Animated rSlidy
Responsive HTML5 Slide Decks

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 Feb 2017

Abstract

This project report tries to give insights into the implementation of refinements of the already existent presentation software *rSlidy* provided by Keith Andrews of TUG.. It includes direct comparisons of the initial and the new version(s) in terms of design and functionality. Not only this contrast, but also the particular ways of implementing certain new features are listed and discussed.The focus of the project was to make the already working version of the web slideshow application closer to well spread desktop solutions. This entitled improving the user interface to be more interactive, responsive and user friendly. A big focus of the project was on animation solutions.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

Our group was assigned with the task of refining the already existent presentation software *rSlidy*. Therefore we first made some usability tests on our own in order to become familiar with the software and to find room for improvement. We interactively agreed with our instructor on features that needed implementation and those which would be nice to have, but not necessary.

The final submission comprises two versions of the new *rSlidy*. One which is completely independent and one which uses two third party libraries. The independent one might not look as impressive in some scenarios, but is free of external code. The other version is arguably better design-wise, but relies on third party libraries, which was not in favor of the instructor.

Chapter 2

rSlidy

Fernando use that pdfyou found to write as much as possible about the original *rSlidy*. And what where the reasons that this upgrade project was started - basicly the faults of the original *rSlidy*.

2.1 History?

2.2 Installation

2.3 Animated upgrade

Why we went to upgrade, how the new code install is, That there are 2 versions

Chapter 3

Changes of the Design

The design of *rSlidy* has undergone numerous major changes throughout our project. A comparison between the old and the new version is given in this chapter.

3.1 The Status Bar

The initial version of *rSlidy* was equipped with a permanent status bar, as shown in Figure 3.1. It has been modified in terms of design and functionality. The final appearance of the status bar is shown in Figure 3.2. The following individual changes have been made.

3.1.1 Progress Bar

A simple blue progress bar has been added on top of the status bar. Its purpose is to give some visual feedback about the current progress within a presentation. Its implementation is fairly simple. The progress bar container's width property is changed on each slide change with an animated transition (see Listing 3.1).

```
1 Rslidy.prototype.showSlide = function (slide_index) {
2   // ...
3   var progress_bar = document.getElementById("progress-bar");
4   progress_bar.style.width =
5         'calc(100%*' + (slide_index + 1) / this.num_slides + ')';
6   // ...
7 }
```

Listing 3.1: Adapting width of the progress bar container for authentic visual feedback

3.1.2 Rearrangement / Extension of the Navigation Elements

We found it simply more intuitive to have the input field for jumping to a specific slide in the middle of the forward / backward buttons. Apart from this, functionalities to jump to the first and respectively the last slide have been added. These two starightforward implementations can be seen in Listing 3.2.

Figure 3.1: Design of *rSlidy*'s original status bar. [Screenshot taken by the authors of this report.]

Figure 3.2: Design of *rSlidy*'s modified status bar. [Screenshot taken by the authors of this report.]

```
1  document.getElementById("status-bar-nav-button-first")
2    .addEventListener('click', function ()
3    {
4      this.showSlide(0);
5    }.bind(this));
6  document.getElementById("status-bar-nav-button-last")
7    .addEventListener('click', function ()
8    {
9      this.showSlide(this.num_slides - 1);
10   }.bind(this));
```

Listing 3.2: Implementation of the buttons for jumping to the first / last slide

3.1.3 Pin Functionality

In opposition to the original *rSlidy* status bar, the new one features pinning / unpinning. The pinned status bar works the same as the old one. The unpinned status bar disappears when not hovering over it. When the mouse is not close to the bottom of the document, only the progress bar is visible in the unpinned mode. Two subtle triangles have been added to the unpinned status bar which are meant to function as little indicators for the actual bar. This implementation may not be the most elegant one, because it is relying on the title of the button to work properly. Some simple boolean variable which describes whether the bar is pinned or not may be a more robust solution. Still, this (see Listing 3.3) is what we came up with and it works fine as long as the title tag of the pin button in the rslidy.js file is either "Pin the status bar" or "Unpin the status bar" (depending on whether the user wants the bar to be pinned or not by default).

```
1  Rslidy.prototype.pinToggleClicked = function (close_only) {
2    var pin_button = document.getElementById("status-bar-pin-button");
3    var status_bar = document.getElementById("status-bar-content");
4    var indicator_left = document.getElementById("progress-bar-indicator-left");
5    var indicator_right = document.getElementById("progress-bar-indicator-right")
         ;
6
7    if (pin_button.title == "Pin the status bar")
8    {
9      pin_button.title = "Unpin the status bar";
10     status_bar.style = "transform: translateY(0);";
11     pin_button.style.WebkitTransition = 'opacity 0.3s';
12     pin_button.style.MozTransition = 'opacity 0.3s';
13     pin_button.style.opacity = 0.5;
14     indicator_left.style.visibility = "hidden";
15     indicator_right.style.visibility = "hidden";
16   }
17   else
18   {
19     pin_button.title = "Pin the status bar";
20     status_bar.removeAttribute('style');
21     pin_button.style.opacity = 1;
22     indicator_left.style.visibility = "visible";
23     indicator_right.style.visibility = "visible";
24   }
25 };
```

Listing 3.3: Implementation of the buttons for pinning / unpinning the status bar

3.2 The Menu

The menu has been modernized and harmonized as seen in a direct comparison of Figure 3.3 and Figure 3.4. Implementation-wise these changes were mostly straightforward:

- Partial transparency has been added to the menu. The actual document thus slightly shines through the menu.
- The corners have been corners in order to get a smoother look in general.
- Shadow effects have been added to the edges of the menu in order to get a very basic 3D-effect.
- Harmonization has taken place. All links were exchanged with buttons. This means more consistency in the design altogether.
- One more checkbox has been added. It allows the user to switch between a version which, as usual, shows the address of a link on hover and a version which suppresses that standard function by removing the "href" property from all initial links.

3.2.1 User Set Default Values

By using the menu the user can always change the settings, however there was no function previously to define his own defualt settings for each seperate presentation. The function was realized through hidden input elements, which in initialization trigger the previously built toggle functions. To keep a constant structure and

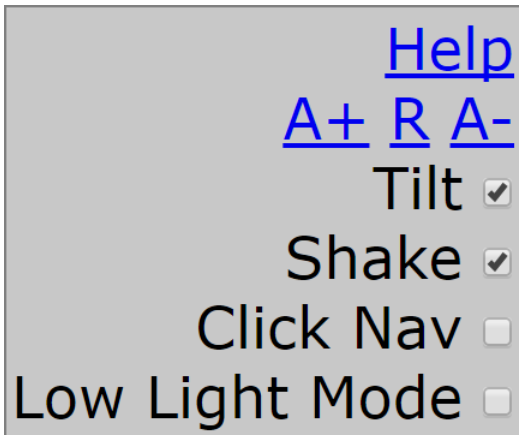


Figure 3.3: Design of *rSlidy*'s original menu. [Screenshot taken by the authors of this report.]

Figure 3.4: Design of *rSlidy*'s modified menu. [Screenshot taken by the authors of this report.]

preventing mistakes such as forgotten setting in middle of code, a container div with id *setupDefaultValues* has to be used inside the body to set the defaults. The constraint is only binded on the container id and each setting value id. The whole array of settings can be seen in code snipplet 3.4. The whole container can be excluded if default values work for the user, otherwise just the wanted changes are needed. Between the options please note, that a default font size option was added, however it requires the use of em values. The constraint is for correct change of font size on button click inside the menu, while the font size reset button will used the specified value. This could be solved in multiple ways, however em having a good scaling interpratation, it was decided that it unneeded.

```
<div id="setupDefaultValues" class="hidden">
  <!-- //font size should be specified in EM for correct zooming -->
  <input type="hidden" id="default-font-size" value="1em">
  <input type="hidden" id="statusbar-pin-locked" value="false">
  <input type="hidden" id="menu-tilt-used" value="true">
  <input type="hidden" id="menu-shake-used" value="true">
  <input type="hidden" id="menu-click-nav-used" value="false">
  <input type="hidden" id="menu-low-light-mode-used" value="false">
  <input type="hidden" id="menu-hide-address-used" value="false">
</div>
```

Listing 3.4: By adding the container div with id setupDefaultValues, the slideshow creator can define, which menu settings or font size should be defualt. The values provided in the snipplet are the default values of *rSlidy*. Only the desired value changes need to have an hidden input included in the container.

3.3 The Help Information

This information, which can be opened from the menu, used to be a usual alert box. Due to the instructor's wish to have no third party libraries included, there are two versions of our implementation for the new help popup. The initially intended version is based on a library called "sweetalert" (`https://limonte.github.io/sweetalert2/`). It allows animated popup messages with focusing on the actual text. The advantage of this way of implementation is the the polished design while having to rely on a third party library. The revised version simply opens a new tab to display the help message. This does not look as sophisticated as the other version, but still works fine and is an independent way of solving the popup problem.

3.4 Side Menus

The side menus in *rSlidy* consist of a slide overview and table of contents, both shortend in the application as Slides and ToC. In original version both menus were fixed in same position on the left side andwere by default hidden. By clicking the the menu buttons with their shorten names, the user could activate one and switch between both or close them. By overlapping different functions and demanding the user to activate either function with buttons in same corner of the screen, the workflow is unnaturally limited if not broken. Even more on mobile devices.

Therefore the refined version changed the design by moving each menu to its own side, Slide to the left and Toc to the right. Hover event was fixed on a much narrower menu container, to optimally trigger menu visibility. The original buttons in status bar were left, however their function became to lock each of the menus to the screen. Their size is also corrected for mobile devices, to optimize screen size, since there are many events triggered on it. Another addition to the side menus is animation for activation and auto-scrolling of hovered slide thumbnails in Slides, as a preview for some was to small in original version.

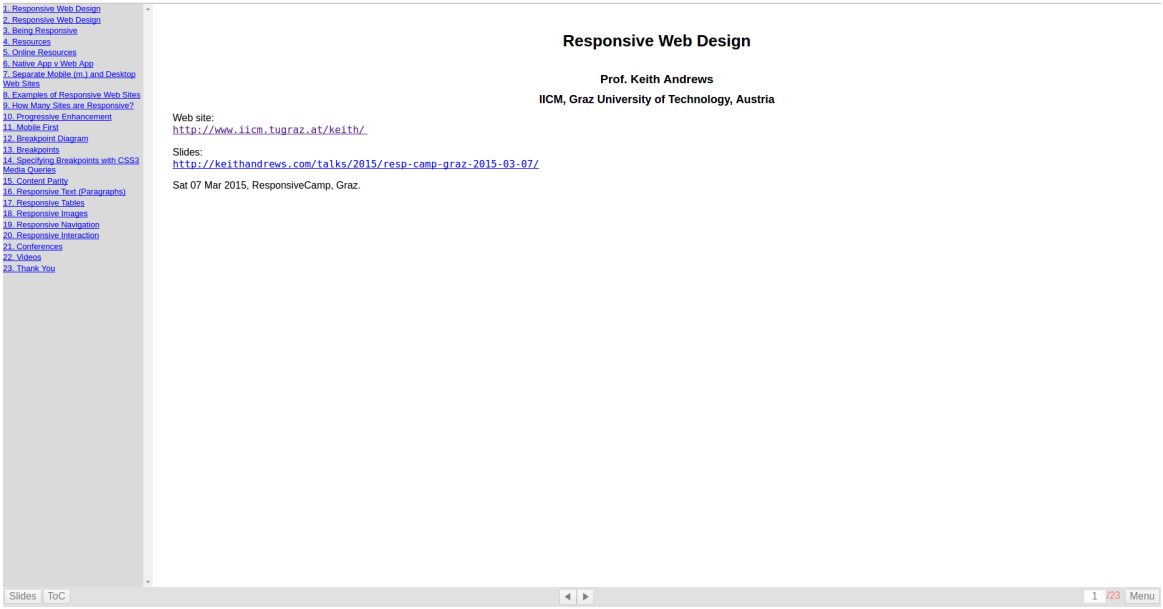


Figure 3.5: Design of *rSlidy*'s original Slides and ToC side menus, both fixed and overlapped on left side of user interface. [Screenshot taken by the authors of this report.]

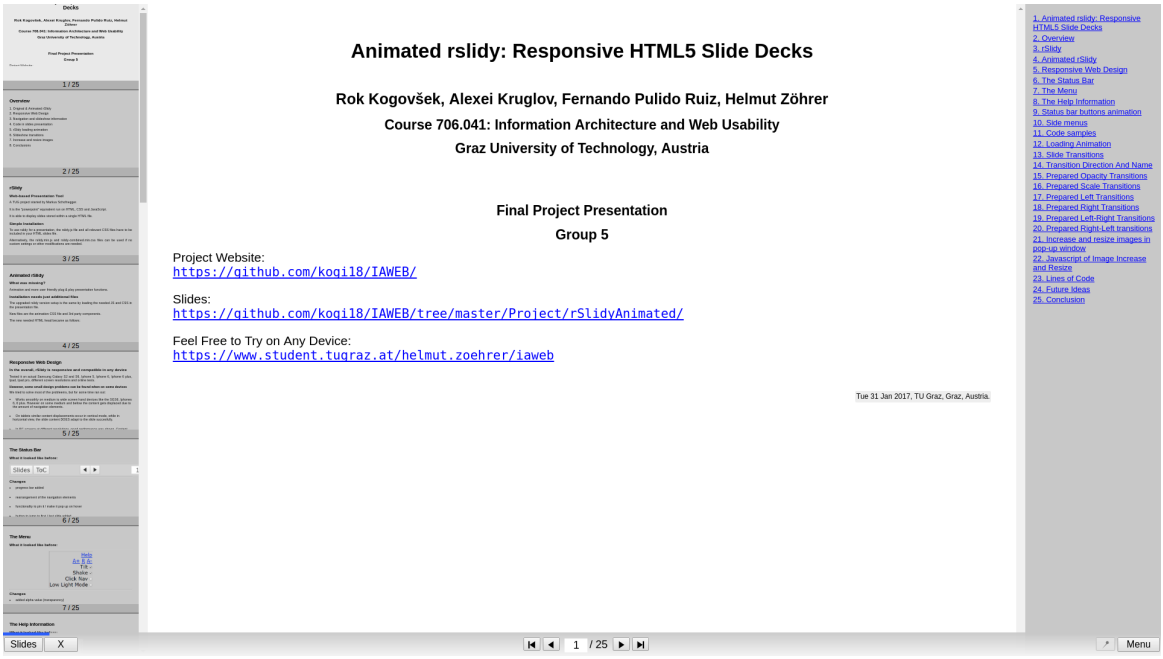
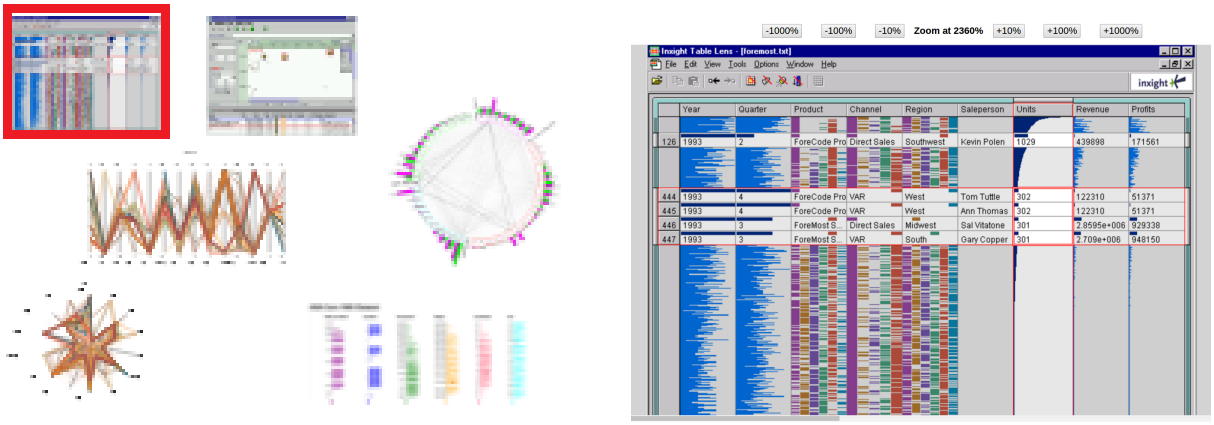


Figure 3.6: Design of *rSlidy*'s modified Slides and ToC side menus, which open on hover and lock to the screen on button click. SLides overview is fixed to left side and ToC is fixed to the right side. [Screenshot taken by the authors of this report.]

Chapter 4

Image Magnification

In this part of the project we have focused on presentation of images. We have added a zoom feature, which opens the clicked picture in a new window or in the enhanced version in a pop-up. The implementation works on different picture formats, of which we tested svg, jpeg, gif and png. Our enhancements give users a better opportunity to see detailed parts of the selected image, by fitting it to the width of the screen, while supporting zooming for smaller details or prefered size readjustment. Since zooming is involved and screen space is limited, we solve the overflow problem with scroll bars.



(a) Size of image set by the presentation creator. **(b)** Zoomed-in part of same image.

Figure 4.1: A SVG image inside the presentation before and after the magnification click. [Screenshots taken by the authors of this report, while the original image is from a provided presentation [Keith, 2015].]

On click the selected image opens in a new tab or in the enhanced version in a pop-up window. While the complex pop-up function is provided by a 3rd party solution, the simplier new tab opening is done by supplying HTML content through JavaScript's document.write function on a new blank document. The provided content includes the page layout, buttons, the magnified picture, and JavaScript, which is needed for image resizing. To simplify the JavaScript inclusion, a seperate function is written, which is then converted to tex through the String function.

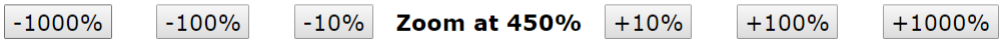


Figure 4.2: The magnification control bar next to the zooming buttons includes an updatable label, to report the current zoom in regards to original image size. [Screenshot taken by the authors of this report.]

In either the popup or tab window we can zoom it for +-10%, +-100%, and +-1000% by pressing the provided buttons. Alternatively with keyboard or mouse support, we can also zoom it with +/- keyboard buttons or CTRL plus mouse scrolling event by a the default step of 10. To fit it back into the default size, where zoom is 100%, we can do it by pressing keyboard button 0.

```
    // Input listeners
    var images = document.getElementsByTagName("img");
    var images  = content_section.getElementsByTagName("img");

    for (var i=0, len=images.length, img; i<len; i++) {
      img = images[i];
      img.addEventListener("click", function() {
      openImageTab(this.src);
      });
    }
  };
}

function openImageTab(imgSrc) {
  var newWindow = window.open();

  var htmlCode ="<head><title>rSlidy Image View</title><link rel='stylesheet'
      href='css/reset.css'><link rel='stylesheet' href='css/normalise.css'>" +
      "<link rel='stylesheet' href='css/rslidy.css'><link rel='stylesheet' href
          ='css/slides-default.css'></head>" +
      "<body><div class='slide imageAlert'><h1><button>-1000\%</button><button
          >-100\%</button><button>-10\%</button>Zoom at <span id='zoomNumber
          '>100</span>\%<button>+10\%</button><button>+100\%</button><button
          >+1000\%</button></h1>" +
      "<div><img id='zoomedImg' src='"+ imgSrc + "'></div></div>"+
      "<script type='text/javascript'>" + String(openImageTabListeners) + ";
          openImageTabListeners();</script></body>";
  newWindow.document.write(htmlCode);
}
```

Listing 4.1: Implementation of image detection and click event binding of content push. The called function is for the tab solution, while the pop-up solution seperates the control bar in the 3rd party sweetalert function call for layout improvement.

```
1
2   window.addEventListener('keypress', function (e) {
3     if (e.key == '+' || e.key == '-' || e.key == '0') {
4       var zoom = parseInt(titleElement.innerHTML);
5       if(e.key == '+'){
6         zoom = zoom + 10;
7       }
8       else if(e.key == '-')
9       {
10        zoom = zoom - 10;
11      }
12      else{
13        zoom = 100;
14      }
15      if(zoom > 0){
16        img.style.height = zoom * heightPer + "px";
17        img.style.width = zoom * widthPer + "px";
18        titleElement.innerHTML = zoom;
19      }
20    }
21  }, false);
22
23  var isCtrl = false;
24  window.addEventListener('keydown', function (e) {
25    if (e.which === 17) {
26            isCtrl = true;
27        }
28    }, false);
29  window.addEventListener('keyup', function (e) {
30    if (e.which === 17) {
31            isCtrl = false;
32        }
33    }, false);
34
35  window.addEventListener("mousewheel", function (e) {
36    if(isCtrl){
37      var delta = Math.max(-1, Math.min(1, e.wheelDelta));
38      var zoom = parseInt(titleElement.innerHTML);
39      if(delta > 0){
40        zoom = zoom + 10;
41      }
42      else if(delta < 0)
43      {
44        zoom = zoom - 10;
45      }
46      if(zoom > 0){
47        img.style.height = zoom * heightPer + "px";
48        img.style.width = zoom * widthPer + "px";
49        titleElement.innerHTML = zoom;
50      }
51    }
52  }, false);
53 }
```

Listing 4.2: The control functions for all events are same for both rSlidy versions.

Chapter 5

Animated Slideshow

The biggest downside of the original *rSlidy* when compared to alternative solutions, with focus on well spread desktop solutions, would be its static state. Namely the presentation flow achieved with the application was an immediate switch between states or slides. Even the user interface was behaving similarly in a state-switching way. In the design changes the effects of hiding elements with hover and transition CSS elements also include a better user experience. This follows the observations from the web animation survey Kogovšek et al. [2016], were the importance of animation for user experience was stressed out. With knowledge gathered from the mentioned survey we enhanced both the user interface as well the presentation flow by incorporating animation with CSS and JavaScript. For better overview, the animated old elements, that were already present in the original *rSlidy*, have CSS added near the original class definitions, while the new concepts, sections 5.1 and 5.5, are defined in the new CSS file *rslidy-animation.css*.

5.1 Initialization Progress Animation

A major problem with the original *rSlidy* appeared when big slideshow files where loaded. Since at load all elements are hidden to be processed before presenting, for big files long unclear loading times appear. Such initialization is not friendly for the user and leaves a bad impression. Therefore we wanted to implement a simple loading animation, that would load quickly, represent the application and be entertaining for the short or long wait. We took inspiration from the simple but effective CSS implementation of a pulz loader[Kogovšek et al., 2016] and modified it to our needs. The result can be observed in figure 5.1, where we can see, that the modification leaves quite a different impression from the inspiration source. To achieve all desired properties of the loader we implemented an animation of the letters of *rSlidy* form a mexican wave by jumping in delays. The design was kept simple also in colors, to be consistent with the color palate from the application.

5.1.1 Rendering Waits for Break

A bit trickier was the inclusion of the the html code on loading time. When onLoad JavaScript is called, the browser rendering process is waiting for a stop, to draw anything. However the break happens once the function is closed, so a go-around with setTimeout function with 1ms delay is needed to call the initialization. The whole initialization is now done so, that firstly the loader HTML code is included in the body, afterwards the setTimeout is called, which calls a function that calls the init function from the Rslidy object to render the loader before the initialization. The loader is then hidden, when the same init function decides the current slide, which triggers the slide loading.

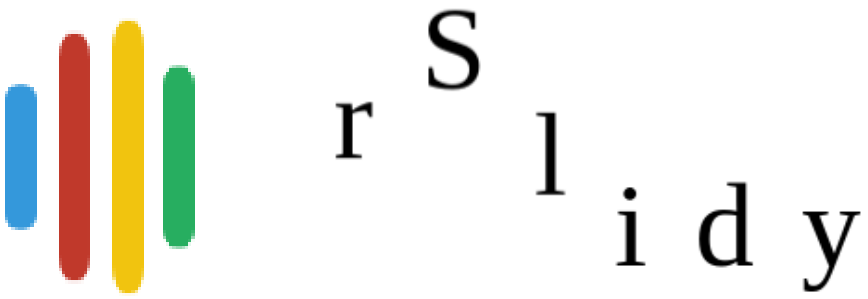


Figure 5.1: Screenshot of the loading animations midway. On the left we see the inspirational pulz loader, while on the right we see the resulting rearrangement for *rSlidy*. [Screenshot taken by the authors of this report.]

```
#loader div{
  display: inline-block;
  padding: 0.2em;
  animation: jump-loading 1s ease-in-out infinite;
}

/*Change delay per child*/
#loader div:nth-child(1) {  animation-delay:  0;  }
#loader div:nth-child(2) {  animation-delay: 0.1s;  }
#loader div:nth-child(3) {  animation-delay: 0.2s;  }
#loader div:nth-child(4) {  animation-delay: 0.3s;  }
#loader div:nth-child(5) {  animation-delay: 0.4s;  }
#loader div:nth-child(6) {  animation-delay: 0.5s;  }

@keyframes jump-loading {
  0%  { transform: translate(0,   0);   }
  20% { transform: translate(0, -1.5em);  }
  40% { transform: translate(0,   0);   }
}
```

Listing 5.1: The designed loader simply has its inner divs that contain letters jump in the first 40% of the animation time with 0.1s delays between jump starts.

5.2 Button Animation

Similar to an animated hamburger icon, which changes its shape on click, the buttons within the status bar of *rSlidy* are animated now as well. Listing 5.2 shows how the animated flip was created. These style changes and the button's text changed to an "X" lead to a simple and intuitive animation of a button which turns around to change its functionality. While the animation is done in CSS, the text change is done by the JavaScript setTimeout, to change the value halfway through the animation, when the text is not visible.

```
#button-overview, #button-toc, #button-menu{
  animation-duration: 0.3s;
  animation-timing-function: ease-in-out;
  animation-fill-mode: forwards;
  animation-name: flip2Face;
}

#button-overview.clicked, #button-toc.clicked, #button-menu.clicked{
  animation-name: flip2Back;
  transform: rotateY(180);
}
```

Listing 5.2: Implementation of the animation of the buttons in the status bar

5.2.1 Same Element Animation Issue

By having a direction and play animation property it seems that there is no need for two keyframe definitions, as in code snipplet 5.2. However getting it to work on same element is another issue. By researching forums, the main issue lies in the nature of CSS, overlaping properties from previous class, which seems to include counters. This becomes troublesome, when animation count is not infinite, like with button rotations example. While a simple transition would work if it was just a rotation, including a bit of scaling in the middle of the transformation to get a better effect, makes it impossible to solve with transitions. This issue became also a problem in section 5.5. The best pure CSS solution solution without adding additional HTML elements or doing JS animation was to actually write a reverse keframe copy and use that name in the needed class, which seems to reset the counter due animation initialization.

5.3 Hiding Elements

While hiding elements on hover elements by itself already raises the user experience, just by adding simple transition or animation CSS element we can direct the switch between states into a smooth way to enhance the user workflow. For better control we also used the Cubic Bezier function, with the values shown in figure 5.2.

5.4 Preview Scrolling

The Slides previous most of the time give you a nice preview of the slides. However depending on the amount of text or even the screen size due to using proportions for the thumbnails the preview may not be clear enough which slide it is due to the text overflow being hidden. Therefore we added a simple on hover animation of translating the thumbnail content by 20% upwards. The only thing also to note was the choosing of the correct content to move. Namely moving the first child also moves the afixed click event, while further children had problems with z-indecies. Understanding the z-index property and the structure of your CSS code is the key for such situations. A great by-product of an on hover animation is that is a great indicator for the current mouse position, which the original *rSlidy* on hover event only covered by slightly changing the color.

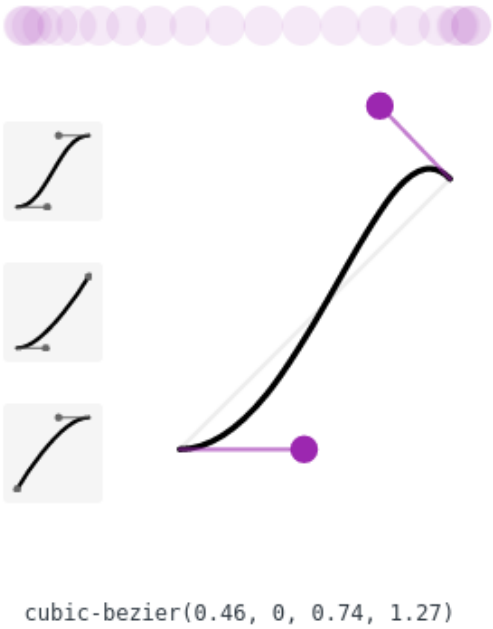


Figure 5.2: The values of the Cubic Bezier function used for smoother hiding of elements and its plot representation [Screenshot of the representation in Chrome Developer Tools taken by the authors of this report.]

5.5 Slide Transitions

The last described animation was also the biggest contribution for a more dynamic presentetion flow. The goal was to include a simple way for the user to chose animated transitions between slides. To keep the previous version of immediate transitions also as an option of *rSlidy*, we simply defined a body class *animated*, on which all animated transition classes are referenced through.

First problem we faced was the original definition of *hidden*, that uses display property set to none. This sets the the element not to be included in the render list of the browser, which however is required for animations to be applied. Since the display property is uncompatible with animation and the hiding of slides is the core of the application, we redefined the the *hidden* for *animated* bodies. The new definition uses visibility and sizes switched to hidden and 0 to achieve similar result as display set to none, while keeping it in the rendering list. Sadly this also means, that all animation keyframe definitions get a suffix for the reseting of this properties to keep the design intact. Since the suffix was needed, it made a much easier to read implementation by defining all slides as hidden and to get the end state of an animation as the afterstate, compared to to defining extra classes or making the code even complexer.

For a slide transition we defined the interchanging slides as *previous* and *active*. The former being the start of the animation, at the start time became marked as *hidden*, while the latter is the only slide that at the same time became without the *hidden* class. Since the *previous* slide is one of many *hidden* slides, the core function *showSlide* had to be slightly modified in its loop over the slides to keep the index of *previous* in memory. This was achieved with the classes *animate*, *animatedForward*, *animatedBackwards*. The first class is the marker class, while the remaining ones define the direction of the transition of *previous* and *active* slides. Each is asserted by the increase or decrease of the slide index. In the definition we see, that with no change in index there should be no animation. This is needed, since the change in hash tag in the URL also triggers *showSlide*. The cases when it is needed is in case of initial loading, refreshing and manual change of slide by address. However, each click to the next or previous slide also changes the hash tag. Since we do not want triggering the animation on initialization, refresh or update after next and previous was processed already, the absence of direction classes does not trigger transitions. Therefore all transition definitions become a combination transition name and direction classes, that use one of the defined keframes. The primary reason for direction

classes was however creation of complex animation where the direction flow has to be followed to not confuse the observer. For example if slides go all to the right, if we step back we would like to see them move backwards or in other words to the left.

```
body.animated .slide{
  display: block;
  overflow: hidden;
  visibility: hidden;
  height: 0;
  width: 0;
  padding: 0;
}

@keyframes opacityOff{
0%  {opacity: 1; visibility: visible; height: auto; width: auto; padding: 2em;}
99% {opacity: 0; visibility: visible; height: auto; width: auto; padding: 2em;}
100%{opacity: 0; visibility: hidden; height: 0; width: 0; padding: 0;}
}
@keyframes opacityOn{
0%  {opacity: 0; visibility: hidden; height: 0; width: 0; padding: 0;}
1%  {opacity: 0; visibility: visible; height: auto; width: auto; padding: 2em;}
100%{opacity: 1; visibility: visible; height: auto; width: auto; padding: 2em;}
}
@keyframes opacityOnList{
0%  {opacity: 0; visibility: hidden; height: 0; width: 0;}
1%  {opacity: 0; visibility: visible; height: auto; width: auto;}
100%{opacity: 1; visibility: visible; height: auto; width: auto;}
}

/* General transition animation settings */
body.animated .slide,
body.animated .slide ul.incremental li:not(.invisible){
  animation-fill-mode: forwards;
  animation-duration: 1s;
  animation-timing-function: linear;
}
body.animated .slide ul.incremental li:not(.invisible){
  animation-fill-mode: backwards;
}
/* Without direction we skip the animation*/
body.animated :not(.animatedForward):not(.animatedBackwards).slide{
  animation-duration: 0s;
  /* for loader preloading it is set to the minimum JS 1ms delay*/
  animation-delay: 0.001s;
}
/* TIME DELAY FOR NEW SLIDE SHOULD BE SAME AS TRANSITION DURATION*/
body.animated :not(.hidden).slide{
  animation-delay: 1s;
}
```

Listing 5.3: Redefinition of hidden on all slides, an example of a keyframe with suffix due to the redefinition and general animation settings for animated transitions.

Since the definition of classes was done rather long and complex, the result in the slide HTML file is, that the user has just ot append the transition class name either in the body class as a gloabal setup or in slide or list class for local setup. The prepared animated transition sum up to six: *opacity, scale, sliding-left, sliding-right, sliding-left-right, sliding-right-left*. Their meaning should be clear from the words and for extra clarification see

figure 5.3. To make it more distincted for the user and easier to read the HTML slide file, the global and local seperation is done even in the class name, by apending *-animation* to the name. For example the defualt global transition (when no transition name was referenced but *animated* body was used) *opacity*, becomes locally *opacity-animation*.

```
/*************************************************/
/*      DEFAULT OPACITY transition      */
/*************************************************/
/* General animation settings for old slide */
body.animated .hidden.slide.animate,
/* Also sets the calls for default animation of opacity change */
body.animated.opacity .hidden.slide.animate,
body.animated .hidden.slide.animate.opacity-animation{
  animation-name: opacityOff; /* Default animation */
}

/* General animation settings for new slide */
body.animated :not(.hidden).slide,
/* Also sets the calls for default animation of opacity change */
body.animated.opacity :not(.hidden).slide,
body.animated :not(.hidden).slide.opacity-animation{
  animation-name: opacityOn; /* Default animation */
}

body.animated .slide ul.incremental li:not(.invisible),
body.animated.opacity .slide ul.incremental li:not(.invisible),
body.animated .slide.opacity-animation ul.incremental li:not(.invisible),
body.animated .slide ul.incremental.opacity-animation li:not(.invisible){
  animation-name: opacityOnList; /* Default animation */
}
```

Listing 5.4: The default opacity transition is simple example of needed binding of HTML elements, transition names, directions and transition keyframes.

5.5.1 Solution Limitations

As noted in section 5.2.1, one execution count animations have problems reversing the keframes for backwards direction. Additionally next to copying the code for the reverse keframe, one has to keep in mind that visibility property is a binary value and cannot be interpolated. An extra copy of keframes is also needed for the list animation, since lists use different padding than the slide keframe suffix. Another drawback in the actual execution of costum transitions is, that atmost 1 slide has sizes different than 0. This means we have to wait for the previous slide to become hidden before the active animation should start. This guideline secures, that the slide position will not jerk in the middle of transition due to the size changes of the other slide. There should be a possible solution, for exmaple with z-index and absolute postion redefiniton, however we ran out of time to test such ideas.

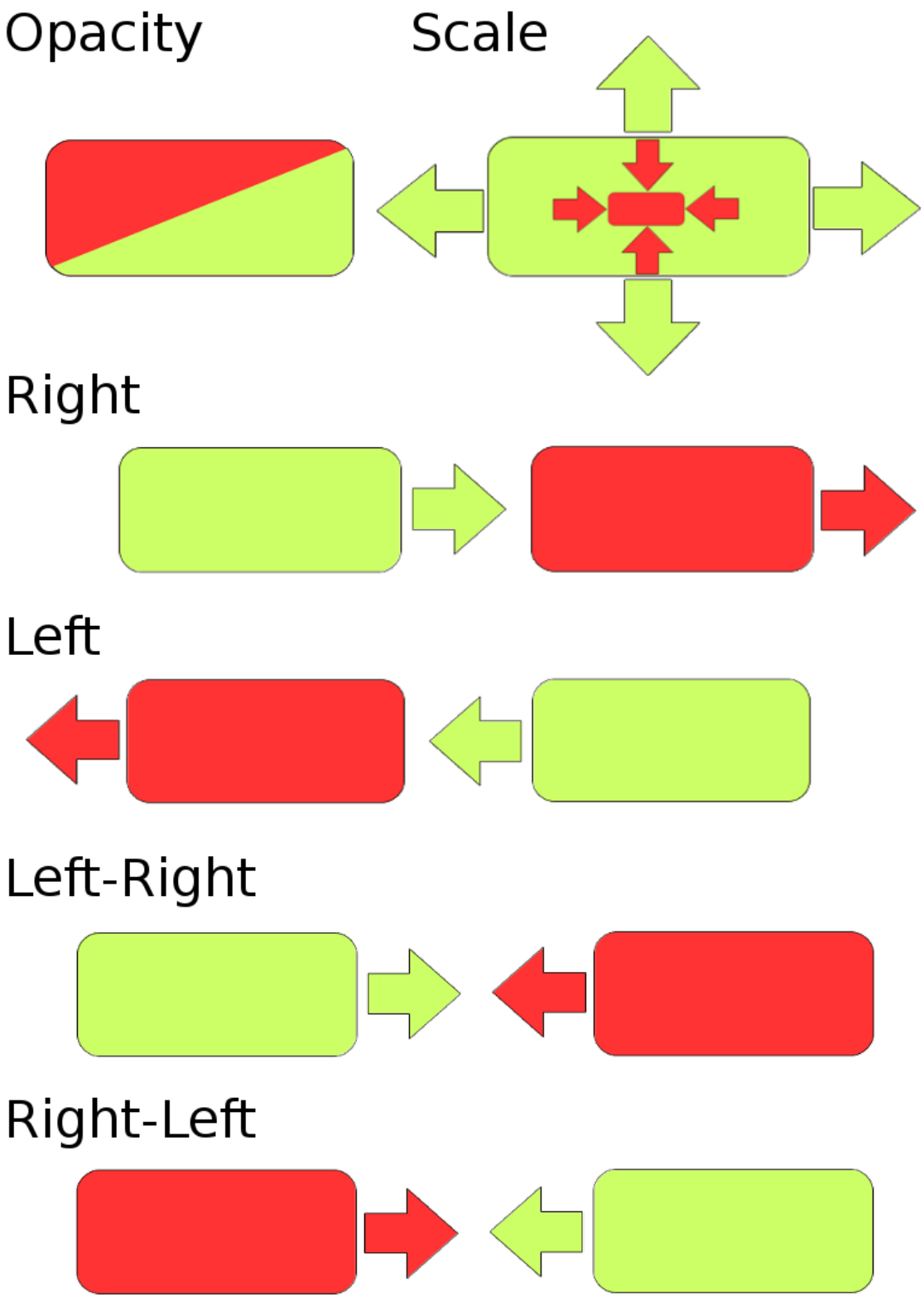


Figure 5.3: Diagram of *rSlidy* included transitions and their flow. Red slides are the previous slides, while green are active slides. The position of active and previous slide is shown as relative position to the other. [Diagram is prepared by the authors of this report.]

Chapter 6

3rd Party Code Support

During development and research for good user experience solutions we also found many 3rd party implementations, of which some could be directly used with *rslidy* with just slight adjustments. Of them we used two with MIT or similar rights range license and included them in the extended code solution.

6.1 Highlight.js

6.2 SweetAlert2

Chapter 7

Concluding Remarks

CSS LOC 800+ JS LOC 500+

Still room for improvement - Some tasks that were disscussed at the planning stage but deemd not needed in this stage

TypeScript for Grunt Including Subheadings with in TOC on heading hover Animated CSS text tyles Canvas to actually draw on the slide - Marker support

Achieved better user experience for slide shows

Interesting problems Same animation on same element cannot be triggered in subclass - either need additional element or reversed keframe

Element rendering knowledge is needed for animation planning - the display:none problem.

RWD testing should not be underestimated — HERE Fernandos text

Javascript preloading and initialization tricky with HTML rendering

Bibliography

Keith, Andrews [2015]. *Data Visualisation*. English. IICM, Graz University of Technology, Austria. 2015. http://keithandrews.com/talks/2015/dd-2015-11-26/ (cited on page 11).

Kogovšek, Rok, Alexei Kruglov, Fernando Pulido Ruiz and Helmut Zöhrer [2016]. *Web UI Animation*. 2016 (cited on page 15).