Videostrates: Collaborative, Distributed and Programmable Video Manipulation

Clemens Nylandsted Klokmose, Christian Remy, Janus Bager Kristensen, Rolf Bagge, Michel Beaudouin-Lafon, Wendy Mackay

Videostrates: Collaborative, Distributed and Programmable Video Manipulation

Clemens N. Klokmose[1], Christian Remy[1], Janus Bager Kristensen[1], Rolf Bagge[1], Michel Beaudouin-Lafon[2,3,4,5], Wendy Mackay[4,2,3,5]
[1]Aarhus University, [2]Université Paris-Sud, [3]CNRS, [4]INRIA, [5]Université Paris-Saclay
{clemens, jbk, rolf}@cavi.au.dk, remy@cc.au.dk, {mbl, mackay}@lri.fr

Figure 1. ***Videostrates*** **examples: A) Two users collaboratively edit the same videostrate, one with a timeline-based editor and the other with a subtitle editor. The results appear in a live, interactive preview on a large screen. B) Videostrates aggregates, broadcasts and records multiple live streams, here from a statically mounted camera and a smartphone. C) A Videostrate-based computational notebook uses Codestrates to programatically create a WebGL animation and synchronize its playback with recorded video composited with a green screen.**

ABSTRACT

We present *Videostrates*, a concept and a toolkit for creating real-time collaborative video editing tools. *Videostrates* supports both live and recorded video composition with a declarative HTML-based notation, combining both simple and sophisticated editing tools that can be used collaboratively. *Videostrates* is programmable and unleashes the power of the modern web platform for video manipulation. We demonstrate its potential through three use scenarios: collaborative video editing with multiple tools and devices; orchestration of multiple live streams that are recorded and broadcast to a popular streaming platform; and programmatic creation of video using WebGL and shaders for blue screen effects. These scenarios only scratch the surface of *Videostrates' potential*, which opens up a design space for novel collaborative video editors with fully programmable interfaces.

Author Keywords

Collaborative Video; Information Substrates; Video Editing

CCS Concepts

•**Human-centered computing** → **User interface toolkits;** *Web-based interaction; Collaborative interaction;*

INTRODUCTION

Video has become a ubiquitous medium on the Internet. YouTube receives over 400 hours of video uploads every minute [52]; the streaming platform Twitch averages over a million concurrent viewers [56]; and Skype users had spent two trillion minutes on video calls by 2016 [53]. The use and consumption of video is often collaborative: Video conferencing and streaming involve multiple users, and recorded videos are shared, annotated and commented on in social platforms. Yet video production is mostly solitary, performed by individuals using single-user applications on personal computers.

Few video tools support real-time collaboration, and if they do, the interface is the same for every user. Furthermore, a substantial functionality gap exists between easy-to-use tools with limited features for non-professionals, and expert tools with sophisticated features for professionals, with correspondingly high skill and budget requirements. Tools for producing live streaming content are trapped in applications separate from those designed for editing recorded video, even though the same effects and manipulations often apply to both. Video conferencing tools inhibit users from manipulating video streams in real time, except for a few novelty features such as overlaying animated emojis.

Our previous work [5, 19, 30] has challenged the traditional concept of applications and documents in favor of dynamic *information substrates* (substrates for short). Substrates are software artifacts that are neither application nor document, but exhibit traits of either depending upon their use. Substrates are malleable, composable with other substrates, and sharable among users and across devices. They can be manipulated by users through tools, as well as through other substrates.

This paper introduces *video substrates*, an information substrate-based approach for working with video. Video substrates provide mechanisms for fundamental video editing functions, from overlaying clips to creating transitions, subtitles and applying effects. They support pre-recorded, live-

UIST 2019, October 20–23, 2019, New Orleans, LA, USA

https://doi.org/10.1145/3332165.3347912.

streamed and procedurally generated video content, as well as creating and storing video files and composing new live streams. Video substrates do not dictate *how* users should interact with video, but rather provide a standard format for expressing video that enables the development of different types of tools. Video substrates can be edited collaboratively across multiple devices, either in real time or asynchronously, using similar or different tools. They are programmable so that developers can create and share new functionality.

We demonstrate these capabilities with *Videostrates*, a web-based implementation of video substrates and a toolkit for developing collaborative, distributed, programmable video editing environments. We showcase its potential through three examples (Figure 1): real-time collaborative video editing with different editors; compositing, annotating and recording live streams; and procedurally manipulating and generating video. *Videostrates* extends the *Webstrates* platform [30] to express composite video objects using HTML. By bringing the real-time sharing capabilities of *Webstrates* to video, *Videostrates* enables the development of collaborative editors where multiple users can edit the same video composition using different tools on different devices.

The core components of the *Videostrates* architecture are a server-side service for playback of videostrates as WebRTC-based streams and a framework for declaratively expressing a video composition in HTML and programmatically controlling video through a JavaScript API. A videostrate can be composed of live, recorded and procedurally generated video material, and exploits modern browser capabilities for handling live video streams and for video processing. *Videostrates* takes advantage of *Webstrates* to synchronise the DOM of a web page across multiple browsers and adds a full architecture for collaborative video editing and live streaming the output to multiple clients.

In the rest of the paper, we review related work, define a general model of video content and manipulation, and describe the *Videostrates* prototype and toolkit. We then illustrate its power with three examples, provide implementation details and conclude with a discussion and avenues for future work.

RELATED WORK

We review related work in three areas: commercial and open-source software, and online video editing platforms; tools and techniques for interacting with video; and the concept of information substrates that underlies video substrates.

Commercial and open source software

Computer-based, non-linear video editing dates to the early 1970s and evolved into professional video editing software, e.g. Adobe's Premiere Pro and Apple's Final Cut Pro. Apple's iMovie and Meltytech's Shotcut provide amateurs with user-friendly interfaces for video editing, albeit with relatively limited functionality. In recent years, dedicated desktop applications have moved to web platforms, as has movie editing, especially for lightweight tasks. Popular services such as YouTube, Instagram and Vimeo offer simple editing features for amateur users, but lack advanced capabilities.

Over the past decade, support for collaboration in professional video editing tools has begun to appear, first through network-based asset management tools that let multiple users access the same raw material, and, more recently, with real-time collaborative editing support, e.g. DaVinci Resolve 15 [10]. While some platforms, such as Frame.io, Wipster.io, WeVideo, and Weaverize support online collaboration over video content, they mostly target professionals and remain in their early stages. The proliferation of streaming has transformed public broadcasting from a professional television service into a mainstream way to promote live video content, with a corresponding change in software, popularizing tools that simplify the process of streaming content, e.g. OBS Studio [4].

Research on video interaction

Research in human-computer interaction in digital video production dates back to the 1980s, e.g., early work by Mackay and Davenport [35]. A recent survey by Schoefmann [49] highlights the great progress in video content manipulation over the past three decades. Although some researchers explored single user or co-located collaborative editing on a shared computer, a few also address real-time collaborative video editing. For example, Terrenghi et al. [54] and Sokoler et al. [51] showcase video editing on an interactive surface; Zigelbaum et al. [62] created tangibles to facilitate co-located collaborative video manipulation; Ramos and Balakrishnan [45] developed a tablet application for video annotation; and Pavel et al. [43] developed VidCrit to provide aynchronous feedback on drafts of edited video. This research offers important insights into the temporal aspects and social dynamics of collaborative video editing, but, except for VidCrit, focuses on co-located interaction on a single device.

Other researchers describe living in media spaces, e.g. EuroPARC's RAVE [20] and WAVE [48] systems, and collaborative production practices, e.g. including collaborative viewing experiences [21], and documentary production within grassroots movements [23]. Together with methodological frameworks for collaborative authoring [40, 9], they offer insights into the different video communication and production practices and further our understanding of video editing practice. *Videostrates* inherently supports collaboration, whether remote or co-located. This means that many insights from the above research are directly relevant and can inform the creation of future *Videostrates*-based collaborative video editing tools.

With a few exceptions, collaborative online video editing has not yet leveraged the possibilities of cloud-based services. Wagner and Keller [58] present a concept of web services for distributed media composition; more recently, Glance [31] introduced crowd-based annotation for video content. The most recent and most advanced tool is LiveMâché [26], an online whiteboard for educational purposes that enables collaborative video sharing and annotation, although with a single static interface for all collaborators.

Another central contribution of *Videostrates* is programmability. About three decades ago, a number of toolkits were developed to support multimedia and video content, including X Toolkit [38], Ttoolkit [25], MET++ [2], Video Widgets [22] and VideoScheme [36]. Since then, work has shifted towards

online platforms: Hop [50] offers web-based multimedia programming similar to Orcc [61] that builds on a dataflow programming model. NUBOMEDIA [18] provides an entire suite based on WebRTC for programmable cloud-based editing. Other recent examples include the Stage Framework [1] and Timesheet.js [7]. TextAlive [28] enables creation of kinetic typography videos generated from audio and text input through a combined design and live programming environment. For live coding performances [8], Troop [29] allows for collaborative programming of procedurally generated audiovisual experiences. While closest in nature to *Videostrates*' dynamic interface, these systems do not currently combine programmability with collaborative and distributed video manipulation.

Streaming is another popular form of content distribution. Research in this area has focused on tools for live broadcasting and studies of user practices. VideoServer [47] offers a web-based media space to simplify sharing of video content, including streams, via URLs. In their studies of streaming services, Reeves et al. [46] and Juhlin et al. [27] highlight design requirements such as giving agency to users and supporting mobile devices. Other novel systems focus on streaming support for mobile development, such as SwarmCam [15] and Streamer.Space [42], which implement mobile live broadcasting. *Videostrates* supports streaming of content on any web-enabled device and is fully customizable.

In summary, previous research typically targets specific categories of use and dedicated user interfaces, whereas the goal of video substrates, as embodied by *Videostrates*, is to encompass a wide range of activities by providing an enabling technology that supports real-time collaborative editing of both recorded and streaming video. We build on early visions of digital video support [35, 6], with the goal of fundamentally changing how users interact with and manipulate video.

Information substrates

Beaudouin-Lafon [5] describes "information substrates" (substrates for short) as an alternative to the traditional model of applications and documents. Substrates are software artifacts that embody traits of both applications and documents, and can be either, depending on their use. Klokmose et al. [30] present Webstrates as a proof-of-concept web-based implementation of substrates. A *webstrate* is a web page where any edits to its DOM (document object model) is made persistent and shared in real time with the other users of that page. Klokmose et al. demonstrate how this enables the creation of collaborative software that blurs the boundary between usage and development. They demonstrate how transclusion (including a webstrate into another webstrate) enables asymmetric collaboration, e.g. two collaborators editing the same document but each using their own personal user interface and editing tools.

With Codestrates [44], Rädle et al. introduce a development environment for Webstrates inspired by computational notebooks, allowing users to seamlessly move between collaboratively developing, using and extending software. *Videostrates* builds upon the principles and technology introduced with Webstrates and Codestrates, extending them to liberate video from the confines of traditional applications and make to video editing inherently collaborative.

WHAT IS VIDEO?

As users, we think about video in terms of the systems we use to access and manage it. Thus, we watch pre-recorded video with players such as QuickTime or VLC or on platforms such as YouTube or Vimeo. We also capture video with cameras and smartphones, and sometimes edit it for later viewing with tools such as iMovie, Adobe Premiere, or Final Cut Pro. We sometimes annotate or extract metadata from video, e.g., when coding video data collected from field or lab studies with tools such as EVA [35], DIVA [34] or Chronoviz [17]. We also procedurally generate or manipulate video, e.g., for creating kinetic typography [28] or editing dialogue-driven scenes [32].

Video is also a live medium that we can watch on TV or livecast on platforms such as Facebook Live or Instagram. These live streams can be time-shifted if the stream is recorded, such as with a TV set-top box. The live stream can also result from real-time editing, e.g., when broadcasting a sporting event captured with multiple cameras or when a VJ creates live video from various sources. Finally, video is a rich communication medium, as attested by the success of videoconferencing services such as Skype, FaceTime, Bluejeans, or Zoom.

While all these activities manipulate "video", each is embedded within a different type of application that insulates it from other uses. For example, videoconferencing applications such as Skype do not permit live editing of conversations, e.g., to combine the live video feed of a participant with that of a screencast. Similarly, platforms such as YouTube share video content, but do not support synchronized viewing of the same video by multiple users. Based on these diverse uses of video, we identify three key dimensions of video: *source type*, *stream multiplicity*, and *interaction* (Fig. 2).

First, video can come from different *source types* i.e. captured while playing, recorded, or generated procedurally from code. Time-shifting records a live source so users can pause, replay, and catch up with the live source, thus combining aspects of both live and recorded video. These different source types represent video in diverse *formats*: Video authoring tools represent the composition of recorded material as *project files* that can be exported as a pre-rendered video or an *edit decision list* (EDL). Streaming services usually transfer live streams, although games, for example, only transfer updates to the scene. Finally, live coding represents procedurally generated video using program code.

Second, video can comprise a *multiplicity of streams*. Multiple streams occur naturally in video-conferencing situations, where participants generate their own streams, but also in multi-camera situations, such as sporting events or music concerts. Video editing applications also treat each video clip as a separate stream.

Third, video can involve diverse forms of *interaction*:

- *Capturing* video, recorded and/or live streamed;
- *Playing* live, time-shifted, or recorded video;
- *Annotating* video by adding metadata from external sources or from analyzing the video content itself;
- *Editing* time-based properties of video segments, including cutting, trimming and transitioning;

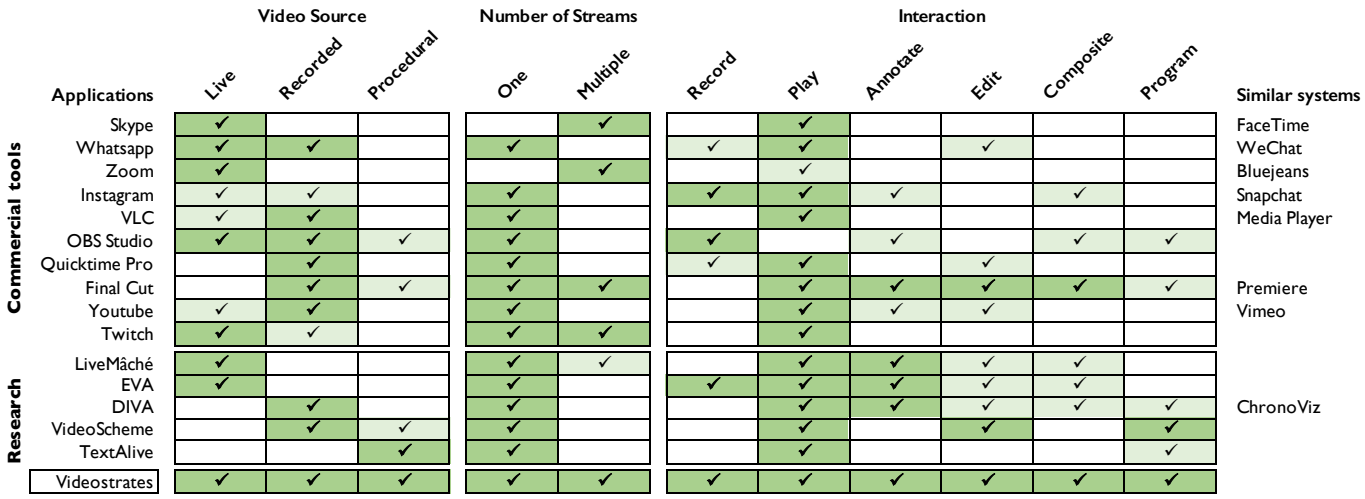

Figure 2. A selection of common commercial video applications and platforms as well as prominent research systems, categorized according to key video dimensions: video source, stream multiplicity, and interaction. Pale green cells represent partially or poorly supported features.

- *Compositing* the spatial combination of video and other content, including overlaying subtitles, adding image filters or green-screen effects, creating a picture-in-picture, and generating a mosaic with multiple video streams; and
- *Programming* video, either by procedurally generating it or manipulating a video stream with code.

Sharing is a key, if often overlooked, aspect of interacting with video. Although natural for video conferencing systems, which are designed to exchange video streams, few tools support synchronized playing of recorded video, despite well established practices such as joint TV and movie watching. Similarly, collaborative annotation, editing, and compositing is rare, even given the complex teamwork required for professional video production. Our goal is thus to create a unified framework and platform for capturing, playing, annotating, editing, and compositing video, with corresponding capabilities for sharing multiple live and recorded video streams.

VIDEOSTRATES PRINCIPLES

A *video substrate* is based on the concept of an information substrate, applied to video. It contains information about source content—whether it is live (e.g. a stream), recorded (in a file), or procedurally generated—and how this content should be edited and composited together. A video substrate can contain functionality to edit and render itself or may be rendered and edited through another substrate via transclusion. A video substrate may contain content, interactions and behaviour unrelated to the rendered video. Thus, one could create a video substrate that represents an entire video production, including storyboards, script, notes, etc.

Videostrates is our proof-of-concept implementation of video substrates. It is based on the Webstrates [30] platform and provides a toolkit for developing video editing tools. A videostrate is a webstrate that expresses a video composed of live, recorded or procedurally generated video content. It can contain a number of `video` HTML elements referencing source material, information about which part of each video should be played when, and style sheets describing animated transitions between videos or the layout of a picture-in-picture display. A videostrate can also contain subtitles and rolling credits animated with scripts and CSS style sheets.

From the developer's perspective, *Videostrates* consists of a data format for expressing video compositions using HTML and DOM manipulation, and an API for playback of a video as a stream and for rendering the video composition to a file. (See the appendix for an overview of the data format and API.)

Listing 1 shows a simple videostrate with three video segments. The first segment plays for the first 10 seconds from the first video file. The second segment plays for the next 10 seconds from the second video file at double speed, starting from the fifth second of the file. The third segment starts playing after 5 seconds for 10 seconds from the third video file with a 1.5 second fade-in animation at the beginning. This is overlaid over the the others using the CSS in the `style` tag.

```
<html>
  <head>
    <style>
      .overlay {
        position: absolute;
        left: 10px;
        top: 10px;
        width: 200px;
        height: 200px;
      }
    </style>
  </head>
  <body>
    <video class="composited" src="one.mp4" data-start="0"
      data-end="10" data-offset="0"></video>
    <video class="composited" src="two.mp4" data-start="10
     " data-end="20" data-offset="5" data-speed="2"></
     video>
    <video class="composited overlay" src="three.mp4"
     data-start="5" data-end="15" data-offset="5" style="
     animation-name: compositor_transition_fade;
     animation-duration: 1.5s;"></video>
  </body>
</html>
```

Listing 1. HTML structure of a simple videostrate.

Listing 1 shows the declarative notation used by Videostrates. Each HTML `video` element with the `composited` class is included[1] in the final video, based on its `data-start`, `data-end`, `data-offset` and `data-speed` attributes, denoting when in the final video the clip starts, ends, its time offset into the source file and the playback speed.

Note that even though Listing 1 features three `video` elements, the semantics of videostrates is that it represents a *single* video clip. This video clip can then be viewed in the `video` element of *another* webstrate. Conceptually, we want a videostrate to be specified as the source of an HTML `video` element. In addition, since a videostrate is a webstrate, it can be shared so that any change is immediately visible from any page that references it. Neither functionality is supported by HTML5, so we need to create our own rendering infrastructure.

We thus developed *vCompositor*, the videostrate compositor, and *vStreamer*, the videostrate streamer, to interpret the notation and render it as a video. The *vStreamer* service takes the URL of a videostrate and generates a WebRTC stream that can be displayed in the `video` element of a browser and controlled from that browser. As described in Implementation, *vStreamer* spawns a server-side browser instance that opens the given videostrate, grabs and streams the graphics output from that browser and streams it to client browsers. *vStreamer* can also render the videostrate to a video file, e.g. in the MPEG-4 format, for later download or use as a source. *vStreamer* is controllable through an HTTP-based API that can be used through a simple JavaScript library (see Appendix). This library can be used, for example, to create a standalone videostrate player or to integrate playback into a video editor.

vCompositor is a Javascript and CSS framework that manages the playback of all composited elements in a videostrate, and is used by *vStreamer* to control the playback. Besides its declarative approach to video composition, *vCompositor* supports programmatic generation of video content. Listing 2 shows code for a video of a red rectangle that follows a sine curve to move across the screen, implemented with the HTML5 canvas.

```
1  <canvas width="640" height="480" class="custom composited"
       data-start="0" data-end="10"></canvas>
2  <script>
3  let c = document.querySelector("canvas");
4  let ctx = c.getContext("2d");
5
6  c.customLocalSeek = (time) => {
7      ctx.clearRect(0, 0, c.width, c.height);
8      ctx.fillStyle = "#FF0000";
9      ctx.fillRect(time/10 * c.width, 150 - Math.sin(time) *
         120,10,10);
10 };
11 </script>
```

Listing 2. Using a custom component to programmatically generate video content. The `div` element on line 1 has the class `custom composited` that tells *vCompositor* to call a callback function (lines 6-10) when the clock changes or when seeking in the video.

Implementing videostrates as webstrates implies that:

- A videostrate is collaboratively editable, meaning that two or more users can edit a videostrate simultaneously.
- If a stream of a videostrate is created using *vStreamer* and the videostrate is updated from elsewhere, the streamed video will update in real time. For example, if a stream is created for the videostrate in listing 1 and the CSS is updated to a new position while the stream plays, the composited video will change its position in real time.
- Transclusion can be used to edit videostrates, thus a *video editor webstrate* can provide the interface and tools for editing a videostrate. This allows videostrates to be edited simultaneously with multiple different editors.
- Since videostrates are webstrates that happen to use the above notation so they can be rendered by *vCompositor*, they are not restricted to video-based content and functionality. They can contain any content, behaviour or interaction available in standard webstrates.

[1]Videostrates make no technical distinction between compositing and editing. We have chosen to use the keyword `composited` to denote a video clip that is either edited or composited into a video.

EXAMPLES

We present three examples developed with *Videostrates* that demonstrate its power and generality. Each example features a short scenario followed by an explanation of how it works, and is illustrated in the accompanying video.

Collaborative video editing

Alice and Bob are interaction design students. Alice is an experienced video editor, but Bob is a novice. They have created paper mockups of a user interface prototype for a class, and must create a video to show how it is used. They record video clips on their mobile phones. To edit them into a short video, Alice creates a new, empty videostrate on her laptop and opens it in a timeline-based video editor. The videostrate is created from a prototype that includes a video upload form. Both Alice and Bob open the videostrate on their phones and upload their video material. As the videos are uploaded, they appear in Bob's video editor on his laptop. Alice shares the link to her editor with Bob, so he can participate in editing the video. The video editor has limited functionality, so in order to add transition effects, Alice manually edits the videostrate's HTML. Finally, Bob opens the videostrate in a simple editor, capable only of adding subtitles and adds explanatory text to the video. Although Alice's editor does not support subtitle *editing*, she can use it to scroll through the video and see where the subtitles appear in the video.

How it works

A videostrate is a webstrate, and new videostrates can therefore be created by copying from a prototype (by appending `?copy` to the prototype's URL). To upload videos, the prototype needs to contain an HTML upload form with an `INPUT` tag with type `video`. Uploaded video clips are attached to the videostrate as assets [13] and are accessible through their URL.

Our video editor (Figure 3) is implemented using Codestrates [44], and is a minimalistic timeline-based editor inspired by Apple's iMovie, comprising 2648 lines of code (LOC). The videostrate is loaded into the editor using transclusion. This means that the editor opens the videostrate in an `iframe` element that is hidden from the user, and manipulates the Document Object Model (DOM) of the videostrate to edit the video. The videostrate is basically treated as the model of

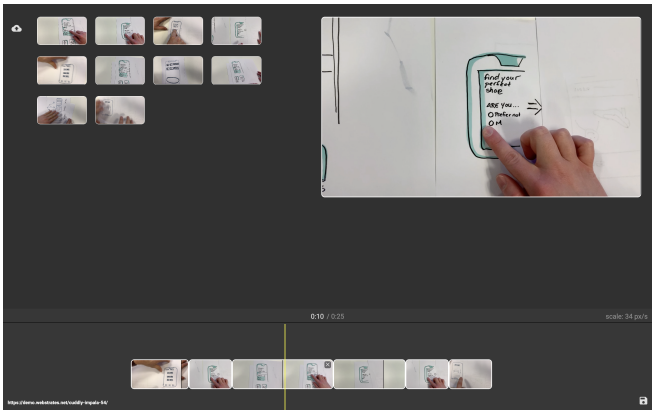

Figure 3. A simple timeline-based video editor, inspired by Apple's iMovie. A videostrate is opened in the editor using transclusion, and video material is uploaded to the videostrate and arranged in the timeline. A preview in the top right is streamed live from *vStreamer*.

a model-view-controller pattern. Video clips uploaded to the videostrate appear in the top left corner. These can be dragged to the timeline at the bottom, causing a `video` tag to be added to the transcluded videostrate with the `composited` class and start and end times matching its place in the timeline. The clip can now be resized and dragged around in the timeline, which updates the videostrate accordingly.

A preview (top right) is generated using the *vStreamer* API. Hovering the cursor over the timeline seeks through the video. In this implementation of the video editor, each user receives their own stream from *vStreamer*, which means that they can independently play back and seek through the video. However, playing and seeking in the video could also be synchronized among clients. To render the videostrate to a file, the user clicks the save icon in the bottom right corner. *vStreamer* starts a pixel- and frame-perfect rendering, and a callback is invoked when it finishes, so the user can download the video.

A file cache on the videostrate server (*vCache*) with a simple API generates the thumbnails used for the video clips and timeline, and caches source files for fast retrieval. The timeline in the video editor updates in real-time based on remote changes to the videostrate, which means that multiple users can collaborate on the same videostrate. Editing conflicts are handled by Webstrates using operational transformation (OT) on the DOM. This guarantees eventual consistency between the state of the videostrate across clients, but does not protect against, e.g., a user deleting a video clip being editing by another user. This is similar to collaborative text editors such as Google Docs, which do not protect users from deleting a sentence that another user is writing.

The subtitle editor (268 LOC) shown in figure 4 is an independent webstrate that transcludes the same videostrate as the editor and displays a table of all subtitle elements in the videostrate. The subtitle editor shows a *vStreamer*-generated stream of the video. The user can add new subtitles by seeking to a time in the video using the progress bar and press a plus icon. Clicking a time in the table updates it to the current

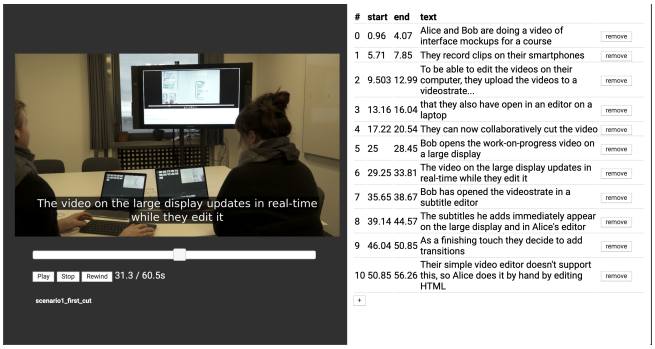

Figure 4. This subtitle editor for *Videostrates* transcludes a videostrate to add, remove and edit subtitles.

position of the progress bar. Listing 3 shows an example of the HTML for a subtitle track.

```
<div class="composited subtitles">
  <span class="composited subtitle" data-start="729"
    data-end="734">Who was he?</span>
  <span class="composited subtitle" data-start="740"
    data-end="745">I have no idea</span>
</div>
```

Listing 3. HTML of a subtitle track

If Alice or Bob are fluent in HTML, they can manually edit the videostrate, for example to add transitions that the editor does not support. They can open the browser's Developer Tools and inspect and edit the elements in the DOM. They can also mount the videostrate as a file in their filesystem (see [60]) and use their favorite code editor to edit the HTML, CSS and JavaScript of the videostrate.

Live streaming studio

A big city library is hosting a popular annual three-day comic book convention. Beth, one of the volunteers, has been tasked to video document the convention, both with a live stream and with daily three-minute highlight videos. She creates a videostrate from the *stream studio* videostrate prototype and opens it on her laptop. She also opens it on a computer connected to a ceiling-mounted camera that captures an overview of the main convention area. She immediately sees the camera feed on her laptop. She sends the URL of the videostrate to the other volunteers and asks them to open it on their phone so they can use their phone camera to record from the convention floor when they have the opportunity. On her laptop, Beth sees a small video thumbnail appear when a volunteer starts to stream and she can composite the different streams from her laptop. She switches between streams, or displays the overview stream plus one or more mobile camera streams simultaneously. Beth also annotates the stream on her tablet by drawing on top of the video with a pen.

To share the composited stream with the world without compromising the library's limited server capacity, Beth connects the library's *Videostrates* server to a popular online streaming platform and embeds the stream on the convention web site. At the end of the day, Beth creates a new videostrate and uses a simple video editor webstrate to cut a three-minute highlight reel of the day that she exports to a video file and uploads to the library website.

Figure 5. The *stream studio* videostrate opens a controller on a laptop and a viewer on a tablet. The controller shows five streams, three of which are minimized as thumbnails and do not appear on the tablet.

How it works

The *stream studio* videostrate (Figure 5) leverages modern browser support for peer-to-peer video streaming using WebRTC [57] and for capturing video [12], and comprises 595 LOC. Webstrates provides a simple API for creating WebRTC streams [14] between clients of the same webstrate, and a stream can be connected to a `video` element by using `videoElement.srcObject = stream`.

When *vStreamer* is asked to open the *stream studio* videostrate, it receives and displays the live WebRTC streams from the other connected clients. The *stream studio* videostrate is thus not time-based – it does not contain elements with the `composited` class, `data-start` and `data-end` attributes.

If the *stream studio* videostrate is opened on a mobile phone (or with a parameter in the URL) it acts as a sender. If opened in a desktop browser it acts as a controller. When a stream is created from a mobile device, it shows up as a thumbnail in the controller. Clicking the thumbnail displays the video full screen, or juxtaposes multiple feeds to fit the screen. This is accomplished by manipulating CSS and DOM in JavaScript.

The two buttons in the bottom right corner record and broadcast via Twitch[2]. If the user clicks the record button, *vStreamer* starts recording the live stream to a file. Clicking the Twitch button prompts the user for the RMTP URL of their Twitch account and begins broadcasting. Listing 4 shows an example of using the broadcasting API with an external service.

```
broadcastButton.addEventListener("click", (e) => {
  if (videostrate.isBroadcasting()) videostrate.
   stopBroadcasting();
  else {
    let rtmpTarget = window.prompt("Enter RTMP URL");
    videostrate.broadcast(rtmpTarget);
  }
});
```

Listing 4. Broadcasting using RTMP

Opening the *stream studio* videostrate on a tablet enables pen input for drawing free-hand SVG annotations on top of the view. These annotations are SVG elements superimposed on the video feed(s), and are rendered by *vStreamer*, as any other videostrate content.

[2] `twitch.tv` is a popular live streaming service.

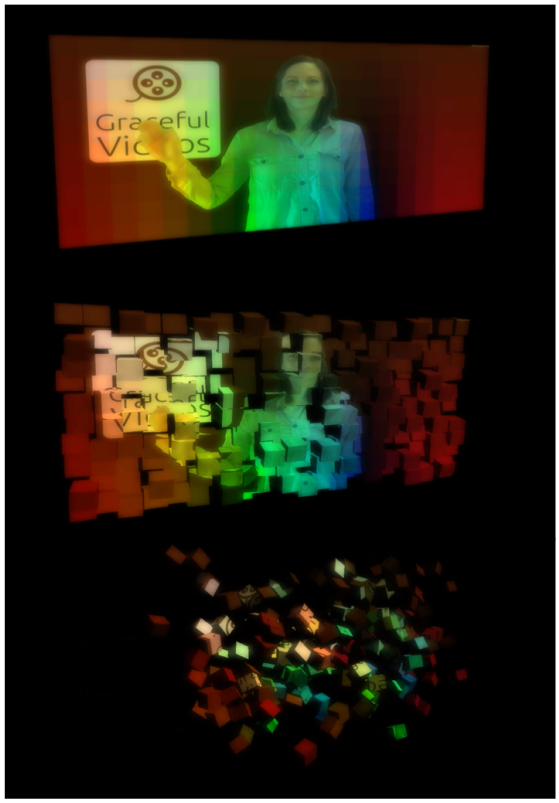

Figure 6. Three frames from Grace's animation consisting of a chroma-keyed video texture-mapped onto geometry in a WebGL scene.

Recorded video can be added to another videostrate for editing into the highlights reel. Time shifting could be implemented by programatically playing back the videostrate containing the recordings inside the stream studio, although this is not currently implemented in our prototype.

Programmable video

Grace runs a small one-woman video production and visual effects company. She has been working on a showreel of her work and wants to create an animated opening logo that she can use in other productions. She has an idea of a video of her waving with the company logo in the background, and the video slowly disintegrating into cubes, disappearing into outer space (Figure 6). She starts by programming the disintegrating cube animation using WebGL in a codestrate. She remembers that she had previously programmed a WebGL shader for chroma-keying that she uses for live streaming tutorials. She creates a chroma-keying videostrate by adapting the code to make it work with recorded video. She uploads a video of herself waving in front of a green screen to the videostrate, creates a background in HTML, and renders a new video. Finally, she loads the chroma-keyed video into the logo animation videostrate. She can now transclude the logo videostrate into the showreel videostrate, or any other videostrate she is working on.

How it works

To procedurally create the animation of the disintegrating cubes, we modified a Three.js example [55] in a codestrate

(307 LOC) to use a video as a texture, and controlled the animation using the time callback on a custom videostrate component. This means that the animation and video are synchronized when seeking back and forth within the video. For the chroma-key videostrate (962 LOC) we implemented a chroma-key shader in the Graphics Library Shader Language (GLSL). The shader receives a background texture, a foreground texture (the video), and a texture defining the color space. The codestrate implementing the chroma key videostrate includes a simple interface for specifying which colors to exclude. This is accessed by adding a parameter to the URL of the videostrate. The videostrate also contains the HTML for the background that is rendered onto the canvas on page load, so that it can be used as a texture for the shader. Both the animation videostrate and chroma-key videostrate are implemented as codestrates (Figure 1(C)) where the development environment is hidden using CSS during playback.

IMPLEMENTATION

Figure 7 shows the *Videostrates* architecture for the video editor from the collaborative video editing example. The main components of *Videostrates* are the Videostrates Streamer (*vStreamer*) and Videostrates Compositor (*vCompositer*).

Videostrates builds on Webstrates [30]: a videostrate is a webstrate that uses the declarative notation and JavaScript API of *vCompositor*. *Videostrates* thus assumes access to a Webstrates server, which can be co-hosted with the *Videostrates* server or hosted externally. We use the publicly available Webstrates server at `https://demo.webstrates.net`.

vStreamer

vStreamer is a service running on a dedicated Linux server with the main responsibility of spawning browser instances, capturing their graphics output and streaming it to clients. When a request is received from a client to open a videostrate, the *vStreamer* controller spawns a Firefox [39] browser process in a security sandbox (using Firejail [16]) on a window surface matching the dimensions given by the client. The window surface is not rendered on the screen, but instead each frame is captured on the graphics card by a frame grabber and sent to the *vStreamer* controller process through a UNIX pipe.

The *vStreamer* controller process (implemented in Java) uses the GStreamer [24] media handling framework to create a stream from the grabbed frames. Using GStreamer modules, *vStreamer* can send this stream back to the client using WebRTC, record it in a file, or broadcast it using the RTMP protocol to streaming services such as uStream or Twitch.

When *streaming* a videostrate to a client, frames may be skipped to avoid slow down. However, it is is essential that no frames are lost when *rendering* to a file. To coordinate when the browser has finished showing a frame and can proceed to the next, the frame grabber hosts a tiny WebSocket server that the *vCompositor* webstrate (see below) can coordinate through. Rendering videostrates can thus be slower than real time, depending on the quality of the source material and the complexity of the videostrate. Typically, rendering takes twice as long as the duration of the video. To support time shifting, recorded files are encoded so that they can be played while recording. Similarly, to support previewing, rendered files can be played before rendering is complete.

The *vStreamer* service includes a simple file hosting service for the files produced when rendering or recording. Furthermore, it contains a cache that can be used through a simple HTTP API to generate alternative encodings of a source video file, e.g. to support showing low resolution thumbnails, or producing a preview video that supports fast seeking through an increased number of key frames. *vStreamer* can, in principle, stream and record any webpage with a public URL. However, to control playback, it must be a videostrate that uses the *vCompositor* notation or API.

vCompositor

vCompositor is implemented as a webstrate in JavaScript and CSS. It is loaded in the Firefox instance opened by the *vStreamer* controller, and the URL of the videostrate that the client has requested to be streamed is given in a query string. *vCompositor* transcludes the given videostrate in a transient[3] `iframe`. It injects JavaScript and CSS for playback control into the transcluded videostrate inlined in transient elements. The client also transcludes the *vCompositor* webstrate, which acts as a communication channel to the *vCompositor* instance opened in *vStreamer*, using Webstrates' signalling mechanism[4]. These signals issue play, pause and seek commands.

vCompositor can play back composited video elements, animated SVG elements, DOM elements animated by CSS, custom elements, and transcluded videostrates. It computes the total video duration based on the start and end times, and the playback speed of each composited element. The composited elements are hidden using CSS until the time matches their start time. For video playback, *vCompositor* bypasses the browser's standard video playback, and instead seeks forward in the video element, frame by frame. We exploit Firefox's experimental API (`HTMLMediaElement.seekToNextFrame` [11]) to quickly seek to the next frame without having to construct it from the previous keyframe.

To play back CSS animations on composited elements, we set the animation delay. Animations applied to composited elements are paused by default (using the `animation-play-state` CSS attribute). When the time on an element is updated through a seek, we update the animation delay to match the time. This causes the animation to display a static frame that corresponds to the current time. For custom components, a callback function `customLocalSeek` is called when the time is updated. Finally, for transcluded videostrates, *vCompositor* recursively opens the transcluded videostrate for playback. When time updates in the parent, seek is called on the transcluded videostrate.

vCompositor monitors the DOM of the loaded videostrate using JavaScript mutation observers and updates its time accordingly when new composited elements appear or are changed.

[3]Transient elements in webstrates are elements that are not made persistent on the server nor synchronized with other clients

[4]Webstrates uses a WebSockets-style API to send signals between clients of the same webstrate.

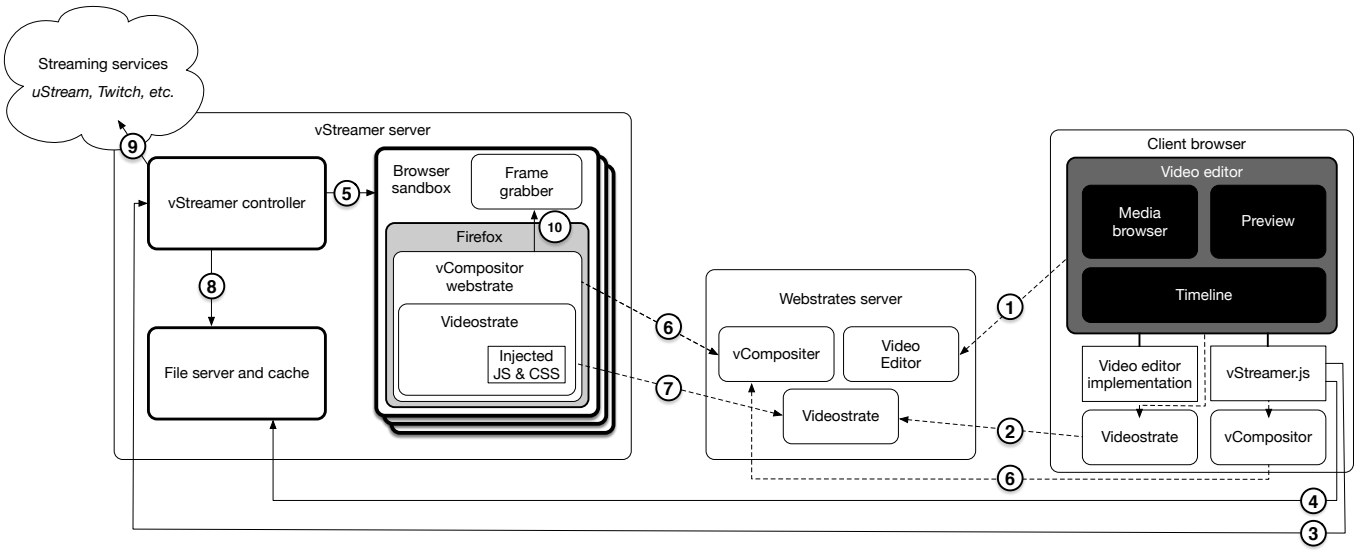

Figure 7. The *Videostrates* architecture (video editing example). 1) The client opens the video editor webstrate in a web browser on her local machine. 2) The video editor transcludes a videostrate for editing. 3) The *vStreamer* JavaScript API generates a WebRTC stream used to preview and control rendering, recording and broadcasting of the videostrate. 4) The cache API generates media previews for the media browser and timeline. 5) When *vStreamer* receives a request to open a videostrate, it spawns an instance of the Firefox browser that opens the *vCompositor* webstrate, which transcludes the given videostrate in an iframe. *vCompositor* injects scripts and styles into the videostrate to control playback. A frame grabber connects to the Firefox instance to pipe graphics output from the Firefox instance into the *vStreamer* controller. 6) Playback is controlled via signals across the *vCompositor* webstrate, which is also transcluded in the client browser. 7) Any changes to the videostrate are immediately rerendered by *vStreamer*. 8) Rendered and recorded files are stored in the file server for later retrieval. 9) Broadcasted streams are sent to external services using RTMP. 10) When rendering, *vCompositor* coordinates with the frame grabber over a WebSocket connection.

DISCUSSION

The *Videostrates* prototype demonstrates the viability and practicality of an information substrates-based approach to video manipulation. Although *Videostrates* was designed as a proof-of-concept, we use it for producing and editing video material: The accompanying video was produced using *Videostrates*, including the use of custom videostrates for screen recording.

Supporting end users

Our examples show how *Videostrates* can be used to let users collaborate on video content without being trapped inside a particular application. Each example highlights a particular aspect of *Videostrates*, but they could easily be combined. Indeed, *Videostrates* provides an enabling technology that developers and designers can use to both replicate existing video production workflows as well as explore new possibilities.

Thanks to a unified representation of video, if a user reaches the limits of an editor created with *Videostrates*, they can use another one, extend it or even create their own. We expect *Videostrates* to foster new forms of video editing, similar to the transition from linear to non-linear editing in the 1990's.

Performance

Providing exact performance measures is difficult, as performance depends heavily on the design of a given videostrate. A videostrate consisting of a simple composition of low-resolution video material naturally renders much faster than a 4K composition with extensive use of WebGL effects. However, our tests show that we can support simultaneous playback of six videostrates with 4K source material to a 1280x720 resolution stream.

Our *Videostrates* server is hosted on a high performance consumer-grade workstation with a 32 core AMD Threadripper 2 CPU and two Nvidia 960 GTX graphics cards. This also means that a powerful laptop can host a local instance of a *Videostrates* server for offline rendering. The current implementation of Videostrates is not designed for 8K cinematic productions. The key bottleneck is video rendering and memory management in the browser, which browser vendors constantly improve, with immediate benefits to *Videostrates*.

Systems-oriented evaluation

We view *Videostrates* as an example of what Olsen calls user interface systems research [41]. We evaluate it by demonstrating its potential [33] according to Olsen's criteria, as relevant.

To our knowledge, Videostrates solves a *previously unsolved problem*: Although collaborative video editors, web-based video editors and programmable or scriptable video editors all exist individually, no system combines them in a single medium. Our analysis of what it means to interact with video led to the concept of video substrates. *Videostrates* demonstrates how this concept unifies uses of video that were previously addressed by incompatible tools.

We demonstrate the *generality* of *Videostrates* with three very different examples that remix video editing tools in novel ways. We show how to interact with *Videostrates* using simple drag-and-drop tools or a code editor. We also show how *Videostrates empowers new design participants*: video editing tools can be built with web development skills and frameworks, and sophisticated video productions that previously would have required expensive software and specialized skills can now be produced with just HTML, CSS and JavaScript.

When used as a toolkit for building video editors, *Videostrates* offers great *flexibility*: it is very easy to change a *Videostrates* editor and experiment with new features, or simultaneously compare different editors on the same videostrate. *Videostrates* enables *inductive combination* by providing a relatively small set of building blocks that can be combined to create sophisticated video editors. It provides *ease of combination*, since the common, underlying interface is the standardized DOM API.

Finally, Olsen poses a key question: *Can it scale up?* We show how live streams can be shared on streaming platforms that support hundreds of thousands, if not millions of users. Our tests show that six simultaneous users can render 4K video material on an inexpensive workstation (the price of two laptops), making it powerful enough for 4K video editing. In terms of performance, we do not believe scalability is an issue for practical use cases. However, scalability of functionality can be costly, and would require a significant design and engineering effort to create video editors for *Videostrates* that match the sophistication of commercial desktop video editing software. Even so, *Videostrates* opens the possibility of an open source community that creates and shares video editing tools.

Limitations and future work

The most salient limitation of *Videostrates* is the lack of audio support. The Web Audio API supports sophisticated audio manipulation including effects, which can be enabled through a declarative notation similar to what we did for video. The requirements for adding audio to collaborative editing is highlighted in previous work [37, 47, 59], and is beyond the scope of this paper, although a top priority for future work.

The *vStreamer* service uses UDP for transmission and does not provide any error correction when packets are lost in WebRTC streams. This may lead to performance issues when streaming over unstable networks. Future work includes implementing error correction techniques, such as retransmission (RTC), forward error correction (FEC), or package loss indicators (PLI) to resend a previous keyframe when data is lost. We should also support a greater variety of resolutions, e.g., low resolution for a preview, but high resolution for rendering to a file, which would greatly enhance scalability.

We have included several CSS animations in *vComposited* that can be used to declaratively apply special effects to videos. We plan to extent this collection of animations, as well as include preprogrammed effects beyond current CSS capabilities. We also want to explore how to integrate real-time or off-line video-processing such as face recognition to complement the video stream with metadata.

Finally, we have not addressed issues of privacy and security beyond the existing support in the underlying *Webstrates* server. We are also limited by the performance and capabilities of browsers to support advanced real-time processing such as color correction and grading, and real-time visualizations such as video scopes.

Future research

Videostrates opens up new ways to edit, share and interact with video, beyond the traditional video editing paradigm. Because *Videostrates* is built on *Webstrates*, we can combine video editing with other forms of video interaction, integrating relevant tools as needed.

Badam et al. [3] demonstrate how a Webstrates-based visualization platform supports integrating collaboratively editable and reprogrammable data visualisation workflows with other tools, e.g., slideshow presentations and computational notebooks. We envision similar possibilities for *Videostrates*. For example, we could add video editing to an online computational notebook for high school students to let them document science experiments. Or, we could create simple tools that let first graders collect and edit insect videos and share them via a multimedia canvas on a classroom electronic whiteboard.

Professional video editing tools currently support a large ecosystem of plug-ins and add-ons. Our goal is not to compete with such professional video editing suites. Instead, we envision *Videostrates* as a basis to support a similar, but community-based approach for developing, sharing, and potentially selling innovative video editing tools. *Videostrates* can let people use their own, personalizable tools so that novices can adopt simple, pre-made editors, and still collaborate with professionals who use sophisticated or custom-made tools on the same video production, leading to new types of workflows.

CONCLUSION

This paper describes the concept of *video substrates*, a form of information substrate that supports collaborative, distributed and programmable video manipulation of both live and recorded video. It also presents *Videostrates*, a proof-of-concept implementation that illustrates both the potential and practical details of implementing video substrates.

Videostrates is an enabling technology that supports a wide variety of video applications by combining a simple declarative HTML-based data format with a toolkit for video manipulation. *Videostrates* makes it possible to create personalized tools, as well as directly edit the HTML code describing a videostrate. *Videostrates* builds on the *Webstrates* platform to supports real-time sharing, enabling multiple users to collaborate simultaneously on the same video production using their own tools. *Videostrates* leverage modern web technology to include live streams from users' personal devices and to support programming video effects with widely available graphics libraries on the web.

Videostrates opens a new design space for collaborative video manipulation while offering new ways to integrate video into other tools and contexts. We illustrate some possibilities with three scenarios, but also invite the community to explore its full potential. *Videostrates* is available at `https://videostrates.projects.cavi.au.dk`.

ACKNOWLEDGMENTS

This project has been partially supported by the Aarhus University Research Foundation, The Carlsberg Foundation, Innovation Fund Denmark 5123-00007B, JPI Urban Europe EU 693443, European Research Council (ERC) grants no. 321135 CREATIV: Creating Co-Adaptive Human-Computer Partnerships, and no. 695464 ONE: Unified Principles of Interaction.

REFERENCES

[1] Rami Aamulehto, Mikko Kuhna, Jussi Tarvainen, and Pirkko Oittinen. 2013. Stage framework: an HTML5 and CSS3 framework for digital publishing. In *Proceedings of the 21st ACM international conference on Multimedia - MM '13*. ACM Press, Barcelona, Spain, 851–854. DOI: http://dx.doi.org/10.1145/2502081.2502228

[2] P. Ackermann. 1994. Direct manipulation of temporal structures in a multimedia application framework. In *Proceedings of the second ACM international conference on Multimedia - MULTIMEDIA '94*. ACM Press, San Francisco, California, United States, 51–58. DOI: http://dx.doi.org/10.1145/192593.192621

[3] S. K. Badam, A. Mathisen, R. Rädle, C. N. Klokmose, and N. Elmqvist. 2019. Vistrates: A Component Model for Ubiquitous Analytics. *IEEE Transactions on Visualization and Computer Graphics* (2019), 1–1. DOI: http://dx.doi.org/10.1109/TVCG.2018.2865144

[4] Jim Baily. 2019. OBS Project. https://obsproject.com/. (2019). Accessed: 2019-29-03.

[5] Michel Beaudouin-Lafon. 2017. Towards unified principles of interaction. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter*. ACM, 1 page. DOI: http://dx.doi.org/10.1145/3125571.3125602

[6] Amy Bruckman. 1992. The electronic scrapbook: knowledge representation and interface design for desktop video. In *Posters and short talks of the 1992 SIGCHI conference on Human factors in computing systems - CHI '92*. ACM Press, Monterey, California, 7. DOI: http://dx.doi.org/10.1145/1125021.1125027

[7] Fabien Cazenave, Vincent Quint, and Cécile Roisin. 2011. Timesheets.js: tools for web multimedia. In *Proceedings of the 19th ACM international conference on Multimedia - MM '11*. ACM Press, Scottsdale, Arizona, USA, 699. DOI: http://dx.doi.org/10.1145/2072298.2072423

[8] Nick Collins, Alex McLean, Julian Rohrhuber, and Adrian Ward. 2003. Live Coding in Laptop Performance. *Org. Sound* 8, 3 (Dec. 2003), 321–330. DOI: http://dx.doi.org/10.1017/S135577180300030X

[9] E. Craighill, M. Fong, K. Skinner, R. Lang, and K. Gruenefeldt. 1994. Scoot: an object-oriented toolkit for multimedia collaboration. In *Proceedings of the second ACM international conference on Multimedia - MULTIMEDIA '94*. ACM Press, San Francisco, California, United States, 41–49. DOI: http://dx.doi.org/10.1145/192593.192618

[10] Blackmagic Design. 2019. Davinci Resolve 15. https://www.blackmagicdesign.com/dk/products/davinciresolve/. (2019). Accessed: 2018-09-29.

[11] MDN Web Docs. 2019a. HTMLMediaElement.seekToNextFrame(). https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement/seekToNextFrame. (2019). Accessed: 2019-07-13.

[12] MDN Web Docs. 2019b. MediaDevices.getUserMedia(). https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia. (2019). Accessed: 2019-07-13.

[13] Webstrates Documentation. 2019a. Assets. https://webstrates.github.io/userguide/api/assets.html. (2019). Accessed: 2019-07-13.

[14] Webstrates Documentation. 2019b. Signaling. https://webstrates.github.io/userguide/api/signaling.html. (2019). Accessed: 2019-07-13.

[15] A. Engström, M. Esbjörnsson, and O. Juhlin. 2008. Mobile collaborative live video mixing. In *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services - MobileHCI '08*. ACM Press, Amsterdam, The Netherlands, 157. DOI: http://dx.doi.org/10.1145/1409240.1409258

[16] Firejail. 2019. Firejail Security Sandbox. https://firejail.wordpress.com. (2019). Accessed: 2019-07-13.

[17] Adam Fouse, Nadir Weibel, Edwin Hutchins, and James D Hollan. 2011. ChronoViz: a system for supporting navigation of time-coded data. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*. ACM, 299–304. DOI: http://dx.doi.org/10.1145/1979742.1979706

[18] Boni Garcia, Luis López, Francisco Gortázar, Micael Gallego, and Giuseppe Antonio Carella. 2017. NUBOMEDIA: The First Open Source WebRTC PaaS. In *Proceedings of the 2017 ACM on Multimedia Conference - MM '17*. ACM Press, Mountain View, California, USA, 1205–1208. DOI: http://dx.doi.org/10.1145/3123266.3129392

[19] Jérémie Garcia, Theophanis Tsandilas, Carlos Agon, and Wendy Mackay. 2012. Interactive Paper Substrates to Support Musical Creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 1825–1828. DOI: http://dx.doi.org/10.1145/2207676.2208316

[20] William Gaver, Thomas Moran, Allan MacLean, Lennart Lövstrand, Paul Dourish, Kathleen Carter, and William Buxton. 1992. Realizing a video environment: EuroPARC's RAVE system. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92*. ACM Press, Monterey, California, United States, 27–35. DOI: http://dx.doi.org/10.1145/142750.142754

[21] David Geerts, Ishan Vaishnavi, Rufael Mekuria, Oskar van Deventer, and Pablo Cesar. 2011. Are we in sync?: synchronization requirements for watching online video together.. In *Proceedings of the 2011 annual conference*

on Human factors in computing systems - CHI '11. ACM Press, Vancouver, BC, Canada, 311. DOI: http://dx.doi.org/10.1145/1978942.1978986

[22] Simon Gibbs, Christian Breiteneder, Vicki de Mey, and Michael Papathomas. 1993. Video widgets and video actors. In *Proceedings of the 6th annual ACM symposium on User interface software and technology - UIST '93*. ACM Press, Atlanta, Georgia, United States, 179–185. DOI: http://dx.doi.org/10.1145/168642.168660

[23] David Philip Green, Simon J. Bowen, Christopher Newell, Guy Schofield, Tom Bartindale, Clara Crivellaro, Alia Sheikh, Peter Wright, and Patrick Olivier. 2015. Beyond Participatory Production: Digitally Supporting Grassroots Documentary. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*. ACM Press, Seoul, Republic of Korea, 3157–3166. DOI: http://dx.doi.org/10.1145/2702123.2702203

[24] GStreamer. 2019. GStreamer Open Source Multimedia Framework. https://gstreamer.freedesktop.org. (2019). Accessed: 2019-07-13.

[25] Nuno M. Guimarães, Nuno M. Correia, and Telmo A. Carmo. 1992. Programming time in multimedia user interfaces. In *Proceedings of the 5th annual ACM symposium on User interface software and technology - UIST '92*. ACM Press, Monteray, California, United States, 125–134. DOI: http://dx.doi.org/10.1145/142621.142637

[26] William A. Hamilton, Nic Lupfer, Nicolas Botello, Tyler Tesch, Alex Stacy, Jeremy Merrill, Blake Williford, Frank R. Bentley, and Andruid Kerne. 2018. Collaborative Live Media Curation: Shared Context for Participation in Online Learning. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. ACM Press, Montreal QC, Canada, 1–14. DOI: http://dx.doi.org/10.1145/3173574.3174129

[27] Oskar Juhlin, Arvid Engström, and Erika Reponen. 2010. Mobile broadcasting: the whats and hows of live video as a social medium. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services - MobileHCI '10*. ACM Press, Lisbon, Portugal, 35. DOI: http://dx.doi.org/10.1145/1851600.1851610

[28] Jun Kato, Tomoyasu Nakano, and Masataka Goto. 2015. TextAlive: Integrated Design Environment for Kinetic Typography. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 3403–3412. DOI:http://dx.doi.org/10.1145/2702123.2702140

[29] Ryan Kirkbride. 2017. Troop: A Collaborative Tool for Live Coding. In *Proceedings of the 14th Sound and Music Computing Conference*. Aalto University, 104–9.

[30] Clemens N Klokmose, James R Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: shareable dynamic media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM, 280–290. DOI: http://dx.doi.org/10.1145/2807442.2807446

[31] Walter S. Lasecki, Mitchell Gordon, Danai Koutra, Malte F. Jung, Steven P. Dow, and Jeffrey P. Bigham. 2014. Glance: rapidly coding behavioral video with the crowd. In *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*. ACM Press, Honolulu, Hawaii, USA, 551–562. DOI: http://dx.doi.org/10.1145/2642918.2647367

[32] Mackenzie Leake, Abe Davis, Anh Truong, and Maneesh Agrawala. 2017. Computational Video Editing for Dialogue-driven Scenes. *ACM Trans. Graph.* 36, 4, Article 130 (July 2017), 14 pages. DOI: http://dx.doi.org/10.1145/3072959.3073653

[33] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation strategies for HCI toolkit research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 36. DOI: http://dx.doi.org/10.1145/3173574.3173610

[34] Wendy E. Mackay and Michel Beaudouin-Lafon. 1998. DIVA: exploratory data analysis with multimedia streams. In *Conference on Human Factors in Computing Systems: Proceedings of the SIGCHI conference on Human factors in computing systems*, Vol. 18. 416–423. DOI:http://dx.doi.org/10.1145/274644.274701

[35] Wendy E. Mackay and Glorianna Davenport. 1989. Virtual Video Editing in Interactive Multimedia Applications. *Commun. ACM* 32, 7 (July 1989), 802–810. DOI: http://dx.doi.org/10.1145/65445.65447

[36] James Matthews, Peter Gloor, and Fillia Makedon. 1993. VideoScheme: A Programmable Video Editing Systems for Automation and Media Recognition. In *Proceedings of the First ACM International Conference on Multimedia (MULTIMEDIA '93)*. ACM, New York, NY, USA, 419–426. DOI: http://dx.doi.org/10.1145/166266.168442

[37] Ketan Mayer-Patel, David Simpson, David Wu, and Lawrence A. Rowe. 1996. Synchronized continuous media playback through the World Wide Web. In *Proceedings of the fourth ACM international conference on Multimedia - MULTIMEDIA '96*. ACM Press, Boston, Massachusetts, United States, 435–436. DOI: http://dx.doi.org/10.1145/244130.244458

[38] Joel McCormack and Paul Asente. 1988. An overview of the X toolkit. In *Proceedings of the 1st annual ACM SIGGRAPH symposium on User Interface Software - UIST '88*. ACM Press, Alberta, Canada, 46–55. DOI: http://dx.doi.org/10.1145/62402.62407

[39] Mozilla. 2019. Firefox. **https://www.mozilla.org/firefox/**. (2019). Accessed: 2019-07-13.

[40] Boris Novikov and Oleg Proskurnin. 2003. Towards Collaborative Video Authoring. In *Advances in Databases and Information Systems*, Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Leonid Kalinichenko, Rainer Manthey, Bernhard Thalheim, and Uwe Wloka (Eds.). Vol. 2798. Springer Berlin Heidelberg, Berlin, Heidelberg, 370–384. DOI: **http://dx.doi.org/10.1007/978-3-540-39403-7_28**

[41] Dan R Olsen Jr. 2007. Evaluating user interface systems research. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. ACM, 251–258. DOI: **http://dx.doi.org/10.1145/1294211.1294256**

[42] Rui Pan and Carman Neustaedter. 2017. Streamer.Space: A Toolkit for Prototyping Context-Aware Mobile Video Streaming Apps. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '17*. ACM Press, Denver, Colorado, USA, 1947–1954. DOI: **http://dx.doi.org/10.1145/3027063.3053083**

[43] Amy Pavel, Dan B. Goldman, Björn Hartmann, and Maneesh Agrawala. 2016. VidCrit: Video-based Asynchronous Video Review. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology - UIST '16*. ACM Press, Tokyo, Japan, 517–528. DOI: **http://dx.doi.org/10.1145/2984511.2984552**

[44] Roman Rädle, Midas Nouwens, Kristian Antonsen, James R Eagan, and Clemens N Klokmose. 2017. Codestrates: Literate Computing with Webstrates. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 715–725. DOI: **http://dx.doi.org/10.1145/3126594.3126642**

[45] Gonzalo Ramos and Ravin Balakrishnan. 2003. Fluid interaction techniques for the control and annotation of digital video. In *Proceedings of the 16th annual ACM symposium on User interface software and technology - UIST '03*. ACM Press, Vancouver, Canada, 105–114. DOI:**http://dx.doi.org/10.1145/964696.964708**

[46] Stuart Reeves, Christian Greiffenhagen, Martin Flintham, Steve Benford, Matt Adams, Ju Row Farr, and Nicholas Tandavantij. 2015. I'd Hide You: Performing Live Broadcasting in Public. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*. ACM Press, Seoul, Republic of Korea, 2573–2582. DOI: **http://dx.doi.org/10.1145/2702123.2702257**

[47] Nicolas Roussel and Sofiane Gueddana. 2007. Beyond "beyond being there": towards multiscale communication systems. In *Proceedings of the 15th international conference on Multimedia - MULTIMEDIA '07*. ACM Press, Augsburg, Germany, 238. DOI: **http://dx.doi.org/10.1145/1291233.1291283**

[48] Pagani Daniele S. and Wendy E. Mackay. 1989. Bringing Media Spaces into the Real World. *Proceedings of the Third European Conference on Computer-Supported Cooperative Work* 32, 7 (July 1989), 802–810. DOI: **http://dx.doi.org/10.1145/65445.65447**

[49] Klaus Schoeffmann, Marco A. Hudelist, and Jochen Huber. 2015. Video Interaction Tools: A Survey of Recent Work. *Comput. Surveys* 48, 1 (Sept. 2015), 1–34. DOI**:http://dx.doi.org/10.1145/2808796**

[50] Manuel Serrano. 2007. Programming web multimedia applications with hop. In *Proceedings of the 15th international conference on Multimedia - MULTIMEDIA '07*. ACM Press, Augsburg, Germany, 1001. DOI: **http://dx.doi.org/10.1145/1291233.1291450**

[51] Tomas Sokoler, Håkan Edeholt, and Martin Johansson. 2002. VideoTable: A Tangible Interface for Collaborative Exploration of Video Material During Design Sessions. In *CHI '02 Extended Abstracts on Human Factors in Computing Systems (CHI EA '02)*. ACM, New York, NY, USA, 656–657. DOI: **http://dx.doi.org/10.1145/506443.506531**

[52] Statista. 2019. Youtube statistics from Statista. **https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/**. (2019). Accessed: 2018-09-26.

[53] The Skype Team. 2019. Microsoft: 10 years of Skype. **https://blogs.skype.com/stories/2016/01/12/ten-years-of-skype-video-yesterday-today-and-something-new/**. (2019). Accessed: 2018-09-26.

[54] Lucia Terrenghi, Torsten Fritsche, and Andreas Butz. 2008. Designing environments for collaborative video editing. In *2008 IET 4th International Conference on Intelligent Environments*. 1–7. DOI: **http://dx.doi.org/10.1049/cp:20081137**

[55] three.js. 2019. WebGL Video Demo. **https://threejs.org/examples/#webgl_materials_video**. (2019). Accessed: 2019-07-13.

[56] TwitchTracker. 2019. Twitch tracker statistics. **https://twitchtracker.com/statistics**. (2019). Accessed: 2018-09-26.

[57] Justin Uberti and Peter Thatcher. 2019. WebRTC. **https://www.w3.org/TR/webrtc/**. (2019). Accessed: 2018-10-01.

[58] Matthias Wagner and Wolfgang Kellerer. 2004. Web services selection for distributed composition of multimedia content. In *Proceedings of the 12th annual ACM international conference on Multimedia - MULTIMEDIA '04*. ACM Press, New York, NY, USA, 104. DOI: **http://dx.doi.org/10.1145/1027527.1027546**

[59] Ge Wang and Perry Cook. 2004. ChucK: a programming language for on-the-fly, real-time audio synthesis and multimedia. In *Proceedings of the 12th annual ACM international conference on Multimedia - MULTIMEDIA '04*. ACM Press, New York, NY, USA, 812. DOI: http://dx.doi.org/10.1145/1027527.1027716

[60] Webstrates. 2019. Webstrates File System. https://github.com/Webstrates/file-system. (2019). Accessed: 2019-07-13.

[61] Herve Yviquel, Antoine Lorence, Khaled Jerbi, Gildas Cocherel, Alexandre Sanchez, and Mickael Raulet. 2013. Orcc: multimedia development made easy. In *Proceedings of the 21st ACM international conference on Multimedia - MM '13*. ACM Press, Barcelona, Spain, 863–866. DOI: http://dx.doi.org/10.1145/2502081.2502231

[62] Jamie Zigelbaum, Michael S. Horn, Orit Shaer, and Robert J. K. Jacob. 2007. The tangible video editor: collaborative video editing with active tokens. In *Proceedings of the 1st international conference on Tangible and embedded interaction - TEI '07*. ACM Press, Baton Rouge, Louisiana, 43. DOI: http://dx.doi.org/10.1145/1226969.1226978

APPENDIX

This appendix gives an overview of the Videostrates format and the vStreamer API.

Videostrates format

DOM elements with the `composited` class define clips that are part of the video. They can have the following attributes:

- `data-start` When the clip should start in the timeline of the video (in seconds).
- `data-end` When the clip should end in the timeline of the video (in seconds).
- `data-offset` The offset into the source timeline, e.g., the offset into a source video clip or into the timed CSS animation of an element (in seconds).
- `data-speed` The playback speed of the video in multiples of the original speed.

Custom elements can be created using the `custom composited` classes. The developer must attach a `customLocalSeek` function to the given element, which is called with the current time during seeking or playback. Videostrates also supports compositing videos from other videostrates using transclusion. Listing 5 shows the HTML for compositing a part of another videostrate into the final video.

```
<div class="custom composited videostrate"
  data-src="<webstrates server>/anotherVideostrate"
  data-start="15" data-end="12" data-offset="5">
</div>
```

Listing 5. Composing in a videoclip from another videostrate.

Videostrates has a standard library of CSS animations for transitions, for creating subtitles and for scrolling credits.

Transitions can be added to a composited element by specifying the `animation-name` and `animation-duration` CSS properties (see Listing 6). The Videostrates library currently contains 23 transition types, including `compositor_transition_fade`, `compositor_transition_barndoor_horizontal_in` or `compositor_transition_radial_hard_out`.

```
<video class="composited" src="clip.mp4" data-start="5"
  data-end="15" data-offset="5"
  style="animation-name: compositor_transition_fade;
  animation-duration: 1.5s;">
</video>
```

Listing 6. HTML of a clip with a transition.

Subtitles are created using a `DIV` element with the classes `composited subtitles` that contains a `SPAN` element with the classes `composited subtitle` for each subtitle. Each subtitle should have a `data-start` and `data-end` attribute. Listing 7 shows an example of a subtitle track.

```
<div class="composited subtitles">
  <span class="composited subtitle" data-start="729"
    data-end="734">Who was he?</span>
  <span class="composited subtitle" data-start="740"
    data-end="745">I have no idea</span>
</div>
```

Listing 7. Example HTML of a subtitle track

Scrolling credits are created using a `DIV` element with the classes `composited creditscroll` and `data-start` and `data-end` attributes. Any content of the credit scroll element will scroll from the bottom to the top of the screen within the given time. Listing 8 shows and example of scrolling credits.

```
<div class="composited creditscroll" data-start="100"
    data-end="130">
  <h1>Actors</h1>
  <h2>Jane Doe as the nameless protagonist</h2>
  <h2>Jack Doe as the nameless antagonist</h2>
  <h1>Producer</h1>
  <h2>Richard Roe</h2>
</div>
```

Listing 8. Example HTML of scrolling credits

vStreamer API

vStreamer has a simple API to create a controllable stream of a videostrate. Listing 9 shows how to create and play a videostrate specified by its URL from vStreamer.

```
<html>
  <head>
    <script src="vstreamer.js"></script>
  </head>
  <body>
    <video></video>
    <script>
    let videostrateUrl = "<webstrates server>/VideoStrate"
    let vs = new VideostrateView(document.querySelector("
     video"), videostrateUrl, {width: 640, height: 360})
    vs.startStreamView()
    vs.play()
    </script>
  </body>
</html>
```

Listing 9. Using the *vstreamer* API to play a videostrate

The videostrate viewer object `vs` has the following methods:

- `vs.play()` starts playback.
- `vs.stop()` stops playback.
- `vs.seek(seconds)` seeks to a given time in the video.
- `vs.record()` starts recording the stream and returns a promise with the resulting file URL.
- `let stopRecordPromise = vs.stopRecord()` stops recording the stream and returns a promise with the resulting file URL.
- `let renderPromise = vs.render()` starts rendering the videostrate and returns a promise with the resulting file URL.
- `let stopRenderPromise = vs.stopRender()` stops rendering the videostrate and returns a promise with the resulting file URL.
- `vs.broadcast(rtmpTarget)` starts broadcasting the stream to a rtmp-compatible streaming services. The `rtmptarget` is the URL of the stream end point.
- `vs.stopBroadcasting()` stops broadcasting.