Design of video players for branched videos

Design av videospelare för förgrenade videor

Eric Lindskog & Jesper Wrang

Supervisor: Niklas Carlsson
Examiner: Marcus Bendtsen
Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under 25 år från publiceringsdatum under förutsättning att inga extraordinöra omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art. Upphovsmannens ideella rätt innefattar rätt att bli nämnt som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart. För ytterligare information om Linköping University Electronic Press se förlagets hemsida [http://www.ep.liu.se/]

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page [http://www.ep.liu.se/]

© Eric Lindskog & Jesper Wrang
Students in the 5 year Information Technology program complete a semester-long software development project during their sixth semester (third year). The project is completed in mid-sized groups, and the students implement a mobile application intended to be used in a multi-actor setting, currently a search and rescue scenario. In parallel they study several topics relevant to the technical and ethical considerations in the project. The project culminates by demonstrating a working product and a written report documenting the results of the practical development process including requirements elicitation. During the final stage of the semester, students create small groups and specialise in one topic, resulting in a bachelor thesis. The current report represents the results obtained during this specialisation work. Hence, the thesis should be viewed as part of a larger body of work required to pass the semester, including the conditions and requirements for a bachelor thesis.
Abstract

Interactive branched video allows users to make viewing decisions while watching, that affect the playback path of the video and potentially the outcome of the story. This type of video introduces new challenges in terms of design, for example displaying the playback progress, the structure of the branched video as well as the choices that the viewers can make. In this thesis we test three implementations of working video players with different types of playback bars: one fully viewed with no moving parts, one that zooms into the currently watched section of the video, and one that leverages a fisheye distortion. A number of usability tests are carried out using surveys complemented with observations made during the tests. Based on these user tests we concluded that the implementation with a zoomed in playback bar was the easiest to understand and that fisheye effect received mixed results, ranging from distracting and annoying to interesting and clear. With this feedback a new set of implementations was created and solutions for each component of the video player were identified. These new implementations support more general solutions for the shape of the branch segments and the position and location of the choices for upcoming branches. The new implementations have not gone through any testing, but we expect that future work can further explore this subject with the help of our code and suggestions.
Acknowledgments

We would like to thank our supervisor Niklas Carlsson for his guidance and support during the project. We would also like to thank Oscar Andell, Albin Andersson, Linnea Lundström, Sebastian Ragnarsson, Samuel Johansson and Karol Wojtulewicz for providing useful feedback.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract</strong></td>
<td>iii</td>
</tr>
<tr>
<td><strong>Acknowledgments</strong></td>
<td>v</td>
</tr>
<tr>
<td><strong>Contents</strong></td>
<td>vi</td>
</tr>
<tr>
<td><strong>List of Figures</strong></td>
<td>viii</td>
</tr>
<tr>
<td><strong>List of Tables</strong></td>
<td>ix</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Background</td>
<td>1</td>
</tr>
<tr>
<td>1.3 Aim</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Research questions</td>
<td>2</td>
</tr>
<tr>
<td>1.5 Contributions</td>
<td>3</td>
</tr>
<tr>
<td>1.6 Delimitations</td>
<td>3</td>
</tr>
<tr>
<td><strong>2 Theory</strong></td>
<td>4</td>
</tr>
<tr>
<td>2.1 Branched video streaming</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Adaptive streaming</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Dynamic Adaptive Streaming over HTTP</td>
<td>5</td>
</tr>
<tr>
<td>2.4 Prefetching</td>
<td>5</td>
</tr>
<tr>
<td>2.5 Web technologies</td>
<td>6</td>
</tr>
<tr>
<td>2.6 Fisheye effect</td>
<td>6</td>
</tr>
<tr>
<td>2.7 Usability testing</td>
<td>7</td>
</tr>
<tr>
<td>2.8 Related work</td>
<td>8</td>
</tr>
<tr>
<td><strong>3 System design and high-level frameworks</strong></td>
<td>9</td>
</tr>
<tr>
<td>3.1 Video for DASH</td>
<td>9</td>
</tr>
<tr>
<td>3.2 DASH Player</td>
<td>9</td>
</tr>
<tr>
<td>3.3 Playback bar</td>
<td>11</td>
</tr>
<tr>
<td>3.4 Branch selection</td>
<td>13</td>
</tr>
<tr>
<td>3.5 Video player</td>
<td>13</td>
</tr>
<tr>
<td><strong>4 Preliminary user study</strong></td>
<td>14</td>
</tr>
<tr>
<td>4.1 Usability tests</td>
<td>14</td>
</tr>
<tr>
<td>4.2 Results</td>
<td>14</td>
</tr>
<tr>
<td>4.3 Summary of pre-study results</td>
<td>16</td>
</tr>
<tr>
<td><strong>5 Improvements and ideas for future work</strong></td>
<td>17</td>
</tr>
<tr>
<td>5.1 Arbitrary functions for branch shapes</td>
<td>17</td>
</tr>
<tr>
<td>5.2 Playback bar transformations</td>
<td>18</td>
</tr>
</tbody>
</table>
5.3 Management of complex branch structures .................................. 19
5.4 Playback bar position .............................................................. 19
5.5 Branch choices ................................................................. 19
5.6 Example implementation ....................................................... 20
5.7 Metafile creation tool ......................................................... 22

6 Discussion ................................................................. 23
  6.1 Pre-study results .............................................................. 23
  6.2 System design .............................................................. 23
  6.3 Pre-study method and usability tests .................................... 24
  6.4 The work in a wider context .............................................. 24

7 Conclusion ................................................................. 26

Bibliography ................................................................. 27
List of Figures

1.1 Branch selection in *Puss in boots: Trapped in an Epic Tale* made by Netflix . . . . . . . 2
1.2 Playback bar in *Puss in boots: Trapped in an Epic Tale* made by Netflix . . . . . . . 2

2.1 Simple example of the branching property of branched videos . . . . . . . . . . . . 4
2.2 The encoding process used in HAS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.3 The fisheye effect visualised . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3.1 The mapping of a linear video to a branched video . . . . . . . . . . . . . . . . . . . 10
3.2 JSON format for representing the branched video . . . . . . . . . . . . . . . . . . . 11
3.3 The full view implementation of the playback bar without any distortions . . . . . . 12
3.4 The zoom and follow implementation of the playback bar without any distortions . . . 12
3.5 The fisheye implementation of the playback bar . . . . . . . . . . . . . . . . . . . . . . . . . 13
3.6 The video player with the full view implementation of the playback bar . . . . . . . 13

4.1 Distribution of the SUS-scores for the *full view* implementation . . . . . . . . . . . . 15
4.2 Distribution of the SUS-scores for the *zoom and follow* implementation . . . . . . 15
4.3 Distribution of the SUS-scores for the *fisheye* implementation . . . . . . . . . . . . . 16

5.1 Example of scaling of the arctangent function to pass through two points . . . . . . 18
5.2 A refined user interface showing two choices on top of the video . . . . . . . . . . . . 20
5.3 A refined user interface showing two choices on top of the playback bar . . . . . . . 21
5.4 A refined user interface showing two choices on top of a playback bar that imple-
ments *zoom and follow* and uses a logarithmic function for the branches . . . . . . 21
5.5 A refined user interface showing a branch being highlighted as a result of hovering
over the corresponding button . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
5.6 The tool used for creating and editing the branch metafiles . . . . . . . . . . . . . . . . . 22
List of Tables

4.1 Standard deviation and Mean score of all implementation . . . . . . . . . . . . . . 16
1 Introduction

1.1 Motivation

Branched video streaming is a way to create interactive non-linear videos where viewers can make viewing choices while watching, and in which these choices will yield different viewing experiences and storylines. This type of video allows for interactive experiences that could be used to create, for example educational videos or movies with multiple alternative stories. Previous work has shown that it is possible to implement this technology, but also that this new type of viewing introduces a lot of new challenges that have to be solved to put it up to par with traditional streaming services that viewers are used to [1]. Solutions to some of these challenges have already been explored, such as a full implementation of a HAS based video player with proper prefetching policies and buffer management [2]. There has also been a study of how different policies for prefetching can create a smoother experience for the end user [3].

An aspect that has not been explored yet is the graphical part of video players for branched videos. In particular, there is a need of a playback bar that can display the playback and buffer condition of a branched video. There is also a need to intuitively display the branch choices for the different paths that can be taken. These things can be implemented in multiple ways, each with varying usability. For traditional linear videos the playback bar is often displayed as a simple bar below the video. This approach might not be ideal for branched videos since these videos splits into multiple different paths that in turn can split into even more paths. On top of that we need to display a selection of choices that has to be well integrated with the rest of the player for a pleasant user experience.

1.2 Background

The streaming service Netflix has recently produced their first interactive video, called *Puss in Boots: Trapped in an Epic Tale*. Throughout the video the viewer is prompted to choose between two alternatives in the form of two pages in a book as can be seen in Figure 1.1.
1.3 Aim

The options are also explained by a narrator as you are making the choice. The playback bar consists of all the choices, in chronological order, that the user can make throughout the video. This is shown in Figure 1.2. This is a specific implementation that the content creators have to adapt to, which may limit him or her in the creation of a general branched video.

In 2008, the video platform YouTube added the ability to add annotations to videos. These annotations can be used to redirect the viewer to another video. This allows content creators to create videos where the viewer can choose between clicking on two or more annotations at the end of the video that redirects them to another video. These videos can in turn redirect the viewers to other videos, creating a similar experience to branched videos. The main disadvantage of this is that the page will reload each time you make a choice. Another issue is that there is no prefetching for the upcoming “branches”. There is also no clear way of getting an overview of the entire structure of the videos. All of this can result in a disruptive and frustrating viewer experience.

1.3 Aim

The aim of this thesis is to create a general solution for video players that can be used for branched videos. This will be an iterative process of creating and testing multiple implementations and then using that information to create new and improved implementations.

1.4 Research questions

- Does applying a fisheye effect to the playback bar improve the understanding of playback conditions?
1.5 Contributions

• Does zooming in on the current position in the playback bar improve the understanding of playback conditions?

• What are some possible solutions for displaying the choices of different branches?

1.5 Contributions

In this thesis we have created three initial implementations of branched video players, which we have evaluated with usability tests. Based on the results from these tests we have developed design improvements and ideas that can be used for further work on branched video players. We have also developed tools and scripts to simplify the creation of branched videos from linear videos.

1.6 Delimitations

We have performed a very limited number of tests for the evaluation of the different playback bars. This small sample size can make it difficult for us to draw well founded conclusions from the patterns we see.

Our test group is also very homogeneous, and contains mostly young IT students. This could affect our results since these students are most likely more familiar with computers and technology than a randomly selected group of people would be.
2 Theory

2.1 Branched video streaming

Branched video streaming is a type of non-linear streaming that allows multiple different paths to be taken in a given video [1]. These videos let the user interact and make choices as they watch. This means that a video can be branched into multiple videos, which in turn can branch into even more videos. An example of this branching can be seen in Figure 2.1. The user would in this case initially watch the video as if it is a regular linear video. When reaching the first vertex, called a branch point, they will be presented with a choice between two paths. After making a choice, the selected path will be streamed until it reaches another branch point.

![Figure 2.1: Simple example of the branching property of branched videos](image)

2.2 Adaptive streaming

One way of reducing stalling while watching a video is to use a streaming technique that adapts the quality of the video depending on the current network condition. This is done by encoding the video at different bit rates and then selecting the bit rate best suited for the situation. To make this bit rate switch possible, the video is split into multiple smaller parts.
called segments. In Figure 2.2 we show a simplified process of creating these video segments at different bit rates from a high quality video.

One variant of this streaming technique is called HTTP Based Adaptive Streaming (HAS). It is used to stream videos over HTTP which means that the videos can be viewed in a browser. This is normally done with linear type videos, but studies [2] have shown that we can utilise this type of adaptive streaming for branched videos as well.

2.3 Dynamic Adaptive Streaming over HTTP

Dynamic Adaptive Streaming over HTTP, or DASH, is a streaming technique that is based on HAS [4]. It shares a lot of the same ideas as HAS, like encoding a video in multiple bit rates and splitting the videos into segments. DASH is the first HAS based streaming technique that is an international standard. It is codec independent, meaning the video and audio can be encoded with any coding format.

Media Presentation Description

DASH uses a meta file for each video, called Media Presentation Description, or MPD for short. This file contains information relevant to the video, for instance the length of a segment, what encoding is used, as well as the different resolutions and bit rates available. Three basic concepts that are used in most MPDs are, periods, adaption sets and representation. The period simply describes the start time and duration of a video. An adaption set allows for basic separation of video and audio streams. A common case is to have to have one video adaption set and then multiple audio sets with different spoken languages. Lastly a representation describes content of a specific quality. With HAS there would be a representation for each bit rate.

dash.js

dash.js is an open source project driven by DASH Industry Forum that implements the DASH technique in the JavaScript language[4]. It works by binding a HTML video element to their implementation of DASH. dash.js requires a MPD file to determine what content it should display and where to look for it. The adaptive property of HAS is built into dash.js and it will therefore switch between different qualities as it sees fit. This video player currently does not support branched videos so it has to be implemented separately.

2.4 Prefetching

Another method for preventing stalling is called prefetching. Put simply, prefetching is just requesting content that is expected to be used. This is done for linear video by requesting segments ahead of time and storing them in a buffer. For non-linear video this is not as trivial, as one needs to establish a more elaborate prefetching policy. When nearing a branch point

\footnote{Available at https://github.com/Dash-Industry-Forum/dash.js/wiki}
we might want to prefetch the first segments of each path that can be taken. This way, we can reliably get a smooth viewing experience independent of what choice the user makes. A more advanced prefetching policy could assign “weights” to the branches depending on how popular they are. This would mean that we prioritise segments from popular branches since those are the most likely to be chosen. These are just some possible policies for prefetching and this subject has been explored in more detail in previous work [3],[5].

### 2.5 Web technologies

The video player will be built inside of a web page that can be viewed in a browser. Web pages usually consist of a combination of three major technologies: HTML, CSS and JavaScript. HTML is a markup language that describes the structure and semantics of a web page, such as paragraphs, links and images. CSS describes the presentation of that content, for instance the layout, colour and the font. JavaScript is the scripting language for web pages that handles things like user interactions and fetching content from servers.

#### Canvas

Canvas is a web technology for rendering 2D shapes and bitmaps inside of a browser. A canvas is defined as an element in HTML and can be accessed in JavaScript. With the canvas we can draw basic figures like rectangles and circles with different colours and sizes. It can also draw different types of curves, like straight lines, quadratic curves and Bézier curves. The canvas API has built in support for different types of transforms, like translations and scaling. With these basic components it is possible to draw complex graphics, for instance a playback bar that can follow the current playback progress.

#### Node.js and npm

Node.js is a way to run JavaScript on the server-side instead of client-side. Node.js allows importing and exporting of functions and variables from different files, which is not currently supported in JavaScript. This means the code can logically be split up into modules that can easily be reused throughout the code. While client-side JavaScript does not support this natively, it can be emulated by inserting all dependencies and bundling everything into one big file before it is served to the client. With Node we can easily include code from libraries that other people have written. This is often done with the JavaScript package manager npm which offers a large database of submitted JavaScript packages.

### 2.6 Fisheye effect

The fisheye effect is a visual distortion for creating a wide panoramic image. The fisheye effect can be implemented in a few different ways [6] but the most common one (hemispherical) creates a spherical view of the image where the centre of the image appears closer and larger than the surrounding objects. This can be done by using an ultra-wide angle lens on a camera. The effect can also be accomplished in software by using different types of mapping functions. A visual demonstration of the fisheye effect can be seen in Figure 2.3. The leftmost image is an overview of a tunnel. The middle image is photo taken inside the tunnel looking straight at the wall. In the right image we can see the same image but with the fisheye effect applied, making it possible to see both ends of the tunnel.

---

2.7 Usability testing

Survey
A common way of testing usability is through surveys where the user answers questions about how they perceived the usability of the service being tested. Surveys can be difficult to make as it is easy to write questions that can be ambiguous resulting in the user not necessarily answering the question one wanted to have answered [8]. There is also a risk of bias in questions that are being asked, steering answers to the results that we expect.

For these reasons we have chosen a popular survey tool called System Usability Scale (SUS) [9]. This tool was introduced by Brooke in 1996 and aims to easily and quickly collect users’ subjective ratings of a product’s usability. SUS has proven to be a reliable tool and is widely used in system engineering [10][11]. A SUS survey contains 10 questions about the product and how the user perceives its usability. The answer to each question is given on a scale of 1 to 5, that is dependent on much they agree with the question. These scores are then added up to get an overall score for the usability of the product.

Observation
Another way of testing usability is through creating a structured test environment with a user and a moderator. The user is given a task consisting of a series of questions or instructions that the user should follow [11]. The moderator observes the user as they are attempting to complete the task. Before the test starts the moderator can either instruct the participant to continuously say its thoughts out loud as it is performing the task, or allow the user to explain its thought process after they have finished the test. These two methods are called concurrent thinking aloud and retrospective thinking aloud respectively. The latter usually requires eye-tracking data or video from the test to help the user recall their thought process. A disadvantage with the thinking aloud technique is that the moderator has to rely on the user to “answer” all the questions he or she wants answered without the user knowing the questions.

An alternative to the techniques mentioned above is verbal probing in which the moderator asks the user about functionality, for example: “How useful did you find the calculate total cost button?”. This technique guarantees that you get an answer for all the questions you want answered unlike the thinking aloud techniques, as users may find it easier to simply answer a question rather than verbalising their thoughts. One disadvantage of using verbal probing is that the answers supplied by the users can be biased as a result of leading question [8].

The use of a fisheye effect can make it easier to interact with larger interfaces and things that require wide field of view. This has been explored in previous work and has proven to be useful [7].

![Figure 2.3: The fisheye effect visualised](image-url)
2.8 Related work

In previous work by Meixner et al. [12][13], they have implemented and tested non-linear video player solutions, for both desktop and mobile units. These solutions rely on video annotations for the desktop implementation and separate buttons next to the video for the mobile version. The desktop version also includes additional features such as a table of contents, a search feature and logging functionality.

Uddin et al. [14] have worked with improving the revisitation performance for both PDF viewers as well as linear video players. Revisitation is when a user returns to a location in a video or PDF that they have already visited. They improved the performance by placing arbitrary icons or thumbnails below the scroll and playback bar. Both the icon and the thumbnail approach were improvements over not using any of them at all, with the thumbnail approach being the better of the two. While this paper focused on linear videos, a similar implementation could be made for non-linear videos.

To improve the performance of applications and websites, designed for large screens, on smaller screens Gutwin et al. [7] have explored the use of three different techniques. These techniques are Two-level zoom, Fisheye and Panning. They found that both the Two-level zoom and Fisheye techniques were effective in improving the manoeuvrability of websites on smaller screens.

Other research on non-linear video streaming have focused on the technical side of the problem. For example Krishnamoorthi et al. [1] have explored using HAS for delivering non-linear videos. They created a basic video player that focused on proving the feasibility of it, with policies for prefetching of upcoming branches that allows seamless playback.

Meixner et al. [15] have done work on an easy to use tool for the creation and management of non-linear videos, called SIVA Producer. They used an iterative process of testing for improving the usability of their tool.
3 System design and high-level frameworks

In order to answer our research questions we have implemented a number of different video players. These video players consists of multiple different components: a DASH video, a video element that can play that video, playback bars that display the branched video structure and a way for user to interact with that player and make choices as they watch. The main focus of the first implementations has been the look and feel of the playback bar.

3.1 Video for DASH

The video *Big Buck Bunny*\(^1\) was chosen as the video that would be used throughout the study. This video was not designed for branched video but it was the only easily available and free video we could find. The video is also available in multiple different sizes and bit rates. The one we picked was at size 1920x1080, had 30 frames per second and was encoded with H.264 (MPEG-4 AVC). We used ffmpeg\(^2\) to reduce the video from 10 to 6 minutes to save encoding time at the later stages.

The video was encoded at 4 different qualities: 1080p, 900p, 720p and 480p. This was done with the program x264\(^3\), which both splits the video into multiple qualities and makes sure they are properly encoded. This process generates raw H.264 files. To make them more manageable we used MP4Box\(^4\) to encapsulate each raw file in a MP4 container. Lastly MP4Box was used with the 4 MP4 files to split them into multiple segments. We gave each quality a unique id and put the segment length to 4 seconds for each quality. The program automatically generates an MPD file that can be used directly in dash.js.

3.2 DASH Player

The video player we created is based on the dash.js framework. The framework does not support branched videos by default and there is currently no standard for formatting the structure of a branched video. For these reasons we had to make changes to the source code of dash.js as well as create our own format for these videos.

---

1. Available at https://peach.blender.org/download/
2. Available at https://www.ffmpeg.org/
3. Available at https://www.videolan.org/developers/x264.html
4. Available at https://gpac.wp.imt.fr/mp4box/
3.2. DASH Player

A single video was used to display the branched video. This means that we have to map a linear video to a non-linear branched video. The video was split into parts of 32 seconds, where one of these parts would represent a single branch. The length of these parts is mostly arbitrary, except for it being a multiple of 4 as a result of the segment length mentioned above. This mapping is shown in Figure 3.1.

Since the video player was written in JavaScript it was natural to use JSON as the format for the branched video structure. This structure is shown in Figure 3.2. We chose a normalised structure with a list of objects instead of a tree structure since it is easier to iterate over the normalised structure. A normalised structure also does not limit us to simple tree structures, allowing structures that connect branches after they have split. The resulting JSON object is saved in a meta file and helps the video player know when branches start and stop.
From this branch file we determine all relevant information about the branched video, for instance what choices the user has, when the player should present the choices and what branch is currently playing.

Because we know what branch is being played, we can also determine how far away we are from a branch point and which branches are coming up. This information can be used to prefetch all related branches when nearing a branch point. dash.js does not have a built in way to load a specific part of a video so this has to be implemented manually. This was done by intercepting all requests sent by dash.js and making additional requests when we are approaching a branch point.

3.3 Playback bar

The playback bars were written in JavaScript using a canvas element for drawing the graphics. The bar was placed directly under the video. We have created classes for the visual objects, the branches, and the branch points, to simplify manipulation of these objects. These objects can be drawn in a regular way as well as with the fisheye effect.

We wanted the playback bars to look and feel familiar to the users, by introducing as few new things as possible. This allows the user to focus on what is important; the differences in the design implementations. We looked at existing playback bars for linear videos, from YouTube and Netflix, and tried to replicate them as much possible. The bar itself has 3 states, empty, buffered and played, which are represented by filling the bar with a certain colour. The colours were inspired by popular streaming sites and consists of the following colour scheme: an empty buffer is a dark grey colour, a buffered bar is a light grey and the played one is red.

A total of 3 types of playback bars have been implemented. The first one is called the full view playback bar and has no distortion, zoom effects nor translations. The full view implementation shares most similarities with linear video examples, with the playback bar being static and playback progression moving along bars. A sketch of this playback bar is shown in Figure 3.3. This implementation is used as a basis for the other playback bars.

5A geometric transformation that moves every object in a vector space in a given direction
Zoom and follow

The second playback bar is called *zoom and follow* and like the name suggests, it is a combination of scaling everything and following the current playback progression. The transformations needed for this already exists in the canvas interface, making it fairly simply to create. An example of this implementation can be seen in Figure 3.4.

Fisheye effect

The last implementation is made using a fisheye effect. This works in similar fashion to the second bar in the sense that the playback bar follows the current playback progression. The fisheye effect takes place slightly ahead of the current playback progression with the goal of expanding the upcoming branches.

This effect was implemented with an external package called *fisheye*[^6]. This package calculates the fisheye transformation of an *xy*-point according to a centre point. The transforms returns a new point as well as a scale factor. By moving the centre of the fisheye we can transform the coordinate system and amplify objects close to the centre. In our implementation, the fisheye centre point is placed ahead of the currently played position. This type of playback bar can seen in Figure 3.5.

[^6]: Available at https://github.com/chtefi/fisheye
3.4 Branch selection

The choices for each branch are represented by buttons placed in a box to the right of the video. These choices are presented 5 seconds before the user reaches a branch point. If the user makes the choice before the branch point the buttons will disappear and the selected branch is played when said branch point is reached. If the user decides to not make a choice in time, the video will stop at the upcoming branch point until a choice is made.

3.5 Video player

By combining all these components, we get a video player that can handle branched videos. The video player with the full view playback bar is shown in Figure 3.6. In this instance two choices are presented in the box displayed to the right of the video. At the bottom we can see that we have hit the first branch point as well as the buffer conditions of the upcoming branches.
4 Preliminary user study

4.1 Usability tests

In order to evaluate our implementations we have performed a number of usability tests. We wanted to keep the collected data quantitative and structured to make it easier to draw well founded conclusions. To accomplish this, a list of tasks was created for each implementation for the users to follow. The users were instructed to read through the tasks before starting the tests. These tasks direct the user to use the video player in a certain way and make choices depending on the buffer conditions. This is to test if the user properly understands the playback conditions.

The collection of data was done by using a SUS form which the user was instructed to fill out after each test. The moderator of the test filled out an observer form while utilising the concurrent thinking out loud method. In the observer form the moderator noted whether the user successfully completed a task and any other observations the moderator made. These observations could be, for example, if the moderator noticed that the user got frustrated when trying to complete a task.

From the SUS form and observer form we can extract information about how the users experienced each implementation. We can determine if the SUS results differ from the results of the observer form. The SUS results produce a score for each user and each implementation which is a value between 0 and 100. The higher the score the better the implementation was received in term of usability.

4.2 Results

A total of 11 usability tests were performed where each subject tested all three implementations in the following order: full view, zoom and follow and fisheye. This order was chosen as each implementation is based on the previous one, creating a logical progression for the users. Each test contained 2 tasks for each implementation, resulting in a total of 22 tasks being performed for each implementation.
4.2. Results

**Full view**

The full view implementation received a mean SUS score of 70.5. The implementation had a standard deviation of 13.7. The distribution of the SUS scores can be seen in Figure 4.1. According to the observations, the users had no problem understanding the playback bar. Out of the 11 users, 2 did not successfully complete all tasks for this implementation. When the choices were presented, 3 users clicked on the playback bar and tried to select a branch that way, even though the user were informed of where the choices would be located on screen.

![Figure 4.1: Distribution of the SUS-scores for the full view implementation](image1)

**Zoom and follow**

The zoom and follow implementation had a mean SUS score of 76.8 and a standard deviation of 10.4. The distribution of the scores can be seen in Figure 4.2. From the observations, we found that most users thought this implementation was easy to understand. One of the users failed to complete all tasks for this implementation. This user was one of the users that failed to complete the tasks of the full view implementation. According to 4 of the users, the buffer was clearer in this implementation in comparison to the full view implementation.

![Figure 4.2: Distribution of the SUS-scores for the zoom and follow implementation](image2)
4.3 Summary of pre-study results

Fisheye

Lastly the fisheye implementation had a mean SUS score of 66.8 and a standard deviation of 19.0. The score distribution can be seen in Figure 4.3.

During the test, 4 of the users stated that they thought the fisheye was an interesting and “cool” effect. Out of the 11 users, 5 stated that the distortion introduced with the fisheye lens was distracting or confusing. All users successfully completed all tasks for this test.

4.3 Summary of pre-study results

A summary of the SUS forms can be seen in Table 4.1. The zoom and follow implementation had the highest mean SUS-score, followed by the full view implementation and, lastly the fisheye implementation. In Table 4.1 we can also see the task failure rate for each implementation. This is the number of tasks out of all tasks related to an implementation that users failed to perform.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Mean score</th>
<th>Standard deviation</th>
<th>Task failure rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full view</td>
<td>70.5</td>
<td>13.7</td>
<td>3 out of 22</td>
</tr>
<tr>
<td>Zoom and follow</td>
<td>76.8</td>
<td>10.4</td>
<td>1 out of 22</td>
</tr>
<tr>
<td>Fisheye</td>
<td>66.8</td>
<td>19.0</td>
<td>0 out of 22</td>
</tr>
</tbody>
</table>

Table 4.1: Standard deviation and Mean score of all implementation

Some users found the colours of the branch choice buttons to be confusing. They said that they associated green buttons with “good” and naturally wanted to click those. They also did not want to press the red button even if the task instructed them to, because they associated red with “bad”. According to 4 users, the straight branches made it hard to see what was going on when the playback bar split into three branches since the branches were overlapping right after the branch point. Two users said that they found the playback bar to be quite clunky as it was a separated from the video itself.
From the results we wanted to improve the usability of our existing implementations. In this chapter we present solutions and improvements for the different components of the video player. The ideas presented here can be used in future studies.

5.1 Arbitrary functions for branch shapes

One improvement for future studies is to more distinctly separate the branches in the playback bar right after a branch point. In the implementations made for the pre-study, all branches were straight lines which would cause branches to briefly overlap. A simple way of solving this is to use a different function for the lines representing the branches. We rewrote our implementation so that the branches can take an arbitrary function that it follows between two points.

The start and end points for the branch are calculated beforehand so the function needs to be properly scaled to go through these points. We can wrap each function \( f \) inside a scaled function as follows:

\[
f_{\text{scaled}}(x) = \frac{y_1 - y_0}{f(x_1)} f(x),
\]

where \( y_1 - y_0 \) is the height difference of the branch’s start and end points. The denominator \( f(x_1) \) is the \( y \)-value of the end point of the unscaled function. An example of how these variables relate can be seen in Figure 5.1. These functions can then be mirrored around the \( x \)-axis to get the upper and lower parts of a branch split.
5.2 Playback bar transformations

From our usability scores we found that the zoom and follow transformation scored the highest as well as having the lowest deviation, which implies that users found it to be the best in our tests. The results for the implementation with a fisheye effect tells us that it was the most controversial in our test. It had the lowest mean score and the highest deviation, nearly twice that of the zoom and follow implementation. While some users liked the effect many thought it was distracting and that it was taking focus away from the video. The implementation with no transforms had a fairly high score and was the second best implementation. It also had a low deviation making it a good potential candidate for future development.

For future work we recommend focusing on using either no transformations or zoom and follow transformations for the reasons above. Although the implementation using fisheye transformations did not score as high as the others, there are a lot of variables that can drastically affect how the bar is transformed. Therefore it would still be interesting to further research the use of fisheye transformations and we will discuss this further below.

Fisheye

In our study we only tried one implementation using a fisheye effect. One could probably come up with multiple ways of implementing the effect with several parameters that could be fine tuned. Therefore it is hard to completely write it off for further research. One example of things to explore in further research is the position for the centre of the lens. We would like to see the following be explored:

- Centring the lens on the mouse cursor.

Figure 5.1: Example of scaling of the arctangent function to pass through two points.

Instead of using a linear function we would recommend using a function that results in a quick separation of the branches. Examples of functions that follow this criteria are: the arctangent function, the square root, the error function, the hyperbolic tangent function or a shifted logarithmic function that passes the origin.
5.3 Management of complex branch structures

- Centring the lens on the upcoming branch point.

Gutwin et al. [6] describe three different ways of implementing a fisheye effect, and the one used in this thesis is only one of them. In future work it would be interesting to investigate if the other two implementations perform better. Finally there are some parameters that can be tweaked to further improve the user experience such as:

- Lens distortion
- Lens radius
- Lens centre offset

The lens centre offset is only relevant for implementations where the lens is following the playback progress.

5.3 Management of complex branch structures

Videos that have complex branching structures could be difficult to properly visualise, as drawing to many branches can confuse users. Large structures can also be hard to fit into the compact space of the playback bar. A simple solution to this is to only draw what is currently relevant to the user instead of drawing the complete structure. This can be done by only drawing what is currently being played, what has been played, as well as the upcoming branches. To further reduce unnecessary information we can limit how many upcoming branches we render in terms of depth.

5.4 Playback bar position

Another problem that needs to be better explored is the position of the playback bar. In the original implementations the bar was placed right beneath the video and we got feedback that it was not well integrated with the rest of the video player. Since the bar is quite big compared bars for linear videos, it is hard to place it in a way that feels natural and is not distracting. A recommendation would be to place the playback bar inside the lower part of the video element making it more similar to other playback bars. One could then hide the playback bar if the user is not currently interacting with the video player.

5.5 Branch choices

The branch choices got the same feedback as the position of the playback bar. They should be better integrated with the video player itself instead of being in a separate box beside the video. Here there are a multiple of different solutions that could be implemented. Some users wanted to click on the playback bar itself like they would with a linear video so there might be a good place to put the choices. Netflix uses another approach and solves this by displaying the choices as big buttons on the screen when the branch point are reached. The connection between a button and a branch in the playback bar might not always be obvious. This could be solved by either matching the colour or matching the text.

Matching colours

Something that we have not explored is to match the background colours of the branches in the playback bar to their respective branch selection buttons. This is worth exploring as the buttons in our first implementation had arbitrary colours. These buttons confused some users resulting in them making the incorrect choice as they thought the colour served a purpose.
5.6 Example implementation

Matching text
An alternative to the solution above is to add a letter to both the branch in the playback bar and to the respective button. Like the solution discussed above this could help with the clarity of branch selection by making a connection between the buttons and the branches in the playback bar, but it might also just add unnecessary visual noise.

Highlight playback bar
A solution similar to the one with matching colours, is to highlight the related branch in the playback bar when hovering over a button. This can be implemented by simply changing the background colour of the highlighted branch.

Clickable playback bar
To improve the intuitiveness of playback bar even further, you can make the branches in the playback bar clickable as well as highlighting them when you hover over them. This allows the user to click the branch he or she wants to watch. This solution would most likely have to be used with the large buttons placed on top of the video mentioned above, as the clickable branches do not convey that a choice has to be made.

5.6 Example implementation
Using the recommendations above multiple different video players can be implemented and evaluated. One such implementation can be seen in Figure 5.2. This particular example has both the branch choices and a transparent playback bar placed on top of the video. The branches are using an arc-tangent function and there is no other scaling or transformation made to the playback bar.

![Figure 5.2: A refined user interface showing two choices on top of the video.](image)

Another implementation can be seen in Figure 5.3. Here the branches are again transformed using an arc-tangent function with no other transformations or scaling. The branch selection is done through the transparent buttons that appear on the playback bar.
5.6. Example implementation

Figure 5.3: A refined user interface showing two choices on top of the playback bar.

The implementation in Figure 5.4 uses the logarithmic function to transform the branches in the playback bar and the zoom and follow transformation to only focus on a specific part of the playback timeline. The branch selection is made through buttons placed on top of the playback bar. This implementation was mainly to show the zoom and follow transformations in a more refined design, as the implementation in the pre-study that used these transformations scored the highest.

Figure 5.4: A refined user interface showing two choices on top of a playback bar that implements zoom and follow and uses a logarithmic function for the branches.

The last example implementation can be seen in Figure 5.5. This implementation uses large on screen buttons that highlight the corresponding branch in the playback bar. The branches are clickable, they follow the arc-tangent function and the playback bar is using the zoom and follow transformation. To make the connection between a button and its corresponding branch even clearer, they are matched using letters.

Figure 5.5: A refined user interface showing a branch being highlighted as a result of hovering over the corresponding button.
5.7 Metafile creation tool

Everything mentioned above is aimed at improving the user experience for the viewer of branch videos. Another aspect of this topic is the creation of the branched videos. To simplify this process we have developed a tool for creating the metafiles for the branched videos as described in Figure 5.2. This tool has the following features:

- A live graphical view of current branch structure
- Toggable fisheye effect that follows the mouse cursor
- Load in existing metafiles for alteration
- Create and delete branches
- Give the branches relevant names for users
- Change the start time and end time for each branch
- Download the current structure as a metafile

This tool was created in JavaScript and uses the same package for the fisheye effect as the video player. But unlike the video player, the tool uses Scalable Vector Graphics (SVG) instead of a canvas element for displaying the branch tree. The tool can be seen below in Figure 5.6 To further expand on this tool, one could implement a thumbnail preview for the video segments. Another thing that could be interesting is the ability to upload multiple videos and have the tool automatically generate a branched video that can be used by a video player.

![Figure 5.6: The tool used for creating and editing the branch metafiles](image-url)

1Available at https://github.com/chtefi/fisheye
6 Discussion

6.1 Pre-study results

One interesting result from the pre-study was the task failure rate. It was lowered with each test; the first implementation had the highest failure rate and the last implementation had the lowest. This, in contrast to the last implementation having a lower SUS score than the first, seems rather contradictory. Although the failure to complete tasks were concentrated to two users meaning that the number of failed tests might not be representative of the system. A potential explanation for this contradiction could be that users get more comfortable with the system over time, resulting in them having an easier time identifying strengths and weaknesses of the latter implementations. The small sample size of our usability tests could play a significant role in skewing certain results.

6.2 System design

We saw that the fisheye implementation had the worst average SUS-score while also having the highest deviation in score. While this result is interesting it does not have to mean that a fisheye effect itself is a bad choice for the playback bar, it just means that our implementation failed at clearly conveying information. While we do not believe that fisheye is the right direction to go according to the results of our pre-study, there might still be potential in the effect.

The same thing goes with zoom and follow implementation that also has multiple variables that can be altered, for instance how much zoom is applied, the offset distance and in what way the offset is calculated. Since the feedback for this implementation was mostly positive there is less incentive to make major changes to the setup that we had created for our initial tests.

Previous work explored and found fisheye to be useful under certain conditions [7], however that study focuses mainly on smaller screens and mobile devices. Our playback bar has similar issues with sizing and fitting a lot of information in a small area, but the scenarios are still quite different since our playback bar and usability tests were based on wide computer screens. As a large part of internet traffic is generated by mobile devices, it would be interesting to create a responsive design that works well on both large computer screens and TVs, as well as on the smaller screens on mobile devices.
6.3 Pre-study method and usability tests

Techniques

The *thinking aloud* technique used during the tests might not have been the best option as some users struggled to consistently verbalise their thoughts resulting in us not getting some of the data we wanted. This is a known disadvantage of the *thinking aloud* technique [8]. One example of this is during the tests that prompted the user to select the branch with the largest buffer. Most users completed this task but many of them did not express why they selected the branch they did. As a result, we could not be completely sure if these users understood the playback bar or if they simply made a correct guess.

This could have been solved through utilising a *verbal probing* technique or perhaps a hybrid of it and a *thinking aloud* technique. This would allow the moderator of the test to ask specific questions to make sure that they get answered.

Number and order of tests

As mentioned in the discussion about the results from the pre-study the task failure rate lowered with each test. This is probably a result of the user getting more familiar with the system and this could have been counteracted through performing the tests in a random order. Another way to prevent familiarity with the system from affecting the results could be through having each user only test two of the implementations and increase the number of participants. This could also reduce any potential fatigue experienced by the user when performing multiple similar tests in a row.

Choice of video

The video chosen for the usability tests is a commonly used test video that is not well suited for branched video. The resulting story of the video did not make a lot of sense as, when choices were made, the video would jump to a seemingly arbitrary position. It would be interesting to test with a video specifically designed for branched video and would probably make the viewer more invested in the test.

Branching structure

The branching structure chosen was a simple symmetric tree with a depth of 3. We did not compare how different structures could affect the clarity of the playback bar shown. There are also structures that connect branches, meaning the tree can shrink in terms of width. While we did not explore this subject, it would be interesting for future work to test and compare the different possibilities of video structures.

6.4 The work in a wider context

Branched videos have a wide range of usage, for instance educational purposes. This could be used to train employees in different scenarios to build an intuition for what decisions they should make in a real situation. It could also be used in schools to create an interactive learning experience for students.

While there is a lot of positive aspects of branched video, problems could occur related to user integrity. One example is tracking the choices that users make. The resulting information could be used to more effectively prefetch branches depending on their popularity. While this might improve user experience, it will come at the cost of integrity. Since a branched video presents the viewer with choices, it naturally produces more information about the user than a linear video would. This information could be used to identify user preferences, that later could be used to give specific ads to the user. One example could be a video that makes the...
user choose between two brands, such as Coca-Cola and Pepsi Cola. This information could then be used to display directed ads from the brand chosen by the user. Eventually the service provider could have a lot of collected information about the user that can be harmful if it gets into the wrong hands.
In this paper we wanted to explore and test possibilities for video players for branched videos. The aim was to construct and test different setups for the components of a branched video player in order to increase the clarity of what is shown. The research questions involved different ways of displaying playback bars and branch choices. The end goal was to find a video player that feels intuitive and clear for end users. More specifically we wanted to test two different types of playback bar transformations: one zoom in version and one for a fisheye effect. From the pre-study we concluded that zooming in on the current position in the playback bar appeared to improve the understanding of the playback conditions while the fisheye effect seemed to mostly confuse the users.

From these results and other feedback received, we created a new set of implementations for the video player with more general solutions for the shapes of the branch segments as well as the position and shapes of branch selection. We believe that these video players are a direct improvement on the implementations made for the pre-study. Our hope is that our implementations and or the ideas that we present in this thesis will come to use in future work.


