Charles University in Prague

Faculty of Mathematics and Physics

BACHELOR THESIS

Šimon Rozsíval

Vector Screencast

Department of Distributed and Dependable Systems

Supervisor of the bachelor thesis:	Mgr. Martin Děcký
Study programme:	Computer science
Specialization:	Programming and software systems

Prague 2015

I would like to thank my supervisor, Martin Děcký, for his valuable pieces of advice, and Otakar Jícha from *Khanova škola*, for the idea of this project and for lending me a graphics tablet for testing.

I would also like to thank my family and friends for supporting me during my studies.

Název práce: Vektorový screencast

Autor: Šimon Rozsíval

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Martin Děcký

Abstrakt: Cílem bakalářské práce je vytvořit software pro záznam a přehrávání výukových videí pro potřeby Khanovy školy. Na rozdíl od běžných videí nejsou obrazová data uložena ve formě bitmap, ale jako vektory, což umožní snížit datovou náročnost a vykreslit obraz ostře při libovolně velkém rozlišení obrazovky uživatele. Přehrávač videa i nástroj pro nahrávání běží ve webovém prohlížeči. Součástí práce je také návrh a implementace vhodného formátu pro uchovávání obrazových a zvukových dat a implementace v softwarové architektuře klient/server.

Klíčová slova: screencast, vektory, video, on-line

Title: Vector Screencast

Author: Šimon Rozsíval

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Martin Děcký

Abstract: The goal of this bachelor thesis is to create a software for recording and playback of educational videos for Khanova škola (Czech clone of Khan Academy). Contrary to common videos the visual data is not stored as a sequence of bitmaps, but as vectors. This allows to reduce the data bandwidth and playback sharp images in any target resolution. The player and also the tool for recording the videos runs in a web browser. The thesis also focuses on designing and implementing a suitable file format for storing the visual and audio data and implementing the software according to the client/server paradigm.

Keywords: screencast, vector, video, on-line

Contents

Introduction

A year ago two members of “Khanova škola”, the Czech brach of the Khan Academy, came up with an idea of improving the current technical solution of recording and displayling educational videos on their website. It turns out that some of the videos recorded only a few years ago do not look very well and the hand-drawn text on these videos is not even legible if it is enlarged. The other extreme is small displays of tablets and smartphones, where the downscaled letters are too small to read and can’t be zoomed well in most video players. Their idea was to create a vector-based animation instead of classical bitmap-based screencasts. This animation could be scaled to any display resolution without any loss of information. The process of recording can be simulated in any distant future based on the recorded data of user’s behaviour. The animation can be rendered using a different algorithm and the author does not have to record his screen again. As a result, the video will never become obsolete because of its poor visual quality.

One of the other reasons for this type of solution was the possible decrease of the size of data transfered over the Internet, as most of the image does not change between each two frames and very often the image does not change at all. The file size of a vector-based format does not correspond to the quality of the video. In regular bitmap formats, the higher the resolution of an image, the the larger the data file. Vector-based format has only one version for every display resolution.

Khan Academy

The idea of *Khan Academy* dates back to 2003 when Salman Khan began tutoring his cousin over an instant messenger via drawing pictures with a computer mouse. Salman then started to record these videos and put them on his Youtube channel, so that someone could watch them later. This channel became the cornerstone of Khan Academy.

Khan Academy is known and has grown a lot, but the style of Khan Academy videos remains the same. A person draws lines and diagrams using a bitmap editor on his computer and talks about the subject aloud while recording his computer screen and recording his voice using a microphone. This style is sometimes called the “Khan-style video” (KSV) [1]. These videos are then uploaded to Youtube and embedded in the Khan Academy website.

Apart from the video lectures, the website also contains exercises and quizzes to encourage students to keep learning. The pace of the lesson depends on the student. They can pause the videos or watch them several times before continuing with the lesson.

Most of the videos are recorded in English, but many of the videos are translated into other languages - by replacing the audio track with a different one or with subtitles. One of the projects working on the localization of Khan Academy videos is an official Czech branch called “Khanova škola”.

[1]https://www.youtube.com/watch?v=Ohu-5sVux28

“Khanova škola”

Khanova škola is the Czech branch of the Khan Academy [19]. It is a non-profit organization. Volunteers around “Khanova škola” translate original videos from English to Czech to make them accessible to children at Czech elementary schools.

Screencast

A screencast is a video created by recording computer screen output, often accompanied with an audio commentary. This process can be used in many different ways, for example to record a tutorial explaining how to use a specific computer program. The quality of the recorded video depends mainly on the resolution of the user’s device and the recorded area of the screen. Khan Academy uses screen capturing tools to record the virtual canvas of a bitmap editor onto which the author draws using the tools of the editor and talks about the covered subject.

Goal of the thesis

The result of this thesis should be an open-source library suitable for extending any web application, such as “Khanova škola”, with the abilities of recording and playing KSVs in all modern web browsers. Screencast will be recorded and replayed using a technology based on vector graphics to guarantee the quality of the video on any screen and to achieve lower file size of the video file. An appropriate vector-based file format should be chosen or defined to store screencast data.

The library should be easily adjustable and configurable for different purposes. The user interface should be fully translatable.

Thesis structure

The first part of the thesis is about distance education in general. Reader should get a notion of current distance education systems and their limits.

The technical requirements of the project are described in the following chapter. The third chapter is the analysis of the available tools and algorithms and which approaches to choose. A file format for the puroposes of this project is defined and described afterwards.

The following chapters contain information about the implementation of the library and its components, how the library can be integrated into websites, and how to use the library to record and play screencasts.

1. Distance education

Distance education is not only a phenomenon of the last few decades, but can be traced at least back to the 18th century, when Caleb Phillipps posted an advertisment called “Teacher of the New Method of Short Hand” in Boston Gazette, saying “Persons in the Country desirous to Learn this Art, may by having the several Lessons sent weekly to them, be as perfectly instructed as those that live in Boston.” [2]

With the development of the Internet and its general accessibility in the developed world, providing distance education has become much easier and has spread widely. In some countries tuition rates are high and young people take loans so they can affoard to study. This topic is covered in a fitting way by John Oliver in his show [3]. The flexibility and low cost of distance education over the Internet gives people, who would not be otherwise able to attend a traditional university, an opportunity to gain knowledge and train skills from their homes [4] spending much less money or even for free.

Students in the developing world are also taking the advantage of educational content available on the Internet. Several of the top U.S. universities, like Harvard, Stanford or MIT, put some of their materials on so called *Massive Open Online Courses* (MOOC) like Coursera, edX or Udacity. This content is then available to anyone with a computer and Internet connection and knowledge of the language. A great example is *Kepler* - a non profit university project in Rwanda [6]. The goal of this project is to “provide an American-accredited degree, a world class education, and a clear path to good jobs for thousands of students for around $1,000 tuition per year.” [7]

A concept of teaching often reffered to as *Flipped Classroom* uses distance education. The idea is to let students study lecture materials at home at their own pace and then apply gained knowledge at school the next day by doing activities to illustrate the concepts. The teacher can then help them or explain details in person. These materials are often in the form of videos either from an open source, like the Khan Academy, or created by the teachers themselves.

1.1 Existing systems

In the next few paragraphs I will try to pick some of existing distance education services and tools available on the Internet. This list is not complete and is only meant to give the reader a notion of available technologies and their paradigms, on which the project of Vector Screencast is based.

1.1.1 Coursera

The mission of *Coursera* is to “provide universal access to the world’s best education.” [10] Anyone can, for free, go through materials published by universities and other organizations aimed at education.

Courses at Coursera consist mainly of video lectures commonly with a transcript and an attached presentation document. These videos can be viewed in a web browser on demand or downloaded to the user’s computer. After studying the materials, students can submit assignment solutions and take quizzes and receive a Verified Certificate for the accomplishment of the course. These certificates are not for free.

Most of the courses are in English and only a few of the courses are also translated into other languages. It is not possible for everyone to publish his materials through Coursera.

1.1.2 Youtube.com

Youtube.com [11] is not an educational service by design. Youtube allows people to create their own channels and upload their video content to share it with other Internet users for free.

Youtube was launched in 2005 and has became one of the most frequently visited websites on the Internet according to Alexa Internet [12]. Uploading video to Youtube is free, however advertisment is displayed to the user while watching the videos.

The ease of making original-created videos available to wide audience makes Youtube a perfect place for all individuals and organizations, who want to share their ideas or any video materials. Many educational channels can be found here, for example *Numberphile* [13], *Veritasium* [14], and *Khan Academy* [15].

Youtube videos can be viewed only online in a web browser or a specialized mobile app. There are only unofficial tools for downloading these videos.

The form and content of the video is practicaly unlimited as long as it does not violate the terms of the service. The maximum file size of a video is 128 GB in size and 11 hours in length. To upload larger or longer videos, the user must split them into several parts. A video can have a text description which might contain the transcription of the video content and any subtitles can be attached to a video. Youtube also downscales videos to multiple resolutions so they can be viewed with a low speed Internet connection, at the cost of quality.

1.1.3 Moodle

Moodle [8] is an open-source project used by millions of users [9]. It is a robust platform for sharing custom learning materials with students. The source code of Moodle can be downloaded and deployed on any server.

1.1.4 Educreations and ShowMe

Educreations is a service for creating and sharing educational videos similar to Khan Academy videos [16]. The idea is that teachers create their own videos by drawing onto a digital whiteboard and then share these videos with their classes.

Educreations is designed for an iPad, but can be used also as a web application from any web browser which supports Adobe Flash Player. Web application supports playing and recording of videos and the video player can be easily embedded into any website and watched without registration. Content of the video is scaled appropriately to the output screen and the lines are drawn smoothly.

The iPad app is free and users download it through the Apple AppStore. Users must create an Educreations account to use the app. There is a free usage

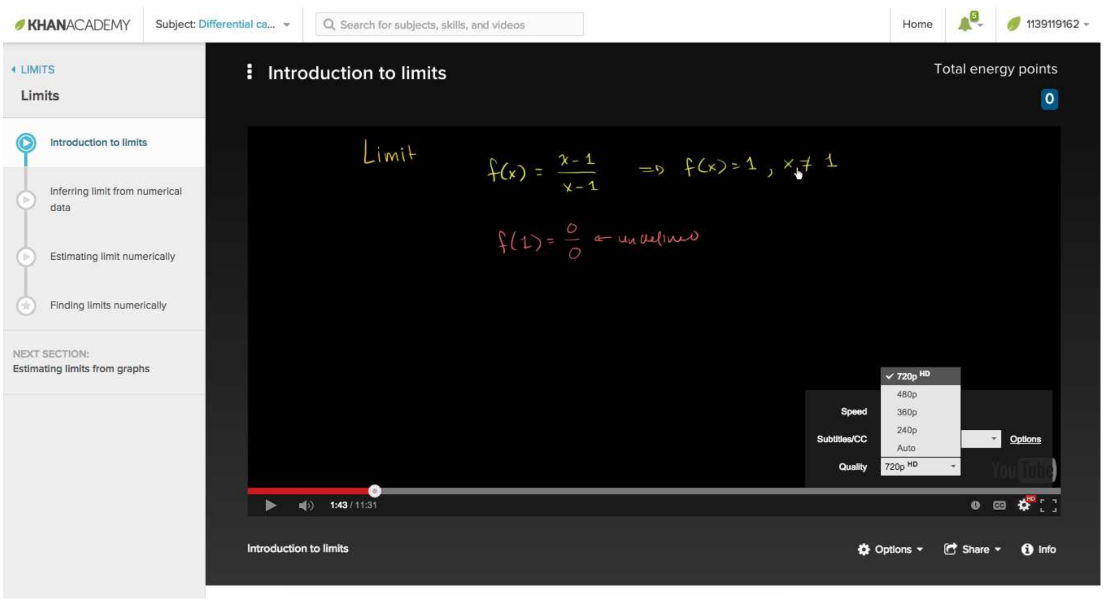

KHANACADEMY
Subject: Differential ca...
Search for subjects, skills, and videos
Home
1139119162
LIMITS
Limits
Introduction to limits
Inferring limit from numerical data
Estimating limit numerically
Finding limits numerically
NEXT SECTION:
Estimating limits from graphs
Introduction to limits
Total energy points
0
Limit
$f(x) = \frac{x-1}{x-1} \Rightarrow f(x) = 1,\ x \neq 1$
$f(1) = \frac{0}{0} \leftarrow$ undefined
Speed
Subtitles/CC
Quality
720p HD
480p
360p
240p
Auto
Options
1:43 / 11:31
Introduction to limits
Options
Share
Info

Figure 1.1: Khan Academy lesson

plan which allows users to watch, create, and share videos. The free plan allows only a limited use of some of the features, some are not even available. For more storage capacity for user recorded videos, and advanced tools, users must pay monthly fees. The software is proprietary and so is the file format of the video.

ShowMe is a very similar service [17]. An iPad app is available for recording and viewing videos. Recorded videos be played in a web browser, but not recorded. Pricing is very similar to Educreations.

Both *ShowMe* and *Educreations* target mainly tablets users. The advantage of tablets is their touchscreen which can be used for drawing in a natural way with fingers or a special stylus.

1.1.5 Khan Academy

A similar service to Coursera is *Khan Academy* which was described earlier on page 2. A glimpse of a Khan Academy lesson is captured in Figure 1.1.

Apart from publicly available videos, Khan Academy provides tools for teachers. Teachers can follow the progress and results of their students during the classes and at home when students are doing their homework. Teachers cannot publish their own content though and must rely on the content provided by the Khan Academy.

2. The Vector Screencast project

The overall project consists of two separate tools — the *recorder* and the *player*. Each of these tools behaves differently and will be used by different people. While the video player will be used by general audience, the recorder will be used by a much narrower group of content creators.

Thanks to the sparse nature of vectors, in comparison to bitmaps, data consumption might also be reduced or at least similar to the data consumption of bitmap videos. This animation would also be linked to an audio track of author's voice commentary forming a complex KSV suitable for educational purposes.

2.1 Screencast recording tool requirements

The user, who wants to create a new screencast, should enter a website in his web browser and without installing any additional software start recording a video using his mouse, touchscreen or digital stylus and a microphone. This video should then be uploaded to a server.

If the user uses a pressure-sensitive stylus, then the applied pressure during drawing should be recorded too. The more pressure the user applies, the thicker the line will be and vice versa. Using this specific hardware might require additional drivers or specific software installed.

Different brush sizes, brush colors and background colors are available to the user. The user must be able to erase certain parts of the canvas or the whole canvas at once. The user must be able to pause and continue recording at any time.

Lines are immediately rendered on the screen as the user draws them, so he or she sees exactly the same output as the viewer will, when he is playing the video later. All the raw data collected from the user should be stored, including the data that have no effect for video playback, like tracking the position of the cursor while recording of the screencast itself is paused. This will allow the user to simulate the process of video recording and use it for further post-processing in any distant future. Recording of this extra information should be optional.

2.2 Screencast player requirements

Any user should be able to play vector video online in all major modern web browsers without any special software or plugin installed, including mobile browsers.

A video should be scaled appropriately to the size of the player and device's screen resolution. The lines should have the same shape they had as the author saw them while he was recording the screencast.

The user can pause and continue with the playback of the video at any time. User can skip to any point of the video either forward or backward.

If the author of the video had recorded his voice, it will be played along with the video. The audio must be synchronized with the video whenever user plays or pauses the video or when he skips to a different point of the timeline.

The user interface should be intuitive and easy to use either on a desktop computer using mouse and keyboard, but also with touchscreens on mobile devices.

3. Analysis

Before we start implementing the Vector Screencast library, it is necessary to make several important decisions that will define the course of the implementation.

These decisions include

- choosing a frontend platform on which the library will be built,
- choosing or defining an algorithm for lines drawing,
- choosing a way of uploading large amounts of data recorded from the microphone,
- choosing or defining a file format for the recorded data,
- choosing a backend platform which will handle the uploading of results.

3.1 Available techonologies

The web is a huge and fast growing environment. In only a few years, it has become a universal place for presenting and exchanging information. This put web in the focus of many software companies and organizations and as a result many different technologies for developing *Rich Internet Applications* (RIA) have been created. Some of them have already faded into obscurity, other are just emerging. One of the main limitations in the selection of the right technology for developing a web application is their compatibility with different operating systems and web browsers.

3.1.1 Java applets

Java applets are used for creating interactive applications within web browsers [21]. Java applets meet all the specified technical requirements of both the player and the recorder.

Java applets are written in any language that can be compiled into bytecode; this bytecode is then downloaded into the web browser and then runs using *Java Virtual Machine* (JVM). This means that to be able to run a Java applet, user needs to have JVM installed on the device and a Java plugin installed and allowed in his web browser. There is no support for mobile operating systems such as iOS and Android [22], and Google Chrome has Java disabled by default in most recent versions for security reasons.

3.1.2 Adobe Flash

Adobe Flash is a multimedia and software platform used for creating vector graphics, animations and games. Flash has all required features: vector graphics manipulation, working with Extensible Markup Language (XML), mouse input capturing, microphone input, and audio streaming [23].

To view Flash animations or to execute Flash applications, Adobe Flash Player is needed. Adobe Flash Player is available and being developed for all major operating systems, although that is not true for mobile platforms. There was never any support for Apple iOS [24] and in 2012 the development of Flash for Android was discontinued [25]. Using Adobe Flash would mean to exclude most users of tablets and smartphones [27] which is a large disadvantage of this technology.

3.1.3 Microsoft Silverlight

Microsoft Silverlight [28] can be used for creating web applications. It is based on the .NET Framework and it is similar to Java applets and Adobe Flash. It was Microsoft's attempt to compete Adobe Flash, but has not gained much popularity. Development of Silverlight was also discontinued by Microsoft in 2012 and it will not be supported in Microsoft's new web browser Microsoft Edge[29]. These facts make Silverlight unsuitable for this project.

3.1.4 HTML5

HTML5 is the fifth revision of *HyperText Markup Language* (HTML) standard by the *World Wide Web Consortium* (W3C). It has been given the Recommendation status at the end of 2014 and all crutial aspects of both tools can be implemented using the proposed standard. Tracking mouse was long supported even in HTML and ECMAScript/JavaScript[1] through *Document Object Model* (DOM) Events [40].

Vector graphics are supported through the *Scalable Vector Graphics* (SVG) format [32] and it can be manipulated through DOM API [39], as well as any other XML content. MediaStream API enables access to audio input from the user's microphone. The `<audio>` tag can be used to stream audio files and replay them in a web browser.

While HTML5 is a new standard, many of the most important features have already been implemented in some web browsers, like Google Chrome and Mozilla Firefox. More specifically, all the features needed to implement the screencast player are supported in the latest versions of all major web browsers. The only catch might be the disagreement on supported audio file formats among the creators of web browsers. This can be overcome by converting the audio into several used formats and passing them to the browser which will then choose the one it supports.

Some of the features needed by the recording tool, like the MediaStream API, are not yet supported by all web browsers, but this won't be an issue for content creators who can easily install a supported web browser on all desktop platforms and even some mobile ones. A fast development in this area is expected and these features will be most likely soon implemented in the next versions of all major browsers.

[1]ECMAScript is familiarly known as JavaScript after its most common implementation [31]

Scalable Vector Graphics (SVG) and Synchronized Multimedia Integration Language (SMIL)

SVG SVG is an XML based file format designed for describing two-dimensional vector images [32]. It is an open format developed and maintained by the *W3C SVG Working Group.* Current W3C Recommendation is SVG 1.1 (Second Edition).

SMIL *SMIL* is an estabelished standard for animation of SVG images using a declarative approach. It is an extension of SVG specification that lets the user to define key-frame animations of SVG elements' attributes. SMIL is not supported by all major browsers[2]. SMIL's future is unsure at the moment, as the Blink rendering engine development team has announced that they will deprecate the technology in Chromium and Google Chrome in favor of CSS animations and the *Web Animation API* [33].

Web Animation API

A very promising technology is the *Web Animation API* [34] that will provide a very effective way of declaring and controlling animations in a web page through a JavaScript API. This technology is currently (3rd July 2015) in the state of Editor's Draft. This means that it is provided for discussion only and may change at any moment [35].

Unfortunatelly, at the time of writing this thesis, it is not supported by several major browsers and is only partially supported by the remaining few [36] [37].

3.1.5 Conclusion

When looking at table 3.1, the bottleneck of most of the technologies is their support in mobile devices. These devices do not allow some of the above mentioned technologies to run inside them. As the popularity and market share of mobile devices is growing, their support them is a top priority. HTML5 APIs features needed to create the screencast player are implemented in all major web browsers, including browsers for mobile devices. Audio recording is not supported by all major browsers at the moment, but this will change with the release of Microsoft Edge [30]. After considering these facts, HTML5 was selected for the implementation of both the recording tool and the screencast player.

3.1.6 Possible issues and known limitations

As mentioned earlier, not all browsers implement all features described in the HTML5 specification in their latest versions. But the trend is to implement as many technologies as possible to give better browsing experience to users to gain bigger market share, therefore this situation will get better soon.

Another fact, that might cause trouble, is that many users still use an old version of their browsers. This is mainly the case of Microsoft Internet Explorer (MSIE) which does not have automatic update mechanism. Fortunatelly, the

[2]https://status.modern.ie/svgsmilanimation

number of MSIE users is dropping and Microsoft Edge, the successor of MSIE, will have automatic updates built in.

So far, there has been no consensus on formats and codecs support in HTML5 *WebAudio API*. The most supported format across web browsers is MP3. This format is not supported by some of the browsers on some platforms due to licence reasons. An up-to-date table with media formats' support can be found at *Mozzila Developer Network* (MDN) website [38]. This inconvenience can be easily overcome by providing several `<source>` elements to the `<audio>` element with different formats. The browser will then select the format it supports and it will download and play this audio track. The flaw is the need of having the data stored in several formats on the server and the video as a whole takes more disk capacity.

	Adobe Flash	**Java applets**	**Microsoft Silverlight**	**HTML5**
drawing vector primitives	**Yes**	**Yes**	**Yes**	**Yes**
microphone input capturing	**Yes**	**Yes**	**Yes**	**Yes**
reading and creating XML documents	**Yes**	**Yes**	**Yes**	**Yes**
Wacom tablet pressure sensitivity	**Yes**	**Yes**	**Yes**	*Some*[3]
player features supported by desktop browsers	**Yes**	*Some* [4]	*Some* [5]	**Yes**
player features supported by mobile browsers	None	None	None	**Yes**
recorder features supported by desktop browsers	**Yes**	*Some* [4]	*Some* [5]	*Some*[6]
recorder features supported by mobile browsers	None	None	None	*Some*[7]

Table 3.1: Comparing capabilities of available technologies and their support in web browsers

[3]Browsers supporting Wacom WebPAPI or Pointer Events API
[4]Java is disabled by default in Google Chrome
[5]Only some browsers and operating systems are supported.
[6]MS Edge will support *getUserMedia*, Safari does not support *getUserMedia* yet
[7]*getUserMedia* is implemented only in Google Chrome for Android

3.2 Drawing algorithm

One of the key aspects of success of this library is the way it draws the lines. The lines must be nice, smooth and feel natural, as if somebody physically drew them on a blackboard with a chalk, and must not be disturbing or distracting. This library cannot improve the contents of the author's video, but the way it is displayed to the user can have a possitive impact on the viewer's impression of the video. It is also required that the stylus pressure has a direct proportion to the width of the drawn line.

After a research, the algorithm called "Dynadraw" by Paul Haeberli [49] from 1989, seems appropriate for this project. It is used for example in *Inkscape* for the calligraphic tool [50].

This algorithm models the tip of the brush as a physical object and simulates its movement. The user exerts force on the brush by moving the mouse or any other pointing device. The original algorithm is designed for calligraphic strokes and uses mouse speed to calculate dynamic line widths. This technique makes lines drawn by a mouse or other pressure-insensitive device more interesting and can be combined with pressure level of pressure-sensitive device.

For the details of the implementation see section 5.5.2 on page 25.

3.3 Audio recording

The audio input from the microphone will have the form of raw uncompressed *Pulse-code modulated* (PCM) data [60]. The sample rate is chosen by the web browser and cannot be changed by the programmer. The sample rate must be between 22.05 and 96 kHz. If you choose the bit depth of 16 bits per sample, then you can calculate the minimum data load of uncompressed audio recording. Therefore you receive at least 44.1 kB of data per second which 2.646 MB of data per minute of recording. From experience, desktop browsers target 44.1 kHz sampling rate which even doubles the data load. The data must be uploaded to the server so that users can watch the screencasts with a voice commentary.

Buffering microphone input The first possible approach is to buffer all the data from the microphone, create an uncompressed audio file and upload this file to the server at the end of the recording. Creating a *Waveform Audio File Format* (WAVE or WAV) file is relatively simple because it contains only a simple header and LPCM data.

Compression of buffered data in the browser The uncompressed audio is relatively large and the upload will take much more time, than the uploading of a compressed file would take. An audio compression is a complex process. There are several JavaScript libraries that implement the audio compression inside a web browser[8][9], but they are large, slow and unstable.

[8] http://bgrins.github.io/videoconverter.js/

[9] https://github.com/akrennmair/libmp3lame-js

Streaming microphone data to the server in background A different method is possible with the use of the *WebSocket protocol*. The WebSocket protocol enables two-way communication between a client and a server [62]. It is built on top of *Transmission Control Protocol* (TCP) sockets which means it provides ordered and reliable transmission of messages. Either textual or binary messages can be sent over a WebSocket. If the data is transfered over the WebSocket at the time of recording in background, then a considerable part of the data might be already uploaded to the server as soon as the user stops recording, depending on his Internet connection upstream speed.

3.3.1 Conclusion

Streaming the recorded audio data in background over a WebSocket seems to be convenient for the user. It will fasten the uploading process at the end of recording, as part of the audio data is already uploaded before the user even clicks the upload button.

The server must implement WebSocket connections and serve clients. Server might also use a native program, like the open-source FFmpeg[10] to compress the audio and convert it to different file formats.

3.4 File format

All the information of the cursor movement and the pre-drawn lines must be stored on the server, including the time at which every single segment of every line is drawn. For this purpose an appropriate file format must be used. There are a few existing formats which were designed for vector graphics and animation.

EVA The *Extended Vector Animation* (EVA) by SHARP [51] is a binary file format developed in 1996. It is used to create vector animations and it is popular in Japan. Unfortunately the documentation and resources are available in Japanese only [52].

SWF Another well-known format is Adobe's *SWF*. It is a binary format with a wide range of usage. It can contain bitmap-based video, vector primitives, and ActionScript [53]. It is possible to manipulate contained vector primitives and morph their properties to create a key-frame animation [26]. The documentation of the format is open since 2008.

SVG *Scalable Vector Graphics* was already mentioned. This format can contain SMIL animation definitions which makes it also a vector animation format. SVG is based on XML and it is therefore a human-readable textual format. As discussed earlier, SMIL animations are not considered to be a good option for this project.

[10]https://www.ffmpeg.org

Custom extension of SVG JavaScript is not very good at manipulating binary files. Until the specification of typed arrays, manipulating binary files in JavaScript had beed very ineffective and unintuitive. Even with these new technologies, working with XML in JavaScript is more straighforward. The fact, that SVG is based on XML means, that it can be extended with custom namespace and define custom animation, that will be better for this project.

3.4.1 Conclusion

After considering available options, creating a custom format based on SVG seems like the best option for its human-readability, simplicity, versatility, ease of manipulation using JavaScript, and the possibility of previewing static scenes of the animation in existing software. This format will serve well as a proof-of-concept and can be replaced with a more sophisticated format in the future and the format must not be therefore hard-coded into the library.

For the specification of this new format, see chapter 4 on page 17.

3.5 Backend

An important part of every website is its backend. Implementation of the backend is not a part of the Vector Screencast library, but this library needs to have compatible processes running on the web server – an *HTTP web server* and *Audio Recording Server*.

3.5.1 HTTP Web Server

An *HTTP web server* is a server process listens for incomming HTTP requests on well-known port 80 or 443 in case of secured connection. The library should be easily integratable into any web server as it has only a few requirements.

Serving static files The server should respond to HTTP GET requests, whose URL corresponds to a file location in the public section of the file system, by serving this file. To enable replaying and skipping parts of audio files, this server must understand the *Content-Range* header of HTTP requests [20]. This is a basic feature of common web servers such as *Apache*, *IIS* or *Express.js*. Any of the mentioned servers or a similar one might be used.

Handling upload of the created SVG file When the recording is finsihed, the result needs to be stored on the server. The web server must therefore be able to handle an HTTP POST request with the uploaded file and information about its type on a specific URL. It is usually possible to define this behavior in some scripting language supported by the server. Handling file uploads is a typical task in web applications and the process should be well documented.

3.5.2 Audio Recording Server

The Vector Screencast library will try to upload as much audio data it receives from the microphone to the server as possible to fasten the process of uploading at the end of recording. A compatible server process must receive this stream of PCM data and store them into an uncompressed WAV file.

WebSocket specification is relatively new and there might not yet be existing libraries for all programming languages. An audio recording server can be a separate process running on the server and listening on a different port. This program can be written for example in JavaScript and run on multiplatform *Node.js*. There are several WebSocket packages available for Node.js on the Internet.

3.5.3 Conclusion

Creating a backend for Vector Screencast is very easy and it can be integrated basically into every existing web application. Administrators of this website must be able to create and run a WebSocket server for audio recording which will run on their platform.

4. Vector Screencast file format

The SVG format is common and is implemented in web browsers and in many programs. These programs can open an SVG image and draw its contents. The idea is to create an SVG file which contains all the information needed using custom namespace attributes and elements and shows the state of the blackboard at the end of recording.

A valid SVG document must have an *svg* root element with specific namespace attributes and specified *width* and *height* attributes. For the purpose of extending the document, a new namespace *http://www.rozsival.com/2015/vector-screencast* is added with the prefix *"a"*. An example of an empty SVG document with this namespace looks as follows:

```
<?xml version="1.0"?>
<svg version="1.1"
        width="470"
        height="100"
        xmlns="http://www.w3.org/2000/svg"
        xmlns:a="http://www.rozsival.com/2015/
           vector-screencast">

</svg>
```

The SVG specification allows inclusion of elements and attributes from foreign namespaces anywhere with the SVG context. Attributes from foreign namespaces might be attached to any element. Both elements and attributes will be included in the DOM by the SVG user agent, but will be ignored otherwise [41].

Vector Screencast `<svg>` root element must have two child elements: `<metadata />` and `<g a:type="chunks" />`

The SVG `<g>` element is intended for grouping related graphics elements. These groups might be also nested.

4.1 Video Metadata

The first child element of the `<svg>` element must be a `<metadata>` element. This element must contain these child elements:

<a:width> and <a:height> The content is the original width and height of the blackboard. These numbers will be used to correct coordinates when playing the video with a different resolution. The `width` and `height` attributes of the `<svg>` element may contain different values.

<a:length> The content is the duration of the video in milliseconds.

<a:audio> Contains the list of audio sources as child elements.

<a:source> Defines one audio source. It has two attributes: `a:src` – containing the URL of the audio source, and `a:type` – containing the MIME type of the audio source.

4.2 Video Chunks

A video is divided into smaller consequent parts, called *chunks*. Chunks have a variable time duration based on the user's behaviour. These chunks are logical units of the video, each of them can be rendered at once as one graphics primitive. This helps optimize skipping parts of the video. Some chunks can be skipped entirely as the canvas will be cleared at some point after the very chunk is rendered, but before the target time point is reached.

There are three types of chunks: *path*, *erase*, and *void* chunk. Each chunk is stored as one SVG group element with a specific attribute `a:type` (*path*, *erase*, *void*) and an attribute `a:t` which is the time, when this chunk starts to be processed, in milliseconds. Chunks contain the prerendered graphics primitive and a list of animation commands.

"Path" This type of chunk represents one line the user draws. The first child element is an SVG `<path>` element. The `fill` attribute should correspond to the real color of the path, but is not neccessary for later screencast playing. The data attribute `d` includes serialized information about the segments of the path in the form of valid SVG path instructions. For more details about the serialization and deserialization of this data, see section 5.6.1 on page 29. One or more child elements might follow after the `<path>` element, all of which must be animation command elements.

"Erase" This type of chunk represents clearing the whole canvas with one color. The first child element is an SVG `<rect>` element. The `fill` attribute should correspond to the real color of the new background. One or more child elements might follow after the `<rect>` element, all of which must be animation command elements.

"Void" This type of chunk does not render anything on the canvas. It might contain one or more child elements, all of which must be animation command elements.

4.3 Animation Commands

Animation commands are necessary for correct actions timing during the video. Chunks contain the information of how the resulting primitive looks like, commands contain the user's gradual forming of these primitives.

An Animation command must be a child element of a chunk element. Every command has an attribute `a:t` with a timestamp of the action in milliseconds. The time value is relative to the time value of its parent chunk – this decreases the number of digits of each time value and thus the number of bytes needed for a textual representation of the value. The value of the time attribute of an element must be greater or equal to the values of preceding sibling action elements. If a command does not have this attribute, it is assumed that the value of this attribute is zero – this decreases the memory size of commands created while the recording is paused.

<`a:m`> *Move cursor* command tells the player to move the cursor to a specific position defined by the `a:x` and `a:y` attributes.

<`a:c`> *Change color* command tells the player to switch the current brush color according to the value of the `a:c` attribute. Value of the attribute must be a valid CSS color value[1].

<`a:s`> *Change brush size* command tells the player to change the current brush size to the value of the `a:w` attribute. The units of this value are pixels and the size must be corrected according to the width and height stated in *metadata*.

<`a:d`> *Draw next segment* will render the next segment of the current path with the current brush color and size. This command does not carry any additional information, all the information about the segment is carried by the *path* chunk. *Draw next segment* commands can be present only in a *path* chunk. There must not be any *change color* or *change brush size* command between the first *draw next segment* command and the last one inside one *path* chunk – color or size cannot be changed in the middle of a line.

[1]http://www.w3.org/TR/CSS2/syndata.html#value-def-color

5. Implementation

This chapter covers interesting parts of the implementation of the Vector Screencast project. The following text summarizes the ideas behind the concepts used in the project and problems that had to be overcome.

5.1 ECMAScript and JavaScript

ECMAScript is a standardized scripting language widely used in website development [55]. The latest approved edition of ECMAScript is *ECMAScript 5.1* which is implemented in most major web browsers. Implementations of ECMAScript in web browsers are commonly called JavaScript.

JavaScript is a dynamic programming language combining multiple aspects of imperative, functional, and object-oriented programming.

Functions are first-class citizens. This means they can be stored in variables, passed as function parameters, returned as results of functions, and included in data structures.

JavaScript doesn't provide any means of static type checking. All types are created during runtime and they are also checked only during runtime. The lack of static type checking during compilation might lead to rarely occurring errors and it is important to take this in mind while writing a JavaScript code. A good practice is to write documentation comments[1], where the types are specified.

JavaScript is an interpreted language, some implementations also use a *Just-In-Time* (JIT) compilation[2] for a better performance. Another very important aspect of ECMAScript which affects its performance is the absence of direct control over memory usage – memory is released when it is not needed any more. This mechanism is called *Garbage collection*. As of 2012, all modern browsers use a mark-and-sweep garbage collector [43].

Object oriented programming (OOP) in JavaScript differs from class-oriented OOP in the way inheritance is implemented. While in class-oriented languages, for example in C++ or C#, inheritance is achieved by declaring classes of objects, in JavaScript, OOP is implemented through object prototypes. A prototype is just a link to another object which has another prototype. A prototype chain is created this way. The last object in this chain has a `null` prototype. When trying to access a property of an object, it is searched in its own properties. If it is not found, then it is searched in its prototype's properties and so on until the end of the prototype chain is reached [42].

5.1.1 TypeScript

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript [44]. It is an open-source project developed by Microsoft. It is, as its name suggests, a strongly typed programming language compatible with JavaScript.

TypeScript extends capabilities of JavaScript by static type checking at the time of compilation which helps the programmer to find errors in the source code

[1]Commonly used syntax in JavaScript projects is JSDoc (http://usejsdoc.org/)

[2]For example Google V8 engine used in Google Chrome. See https://code.google.com/p/v8/

sooner, and speeds up the process of coding. TypeScript also introduces *class* semantics. These classes are transpiled into JavaScript prototypes. TypeScript also includes concepts of interfaces and polymorphism which makes programs written in TypeScript more understandable to programmers familiar with other object-oriented programming languages like Java, C# or C++.

TypeScript uses several features of ECMAScript 6, but it is transpiled into ECMAScript 5.1, and therefore does not bring any new functionality. The reason for choosing TypeScript is the clarity of code and more convenient development process for the programmer.

5.2 Event driven programming

The idea behind event driven programming is to break direct references between objects and to communicate with *events* instead of calling object methods directly. The advantage of the this approach is *loose coupling* [45] – features might be added or removed without breaking the core of the application.

Each object handles only its own task without knowing anything about the other objects it is collaborating with. When an object completes its task, it publishes the result with a specific event. Objects subscribed for this event are notified and are given the outcomes of the previous task.

This mechanism is sometimes called the *Event Aggregator* design pattern. It is implemented through the `VideoEvents` class. This class provides an interface for registering and triggering callbacks for specified events. It is worth mentioning that web browsers execute all scripts in a single thread and the `trigger` function blocks the UI thread untill all the attached callbacks run to the end.

Each instance of the `Player` class or the `Recorder` class has its own instance of the `VideoEvents` class and this instance is publicly accessible for reading. This means that several screencasts might be played or recorded simultaneously without affecting each other and that the playback or recording of a screencast might be examined or influenced from the outside of the library itself.

An example of `VideoEvents` usage is shown in the following snippet of JavaScript code:

```javascript
var VET = VideoEventType;
var events = new VideoEvents();

events.on(VET.Message, function(message) {
  console.log("received message:", message);
});

events.on(VET.Message, function(message) {
  console.log("received message backwards:",
     message.split("").reverse().join(""));
});

events.trigger(VET.Message, "Hello world.");
```

The list of all event types can be found in the enclosed API documentation of the `VideoEventType` enumeration[3].

[3]The "Message" event used in the example is not used in the Vector Screencast project and

5.3 HTML5

HTML5 is a term used to refer to modern web technologies. The core is the *HyperText Markup Language*, designed to describe semantics of structured docu ents [46]. HTML5 extends the semantics of HTML 4.1 with new HTML tags such as `<header>` or `<footer>`, but also defines a huge set of new APIs which allow web developers to create much richer websites and web applications. Some of these new technologies are needed by the Vector Screencasts project.

Working with XML documents Working with XML data is very similar to working with regular website DOM in JavaScript. The `Document` interface is used for traversing the XML tree, modifying it or for creating a new one. In HTML5, XML files can be opened using a HTTP GET request via the `XMLHttpRequest` object which parses the data and returns a `Document` instance in its `responseXML` property if the document meets criteria specified in [48].

WebSockets WebSockets allow applications to open bidirectional communication channels with server-side processes [62]. WebSockets are built on top of the *Transmission Control Protocol* (TCP) connection. This means that the delivery of data is reliable and ordered.

Web Workers JavaScript code of a website normaly runs in a single thread. It is common to run functions asynchronously, for example callback functions and event handlers, but all these functions still run in the same thread. Web Workers are a means of running heavy tasks in the background without affecting the user interface. A real OS-level thread is spawn for each `Worker` object instance.

Web Workers have several limitations. They cannot change the DOM and have direct links to objects in the main thread. Web workers receive messages from the parent thread through an `onmessage` event and can send a response via the `postMessage` function. These messages contain serialized JavaScript objects which cannot contain any references. As a result, concurrency problems are not typically an issue. Web Workers can perform HTTP requests using the `XMLHttpRequest` object, but the content is not parsed if the target is an XML or HTML file. For more details see the specification of the Worker object.

5.4 Screencast playback

The screencast player uses a "stopwatch" – high resolution timer – to keep track of current video time. The key job of the screencast player is to synchronize the state of the canvas with the state of the recorded video according to current time.

The simplest way to synchronize the video is to sequentially execute video commands, whose timestamp is lesser or equal to current time. This process must be repeated periodically whenever the video is started or unpaused. The optimal frequency of an animation is 60 Hz. This frame rate can be targeted by periodically running the `requestAnimationFrame` method.

was used only as an illustration.

Executing commands consequently will become ineffective, when the user skips to a random part of the video. Naive implementation would set the timer to the chosen time and for a point in the future, execute all the commands, until the states are synchronized. For a point in the past, the blackboard would be cleared and the initial state will be synchronized with the state at the chosen time.

This approach can be optimalised by looking at the last time, when the board was cleared before the chosen time. Synchronization must then start at this point. This means looking up the chunk, into which the time steps, and looking up the previous "erase" chunk. This can be done at the time of parsing the screencast input file.

Each chunk represents a graphical primitive and can be rendered at once. This can be used to render all the chunks but the last one during the synchronization. The commands for changing the current brush size and the current brush color are skipped, therefore each chunk must remember only the color and the brush size at the moment its processing starts.

Most of the commands are cursor movement commands – the cursor is moved only to its final position during one synchronization step. This reduces the extra triggered events.

5.5 Line drawing

Khan Academy videos are known for their consistent and simple style. A person draws on a virtual canvas (evoking a school blackboard) with a brush (or possibly a chalk) of a round shape.

A tutor has a pointing device, typically a computer mouse or a digital stylus, and its position on the canvas is marked with a moving cursor. When the tutor clicks, a dot is marked on the canvas in the current position of the cursor. The tutor can produce a line (typically a curve) following this cursor when he presses a mouse button or increases the digital stylus pressure while moving the cursor. The curve ends when he releases the button or lowers the stylus pressure. The color and the size of the dot or line corresponds to the current settings. The pressed mouse button corresponds to the maximum applied pressure and the released mouse button to no pressure.

Rendering at the time of playback gives us the opportunity to adjust the outcome to the environment of the end user. This means that the result can be sharp on a screen of any resolution without the need of having multiple versions of the same video for each resolution.

Rendering graphics using HTML5

Displaying text and static visual content is the main purpose of older versions of HTML and *Cascade Style Sheets* (CSS). Creating complex polygons or curves would be very hard, involve various tricks [4] or would be even impossible.

[4] For example icon shapes can be created using pure CSS: http://nicolasgallagher.com/pure-css-gui-icons/

The new HTML specification takes this in mind and provides ways of creating a richer and more dynamic content within a web page. There are two technologies that should be taken into account – *Canvas 2D Context* and *SVG*.

Canvas 2D Context The `<canvas>` element provides scripts with a resolution-dependent bitmap canvas which can be used for rendering graphs, game graphics, or other visual images on the fly [47].

Using canvas seems appropriate for this project. Canvas could be created with respect to the user's resolution and the web browser window size. All elements can be scaled to fit this viewport. This will make them look sharp and there will not be any artefacts, noise or blur caused by interpolation used for scaling a regular bitmap video content.

When the dimensions of the `<canvas>` element change, the image is scaled, but the output will not be very sharp. The canvas contains a bitmap image, onto which the graphics primitives are drawn. All these primitives must be redrawn by the programmer to achieve smooth scaling.

SVG SVG can be used for animation inside a website by adding the `<svg>` element to the DOM. SVG elements are then added, deleted, and modified like any other DOM content. When the content of the SVG element changes or some attributes of SVG elements change, the image is redrawn automatically. SVG content is also scaled automatically when the size of the root `<svg>` element changes. It is important to set correct value of `viewBox` attribute. The value should correspond to the size of the original canvas.

WebGL *WebGL* is a JavaScript API for rendering 3D and 2D graphics in a webpage. It is based on OpenGL ES 2.0. A vertex and a fragment shader code written in HLSL is executed directly on the user's *Graphics Processing Unit* (GPU).

5.5.1 Drawing strategy

There are several methods of rendering graphics in HTML5. Two of them – Canvas 2D Context and SVG – are implemented in the Vector Screencasst library. They are not hard-coded into the `Player` or `Recorder`. The user can specify the strategy he wants to use when he creates the object of the tool. The Canvas 2D Context is the default drawing strategy.

If someone wants to use a different rendering method, for example *WebGL*, or rewrite one of the already implemented strategies, he or she might write an own class which would implement the `{DrawingStrategy` interface and a class extending the `{Path` class. One could use this rendering method in his project without modifying the source code of the library.

Drawing strategy receives segments which were generated by the Dynadraw algorithm earlier, and renders them with the current color. Both ends of a line are ended with a rounded cap.

5.5.2 Dynadraw algorithm

Paul Haeberli has created a simple algorithm called *"Dynadraw"* in 1989 which is suitable for calligraphy. Brush is modeled as a physical object with its *mass*, *velocity* and *friction coefficient* [49]. The mouse movement is interpreted as a way of exerting a force on the brush – the faster you move the brush, the greater the force applied on the brush. Acceleration is then calculated according to Newton's second law of motion considering the mass of the brush. The velocity of the brush is derived with respect to the value of the friction coefficient. This velocity is then applied and the brush is moved. The trace brush should leave behind is then drawn onto the virtual canvas.

The advantage of this algorithm is its simplicity and the possibilities of configuring the brush with different values of mass and friction (the author of the algorithm refers to this constant also as *drag* which better fits the purpose of slowing down the brush).

Heavier brushes move slower than the light ones, but the path they leave behind is much smoother, as the shaking of a hand is eliminated by the composition of forces in different directions.

Light brushes move faster and are often very close to the cursor during the movement. When the cursor stops abruptly, light brushes with little friction tend to keep moving past the cursor and wrap around it. This produces little curls at the end of lines.

Brush movement simulation

One step of the simulation applies a force on the brush according to the current mouse position and thus moves it in the direction of the cursor. This process of applying a force must be done periodically, at the frequency of 60 Hz in an ideal case[5]. Implementation of this simulation is not identical to the original *Dynadraw* algorithm, but all of its key aspects are preserved. Pseudocode 1 describes the algorithm used in this thesis.

Even though it is intuitive to measure the time elapsed between two steps and to correct the brush movement using this time difference, this aspect did not yield good results. When the frequency of `requestAnimationFrame` drops, the user's cursor moves further from the previous point, and the force applied on the brush is bigger. Correcting the force caused the brush to wiggle around the cursor wildly.

It turned out to be sufficient to apply the force as frequently as possible, targeting 60 Hz, but not to calculate the speed based on the elapsed time. Animation of a line drawing recorded at a lower frequency feels less smooth and the line lags behind the cursor more distinctively, than at the ideal frequency. Even though the result looks subjectively better and is more consistent than with time corrections.

The main difference between the original implementation and the one used in Vector Screencast project is the brush width calculation. The original algorithm calculates the width of the line in a specific point by measuring its velocity – the

[5]HTML5 provides requestAnimationFrame function which is intended for animations and which targets 60 frames per second

faster the brush is moving, the thinner the drawn line is. This width dynamics gave the algorithm its name.

In our implementation, this effect is preserved, but it is much more subtler. It can be disabled entirely, if the user does not want this feature. The brush dynamics relies mainly on the pressure of a digital stylus on a graphics tablet. Since the exact value of the pressure in the point of the current location of the brush is not always known, the value is linearly interpolated between the value of the pressure in the previous position of the brush and the value of the pressure in the current mouse position according to the distance from each of these points.

As this simulation is not deterministic, this process cannot be reconstructed afterwards. When the video is being played, only the precomputed values of path segments must be used. The precise value of the pressure is therefore irrelevant for later playback of the screencast and does not have to be stored. The advantage of storing this values is the possibility of reconstructing the process of recording later and rendering the lines again. Even a different algorithm might be used in this case.

Pseudocode 1 One step of the brush movement simulation

```
function OneStep($M$)                                 ▷ $M$ - mouse position
    if $M \neq \emptyset$ then
        $brushMoved \leftarrow$ APPLY($\vec{M}$)
        if $brushMoved = true$ then
            DrawSegment
        end if
    end if
end function
function APPLY($M$)
    $\vec{F} \leftarrow M - P$                        ▷ $P$ - current position of the brush
    $\vec{a} = \frac{\vec{F}}{m}$                     ▷ $m$ - mass of the brush
    if $\|\vec{a}\| \leq C_a$ then                    ▷ $C_a$ - minimum acceleration constant
        return false;
    end if
    $\vec{v} \leftarrow \vec{v} + \vec{a}$
    if $\|\vec{v}\| > C_v$ then                       ▷ $C_v$ - minimum velocity constant
        $P \leftarrow P + \mu\vec{v}$                 ▷ $\mu$ - coefficient of friction
        return true
    end if
    return false
end function
```

Rendering of one line segment

A single line consists of many segments that are drawn after every simulation step which moves the brush. There are several ways to draw the segment. The most straightforward is to draw a simple quadrilateral as shown in Figure 5.1 and pseudocode 2.

The curves drawn using quadrilaterals are not very smooth. For smoother curves, straight lines must be replaced with interpolation splines. Both SVG

Pseudocode 2 Draw one segment of a line

```
function DrawSegment
    $\vec{n} \leftarrow \frac{(-\vec{v}_y, \vec{v}_x)}{\|\vec{v}\|}$
    $w \leftarrow$ CurrentBrushPressure $\cdot\, b$                ▷ $b$ - brush size
    $L \leftarrow P - w\vec{n}$
    $R \leftarrow P + w\vec{n}$
    BeginPath
    MoveTo($L'$)                  ▷ $L'$ and $R'$ - previousely drawn point
    LineTo($R'$)
    LineTo($R$)
    LineTo($L$)
    ClosePath
    Fill(c)                                   ▷ c - current brush color
    $L' \leftarrow L$
    $R' \leftarrow R$
end function
```

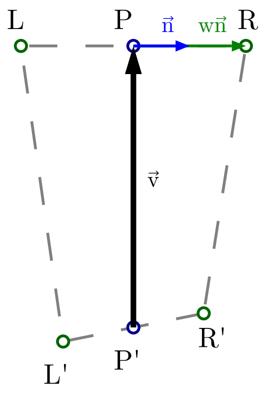


Figure 5.1: Drawn quadrilateral segment of a line

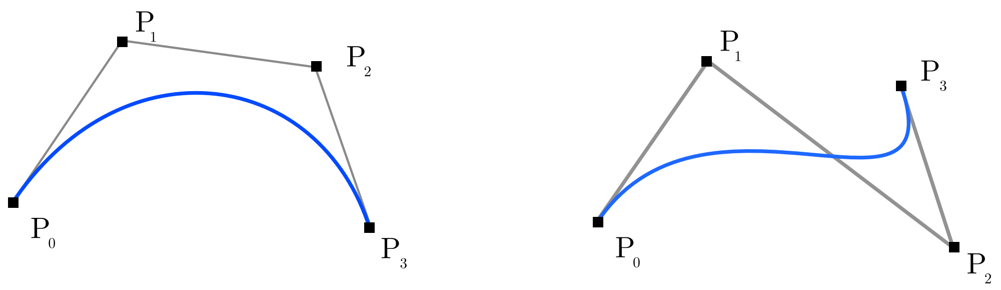


Figure 5.2: Bézier curve interpolation examples

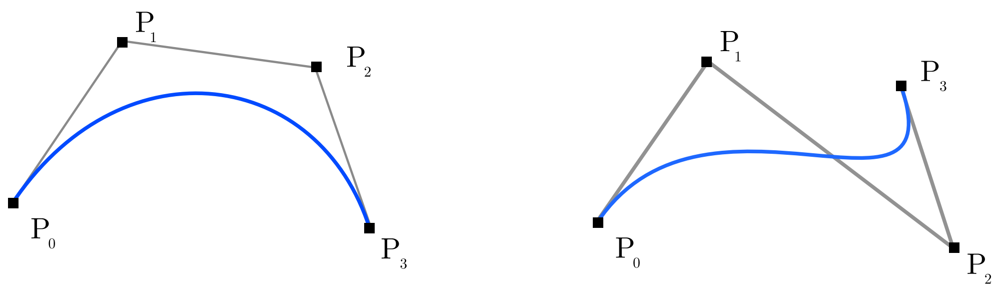


Figure 5.3: Bézier curve interpolation examples

and Canvas 2D Context implement *cubic Bézier curves*. Cubic Bézier curves are defined by four control points, $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$. The path starts in $\mathbf{P}_0$ and ends in $\mathbf{P}_3$, it does not usually go through points $\mathbf{P}_1, \mathbf{P}_2$. The interpolation forumla of the curve [63] is

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)^2 t^2\mathbf{P}_2 + t^3\mathbf{P}_3, \quad 0 \leq t \leq 1$$

To calculate the control points of a cubic Bézier curve for the segment between points $\mathbf{X}_i$ and $\mathbf{X}_{i+1}$, calculated by the DynaDraw algorithm, we can also take points $\mathbf{X}_{i-1}$ and $\mathbf{X}_{i+2}$ and look at these four points as *Catmull-Rom spline* control points. Catmull-Rom is a special type of *Cardinal spline*, with the tension parameter $\tau = 0$ [64]. This approach gives us a nice smooth curve calculated just from the consequent points calcuated by the Dynadraw algorithm.

A special conversion matrix between Catmull-Rom spline and Bézier curve is defined [65] and so the conversion is very straighforward. The formula for calculating cubic Bézier control points $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$, from the points $\mathbf{X}_{i-1}, \mathbf{X}_i, \mathbf{X}_{i+1}, \mathbf{X}_{i+2}$ is

$$\begin{pmatrix} \mathbf{P}_0 & \mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_3 \end{pmatrix} = \frac{1}{6} \begin{pmatrix} 0 & 6 & 0 & 0 \\ -1 & 6 & 1 & 0 \\ 0 & 1 & 6 & -1 \\ 0 & 0 & 6 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{X}_{i-1} \\ \mathbf{X}_i \\ \mathbf{X}_{i+1} \\ \mathbf{X}_{i+2} \end{pmatrix}$$

The first and the last segments must be processed in a special way, as there is no preceding point, or following respectively. For an example of cubic Bézier curve, see figure 5.3.

Drawing one segment of the line using Bézier interpolation curves enhances the resulting line distinctively as can be seen in Figure 5.4. The connection of individual segments creates a subjectively much smoother line when the segments are curved and the result also better resembles a line drawn with a marker pen or a chalk.

5.6 Parsing and generating Vector Screencast SVG

Parsing and generating the XML structure of the vector screencast file uses factories and chains of responsibility to avoid large if-else statements or switch blocks [67].

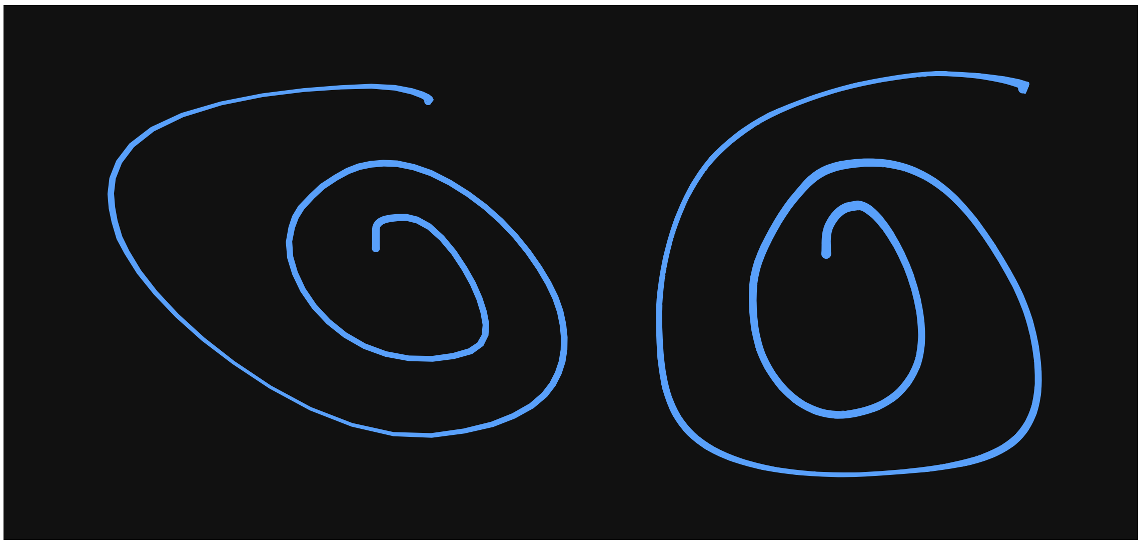


Figure 5.4: A line formed with quadrilaterals (on the left) and a line formed using Bézier curves (on the right)

To make processing of the commands and XML elements faster, the factories in the chain are ordered by their expected occurrence – the factory processing cursor movement is the first one in the chain, and the draw-next-segment command factory is the second one.

A similar approach might be used for creating different file format manipulation classes.

5.6.1 SVG path "d" attribute

Path chunks use the SVG `<path>` element to visualize the path shape when the file is opened in a regular SVG editor. Each path is defined as a set of instructions which form its shape and this shape is then filled with as a solid color.

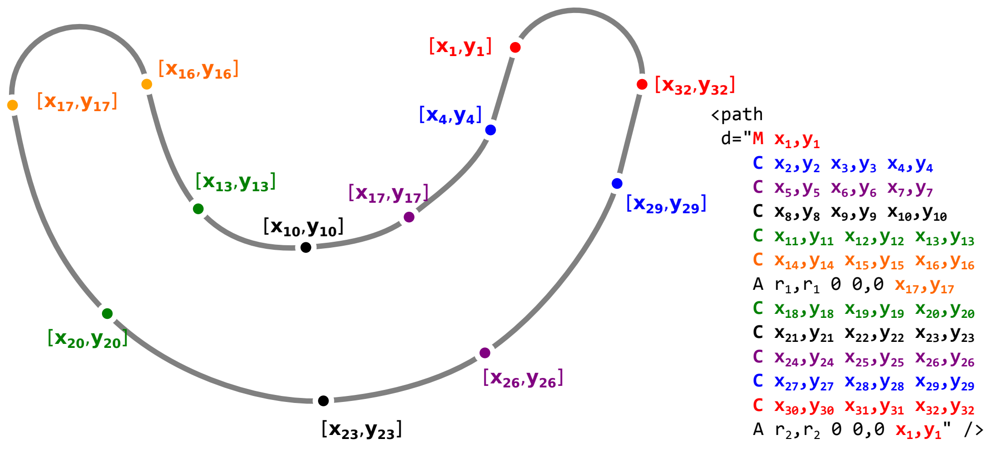
These instructions correspond to path chunks. To save extra instructions, the shape is defined along the circumference. Creating the string of instructions is not difficult, but deserializing is unintuitive – the instructions string must be read forwards and backwards at the same time. For an ilustration see figure 5.5. A chain of factories is used to serialize and deserialize path instructions from the textual representation.

Audio capturing, processing and upload

HTML5 provides only one way to access the microphone data at the moment and it is through the *getUserMedia API* [56]. The `navigator.getUserMedia` function prompts the user to permit the use their audio input[6].

If the user's device has a connected microphone and the user gives his permission to use his audio input, then a `MediaStream` object is provided by the browser and from this time on the audio can be processed. Error callback with `MediaStreamError` instance as a parameter is called otherwise.

[6] |getUserMedia— function is also used to access webcam stream in other applications



```
<path
 d="M x1,y1
    C x2,y2 x3,y3 x4,y4
    C x5,y5 x6,y6 x7,y7
    C x8,y8 x9,y9 x10,y10
    C x11,y11 x12,y12 x13,y13
    C x14,y14 x15,y15 x16,y16
    A r1,r1 0 0,0 x17,y17
    C x18,y18 x19,y19 x20,y20
    C x21,y21 x22,y22 x23,y23
    C x24,y24 x25,y25 x26,y26
    C x27,y27 x28,y28 x29,y29
    C x30,y30 x31,y31 x32,y32
    A r2,r2 0 0,0 x1,y1" />
```

Figure 5.5: SVG path instructions corresponding to path segments

To receive information from the microphone a `ScriptProcessorNode` is created and attached to the `MediaStream` instance. This processing node has an event called `onaudioprocess` which receives data buffers from the microphone. It is very important to have an own reference to the `ScriptProcessorNode` even though some event handlers are attached to its events. The `MediaStream` does not hold any internal reference to this object. Unless there is a reference to the proessor node created by the programmer, garbage collector will destroy the object at some point and the stream of data from the user's microphone will be stopped. This will result in a trimmed audio recording.

Current browsers implement `ScriptProcessorNode` according to the W3C Working Draft from 10 October 2013 of *Web Audio API*. Unfortunately, the `ScriptProcessorNode` interface is deprecated in newer drafts of the specification and should be replaced with `AudioWorkerNote` in the future [58]. This new approach is not yet standardised, implemented and documented in web browsers. Both of these methods provide access to `AudioBuffer` instance which contains *linear pulse-code modulation* (LPCM) [61] sampled data from the microphone [60].

The `Float32Array` buffer containing LPCM data gained from the AudioBuffer is converted to an `Int16Array`. This reduces the amount of data that will be transferred over the Internet.

High Resolution Timer

To make the video look as good as possible, we need to store as precise data as possible. The `Date.now()` function returns the number of milliseconds elapsed since 1 January 1970 00:00:00 UTC. The millisecond accuracy is satisfactory, but modern browsers provide even more accurate data via the *High Resolution Time* via the `window.performance.now()` function with the accuracy of microseconds. The `window.performance.now()` function does not provide data related to current time, but the number of milliseconds elapsed since current page was loaded.

Video Screencast library chooses the most accurate timing function available by the web browser and uses it for making timestamps on recorded data or to control the playback of a screencast.

5.7 User Interface

Both the `Player` and the `Recorder` object instances receive an ID of an empty HTML element. User interface (UI) HTML elements are created by the `PlayerUI` and the `RecorderUI` object instances. These objects create the whole UI, except for the blackboard ("canvas"). The blackboard is generated by an instance of the `DrawingStrategy` interface.

An instance of an object inheriting from `PlayerUI` or `RecorderUI` can be passed to a new instance of the tools in its constructor and override the default structure of the `Player`, or `Recorder` respectively.

UI object instance must map HTML elelements' events to the internal `VideoEvents` and let the user control the recording or playback process.

Input from pointing devices

Detecting mouse movement and the state of its buttons is a very common task in web development. Users navigate through web pages mainly by clicking on hypertext links with their computer mouse. Therefore mouse events are well specified and work across all desktop web browsers and desktop platforms.

Unfortunately, the situation among other pointing devices other than computer mice is much less uniform. With the boom of smartphones and tablets, touchscreens are very common. Also computer graphics tablets are used by artists and many people use them when creating a Khan Academy style video.

Mouse input Event handler receives an instance of `MouseEvent` object as an argument. The relevat information are the `clientX` and `clientY` properties, stating current mouse posititon relative to the position of the element it is attached to.

Wacom Webplugin pen API The Wacom Webplugin pen API (WebPAPI) is a browser plugin interface for pen data access from all Wacom consumer and professional tablets. Unfortunatelly support for this plugin was discontinued by Chromium and Google Chrome [66] and therefore it should not be relied on.

Touch Events API Touch Events API is an API for handling touch input from touch screens. The standard is proposed by Apple and is implemented across many platforms and in many mobile web browsers.

This API supports multiple touches at once, but this feature is not needed and neither implemented in this project. Unfortunately, this API provides no touch pressure information.

Pointer Events API Pointer Events API is an open API created by Microsoft. Its purpose is to unify the way of handling mouse events, touch screen events, stylus and other (i.e. Kinect) similar ways into one API. This technology is implemented in Internet Explorer and will be also present in the final version of the Microsoft Edge browser. Firefox implements this API, but it is so far accessible only if a specific hidden flag is enabled. Google has also announced the intent to implement this functionality in one of the upcoming releases of Google Chrome across all platforms.

This API also provides pressure information for pointing devices, that support this feature, including Wacom graphics tablets.

5.8 Building the library

The library is written in TypeScript which needs to be transpiled into pure JavaScript before it can be used. The theme uses the *Less* CSS preprocesor which also needs to be transpiled before it can be used. To simplify the process of building the library and its components, *Gulp* streaming build system[7] is used.

Environment setup Source code of the project can be found either among the attached files, or in git repository *https://github.com/simonrozsival/vectorvideo*. Copy or clone this code to your computer.

To be able to compile the library, it is necessary to install *Node.js*[8] on your computer. After installing Node.js, install all dependencies of this project by running the `npm install` command in the root directory of the project's source files. You must also install *Gulp* globaly by running the `npm install -g gulp` command.

Building library components Build all code by running `gulp` command in the same directory. This will compile all TypeScript source files into JavaScript. You can then use the compiled and minified library located in *./release/vector-screencast/vector-screencast.min.js* JavaScript file. Along with the library, a Web Worker file *./release/workers/RecordingWorker.js* which is necessary for audio recording. The default CSS theme of the player and the recorder was compiled from LESS source to *./release/themes/default/theme.min.css*.

A demo audio recording server was also compiled in this process and is available in the *./release/audio-server/AudioServer.js* JavaScript file. This server can be used in a Node.js program. For details on how to use the library in your project, see chapter 6 on page 34.

[7]http://gulpjs.com

[8]https://nodejs.org/

`gulp`	Builds all parts of the project.
`gulp release`	Builds all parts of the project and places them in the */release* folder. Sources are minified and are ready for production use.
`gulp demo`	Builds or sources for an example project. Sourcemaps are generated, so this demo is suitable for debugging.
`gulp doc`	Generates API documentation from the TypeScript source files and comments.
`gulp clean`	Deletes all the files that were created by the build.
`gulp clean-release`	Deletes all the files that were created by the release build.
`gulp clean-demo`	Deletes all the files that were created by the build of example project.
`gulp clean-doc`	Deletes the generated API documentation.

Table 5.1: Available gulp commands

6. Using Vector Screencast library

Using Vector Screencast library is intended to be as simple as possible. All the user needs to do is include a JavaScript file of the library and a CSS file of a theme into his HTML code and configure the player in a few lines of the code.

6.1 Obtaining the Vector Screencast library

Compiled library files can be obtained either from the attached files or from the Git repository *https://github.com/simonrozsival/vectorvideo* in the `/release/VectorScreencast` and `/release/themes/` folders. You only need to copy files `vector-screencast.min.js` and `theme-default.min.css` into your project.

The library files must be linked to your document and you should make sure that your website will be displayed properly on mobile devices by specifying the `viewport` meta tag in the `head` section of your document [68]. Also create an empty element with a specific `id` attribute – this will be the container, into which either the screencast player or the recorder will be placed.

An example of a HTML5 template with correct setup can look similarly (irrelevant parts of the document were let omitted and replaced by suspension points):

```html
<!DOCTYPE html>
<html>
  <head>
    ...
    <meta name="viewport" content="width=device-width,
       initial-scale=1">
    <link rel="stylesheet" type="text/css" href="/path/
       to/theme-default.min.css" media="screen,
       projection">
    ...
  </head>
  <body>
    ...
    <div id="some-specific-id"></div>
    ...
    <script src="/path/to/vector-screencast.min.js">
       </script>
  </body>
</html>
```

The initialisation scripts must be executed afte all web page resources are downloaded and the DOM is ready – this can be achieved by putting this code inside a handler of the `window.onload` event.

6.2 Vector Screencast Player

Inside your scripts, create a new instance of `VectorScreencast.Player`. The constructor takes two arguments, first of them is the `id` attribute of a container element and the second is a configuration object. The only obligatory property of the configuration object is the `Source` property – the URL of the source SVG file.

There are several other interesting optional settings that will help you customize the screencast player. One of them is the `Localization` property which takes an object implementing
the `VectorScreencast.Localization.PlayerLocalization` interface. To see the complete list of all configuration options and further details, please refer to the `VectorScreencast.Settings.PlayerSettings` interface in the API reference of the project in the `/docs/` folder of the attached files. You can see an advanced example of `VectorScreencast.Player` usage `/demo/public/play.html` in the attached files.

6.3 Vector Screencast Recorder

Creating a new instance of `VectorScreencast.Recorder`, which enables recording of screencasts in your web page, is similar to creating an instance of the player. The constructor takes two arguments, first of them is the `id` attribute of a container element and the second is a configuration object implementing the `VectorScreencast.Settings.RecorderSettings` interface. For details of the settings objects please read the attached API documentation.

6.3.1 Audio recording server process

The recording tool uploads recorded data through a WebSocket during the recording. You must run a WebSocket server process on your server which will receive audio LPCM data. After the recording is finished, the server sends a response with the names and the MIME types of the audio files which were created during the recording.

The protocol of communication between the client (web browser) and the server is described in the following paragraphs. The URL and the port number on which the server listens must be passed to the instance of the `Recorder` object.

The logical unit of communication between a server and a client is a message. Messages do not have a fixed length and their payload can be either textual or binary. Each message has a flag indicating whether the payload of the message is binary or not.

New recording establishing After a new connection of a client to the WebSocket server is opened, the client sends a textual message containing information about the nature of the audio data that will be sent through the socket later. This message should contain a valid JSON message. This message must be parsed into an object literal. The audio recording server expects the object to have the properties defined in table 6.1. Any other property of the object is ignored.

property name	**value**	**purpose of the property**
`type`	`"start"`	Identification of the message type. The value must be lower-case.
`channels`	*number*	Number of audio channels of the audio track.
`sampleRate`	*number*	Number of samples per second.
`bitDepth`	*number*	Number of bits per sample.

Table 6.1: New recording esabelishing JSON message structure

property name	**value**	**purpose of the property**
`error`	*boolean*	`false` if the data is stored well, `true` if an error occured.
`files`	*array*	Array of the audio files created by the server based on the received data from the client. Each item of the array must have the properties described in table 6.3. This property must be an empty array in case the `error` property is set to true.

Table 6.2: The format of the response of the server at the end of recording

If the server cannot handle the client or the content, the message cannot be parsed, or some required information is missing, the server can close the WebSocket connection and thus rejec the client. Otherwise, the server does not respond anyhow and should await binary messages containing the recorded audio data.

Binary audio data The client can start sending binary messages containing the audio data. This data of all the channels should be already interleaved and the data should correspond to the bit depth that was declared in the first message from the client. The server should not interpret or modify any of the received binary data and should store it directly into an uncompressed WAV file.

End of recording The client can end recording by sending a textual message simillar to the very first message. This message must also contain a serialised object literal in JSON. This object must have a property `type` with the value of `"end"`[1]. Other properties are ignored. The server stops receiving any data from this client. The server must process the data it has received and send a response through the WebSocket. The audio recording should be converted into several audio formats supported by web browsers as a source of the `<audio>` element. The response must be a JSON object with properties defined in table 6.2. After this message is sent, the server can close the connection with the clinet.

Lost connections If the connection of a WebSocket is lost before the communication is finished properly according to the protocol, the server is allowed to throw away all the data received by the WebSocket.

[1]The value must be lower-case.

property name	**value**	**purpose of the property**
`url`	*string*	Absolute URL of the audio file.
`type`	*string*	MIME type of the audio file.

Table 6.3: Format of an information about an audio file

Example An example implementation of the audio recording server is included in the demo project which is described in section 6.6. This implementation is written in TypeScript and runs on Node.js and you can examine its source and use is for testing. The source is placed in the `/src/AudioServer` directory and is used from the `/demo/audio.js` script.

You should be aware of the fact that this implementation of the audio server is only demonstrational and should not be relied on in a heavy production use.

6.4 Embedding Vector Screencast Player

To allow users to embed videos from your website in their websites, create a HTML page, where the player will stretch over the whole screen. Then generate an HTML snippet with an `<iframe>` pointing to your player in the `src` attribute.

6.5 Custom theme

You can customize the look of the player and the recorder to match the design of your website or to change the layout of the controls. Use a custom CSS style sheet to override the default style.

You may want to start by editing the default style of the Vector Screencast. The source files are located in the
`/src/Themes/default` directory. These files are transpiled using the *Less* CSS preprocessor. You can create a new theme by just modifying values of variables in the `/src/Themes/default/variables/variables.less` file.

After you have created your CSS theme, use HTML `<link>` tag to link the CSS stylesheet to your website.

6.6 Demo project

To make life easier, we have prepared an example project, where you can see the library in action. This project shows the use of

- recording tool
- the player
- embedding the player
- demo HTTP server based on Node.js and Express.js
- demo audio recording server based on Node.js

To try the example project, follow section 5.8 on page 32 and build the demo project by running the `gulp demo` command.

When the project is built, start the HTTP server and the audio recording server. To do that, open your shell console and change your working directory to `/demo`. Here, start the servers by executing

```
node server.js 3000 &
node audio.js http://localhost:3000 4000 &
```

This will start the local HTTP server listening on port 3000 and a local audio recording server listening on port 4000. If you need to change port numbers, do not forget to change the port number of the audio recording server also in `/demo/public/record.html`. Both servers will run in the background, but will keep printing log messages into your console window.

Open your web browser and go to `http://localhost:3000`. This will open the main page of the demo project. You can then record videos, upload and play them and embed them into other local HTML files.

Online demo If you do not want to deploy the demo project at your own computer, visit `http://www.rozsival.com` to try the demo online.

7. Users' documentation

The Video Screencast tools were designed to be easy to use and allow users to work with them right away. The following sections summarize the way user can interact with the default UI of both library tools. UI might be changed by the user of the library and the user experience might change. It is then up to the user of the library to explain the specifics of the new interface to the screencast authors and audience.

7.1 Vector Screencast Player

Once the screencast is ready to be played, you can start learning. You should see a simillar interface to the one shown at figure 7.1. The area of the player consists of two main parts – the blackboard and control bar.

Play/Pause/Replay In the bottom left part of the screencast player UI there is a button with a "play" icon. You start or resume the playback by clicking on the button.

When the screencast is playing, the icon changes to a "pause" icon. By clicking on the same button, the playback of the screencast is paused. The icon will then change to the "play" icon.

When the end is reached, the playback stops and the icon changes to a "replay" icon. When you click on the button now, the playback will start over from the very beginning.

Instead of clicking on the button, you may press the spacebar key on your keyboard, if you have one. You can also click or tap the blackboard to play/pause the playback.

Skipping parts of the screencast In the top section of the control bar there is a timeline. It shows the progress of the video. You can click on the timeline to change the current position of the video to the one corresponding to your selection. If you hover your mouse over the timeline, a box with time information of that point will appear. With some touch devices, you might have to tap second time to confirm the selected position.

Instead of clicking on the timeline, you can press left and right arrow keys on your keyboard. Each stroke of the keys will skip five seconds of the screencast forwards or backwards.

Control bar hiding When the screencast is playing, the control bar is not needed and could be hidden. To turn hiding on and off, press the rightmost button on the control bar. You probably would not see the effect immediatelly. The control bar hides only if the video is being played. It is shown automatically when the video is paused or reaches the end. It is shown also whenever you hover your mouse over the remaining visible parts of the control bar.

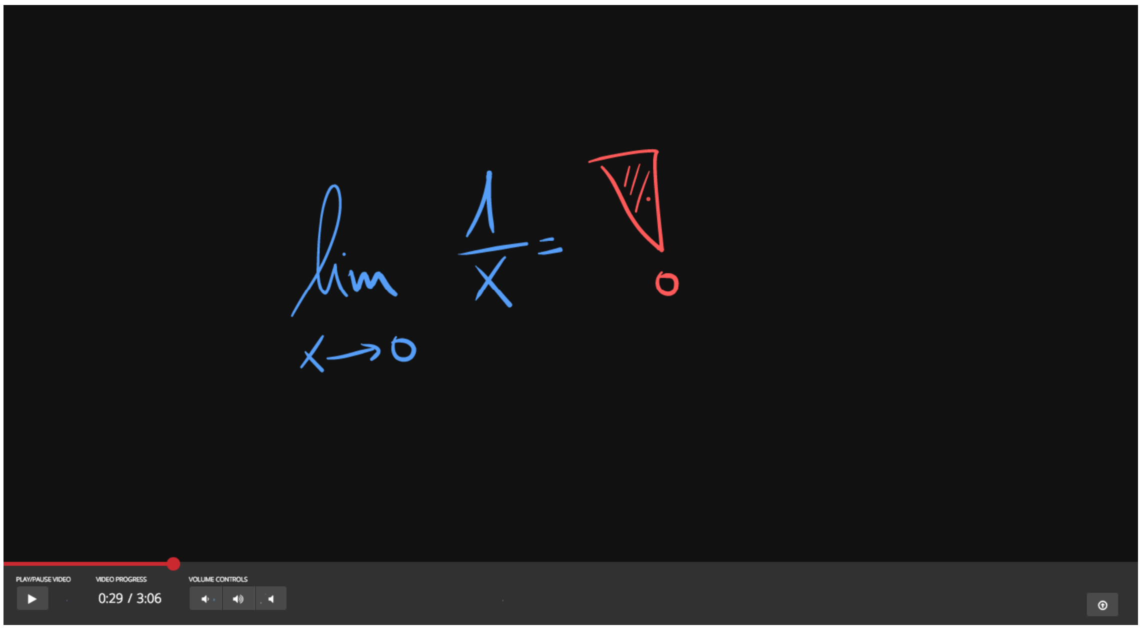


Figure 7.1: The default UI of Vector Screencast Player

Audio controls You can control the audio volume by clicking on buttons in the "Volume controls" section of the control bar. The volume can be decreased, increased or muted/unmuted.

7.2 Vector Screencast Recorder

After you enter the page with the recorder, you should see am interface simillar to the one shown in Figure 7.2. The area of the recording tool consists of two main parts – the blackboard and the control bar.

Microphone access and audio recording Your browser will ask you to allow access to your microphone. Confirming this prompt is neccessary for audio recording. If this message does not appear, your computer probably does not have a microphone, your browser does not support audio recording or you have denied access to your microphone earlier.

A white not crossed out icon of a microphone should appear in the "Audio recording" section of the control panel. If the icon is red, the audio recording is not available. Please make sure you use the latest version of your web browser and check your browser and your operating system microphone settings if you encounter any problems.

If you do not want to record audio or you want to mute the microphone in the middle of recording, press the microphone button. You will resume recording your voice by clicking the button one more time.

Drawing You can start drawing as soon as the audio recording is set up. You can now prepare the inital appearance of the screen. When you click the "Start" button, your cursor movement and voice will start being recorded. You can pause and resume recording at any time and as many times as you want.

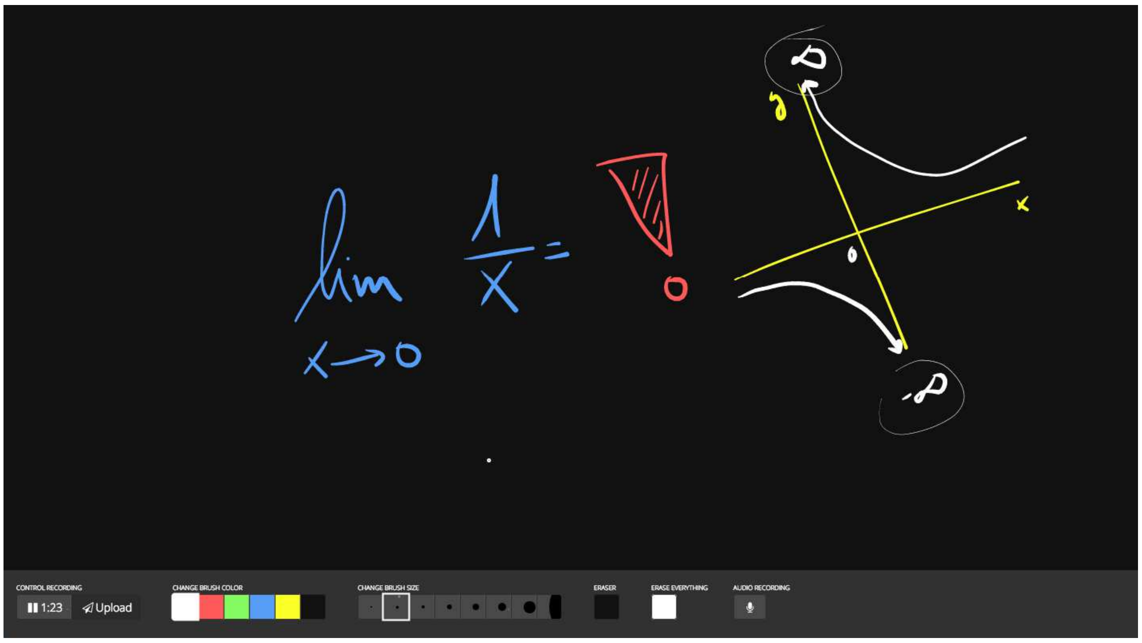


Figure 7.2: The default UI of Vector Screencast Recorder

Before you start drawing a line, select the brush size and the brush color from the palette in the control bar. Draw a line by pressing your mouse, your stylus or touching your touchscreen and moving the cursor around the blackboard. End a line by releasing the mouse button, the stylus pressure or by lifting your finger.

It is recommended to use a pressure-sensitive graphics tablet to achieve the best results. If you use a touchscreen, consider using a specialized stylus instead of fingers to create better drawings.

Erasing If you want to clear the whole blackboard, select a color from the color palette and click on the button in the "Erase everything" section of the control bar. The color of the button should correspond to the color previously selected by you. Everything will be removed from the balckboard and the background color of the "*black*board" will be set to the selected color.

Saving the recording After you have finished recording and the recording is stopped, press the "Upload" button. The data will be uploaded to the server. This process will take some time, even a few minutes, depending on your Internet connection uplink speed and the length of your recording. After the video is uploaded, you will be informed

Conclusion

The goal of this thesis was to create an open-source library which will allow users to create educational websites for creating and playing KSVs. The library uses a vector-based approach. The video is rendered with respect to the user's screen resolution and the size of the viewport.

The project of Vector Screencast is available online as a public Git repository[1]. The library is written in a programming language which should be easy to learn by all web developers. The library uses an object oriented design and is very modular which makes it easy to extend with a custom rendering method, file format support or user interface.

File format Vector Screencast data is stored in a specificaly formed SVG document extended with a foreign namespace. The data contains information of the author's cursor movement and applied pressure and prerendered lines. This type of animation cannot be viewed in any other video player, but a preview of the final state of the blackboard is displayed when the file is opened in a regular SVG viewer. The audio is recorded as an uncompressed WAV file. This file should be converted into one or more common binary audio formats, with respect to web browsers support of audio formats.

7.3 Future work

The library is ready and anyone can use it today in his website for creating and playing KSVs. There are tools for drawing lines of different widths and colors onto a virtual canvas, erasing parts of the canvas and clearing the whole canvas. Some authors might be missing some tools they use in their bitmap editors when recording KSV – including bitmap images or photos, using a text tool or drawing straight lines. It would not be hard to implement these tools, but it would have to be thought over, whether they do not breach the idea of a simple, natural look of KSV.

Binary file format SVG has proven to be a sufficient container for Vector Screencast data. A binary format should be used to achieve more data savings. Most of the data consists of coordinates and time values. Numbers are stored as strings in XML. Each digit character in a UTF-8 document takes up 1 B of data storage. Each component of a coordinate is a number with typical value in the range of 0 to 9999 with several decimal places – the library allows three decimal places at maximum. A typical coordinate component value, like "123.5", takes up more than 4 B, with the decimal dot character included. If these numbers were saved binary as a 32-bit integer, or 32-bit floating point number, it would take up only 4 bytes of memory in total while maintaining the same precission in most cases.

This project was designed to be independent on a specific file format. The library can be easily extended to support different file structures without editing

[1]https://github.com/simonrozsival/vectorvideo

its source code.

Library for native mobile applications The library can be used in any HTML website and playback of a screencast will work well in all modern mobile web browsers. However, users of tablets and smartphones are used to obtaining application in application markets and use them instead of web browsers. Many developers would certainely welcome a Vector Screencast component, they could use in their native Android or iOS app.

Known issues

Audio recording *ScriptProcessorNode* interface is deprecated in W3C Editor's Draft of *Web Audio API* from 21 June 2015 [58]. This interface is used in this project and should be replaced using *Audio Workers*, when the specification is finalized [59].

Demo audio server *Audio recording server* program which is included in the project as a demo should be rewritten. This tool is not very robust and would be ineffective for handling a large number of clients. Users should write the audio server themselves at the moment.

Bibliography

[1] O Khanově škole. *Khanova škola* [online]. [cit. 2015-07-16]. Available from: https://www.khanovaskola.cz/o-skole

[2] Bower, Beverly L. Hardy, Kimberly P. From correspondence to cyberspace: Changes and challenges in distance education. *New Directions for Community Colleges* [online], 2004, 2004(128): 5-12 . DOI: 10.1002/cc.169.

[3] Last Week Tonight with John Oliver: Student Debt (HBO). *YouTube* [online]. 7.9.2014, [cit. 2015-07-09]. Available from: https://www.youtube.com/watch?v=P8pjd1QEA0c

[4] Online, All Students Sit in the Front Row. Gates, Bill. *gatesnotes* [online]. November 18, 2014, [cit. 2015-12-03]. Available from: http://www.gatesnotes.com/Education/Colleges-Without-Walls-Arizona

[5] In the Developing World, MOOCs Start to Get Real. Leber, Jessica. *MIT Technology Review* [online]. March 15, 2013, [cit. 2015-07-16]. Available from: http://www.technologyreview.com/news/512256/in-the-developing-world-moocs-start-to-get-real/

[6] *Kepler kigali* [online]. © 2015, [cit. 2015-07-16]. Available from: http://kepler.org

[7] *Generation Rwanda* [online]. [cit. 2015-07-16]. Available from: http://www.generationrwanda.org

[8] *Moodle* [online]. [cit. 2015-07-16]. Available from: https://moodle.org

[9] Moodle Statistics. *moodle.net* [online]. 2015, [cit. 2015-07-16]. Available from: https://moodle.net/stats

[10] *Coursera* [online]. © 2015, [cit. 2015-03-03]. Available from: https://www.coursera.org/about

[11] *YouTube.com* [online]. © 2015, [cit. 2015-07-16]. Available from: https://www.youtube.com

[12] The top 500 sites on the web. *Alexa* [online]. 2015, [cit. 2015-07-16]. http://www.alexa.com/topsites

[13] *Numberphile Youtube channel* [online]. © 2015, [cit. 2015-07-16]. Available from: https://www.youtube.com/user/numberphile

[14] *Veritasium YouTube channel* [online]. © 2015, [cit. 2015-07-16]. Available from: https://www.youtube.com/user/veritasium

[15] *Khan Academy YouTube channel* [online]. © 2015, [cit. 2015-07-16]. Available from: https://www.youtube.com/user/khanacademy

[16] *Educreations* [online]. © 2015, [cit. 2015-07-15]. Available from: https://www.educreations.com

[17] *ShowMe* [online]. © 2015, [cit. 2015-07-15]. Available from: https://www.showme.com

[18] *Khan Academy* [online]. © 2015, [cit. 2015-07-08]. Available from: https://www.khanacademy.com

[19] *Khanova škola* [online]. [cit. 2015-07-16]. Available from: https://khanovaskola.cz

[20] Hypertext Transfer Protocol (HTTP/1.1): Range Requests [online]. June 2014, [cit. 2015-07-16]. Available from: https://tools.ietf.org/html/rfc7233

[21] Java applet. *Wikipedia, the free encyclopedia* [online]. Last modified on 4 July 2015, [cit. 2015-07-08]. Available from: https://en.wikipedia.org/wiki/Java_applet

[22] How do I get Java for Mobile device?. *Java.com* [online]. [cit. 2015-04-16]. Available from: http://www.java.com/en/download/faq/java_mobile.xml

[23] Adobe Flash Player/Features. *Adobe.com* [online]. © 2015, [cit. 2015-07-08]. Available from: http://www.adobe.com/cz/products/flashplayer/features.html

[24] Thoughts on Flash. JOBS, Steve. *Apple.com* [online]. April 2010, [cit. 2015-04-16]. Available from: http://www.apple.com/hotnews/thoughts-on-flash

[25] An Update on Flash Player and Android. ALJABER, Tareq. *Adobe AIR and Adobe Flash Player Team Blog* [online]. June 28, 2012, [cit. 2015-04-16]. Available from: http://blogs.adobe.com/flashplayer/2012/06/flash-player-and-android-update.html

[26] SWF and AMF Technology Center. *Adobe.com* [online]. © 2015, [cit. 2015-07-08]. Available from: http://www.adobe.com/devnet/swf.html

[27] Mobile Marketing Statistics 2015. BOSOMWORTH, Danyl. *Smart Insights* [online]. January 15, 2015, [cit. 2015-04-16]. Available from: http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics

[28] Get Silverlight 5. *Microsoft Silverlight* [online]. [2015-04-16]. Available from: http://www.microsoft.com/silverlight/

[29] Moving to HTML5 Premium Media. *Microsoft Edge Dev Blog* [online]. July 2, 2015, [cit. 2015-07-08]. Available from: http://blogs.windows.com/msedgedev/2015/07/02/moving-to-html5-premium-media

[30] Announcing media capture functionality in Microsoft Edge. *Microsoft Edge Dev Blog* [online]. May 13, 2015, [cit. 2015-07-16]. Available from: http://blogs.windows.com/msedgedev/2015/05/13/announcing-media-capture-functionality-in-microsoft-edge

[31] A brief history of ECMAScript versions (including Harmony/ES.next). Rauschmayer, Axel. *DZone* [online]. June 28, 2011, [cit. 2015-07-08]. Available from: https://dzone.com/articles/brief-history-ecmascript

[32] Scalable Vector Graphics (SVG) 1.1 (Second Edition). *The World Wide Web Consortium (W3C)* [online]. 16 August 2011, [cit. 2915-07-08]. Available from: http://www.w3.org/TR/SVG/

[33] Web Platform Features: Deprecate SMIL. *Chromium Dashboard* [online]. [cit. 2015-07-16]. Available from: https://www.chromestatus.com/features/5371475380928512

[34] Web Animations 1.0. *The World Wide Web Consortium (W3C)* [online]. 5 June 2014, [cit. 2015-07-16]. Available from: http://www.w3.org/TR/web-animations/

[35] Web Animations. *The World Wide Web Consortium (W3C)* [online]. 16 July 2015, [cit. 2015-07-16]. Available from: http://w3c.github.io/web-animations/

[36] Web Animations JavaScript API. *status.modern.IE* [online]. [cit. 2015-07-16]. Available from: https://status.modern.ie/webanimationsjavascriptapi

[37] Web Animations API. *Can I use?* [online]. [cit. 2015-07-16]. Available from: http://caniuse.com/#feat=web-animation

[38] Media formats supported by the HTML audio and video elements. *Mozilla Developer Network (MDN)* [online]. Last updated on June 9, 2015, [cit. 2015-07-16]. Available from: https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats

[39] Document Object Model. *Wikipedia, the free encyclopedia* [online]. Last modified on 12 June 2015, [cit. 2015-07-08]. Available from: https://en.wikipedia.org/wiki/Document_Object_Model

[40] Document Object Model (DOM) Level 2 Events Specification. Pixley Tom. *The World Wide Web Consortium (W3C)* [online]. 13 November, 2000, [cit. 2015-07-08]. Available from: http://www.w3.org/TR/DOM-Level-2-Events/

[41] Scalable Vector Graphics (SVG) 1.1 (Second Edition). *The World Wide Web Consortium (W3C)* [online]. 16 August 2011, [cit. 2915-07-08]. Available from: http://www.w3.org/TR/SVG/extend.html

[42] Inheritance and the prototype chain. *Mozilla Developer Network (MDN)* [online]. Last updated on April 2, 2015, [cit. 2015-07-08]. Available from: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain

[43] Memory Management. *Mozilla Developer Network (MDN)* [online]. Last updated on 11. 4. 2015, [cit. 2015-07-08]. Available from: https://developer.mozilla.org/cs/docs/Web/JavaScript/Memory_Management

[44] *TypeScript* [online]. © 2012-2015, [cit. 2015-07-08]. Available from: http://www.typescriptlang.org/

[45] Coupling (computer programming). *Wikipedia, the free encyclopedia* [online]. Last modified on 13 July 2015, [cit. 2015-07-14]. Available from: http://en.wikipedia.org/wiki/Coupling_(computer_programming)

[46] HTML Living Standard. *The Web Hypertext Application Technology Working Group (WHATWG)* [online]. Last updated on 5 May 2015, [cit. 2015-07-08]. Available from: https://html.spec.whatwg.org/#is-this-html5

[47] HTML5. *The World Wide Web Consortium (W3C)* [online]. 24 June 2010, [cit. 2015-07-08]. Available from: http://www.w3.org/TR/2010/WD-html5-20100624/the-canvas-element.html

[48] XMLHttpRequest Living Standard. *The Web Hypertext Application Technology Working Group (WHATWG)* [online]. Last updated 14 July 2015, [cit. 2015-07-08]. Available from: https://xhr.spec.whatwg.org/#document-response

[49] Dynadraw: A dynamic drawing technique. HAEBERLI, Paul. *GRAFICAObscura* [online]. June 1989, [cit. 2015-07-08]. Available from: http://www.graficaobscura.com/dyna/

[50] Inkscape [online]. [cit. 2015-07-16]. Available from: http://bazaar.launchpad.net/ inkscape.dev/inkscape/trunk/view/head:/ src/ui/tools/calligraphic-tool.h

[51] EVA animater plaza. *SHARP* [online]. [cit. 2015-07-08]. Available from: http://www.sharp.co.jp/sc/excite/evademo/evahome.htm

[52] Extended Vector Animation. *Wikipedia, the free encyclopedia* [online]. Last modified on 13 December 2012, [cit. 2015-07-08]. Available from: https://en.wikipedia.org/wiki/Extended_Vector_Animation

[53] SWF. *Wikipedia, the free encyclopedia* [online]. Last modified on 11 May 2015, [cit. 2015-07-08]. Available from: https://en.wikipedia.org/wiki/SWF

[54] WAVE PCM soundfile format. SAPP, Craig Stuart [online]. [cit. 2015-07-09]. Available from: http://soundfile.sapp.org/doc/WaveFormat/

[55] ECMAScript Programming Language. *ecmascript* [online]. [cit. 2015-07-15]. Available from: http://www.ecmascript.org/

[56] Media Capture and Streams. *The World Wide Web Consortium (W3C)* [online]. 14 April 2015, [cit. 2015-07-09]. Available from: http://www.w3.org/TR/mediacapture-streams/#dom-mediadevices-getusermedia

[57] Web Audio API. *The World Wide Web Consortium (W3C)* [online]. 10 October 2013, [cit. 2015-07-09]. Available from: http://www.w3.org/TR/webaudio/#ScriptProcessorNode-section

[58] Web Audio API. *The World Wide Web Consortium (W3C)* [online]. 21 June 2015, [cit. 2015-07-09]. Available from: https://webaudio.github.io/web-audio-api/#the-scriptprocessornode-interface—deprecated

[59] Web Audio API. *The World Wide Web Consortium (W3C)* [online]. 21 June 2015, [cit. 2015-07-09]. Available from: http://webaudio.github.io/web-audio-api/#the-audioworkernodeprocessor-interface

[60] Web Audio API. *The World Wide Web Consortium (W3C)* [online]. 10 October 2013, [cit. 2015-07-09]. Available from: http://www.w3.org/TR/webaudio/#AudioBuffer

[61] Pulse-code modulation. *Wikipedia, the free encyclopedia* [online]. Last modified on 5 July 2015, [cit. 2015-07-09]. Available from: https://en.wikipedia.org/wiki/Pulse-code_modulation

[62] The WebSocket Protocol [online]. December 2011, [cit. 2015-07-09]. Available from: https://tools.ietf.org/html/rfc6455

[63] Bartels, Richard H., John C. Beatty, Brian A. Barsky. An introduction to splines for use in computer graphics [online]. San Francisco: Morgan Kaufmann publishers, INC., 1986 [cit. 2015-07-16]. ISBN 0934613273. Available from: https://cs.uwaterloo.ca/research/tr/1983/CS-83-09.pdf

[64] Spline Curves. House, Donald H. [online]. 2014, [cit. 2015-07-15]. Available from: http://people.cs.clemson.edu/ dhouse/courses/405/notes/splines.pdf

[65] Matrices and Conversions for Uniform Parametric Curves. Millington, Ian. *The R'n'D Guy* [online]. November 17, 2009, [cit. 2015-07-15]. Available from: http://therndguy.com/papers/curves.pdf

[66] The Final Countdown for NPAPI. Schuh, Justin. *Chromium Blog* [online]. November 24, 2014, [cit. 2015-07-16]. Available from: http://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html

[67] Gamma, Erich, Ralph Johnson, Richard Helm, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* USA: Addison-Wesley, 1994. 1. ISBN 02-016-3361-2.

[68] https://developers.google.com/speed/docs/insights/ConfigureViewport

List of Abbreviations

API	Application Programming Interface, page 11
CSS	Cascade Style Sheets, page 23
DOM	Document Object Model, page 10
EVA	Extended Vector Animation, page 14
HTML	HyperText Markup Language, page 10
HTTP	HyperText Transfer Protocol, page 15
IIS	Internet Information Services, page 15
JIT	Just-In-Time compilation, page 20
JVM	Java Virtual Machine, page 9
KSV	Khan-style video, page 2
LPCM	Linear Pulse-code modulation, page 13
MDN	Mozzila Developer Network, page 12
MOOC	Massive Open Online Course, page 4
MSIE	Microsoft Internet Explorer, page 11
OOP	Object oriented programming, page 20
OS	Operating system, page 22
PCM	Pulse-code modulation, page 13
RIA	Rich Internet Application, page 9
SMIL	Synchronized Multimedia Integration Language, page 11
SVG	Scalable Vector Graphics, page 10
TCP	Transmission Control Protocol, page 22
UI	User interface, page 31
URL	Uniform resource locator, page 15
W3C	World Wide Web Consortium, page 10
WAV	Waveform Audio File Format, page 13
XHR	XMLHttpRequest, page 22
XML	Extensible Markup Language, page 9

Contents of the attached files