Methoden zur Optimierung auch dazu zu verleiten, hohen Aufwand für kaum merkbare Performance-

Verbesserungen zu betreiben. Geht man schließlich weiter ins Detail, so findet man immerhin auch bei den hier angeführten Beispielen noch deutliches Verbesserungspotential. Dieser Aspekt sollte stets bedacht werden, um nicht andere relevante Bereiche außen vor zu lassen.

Zukünftige Arbeiten könnten sich mit der Messung der Performance der genannten Bild-Dateiformate befassen und einen objektiven Vergleich schaffen. Dies ist keineswegs trivial, muss doch die Dateigröße der subjektiv wahrgenommenen Bildqualität gegenübergestellt werden. Immerhin bieten viele Formate mehr als nur eine Qualitätsstufe an, was zusätzliche Komplexität in einen Vergleich bringt. In der vorliegenden Arbeit hätte eine solche detaillierte Untersuchung den Rahmen gesprengt.

Literatur

[1] „A new image format for the web“, Google Developers, [Online]. Adresse: `https://developers.google.com/speed/webp` (besucht am 02. 01. 2021).

[2] „Serve images in next-gen formats“, web.dev, [Online]. Adresse: `https://web.dev/uses-webp-images/` (besucht am 03. 01. 2021).

[3] „HTML standard: Embedded content“, Web Hypertext Application Technology Working Group (WHATWG), [Online]. Adresse: `https://html.spec.whatwg.org/multipage/embedded-content.html` (besucht am 03. 01. 2021).

[4] „Squoosh“, Squoosh, [Online]. Adresse: `https://squoosh.app` (besucht am 17. 02. 2021).

[5] J. Hu, S. Song und Y. Gong, „Comparative performance analysis of web image compression“, in *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, IEEE, 2017, S. 1–5.

[6] „WebP image format“, caniuse.com, [Online]. Adresse: `https://caniuse.com/webp` (besucht am 03. 01. 2021).

[7] N. Barman und M. G. Martini, „An evaluation of the next-generation image coding standard AVIF“, in *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, IEEE, 2020, S. 1–4. [Online]. Adresse: `https://ieeexplore.ieee.org/document/9123131/` (besucht am 03. 01. 2021).

[8] „Feature: AVIF image decode“, Chrome Platform Status, [Online]. Adresse: `https://www.chromestatus.com/feature/4905307790639104` (besucht am 03. 01. 2021).

[9] „Experimental features in Firefox“, MDN Web Docs, [Online]. Adresse: `https://developer.mozilla.org/en-US/docs/Mozilla/Firefox/Experimental_features#AVIF_AV1_Image_File_format_support` (besucht am 03. 01. 2021).

[10] „AVIF image format“, caniuse.com, [Online]. Adresse: `https://caniuse.com/avif` (besucht am 17. 02. 2021).

[11] „JPEG XR image format“, caniuse.com, [Online]. Adresse: `https://caniuse.com/jpegxr` (besucht am 03. 01. 2021).

[12] „JPEG 2000 image format“, caniuse.com, [Online]. Adresse: `https://caniuse.com/jpeg2000` (besucht am 03. 01. 2021).

[13] M. W. Marcellin, M. J. Gormish, A. Bilgin und M. P. Boliek, „An overview of JPEG-2000“, in *Proceedings DCC 2000. Data Compression Conference*, IEEE, 2000, S. 523–541. [Online]. Adresse: `http://ieeexplore.ieee.org/document/838192/` (besucht am 03. 01. 2021).

[14] F. Dufaux, G. J. Sullivan und T. Ebrahimi, „The JPEG XR image coding standard [standards in a nutshell]“, *IEEE Signal Processing Magazine*, Jg. 26, Nr. 6, S. 195–204, 2009. [Online]. Adresse: `http://ieeexplore.ieee.org/document/5230820/` (besucht am 03. 01. 2021).

[15] „HTML standard: Images“, Web Hypertext Application Technology Working Group (WHATWG), [Online]. Adresse: `https://html.spec.whatwg.org/multipage/images.html` (besucht am 03. 01. 2021).