Web UI Animation

Group 5

Rok Kogovšek, Alexei Kruglov, Fernando Pulido Ruiz, and Helmut Zöhrer

706.041 Information Architecture and Web Usability WS 2016
Graz University of Technology
A-8010 Graz, Austria

06 Dec 2016

Abstract

This survey tries to give insights into the uses of web animations and possible ways of creating them. Not only why they should be used on a website but also when to use which kind of the two main implementation methods (CSS and JS) will be covered. Even perfectly scalable animations (animated SVGs) will be a part of this survey. In addition to the theoretical background, numerous practical code examples will be included and discussed.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

Web animations are not only there to look pretty. Animated objects may carry meaning and we will see that there is much more thinking and planning necessary than expected at first sight.

Knowing this makes it difficult to find the right amount of animation without overdoing it. Some animation just for the purpose that something happens on the screen will not do any good to the user and might lead to distraction.

The basic way of implementing animations is with Cascading Style Sheets (CSS). It is widely supported and offers rather simple but effective ways of transforming objects in a meaningful way. Especially keyframes and easing effects are widely used for animating objects. The explicit implementation of hamburger menus, loading icons and navigation bars is described in detail in this survey.

Similar to CSS, SVG animations are possible as well. They have the feature of including basic graphical elements and have the advantage of being infinitely scalable which might come handy in the online use.

Another way of implementing animations is via JavaScript (JS). It allows more complex creations and gives the developer much more possibilities and freedom. It requires longer loading times and generally takes up more memory than pure CSS animations though. A combination of CSS and JS can be useful for a wide variety of animation effects. Especially the possibilities of animating text with JS and CSS are underlined in this survey.

Chapter 2

Animation

"Animation is defined as changing some property over time. On the other hand, motion is the act of moving or the process of being moved. . . . To put it more simply, all motion is animation, but not all animation is motion."[**head2016designing**]

2.1 Not Just Motion!

In animation we change an object's attribute(s) over time to achieve an objective. As stated by **head2016designing** animation is much more than adding movement to an object. Movement of an object can be described by a change of its 6 degrees of freedom, them being the coordinates in 3D space and the rotation around the 3D space axises. However an object has usually many more changeable attributes or properties. One could animate color changing and fading, invisibility through transparency, growth with scaling, focus with blurring, etc.

2.2 Why Animation in Web UI?

While writing this survey, with so much animation theory, a hilarious idea was just coming up to mind. Have you ever stopped to think about why some people just seem to be more funny than others? And of course, this happens everyday. Just think about a good joke told by someone with no real humour. It is meaningless. Now, think about the same joke being told by a profesional comedian on TV telling jokes over the weekend. Way more funny, is it not? But the point is, why and how is this be possible? It has lots to do with how they speak and how they move! As well of other minor factors. That, I am afraid, is the same as with animation. To have a better understanding of what factors are involved and how they can improve the user experience in UIs, we look deeper into it in this section with **head2016designing** as the reference point.

Animation, be it in a cartoon, webpage, app or anywhere else, adds a new level of communication which can be hard to explain , but easy to feel. It creates a special connection between what the message is intended to be, and the receiver. It allows an additional cannel of invisible information to exist, and makes the viewer feel part of the process, and gets his attention grabbed in a practical way.

We always need to understand, where we are and where are we going to. Animation can helps us with this and even can improve the use of resources. Just think about when trying to navigate through a small screen. Having some animated window coming out when you put your mouse on, does not only save space (which is already a good enough reason to implement it), but makes the entire process of navigating easier and simpler. A user does not really want to think while navigating, and this can be fulfilled with animation. They can be guided, orientated or helped in going through a process just with animation, which will of course improve their user experience and make them feel more sure and comfortable about what they are doing, without thinking they are in a real puzzle and need to make big use of their brains.

Putting it all together, we can effectively ensure that a well planned animation will improve decision making, as it will guide the user through the entire process, allowing him to be more aware about the real process or

task he or she is willing to achieve, and not about the real animation. Of course, this can be enforced by the fact that the user will feel listened and will probably trust more the site because of the animation being interactive, giving in some way feedback, making the user feel comfortable when its input is being taken into consideration.

Nowadays, it´s very common to access many sites through different devices. Just think about when trying to know any train timetable. You probably have the app in your Smartphone, but depending on where are you at the moment, you might just google it up and search it through their webpage. Having animation allows sites to connect context and media. A user will just find himself guided throught the process no matter where he is or what device is he using, because of being under some potential animation process.

But, how do these animations grab people´s atention? There are over twelve basic principles, written by Disney, but two of them are like the foundation of animation, and the rest basically build up from them. Timing and spacing. Timing can be understood as the time length of an action to happen, let's say the duration of it, while spacing could be explained as the speed changes during one of these actions in an animation. **head2016designing** (Chapter 2, page 18) describes their impact with: *"Timing and spacing convey the mood, emotion, and reaction of an object."*

2.3 Aim For Invisible Animation

In animation, it is very common during the designing process to end up messing things up. As a general rule, as a designer, your one good way to show love to the user is actually not making his or her life harder accomplishing the task at hand. Certainly designers do not hold such ill intentions, but as said before, it is quite easy and happens very often just to have users unnecesarily waiting for an animation to finish or having a hard time finishing a task due to animation. It could be said that the perfect animation is the one which is not noticed. That´s not 100% accurate, but a good idea in general.

*"Good interface animations need to be flexible and always feel responsive to a user's input even if the animation is currently animating."***head2016designing** (Chapter 3, page 48)

Users need feedback to feel themselves listened and understood. Animation can be understood as a bridge which takes the user to a higher quality experience. If an animation doesn't respect having the user happily informed with feedback, its experience quality will of course drop and so will the trust towards the UI. A user will never feel comfortable in a system which does not take into consideration its input.

It is also important, that an animation should never be a show-off work. It should be always remembered, that in general, the user interacting with an animation, is probably willing to do or finish something. A UI user is not just laying there waiting for an animation to happen and look at it itself and enjoy it. That could be the aim of animators years ago where they were really telling stories and people were fascinted about it. But times changed, and now the timing requirements are no longer the same as in those days. Study and carefully think about the suitable timing for each animation, because a short amount of milliseconds can be the limit between failing or succeeding. Good timing is more an art than a science [**head2016designing**].

2.4 Development of animation

There are lots of guidelines to help developers through the development of an animation, but what about some guidelines for the actual development itself? When coming up to a project, the animation issues are often discussed at the end of it. It is quite normal to just understand animation in a project as extra decoration, as something which is not mandatory, which will just make the whole running project look a little nicer. This should not be the way of proceeding, but it is the reality of many projects. Animation should be part of the design, due to the potential for a better user experience, like described in section **??**. Just bare in mind that when the project is finished, it is usually too late to replan the design involving animation at its full potential, and this leads to many projects to just leave beside some good animation ideas. Trying to makes things easier for the user (that is what animation is all about) at the end of a project, when everything is quited fixed, doesn´t sound

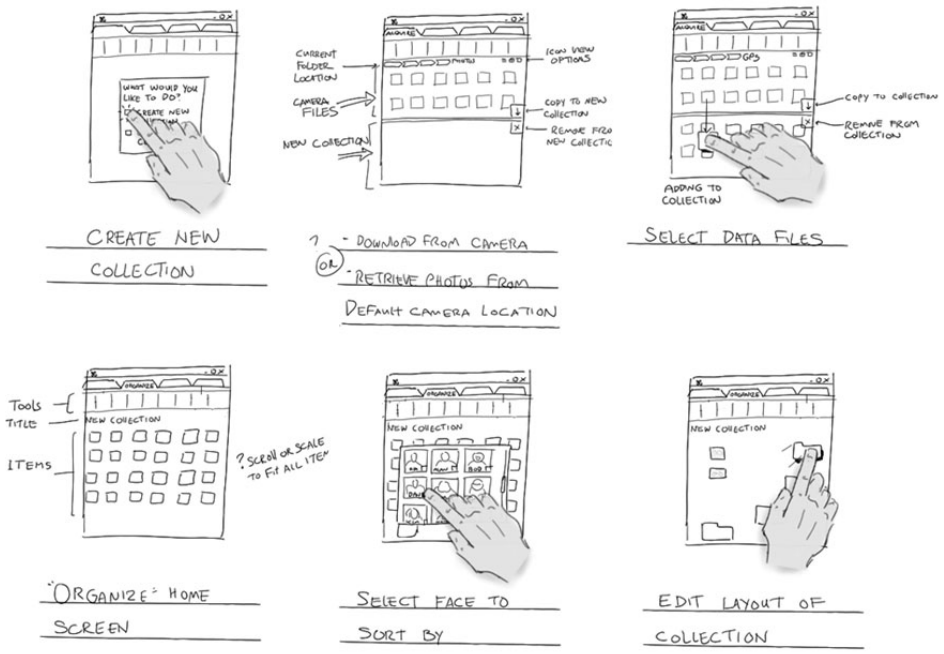

Figure 2.1: Example of storyboard sketching for drag and drop animation [**microsoftStoryboard**]. [Used with permission from Microsoft - Microsoft Copyrighted Content Guidelines]

like a good plan. Always start planning animation schemes in early stages! It is always better to have a pause after each step of the project design to talk about animation, than to rather avoid it at all because its already too late! Remember that animation can replace a fixed design element which might make the user have a bigger quality of the experience, or less brain demanding. And of course, as taken as a part of the project design, style guides and documentation about the used animation should be reported. And no need to say, understanding the basics of animation principles, and how they can influence the user, and bearing in mind the accessibility issue . . . as a conclusion, just make sure you don´t forget any of the points mentioned in the earlier sections of this survey. When coming up to the how to plan it, probably the best way in early stages is storyboard sketching. This is a great, simple and quick way of more or less understanding what we are doing (or planning to do). When moving further, however, building an animation prototype inside the design prototype can be very suitable before moving on to the next stage, as it can allow designers to really test and see if what they´re doing its actually as good and helpful as they think.

Chapter 3

Cascading Style Sheets (CSS)

Knowing the usefulness of animation in web UI and the correct way of animation planing are just the fundemantals for our conceptual plans. Those still need to be implemented to get the end product and here we usually hit a wall build from the various tools, that say they can all solve our problems. Even well established people in the field have stories as such to tell. Val Head actually started with animation due to an interesting Flash workshop. Flash was at that time the de facto king in its era, however as we know, that era is already dead. Nowdays we can acomplish all we could with Flash and more with just the core parts of the web, namely HTML, CSS and JS [**head2016designing**].

3.1 Do Everything You Can With CSS

With Responsive web design (RWD) in our websites and animation being part of the design, see section 2.4, it should be natural to use the guidelines of RWD also in animation planning. **IAWEB** teaches us that one of the RWD guidelines is also Progressive enhancement, which is best described with words of the conceptual authors **champeon2003inclusive** *"Leave no one behind. . . . accessibility is for everyone, not just the disabled"*. With CSS nowdays being a core part of the web and at the same time being the lowest web component that enables animation with RWD guidelines[1], one should always implement with CSS and HTML alone as much of the desired animation as possible. One has only to make sure the browser support for the animated attribute.

Other supporting arguments for use of CSS as the starting point for web UI animation beside responsiveness can be summarized with the the so called "Simple CSS Truths", a list of truths by **palermoCSS** enhanced with the teachings of **IAWEB**

CSS allows for separation of concerns - With CSS the form is separated from the page's HTML structure and content. Makes it easier to read, maintain and crawl the code.

CSS has a captive audience - Support for CSS development is huge. At the same time more and more libraries, tools and frameworks focus on improving and simplifing CSS development.

CSS is fast - External CSS speeds up HTML downlaod and loading compared to HTMLs with duplicated inline styles. Compared to JavaScript it also processes transitions and animations faster.

CSS is fault-tolarent - Browser-unknown enhancements are simply ignored by the browser, while the remainder is still used and displayed.

CSS is everywhere - Modern browsers embrace CSS and feature support by each can be easily found online.

[1] Of course one can just use an animated image, e.q. a GIF with an image sequence, and just append it with HTML into the design. However, this image will become a static component of the design and will not follow RWD guidelines.

3.2 CSS Animation Declaration

As stated in section 2.1, animation is about changing an element's attribute(s) over time. In CSS we can redefine it as a switch between CSS styles for a HTML element that happens gradually over time. It is stated that CSS animation should be done with *Animation* property(ies) and *Keyframe* rule(s). This may be the most efficient CSS way to accomplish animation, but CSS animation can also be achieved with *Transition* property(ies) and *selector* pattern(s)[**w3schoolAnime**; **w3schoolTrans**].

3.2.1 Animation Property and Keframe Rule

The prefered declaration for animation in CSS is done by setting animation properties to the element, that will change through time. This properties just define to which animation steps or keyframes the element is linked to and how the in-between-keyframe states are interpolated through time. **w3schoolAnime** explains the different properties as follows:

animation-name: Here the *@keyframe* rule name is used to link the rule to the element. Since CSS animation is a set of animation properties and keyframe rules, it is a neccasery property.

animation-duration: Like-wise to the name, duration of the animation is also a neccasery property, since it defaults to 0. Animation is defined as a change over a time, so with the time duration being zero, of course there is no animation. Actual working values can be either seconds (#s) or miliseconds (#ms), that define one cycle of the animation.

animation-timing-function: Defines the progression curve over time, that defines how the animation is interpolated. The valid values are linear, ease, ease-in, ease-out, ease-in-out, cubic-bezier(x1,y1,x2,y1) and steps(stepSize, start, end). Acording to **head2016designing** most animators have a hard time imagining the effect of functions that start with ease*. While linear and steps are simple to understand and enough for simple animations, the cubic-bezier function should be used, since it can define any desirable curve. The way the cubic-bezier function works, can be observed in figure 3.1.

animation-delay: Defines how many seconds or miliseconds should the animation start be delayed. The default value is 0. It is useful, when we want multiple elements be animated one after another.

animation-iteration-count: We can define with a number, how many loops of the cycle should be done or set it to infinite cycles for a non-stop replay. The default value is 1.

animation-direction: With this property we describe in which way the animation is interpolated and what happens after the 1st cycle of animation. By setting it to the default normal, set it to reset the element into the state before animation and start it again. Reverse always starts from the end keyframe and goes towaqrds the starting keyframe, while alternate will alternate between normal and reverse by the oddness of the cycle number. Another option is the self-explanatory alternate-reverse.

animation-fill-mode: Defines what happens to an unplayed element, be it after finishing the animation or during a delay wait. By the default none the element has the CSS style outside the keyframe rules. Forwards will use the style of the last keyframe, while backwards will use the style of the first keyframe. A special option is both, that uses the styles of both the start and end keyframe.

animation-play-state: It is mostly a property used for testing and with control triggers. As the property name states it is either paused or running.

Same as with other CSS properties animation properties can be combined into a single property, simply called animation and where the property values follow as stated:

animation: name duration timing-function delay iteration-count direction fill-mode play-state;

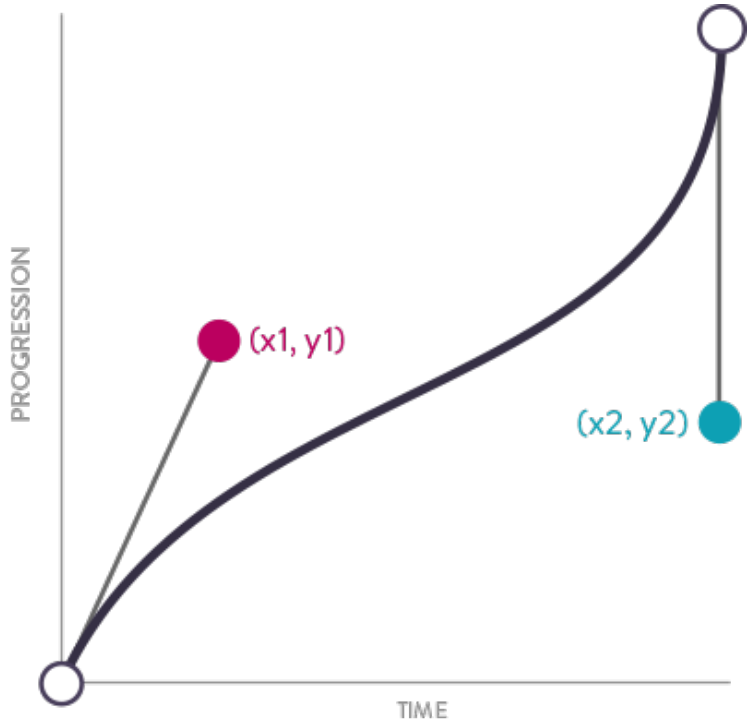

Figure 3.1: With coordinates for the location of the start and end weight we can define the desired function curve for progression through time by how much each half of the function should be deformed away from a linear function [**head2016designing**]. [The image is from the book "Designing interface animation" by **head2016designing** used under CC BY 2.0 / It is accessible at *https://www.flickr.com/photos/rosenfeldmedia/albums/72157671107313626*.]

While the animation properties just link the element to the desired changes, the actual CSS changes are defined in the @keyframes rule. In the rule we can set the keyframes, that are the finishing times, when a change should end. The keyframe times are simply described with percantage from 0% as the start keframe and 100% as the end keyframe. To this times we simply set, which styles should be used at the appointed time. The interpolation function set in with animation property will during proccessing interpolate the undefined percantages. For a better understanding see listing 3.2.1.

```
1  div{
2    animation: desiredName 4s linear 0s infinite alternate;
3  }
4
5  @keyframes desiredName {
6    0%   {background-color:red;opacity: 1;transform: scale(1);}
7    25%  {background-color:yellow;transform: scale(0.8);}
8    50%  {background-color:blue;transform: scale(1);}
9    75%  {background-color:green;transform: scale(1.5);}
10   100% {background-color:red;opacity: 0.2;transform: scale(2);}
11 }
```

Listing 3.1: Simple example of non-motion animation with animation property and keyframes rule. A working example can be found between the attached code examples under nonmotionAnimationCSS.html.

3.2.2 Transition Property and Selector Pattern

The transition property can be used the same way as th animation property for defining the linking and interpolation details. There are even similarities int hte properties:

transition-property: With this property we can link the the transition to a change in CSS, similar as with animation-name. However here we specify which of the CSS properties of the element should be interpolated during the change. It can be one or more properties seperated with a comma. We also have the default option none and convenient all, that sets all propertis to interpolate.

transition-duration: Same as animation-duration property.

transition-timing-function: Same as animation-timinig-function property.

transition-delay: Same as animation-delay property.

The unified property is also defined by the same order, with just less options:

transition: property duration timing-function delay;

One can imagine the transition being a simplified version of the animation property, that can only have two keyframe, namely the start and end of the aniamtion or transtion. The start keyframe style is defined in the CSS style of the unchanged element, while the end frame is defined in a declaration where the specified transition property is changed. The easiest way to control such animations is with event seperators, such as :hover. This way we can define an animation to happen after hovering over the element. Since we can define just two keyframes and there ano repetitions, we are quite limited with the possible animations, which is also the reason for the animation property and keyframes rule to be used most of the time for animations. The limitations can be easily observed, when comparing the codes and results for 3.2.1 and 3.2.2.

```
1  div{
2    background-color: red;
3    opacity: 1;
4    transition: all 4s;
5  }
6  div:hover{
7    background-color:green;
8    opacity: 0.2;
9    transform: scale(2);
10 }
```

Listing 3.2: Simple example of non-motion animation with transition property and hover selector. A working example can be found between the attached code examples under nonmotionTransitionCSS.html.

3.2.3 Powerful Effect from Single Property

While any CSS property can be defined for the animated change, there are some that can bring much more effect than others. Such would be the transform property, that can define translation, trotation and scalation in 2D and 3D space, plus also having options for skewing and puting into perspective the element. What is even more, is that all can be achieved at the same time. Just with combining it with the transform-origin property any kind of motion animation can be achieved.

3.3 CSS Examples

While the previous sections tell us why animation can be useful, why CSS should be used and how to declare animation in CSS, we have yet to see CSS examples that show the difficulty to implement animation that brings benifits to the UI.

3.3.1 Navigation Animation

As stated in section 2.2, animation can help with navigation through the page. In the code examples we show two typical cases of such usage, namely the famous hamburger icon and menu position animation.

Hamburger Icon

The name comes from the icon being a set of three horizontal parallel lines sticking close together, the same way as the hamburger patty and meet sticks together. The icon is a metaphor for the menu to be tightly packed together, when we hide it "within" the icon. From the metaphor we also see the usefulness of such a concept, being the saving of space with hiding the menu and by showing the menu showing also the navigation path we are taking.

Implementations use various kinds of animations to make the use of the icon subtile and easy to use. Usually it is doen by rotating the lines in the icon into an X for showing the way of hiding or closing the menu and a sliding animation of the menu apearring from the border to show the navigation flow. An example of implementaations can be also observed in the code examples hamburgerVariationCSS.html and hamburgerMenuCSS.html, which are based on the code snippets provided by **hamburgerMenu**; **hamburgerVar**

By the information provided by **vtldesign** it is believed that the origins of the concept go back to 1981 and the famous company Xerox. However the explosion of usage came into being in the last decade, when designers were face with the problem of small screens on mobile devices.

```
HTML structure:
<!-- ICON -->
<div id="menu-button" role="button">
  <div class="hamburger"><div class="inner"></div></div>
</div>
<!-- HIDDEN MENU -->
<nav><ul>
    <li><a href="">Home Page</a></li>
    <li><a href="">Other stuff</a></li>
</ul></nav>
Key CSS:
  /*Creates the hamburger lines*/
  .hamburger::before, .hamburger::after, .hamburger .inner {
    transition: all 750ms ease-in-out; ...
  }
  /*Upper and bottom lines rotate into an X*/
  .paperNav-container:hover .hamburger::before {
    transform: translate3d(-4px, 1px, 0) rotateZ(-45deg);
  }
  .paperNav-container:hover .hamburger::after {
    transform: translate3d(-4px, -1px, 0) rotateZ(45deg);
  }
  /*Middle line rotates into a point*/
  .paperNav-container:hover .hamburger .inner {
    transform: rotateY(-90deg);
  }
```

Listing 3.3: Extracted key HTML structure and CSS code for rotating the icon into an X. For details check code example hamburgerMenuCSS.html.

Figure 3.2: One the left we have a screenshot of code example hamburgerVariationCSS.html, which shows some of the different ways how hamburger icons are animated. On the right side we have hamburgerMenuCSS.html, that demonstrates the showing and hiding of the menu by hovering over the hamburger icon. [Screenshot taken by the authors of this survey. The code behind the pages is by using the **hamburgerMenu**; **hamburgerVar** online snippets as a base.]

Current menu position indicator

A common problem with big menus is to keep track, where we currently are. The usual solutions are making a CSS change on the currently hovered or selected element. This can barely be said to be an animation, since the transition is instant. But such solutions can be even further improved with small animations, that are just extended previous solutions with the transiton having a duration. With adding duration to the animation, the user does not need to find where the change happened but just by observing it becomes a natural knowledge to the user, which also reduces the memory load and adds continuity to the navigation process. Typical solutions are combinations of opacity changes and translations of lines or even the menu words themselves. Such solutions can be observed in the code example in navigation.html, which is based on the code snippets by **menuIndex**

```
HTML structure:
<div class="container [color] [effectClass]">
  <a alt="HOME">HOME</a>
  <a alt="ABOUT">ABOUT</a>
  <a alt="CONTACT">CONTACT</a>
</div>
Key CSS:
  /* Line that transits slowly under the text*/
  div.[class] a:before, div.borderYtoX a:after
  {
    height: 100%; /*vertical*/
    width: 2px;   /*  line  */
    transition: all 0.3s; ...
  }
  div.topBotomBordersOut a:after
  {
    bottom: 0px;
    transform: translateY(-10px);
  }

  /*Text jumps out*/
  div.highlightTextOut a:before,
  div.highlightTextIn a:before
  {
      content: attr(alt);
      transition: all 0.3s;
      transform: scale(0.8);
      opacity: 0; ...
  }
  div.highlightTextOut a:hover:before,
```

```
31   div.highlightTextIn a:hover:before
32   {
33       transform: scale(1);
34       opacity: 1;
35   }
```

Listing 3.4: Extracted key HTML structure and CSS code for presenting a few animations for menu position. For details check code example navigation.html.

3.3.2 Loading Animation

Another important use of animation is user feedback, see 2.2. The most common case being loading animations, that give the user a confirmation that the process is running, to not worry him of possible invisible crashes. The variety of animations is big but most could be specified as a continues rotation or a continues movement, usually in the horizontal direction. It is important to note, that all provided examples are pure HTML and CSS, without any JavaScript or SVGs. This also shows how powerful CSS can be.

Rotating Icon

A quick look at listing 3.3.2 shows that the needed structure is just a div that has either a curve border created or it has a background color added. The circle animation is done with rotation in 2D space, while the square animation is done with rotation in 3D space. Working code examples are present in circleRotate2DCSS.html and squareRotated3DCSS.html, which are built from snippets done by **cirleLoader**; **otherLoaders**

```
1  HTML structure:
2    <div class="loaderClass"></div>
3  Key CSS:
4  .circleLoader {
5    border: 16px solid #f3f3f3; /* Light grey */
6    border-top: 16px solid #3498db; /* Blue */
7    border-radius: 50%; /*the border is turned into the circle*/
8    width: 120px;
9    height: 120px;
10   animation: spin 2s linear infinite;
11 }
12 @keyframes spin {
13   0% { transform: rotate(0deg); }
14   100% { transform: rotate(360deg); }
15 }
16
17 .squareloader {
18   width: 60px;
19   height: 60px;
20   margin: 60px;
21   animation: rotate 1.4s infinite ease-in-out;
22 }
23 @keyframes rotate {
24   0% { transform: perspective(120px) rotateX(0deg) rotateY(0deg);}
25   50% { transform: perspective(120px) rotateX(-180deg) rotateY(0deg);
26   }
27   100% { transform: perspective(120px) rotateX(-180deg) rotateY(-180deg);}
28 }
```

Listing 3.5: The actual needed code for both rotation loading animations. Working examples can be seen in circleRotate2DCSS.html and squareRotated3DCSS.html.

Figure 3.3: A screenshot of rotating loaders, with the left one being a circle rotated in 2D and the right one a square rotated in 3D. The example code is located in circleRotate2DCSS.html and squareRotated3DCSS.html. [Screenshot taken by the authors of this survey. The code behind the pages is by using the **cirleLoader**; **otherLoaders** online snippets as a base.]

Horizontal movement

The most common continues movement loader are horizontal. Therefore we provided 3 different types of movement in horizontal direction, namely moving dots, a progressbar and pulz-like wave motion. The working examples can be found in movingDotsCSS.html, progressbarCSS.html and pulzLoadCSS.html, which we built using the **otherLoaders** online snippets as a base.

```
HTML structure:
<div class="loaderClass"><div></div><div></div><div></div><div></div></div>
Key CSS:
.dotsLoader div {
  background-color: black;
  border-radius: 50%;
  animation: dots-move 4s infinite cubic-bezier(.2,.64,.81,.23); ...
}
.dotsLoader div:nth-child(2) {
  animation-delay: 150ms; /*Change delay per child*/
}
@keyframes dots-move {
  0% {left: 0%;}
  75% {left:100%;}
  100% {left:100%;}
}

.progressbarloader:before{
  display: block;
  position: absolute;
  left: -200px;
  width: 200px;
  height: 4px;
  background-color: #2980b9;
  animation: bar-loading 2s linear infinite;
```

```
}
@keyframes bar-loading {
  from {left: -200px; width: 30%;}
  50% {width: 30%;}
  80% { left: 50%;}
  95% {left: 120%;}
  to {left: 100%;}
}

.pulzLoader div {
  animation: pulz-loading 1s ease-in-out infinite; ...
}
.pulzLoader div:nth-child(1) {
  background-color: #3498db;  /*Change color per child*/
  animation-delay: 0;     /*Change delay per child*/
}
@keyframes pulz-loading {
  0% { transform: scale(1);}
  20% { transform: scale(1, 2.2);}
  40% { transform: scale(1);}
}
```

Listing 3.6: Extracted key HTML structure and CSS code for dots, progressbar and pulz loader. For details check code example movingDotsCSS.html, progressbarCSS.html and pulzLoaderCSS.html.

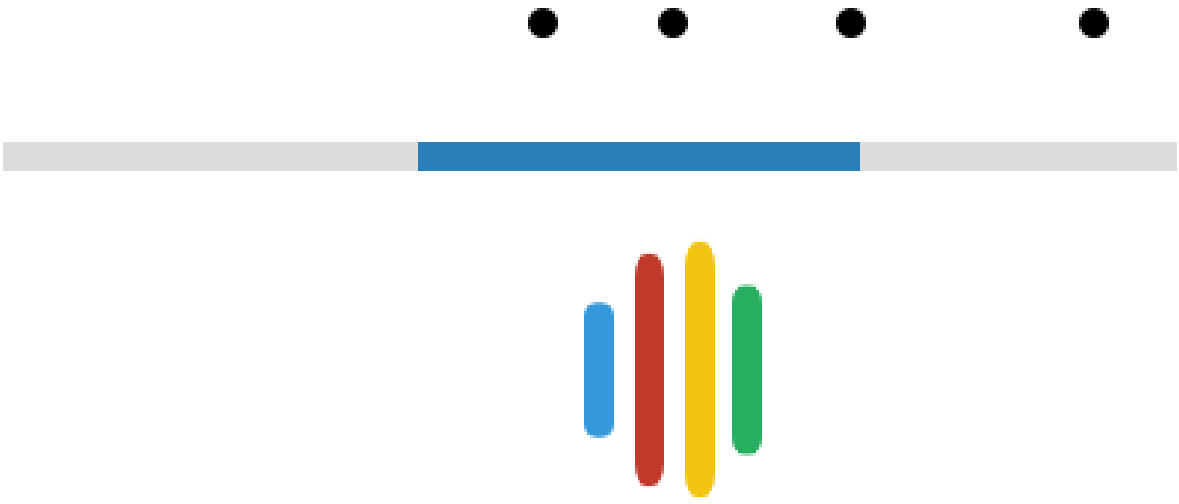

Figure 3.4: A screenshot of continues movement loaders in horizontal direction. On top we have animated dots that slow down in the middle of the animation. Second loader is the common progresbar, that is done by changing the left and width property. At the bottom we have an example of a pulz loader, that moves in wave form horizontaly and scales the bar when the "wave" comes. The example code is located in movingDotsCSS.html, progressbarCSS.html and pulzLoadCSS.html. [Screenshot taken by the authors of this survey. The code behind the pages is by using the **otherLoaders** online snippets as a base.]

3.4 Scalable Vector Graphics (SVG)

SVG (Scalable Vector Graphics) is an XML (Extensible Markup Language) based specification for describing two-dimensional graphics in a vector form. It is specified by W3C (World Wide Web Consortium). Initial release was made in 2001, and the latest release (1.1) was in 2011. Currently it is used in all latest major web browsers, such as Google Chrome, Mozilla Firefox and Internet Explorer, and in some vector graphics software editors, as Inkscape and Adobe Illustrator. In contrast to raster images, vector images are resolution independent, as the format itself is lossless. As mentioned before, the graphic is described in an XML textual file, which can be compressed with gzip. As a result of compression, web pages, where SVG is used load faster than those which use standard JPEG or PNG graphics solutions. Because graphics can be directly embedded into HTML, no extra HTTP requests are necessary. Above approaches lead to saving of the bandwidth, which is extremely important on the pages with higher loads. SVG has a support for different graphics effects and animations, which can be further enhanced with CSS and JavaScript. In case SVG graphic is made in a very complex node structure, browser can have problems rendering the image. In this case other technologies can be used. SVG integrates with other W3C standards such as the DOM and XSL[**w3schoolSVG**].

3.4.1 SVG basic graphical elements

As mentioned before SVG graphics can be embedded directly into HTML pages

```
1 <html>
2 <body>
3
4 <svg width="100" height="100">
5   <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow"
      />
6 </svg>
7
8 </body>
9 </html>
```

Listing 3.7: Simple example code one circle embedded directyl into HTML.

Figure 3.5: This example shows how simple to add one element (circle) embedded directyl into HTML. [Screenshot taken by the authors of this survey. The code behind the pages is by using the **CircleHtml** online snippets as a base.]

SVG consists of different basic graphical elements, from which more complex elements can be derived. Basic elements included in SVG are:

<rect> -Element is used to draw a rectangle and variations of the shape of a rectangle.

<circle> -Element is used to create a simple circle with color options.

<line> -Element is used to create a line.

<polygon> -Element is used to create a graphic that contains at least three sides.

<polyline> -Element is used to provide any shape consisting of straight lines.

<path> -Element is used to define a path. Participates in paths drawing strongly to use to encourage the SVG editor to create complex graphics.

With these simple, basic of elements, you can build a very complex shapes. With heli additional graphics functions, you can create a moving figures, but if these figures create just only SVG, the program code will be very large and difficult to read.

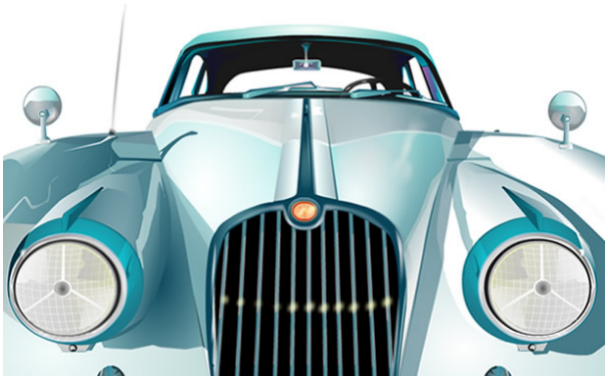

Figure 3.6: Example to create one complex picture with simple SVG elements, with help geomety and color options. [Screenshot taken by the authors of this survey. The code behind the pages is by using the **CarSVGl** online snippets as a base.]

3.4.2 SVG animation

SVG element is a specific DOM element, which includes himself syntax standard HTML-element. SVG elements have unique tags, attributes and behaviors that enable them to determine any form that offer the opportunity to essentially produce directly to the DOM, images, and thereby benefit from the JavaScript and CSS-based manipulation. There are three main advantages to create graphics in SVG, than an image (PNG, JPEG, etc.): First, compresses incredibly well, certain terms in the SVG file format is smaller than their PNG / JPEG equivalents. Secondly, SVG graphics to scale to any resolution without loss of clarity; they look sharp on all desktop and mobile screens. Third, you can animate the individual components of the SVG graphics performance (with help JavaScript and CSS).SVG elements take some of the standard CSS properties, but not all. In addition, SVG takes a certain set of "presentation" attributes, such as fill, x and y, which also serve to determine both the SVG is visually observed. There is no functional difference between the SVG specification of style using CSS, or as an attribute - SVG specification only divide property under two. In following examples shows how SVG work with CSS together:

Define the imeage as vectors in SVG

Tranform the vectors with CSS or SVG

```
1 <animateTransform attributeName="transform"
2     attributeType="XML"
3     type="translate"
4     values="0 50;0 -50;"
5     dur="2s"
6     repeatCount="indefinite"/>
```

Listing 3.8: Define the attribute for animate transorm.

Create SVG stucture into HTML

```
<svg class="svg-icon" viewBox="0 0 20 20">
    <path id="svg-doc" d="M15.475,6.692l-4.084-4.083C11.32,
    2.538,11.223,2.5,11.125,2.5h-6c-0.413,0-0.75,0.337-0.75,
    0.75v13.5c0,0.412,0.337,0.75,0.75,0.75h9.75c0.412,
    0,0.75-0.338,0.75-0.75V6.94C15.609,6.839,15.554,6.771,
    15.475,6.692 M11.5,3.779l2.843,2.846H11.5V3.779z
    M14.875,16.75h-9.75V3.25h5.625V7c0,0.206,0.168,0.375,
    0.375,0.375h3.75V16.75z"
    transform="scale(0.25) translate(27.5)"></path>

    <path id="svg-bottom" fill="none" d="M16.471,
    5.962c-0.365-0.066-0.709,0.176-0.774,0.538l-1.843,
    10.217H6.096L4.255,6.5c-0.066-0.362-0.42-0.603-0.775-0.538
    C3.117,6.027,2.876,6.375,2.942,6.737l1.94,10.765c0.058,
    0.318,0.334,0.549,0.657,0.549h8.872c0.323,0,0.6-0.23,
    0.656-0.549l1.941-10.765C17.074,6.375,16.833,6.027,16.471,
    5.962z"
    transform="scale(0.75) translate(4,20)"></path>

    <path id="svg-deck" fill="none" d="M16.594,3.804H3.406
    c-0.369,0-0.667,0.298-0.667,0.667s0.299,0.667,0.667,
    0.667h13.188c0.369,0,0.667-0.298,0.667-0.667S16.963,
    3.804,16.594,3.804zM9.25,3.284h1.501c0.368,0,0.667-0.298,
    0.667-0.667c0-0.369-0.299-0.667-0.667-0.667H9.25c-0.369,
    0-0.667,0.298-0.667,0.667C8.583,2.985,8.882,3.284,9.25,3.284z"
    transform="scale(0.75) translate(4,20)"></path>
  </svg>
```

Listing 3.9: Define svg class with three simple *<path>* SVG elements with transform function .

Define keyframe in CSS

```
#svg-doc{
  fill:red;
  animation: doc-delete 4s linear infinite;
}

#svg-deck{
  transform-origin: 100% 50%;
  animation: deck-rot 4s linear infinite;
}

@keyframes doc-delete {
  0% { transform: translate(75%) scale(0.25); }
  15% { transform: translate(75%) scale(0.25); }
  65% { transform: translate(85%,150%) scale(0.1) rotate(-180deg);}
  100% { transform: translate(85%,150%) scale(0.1); opacity: 0;}
}

@keyframes deck-rot {
  0% { transform: translate(-9%, 440%) rotate(0deg) scale(0.75); }
  25% { transform: translate(-9%, 440%) rotate(90deg) scale(0.75);}
  40% { transform: translate(-9%, 440%) rotate(90deg) scale(0.75);}
```

```
22    65% { transform: translate(-9%, 440%) rotate(0deg) scale(0.75);}
23    100% { transform: translate(-9%, 440%) rotate(0deg) scale(0.75);}
```

Listing 3.10: Create CSS with *@keyframes* for picture visualization .

Figure 3.7: Example to create one complex moving animation in SVG with help of CSS. [Screenshot taken by the authors of this survey. The code behind the pages is by using the **beerSVG** online snippets as a base.]

Figure 3.8: Example to create animation with help prepared simple icons. [Screenshot taken by the authors of this survey. The code behind the pages is by using the **trashSVG** online snippets as a base.]

Chapter 4

JavaScript (JS)

As an addition to plain CSS, animations can also be brought to life via JavaScript (JS) or a combination of CSS and JS. When to use which kind of implementation method is dependent on the possibilities in terms of control and effect one wants to achieve.

4.1 When to Use JS instead of CSS

As a rule of thumb, JS should not be used if the same effect could be achieved with plain CSS. This is due to performance and resource management reasons. Just when CSS is stretched to its limits, JS should be used. There is a rough distinction whether to use CSS or JS for particular kinds of tasks:

- Use CSS animations for simple transitions, like changing the state of a UI element.
- Use JS animations to get advanced effects like bouncing, stop, pause, rewind, or slow down (there is more control over animations).
- When choosing to animate with JS, considering using the Web Animations API or a modern framework (comfortable to work with) may be desirable.

Apart from this clear distinction, using both CSS and JS works also well. One could perform animations with CSS and control states with JS. [**googleDev**]

4.2 Examples of Useful Animations with JS

The following example taken from **transformsJS** is capable of breaking the CSS-habit of isolating transformations such as scaling, rotations and transposing. To be more precise, this example allows independent animations at a time with the help of partially overlapping start and ending times as well as different easing effects. This example would not be implementable with pure CSS.

```
1  //pulsate the box using scaleX and scaleY
2  TweenMax.to($box, 1.2, {scaleX:0.8, scaleY:0.8, force3D:true, yoyo:true, repeat
       :-1, ease:Power1.easeInOut});
3
4  $("#rotation").click(function() {
5  rotation += 360;
6  TweenLite.to($box, 2, {rotation:rotation, ease:Elastic.easeOut});
7  });
8
9  $("#rotationX").click(function() {
10 rotationX += 360;
```

```
11 TweenLite.to($box, 2, {rotationX:rotationX, ease:Power2.easeOut});
12 });
13
14 $("#rotationY").click(function() {
15 rotationY += 360;
16 TweenLite.to($box, 2, {rotationY:rotationY, ease:Power1.easeInOut});
17 });
18
19 $("#move").click(function() {
20 if (wanderTween) {
21 wanderTween.kill();
22 wanderTween = null;
23 TweenLite.to($box, 0.5, {x:0, y:0});
24 } else {
25 wander();
26 }
27 });
28
29 //randomly choose a place on the screen and animate there, then do it again,
       and again.
30 function wander() {
31 var x = (($field.width() - $box.width()) / 2) * (Math.random() * 1.8 - 0.9),
32 y = (($field.height() - $box.height()) / 2) * (Math.random() * 1.4 - 0.7);
33 wanderTween = TweenLite.to($box, 2.5, {x:x, y:y, ease:Power1.easeInOut,
       onComplete:wander});
34 }
```

Listing 4.1: This example shows independent transformations, such as scaling, rotating and changing the position of some text.

Another example (**imposJS**) which uses the combined power of CSS and JS is the following. Some random text is animated with a lively turn-around effect. It is based on the ability to split up stings into separate characters via JS.

```
1 tl = new TimelineLite({onUpdate:updateSlider, onComplete:onComplete,
       onReverseComplete:onComplete, paused:true});
2
3 //do a simple split of the text so we can animate each character (doesn't
       require the advanced features of SplitText, so we just use split() and join
       ())
4 $text.html("<span>" + $text.html().split("").join("</span><span>").split("<span
       > </span>").join("<span> </span>") + "</span>");
5
6 //set a perspective on the container
7 TweenLite.set($text, {perspective:500});
8
9 //all of the animation is created in this one line:
10 tl.staggerTo($("#text span"), 4, {transformOrigin:"50% 50% -30px", rotationY
       :-360, rotationX:360, rotation:360, ease:Elastic.easeInOut}, 0.02);
11 }
```

Listing 4.2: This example implements a turn-around effect of text with the help of splitting up strings into single characters and animating them individually.

Chapter 5

Concluding Remarks