Experten-beitrag

Moderne Weblayouts und Progressive Enhancement

Posted on 1. April 2019 by Jonas Hellwig

Das Layout einer Website muss heute vielen Anforderungen gerecht werden. Es soll einerseits flexibel sein und sich an verschiedene Displaygrößen und -auflösungen anpassen. Es muss zudem auf verschiedene Bedienkonzepte wie Mouse, Stift, Touch, Tastatur oder Sprache reagieren können. Auch die fehlerfreie Darstellung in unterschiedlichen Browsern und Betriebssystemen sowie in verschiedenen Versionen dieser Umgebungen muss gewährleistet sein. Und zuletzt soll es natürlich auch noch ästhetisch ansprechend gestaltet sein und modernen Sehgewohnheiten gerecht werden.

Was auf den ersten Blick vielleicht wie eine unlösbare Aufgabe erscheint, ist kein allzu großes Problem mehr, wenn einerseits moderne Webtechnologien sinnvoll eingesetzt werden und andererseits eine zeitgemäße Philosophie vom gesamten Projekt-Team gelebt wird.

Eine Website sieht nicht überall gleich aus

Der Workflow bei der Gestaltung einer Website sah lange Zeit so aus: Zunächst wurde die Seite geplant und ein Layout gestaltet. Priorität hatte die visuelle Brillanz des Layouts, dass im nächsten Schritt dem Kunden zur Abnahme vorgelegt wurde. Anschließend wurde das Layout an den Entwickler übergeben, der alles möglichst exakt umzusetzen hatte. Andere Disziplinen wie Suchmaschinenoptimierung, Performance, Cross-Browser-Kompatibilität

usw. wurden im Projektablauf nach hinten geschoben. Doch genau diese Themen haben dazu geführt, dass es Probleme während der Umsetzung des Layouts gab und daher häufig gestalterisch Abstriche in älteren Browsern gemacht wurden bzw. der Aufwand erheblich anstieg.

Mit der Erfindung von Responsive Design hat sich dieses Problem in vielen Agenturen nur vervielfältigt. Anstelle einzelner Layouts für die Desktop-Ansicht wurden nun auch noch Layouts für Smartphone, Tablet und Co gestaltet. Das Problem blieb dasselbe: Designer haben perfekte Layouts für den besten vorstellbaren Fall gestaltet und diese Layouts dem Kunden präsentiert. Das führt zu der Erwartungshaltung, dass die Website später überall genau so aussieht. Das wiederum führt dazu, dass Entwickler später gezwungen sind, erheblichen Mehraufwand zu betreiben, um Funktionen oder Effekte in alten Browsern nachzurüsten, die dort nicht existieren. Meist geschieht das auf Kosten der Usability, der Suchmaschinenoptimierung, der Wartbarkeit und der Geschwindigkeit einer Seite. De fakto wurden diese Disziplinen also dem Visual Design der Website untergeordnet. Das würde man heute nicht mehr machen.

Eine Website kann nicht überall identisch aussehen. Das Layout muss auf die verschiedenen Umgebungen, in denen es dargestellt werden soll, flexibel reagieren können. Die Website muss überall fehlerfrei dargestellt werden, sie muss das Corporate Design transportieren, und sie muss den Inhalt präsentieren können. Aber eine 1:1-Adaption des Layouts auf verschiedenen Geräten ist schlicht nicht möglich. Diese Tatsache sollte als großer Vorteil bei der Gestaltung im Web verstanden werden, und nicht als Einschränkung.

Progressive Enhancement

Wenn einem Projekt das Prinzip »Progressive Enhancement« zugrunde liegt, wird das Layout aus der Perspektive älterer Browser gestaltet. Man überlegt sich zunächst eine gestalterische Lösung, die in der schlechtesten projektrelevanten Umgebung gut funktioniert. In moderneren Browsern werden dann zusätzlich moderne Techniken eingesetzt, um die Darstellung zu optimieren. Das führt dazu, dass die Website in verschiedenen Browsern etwas unterschiedlich aussehen kann – doch nirgendwo fehlerhaft. Wenn neue Browser mit neuen Funktionen auf den Markt kommen, werden diese Features oben aufgesattelt. Die funktionierende Grundlage des Layouts ändert sich dadurch nicht – die Umsetzung ist daher zukunftssicher und der Entwicklungsaufwand sinkt mitunter erheblich. Progressive Enhancement steht jedoch im krassen Widerspruch zur Vorgehensweise vieler visuellen Designer, die zu Beginn eines Projekts Layouts für den »Best Case« gestalten.

Im Folgenden schauen wir uns einige CSS-Techniken an, die in modernen Weblayouts sinnvoll zum Einsatz kommen können.

CSS Overwrites

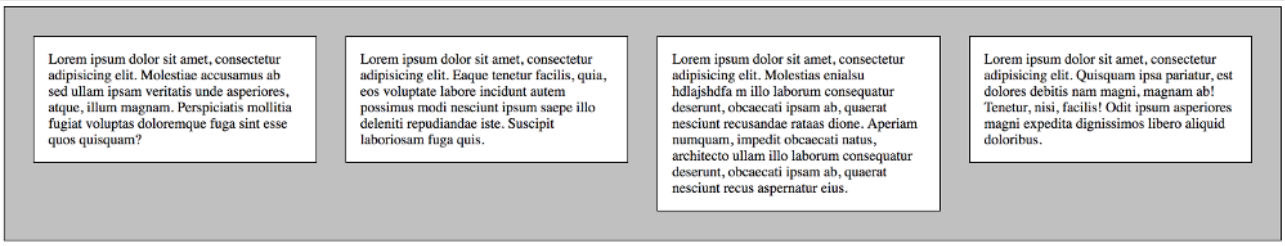
Das Prinzip des Progressive Enhancement ist in modernem CSS fest verankert: Neuere CSS-Eigenschaften überschreiben beispielsweise ältere Eigenschaften, um das Layout zu optimieren. Sehr anschaulich ist dies u.a. an der Kombination von Float und Flexbox. Das folgende Layout ist mit CSS Float hergestellt. Da die Boxen unterschiedlich viel Inhalt besitzen, sind sie unterschiedlich hoch.

HTML

```
<div class="container">
<div class="box"> … </div>
<div class="box"> … </div>
<div class="box"> … </div>
<div class="box"> … </div>
</div>
```

CSS

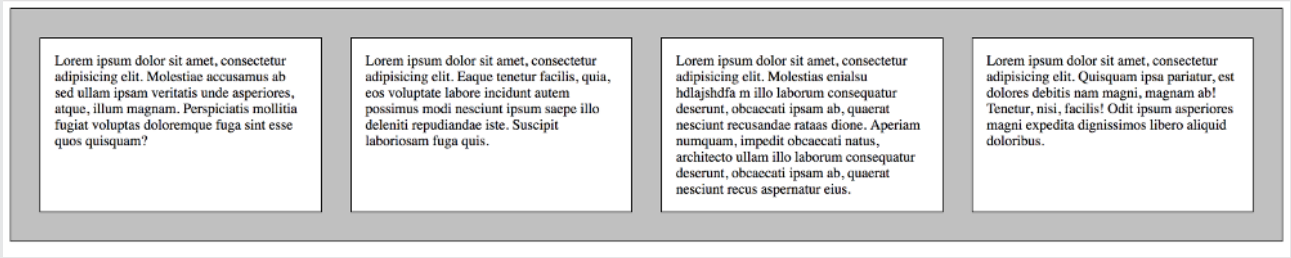
```
* {
box-sizing: border-box;
}

.container {
border: 1px solid black;
background: silver;
padding: 1em;
}

/* Clearfix > moderner mit display:flow-root; */
.container::after {
content: '';
display: block;
clear: both;
}

.box {
border: 1px solid black;
background: white;
float: left;
width: calc(25% - 2em);
padding: 1em;
margin: 1em;
}
```



Wenn nun lediglich der .container zur Flexbox umgewandelt wird, sind automatisch alle Boxen gleich hoch. Flexbox wurde absichtlich so entwickelt, dass es die seinerzeit populären CSS-Float-Grids überschreiben kann und die Basis-Gestaltung um häufig gewünschte visuelle Lösungen – z.B. gleich hohe Elemente oder vertikales Zentrieren – ergänzen kann.

CSS

```
.container {
display: flex;
}
```

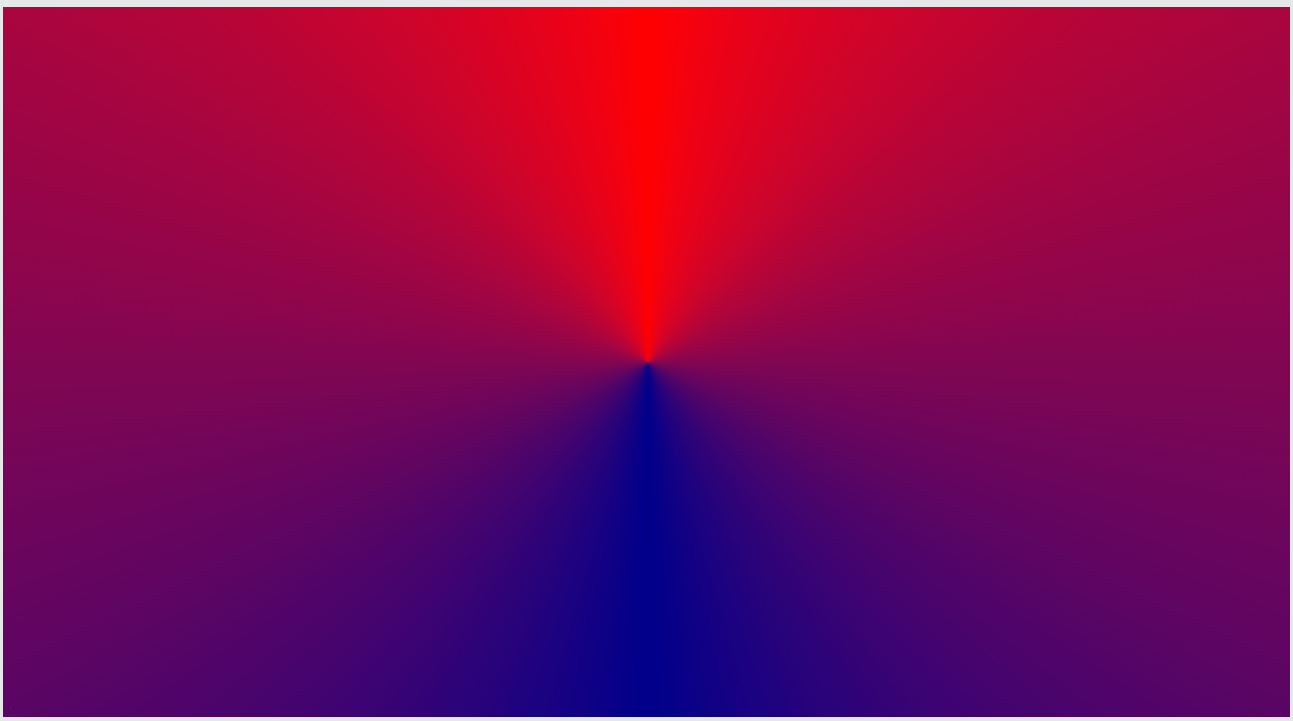


Feature Detection und @supports

Wenn CSS Overwrites nicht ausreichen, kann auch mittels @supports abgefragt werden, ob ein Browser einen CSS-Befehl versteht. Der folgende Code prüft, ob gewinkelte Verläufe (conic-gradients) unterstützt werden. Wenn ja, werden sie genutzt. Wenn nein, ist der Hintergrund einfarbig rot.

CSS

```
body {
background: darkred;
}

@supports (background: conic-gradient(red, darkblue, red)) {
body {
background: conic-gradient(red, darkblue, red);
}
}
```



Eine vollständige Erklärung der @supports-Regel, bietet u.a. die CSS-Referenz von Mozilla.

Hintergrundinformationen zu CSS conic gradients finden sich bei CSS-Tricks.

Flexbox

Die oben beschriebenen float-basierten CSS-Gestaltungsraster werden längst durch die CSS-Layoutmodelle Flexbox und Grid ersetzt. Der Vorteil dieser beiden »echten« CSS-Layoutmodelle besteht darin, dass der HTML- und CSS-Code i.d.R. deutlich schlanker ist und zudem das Layout vollständig unabhängig vom HTML-Markup gestaltet werden kann.

Flexbox arbeitet mit einem Container-Element und den direkt darauffolgenden Kind-Elementen.

HTML

```
<div class="container">
<div class="box"></div>
<div class="box"></div>
<div class="box"></div>
</div>
```

Die Elemente werden bei Flexbox auf Achsen angeordnet. Wenn für das Container-Element die Eigenschaft `display:flex;` vergeben wird, ordnen sich die enthaltenen Elemente automatisch auf der Hauptachse von links nach rechts an und nehmen die Höhe des Containers ein. Die Kreuzachse verläuft von oben nach unten.

Mit `flex-direction` können beide Achsen gemeinsam gedreht werden. Mit den Befehlen `justify-content` und `align-items` werden die Kind-Elemente auf Haupt- und Kreuzachse positioniert.

Der folgende Code positioniert die Elemente am Ende der Haupt- und der Kreuzachse. Sie befinden sich somit unten rechts im Container.

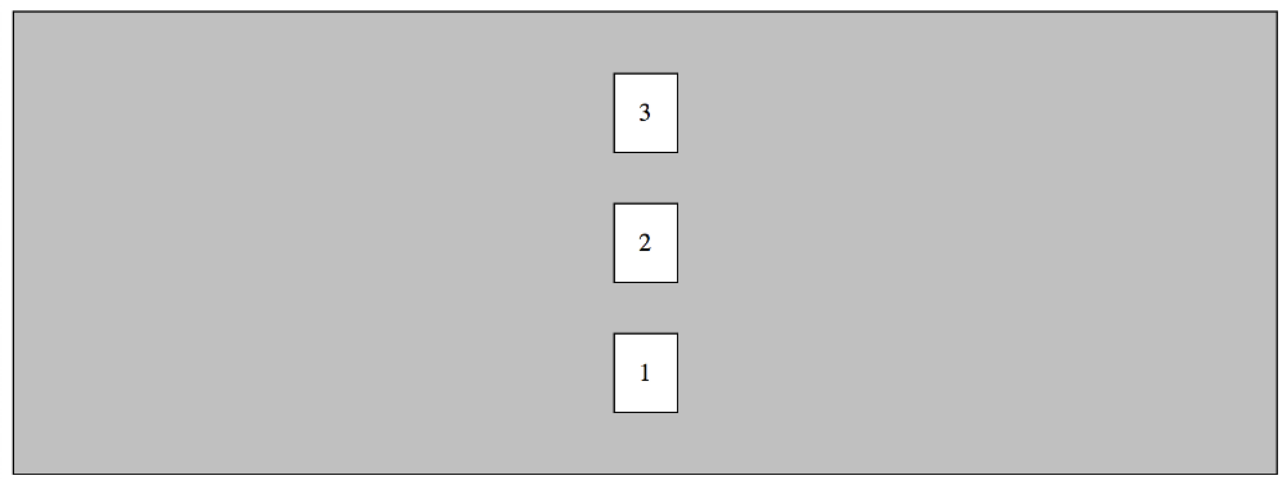
CSS

```
.container {
display: flex;
align-items: flex-end;
justify-content: flex-end;
}
```

Wenn die Elemente zentriert von unten nach oben laufen sollen, kann folgender Code genutzt werden:

CSS

```
.container {
display: flex;
flex-direction: column-reverse;
justify-content: center;
align-items: center;
}
```



Aufgrund der Positionierung über Achsen, bietet sich die Flexbox vor allem für lineare Layouts wie One-Pager sowie für lineare Komponenten wie Navigationsleisten etc. an. Zwar sind mit dem Befehl `flex-wrap` auch Mehrspaltigkeiten möglich, aber für komplex verschachtele Strukturen bietet sich die Technik nicht gut an.

Ein Grundlagenartikel zu Flexbox, mit allen verfügbaren Funktionen, findet sich u.a. auf CSS-Tricks. Wer eine Quelle auf Deutsch sucht, wird bei uns im kulturbanause-Blog fündig.

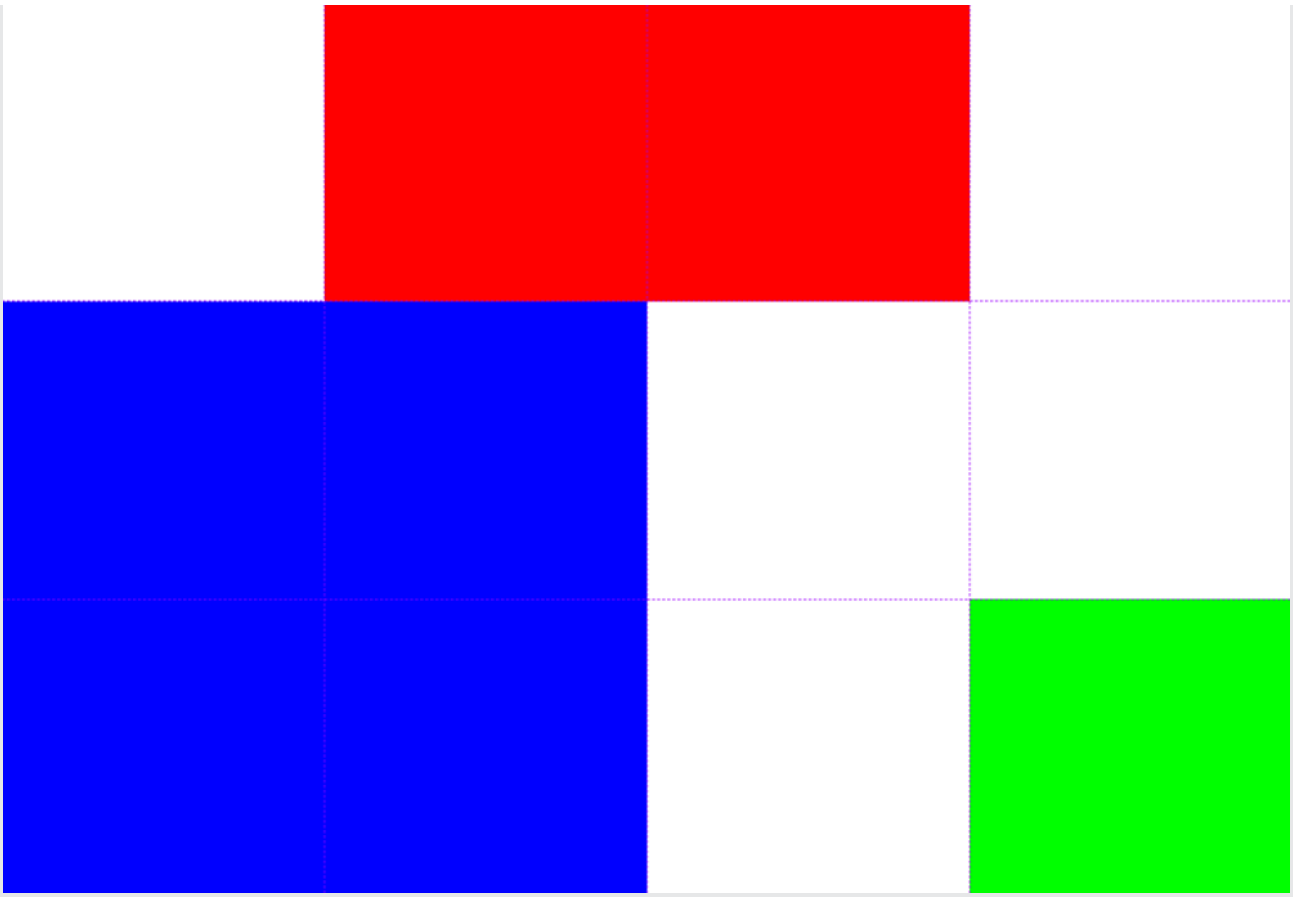
CSS Grid

Für mehrdimensionale Layouts, die Elemente auf X- und Y-Achse positionieren wollen, bietet sich CSS Grid besser an. Auch Grid arbeitet mit einem Container-Element und den direkt darauf folgenden Kindelementen.

HTML

```
<div class="container">
<div class="box"></div>
<div class="box"></div>
<div class="box"></div>
</div>
```

Im Container-Element wird mit `display: grid;` das Grid-Layoutmodell aktiviert. Anschließend werden mithilfe der Befehle `grid-template-rows` und `grid-templates-columns` Zeilen und Spalten definiert. Die Kind-Elemente können im Raster u.a. mit Hilfe der Befehle `grid-column` und `grid-row` positioniert werden, wodurch sehr komplexe und interessante Layouts möglich sind.



Der folgende Code definiert im Container ein Raster mit vier Spalten zu je 25 Prozent sowie drei Zeilen zu je 33 Prozent. Die farbigen Boxen werden anschließend mit `grid-column` und `grid-row` im Raster positioniert.

CSS

```
.container {
height: 100vh;
display: grid;
grid-template-columns: 25% 25% 25% 25%;
grid-template-rows: 33% 33% 33%;
}

.box-1 {
grid-column: 2 / span 2;
background: red;
}

.box-2 {
grid-column: span 2;
grid-row: 2 / span 2;
background: blue;
}

.box-3 {
grid-column: 4 / span 1;
grid-row: 3 / span 2;
background: lime;
}
```

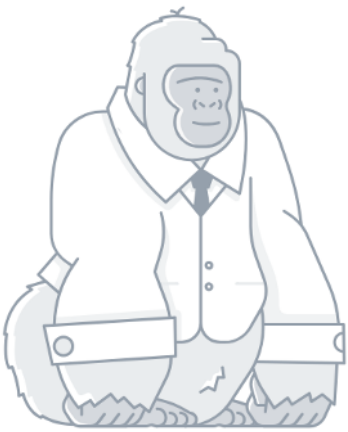
Die Einsatzmöglichkeiten von Grid und Flexbox sind sehr umfangreich und können in diesem Artikel nur angerissen werden. Interessant ist allerdings, dass – dem Prinzip Progressive Enhancement folgend – Flexbox häufig als Fallback für Grid eingesetzt wird. Als Fallback für Flexbox wiederum dienen dann Floats oder lineare Layouts.

Die Website gridtoflex.com fasst Lösungen dieser Art zusammen. Einen detaillierten Artikel zu CSS Grid findet ihr u.a. bei uns im kulturbanause-Blog.

CSS Shapes

Mit CSS Shapes können Texte um gefloatete Elemente herumfließen, auch wenn diese optisch nicht rechteckig sind. Somit sind Layouts in Anlehnung zum Print-Design möglich und das bei dynamisch geänderten Inhalten und einem responsiven Verhalten. Auch diese Funktion ist nach dem Prinzip des Progressive Enhancement einsetzbar. Wenn ein Browser keine Shapes unterstützt, fließt der Text um die rechteckige Außenkante des Elements. Wenn Shapes genutzt werden, wird das Layout aufgehübscht.

Design for successful digital products and websites



My name's **Andy Clarke**. I'm a well-known website designer, consultant, speaker, and writer on art direction and design for products and websites.
For 20 years, I've helped companies big and small to improve their website and product designs by providing consulting and design expertise.
My work is distinctive and original and I deliver engaging customer experiences and world-class designs.

I'm based in beautiful North Wales and work with businesses, charities, companies, and NGOs all over the world. I regularly work alongside designers, developers, and leadership teams to create designs, processes, and systems which deliver results.
I consult with companies on art direction for the web, design systems, and style guides, speak at conferences world-wide, and write books about art direction and web design.

Design for successful digital products and websites

My name's **Andy Clarke**. I'm a well-known website designer, consultant, speaker, and writer on art direction and design for products and websites.

For 20 years, I've helped companies big and small to improve their website and product designs by providing consulting and design expertise.

My work is distinctive and original and I deliver engaging customer experiences and world-class designs.

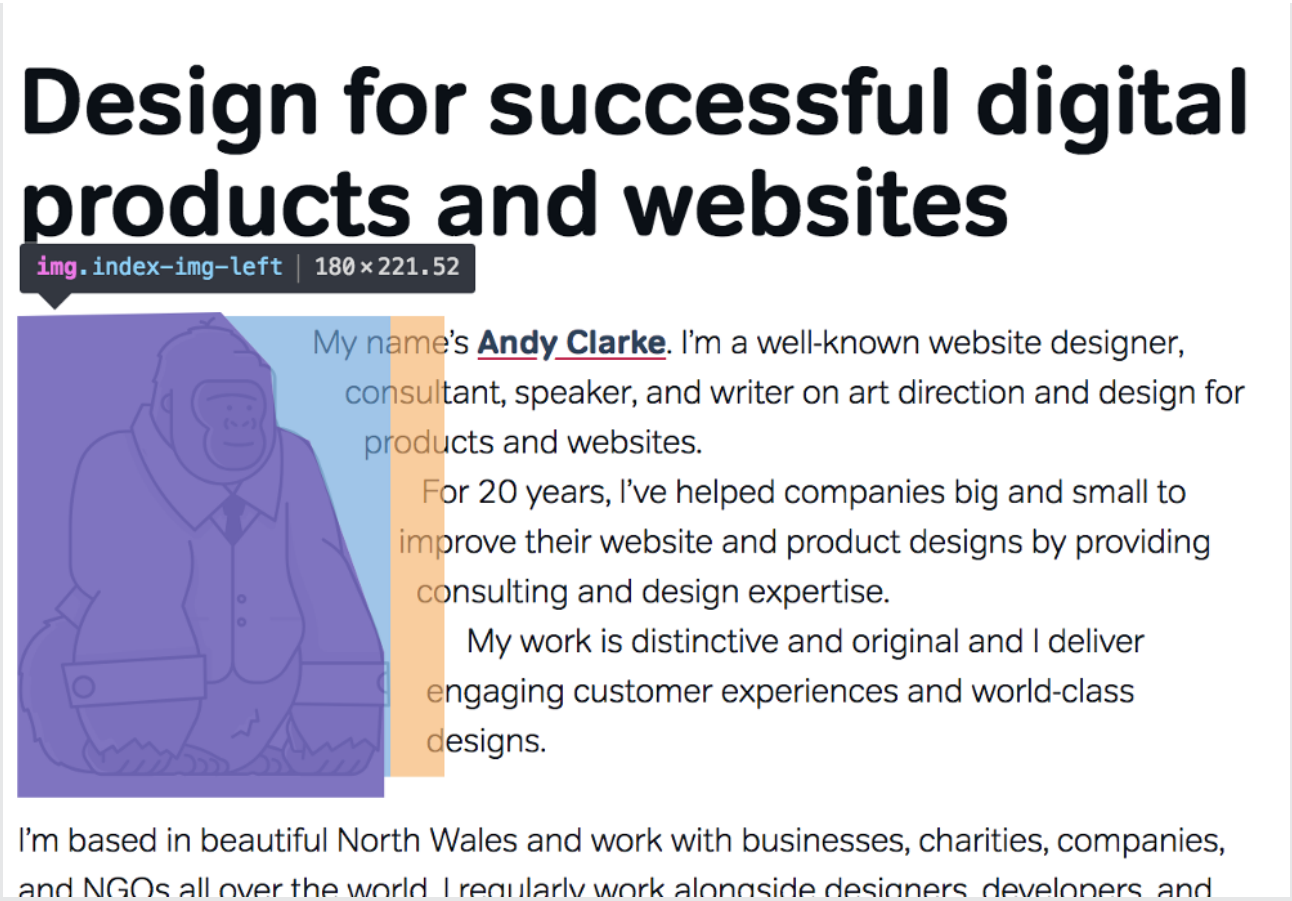
I'm based in beautiful North Wales and work with businesses, charities, companies, and NGOs all over the world. I regularly work alongside designers, developers, and leadership teams to create designs, processes, and systems which deliver results.

I consult with companies on art direction for the web, design systems, and style guides, speak at conferences world-wide, and write books about art direction and web design.

Shapes stehen in verschiedene Grundformen wie Kreisen oder Ellipsen sowie als Polygon und als URL-Funktion zur Verfügung. Welche Basis auch immer gewählt wird – ein Shape fungiert wie eine Schutzfläche, in die der Text nicht hineinfließen darf. An den Stellen wo kein Shape definiert ist, fließt Text optisch in ein Element hinein und nicht wie üblich an der Außenkante entlang.

CSS

```
.index-left-img {
shape-margin: 20px;
shape-outside: polygon(0 0,98px -2px,122px 24px,125px 53px,141px 60px,177px 162px,177px 232px,0 232px);
}
```



Detaillierte Informationen zur CSS Shape-Technik, findet ihr bei Tympanus.

Clip-Path

Mit der Clip-Path-Eigenschaft von CSS ist es möglich, Objekte mit Vektor-Masken optisch zu beschneiden. Häufig wird Clip-Path auch in Kombination mit Shapes eingesetzt. Für Clip-Path kann es sich mitunter als schwierig erweisen, eine solide Grundlage nach dem Prinzip des Progressive Enhancement zu schaffen, weshalb die Eigenschaft gerne in Kombination mit @supports zum Einsatz kommt.

Der Folgende CSS-Code schrägt mit Hilfe der Polygon-Funktion von Clip-Path den Kopfbereich einer Website an der Unterkante ab. Die Polygon-Funktion erwartet immer die X- und Y-Position der Polygon-Punkte. Im Beispiel sind im Uhrzeigersinn vier Punkte definiert, wobei der Punkt unten rechts mit 90 Prozent etwas nach oben gezogen wurde und somit die schräge Unterkante erzeugt wird.

CSS

```
header {
background: linear-gradient(135deg, lightblue, blue);
height: 50vh;
clip-path: polygon(0 0, 100% 0, 100% 90%, 0 100%);
}
```

Wenn Clip-Path nicht zur Verfügung steht, ist die Kante gerade ausgerichtet.