4 Visuelle Effekte

15 Einseitige Schatten

Das Problem

Eine der häufigsten Fragen zu `box-shadow` auf Q&A-Websites (z. B. **StackExchange** (*http://stackexchange.com/search?q=CSS*)) lautet: »Wie kann man einen Schatten für nur eine Seite (oder seltener auch zwei) definieren?« Eine schnelle Suche zeigt, dass es hierzu fast 1.000 Antworten gibt. Das erscheint sinnvoll, denn wenn der Schatten nur auf einer Seite erscheint, ist der Effekt subtiler, aber gleichermaßen realistisch. Oftmals schreiben die frustrierten Entwickler dann an die Mailingsliste der CSS-Arbeitsgruppe und fordern neue Eigenschaften wie `box-shadow-bottom`, um den gewünschten Effekt zu erzielen. Richtig angewandt, ist das aber auch jetzt schon mit dem guten alten `box-shadow` möglich.

Schatten nur auf einer Seite

Die meisten Leute benutzen `box-shadow` mit drei Längenangaben und einer Farbe wie hier:

```
box-shadow: 2px 3px 4px rgba(0,0,0,.5);
```

Die folgenden Schritte bieten eine gute (wenn auch technisch nicht ganz genaue) Möglichkeit, sich vorzustellen, wie der Schattenwurf erzeugt wird (**Abbildung 4.1**):

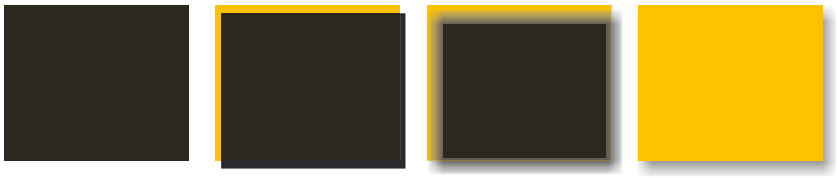


Abbildung 4.1
Beispielhaftes mentales Modell für die Erzeugung von Schattenwürfen mit `box-shadow`.

1. EinRechteck mit der Farbe `rgba(0,0,0,.5)` wird gezeichnet. Es hat die gleichen Dimensionen und die gleiche Position wie unser Element.
2. Das Rechteck wird um `2px` nach rechts und um `3px` nach unten verschoben.
3. Es wird um `4px` mit dem Gaußschen Weichzeichner (oder einem ähnlichen Effekt) weichgezeichnet. Das bedeutet, der Farbübergang an den Kanten des Schattens zwischen der Schattenfarbe und vollständiger Transparenz entspricht ungefähr dem doppelten Weichzeichnungsradius (in unserem Beispiel `8px`).
4. Das weichgezeichnete Rechteck **wird dort, wo es sich mit unserem Ausgangselement überlappt, beschnitten**, sodass der Schatten sich visuell »hinter« dem Element befindet. Das unterscheidet sich etwas von der allgemeinen Vorstellung, die viele Entwickler von Schatten haben (ein weichgezeichnetes Rechteck *hinter* dem Element). Für einige Anwendungsfälle ist es allerdings wichtig, zu wissen, dass **der Schatten nicht hinter dem Element erzeugt wird**. Wenn Sie das Element beispielsweise mit einem halbtransparenten Hintergrund versehen, ist dahinter kein Schatten zu sehen. Bei `text-shadow` ist das anders. Der hiermit erzeugte Schatten wird hinter dem Text nicht beschnitten.

Sofern nicht anders vermerkt, beziehen sich die Dimensionen eines Elements hier auf dessen *Border Box*, **nicht** auf die CSS-Höhe bzw. -Breite.

Die Verwendung eines Weichzeichnungsradius von `4px` bedeutet, dass der Schatten ungefähr `4px` größer ist als die Ausmaße unseres Elements. Dadurch ist der Schatten im Moment an allen Seiten des Elements sichtbar. Wenn wir die Verschiebungen (die ersten beiden Werte für `box-shadow`) auf mindestens `4px` erhöhen, können wir zumindest den oberen und linken Schatten verstecken. Das Ergebnis ist jedoch viel zu auffällig und nicht gerade schön (**Abbildung 4.2**). Aber auch, wenn das kein Problem wäre, wollten wir ja eigentlich nur an einer Seite einen Schattenwurf definieren.

Genauer gesagt, hat der Schatten folgende Breiten: `1px` an der Oberseite (`4px - 3px`), `2px` an der linken Seite (`4px - 2px`), `6px` auf der rechten Seite (`4px + 2px`) und `7px` an der Unterseite (`4px + 3px`). In der Praxis erscheint der Schatten jedoch kleiner, weil der Farbübergang an den Kanten nicht linear verläuft wie beispielsweise bei einem per `linear-gradient` erzeugten Farbverlauf.

Abbildung 4.2
Der Versuch, die Schattenwürfe oben und links zu verstecken, indem die Verschiebungen dem Weichzeichnungsradius entsprechen.

Die Lösung liegt in der Verwendung des weniger bekannten **vierten Größenparameters**, der nach dem Weichzeichnungsradius angegeben und als *Ausdehnungsradius* (engl. *spread radius*) bezeichnet wird. **Der Ausdehnungsradius erweitert oder verringert (bei negativen Werten) die Größe des Schattens basierend auf dem angegebenen Wert**. Wenn Sie beispielsweise einen Ausdehnungsradius von `-5px` angeben, verringern sich Breite und Höhe des Schattens um `10px` (`5px` an jeder Seite).

Wenn wir also einen negativen Ausdehnungsradius angeben, dessen absoluter Wert dem Weichzeichnungsradius entspricht, dann hat der Schatten exakt die gleiche Größe wie das Element, für das er definiert wurde. Sofern wir den Schatten nicht (anhand der ersten beiden Längenangaben) verschieben, **werden wir ihn überhaupt nicht sehen**. Geben wir aber einen positiven Wert für die vertikale Verschiebung an, beginnen wir, an der Unterseite des Elements einen Schatten zu sehen, an den übrigen Seiten jedoch nicht. Und genau das war unser Ziel:

Abbildung 4.3
`box-shadow`nur an der Unterseite des Elements.

```
box-shadow: 0 5px 4px -4px black;
```

Das Ergebnis sehen Sie in **Abbildung 4.3**.

▸ **PLAY!** play.csssecrets.io/**shadow-one-side**

Schatten an zwei benachbarten Seiten

Abbildung 4.4
`box-shadow` an zwei benachbarten Seiten.

Eine weitere Frage dreht sich um die Definition von Schatten für zwei Seiten. Wenn zwei Seiten benachbart sind (z. B. rechts und unten), ist das einfacher: Sie können entweder einen Effekt wie in **Abbildung 4.2** verwenden oder den Trick aus dem vorigen Abschnitt abwandeln. Dabei gibt es folgende Unterschiede:

- Der Schatten soll nicht verkleinert werden, um die Weichzeichnung auf beiden Seiten auszugleichen, sondern nur auf einer. Anstatt dem Ausdehnungsradius den gegenteiligen Wert des Weichzeichnungsradius zu geben, wird er halbiert.

- Wir benötigen beide Werte für die Verschiebung, da wir den Schatten horizontal und vertikal verschieben müssen. Ihr Wert muss größer oder gleich dem Weichzeichnungsradius sein, da der Schatten für die anderen beiden Seiten versteckt werden soll.

  Hier ein Beispiel für die Defintion eines schwarzen (`black`) `6px` großen Schattens für die rechte und die Unterseite:

```
box-shadow: 3px 3px 6px -3px black;
```

Das Ergebnis sehen Sie in **Abbildung 4.4**.

▸ **PLAY!** play.csssecrets.io/**shadow-2-sides**

Schatten auf zwei gegenüberliegenden Seiten

Etwas kniffliger wird es, wenn der Schatten auf zwei gegenüberliegenden Seiten erscheinen soll, z. B. links und rechts. Da der Ausdehnungsradius für alle Seiten gleich ist (d. h., der Schatten kann nicht horizontal vergrößert und gleichzeitig vertikal verkleinert werden), besteht die einzige Möglichkeit darin, **für jede Seite einen separaten Schatten** zu definieren. Danach wenden wir den Trick aus dem **Abschnitt »Schatten nur auf einer Seite« auf Seite 138** für jede Seite einzeln an:

Es gibt Diskussionen in der CSS-Arbeitsgruppe, ob **separate Werte für den horizontalen und vertikalen Ausdehnungsradius** in der Zukunft möglich sein sollen, was diese Aufgabe vereinfachen würde.

```
box-shadow: 5px 0 5px -5px black,
           -5px 0 5px -5px black;
```

Abbildung 4.5
`box-shadow` an zwei gegenüberliegenden Seiten.

Das Ergebnis sehen Sie in **Abbildung 4.5**.

▸ **PLAY!** play.csssecrets.io/**shadow-opposite-sides**

Verwandte Spezifikationen

- **CSS Backgrounds & Borders** (*http://w3.org/TR/css-backgrounds*)

16 Unregelmäßige Schattenwürfe

Voraussetzungen

box-shadow

Das Problem

`box-shadow` funktioniert wunderbar, wenn ein Schatten für ein Rechteck oder eine Form definiert werden soll, die mit `border-radius` (einige Beispiele hierzu finden Sie im **Abschnitt »Flexible Ellipsen« auf Seite 80**) erstellt wurden. Das funktioniert aber nur eingeschränkt, sobald Sie **Pseudoelemente oder halbtransparente Stile verwenden**, weil `box-shadow` Transparenz schlicht ignoriert. Ein paar Beispiele hierzu:

- Halbtransparente Bilder, Hintergrundbilder oder per `border-image` eingebundene Grafiken (z. B. ein alter goldener Bilderrahmen)
- Gepunktete, gestrichelte oder halbtransparente Rahmen ohne Hintergrund (oder wenn `background-clip` einen anderen Wert als `border-box` hat)
- Sprechblasen, deren Pfeil mit einem Pseudoelement erzeugt wurde

- Angeschnittene Ecken wie die im **Abschnitt »Abgeschnittene Ecken« auf Seite 100**
- Die meisten »Eselsohren«-Effekte, inklusive dem Beispiel weiter hinten in diesem Kapitel
- Mit `clip-path` erzeugten Formen wie die Rauten aus dem **Abschnitt »Rauten« auf Seite 94**

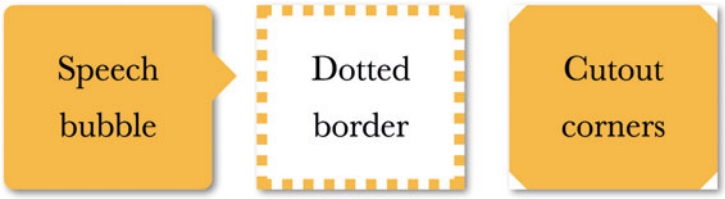


Abbildung 4.6
Elemente mit CSS-Stilen, bei denen `box-shadow` nicht funktioniert; der `box-shadow`-Wert für diese Beispiele war `2px 2px 10px rgba(0,0,0,.5)`.

Der Versuch einer Definition von `box-shadow` ist hier nicht zufriedenstellend, wie in **Abbildung 4.6** zu sehen ist. Gibt es für diese Fälle eine Lösung, bei der die Schatten korrekt funktionieren?

Die Lösung

Die Spezifikation für Filter-Effekte (**Filter Effects** (*`http://w3.org/TR/filter-effects`*)) bietet mit der von SVG entlehnten `filter`-Eigenschaft eine Lösung für dieses Problem. Und obwohl die CSS-Filter grundsätzlich den **SVG-Filtern** entsprechen, wird hierfür **kein SVG-Wissen benötigt**. Stattdessen werden sie über eine Reihe bequemer Funktionen gesteuert wie `blur()`, `grayscale()` oder – Achtung! Tusch! – `drop-shadow()`! Diese Filter lassen sich bei Bedarf sogar miteinander kombinieren, indem sie durch Leerzeichen voneinander getrennt angegeben werden wie hier:

```
filter: blur() grayscale() drop-shadow();
```

Der `drop-shadow()`-Filter funktioniert wie eine einfache Version von `box-shadow`, nur ohne die Möglichkeit, einen Ausdehnungsradius oder das Schlüsselwort `inset` anzugeben. Anstelle von:

```
box-shadow: 2px 2px 10px rgba(0,0,0,.5);
```

… würden wir jetzt schreiben:

```
filter: drop-shadow(2px 2px 10px rgba(0,0,0,.5));
```

Die beiden Eigenschaften verwenden möglicherweise unterschiedliche Filter-Algorithmen. Unter Umständen müssen Sie den Wert für die Weichzeichnung anpassen!

Wenn wir den `drop-shadow()`-Filter auf die gleichen Elemente wie in **Abbildung 4.6** anwenden, erhalten wir das Ergebnis in **Abbildung 4.7**.

Abbildung 4.7
Ein `drop-shadow()`-Filter, angewandt auf die Elemente in **Abbildung 4.6**.

Das Beste an CSS-Filtern ist, dass sie **nichts kaputtmachen**, wenn es keine Browserunterstützung gibt. Der Effekt wird einfach nicht angewandt. **Etwas besser wird die Browserunterstützung, indem Sie zusätzlich einen SVG-Filter verwenden**, damit der Effekt in möglichst vielen Browsern funktioniert. Welche SVG-Filter den einzelnen CSS-Filterfunktionen entsprechen, können Sie in der **Filter Effects-Spezifikation** (*http://www.w3.org/TR/filter-effects/*) nachlesen. Sie können den SVG-Filter und die vereinfachte CSS-Version nebeneinander angeben, wobei die Kaskade sich darum kümmert, welcher Filter tatsächlich benutzt wird:

```
filter: url(drop-shadow.svg#drop-shadow);
filter: drop-shadow(2px 2px 10px rgba(0,0,0,.5));
```

Wenn sich der SVG-Filter in einer separaten Datei befindet, lässt er sich allerdings nicht so gut anpassen wie die menschenfreundliche Funktion im CSS-Code. Ist der Filter dagegen eingebettet, verstopft er schnell den

Code. Die Parameter sind in der Datei festgelegt, und es ist nicht praktikabel, mehrere Dateien zu verwenden, wenn Sie einen leicht unterschiedlichen Schattenwurf brauchen. Wir könnten Daten-URIs benutzen (und so den zusätzlichen HTTP-Requests sparen), was die Dateigröße aber trotzdem erhöht. Da es sich hier um eine Fallback-Lösung handelt, ist es wenig sinnvoll, verschiedene Variationen zu verwenden, selbst wenn die `drop-shadow()`-Filter sich leicht unterscheiden.

Außerdem sollten Sie nicht vergessen, dass **jeder nicht-transparente Bereich unterschiedslos einen Schatten wirft**, inklusive Text (sofern Ihr Hintergrund transparent ist), wie Sie bereits in **Abbildung 4.7** gesehen haben. Vielleicht glauben Sie, dass Sie das durch die Verwendung von `text-shadow: none;` ausgleichen können. Tatsächlich ist `text-shadow` jedoch vollkommen unabhängig von anderen Regeln und wird die Auswirkungen von `drop-shadow()` auf einen Text nicht unwirksam machen. Wenn Sie also `text-shadow` benutzen, um den Text mit einem Schatten zu versehen, dann **erhält der mit `text-shadow` erzeugte Schatten sogar seinerseits einen Schatten**! Sehen Sie sich hierzu den folgenden CSS-Code (und entschuldigen Sie bitte, dass das Ergebnis so kitschig ist – es soll das Problem so drastisch zeigen wie möglich):

Abbildung 4.8
`text-shadow` verwendet für die Schattenwürfe ebenfalls einen `drop-shadow()`-Filter.

```
color: deeppink;
border: 2px solid;
text-shadow: .1em .2em yellow;
filter: drop-shadow(.05em .05em .1em gray);
```

In **Abbildung 4.8** sehen Sie ein Beispiel, in dem sowohl der mit `text-shadow` als auch der mit `drop-shadow()` zusätzlich erzeugte Schattenwurf sichtbar sind.

▸ **PLAY!** play.csssecrets.io/**drop-shadow**

Verwandte Spezifikationen

- **Filter Effects** (*http://w3.org/TR/filter-effects*)

17 Farbtönungen

Voraussetzungen

Das HSL-Farbmodell, `background-size`

Das Problem

Die Verwendung einer Farbtönung für Graustufenbilder (oder Bilder, die in Graustufen umgewandelt wurden) ist eine beliebte Methode, um mehreren Fotos mit sehr unterschiedlichen Stilen eine visuelle Einheit zu geben. Oft wird dieser Effekt statisch angewandt und bei `:hover` oder einer anderen Interaktion wieder entfernt.

Traditionell würden wir ein Bildbearbeitungsprogramm benutzen, um die zwei Versionen des Bilds zu erstellen, die dann bei Bedarf mit ein wenig CSS-Code ausgetauscht werden. Das funktioniert, führt aber auch zu größeren Datenmengen und zusätzlichen HTTP-Requests. Außerdem ist die Pflege ein Albtraum. Stellen Sie sich vor, was passiert, wenn die Effektfarbe

geändert werden soll: Sie müssten sämtliche Bilder erneut bearbeiten, um neue monochrome Versionen zu erstellen!

Abbildung 4.9
Die CSSConf 2014-Website benutzte diesen Effekt für die Sprecherfotos, zeigte aber bei `:hover` und `:focus` das komplett farbige Bild.

Ein weiterer Effekt überlagert das Bild mit einer halbtransparenten Farbe, oder er benutzt die Eigenschaft `opacity` und hinterlegt das Bild mit einer Vollfarbe. Das ist aber keine richtige Tönung: Es fehlt die Umwandlung der Bildfarben in Schattierungen der Zielfarbe, außerdem wird der Kontrast erheblich reduziert.

Es gibt ein paar Skripte, die das Bild in ein `<canvas>`-Element konvertieren und die Tönung dann per JavaScript durchführen. Hierdurch ist die Tönung zwar korrekt, aber die Technik ist sehr langsam und ziemlich eingeschränkt.

Wäre es nicht viel einfacher, die Tönung direkt im CSS-Code auf die Bilder anwenden zu können?

Filterbasierte Lösung

Da es für diesen Effekt keinen speziellen Einzelfilter gibt, müssen wir uns ein bisschen schlauer anstellen und **mehrere Filter kombinieren**.

Abbildung 4.10
Oben: Originalbild.
Unten: Bild nach Verwendung des `sepia()`-Filters.

Zuerst benutzen wir einen `sepia()`-Filter, was dem Bild eine **entsättigte orange-gelbe Färbung** verleiht, bei der die meisten Pixel einen Hue-Wert von 35–40 haben. Ist dies sowieso die gewünschte Farbe, sind Sie bereits fertig, was jedoch eher selten der Fall ist. Soll die Zielfarbe eine höhere Sättigung haben, können Sie zusätzlich den `saturate()`-Filter benutzen, um die Sättigung der einzelnen Pixel zu erhöhen. Angenommen, das Bild soll eine Färbung von `hsl(335, 100%, 50%)` bekommen. Dann müssten wir die Sättigung deutlich erhöhen. Wir benutzen hier den Parameter `4`, wobei der genaue Wert von Fall zu Fall unterschiedlich sein wird. Sie werden also nach Augenmaß entscheiden müssen. Wie in diesem Beispiel gezeigt, erzeugt die Kombination der Filter einen **warm-goldenen Farbton**.

So schön unser Bild auch aussieht. Wir wollten keinen orange-gelben, sondern eher ein tiefes, helles Pink haben. Also brauchen wir zusätzlich einen `hue-rotate()`-Filter, um **den Farbton für jedes Pixel** um die angegebene Gradzahl **zu verschieben**. Um den Hue-Wert von 40 nach 33 zu verschieben, müssen wir ca. 295 Grad (335 – 40) addieren:

Abbildung 4.11
Unser Bild nach Verwendung eines zusätzlichen `saturate()`-Filters.

```
filter: sepia() saturate(4) hue-rotate(295deg);
```

Jetzt hat das Bild das gewünschte Aussehen. Das Ergebnis sehen Sie in **Abbildung 4.12**. Soll der Effekt per `:hover` ausgelöst werden, könnten wir sogar noch einen CSS-Übergang (transition) dafür definieren:

Abbildung 4.12
Unser Bild nach Verwendung eines zusätzlichen `hue-rotate()`-Filters.

```
img {
    transition: .5s filter;
    filter: sepia() saturate(4) hue-rotate(295deg);
}

img:hover,
img:focus {
    filter: none;
}
```

▸ **PLAY!** play.csssecrets.io/**color-tint-filter**

Lösung mit Mischmodi (»blending modes«)

LIMITED SUPPORT

Die `filter()`-Lösung funktioniert zwar, aber das Ergebnis entspricht nicht ganz dem, was mit einem Bildbearbeitungsprogramm möglich gewesen wäre. Selbst wenn wir versuchen, eine sehr helle Farbe für die Tönung zu verwenden, wirkt das Ergebnis ziemlich **verwaschen**. Versuchen wir, das mit dem `saturate()`-Filter auszugleichen, erhalten wir einen **anderen, überstilisierten Effekt**. Zum Glück gibt es noch einen anderen Weg: **Mischmodi** (*Blending Modes*)!

Wenn Sie jemals ein Bildbearbeitungsprogramm wie Adobe Photoshop benutzt haben, sollten Sie die Mischmodi bereits kennen. Wenn sich zwei Elemente überschneiden, können Sie über den Mischmodus festlegen, **mit welcher Methode die Farben des obersten Elements mit denen des darunterliegenden Elements gemischt werden**. Wenn Sie Bilder tönen wollen, brauchen Sie den Mischmodus Luminanz (`luminosity`). Dieser Mischmodus **erhält die HSL-Helligkeit des obersten Elements, während Farbton und Sättigung des darunterliegenden Elements übernommen werden**. Entspricht das darunterliegende Element unserer Zielfarbe und das obere Element unserem Bild, haben wir unser Ziel doch erreicht, oder?

Abbildung 4.13
Vergleich zwischen der Filtermethode (oben) und der Mischmodusmethode (unten).

Um ein Element mit einem Mischmodus zu versehen, stehen uns zwei Eigenschaften zur Verfügung: `mix-blend-mode` für Mischmodi, die **für komplette Elemente** gelten sollen, und `background-blend-mode`, um Mischmodi **auf jede Hintergrundebene separat** anzuwenden. Wenn Sie mit dieser Methode ein Bild bearbeiten, haben Sie zwei Möglichkeiten, die aber beide nicht ideal sind:

- Umgeben Sie das Bild mit einem Container, dessen Hintergrund die von uns gewünschte Farbe hat.
- Verwenden Sie anstelle des Bilds ein `<div>`-Element, dessen `background-image` das gewünschte Bild ist, und eine zweite Hintergrundebene mit der gewünschten Farbe.

Je nach Anwendungsfall können wir eine der beiden Methoden wählen. Soll der Effekt auf ein `<img>`-Element angewandt werden, müssen wir es mit einem weiteren Element umgeben. Gibt es allerdings schon ein weiteres Element, z. B. `<a>`, können wir auch dieses verwenden:

HTML

```
<a href="#something">
    <img src="tiger.jpg" alt="Rawrrr!" />
</a>
```

Danach reichen zwei Deklarationen, um den gewünschten Effekt zu erzielen:

```
a {
    background: hsl(335, 100%, 50%);
}

img {
    mix-blend-mode: luminosity;
}
```

Wie bei CSS-Filtern, gibt es keine Probleme, wenn Mischmodi nicht unterstützt werden. Der Effekt wird einfach nicht angewandt, aber das Bild bleibt weiterhin sichtbar.

Hierbei sollte man noch beachten: **Filter sind animierbar, Mischmodi aber nicht**. Zwar können Sie einen CSS-Übergang benutzen, um die `filter`-Eigenschaft zu animieren und so eine langsame Überblendung zum monochromen Bild zu erreichen. Mit Mischmodi funktioniert das jedoch nicht. Das heißt aber nicht, dass Animationen überhaupt infrage kommen. Wir müssen nur ein bisschen um die Ecke denken.

Wie bereits erklärt, mischt `mix-blend-mode` das gesamte Element mit dem, was sich darunter befindet. Verwenden wir hierbei den Wert `luminosity`, wird das Bild grundsätzlich mit **irgendetwas** gemischt. Verwenden wir dagegen die Eigenschaft `background-blend-mode`,

wird jede Hintergrundbild-Ebene mit den darunterliegenden gemischt, ohne zu wissen, was außerhalb des Elements passiert. Was passiert wohl, wenn wir nur ein Hintergrundbild haben und als Hintergrundfarbe `transparent` gewählt wurde? Richtig: **zunächst einmal gar nichts**!

Diese Erkenntnis können wir uns zunutze machen, wenn wir `background-blend-mode` für unseren Effekt benutzen. Dafür müssen wir den HTML-Code ein wenig umschreiben:

HTML

```
<div class="tinted-image"
     style="background-image:url(tiger.jpg)">
</div>
```

Danach müssen wir das `<div>`-Element nur noch mit etwas CSS versehen. Weitere Elemente werden bei dieser Technik nicht gebraucht:

```
.tinted-image {
    width: 640px; height: 440px;
    background-size: cover;
    background-color: hsl(335, 100%, 50%);
    background-blend-mode: luminosity;
    transition: .5s background-color;
}

.tinted-image:hover {
    background-color: transparent;
}
```

Wie bereits gesagt, ist trotzdem **keine dieser beiden Techniken ideal**. Die größten Probleme hierbei sind:

- **Höhe und Breite des Bilds müssen** im CSS-Code **hart kodiert werden**.
- **Semantisch gesehen**, ist dies kein Bild und wird von Screen-Readern auch nicht als solches behandelt.

Wie bei den meisten Dingen im Leben gibt es keine perfekte Methode, das Ziel zu erreichen. In diesem Abschnitt haben wir drei Möglichkeiten kennengelernt, den Effekt zu erzielen, die alle ihre Vor- und Nachteile haben. Welchen Weg Sie wählen, sollte von den jeweiligen Vorgaben Ihres Projekts abhängen.

▸ **PLAY!** `play.csssecrets.io/color-tint`

Hut ab

Hut ab vor ***Dudley Storey*** *(`http://demosthenes.info`), der sich den* ***Animationstrick für Mischmodi*** *(`http://demosthenes.info/blog/888/Create-Monochromatic-Color-Tinted-Images-With-CSS-blend`) ausgedacht hat.*

Verwandte Spezifikationen

- **Filter Effects** (*`http://w3.org/TR/filter-effects`*)
- **Compositing and Blending** (*`http://w3.org/TR/compositing`*)
- **CSS Transitions** (*`http://w3.org/TR/css-transitions`*)

18 Milchglas-Effekt

Voraussetzungen

RGBA-/HSLA-Farben

Das Problem

Der Begriff *Backdrop* (im Gegensatz zum *Hintergrund*) steht hier für den **Teil der Seite, der sich visuell hinter einem Element** befindet und durch einen halbtransparenten Hintergrund durchscheinen würde.

Einer der ersten Anwendungsfälle für halbtransparente Farben waren Hintergründe vor fotografischen oder anderweitig unruhigen Backdrops, um den Kontrast zu erhöhen und den Text lesbar zu machen. Das Ergebnis ist recht eindrucksvoll, kann aber immer noch schwer zu lesen sein. Das gilt besonders bei stark durchscheinenden Farben und/oder sehr unruhigen Backdrops. Ein Beispiel sehen Sie in **Abbildung 4.14**. Hier hat das Hauptelement einen halbtransparenten weißen Hintergrund. Der Markup-Code sieht so aus:

HTML

```
<main>
    <blockquote>
```

```
        »The only way to get rid of a temptation[…]«
        <footer>–
            <cite>
                Oscar Wilde,
                The Picture of Dorian Gray
            </cite>
        </footer>
    </blockquote>
</main>
```

Und hier der dazugehörige CSS-Code (die unwichtigen Teile haben wir aus Gründen der Klarheit weggelassen):

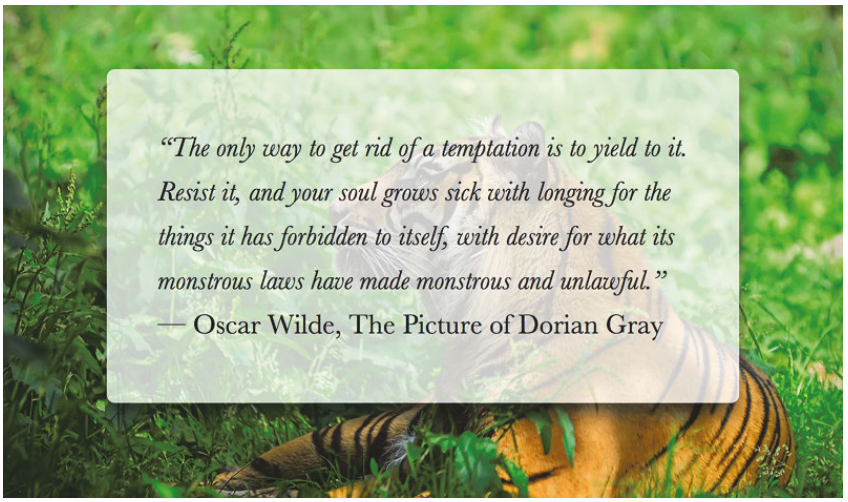
```
body {
    background: url("tiger.jpg") 0 / cover fixed;
}

main {
    background: hsla(0,0%,100%,.3);
}
```

Abbildung 4.14
Der halbtransparente weiße Hintergrund erschwert das Lesen des Texts.

Wie man sehen kann, ist der Text wirklich schwer zu lesen, weil das dahinterliegende Bild (der Backdrop) sehr unruhig und die Hintergrundfarbe nur zu 25% opak ist. Wir könnte die Lesbarkeit erhöhen, indem wir den Wert für den Alphakanal der Hintergrundfarbe erhöhen. Dadurch wirkt der Effekt aber längst nicht mehr so interessant wie vorher (siehe **Abbildung 4.15**).

Abbildung 4.15
Ein höherer Wert für den Alphakanal erleichtert das Lesen, macht das Design aber auch weniger interessant.



Im traditionellen Print-Design wird dieses Problem oft gelöst, indem **der Teil des Fotos hinter dem Textcontainer weichgezeichnet wird**. Weichgezeichnete Hintergründe wirken ruhiger und machen so den davorliegenden Text leichter lesbar. Da Weichzeichnung jedoch viel Rechenleistung benötigt, war diese Technik für Websites und im UI-Design nur beschränkt verwendbar. Allerdings werden die GPUs immer besser, und Hardware-Beschleunigung kann heutzutage für immer mehr Dinge eingesetzt werden. In den letzten Jahren kam diese Technik sowohl unter Microsoft Windows wie auch bei Apple iOS und Mac OS X (**Abbildung 4.16**) zum Einsatz.

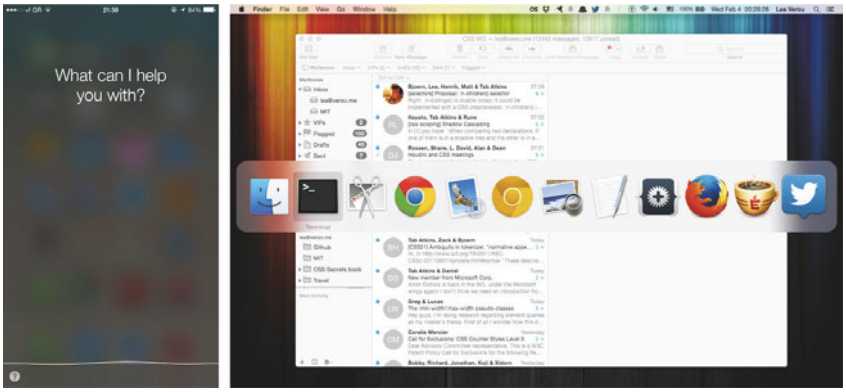


Abbildung 4.16
Durchscheinende UIs mit einem weichgezeichneten Backdrop sind in den letzten Jahren immer häufiger geworden, weil die Ressourcen für die Weichzeichnung deutlich günstiger geworden sind (**links: Apple iOS 8.1**, **rechts: Apple OS X Yosemite**).

Auch in CSS haben wir die Möglichkeit, Elemente mit dem `blur()`-Filter weichzuzeichnen. Dies ist im Prinzip eine hardwarebeschleunigte Version des entsprechenden SVG-Weichzeichnungsfilters, den es für SVG-Elemente schon immer gab. Wenden wir `blur()` direkt auf unser Beispiel an, wird allerdings das gesamte Element weichgezeichnet, wodurch es sogar noch schlechter lesbar wird (**Abbildung 4.17**). Gibt es eine Möglichkeit, die Weichzeichnung nur auf den Backdrop des Elements (den Teil des Hintergrunds, der sich **hinter** dem Element befindet) anzuwenden?

Abbildung 4.17
Die Anwendung eines `blur()`-Filters auf das Element macht die Dinge noch schlimmer.

Die Lösung

Das ist sogar mit nicht-fixierten Hintergründen möglich, allerdings etwas unordentlicher.

Hat `background-attachment` für unser Element den Wert `fixed`, so gibt es eine, wenn auch etwas komplizierte, Lösung. Da wir die Weichzeichnung nicht direkt auf unser Element anwenden können, **benutzen wir ein Pseudoelement, das hinter dem Element platziert wird und dessen Hintergrund sich 1:1 mit dem von `<body>` deckt**.

Zu Beginn fügen wir das Pseudoelement hinzu und positionieren es absolut. Hierbei werden sämtliche Abstände auf `0` gesetzt, wodurch das gesamte `<main>`-Element verdeckt wird:

```
main {
    position: relative;
    /* [Weitere Stildefinitionen] */
}

main::before {
    content: '';
    position: absolute;
    top: 0; right: 0; bottom: 0; left: 0;
    background: rgba(255,0,0,.5); /* Zum Debuggen */
}
```

Seien Sie vorsichtig, wenn Sie ein Kindelement per `z-index` hinter sein Elternelement verschieben wollen: Ist das besagte Elternelement in andere Elemente mit Hintergründen verschachtelt, wird das Kindelement auch unter diese verschoben.

Warum verwenden wir hier nicht `background: inherit;` für `main::before`? Weil das Pseudoelement dann von `main` anstelle von `body` erbt, also ebenfalls einen halbtransparenten weißen Hintergrund erhält.

Wir haben hier zusätzlich einen halbtransparenten Hintergrund in der Farbe `red` definiert, um besser sehen zu können, was passiert. Ansonsten kann das Debugging mit transparenten (und daher unsichtbaren) Elementen ziemlich schwierig werden. Wie Sie in **Abbildung 4.18** sehen können, befindet sich unser Pseudoelement im Moment noch **vor** dem Inhalt und verdeckt ihn dadurch. Das können wir durch die Deklaration `z-index: -1;` in Ordnung bringen (**Abbildung 4.20**).

Jetzt ist es Zeit, den halbtransparenten roten Hintergrund durch den tatsächlichen Backdrop zu ersetzen. Entweder kopieren wir hierfür den Hintergrund von `<body>` oder definieren eine eigene Regel. Können wir jetzt weichzeichnen? Versuch macht klug:

```
body, main::before {
    background: url("tiger.jpg") 0 / cover fixed;
}

main {
    position: relative;
    background: hsla(0,0%,100%,.3);
}

main::before {
    content: '';
    position: absolute;
    top: 0; right: 0; bottom: 0; left: 0;
    filter: blur(20px);
}
```

Wie in **Abbildung 4.21** zu sehen ist, haben wir es fast geschafft. Der Weichzeichnungs-Effekt sieht in der Mitte bereits perfekt aus, zu den Ecken hin ist er aber noch zu schwach. Das liegt daran, dass der von einer Vollfarbe bedeckte Bereich durch die Weichzeichnung um den angegebenen Weichzeichnungsradius verkleinert wird. Geben wir unserem Pseudoelement einen roten (red) Hintergrund, sehen wir schnell, was hier passiert (**Abbildung 4.22**).

Abbildung 4.18
Das Pseudoelement verdeckt im Moment noch den Text.

Abbildung 4.19
Die fehlende Weichzeichnung an den Kanten ist behoben, dafür erstreckt sich die Weichzeichnung jetzt auch außerhalb des Elements.

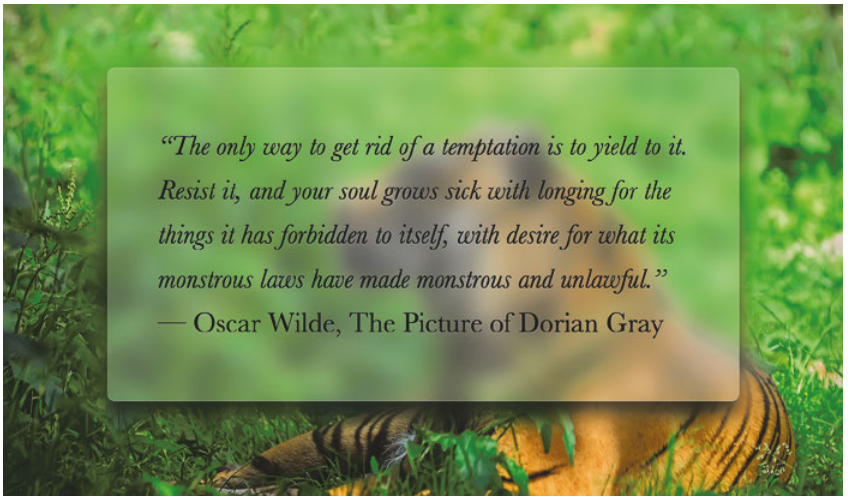
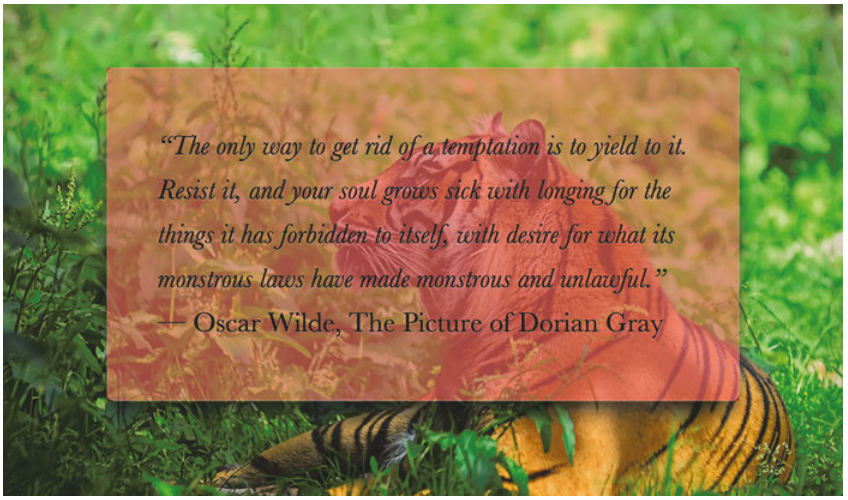


Abbildung 4.20
Das Pseudoelement wird per `z-index: -1;` hinter sein Elternelement verschoben.



Um dieses Problem zu umgehen, machen wir das Pseudoelement um **mindestens `20px` größer als die Höhe und Breite seines Containers**, indem wir einen Außenabstand von **`-20px`** oder weniger definieren. So sind wir auf der sicheren Seite, auch wenn verschiedene Browser unterschiedliche Algorithmen für die Weichzeichnung verwenden. Wie **Abbildung 4.19** zeigt, ist das Problem mit der mangelnden Weichzeichnung an den Rändern zwar behoben. Allerdings ist der Bereich **außerhalb des Containers jetzt ebenfalls teilweise weichgezeichnet**. Das sieht weniger nach Milchglas, sondern eher verwischt aus. Glücklicherweise lässt sich

aber auch das Problem leicht beheben, indem wir `overflow: hidden;` für das `main`-Element definieren und die überflüssige Weichzeichnung einfach beschneiden. Der fertige Code sieht aus wie folgt (**Abbildung 4.23**):

```
body, main::before {
    background: url("tiger.jpg") 0 / cover fixed;
}

main {
    position: relative;
    background: hsla(0,0%,100%,.3);
    overflow: hidden;
}

main::before {
    content: '';
    position: absolute;
    top: 0; right: 0; bottom: 0; left: 0;
    filter: blur(20px);
    margin: -30px;
}
```

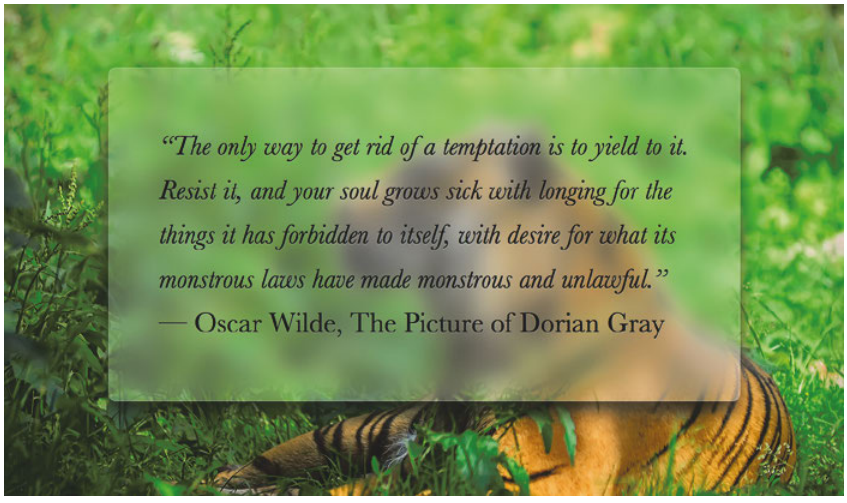


Abbildung 4.21
Die Weichzeichnung des Pseudoelements funktioniert bereits. Allerdings ist sie an den Kanten abgeschwächt, wodurch der Milchglas-Effekt ebenfalls verringert wird.

Abbildung 4.22
Die Definition eines roten Hintergrunds hilft beim Aufspüren des Problems.

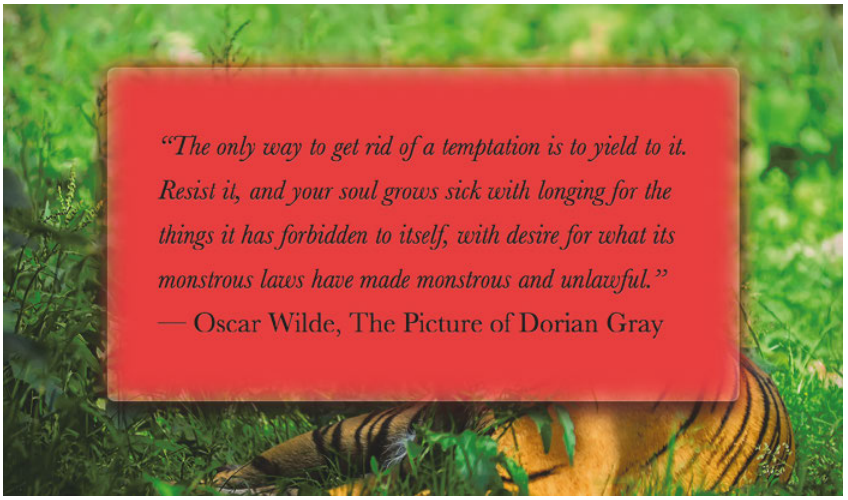
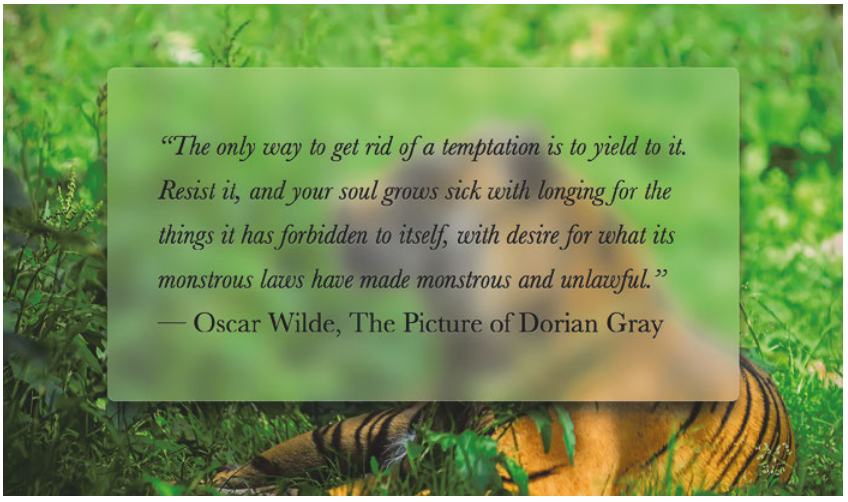


Abbildung 4.23
Unser Endergebnis.



Wie Sie sehen, ist unsere Seite jetzt viel besser lesbar, und sie wirkt dabei auch noch viel eleganter als vorher. Man kann sich darüber streiten, ob der Fallback für diesen Effekt wirklich als »Graceful Degradation« betrachtet werden kann. Werden die Filter nicht unterstützt, erhalten wir wieder das Ergebnis vom Anfang (**Abbildung 4.14**). Wir können den Fallback etwas lesbarer machen, indem wir die Hintergrundfarbe weniger durchscheinend machen.

▸ **PLAY!** play.csssecrets.io/**frosted-glass**

Verwandte Spezifikationen

- **Filter Effects**
  (*http://w3.org/TR/filter-effects*)

19 Eselsohren-Effekt (abgeknickte Ecken)

Voraussetzungen

CSS-Transforms, CSS-Farbverläufe, das »Innen abgerundete Ecken«-Secret auf Seite 40

Das Problem

Die Ecke eines Elements (üblicherweise rechts oben oder unten) soll so dargestellt werden, dass sie **abgeknickt** (als »Eselsohr«) erscheint. Mal mehr, mal weniger realitätsnah ist dies seit einigen Jahren ein sehr populäres Designelement.

Heutzutage gibt es hierfür eine **Vielzahl hilfreicher CSS-Lösungen**. Die erste wurde bereits 2010 vom Pseudoelemente-Meister **Nicolas Gallagher** (*http://http//nicolasgallagher.com/pure-css-folded-corner-effect*) veröffentlicht. Hierbei wurden links oben normalerweise zwei Dreiecke hinzugefügt: eines für das Eselsohr und ein weißes, um die Ecke des Hauptelements zu verstecken. Die Dreiecke wurden meistens mit dem alten `border`-Trick erzeugt.

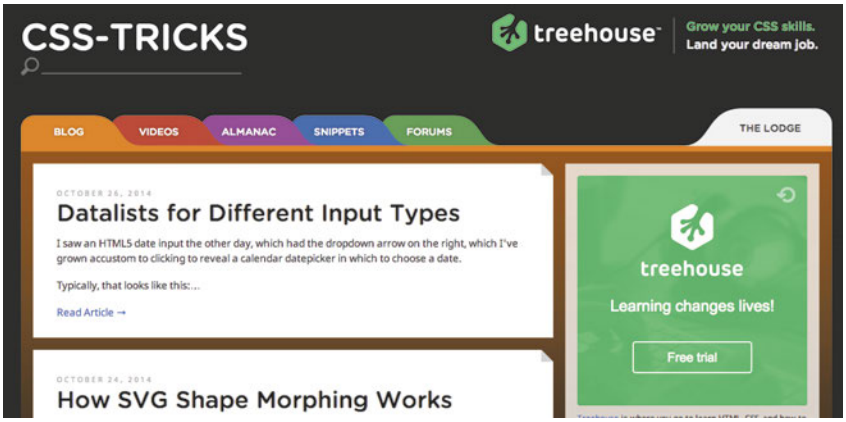


Abbildung 4.24
Bei mehreren früheren Redesigns von `css-tricks.com` wurden Eselsohren in der rechten oberen Ecke jeder Artikelbox verwendet.

Obwohl diese Lösungen damals ziemlich beeindruckend waren, haben sie heute eine Reihe von Problemen und Einschränkungen:

- Wenn der Hintergrund keine opake Farbe, sondern ein Muster, ein Foto, ein Farbverlauf oder eine andere Art von Hintergrundbild ist
- Wenn ein anderer Winkel als 45 Grad und/oder ein gedrehtes Eselsohr benutzt werden soll

Gibt es eine Möglichkeit, einen Eselsohren-Effekt erzielen, der auch in diesen Fällen funktioniert?

Die 45°-Lösung

Wir beginnen mit einem Element, dessen rechte obere Ecke »abgeschnitten« ist. Hierfür benutzen wir die im **Abschnitt »Abgeschnittene Ecken« auf Seite 100** beschriebene Technik. Um rechts oben eine `1em` große angeschnittene Ecke zu erzeugen, kommt der folgende Code zum Einsatz. Ein Beispiel für die Darstellung sehen Sie in **Abbildung 4.25**:

> "The only way to get rid of a temptation is to yield to it."
> —Oscar Wilde, The Picture of Dorian Gray

Abbildung 4.25
Unser Startpunkt: ein Element, dessen rechte obere Ecke mithilfe eines Farbverlaufs »abgeschnitten« wurde.

```
background: #58a; /* Fallback */
background:
    linear-gradient(-135deg, transparent 2em, #58a 0);
```

Damit sind wir auch schon halb fertig. Wir müssen nur noch **ein dunkleres Dreieck für das Eselsohr hinzufügen**. Um das nötige Dreieck zu erzeugen, **benutzen wir einen weiteren Farbverlauf**, dessen Größe wir nach Bedarf per `background-size` anpassen und **in der rechten oberen Ecke** platzieren.

Um das Dreieck zu erzeugen, benötigen wir nur einen abgewinkelten linearen Farbverlauf mit zwei Stops, die sich in der Mitte treffen:

```
background:
    linear-gradient(to left bottom,
        transparent 50%, rgba(0,0,0,.4) 0)
        no-repeat 100% 0 / 2em 2em;
```

"The only way to get rid of a temptation is to yield to it."
—Oscar Wilde, The Picture of Dorian Gray

Benutzen wir **nur** diesen Hintergrund, erhalten wir das Ergebnis in **Abbildung 4.26**. Der letzte Schritt wäre jetzt, beide Teile zu verbinden, oder? Hierbei müssen wir sicherstellen, dass sich das Dreieck für das Eselsohr **vor** dem »abgeschnittenen« Dreieck befindet:

Abbildung 4.26
Der zweite Farbverlauf für das Eselsohr für sich genommen; damit der Text besser erkennbar ist, wird er hier hellgrau statt weiß dargestellt.

```
background: #58a; /* Fallback */
background:
    linear-gradient(to left bottom,
        transparent 50%, rgba(0,0,0,.4) 0)
        no-repeat 100% 0 / 2em 2em,
    linear-gradient(-135deg, transparent 2em, #58a 0);
```

"The only way to get rid of a temptation is to yield to it."
—Oscar Wilde, The Picture of Dorian Gray

Abbildung 4.27
Die Kombination beider Farbverläufe führt noch nicht zum gewünschten Ergebnis.

Wie in **Abbildung 4.27** zu sehen ist, entspricht das Ergebnis noch nicht unseren Erwartungen. Warum passen die Größen nicht zueinander? Sie sind beide `2em` groß!

Der Grund ist, dass (wie im **Abschnitt »Innen abgerundete Ecken« auf Seite 40** besprochen) die Größenangabe von `2em` im zweiten Color Stop definiert wird. Dadurch wird sie **entlang der Gradientenlinie gemessen**, und die ist diagonal. Gleichzeitig bezieht sich die Angabe `2em`

in `background-size` auf die **Breite und Höhe des Hintergrundbilds**, das wiederum horizontal und vertikal gemessen wird.

Damit beide zueinanderpassen, müssen wir eine der folgenden Aktionen durchführen, je nachdem, welche der beiden Größenangaben beibehalten werden soll:

- Um `2em` für die Diagonale beizubehalten, können wir den Wert für `background-size` mit $\sqrt{2}$ **multiplizieren**.
- Um die horizontale und vertikale Größe von `2em` beizubehalten, können wir die Color-Stop-Position für den Farbverlauf, der die angeschnittene Ecke erzeugt, durch $\sqrt{2}$ **dividieren**.

Da `background-size` zweimal wiederholt wird und die meisten anderen CSS-Angaben nicht diagonal gemessen werden, ist die zweite Methode meistens die bessere. Die Position für den Color Stop wird zu $\frac{2}{\sqrt{2}} = \sqrt{2} \approx 1.414213562$, was wir auf `1.5em` runden können:

```
background: #58a; /* Fallback */
background:
    linear-gradient(to left bottom,
        transparent 50%, rgba(0,0,0,.4) 0)
        no-repeat 100% 0 / 2em 2em,
    linear-gradient(-135deg,
        transparent 1.5em, #58a 0);
```

Wie Sie in **Abbildung 4.28** sehen, ist das Ergebnis eine schöne, flexible abgeknickte Ecke.

▸ **PLAY!** play.csssecrets.io/**folded-corner**

"The only way to get rid of a temptation is to yield to it."
—Oscar Wilde, The Picture of Dorian Gray

Abbildung 4.28
Nach der Anpassung der Color-Stop-Position für den blauen Farbverlauf funktioniert unser Eselsohren-Effekt endlich richtig.

! Sorgen Sie dafür, dass **der Wert für den Innenabstand mindestens der Größe für die Ecke** entspricht. Ansonsten ragt der Text über die Ecke hinaus (weil sie einfach nur ein Hintergrund ist), was den Eselsohren-Effekt zunichtemacht.

Lösung für andere Winkel

Im wahren Leben haben Eselsohren nur selten einen Winkel von 45 Grad. Soll der Effekt etwas realistischer sein, können wir einen leicht unterschiedlichen Winkel, etwa `-150deg`, angeben, um eine Neigung von 30

"The only way to get rid of a temptation is to yield to it." —Oscar Wilde, The Picture of Dorian Gray

Abbildung 4.29
Wenn wir den Winkel ändern, funktioniert der Eselsohren-Effekt nicht mehr.

Grad zu erhalten. Ändern wir nur den Winkel für die angeschnittene Ecke, passt das Dreieck für den umgeknickten Teil der Seite nicht mehr dazu. Wie in **Abbildung 4.29** funktioniert der Effekt dadurch nicht mehr. Die Anpassung der Größenangaben ist jedoch nicht so einfach, wie es scheint. Die Größe des Dreiecks wird nicht durch den Winkel definiert, sondern durch seine Höhe und Breite. Wie können wir die nötigen Angaben finden? Zeit für ein bisschen Trigonometrie – Schluck!

Im Moment sieht der Code so aus:

```
background: #58a; /* Fallback */
background:
    linear-gradient(to left bottom,
        transparent 50%, rgba(0,0,0,.4) 0)
        no-repeat 100% 0 / 2em 2em,
    linear-gradient(-150deg,
        transparent 1.5em, #58a 0);
```

Ein *30-60-90-Dreieck* ist ein rechtwinkliges Dreieck, dessen andere Winkel 30 und 90 Grad betragen.

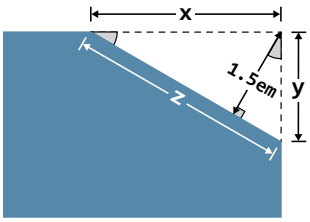
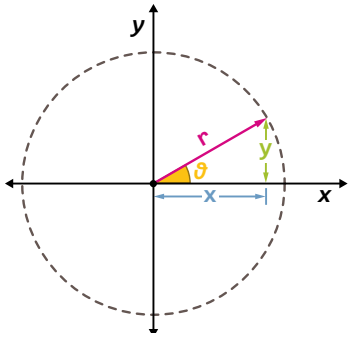


Abbildung 4.30
Eine vergrößerte Version der angeschnittenen Ecke (die grau markieren Winkel betragen 30 Grad).

Wie in **Abbildung 4.30** zu sehen ist, müssen wir eigentlich die Länge der Hypotenuse eines *30-60-90-Dreiecks* berechnen, wobei die Länge eines Schenkels bekannt ist. Wie der in **Abbildung 4.31** gezeigte trigonometrische Kreis verdeutlicht, können wir anhand von Sinus, Kosinus und dem Satz des Pythagoras die Länge zweier Schenkel berechnen, wenn wir den **Winkel und die Länge eines Schenkels eines rechtwinkligen Dreiecks** kennen. Aus der Mathematik (oder von einem Taschenrechner) wissen wir, dass $\cos 30° = \frac{\sqrt{3}}{2}$ und $\sin 30° = \frac{1}{2}$ sind. Durch den trigometrischen Kreis wissen wir außerdem, dass in unserem Fall $\sin 30° = \frac{1.5}{x}$ und $\cos 30° = \frac{1.5}{y}$ sind. Daraus folgt:

$$\frac{1}{2} = \frac{1.5}{x} \Rightarrow x = 2 \times 1.5 \Rightarrow x = 3$$

$$\frac{\sqrt{3}}{2} = \frac{1.5}{y} \Rightarrow y = \frac{2 \times 1.5}{\sqrt{3}} \Rightarrow y = \sqrt{3} \approx 1.732050808$$



$$x^2 + y^2 = r^2$$

$$\cos\vartheta = \frac{y}{r}$$

$$\sin\vartheta = \frac{x}{r}$$

Abbildung 4.31
Sinus und Kosinus helfen uns, die Schenkel rechtwinkliger Dreiecke anhand ihres Winkels und ihrer Hypotenuse zu berechnen.

Danach können wir *z* mit dem Satz des Pythagoras berechnen:

$$z = \sqrt{x^2 + y^2} = \sqrt{\sqrt{3}^2 + 3^2} = \sqrt{3 + 9} = \sqrt{12} = 2\sqrt{3}$$

Jetzt können wir die Größe des Dreiecks nach Bedarf anpassen:

```
background: #58a; /* Fallback */
background:
    linear-gradient(to left bottom,
        transparent 50%, rgba(0,0,0,.4) 0)
        no-repeat 100% 0 / 3em 1.73em,
    linear-gradient(-150deg,
        transparent 1.5em, #58a 0);
```

Im Moment sieht unsere Ecke aus wie in **Abbildung 4.32**. Wie Sie sehen, **passt das Dreieck jetzt zur abgeschnittenen Ecke**. Trotzdem sieht das Ergebnis **nicht realistischer** aus! Auch wenn die Fehlersuche nicht einfach ist, haben unsere Augen im Leben schon viele Eselsohren gesehen und wissen sofort, dass unser Ergebnis stark vom gewohnten Muster abweicht. Wenn Sie Ihrem Bewusstsein helfen wollen, den Grund für das falsche Aussehen zu verstehen, können Sie einfach ein **echtes Stück Papier** in etwa dem gleichen Winkel **falten**. Es gibt **absolut keine Möglichkeit**, das Papier so zu falten, dass es auch nur annähernd aussieht wie in **Abbildung 4.32**.

> "The only way to get rid of a temptation is to yield to it."
> —Oscar Wilde, The Picture of Dorian Gray

Abbildung 4.32
Obwohl wir den gewünschten Effekt erzielt haben, sieht er aktuell noch unrealistischer aus als vorher.

Um ein lebensechtes Eselsohr wie in **Abbildung 4.33** zu erzeugen, muss das erzeugte Dreieck **ein Stück gedreht** werden und dabei die gleiche Größe haben wie das von der Ecke »abgeschnittene« Dreieck. Da wir Hintergründe nicht drehen können, lagern wir den Effekt in ein Pseudoelement aus:

Abbildung 4.33
Eine analoge Version des Eselsohren-Effekts (Papiergestaltung von Leonie und Phoebe Verou).

```
.note {
    position: relative;
    background: #58a; /* Fallback */
    background:
        linear-gradient(-150deg,
            transparent 1.5em, #58a 0);
}
.note::before {
    content: '';
    position: absolute;
    top: 0; right: 0;
    background: linear-gradient(to left bottom,
        transparent 50%, rgba(0,0,0,.4) 0)
        100% 0 no-repeat;
    width: 3em;
    height: 1.73em;
}
```

Damit haben wir den gleichen Effekt wie in **Abbildung 4.32** erzeugt. Allerdings benutzen wir diesmal ein Pseudoelement. Der nächste Schritt wäre, die Ausrichtung des Dreiecks zu verändern, indem wir dessen **`width` und `height` tauschen**. Dadurch wird die **abgeschnittene Ecke** nicht verdeckt, sondern **gespiegelt**. Danach drehen wir es um 30 Grad ((90° – 30°) – 30°) gegen den Uhrzeigersinn, sodass seine **Hypotenuse parallel zur abgeschnittenen Ecke** platziert wird.

```
.note::before {
    content: '';
    position: absolute;
    top: 0; right: 0;
    background: linear-gradient(to left bottom,
        transparent 50%, rgba(0,0,0,.4) 0)
        100% 0 no-repeat;
    width: 1.73em;
    height: 3em;
    transform: rotate(-30deg);
}
```

Nach diesen Änderungen sieht unser Beispiel jetzt aus wie in **Abbildung 4.34**. Im Prinzip haben wir es fast geschafft. Jetzt müssen wir das Dreieck nur noch so bewegen, dass die Hypotenusen beider Dreiecke (das dunkle und das ausgeschnittene) zur Deckung kommen. Im Moment muss das Dreieck hierfür horizontal und vertikal verschoben werden. Dadurch ist es schwieriger, die nötigen Arbeitsschritte zu ermitteln. Wir können uns die Sache aber erleichtern, indem wir `transform-origin` den Wert `bottom right` geben, wodurch **die rechte untere Ecke des Dreiecks als Drehpunkt** verwendet wird und damit an der ursprünglichen Position bleibt:

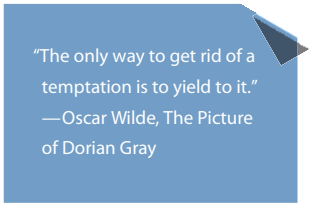


Abbildung 4.34
Langsam kommen wir unserem Ziel näher. Allerdings muss das Dreieck verschoben werden.

```
.note::before {
    /* [Weitere Stildefinitionen] */
    transform: rotate(-30deg);
    transform-origin: bottom right;
}
```

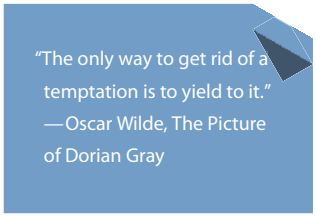


Abbildung 4.35
Die Verwendung von `transform-origin: bottom right;` erleichtert unsere Arbeit, und wir brauchen das Dreieck nur noch vertikal zu verschieben.

Wie Sie in **Abbildung 4.35** sehen können, müssen wir das Dreieck jetzt nur noch vertikal nach oben verschieben. Um die genaue Distanz zu ermitteln,

Stellen Sie sicher, dass der `translateY()`-Transform **vor der Rotation** durchgeführt wird! Ansonsten bewegt sich das Dreieck entlang seines 30-Grad-Winkels, weil **jede Transformation das gesamte Koordinatensystem** des Elements **verschiebt**, nicht nur das Element selbst.

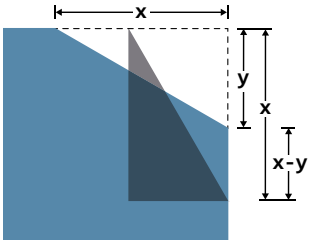


Abbildung 4.36
Es ist leichter als gedacht, herauszufinden, wie weit das Dreieck verschoben werden muss.

können Sie auch hier etwas Geometrie verwenden. Wie in **Abbildung 4.36** zu sehen ist, beträgt die benötigte vertikale Verschiebung für unser Dreieck $x - y = 3 - \sqrt{3} \approx 1.267949192$, was wir auf `1.3em` runden können:

```
.note::before {
    /* [Übrige Stildefinitionen] */
    transform: translateY(-1.3em) rotate(-30deg);
    transform-origin: bottom right;
}
```

Die Beispieldarstellung in **Abbildung 4.37** bestätigt, dass wir endlich den gewünschten Effekt erzielt haben. Uff, das war ziemlich *intensiv*! Und weil unser Dreieck jetzt mithilfe von Pseudoelementen erzeugt wird, können wir es **noch realistischer** gestalten, indem wir abgerundete Ecken, (tatsächliche) Farbverläufe und `box-shadow` verwenden! Der fertige Code sieht aus wie folgt:

"The only way to get rid of a temptation is to yield to it."
—Oscar Wilde, The Picture of Dorian Gray

Abbildung 4.37
Endlich sind unsere Dreiecke so ausgerichtet, dass sie sich berühren.

```
.note {
    position: relative;
    background: #58a; /* Fallback */
    background:
        linear-gradient(-150deg,
            transparent 1.5em, #58a 0);
    border-radius: .5em;
}
.note::before {
    content: '';
    position: absolute;
    top: 0; right: 0;
    background: linear-gradient(to left bottom,
        transparent 50%, rgba(0,0,0,.2) 0, rgba(0,0,0,.4)
        100% 0 no-repeat;
    width: 1.73em;
    height: 3em;
```

"The only way to get rid of a temptation is to yield to it."
—Oscar Wilde, The Picture of Dorian Gray

Abbildung 4.38
Mit ein paar weiteren Effekten können wir unser Eselsohr zum Leben erwecken.

```
    transform: translateY(-1.3em) rotate(-30deg);
    transform-origin: bottom right;
    border-bottom-left-radius: inherit;
     box-shadow: -.2em .2em .3em -.1em rgba(0,0,0,.15);
}
```

In **Abbildung 4.38** können Sie schließlich die Früchte Ihrer Arbeit bewundern.

▸ **PLAY!** play.csssecrets.io/**folded-corner-realistic**

Der Effekt sieht sehr hübsch aus, aber ist er auch DRY? Denken wir einmal über ein paar häufige Änderungen und Variationen nach:

- Um die **Dimensionen und andere Größenangaben** des Elements (Innenabstand etc.) zu ändern, muss **der Code an einer Stelle verändert** werden.
- Um die **Hintergrundfarbe** (ohne den Fallback) zu ändern, werden **zwei Anpassungen** benötigt.
- Um die **Größe** des Eselsohrs anzupassen, werden **vier Änderungen und einige nicht-triviale Berechnungen** gebraucht.
- Um den **Winkel** des Eselsohrs anzupassen, müssen **fünf Änderungen und einige noch weniger triviale Berechnungen** durchgeführt werden.
- Besonders die letzten beiden Punkte sind ziemlich schlecht. Vermutlich ist es Zeit für ein Präprozessor-Mixin:

SCSS

```
@mixin folded-corner($background, $size,
                     $angle: 30deg) {
position: relative;
background: $background; /* Fallback */
background:
    linear-gradient($angle - 180deg,
        transparent $size, $background 0);
border-radius: .5em;
```

```
$x: $size / sin($angle);
$y: $size / cos($angle);

&::before {
    content: '';
    position: absolute;
    top: 0; right: 0;
    background: linear-gradient(to left bottom,
        transparent 50%, rgba(0,0,0,.2) 0,
        rgba(0,0,0,.4)) 100% 0 no-repeat;
    width: $y; height: $x;
    transform: translateY($y - $x)
               rotate(2*$angle - 90deg);
    transform-origin: bottom right;
    border-bottom-left-radius: inherit;
    box-shadow: -.2em .2em .3em -.1em rgba(0,0,0,.2);
}
}
/* verwendet als ... */
.note {
    @include folded-corner(#58a, 2em, 40deg);
}
```

▸ **PLAY!** play.csssecrets.io/**folded-corner-mixin**

Beim Schreiben dieses Buchs gab es in **SCSS** noch keine native Unterstützung für trigonometrische Funktionen. Um diese nachzurüsten, können Sie unter anderem das **Compass-Framework** (*http://compass-style.org/*) verwenden. Anhand der Taylorreihe für diese Funktionen könnten Sie sie sogar selbst schreiben! **LESS** bringt die Funktionen dagegen bereits von Haus aus mit.

Verwandte Spezifikationen

- **CSS Backgrounds & Borders** (*http://w3.org/TR/css-backgrounds*)
- **CSS Image Values** (*http://w3.org/TR/css-images*)
- **CSS Transforms** (*http://w3.org/TR/css-transforms*)