DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Development of a Sophisticated Session Recording Exporter for the BigBlueButton Web Conferencing System

Daniel Petri Rocha

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Development of a Sophisticated Session Recording Exporter for the BigBlueButton Web Conferencing System

Entwicklung eines fortgeschrittenen Tools zum Export von Sitzungsaufzeichnungen im BigBlueButton Konferenzsystem

Author:	Daniel Petri Rocha
Supervisor:	Prof. Dr.-Ing. Jörg Ott
Advisors:	Dipl.-Inf. (Univ.) Martin Uhl
	Christian Menges, B.Sc.
	Fabian Sauter, B.Sc.
	Sebastian Kappes, B.Sc.
Submission Date:	September 15th, 2021

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, September 15th, 2021 Daniel Petri Rocha

Acknowledgments

I am thankful to Prof. Dr.-Ing. Jörg Ott for giving me the opportunity of writing a Bachelor's thesis at the Chair of Connected Mobility[1] from the Department of Informatics at the Technical University of Munich.

Likewise, I am grateful for the time Dipl.-Inf. Martin Uhl, Christian Menges, B.Sc., Fabian Sauter, B.Sc., and Sebastian Kappes, B. Sc., have committed weekly to the implementation aspects of this 4-month long undertaking. Their attentive involvement provided constructive feedback of paramount importance, without which the contributions made to BigBlueButton could not have been realized.

At last, I am beholden to the BigBlueButton community for being so welcoming throughout all development phases. Their support and warm embrace provided crucial guidance to a newcomer to their open-source project.

[1] `https://www.in.tum.de/cm/home/`

Abstract

BigBlueButton is an open-source web conferencing system designed for remote teaching. A longstanding feature request by community members has been to give users the ability to download the recording of a meeting as a video file, including the presentation, webcams, and chat messages. This thesis implemented that enhancement as a script executed automatically by the server once a recorded conference ends, performing significantly faster than previous solutions capturing the screen of the web player in real-time. Additionally, annotated slides can be output as PDF. Processing the presentation into both formats can also be done client-side if the user has no access to the server. Development of the converter did not introduce new dependencies into BigBlueButton and resulted in contributions to a Ruby library providing a data structure for efficient queries on intervals and FFmpeg, a command-line multimedia framework. The code offers a foundation for other improvements on BigBlueButton's roadmap, such as giving instructors the option to import work done on documents in breakout rooms into the main session.

Zusammenfassung

BigBlueButton ist ein quelloffenes online Konferenzsystem, dass für den Fernunterricht entwickelt wurde. Ein lang ersehnter Wunsch der Nutzer war es die Möglichkeit zu haben, die Aufzeichnung einer Sitzung inklusive der Präsentation, geteilten Webcams und Chatnachrichten als Videodatei herunterladen zu können. In dieser Arbeit wurde diese Erweiterung in Form eines Skripts implementiert, welches automatisch von einem Server ausgeführt wird, sobald eine aufgezeichnete Konferenz beendet wurde. Dies geschieht wesentlich schneller als eine Echtzeitbildschirmaufnahme des Webplayers welche zuvor als Ersatz verwendet wurde. Zusätzlich können mit Notizen versehene Folien als PDF erhalten bleiben. Die Verarbeitung der Präsentation in beiden Formaten kann zusätzlich lokal erfolgen sofern der Nutzer keinen Zugang zum Server hat. Die Entwicklung des Konverters führte keine neuen Abhängigkeiten in BigBlueButton ein und resultierte zum einen in Beiträgen zu einer Ruby-Bibliothek, die eine Datenstruktur für effiziente Abfragen von Intervallen bereitstellt, und zum anderen in FFmpeg, bei welchem es sich um ein kommandozeilenbasiertes Multimedia-Framework handelt. Der Code bietet eine Grundlage für zukünftig geplante Verbesserungen von BigBlueButton, wie beispielsweise die Möglichkeit für Lehrkräfte, die in Nebenräumen erarbeiteten Dokumente in die Hauptsitzung zu importieren.

Contents

1 The BigBlueButton Web Conferencing System

1.1 Introduction

BigBlueButton (BBB) is an open-source initiative that provides a virtual classroom environment for online learning, giving teachers a platform on which students can be engaged remotely for distance education. BBB has been among the videoconferencing tools adopted by the Department of Informatics at the Technical University of Munich (TUM) since the academic summer semester of 2020, when the COVID-19 pandemic moved lectures and tutor classes online indefinitely.

A critical shortcoming of BBB turned out to be its limited options to export recorded sessions held on the platform, with the most commented-on feature request on BBB's GitHub repository being about enabling users to download recorded meetings as a single file [Dix12]. Playback of a recorded conference occurs on a web player that can be accessed once a recorded meeting ends.

This bachelor's thesis aims to implement the export of recordings into BBB's source code as a server-side process that renders a unique video containing all assets shown in the playback interface. The exporter's usage and implementation details are documented throughout this work to explain the rationale behind design decisions while simultaneously acting as a concrete contributing demonstration for new BBB developers.

1.2 Functionality supported by the web player

Most actions taken by conference participants will show up in BBB's web player during playback of a recorded meeting, the only exception being when users embed an external video player in the presentation area from sources such as YouTube, Vimeo, and Twitch. All other events are captured by the server and stored in a format that allows the presentation to be recreated within the browser using the no longer maintained Popcorn.js[1] framework. As a result, playback is not entirely supported across different browsers and devices.

The module realizing the player[2] is not necessarily tied to the server since an independent folder can be created containing the files a recording requires. Understanding how that data is stored and structured is a crucial aspect for the exporter's implementation;

[1] `https://github.com/mozilla/popcorn-js`

[2] `https://github.com/bigbluebutton/bbb-playback`

for now, it suffices to be aware that the following functionalities need to be supported by the project to resemble BBB's playback interface truthfully:

Annotations Users may add scribbles, circles, lines, rectangles, and triangles in varying thicknesses and colors on top of any presentation slide.

Breakout rooms Moderators can send participants into breakout rooms, during which an active recording continues taping the main room.

Captions Instructors can manually type the closed captions of what other participants are saying while the conference is still ongoing.

Chat Users may send public messages at any point in time.

Cursor The presenter's mouse pointer may appear on top of the slides in the whiteboard area during the entirety of the session.

Polls The presenter can quiz the audience and choose to display the results publicly. Response types include giving participants a text box to type their answer in or presenting multiple options from which one can be chosen.

Screen shares Multiple users may be sharing their screens simultaneously. Often called "deskshare" among BBB developers.

Shared notes Users can anonymously write formatted text into a shared document that all other attendees can see and edit. Only the final state of these notes can be retrieved from the web player; there is no timing information associated with the input. Converting the pad into different formats is already possible and will not be regarded henceforth.

Slides A presenter can upload documents in the Office, Portable Document Format (PDF), or text file format, and images in the Joint Photographic Experts Group (JPEG) or Portable Network Graphics (PNG) format, on which selected users can add annotations concurrently.

Text Alongside annotations, text boxes can be added to the whiteboard containing text in different sizes and colors.

Webcams Participants of the meeting can choose to share their microphones and webcams throughout the call, as long as the session's moderator allowed them to.

Zooms The current presenter can enlarge a section of a slide and move it around. Internally called a "panzoom."

1.3 Existing approaches

Due to the demand for an export feature, several scripts which attempt to offer the functionality have been listed on the issue's page [Dix12]. A common workaround is to overlay the only two video files provided by BBB — the webcam and deskshare videos — using the Command-Line Interface (CLI) tool FFmpeg[3] to combine the media tracks. System administrators can obtain that solution by installing the `bbb-playback-screenshare` package[4], but troubleshooting is necessary to run it in the newest BBB 2.3 release [Dix21a].

Though that approach might suffice in specific use cases, it leaves out key parts of a meeting, such as slides shown on the whiteboard area. The solutions capable of handling uploaded presentation files rarely support annotations that users may have drawn on the whiteboard itself.

A notable exception is BBB-Render[5], which is actively being developed and achieves a single video by constructing a GStreamer Editing Services (GES)[6] project through a collection of Python scripts. GES is a library that allows timeline-based editing of media streams in a non-linear fashion, serving as a basis for video editors such as Pitivi[7]. It follows that BBB-Render's output can be visually customized before rendering starts, though it is meant for client-side use and the chat, captions, and panzooms are not supported. As of 07/2021, a Pull Request (PR) that makes text visible on the whiteboard is waiting to be merged.

As such, the most suitable option for client-side users remains playing the recording back in real-time while performing a screen capture locally, despite the fact that partially working exporters exist. The requirements to run them, e.g., BBB-Render expecting a Ubuntu 20.04 system to convert a recording in addition to the installation of numerous dependencies, creates an unrealistic entry barrier that no meeting participant can be reasonably expected to overcome on their own. Hence, a server-side script integrated into BBB that automatically performs the necessary work after the end of a session is the approach chosen for this thesis to make the file publicly available for download in BBB's Graphical User Interface (GUI), Greenlight[8].

In an email exchange with Pedro Marin, one of BBB's core committers, he pointed an existing project run by community members out called BBB Video Download,[9] one of the few server-side scripts that renders many of the recording's visible assets. Compared with the built-in player, however, it has an output resolution of only 800×600. It lacks both the chat and the polls, two areas BBB deems essential for student engagement and for which new features are already on the official roadmap. The video is in part

[3] `https://ffmpeg.org/`
[4] See the *screenshare* workflow in Section 2.1
[5] `https://github.com/plugorgau/bbb-render`
[6] `https://gstreamer.freedesktop.org/`
[7] `https://www.pitivi.org/`
[8] `https://github.com/bigbluebutton/greenlight`
[9] `https://github.com/tilmanmoser/bbb-video-download`

assembled by having a headless browser run in the background while taking screenshots; other server-side scripts such as BBB-Recorder[7] use a similar method to perform a complete screen recording by opening a tab with Google Chrome and capturing it with Puppetcam.[8] Even though they output a video resembling the original, that approach takes at least as long as the recording itself in addition to the time BBB took to process them in the first place.

1.4 Requirements

The previously mentioned programs and their methods form the basis of the exporter developed throughout this project. Its elicited requirements follow.

1.4.1 Functional requirements

- A single video file is returned, whose container format is widely used.
- All applicable multimedia assets previously outlined in Section 1.2 are supported.
- Integration into BBB's 2.3 source code.
- The conversion process begins as soon as BBB finishes processing the required files.
- Users can download the video through BBB's interface.

1.4.2 Non-functional requirements

- Converting a video is much faster than performing a screen recording.
- No significant new dependencies are introduced into BBB.
- The exporter is backward-compatible BBB's 2.2 version.
- The script relies on open-source software exclusively.

1.4.3 Additional features

- Annotated slides can be downloaded as PDFs.
- Chapter marks are part of the video's metadata for easier navigation.
- Individual meetings can be re-rendered.
- The exporter offers rendering parameters to control the output quality and resource usage.

[7] `https://github.com/jibon57/bbb-recorder`
[8] `https://github.com/muralikg/puppetcam`

2 Theoretical Background

2.1 Recording phases

The reasoning behind storing the media files separately to assemble a recording from the stored data later is that it allows BBB to be easily extended with new workflows [Dixb]. Each workflow is a different way of replaying and consuming the media; the *presentation* workflow, for instance, is included out-of-the-box and responsible for the current state of recordings. At the same time, *podcast* only yields the meeting's audio track, and *screenshare* returns the webcams overlaid onto the deskshare video. These have to be separately installed and enabled server-side.

Workflows are implemented with Ruby 2.5 scripts that take a recording's data as input and generate the desired format as output. They address the steps of processing and publishing a recording, two of six phases which recordings undergo internally:

1. Capture 2. Archive 3. Sanity 4. Process 5. Publish 6. Playback

The first phase stores the data streams and events emitted by BBB's modules throughout a session on the server. Chat messages, cursor movements and whiteboard annotations are stored in Redis[1], a key-value database. Deskshare and webcam videos are captured with Red5[2]; the meeting's audio with FreeSWITCH[3] [Dixb]. During *Archive*, the raw files are placed in a separate folder and marked for deletion if an instructor did not choose to record a meeting. However, a system administrator can opt to rebuild the whole recording regardless [Dixb]. *Sanity* consists of scripts that certify the data's integrity to ensure the upcoming stages can run without encountering any issues. Processing then occurs on a per workflow basis, combining the media to achieve the desired result depending on the metadata collected in the events. At last, produced files are published by moving them to a folder Nginx [Dixa] allows unrestricted access to, from which playback can begin.

The *Archive*, *Process*, and *Publish* stages pass through *post* phases that run after the respective step completes, e.g., a post-script that sends out a notification once a new recording is available for playback. Since the exporter needs to mimic the behavior of the web player closely, it will be implemented as an extension to the presentation workflow, i.e., a post-publish script that works with the same playback files. Introducing an entirely new workflow instead would require parsing the raw data again and hinder the coordination between the web player and the exporter as new features come in.

[1] `https://github.com/redis/redis`
[2] `https://www.red5pro.com/open-source/`
[3] `https://freeswitch.com/`

```
[
{"locale": "en", "localeName": "English"},
{"locale": "pt-BR", "localeName": "portugu\u00eas (Brasil)"},
{"locale": "de", "localeName": "Deutsch"}
]
```

Listing 2.1: Example `captions.json` file

```
WEBVTT

00:00:10.644 --> 00:00:17.144
Instructors can download live captions in different formats during

00:00:17.144 --> 00:00:22.255
a meeting, such as plain text, Microsoft Word, PDF, ODF or HTML.
```

Listing 2.2: Example English subtitles in `captions_en.vtt`

2.2 Encoding of the processed files

The following items outline the scheme the processed and published files are encoded in, specifically for the presentation workflow. How data is stored will be leveraged during the implementation period to realize the exporter. Examples used are based on a close to nine-hour-long BBB tutorial held by the instructors of a mandatory bachelor's course at TUM. The exercise session was chosen as a reference for being a real-life use case of BBB being utilized in teaching environments. Its length ensures that the tutors and students had enough opportunities to use most of the functionality offered by BBB during the meeting.

2.2.1 Captions

The closed-captioning feature is still "very limited," [Ber21] seeing that no automatic transcription is available. A stenographer can select different languages to write subtitles in, which are stored in a file called `captions.json`. For a meeting captioned in English, Brazilian Portuguese, and German, Listing 2.1 illustrates how data is encoded.

For each locale, a file `caption_<locale_name>.vtt` is created in the Web Video Text Tracks (WebVTT)[4] format similar to the one in Listing 2.2.

2.2.2 Chat

The chat is kept in an Extensible Markup Language (XML) file called `slides_new.xml`, which has a root element called `popcorn`. It has `chattimeline` elements as child nodes,

[4] `https://www.w3.org/TR/webvtt1/`

```
<?xml version="1.0"?>
<popcorn>
  <chattimeline in="60" direction="down" name="Alice" message=
  "The Tweedback session is still closed, right?" target="chat"/>
  <chattimeline in="65" direction="down" name="Bob"
    message="It's working for me :)" target="chat"/>
  <chattimeline in="470" direction="down" name="Charlie" message=
    "&lt;a href=&quot;https://tweedback.de/pq4t&quot; rel=&quot;
    nofollow&quot;&gt;&lt;u&gt;https://tweedback.de/pq4t&lt;
    /u&gt;&lt;/a&gt;" target="chat"/>
</popcorn>
```

Listing 2.3: Example chat messages in `slides_new.xml`

which each have attributes storing the timestamp in seconds a message is first shown (`in`), the movement made by the chat window during playback (`direction`), the username of the message's author including its contents (`name`, `message`), and which part of the playback GUI it appears on (`target`).

In the example from Listing 2.3, Alice asked a question a minute into the session and got a reply five seconds later. Since the attribute `in` has a timestep of one second, `chattimeline` elements are given the same timestamp if multiple users send messages within that timeframe. As a result, it suffices to render the chat with a single frame per second when fewer messages come in at once than what can fit in the chat area.

Charlie shared a hyperlink that is saved alongside surrounding Hypertext Markup Language (HTML) markup, which participants can open to see the externally referenced page. Since a video player does not support such functionality, the exporter will need to strip unsafe tags to display their inner text.

Seeing that chat messages always move in the same direction and have the same target, the exporter can disregard these attributes.

2.2.3 Cursor

The cursor is rendered as a solid red circle over the presentation slides. Its data is stored in `cursor.xml`, an instance of which can be found in Listing 2.4.

The root element holds a series of events as children, which in turn store where the cursor is at each given timestamp. A new entry is only created when the cursor's position changes, having a timestep of 0.1 seconds. To animate the cursor, ten frames per second (FPS) are therefore enough.

The cursor's position is described in relation to the width and height of the whiteboard's visible area. Frequently that corresponds to the dimensions of the slide the cursor is on, but the possible change in these parameters caused by zooming needs to be considered too.

Assuming dimensions of 1600×900 at the 4.2-second mark, the cursor in the XML

```
<?xml version="1.0"?>
<recording id="cursor_events">
  <event timestamp="0.0">
    <cursor>-1.0 -1.0</cursor>
  </event>
  <event timestamp="4.2">
    <cursor>0.1 0.5</cursor>
  </event>
</recording>
```

Listing 2.4: Example `cursor.xml` file

```
<?xml version="1.0"?>
<recording id="panzoom_events">
  <event timestamp="0.0">
    <viewBox>-0.0 -0.0 1600.0 900.0</viewBox>
  </event>
  <event timestamp="10.0">
    <viewBox>110 120 1280.0 720.0</viewBox>
  </event>
</recording>
```

Listing 2.5: Example `panzooms.xml` file

fragment from Listing 2.4 is located at (160, 450). The coordinates can be obtained by multiplying the vector components found in the inner text of the cursor element by the size of the rectangle describing the visible area at the given timestamp, which is saved in the panzooms file described in this chapter. Negative coordinates are returned when the cursor is not being shown.

2.2.4 Panzooms

As exemplified in Listing 2.5, the `panzooms.xml` document is structured in a similar way to the cursor events and chat files:

The `viewBox` elements contain a four-dimensional vector describing the zoom and pan of a slide for a given timestamp in the recording. A new node is added when a change occurs. The entries in the list correspond to the following properties [MDN21c], respectively:

1. `min-x` 2. `min-y` 3. `width` 4. `height`

The first two entries define the starting coordinates of a rectangle whose upper-left-corner is set at (`min-x`, `min-y`). From this new origin, a rectangular area is spanned given the width and height dimensions. The region constrained by the rectangle is what the user sees.

Presume a slide is being shown that is 1600 pixels wide and 900 pixels high. Given Listing 2.5, the slide would be entirely visible during the first 10 seconds of a recording. A user then scales a region, reducing the displayed area to a box whose sides are 1280×720 — keeping the same aspect ratio as before. The box is additionally shifted along the `x` and `y`-axis by a few hundred pixels.

The viewBox attribute is defined in the Scalable Vector Graphics (SVG) XML namespace and used by the whiteboard's slides to realize panzooms during playback in the browser. The transformed viewport is also taken into account by the cursor.

2.2.5 Video files

A `presentation.yml` file in BBB's core configures the dimensions, format, and framerate the output video has during playback. Per default on BBB 2.2 and 2.3, videos are rendered into the WebM format due to its permissive license[5]; MP4 is offered as an alternative.

Deskshare

The deskshare video file is only created for meetings in which participants share their screens. It is as long as the recording itself and displays a blank white screen when no deskshare takes place. Since the whiteboard stops being shown when a screen share begins and vice-versa, the exporter can play the deskshare video under the slides at all times without having to take any kinds of interval data into account. It is rendered into a size of 1280×720 pixels by default at 5 FPS.

Webcams

The generated file has a default width in pixels of 640, a height of 480, and a framerate of 15 FPS. In a similar fashion to the deskshare video, the file's length corresponds to the duration of the recording. The published file contains the captured audio and video from all participants of the session. If multiple people are sharing their webcams simultaneously, these are arranged in a grid-like pattern. Only a white background is shown with the audio track playing over it when all cameras are turned off.

2.2.6 Whiteboard

The whiteboard's data is stored in a document called `shapes.svg` that Popcorn.js uses to create a timeline-based project. In contrast to what the file extension suggests, the contents are not well-formed SVG due to additional attributes the video and media library requires. When opened in a browser or other viewing tools, the invalid markup outside of SVG's namespace and document type definition is ignored. Still, no graphics appear since all media types have been purposively made invisible. The general idea is

[5] `https://www.webmproject.org/license/software/`

```
<g class="canvas" id="canvas1" image="image1" display="none">
  <g id="image1-draw1" class="shape" timestamp="0.0" undo="-1"
  shape="image1-pencil1" style="stroke:#ff0000;stroke-linecap:round;
  stroke-linejoin:round;stroke-width:2;visibility:hidden;fill:none">
    <path d="M94.51219 457.14286L91.72473 459.93032L91.72473"/>
  </g>
  <g id="image1-draw2" class="shape" timestamp="0.3" undo="-1"
  shape="image1-pencil2" style="...">
    <circle cx="200" cy="550" r="1.4"/>
  </g>
</g>
```

Listing 2.6: Example `shapes.svg` strokes

to deduce which assets should be visible given a timestamp and only display that subset of the `shapes` file. Rendering a succession of sorted timecodes in ascending order then visually recreates the animations on the whiteboard when played back at the correct speed.

Annotations

Annotations are either strokes on the whiteboard or geometric shapes. For each presentation slide, a group container element is created to act as a canvas that stores the sketches. The `g` element of the canvas class has an `id` and references a slide in the `image` attribute. Its children are further group elements from which the actual SVG shapes inherit their attributes [MDN21a], such as style and timing information. Since annotations can be undone, the `undo` attribute is either set to -1 or the last timestamp the shape was visible in; its first appearance is kept in `timestamp`. The shapes themselves are valid SVG 1.1 elements describing circles, lines, rectangles, triangles, and paths. Path elements, for instance, combine multiple curves and straight lines [MDN21b] to recreate the natural flow found in hand-written notes.

In BBB 2.3, annotations are only shown to participants once the shape has been fully drawn. The partial drawings leading up to the full one used to be shared and stored in earlier versions but were removed due to the increase in resource consumption. The example from Listing 2.7 illustrates this change, showing the path's data when a pencil is used to underline a phrase.

In BBB 2.2, the stroke's path keeps being expanded with more information, duplicating the data the previous frame needed until the same final path is reached. Strokes belonging to the same pencil can be identified with the `shape` attribute. Even though this variant has a considerable overhead cost, at least an option to use it may be introduced again in the future since a live whiteboard feature is especially desired [Sug21b] in classroom settings. As a result, the exporter needs to be able to handle both approaches.

```
<!-- BBB 2.2 -->
<g id="..." class="..." timestamp="0.0" undo="-1" shape="..." style="...">
  <path d="M 0 0 L 289 3"/>
</g>

<g ... timestamp="0.1" ...>
  <path d="M 0 0 L 289 3 L 631 9"/>
</g>

<!-- Only this node is included in BBB 2.3 recordings -->
<g ... timestamp="0.2" ... >
  <path d="M 0 0 L 289 3 L 631 9 L 995 3"/>
</g>
```

Listing 2.7: Comparison between BBB 2.2 and 2.3 annotations

```
<g id="image1-draw1" class="shape" timestamp="11.9"
  undo="47.1" shape="image1-poll1" style="visibility:hidden">
    <image width="544.0" height="119.88" x="1056.0" y="780.12"
      xlink:href="presentation/.../poll_result1.svg"/>
</g>
```

Listing 2.8: Example `shapes.svg` poll fragment

Polls

Polls are part of a slide's canvas and therefore share the same attributes as an annotation does. The image displaying the results is loaded externally through SVG's `image` element and an `xlink:href` reference. The figure appears in the lower right corner of the current slide and scales according to the panzoom.

Slides

The slides shown when a presentation is replayed are individual PNG images that have been extracted from the original input file, e.g., PDF or PowerPoint presentations. They are always resized to 1600×1600 while keeping the original aspect ratio, meaning either the width or height of the resulting image will be 1600 pixels long.

Just as is the case with the polls, a reference is responsible for showing the picture in the background, which is placed at the coordinate's system origin. Unlike the canvas elements that only store the timestamp they first appear in, slides have two attributes — `in` and `out` — to define the interval they were shown in. BBB stores the textual content of the slide in a file to enable searching for specific keywords of a recording and find the corresponding position in it. This file's path is found in the text attribute.

```
<image id="image1" class="slide" in="0.0" out="335.4"
  xlink:href="presentation/.../slide-1.png" width="1600" height="1200"
  x="0" y="0" style="visibility:hidden"
  text="presentation/.../textfiles/slide-1.txt"/>
```

Listing 2.9: Example slide reference in `shapes.svg`

```
<g ... shape="image1-text1"
  style="color:#ff0000;word-wrap:break-word;visibility:hidden;
  font-family:Arial;font-size:16px">
<switch>
  <foreignObject width="480.0" height="180.0" x="300" y="400">
    <p xmlns="http://www.w3.org/1999/xhtml" style="margin:0;padding:0">
      Hello World!
    </p>
  </foreignObject>
</switch>
</g>
```

Listing 2.10: Example `shapes.svg` canvas text

Text

When participants of a meeting add text to the whiteboard, Extensible HyperText Markup Language (XHTML) elements are added to the whiteboard's canvas instead of rendering them as pure SVG. This is due to the fact that SVG 1.1 is not capable of automatically wrapping text given a bounding box, while XHTML is. SVG Tiny 1.2 does so as well with the `textArea` element [Neu+08], but its support status across mobile and desktop platforms is severely limited [Sch08]. Likewise, SVG 2 plans on introducing automatic line breaks [Wil+18a], but the specification has not yet left the recommendation stage [Wil+18b] and still awaits to be implemented by any major browser.

BBB gives users a rectangular content area to type in, such as 480×180 in the example below. In order to embed the XHTML namespace and constrain the size of that field, the `switch` and `foreignObject`[6] tags are used. The font color, family, and size are passed in the style attribute.

[6] `https://developer.mozilla.org/en-US/docs/Web/SVG/Element/foreignObject`

3 Implementation

The exporter's development occurred on a public personal repository[1] before opening a PR on BBB's GitHub page. The coming sections focus on the server-side version of the script, structured in the order the code is executed in to exemplify a sample run of the code. Client-side implementation details follow, later mentioning how the script was modified to enable annotated slide exports and the changes made to BBB's GUI to add download buttons.

To avoid introducing new major dependencies, the exporter relies on BBB's usage of FFmpeg to render a single MP4 file — the format having been chosen for its widespread use and compatibility. SVG rasterization is supported in the FFmpeg version used by BBB due to its compilation having taken place with the librsvg[2] codec enabled. Since FFmpeg is not a video editor, but instead a CLI tool to convert and mix media streams into different formats, a central part of the exporter's implementation revolves around generating suitable streams for FFmpeg from the published presentation data.

3.1 Post-publish script example file

BBB provides a post-publish script template[3] that contains boilerplate code for new additions. The framework expands relative paths after obtaining the meeting parameters and loading required modules, serving as the basis of the exporting script as shown in Listing 3.1.

A logger object is created to write information related to the script's execution in BBB's log directory, such as which presentation is being converted, how long it ran for, and possible error messages. Its contents are set to clear weekly. CLI arguments are parsed with Trollop[4], a Ruby gem: the meeting ID is passed with the `-m` flag and used to reference the published presentation files; BBB's record and process worker invokes the script with the playback format name passed in `-f`. An `@` is prefixed to the declaration of `published_files`, changing its scope to all of the exporter's methods as a Ruby instance variable.

[1] `https://github.com/danielpetri1/bbb-recording-exporter`
[2] `https://gitlab.gnome.org/GNOME/librsvg`
[3] `https://github.com/bigbluebutton/bigbluebutton/blob/develop/record-and-playback/core/scripts/post_publish/post_publish.rb.example`
[4] `https://rubygems.org/gems/trollop/versions/2.9.10`

```
require File.expand_path('../../../lib/recordandplayback', __FILE__)

opts = Trollop.options do
    opt :meeting_id, "Meeting id to archive", type: String
    opt :format, "Playback format name", type: String
end

meeting_id = opts[:meeting_id]

logger = Logger.new("/var/log/bigbluebutton/post_publish.log", 'weekly')
logger.level = Logger::INFO
BigBlueButton.logger = logger
BigBlueButton.logger.info("Started exporting presentation for [#{meeting_id}]")

@published_files = "/var/bigbluebutton/published/presentation/#{meeting_id}"
```

Listing 3.1: Preamble of the exporter derived from BBB's template

3.2 Constants and switches

A series of values control the exporter's layout, output resolution, and resource usage. By default, the output video size is set to 1920×1080 in the `OUTPUT_WIDTH` and `OUTPUT_HEIGHT` constants. Given these dimensions, exported components are dynamically resized to fit the window, keeping the desired aspect ratio. The processed webcam video is scaled down to 320×240 and placed in the upper left corner. Since the chat is as wide as the webcam video, and the messages occupy the remaining height, `CHAT_WIDTH` equals `WEBCAMS_WIDTH` and `CHAT_HEIGHT` is the difference between `OUTPUT_HEIGHT` and `WEBCAMS_HEIGHT`.

All chat messages are rendered onto an SVG canvas that is cropped according to the current timestamp in the presentation. `CHAT_CANVAS_WIDTH` and `CHAT_CANVAS_HEIGHT` contain these dimensions, which are close to the maximum 8032×32767 supported by librsvg[5]. Due to SVG's previously discussed text limitations, a monospaced font was chosen to ease calculation of where line breaks should happen, given that each character then always has exact dimensions. `CHAT_FONT_SIZE` is instanciated with 15, standing for the height in pixels each character is tall. Through trial and error, the relationship between the width and height of Ubuntu's 18.04 default monospaced font (Ubuntu Mono) on which BBB runs was determined to be 3:5. `CHAT_FONT_SIZE_X`, a character's width, is therefore 0.6 times the font size. Though the exact width can be determined with `rmagick`[6] or `ttfunk`[7], these tools were refrained from to avoid introducing new dependencies.

The area reserved for the whiteboard is shared with the deskshare, which is scaled to

[5] This limitation arises from Cairo, the rendering engine used by librsvg.
[6] `https://github.com/rmagick/rmagick`
[7] `https://github.com/prawnpdf/ttfunk`

```
1 DESKSHARE_Y_OFFSET = ((SLIDES_HEIGHT - ([SLIDES_WIDTH.to_f /
    ↪ DESKSHARE_INPUT_WIDTH, SLIDES_HEIGHT.to_f / DESKSHARE_INPUT_HEIGHT].min
    ↪ * DESKSHARE_INPUT_HEIGHT)) / 2).to_i
```

Listing 3.2: Centering the deskshare video in the exporter's whiteboard area

fill the whiteboard area whilst maintaining the input aspect ratio. To center the video stream accordingly, `DESKSHARE_Y_OFFSET` is calculated taking `DESKSHARE_INPUT_WIDTH`, `DESKSHARE_INPUT_HEIGHT`, and the whiteboard dimensions where slides are shown into account. Assuming the deskshare has the default size of 1280×720, the procedure in Listing 3.2 yields an offset of 90 pixels from the top part of the video once it was scaled to fit the resulting 1600×1080 whiteboard area. This positioning of the components is summarized as an illustration in Figure 3.1, closely resembling the web player's layout.

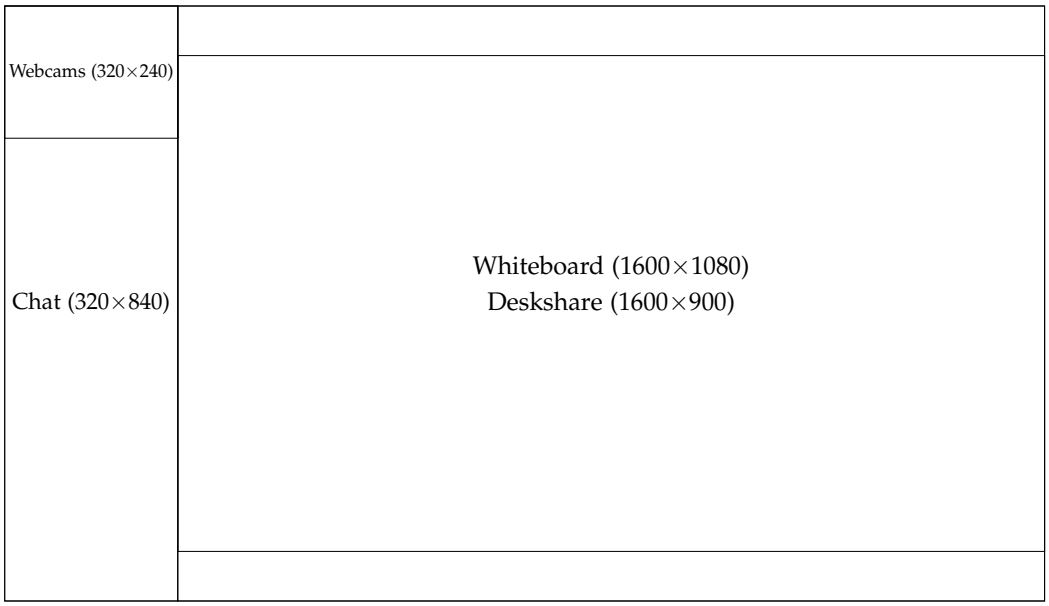

Figure 3.1: Default output video layout

BBB servers that support a live whiteboard (default on version 2.2) need to enable `REMOVE_REDUNDANT_SHAPES` to correctly show SVG annotations, due to the differences in stroke encoding across versions. Since SVG files are made up of plain text, they can be losslessly compressed with GNU zip (GZIP). The resulting SVGZ file is supported by librsvg. The exporter has a `SVGZ_COMPRESSION` switch that, when set to true, will ensure the script only writes compressed image data on disk. The number of threads used to export the video can be limited in `THREADS`, considering community members asked for a way to contain FFmpeg's resource consumption. Similarly, the quality and file size of the output can be controlled with the Constant Rate Factor (CRF) parameter, as laid out in FFmpeg's documentation [FFM21]. At the end of an encode, the time taken and the

maximum amount of memory consumed can be displayed by setting `BENCHMARK_FFMPEG` to true.

FFmpeg is not capable of rendering embedded background images in each slide since librsvg requires a base Uniform Resource Identifier (URI) when utilizing the file URI scheme due to security concerns, a parameter FFmpeg does not support as of August 2021. A working alternative is the data URI scheme, which encodes the referenced file in the Base64 format. A patch[8] adding the `base_uri` parameter to FFmpeg's librsvg decoder was submitted and awaits merging. When `FFMPEG_REFERENCE_SUPPORT` is enabled, the exporter will assume FFmpeg has been recompiled with the patch applied and skip the Base64 conversion step, which can become quite resource-intensive. Likewise, `CAPTION_SUPPORT` can be switched on when the `movtext` codec is present in FFmpeg for subtitles.

3.3 Client-side export

Since the exporter works with the published presentation files, all assets can be publicly accessed over the internet without having root permissions on a BBB server. A download script is provided on a separate branch which uses the recording's hostname and meeting ID to obtain the files. The script itself is identical to the server-side exporter, except for the header, log keeping, and CLI arguments demonstrated in Listing 3.1. The client-side script needs to be executed in the same directory the downloaded files are available in, since the `@published_files` path is set to that folder.

3.4 Exporting the presentation

To render a video from presentation data, the exporter will keep track of every change that occurred in the whiteboard, to then create an SVG frame containing what is displayed at the timestamp the change occurred. Changes to the whiteboard include adding annotations, zooming in and out, switching slides, and starting a deskshare. Cursor movements will be treated separately since it does not affect the whiteboard's contents, even though the pointer is rendered on top of it. Registering a change for every mouse movement would result in a massive amount of exported frames in which the whiteboard stayed the same; creating frames for changes in the whiteboard only instead ensures no resources were consumed unnecessarily. The chat and webcam are separate components to the whiteboard as well. Temporarily generated files are stored in scratch directories that are deleted as soon as the exporter finishes its execution.

[8] `https://ffmpeg.org/pipermail/ffmpeg-devel/2021-August/282985.html`

3.4.1 Whiteboard shape conversion

The exporter begins by parsing the `shapes.svg` document using the Nokogiri[9] gem to convert whiteboard assets into a format compatible with FFmpeg. All shapes and annotations are selected using an XML Path Language (XPATH) query to change their visibility attribute from hidden to visible. If the element is a poll, it contains an external reference to another SVG file embedded in the whiteboard that is loaded either if a base URI has been passed, or the linked image is in the Base64 plaintext format [GNOb]. A helper method performs the conversion to Base64 if FFmpeg does not support the base parameter. When a text node contains XHTML text, its contents need to be recreated using SVG elements. Font size, color, and positional attributes are stored along the text box's dimensions to reconstruct an approximate copy, akin to what is done to the chat. Generated SVG text elements are normalized to conform to the standards imposed by the World Wide Web Consortium (W3C) on Unicode and sanitized with the Loofah[10] gem by stripping it of potentially unsafe tags; both Nokogiri and Loofah were already in use by BBB.

3.4.2 Parsing whiteboard timestamps

Changes made to `shapes.svg` are temporarily saved in a separate `shapes_modified.svg` file to avoid interfering with the web player. Parsing the modified file returns the shapes, slides, and timing information required to rebuild the whole presentation as a video. One pass of the document suffices, so Nokogiri's XML Reader can be used since it is significantly faster than using XPATH queries. The XML reader receives the modified file and returns nodes that can be traversed iteratively in an `each` loop [Nok21]. Timestamps of when slides entered and left the whiteboard are stored in an array declared as `timestamps` and in a `WhiteboardSlide` struct containing either the data of or a reference to the background image and its dimensions.

Unlike slides, shapes store timing information in the `timestamp` and `undo` attributes. Determining the intervals in which the shape was shown is therefore not as trivial. When going back to a previous page in the displayed whiteboard document, the annotations are reinserted as they had been last left, making the timings appear off. If a user goes back to a slide in the 10th minute of a meeting in which the annotations were added during the first, the shape will keep its original attributes (e.g., `timestamp="60.0"`, `undo="-1"`) even though the actual playback time is at the 10-minute mark. The `undo` attribute complicates things further, as annotations that were already deleted are present regardless in the whiteboard's canvas, and drawings can be removed far into the future due to the trash bin-button in the GUI that removes all annotations the user pressing the icon has made. For a given shape, its actual interval information can therefore be derived as follows:

[9] `https://github.com/sparklemotion/nokogiri`
[10] `https://github.com/flavorjones/loofah`

```
1 shape_undo = slide_out if shape_undo.negative?
2
3 shape_enter = [shape_timestamp, slide_in].max
4 shape_leave = [[shape_undo, slide_in].max, slide_out].min
```

Listing 3.3: Determining a shape's interval start (`shape_enter`) and end (`shape_leave`)

- When `undo` is -1, the shape is not deleted from the whiteboard. As a result, it is displayed for as long as the slide it lies on.
- The shape's starting timestamp is either already correct or was already present when displaying the current slide. The maximum of both values is the annotation's starting timestamp.
- The shape stops being displayed when the slide changes or undo is pressed. An undo before the slide begins means the shape can be discarded.

Listing 3.4.2 contains the Ruby implementation of the verbose description above. That approach is similar to BBB-Render's code. However, they perform a redundant minimum operation by setting the shape's initial timestamp to `[[[shape_timestamp, slide_in].max, slide_out].min`, which is not required since `shape_timestamp` and `slide_in` are always equal or less than `slide_out`. In addition to the interval data, each element in the `shapes` array contains the drawing's SVG markup.

Panzooms stored in `panzooms.xml` are parsed in a similar fashion. The `parse_panzooms` method is called with the `timestamps` array passed as an argument, once it has been returned by the previous method. To this array further instances of whiteboard movements will be appended, i.e, changes to the `viewBox` parameter of the slides. A `panzooms` queue stores the timestamp of the movement alongside the SVG `viewBox` parameter. Once all changes to the whiteboard have been parsed, the duration of the meeting obtained from the metadata is included as the final whiteboard timestamp. The exporter at this point has all the information it needs to render individual whiteboard frames.

3.4.3 Rendering the chat

By parsing `slides_new.xml` with Nokogiri's XML reader, the username behind every chat message is kept with its contents and timing information in an array. The timecode next to the message is included in the `HH:MM:SS` format, as is done on the web player. Messages and usernames are scrubbed with the Loofah gem to strip it of HTML tags, that are present when users send external links. An example of this can be seen in the last message sent in Listing 2.3: the markup from the hyperlink should not appear in the chat. After sanitization, only the markup's inner text is left. To comply with W3C's validation service[11], the string is normalized to a canonical form[12]. Both these steps in

[11] `https://validator.w3.org/`

[12] `https://apidock.com/ruby/v2_5_5/String/unicode_normalize`

```
60.0 crop@c x 0, crop@c y 60;
65.0 crop@c x 0, crop@c y 105;
470.0 crop@c x 0, crop@c y 150;
```

Listing 3.4: Example FFmpeg commands to crop the chat canvas

combination ensure the chat does not break when dealing with Universal Transformation Format (UTF)-8 encoded text, which was verified by performing tests with W3C's sample plain-text file[13]. Languages written from left to the right are supported as well, with the timecode in that case coming before the username instead.

The text messages are then placed on a SVG canvas that is built with the XML Builder[14] gem, which proved to be faster than Nokogiri's markup builder. Due to the usage of a monospaced font, the width of the chat box will always be able to hold a fixed amount of characters. Messages are broken up into individual lines, with line breaks being inserted at whitespace characters if the next few words do not fit the chat's width. By multiplying the amount of lines with the font's height in pixels, the total message height is obtained. Once the messages reach the bottom of the canvas, the next ones are printed with an offset of `CHAT_WIDTH`, beginning a new chat column on the canvas. For a seamless transition, the last few messages are duplicated at the top. The coordinates of the messages are written in a text file (`chat_timestamps`) that is used to send commands to FFmpeg, which then crops the canvas appropriately to create an illusion of movement at a given timestamp.

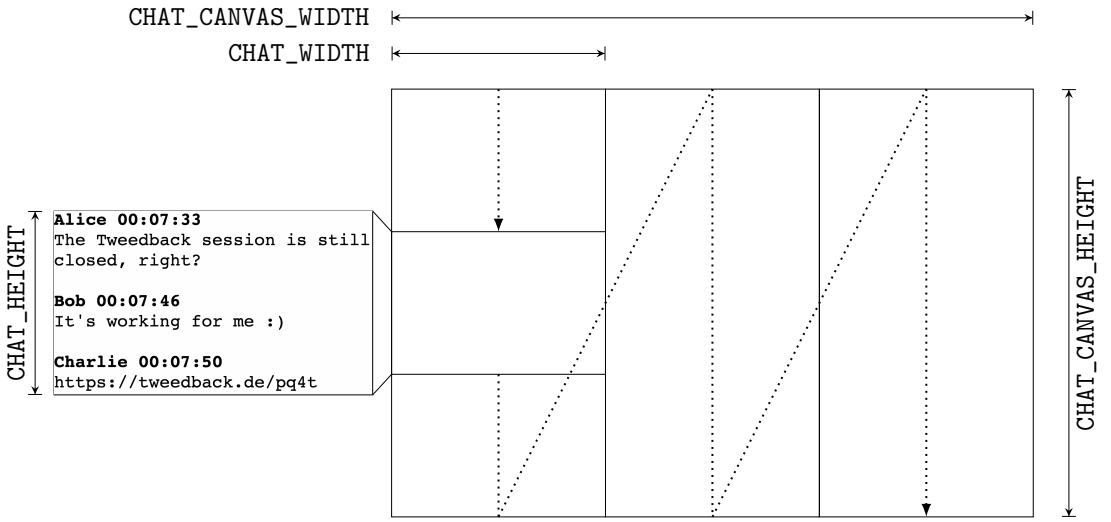

Figure 3.2: Example chat from Listing 2.3 rendered onto the canvas

[13] https://www.w3.org/2001/06/utf-8-test/UTF-8-demo.html
[14] https://github.com/jimweirich/builder

```ruby
builder = Builder::XmlMarkup.new

builder.svg(width: "16", height: "16") do
    builder.circle(cx: "8", cy: "8", r: "8", fill: "red")
end

File.open("#{@published_files}/cursor/cursor.svg", "w", 0o600) do |svg|
    svg.write(builder.target!)
end
```

Listing 3.5: Ruby code from which the mouse pointer SVG is built

```svg
<svg width="16" height="16">
    <circle cx="8" cy="8" r="8" fill="red"/>
</svg>
```

Listing 3.6: Generated mouse pointer SVG

3.4.4 Rendering the cursor

The `render_cursor` method is called with two arguments: the parsed panzoom data, and a Nokogiri XML Reader object to parse the `cursor.xml` file. The mouse pointer itself is created from scratch using the XML Builder gem. It produces an SVG file containing the markup for the red dot in Figure 3.3 which will be used as a cursor on the whiteboard.

Figure 3.3: Mouse pointer rendered with FFmpeg

The cursor's coordinates in `cursor.xml` are relative to the current's slide size and view box. As a result, they need to be transformed to correspond to the actual coordinates on the slide, taking into account how the slide itself was scaled to fit the exporter's layout. The exporter uses W3C's algorithm outlined in natural language in SVG's 2 specification to compute the equivalent transform of the SVG viewport [Wil+18c].

The cursor's radius is subtracted from the calculated `x` and `y` coordinates since FFmpeg will place the top-left corner of the image on the specified coordinate, resulting in a small yet visible offset from the desired location. The final, centered coordinates are saved as commands which get sent to FFmpeg that contain the timestamp and the position of the cursor. FFmpeg overlays the cursor's SVG image onto the slide for each entry in the `cursor_timestamps` file, leaving the cursor stationary in that location until the timestamp of the next command is reached. Negative timestamps are instances in which the cursor is not visible on the whiteboard.

```
0.0 overlay@m x 1602.464, overlay@m y 515.719;
0.3 overlay@m x 1583.616, overlay@m y 532.063;
0.7 overlay@m x -1288.0, overlay@m y -818.0;
1.1 overlay@m x 1428.992, overlay@m y 519.49;
1.4 overlay@m x 1484.304, overlay@m y 552.169;
```

Listing 3.7: Example commands sent to FFmpeg with the mouse pointer's position

3.4.5 Rendering the whiteboard

The `render_whiteboard` method receives the parsed `panzooms`, `slides`, `shapes`, and `timestamps` arrays as arguments. Elements from the `timestamps` array are then sorted and duplicates removed, resulting in a timeline of events in which changes on the whiteboard occurred. Figure 3.4 shows a video timeline instance. A frame is created for each interval present in the timeline, which when played back in order produce an animation of the whiteboard over time.

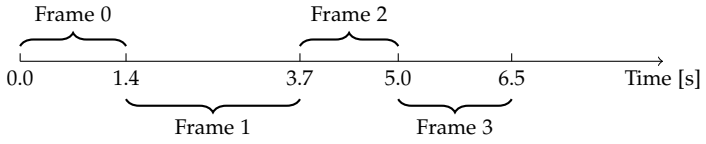

Figure 3.4: Example whiteboard timeline intervals

Since the `panzooms` and `slides` array are in the correct order, they can be thought of as queues from which the first element is popped when the interval's start corresponds to the timestamp of the zoom or slide change. The background image or viewbox parameter is kept until the next element in the queue has an initial timestamp greater or equal to the current timestamp in the video timeline. Shapes and whiteboard annotations, however, can overlap, and thus require a more elaborate approach. In the example shown in Figure 3.5, a blue shape (such as a circle) is displayed from the time `t = 1.4` to `t = 5.0`, whilst another green one is shown from `3.7` to `6.5`. Both ranges overlap in the interval from 3.7 to 5.0; the third frame in the exported video will therefore be rendered with the SVG data of both shapes. The capability of obtaining the overlapping shapes given a timestamp is provided by the interval tree data structure, discussed in the next section.

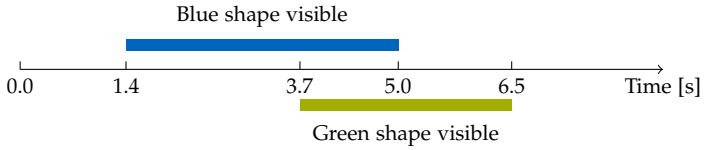

Figure 3.5: Example overlapping whiteboard shapes

As illustrated in Section 2.7, BBB handles annotations differently depending on the

```ruby
def remove_adjacent(array)
    index = 0

    until array[index + 1].nil?
        array[index] = nil if array[index].id == array[index + 1].id
        index += 1
    end

    array.compact! || array
end
```

Listing 3.8: `remove_adjacent` method, invoked when `REMOVE_REDUNDANT_SHAPES` is set

version. When the whiteboard is capable of displaying stroke animations, sections of the final path have the same shape ID. The `remove_adjacent` method uses the fact that segments of the same path come after another in the `shapes.svg` file to retain only the last state of each stroke with an in-place algorithm listed in Listing 3.8. Each shape is compared with its successor, if their IDs are equal, the current node is set to `nil` to remove redundant shapes. A copy of the modified array is returned without the `nil` values using Ruby's `compact` method[15]; the total runtime of `remove_adjacent` is therefore linear due to the two passes through the array.

3.4.6 The interval tree data structure

BBB-Render uses a Python module providing an interval tree to efficiently query visible drawings at each timestamp in the recording. It uses $\mathcal{O}(n)$ storage, takes $\mathcal{O}(n\ log(n))$ time to build, and returns queries in $\mathcal{O}(log(n) + k)$, where k is the number of intervals reported [Ber08a].

Ruby gems providing the same functionality, such as *itree*[16] and *interval-tree*[17], did not fulfill the exporter's requirements. The former uses an AVL tree for dynamic insertion of nodes, while the latter is implemented as a static binary tree. Even though both approaches are faster than the brute-force approach of checking each timestamp against all intervals in the array, taking $\mathcal{O}(n)$ time per query, *itree* turned out not to be capable of returning duplicate intervals even when the nodes held different data.

When a slide is shown that already has drawings on it, it may be the case that two or more shapes are displayed for the same period of time, even though they contain distinct SVG markup. *itree*, however, reports only one of the annotations. In addition, future maintenance of the repository seems unlikely due to inactivity since 2013. Though *interval-tree* has an option to return multiple matches, its implementation was slow and missed a corner case. Additionally, the gem did not augment the tree to be able to store

[15] `https://apidock.com/ruby/v1_9_3_392/Array/compact`
[16] `https://github.com/hoxworth/itree`
[17] `https://github.com/greensync/interval-tree`

custom data alongside each interval.

Due to these problems, the exporter initially resorted to the trivial — yet inefficient — method of selecting the right whiteboard annotations. Fortunately, a PR[18] was made implementing the support for custom objects, turning the issue with the duplicate intervals into the only problem to solve. A PR was made as part of this thesis to the *interval-tree* repository, implementing the search function iteratively to prevent stack overflows from occuring and considering the passed `unique` parameter which was previously implemented incorrectly. While both submissions await approval, the gem was temporarily added into the existing `lib` folder of BBB recording scripts which contains, among others, a custom hash function for Nokogiri documents and the workers initiating each recording phase.

Segment trees are comparable data structures also capable of reporting intervals containing a point. Their advantage comes when generalized to higher-dimensional objects, such as segments arbitrarily orientated on a plane or when the amount of returned elements is of interest but not the objects themselves. For the exporter which utilizes 1-dimensional segments, i.e., intervals, usage of an interval tree is more reasonable since a segment tree consumes $\mathcal{O}(n\ log(n))$ space [Ber08b].

3.4.7 Exporting whiteboard frames

Frames are rendered by the `svg_export` function, which builds an `SVG` frame with the XML builder from the shapes obtained through the interval tree, the slide's viewbox parameter, and its dimensions. If the `SVGZ_COMPRESSION` flag is enabled, the frame is GZIPped into SVGZ. Frames have the fixed size specified in the `SLIDES_WIDTH` and `SLIDES_HEIGHT` constants, which correspond to the whiteboard area of the exporter.

Uniform scaling of the frames is done in this step instead of later in FFmpeg due to an issue in which frames whose width and height differ from the first SVG input are dropped, since the streams from all files need to share the same properties [FFmb]. The frame's background image is scaled to maximize the viewing area of the whiteboard whilst still maintaining the aspect ratio. This behavior differs from the web player, which has a static viewport corresponding to the slide's original dimensions. When zooming into a slide that has the A4 paper format, for instance, playback in the web player will maintain the window's shape throughout, while the exported video fills the whiteboard area with the slide. Another contrasting implementation aspect is the fact that the mouse pointer is scaled alongside panzooms in the presentation playback, a characteristic regarded as a bug by the exporter.

The whiteboard frames are loaded sequentially into FFmpeg using the concatenation demuxer, which makes it possible to create a video file from a set of images and their respective durations [FFm17]. The exporter creates a temporary `whiteboard_timestamps` text file containing the path to the whiteboard's frames and the interval's length it should be displayed for. Listing 3.9 illustrates the format used. In effect, the whiteboard video

[18] `https://github.com/greensync/interval-tree/pull/13`

```
1 file ../frames/frame0.svg
2 duration 2.1
3 file ../frames/frame1.svg
4 duration 0.1
5 file ../frames/frame2.svg
6 duration 0.1
7 file ../frames/frame3.svg
8 duration 0.3
```

Listing 3.9: Example FFmpeg `whiteboard_timestamps` for the whiteboard slideshow

is a slideshow in which the framerate varies.

3.4.8 Rendering the video

The meeting is rendered by a single FFmpeg instruction that uses the generated assets and commands as input, multiplexing the audio, video, and image streams into a single file in the MP4 container format. Only the meeting's chapter marks and captions are added retroactively. FFmpeg is invoked using Ruby's `system` method, which is called with an FFmpeg instruction that varies depending on whether a deskshare video took place during the meeting. This section will examine the case where a screen share occurred, as the FFmpeg command without the additional data stream follows analogously.

The first argument — with index 0 — passed to `ffmpeg` on line 3 in the command line is a solid white color stream from the `lavfi` filter acting as the exporter's background. The dimensions of the generated background frame define the output size of the exported video. Some developers in the BBB community have reported replacing the backdrop with an image to suit the needs of their customers, e.g., displaying their brand through a watermark in the resulting video file.

The whiteboard is loaded on line 4 by reading the exported frames as a slideshow from a similar file to the one shown in Listing 3.9. Since the paths in the file are relative, `-safe 0` is passed as an argument to enable such URIs. Similarly, the `base-uri` is set in case FFmpeg supports the patch implemented in Section 3.6. Cursor and chat SVGs are loaded with their framerates set to 10 and 1 respectively and looped continuously throughout the video, allowing them to receive commands describing their respective movements on screen.

As discussed in Section 2.2.5, the processed video file of the webcams has the length of the meeting itself and is present even under the circumstance in which no attendee turned their cameras on. on line 8, the deskshare video is loaded, suitably setting the file extension either to the MP4 or WebM format. These five separate input streams will be moved, cropped, scaled, and overlaid onto each other starting from line 10 to recreate the events of the recorded meeting through a complex filtergraph[19].

[19]https://trac.ffmpeg.org/wiki/FilteringGuide

```
ffmpeg

-f lavfi -i color=c=white:s=#{OUTPUT_WIDTH}x#{OUTPUT_HEIGHT}
-f concat -safe 0 #{BASE_URI} -i #{@published_files}/timestamps/whiteboard_timestamps
-framerate 10 -loop 1 -i #{@published_files}/cursor/cursor.svg
-framerate 1 -loop 1 -i #{@published_files}/chats/chat.svg
-i #{@published_files}/video/webcams.#{VIDEO_EXTENSION}
-i #{@published_files}/deskshare/deskshare.#{VIDEO_EXTENSION}

-filter_complex
   '[2]sendcmd=f=#{@published_files}/timestamps/cursor_timestamps[cursor];
    [3]sendcmd=f=#{@published_files}/timestamps/chat_timestamps,
        crop@c=w=#{CHAT_WIDTH}:h=#{CHAT_HEIGHT}:x=0:y=0[chat];
    [4]scale=w=#{WEBCAMS_WIDTH}:h=#{WEBCAMS_HEIGHT}[webcams];
    [5]scale=w=#{SLIDES_WIDTH}:h=#{SLIDES_HEIGHT}:force_original_aspect_ratio=1[deskshare];
    [0][deskshare]overlay=x=#{WEBCAMS_WIDTH}:y=#{DESKSHARE_Y_OFFSET}[screenshare];
    [screenshare][1]overlay=x=#{WEBCAMS_WIDTH}[slides];
    [slides][cursor]overlay@m[whiteboard];
    [whiteboard][chat]overlay=y=#{WEBCAMS_HEIGHT}[chats];
    [chats][webcams]overlay'

-c:a aac
-crf #{CONSTANT_RATE_FACTOR}
-shortest
-t #{duration}
-threads #{THREADS}
-metadata title='#{meeting_name}'
#{BENCHMARK}
#{@published_files}/meeting-tmp.mp4
```

Listing 3.10: FFmpeg command used to export the recorded meeting with deskshare

Cursor and chat commands need to be differentiated, which is done by adding `@m` to the overlay commands sent to the mouse pointer and `@c` for the chat. The overlay command places the cursor on the desired position at a given timestamp, while the chat canvas is cropped to show the messages displayed at a given timestamp. The streams receiving those commands are labeled as `cursor` and `chat`, representing individual layers the exporter uses to create the output video. Webcam and deskshare videos are scaled to fit the exporter's layout and positioned per Figure 3.1.

At last, the components are overlaid. The deskshare, webcam videos, and chat layers are placed over the background image, as illustrated in Figure 3.6. The whiteboard frames are placed over the centered deskshare video: once a screen share begins, the frame shown on the whiteboard is transparent, revealing the deskshare underneath. The cursor composes the final upper layer, being bounded by the whiteboard's dimensions.

Audio from the webcams and deskshare video is re-encoded with the Advanced Audio

Coding (AAC) codec suited for MP4 files on line 22 to ensure compability with the Opus audio format used by WebM inputs. The video quality output option is set with the CRF parameter. `-shortest` secures the length of the output video equals the length of the shortest input, a safeguard in case the duration passed with `-t` is imprecise. The threads parameter gives the user an option to balance rendering speeds with resource consumption. The time it took to render the video and maximum memory consumed is shown by FFmpeg if `BENCHMARK` is enabled. To the `meeting-tmp.mp4` output file, the corresponding BBB room name is added as a title in the metadata. Rendering is initially done to a temporary file and renamed to `meeting.mp4` after completion to prevent downloads of a broken file while the export is still in progress, given the directory is publicly exposed.

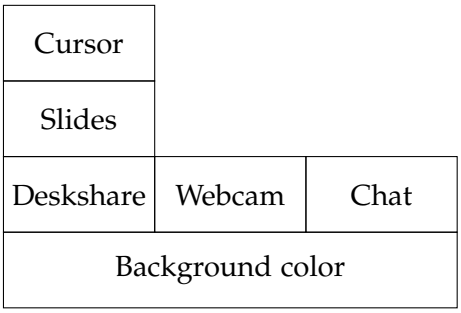

Figure 3.6: Exported video layers in FFmpeg

3.4.9 Adding captions and chapter marks

WebVTT subtitles were added as a separate stream in the MP4 container to avoid burning the subtitle tracks into the video, a more resource-intensive process. Rendering the captions into the video would require one encoding pass per language, yielding multiple versions of the output file. The locale required to set the subtitle language in the caption's JavaScript Object Notation (JSON) file is not recognized by FFmpeg; instead, the language's 3-letter International Organization for Standardization (ISO) 639[20] code is needed. An approximation is made by taking the initials of the `localeName`, as illustrated in Listing 2.1.

The `mov_text` codec is needed [FFma] to add multiple subtitle tracks to an MP4 file, but neither the encoder nor the decoder is installed along with BBB's FFmpeg default configuration. `movtext` must be added as a codec option when recompiling FFmpeg for caption support. This change does not interfere with FFmpeg's license and thus still allows for redistribution.

Metadata of the rendered `meeting.mp4` file was extended to add chapter marks for user-friendlier navigation, mirroring the slide thumbnails in BBB's player. Lines 1 to 6 in Listing 3.11 show the existing metadata of an example meeting. A `CHAPTER` entry is

[20] `https://www.loc.gov/standards/iso639-2/php/English_list.php`

```
;FFMETADATA1
major_brand=isom
minor_version=512
compatible_brands=isomiso2avc1mp41
title=Home Room
encoder=Lavf59.3.101

[CHAPTER]
START=0.0
END=335400000000.0
title=Slide 1

[CHAPTER]
START=335400000000.0
END=351300000000.0
title=Slide 2
```

Listing 3.11: Example MP4 video metadata, containing chapter marks

made with the slide's number and timing information in nanoseconds for each page displayed for more than 0.25 seconds. A quarter of a second was arbitrarily chosen as a limiting value to prevent new chapters from being added when rapidly navigating the document. The text file is then re-inserted into the video, replacing the former metadata.

3.5 PDF export feature

The exporter's code can be modified to convert the presentation with annotations into a PDF file by rendering the last state of the whiteboard for each slide. Cursor movements and panzooms are disregarded: only slide changes are taken into consideration. The interval tree data structure returns the drawings at the timestamp immediately preceding such events. The methods used to parse the shapes and slides are reused from the video exporter.

The behavior of this approach differs from what an end-user may expect, which is to see the uploaded document in its original order, with the annotations overlaid as they were last shown in the meeting. Due to new `image` elements being created alongside a canvas in `shapes.svg`, however, many copies of the same slide may appear in the final PDF if the presenter went back and forth in the document during the meeting. A method to only keep the last version of each slide maintaining the sequence is listed in Listing 3.12, in which two slides are said to be equal if they reference the same background image. The first instance of a slide is replaced with its last occurrence, including annotations. Its duplicates are set to `nil` and later removed from the array containing the final slides, using the compact method referenced in Section 3.4.5.

The PDF export feature is implemented as a separate post-publish `export_slides.rb`

```
1  def unique_slides(slides)
2      (0..slides.size - 1).each do |i|
3          ((i + 1)..slides.size - 1).each do |j|
4              next if slides[i].nil? || slides[j].nil?
5
6              if slides[i].href == slides[j].href
7                  slides[i] = slides[j]
8                  slides[j] = nil
9              end
10         end
11     end
12
13     slides.compact! || slides
14 end
```

Listing 3.12: Method to maintain the original slide order for the PDF export

script that uses `rsvg-convert` to convert the SVGs into PDFs, a lightweight CLI wrapper for `librsvg`. `rsvg-convert` is capable of combining multiple SVG files into a single PDF document, but not when the size of the input files differ[21] (state: 08/2021). A temporary workaround was to convert every SVG into a temporary PDF, combining them using a Ruby gem called Combine PDF[22].

Only recorded slides appear in the exported PDF file due to its dependence on the `shapes.svg` file. To obtain all slides, a possible approach is to implement a new workflow that is entirely independent from the presentation one. A separate workflow could also allow the raw uploaded document to be used in place of the rasterized images, allowing text to be searched in the resulting PDF. A concrete implementation, however, would go beyond the scope of this thesis: the current presentation conversion flow[23] done by `bbb-web` converts documents into rasterized SVGs.

3.6 Adding a `base-uri` option to FFmpeg

FFmpeg rasterizes SVGs with a Librsvg wrapper[24] that loads SVG data with the `rsvg_handle_new_from_data` function, which does not provide the ability to set a base URI [GNOa]. `rsvg_handle_new_from_stream_sync` is used instead that creates the handle from the input stream, flags, and a base file [Pro]. The input stream corresponds to the SVG data; the flags and URI path can be passed in FFmpeg as one of the audiovisual options. The patch was submitted to FFmpeg's mailing list and did not fail the tests

[21] `https://gitlab.gnome.org/GNOME/librsvg/-/issues/783`
[22] `https://github.com/boazsegev/combine_pdf`
[23] `https://docs.bigbluebutton.org/2.3/architecture.html`
[24] `https://github.com/FFmpeg/FFmpeg/blob/master/libavcodec/librsvgdec.c`

provided by their fully automated testing environment[25]. As of 08/2021, no manual inspection or feedback was given by FFmpeg developers.

3.7 Integration into Greenlight

BBB's GUI, Greenlight, was modified to show download buttons for the exported MP4 video and PDF files. BBB's documentation provides a guide[26] on how to customize Greenlight's interface, which is written in Ruby on Rails[27]. Members of the BBB community had already explored possible ways to add such buttons in the interface[28] for alternative exporting scripts[29], providing a foundation to make changes upon.

Adaptations encompassed making the buttons compatible with BBB 2.3 and enabling a direct download of the file instead of loading it in the browser's player first. Additionally, the buttons use labels rather than plain text to follow the browser's locale setting, displaying translated contents as a result. The modifications are available in a fork of Greenlight[30].

A drawback of that method is that the button is shown while the MP4 video is being rendered, throwing a `404 Not Found` error in the meantime. To fix that problem, a user on the exporter's repository issue tracker reported[31] that no modification to Greenlight is required, since a button to access each playback format is automatically added to the GUI once there is metadata as XML markup in BBB's published directory. The exporter uses the presentation's `metadata.xml` file as a blueprint, replacing the format name and link. Figure 3.7 shows both variants: the dropdown menu that appears after integrating the code in the front end, and the small blue buttons added by Greenlight once it locates the metadata files.

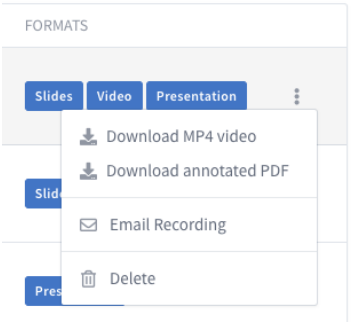

Figure 3.7: Download buttons in Greenlight, in two variants

[25] https://patchwork.ffmpeg.org/project/ffmpeg/patch/20210803094534.1000-1-daniel.petri@tum.de/
[26] https://docs.bigbluebutton.org/greenlight/gl-customize.html
[27] https://rubyonrails.org/
[28] https://groups.google.com/g/bigbluebutton-greenlight/c/pT9rF_9VKFU/m/sYg8XBvrAAAJ
[29] https://groups.google.com/g/bigbluebutton-dev/c/ZVq9LvB5w88
[30] https://github.com/danielpetri1/greenlight
[31] https://github.com/danielpetri1/bbb-recording-exporter/issues/33

4 Benchmarking

Benchmarks were done on a 2020 Mac Mini with the Apple M1 chip and 8 GB of memory. Table 4.1 contains a hardware overview of the model [Fru20].

		L1 cache [kB]		
CPU cores	Clockrate [GHz]	Data	Instructions	L2 cache [MB]
Efficiency: 4	2.06	64	128	12
Performance: 4	3.2	128	196	4

Table 4.1: Hardware information for Mac Mini's M1 system on a chip

The Graphics Processing Unit (GPU) has eight cores with 128 execution units supporting Apple's VideoToolbox[1] framework to make encoding and decoding video with hardware acceleration possible, which FFmpeg does implement. VideoToolbox lacks the CRF parameter, however, and is not compatible with other operating systems besides macOS [FFm20], such as the Ubuntu server hosting BBB throughout development of the exporter. Therefore, `-hwaccel` was not passed in the command from Listing 3.10 to explore GPU-based improvements, instead relying on more compatible yet CPU-intensive software decoders to render the video. FFmpeg was compiled from source from a development branch snapshot (version `N-102809-gde8e6e67e7`).

Two meetings served as a reference: an actual tutorial held at TUM described in Section 2.2 and a stress test with 10 participants, each actively interacting with the whiteboard. Under the circumstances summarized in Table 4.2, the exporter performs up to 39% faster than a screen capture on consumer-grade hardware. The exporter converted the tutorial up to 4.2 times quicker than a playback in real-time. Tables 4.3 and 4.4 show the impact the CRF parameter has on the file size of the final output and how enabling both the usage of SVGZs and the file scheme reduces data written on disk by over 64% and 94%, respectively. Additionally, trade-offs between GZIP compression policies are presented: per default a "compromise between speed and compression" is requested [Adl21]; `BEST_SPEED` as used by the exporter is fastest but compresses SVGZ data the least.

These parameters did not affect rendering speeds. As shown in Table 4.5, the FFmpeg thread count and the size of the chat canvas size does. For the load test, for instance, a reduced chat canvas size of 1280×32760 can hold all messages; for the tutorial 320×1020 suffices. These minimum values were manually determined to bring the improvements

[1] `https://developer.apple.com/documentation/videotoolbox`

Meeting	Length	BBB	Commands [MB]	Messages	Strokes	Frames
Load test	7min 51s	2.3	0.167	671 (430.0 kB)	3424	1727
Tutorial	8h 46min	2.2	3.4	3 (0.572 kB)	10552	9220

Table 4.2: Comparison between reference recordings used for benchmarking purposes

	Output size [MB]	
CRF	Load test	Tutorial
0	68.9	5610
23	39.95	1120
51	12.8	561

Table 4.3: Example impact of the CRF parameter on the rendered video file size

			Frame data written [MB]		Encoding [s]	
Policy	SVGZ	References	Load test	Tutorial	Load test	Tutorial
-	`false`	`false`	438.6	1110.0	2.7	23.5
-	`false`	`true`	412.8	206.3	2.1	2.3
Best speed	`true`	`false`	173.6	737.8	6.7	44.1
Best speed	`true`	`true`	158.5	62.0	5.5	3.5
Default	`true`	`false`	159.4	718.6	14.9	51.6
Default	`true`	`true`	145.2	56.1	13.3	5.9

Table 4.4: Example impact using the SVGZ and `base-uri` options

to light; users of the script are advised to adapt the constants to suit their needs. Render times stem from the `BENCHMARK` constant, which also reports the maximum Resident Set Size (RSS) occupied during the encode. For the tutorial, FFmpeg on the BBB server consumed circa 4100 MB of memory, while `maxrss` hovered around 850 MB for the stress test. On a Ryzen 3900XT server with 24 cores and 32 GB of memory, an enterprise BBB host service reported usually taking 10 to 15% of the meeting's duration to offer the replay to its customers [Dam21]; their setup copes with more than 4 concurrent exports.

4.1 Performance improvements

Besides the time it takes to render the actual video, the encoding of the presentation's files, as described in Section 2.2, also plays a role in the total execution time of the exporter. This section will explore the performance gains made over time in that area by comparing the preprocessing speeds for the chat, cursor, and whiteboard components across its versions.

The first few iterations of the exporter were noticeably slow. A prototype took

	Render time			
	Maximum chat canvas size		Minimum chat canvas size	
Threads	Load test	Tutorial	Load test	Tutorial
1	7min 40s	5h 22min 59s	7min 18s	4h 04min 03s
2	6min 03s	4h 21min 10s	5min 31s	2h 23min 07s
3	5min 44s	4h 04min 17s	5min 05s	2h 07min 19s
4	5min 25s	3h 23min 08s	4min 46s	2h 05min 03s

Table 4.5: Sample impact of the thread count and chat canvas size on exporting speeds

```
1  # Get shape IDs
2  draw.each do |drawing|
3      shapes << drawing.attr('shape')
4  end
5
6  # Obtain set of shape IDs
7  shapes = shapes.uniq
8
9  # Filter out duplicates: we're only interested in the last shape of each ID
10 shapes.each do |shape|
11     selection = draw.select {|drawing| drawing.attr('shape') == shape}
12     render << selection.last
13 end
```

Listing 4.1: Innefficient `remove_adjacent`, executed for every whiteboard frame

around 18 minutes to generate SVG slides and convert them to PNG, which is an unnecessary step. Once skipped, another origin for the sluggishness turned out to be the implementation of the `remove_adjacent` function shown in Listing 4.1. Originally, the shape ID attributing each stroke part to a whiteboard annotation was copied into an array on which Ruby's `uniq` function was called to obtain the set of unique shapes. For each of those, their last occurrence was filtered out using Ruby's `select` and placed in an array containing the final drawings to be rendered. Essentially, redundant copies and `uniq` not exploiting the fact that the shapes array is sorted in order of appearance curbed the method. At first, this method was scrapped entirely, arguing that it was unnecessary in BBB 2.3 and dropping the idea of backward compatibility. It was re-added later once the efficient implementation was come up with.

In the beginning, each mouse movement and incoming chat message resulted in a new frame being produced in addition to the ones required to render the whiteboard. For a long presentation such as the tutorial, this approach resulted in over 64400 separate SVG frames containing a copy of the same red pointer at a different location on the screen. The use of FFmpeg's `sendcmd` got rid of the need for separate frames, saving many costly Input/Output (IO) operations. Further IO improvements came from opening the text

files holding the instructions for FFmpeg once before the loop writing the commands instead of opening and closing it every iteration.

XPATH queries were optimized by either parsing the files with Nokogiri's XML Reader to handle the data in a more efficient data structure, e.g., queues and the interval tree, or by making the query itself more efficient by replacing instances of the // operator. The reader parser is only called once, traversing the `XML` file similarly to how a "cursor would move" [Nok21] throughout a document. Data the exporter needs is extracted during that single pass. On the other hand, queries starting with // — which selects the node itself or all descendants matching the request — were called multiple times for each interval in the video timeline, repeatedly traversing the entire document. In the few remaining cases where the use of XPATH expressions was deemed fit, // was replaced with direct steps to an element in the subtree from which the search starts. The path to the nodes is derived from the file's encoding scheme.

Nokogiri's `XML` Builder turned out to be slower than the `XML` Builder gem by a factor of 10 when creating whiteboard `SVG` frames.

Markup such as chat `tspan` elements are concatenated with the « operator instead of += on the same grounds throughout the exporter's code. This is due to « concatenating in-place without making a copy of the string object, an adjustment especially felt in the stress test due to the hundreds of messages.

Re-introducing an optimized version of the `remove_adjacent` function thwarts the exporter from writing around 1 GB of extra data to the disk for the tutorial. By including references to linked images instead of performing a conversion to the Base64 format, a further 0.9 GB is saved. Figure 4.1 plots the effects these improvements had on the time taken to encode the files.

Multithreading the export of frames with user-level threads did not result in speed improvements. Likewise, shortening negative cursor coordinates in an attempt to write less data when the mouse is outside the whiteboard's area introduced branches and comparisons that ultimately slowed the encoding process.

4.2 Static Analysis

Three Ruby gems were used during development to improve the code's readability and performance: Fasterer[2], which suggests changes resulting in speed gains, Reek[3], to remove code smells, and Rubocop[4], a linter based on Ruby's community style guide[5]. Fasterer, for instance, reports that using Ruby's `each_with_index` is slower than a `while` loop and that adding an element to the start of an array with `unshift` is over 260 times quicker than using `insert` [Fas19]. Reek was used to remove most duplicate method calls in the exporting process, and Rubocop to format the code enforcing the convention.

[2] `https://github.com/DamirSvrtan/fasterer`
[3] `https://github.com/troessner/reek`
[4] `https://github.com/rubocop/rubocop`
[5] `https://rubystyle.guide/`

With the help of these tools, the post-publish scripts passed Sonar Cloud's[6] quality gates concerning bugs, security vulnerabilities, code smells, and code duplication in the PRs submitted to BBB.

4.3 Dynamic Analysis

Rbspy[7] 0.8.1 was used to profile the exporter while it encodes the chat, cursor, and whiteboard frames. The resulting flamegraph in Figure 4.2 has a minimum flame width of 0.1% and arises from running the exporter for the tutorial in its default use case, i.e., with SVGZ support enabled and references in FFmpeg disabled. Stack traces for the `remove_adjacent`, `render_chat`, and `render_cursor` functions notably barely showed alongside lookups in the interval tree. The first block in `render_whiteboard` opens the file where the frame's location and duration are written down. The block two levels deep is the loop that begins by determining the current slide, its `view_box` parameter, and which shapes to draw. Once queried, most time is spent exporting SVG frames created with the XML Builder gem due to the SVGZ compression and writing images in the Base64 format. Given the samples from Table 4.4 and the flamegraph, it appears that turning compression on with FFmpeg references enabled yields the best trade-off between speed and resource usage.

The flamegraph for the stress test is similar, exception being that the `render_chat` function is responsible for a bit more than 10% of the total execution time. Line numbers in the figure refer to the client-side version of the exporting script.

4.4 Benchmarking the PDF export

Converting a presentation into a PDF file is not as efficient as encoding the assets for the video export since the SVG slides are converted to individual PDFs before merging them into one. Table 4.6 summarizes the impact had on the conversion speed contingent upon the SVGZ option and its policy. The PDF for the load test contains 5 slides (208 kB), the tutorial has 178 taking up 13.2 MB of space, respectively.

		Load Test		Tutorial	
Policy	SVGZ	Data written [kB]	Time [s]	Data written [MB]	Time [s]
-	`false`	597	0.698	10.3	28.4
Best speed	`true`	370	0.713	24.7	10.9
Best compression	`true`	358	0.724	24.5	11.1

Table 4.6: Benchmarks for the PDF export

[6] `https://sonarcloud.io/`
[7] `https://rbspy.github.io/`

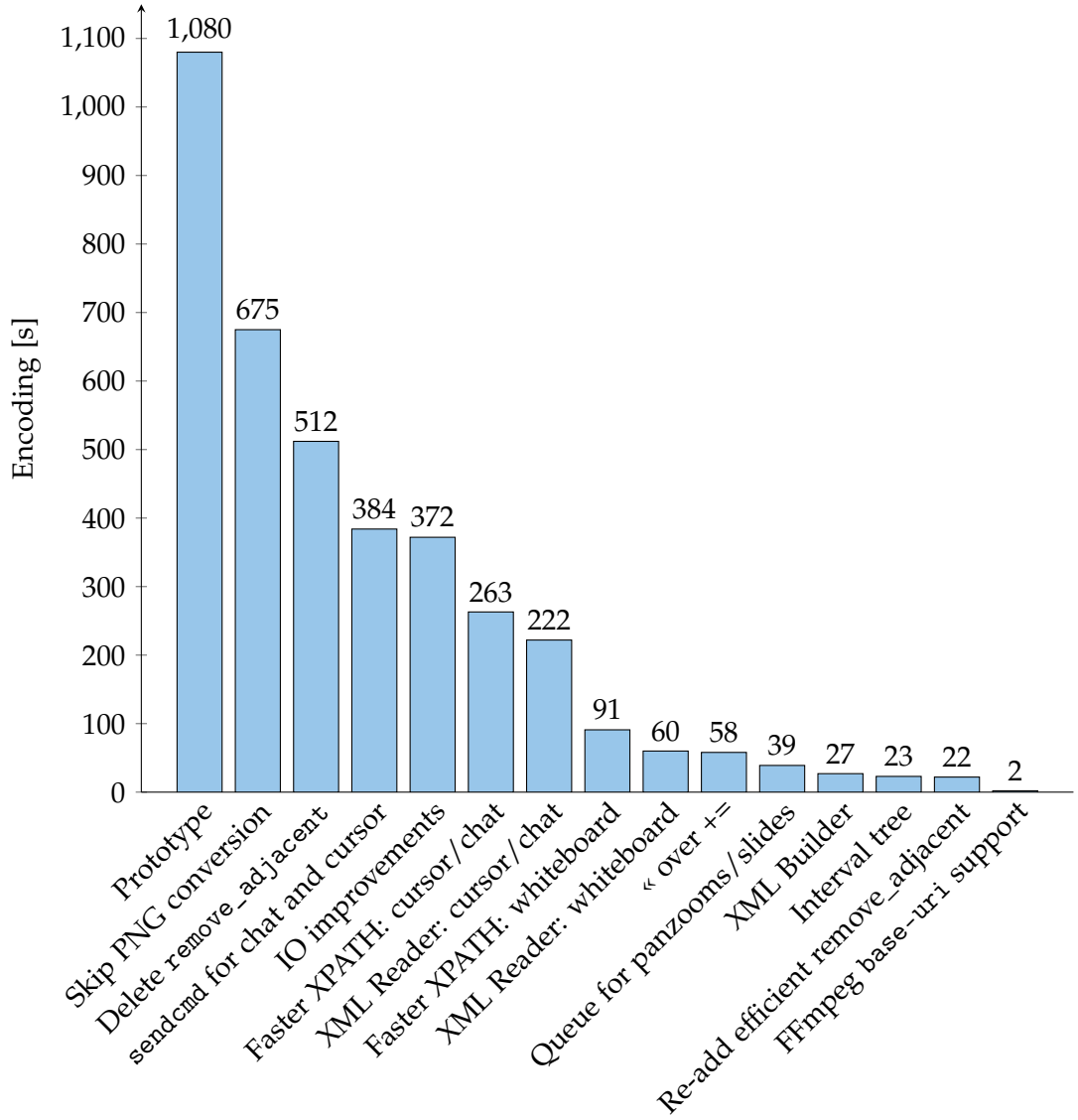

Figure 4.1: Timeline and impact of the performed optimization steps on the tutorial

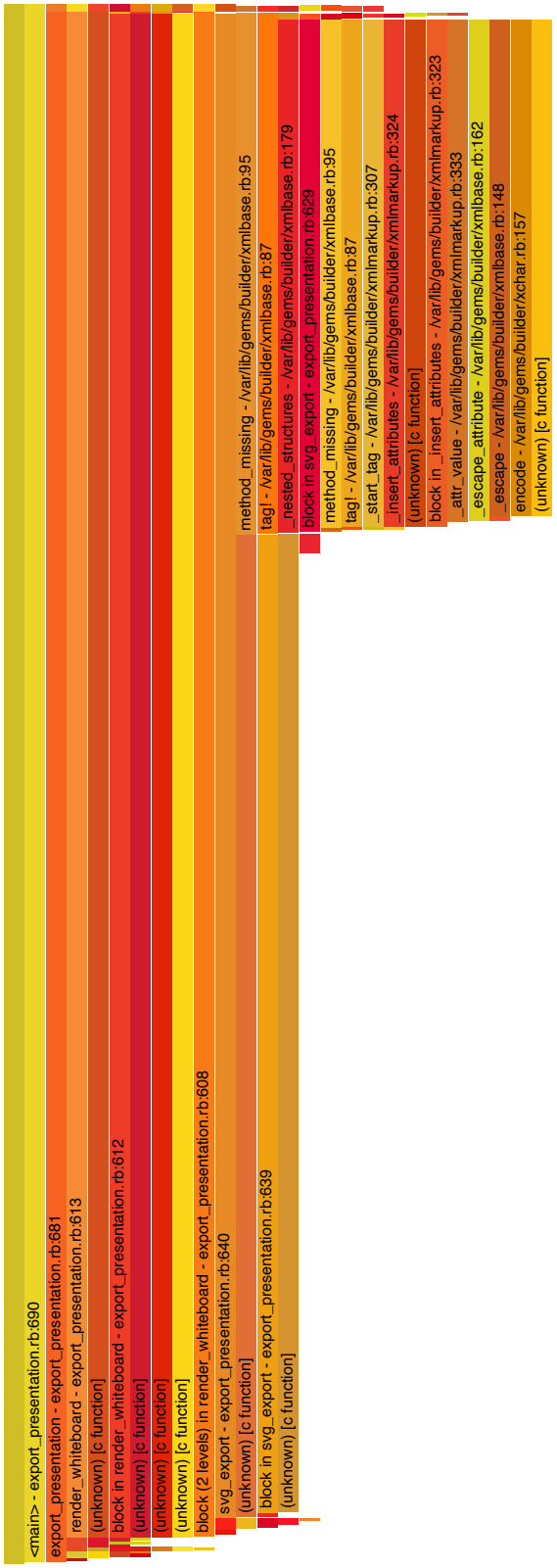

Figure 4.2: Flamegraph with a cutoff flame width of 0.1% for the tutorial when SVGZ is on and FFmpeg references off. Paths were shortened or ommited, and the background was removed

5 Conclusion

All requirements listed in Section 1.4 were fulfilled.

5.1 Deployment

Two separate PRs were submitted to BBB's repository with the video[1] and PDF export code[2]. To this end, BBB's contributor license agreement was signed and sent in. Fred Dixon, BBB's product manager, has stated that the script will be turned into a workflow that can be separately installed starting from BBB version 2.5, planned towards the end of 2021. Introducing the exporter not as an extension to the presentation workflow but as a complementary feature maintained by the community prevents upcoming changes to BBB's core from being held back by compatibility issues with the script. He reasons that the attention of BBB developers needs to be focused on the central components of the product to ensure it is shipped in a stable state [Dix21b].

5.2 Feedback

BBB community members seemed to welcome the exporter. Stephen Dame from HostBBB stated that "the approach scales very well" [Dam21] and offers free conversions from a link on his website, besides offering the exporter as a premium feature to his private customers. Felipe Caetano implemented the code on the BBB cluster of a Brazilian university (Universidade Federal de Juiz de Fora)[3], saying that the "solution is simple, yet efficient and elegant." [Cae21] Hiroshi Suga, a biology professor at the Prefectural University of Hiroshima and BBB contributor, called the exporter "the best one" in his "personal opinion." [Sug21a]

5.3 Future work

New features are being added to BBB 2.4, some of which are displayed during the replay of a recording and therefore need to be addressed by the exporting script. It is scheduled to launch in September 2021 officially. The recording's public chat will contain the polling results and links to external videos shared during the meeting, besides making

[1] https://github.com/bigbluebutton/bigbluebutton/pull/12533
[2] https://github.com/bigbluebutton/bigbluebutton/pull/13016
[3] https://www2.ufjf.br/ufjf/

instructor messages easier to discern by printing them in bold.[4] Exported BBB 2.4 recordings did not show these upgrades: formatting and contents of the chat remained consistent with BBB 2.3.

Additionally, enabling a webcam filter to blur out the background will be a privacy option users have. No action should be required on the exporter's part as long as the filters appear in the processed webcams video file. This equally applies to a PR adding the external shared videos to the deskshare file.

An eraser and marker function for the whiteboard may be on BBB's roadmap as well, given the PRs[5] [6] by Hiroshi Suga. Supporting these new shapes is challenging due to extensive SVG clipping and masking, again using references that FFmpeg has difficulties with. The exporter currently does not support either tool.

Together with a dynamic adjustment of the chat's framerate to guarantee no message is missed when more messages come than what can fit in the chat area, a method to determine the `CHAT_CANVAS_WIDTH` and `CHAT_CANVAS_HEIGHT` constants automatically on a per-export basis should be implemented to further improve its efficiency. A closer look into hardware acceleration options for FFmpeg using the GPU is warranted, as well as checking how it handles splitting the deskshare video into multiple overlays to remove the intervals in which the deskshare is blank. Think Modular, a startup providing a commercial BBB hosting service, requested more flexible layout customization options to hide the webcams and anonymize chat names due to privacy concerns. Access to the rendered output file should only be granted to those logged in. They also vouched for prettier output in terms of the layout's design, a suggestion initially made by end-users of an Iranian BBB provider[7]. Changes would entail adding separation lines to distinguish the chat from the cameras and whiteboard, increasing the margins between them as well.

Likewise, Blindside Networks, the company that founded BBB, envisions that the code handling whiteboard annotations can be reused in the HTML5 client to bring work done on documents during breakout rooms back into the main session. This enhancement could potentially arise from the PDF exporting script, which, as it stands, requires performance improvements since users would not find a wait of up to half a minute to move slides between rooms acceptable. Speed gains could come from extending librsvg to accept a base URI parameter, apart from combining the final PDF directly from the SVGs once librsvg uses their page size to assemble multipage PDFs.

[4] `https://docs.bigbluebutton.org/2.4/new.html`
[5] `https://github.com/bigbluebutton/bigbluebutton/pull/11018`
[6] `https://github.com/bigbluebutton/bigbluebutton/pull/11021/commits`
[7] `https://github.com/danielpetri1/bbb-recording-exporter/issues/41#issue-972486167`

Acronyms

AAC Advanced Audio Coding

BBB BigBlueButton

CLI Command-Line Interface

CPU Central Processing Unit

CRF Constant Rate Factor

FPS Frames per second

GES GStreamer Editing Services

GPU Graphics Processing Unit

GUI Graphical User Interface

GZIP GNU zip

HTML Hypertext Markup Language

IO Input/Output

ISO International Organization for Standardization

JPEG Joint Photographic Experts Group

JSON JavaScript Object Notation

PDF Portable Document Format

PNG Portable Network Graphics

PR Pull Request

RSS Resident Set Size

SVG Scalable Vector Graphics

TUM Technical University of Munich

URI Uniform Resource Identifier

UTF Universal Transformation Format

W3C World Wide Web Consortium

WebVTT Web Video Text Tracks

XHTML Extensible HyperText Markup Language

XML Extensible Markup Language

XPATH XML Path Language

Glossary

Base64 Format providing a textual representation of binary data using 64 characters. 16, 17, 34, 35

deskshare Sharing the computer's screen with others. Portmanteau formed by combining desktop and share. 2, 3, 5, 9, 14–16, 24–26, 40, 49

gem Instance of an open-source Ruby library. 13, 17–20, 22, 23, 28, 34, 35

panzoom Act of zooming into a slide and moving it around. Blend of panning and zooming. 2, 3, 8, 11, 18, 20, 21, 23, 27

SVGZ Losslesly compressed SVG file using GZIP. 15, 23, 31, 32, 35, 37, 45, 47

workflow Conversion process a recording goes through in order to be published in another format. 3, 5, 28, 39

List of Figures

List of Tables

Listings

Bibliography

[Adl21] J.-l. G. M. Adler. *Zlib Manual, version 1.2.11*. Sept. 7, 2021. URL: https://zlib.net/manual.html (visited on Jan. 15, 2017).

[Ber08a] M. de Berg; Otfried Cheong; Marc van Kreveld; Mark Overmars. "Computational Geometry: Algorithms and Applications." In: 3rd ed. Springer, 2008. Chap. 10, pp. 220–226. ISBN: 978-3-540-77974-2. DOI: 10.1007/978-3-540-77974-2.

[Ber08b] M. de Berg; Otfried Cheong; Marc van Kreveld; Mark Overmars. "Computational Geometry: Algorithms and Applications." In: 3rd ed. Springer, 2008. Chap. 10, pp. 231–237. ISBN: 978-3-540-77974-2. DOI: 10.1007/978-3-540-77974-2.

[Ber21] D. Berman. *Accessibility Compliance Analysis And Opinion*. July 2021. URL: https://bigbluebutton.org/accessibility/ (visited on Sept. 7, 2021).

[Cae21] F. Caetano. *Comment on the project's issue tracker*. June 7, 2021. URL: https://github.com/danielpetri1/bbb-recording-exporter/issues/28#issue-911531735 (visited on Sept. 7, 2021).

[Dam21] S. Dame. *Comment on BBB-Exporter's issue tracker*. July 9, 2021. URL: https://github.com/danielpetri1/bbb-recording-exporter/issues/40#issuecomment-901082740 (visited on Aug. 18, 2021).

[Dixa] F. Dixon. *BigBlueButton's Architecture Documentation*. URL: https://docs.bigbluebutton.org/2.3/architecture.html (visited on Sept. 7, 2021).

[Dixb] F. Dixon. *BigBlueButton's Recording Documentation*. URL: https://docs.bigbluebutton.org/dev/recording.html (visited on Sept. 7, 2021).

[Dix12] F. Dixon. *Enable users to download a recording as video*. May 3, 2012. URL: https://github.com/bigbluebutton/bigbluebutton/issues/1969 (visited on Sept. 7, 2021).

[Dix21a] F. Dixon. *Additional recording formats are not automatically enabled when their packages are installed*. May 2, 2021. URL: https://github.com/bigbluebutton/bigbluebutton/issues/12241 (visited on Sept. 7, 2021).

[Dix21b] F. Dixon. *BigBlueButtonWorld - Roadmap for BigBlueButton*. July 2, 2021. URL: https://www.youtube.com/watch?v=pkd_F8hJwgM&t=2005s (visited on Sept. 7, 2021).

[Fas19] Fast Ruby contributors. *Fast Ruby - Array*. Mar. 16, 2019. URL: https://github.com/JuanitoFatas/fast-ruby#array (visited on Sept. 7, 2021).

[FFma] FFmpeg contributors. *FFMPEG An Intermediate Guide: subtitle options*. URL: https://en.wikibooks.org/wiki/FFMPEG_An_Intermediate_Guide/subtitle_options (visited on Sept. 8, 2021).

[FFmb] FFmpeg contributors. *FFmpeg Formats Documentation: concat*. URL: https://ffmpeg.org/ffmpeg-formats.html#concat (visited on Sept. 8, 2021).

[FFm17] FFmpeg contributors. *FFMpeg Documentation: Slideshow*. Sept. 16, 2017. URL: https://trac.ffmpeg.org/wiki/Slideshow (visited on Sept. 7, 2021).

[FFm20] FFmpeg contributors. *FFmpeg: Hardware Acceleration*. Nov. 18, 2020. URL: https://trac.ffmpeg.org/wiki/HWAccelIntro (visited on Sept. 7, 2021).

[FFM21] FFMpeg contributors. *H.264 Video Encoding Guide*. Mar. 11, 2021. URL: https://trac.ffmpeg.org/wiki/Encode/H.264 (visited on Sept. 8, 2021).

[Fru20] A. Frumusanu. *The 2020 Mac Mini Unleashed: Putting Apple Silicon M1 To The Test*. Nov. 17, 2020. URL: https://www.anandtech.com/show/16252/mac-mini-apple-m1-tested (visited on Sept. 7, 2021).

[GNOa] GNOME contributors. *RsvgHandle*. URL: https://developer-old.gnome.org/rsvg/unstable/RsvgHandle.html#rsvg-handle-new-from-data (visited on Sept. 7, 2021).

[GNOb] GNOME contributors. *Security and locations of referenced files*. URL: https://gnome.pages.gitlab.gnome.org/librsvg/doc/librsvg/index.html (visited on Sept. 7, 2021).

[MDN21a] MDN contributors. *MDN Web Docs: <g>*. June 10, 2021. URL: https://developer.mozilla.org/en-US/docs/Web/SVG/Element/g (visited on Sept. 7, 2021).

[MDN21b] MDN contributors. *MDN Web Docs: Paths*. Mar. 25, 2021. URL: https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths (visited on Sept. 7, 2021).

[MDN21c] MDN contributors. *MDN Web Docs: viewBox*. Aug. 30, 2021. URL: https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/viewBox (visited on Sept. 7, 2021).

[Neu+08] A. Neumann, D. Schepers, C. Lilley, E. Dahlström, C. McCormack, N. Ramani, S. Hayman, C. Northway, V. Hardy, A. Shellshear, A. Emmons, D. Jackson, A. Grasso, O. Andersson, J. Ferraiolo, A. Quint, and R. Berjon. *Scalable Vector Graphics (SVG) Tiny 1.2 Specification*. W3C Recommendation. W3C, Dec. 22, 2008. Chap. 10. URL: https://www.w3.org/TR/2008/REC-SVGTiny12-20081222/.

[Nok21] Nokogiri contributors. *Class: Nokogiri::XML::Reader*. Sept. 7, 2021. URL: https://www.rubydoc.info/github/sparklemotion/nokogiri/Nokogiri/XML/Reader (visited on Sept. 7, 2021).

[Pro] T. G. Project. *Using RSVG with GIO: Functions*. URL: https://developer-old.gnome.org/rsvg/unstable/rsvg-Using-RSVG-with-GIO.html#rsvg-handle-new-from-stream-sync (visited on Sept. 7, 2021).

[Sch08] D. Schepers. *SVG 1.2 Tiny Test Suite Implementation Matrix*. Nov. 13, 2008. URL: https://www.w3.org/Graphics/SVG/1.2/Tiny/ImpReport.html (visited on Sept. 7, 2021).

[Sug21a] H. Suga. *Comment on BBB's issue tracker*. Aug. 7, 2021. URL: https://github.com/bigbluebutton/bigbluebutton/issues/12935#issuecomment-894724794 (visited on Sept. 7, 2021).

[Sug21b] H. Suga. *Optional real-time update of whiteboard annotations (2.3)*. May 11, 2021. URL: https://github.com/bigbluebutton/bigbluebutton/issues/12345 (visited on Sept. 7, 2021).

[Wil+18a] E. Willigers, D. Schulze, D. Storey, C. Lilley, B. Brinza, and A. Bellamy-Royds. *Scalable Vector Graphics (SVG) 2*. Candidate Recommendation. W3C, Oct. 4, 2018. Chap. 11. URL: https://www.w3.org/TR/2018/CR-SVG2-20181004/.

[Wil+18b] E. Willigers, D. Schulze, D. Storey, C. Lilley, B. Brinza, and A. Bellamy-Royds. *Scalable Vector Graphics (SVG) 2*. Candidate Recommendation. W3C, Oct. 4, 2018. URL: https://www.w3.org/TR/2018/CR-SVG2-20181004/.

[Wil+18c] E. Willigers, D. Schulze, D. Storey, C. Lilley, B. Brinza, and A. Bellamy-Royds. *Scalable Vector Graphics (SVG) 2*. Candidate Recommendation. W3C, Oct. 4, 2018. Chap. 8. URL: https://www.w3.org/TR/2018/CR-SVG2-20181004/.