Project Disco: Developing a video alternative suitable for online learning

Allan Reyes
Georgia Tech
Atlanta, USA
areyes37@gatech.edu

Govin Vatsan
Georgia Tech
Atlanta, USA
gvatsan@gatech.edu

Robert Almendarez
Georgia Tech
Atlanta, USA
ralmendarez3@gatech.edu

ABSTRACT

Video is the dominant format by which online learning providers deliver their content, yet its production, maintenance, and transmission costs remain the highest. This paper presents an alternative format for online learning video that reduces file sizes, enables open-sourced maintenance and collaboration, and promotes native translation.

Author Keywords
MOOC; online learning; video; animation; bandwidth; HTML5; canvas; web; browser; JavaScript

ACM Classification Keywords
H.5.1. Information Interfaces and Presentation (e.g. HCI): Multimedia Information Systems: Video

INTRODUCTION

An overwhelming majority of online learning and Massive Open Online Course (MOOC) providers utilize video as the primary format to deliver their content. However, video production and maintenance continues to be the major cost center in producing MOOCs and other online courses [5]. Moreover, online learning videos pose several logistical challenges and problems:

1. *Translation*. The scope of translation and internationalization in videos today is limited only to subtitles. Consumers cannot "watch" a video in another language; rather, any text or speech remains in the same native language of which it was produced.

2. *Bandwidth*. Videos that teach technology (e.g. computer programming) often contain large amounts of text. From an information density standpoint, the encoding is far from ideal. For example, a snippet of code (bytes) is rasterized to pixels in video (kilobytes).

3. *Maintainability*. Along the same lines, if there are typos in rasterized text, there are few ways to fix or patch it. For example, typos in YouTube videos are addressed with pop-up overlays, and mistakes in online learning videos are addressed with instructors' notes. With regards to tech-focused videos, the moment the video is produced, it immediately becomes out of date. There's simply no upgrade strategy short of re-editing or re-rendering the video, all costly endeavors.

4. *Closed Source*. Raw materials and footage are unavailable in a compiled video. Unlike collaborative platforms like Wikipedia and GitHub, there is simply no way for users and content producers alike to contribute improvements.

5. *Lack of Semantic Information*. Video does not encapsulate the semantic information and knowledge in a machine-readable format. For example, course notes and summaries are produced separately and are only manually synchronized to the video content.

Several predecessors have attempted to address these problems in part, such as Animatron [2]. This product allows users to edit and drag-and-drop audio, video, text, and image files into a real-time animation rendering. While a good implementation of vectorized video, there are no facilities for translation, and it is geared more towards animators and advertisers than online education. Likewise, Adobe Shockwave Flash [1] attempted similar methods of vectorization, but remained closed source, was riddled with security issues, and was incompatible with modern mobile and web technologies. Finally, a previous OMS student, James Jackson, describes a similar project in his paper, "Designing a MOOC for Constrained Internet Access" [7]. In his work, he created an animated vector graphics video of a slide-based Udacity lecture. He converted each original PowerPoint slide into an SVG image and then animated those images in real-time in the browser. However, this work did not support customizable text or audio components. To our knowledge, no technology exists that addresses the presented challenges in whole. Our work, inspired by the predecessors, attempts to build off of and improve previous efforts.

IMPLEMENTATION

The library is composed of three major components: the engine, the web player, and the renderer. To initiate, a base "Disco XML" format is inputted to the engine. The XML specification annotates a semantic representation of the video alternative, as well as the location and timing of static assets, including audio, images and video. The XML is processed by the engine, which feeds the appropriate timing and information to the renderer. The renderer interacts with the web player and canvas elements to paint and play the individual Disco elements at the correct time and location. Figure 1 shows the architecture of the Disco library.

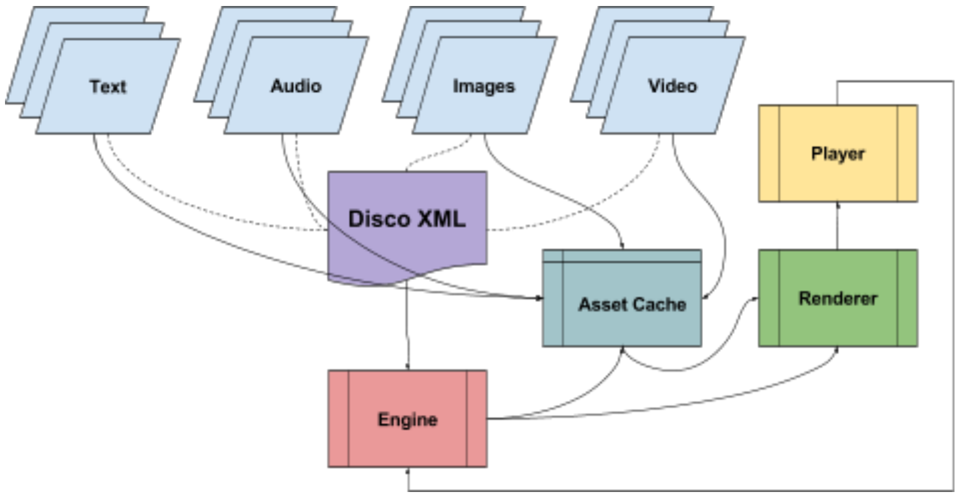


Figure 1: Architecture of the Disco library

Engine
The engine parses the XML and transforms the schema based on the selected language or locale. This component then pre-downloads the required assets into a local cache. The engine accepts inputs from the client via the web player, and communicates with the renderer to specify where and when content is displayed.

Web Player
At a high level, the web player is constructed with simple HTML, CSS, and JavaScript. Whenever a user interacts with the player (e.g. press pause), the web player notifies the engine, which takes the corresponding action (e.g. pauses the video).

Renderer
The renderer is the final component loaded by the main Disco.js script. This was done to ensure that all assets would be preloaded first, enabling the engine to register all of the assets with the renderer during initialization. Additionally, this guarantees that all disco components are initialized first before attempting to render assets, since all signals are routed from the web player, through the engine, before finally reaching the renderer.

During initialization, the renderer takes in a reference to the HTML5 Canvas that is first created by the web player. Using CreateJS's EaselJS, a stage is created on the canvas to display visual assets, and using TweenJS, a timeline is built to manage asset animation, called tweens [3].

After initialization, the engine gathers the loaded assets from the processed XML and registers each asset with the renderer, frame by frame. The renderer breaks apart each frame into its asset component – called atoms. Each atom is specialized to the asset type it contains and therefore knows how to start, stop, display, hide, and seek the asset. Every atom is then built up into a tween that is registered with the timeline. Each tween has a start and stop action based on the frame's specification. This enables the renderer to synchronize atoms by frame, ensuring that all assets are displayed at their correct times, and in the correct order.

Finally, once all assets are registered, the timeline is kicked off by the renderer – which starts a process known as the global 'tick' event. This tick event is left constantly running, even when the overall disco is paused in order to ensure that the tweens are always synchronized.

When the user presses the play button on the web player, the engine signals to the renderer to "un-pause" the timeline. The timeline is run in milliseconds, with every tween having its own start and stop time. While the timeline is running, the ticker constantly updates each tween with the current elapsed time. Once each tween's start time has elapsed, the tween executes its atom specific start action. This means the atom will transition its visibility attribute to true, thus displaying it on the stage. Once the tween's stop time has elapsed, the tween executes its stop action, which signals the atom to transition its visibility attribute to false, hiding it from view. For each atom type additional processing also occurs during the stop action to reset the tween and guarantee it's still in a valid state in case the user later seeks back into that tween's active time.

When the user presses the pause button in the web player, the engine signals to the renderer to halt the timeline, pausing all tweens. Importantly, the ticker's tick event still occurs, which allows the tweens to stay synchronized and enables them to be seeked out while still paused. Also, the renderer pauses the progress bar animation. This is important because the progress bar's animation, which is where the bar moves along with the elapsed time of the disco, is provided by the renderer itself, as opposed to the web player. This is because only the renderer has information regarding each asset's duration and location on the timeline, which means that only the renderer knows how long the actual disco is and where the user is in the current disco.

When the user clicks the left mouse button down on the progress bar, the "mousedown" event will trigger the engine to call the renderer's preview() function. This function informs the renderer that the user is currently "seeking" via the progress bar, and therefore the renderer needs to pause the progress bar's animation. Pausing the progress bar here ensures that the bar is not skipping or jumping between the current location and the user's desired location while the user is seeking.

Finally, when the user releases the left mouse button from the progress bar, the "mouseup" event triggers the engine to call the renderer's seek() function. The seek() function fires off a custom "seek" event that tells the timeline to move to the new position selected by the user, and informs the audible atoms (video and audio) to move to their new locations. Moving the timeline ensures that the proper animations and assets are playing after the new time selection and that any tweens that were in play but now are not should have their stop actions executed. Upon completion of the seek, animation for the progress bar is re-enabled, thus completing the full seek operation.

RESULTS

To show the effectiveness of our format, we converted one video clip from the Georgia Tech Knowledge-Based Artificial Intelligence course [4] to a disco. The video clip consists of five separate image slides, an audio overlay of the professor's voice, and a "floating hand" - which is the professor pointing at various elements on each image. To convert the lecture to disco, we first broke it down into its "atoms" - that is the minimal individual components with which we could re-create the original video. We separated these atoms into four categories: audio, image, text, and video. We then added a Spanish translation of the audio and text elements. This process resulted in:

1. Five image atoms stored as png files (note that text was removed from the images)

2. Ten audio atoms stored as ogg files (one to correspond with each image for both English and Spanish)

3. 42 text atoms (Each text atom corresponds to a line of text from the original lecture slides, 21 for each language)

4. 21 image atoms to store the virtual pointer.

Then we wrote an XML script to link these static assets together with the timing information needed to render them together in real-time in the browser. See the Appendix for a sample portion of this XML script.

We were then able to use our disco player to render and re-create the video lecture in real-time in the browser without any noticeable loss of quality. In fact, we were able to improve upon the original video by adding a Spanish audio and text translation to the disco format. Figure 2 shows an example of slide 1 from the original lecture, with a floating hand. Figure 3 shows the image atom we created from that slide. Figure 4 shows the English translation of our slide seen during runtime as a disco, and Figure 5 shows the Spanish translation seen during runtime. Note that all text and pointers on a slide is overlaid during runtime.

The sum total of all the data files necessary to render our disco was 2.2 MB, including the additional Spanish atoms. By contrast, the original mp4 file with no Spanish translation was 13.2 MB. That is a 6x file reduction achieved by using our custom format and player. And without the Spanish audio, our data files total to 1.6 MB, which is an 8.25x filesize reduction from the original. This clearly shows there is a strong case to be made for using a component based approach to video creation.

FUTURE WORK

There are many aspects to our format that could be improved. Currently, there is no support for offline access. One simple solution would just be to convert a disco format to a traditional video format, such as an mp4, if a user wants to download a disco. We also do not have a true "floating hand" as seen in the original lectures. Instead, we created a "virtual pointer" by using 8 images to point to various parts of each slide. A more realistic pointer could be created by using one pointer image and then animating it across each slide to replicate the movement of the professor's hand. This was successfully created in Jackson's paper [7]. Additionally, we could improve the current GUI of the disco player to better interface with the XML format. This would help users with maintaining and customizing videos, especially those who do not have a background in web development. However, given the promising results our prototype has shown, we believe the most interesting future work lies in scaling this format into an open-source, widely available tool compatible with any type of video, not just purely educational.

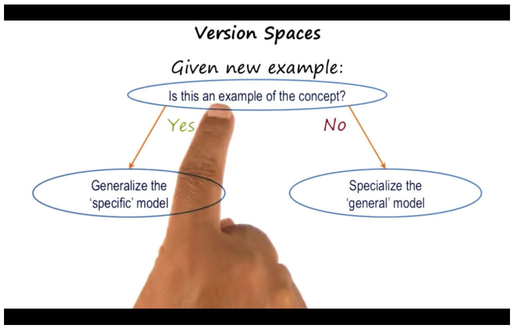

Version Spaces
Given new example:
Is this an example of the concept?
Yes
No
Generalize the 'specific' model
Specialize the 'general' model

Figure 2: Slide 1 from the original KBAI lecture

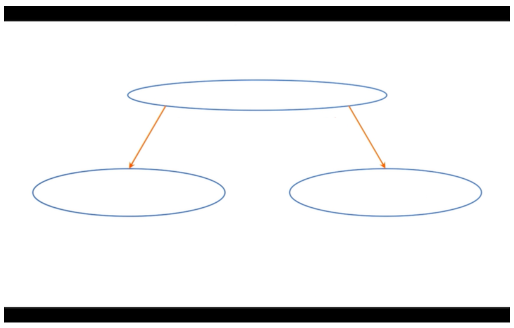


Figure 3: Image atom created from slide 1 (text and pointers removed)

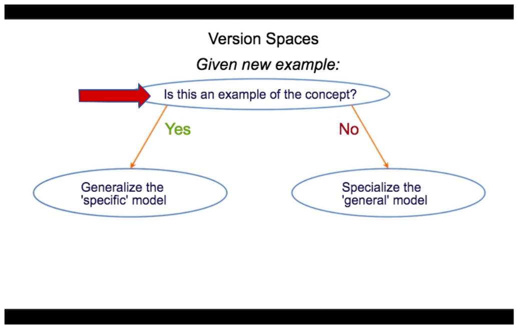

Version Spaces
Given new example:
Is this an example of the concept?
Yes
No
Generalize the 'specific' model
Specialize the 'general' model

Figure 4: English translation of slide 1 as seen during runtime

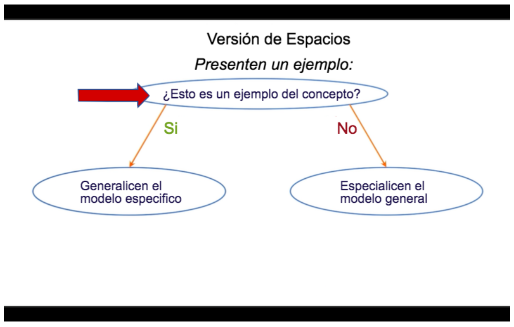

Versión de Espacios
Presenten un ejemplo:
¿Esto es un ejemplo del concepto?
Si
No
Generalicen el modelo especifico
Especialicen el modelo general

Figure 5: Spanish translation of slide 1 as seen during runtime

CONCLUSION

We believe that Disco has potential to be the preeminent format of online learning. It is our belief that adoption of this medium helps online providers produce content that is scalable, encourages open- and crowd-sourced contribution, democratizes access to demographics with low bandwidth, and provides rapid translation and internationalization. Disco is a modern video alternative, leveraging modern web technologies, built for present and future internet users. While the technology demonstrated addresses the logistical challenges associated with video in online learning, it does not directly address pedagogical shortcomings. Given the ubiquity with which traditional video formats are used in online learning, limited research has been done to confirm if it is the ideal medium and an effective method for learning [6]. Yet, Disco provides the raw canvas that enables providers to conduct that research. Disco empowers providers to rapidly iterate and improve their content, present robust and personalized experiences to online learners, and produce content ideal for tomorrow's internet users.

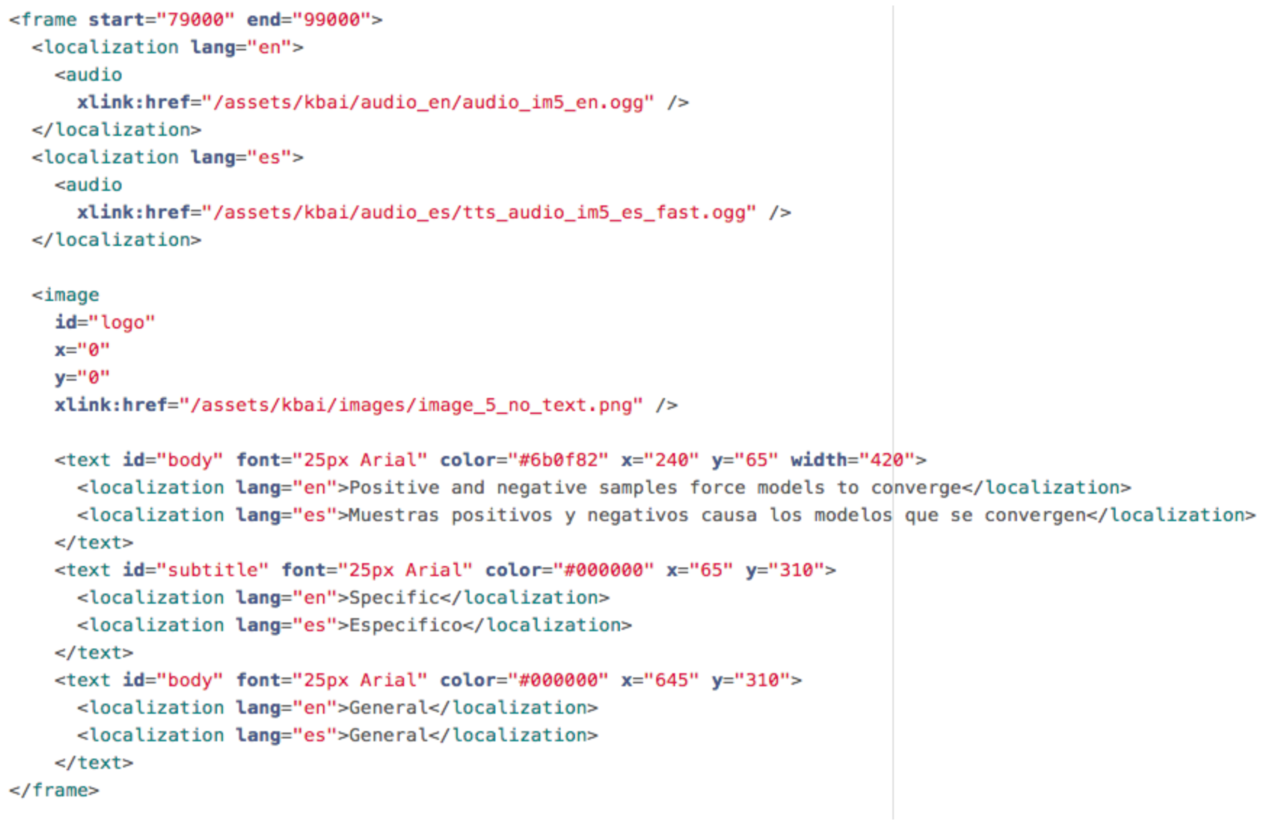
ACKNOWLEDGMENTS

We would like to thank Dr. Joyner and Chris Kreider for the mentoring support they provided, and we would also like to thank the students of OMS 6460 for reviewing and offering valuable critique on our intermediate milestones.

REFERENCES

1. Adobe Flash Player. Retrieved April 29, 2017 from https://www.adobe.com/products/flashplayer.html
2. Animatron. Retrieved April 29, 2017 from https://animatron.com/
3. CreateJS. Retrieved April 29, 2017 from http://www.createjs.com/
4. Ashok Goel. Knowledge Based Artificial-Intelligence, Abstract Version Spaces. Georgia Institute of Technology. Retrieved April 29, 2017 from http://udacity.com/
5. Philip J. Guo, Juho Kim, and Rob Rubin. 2014. How video production affects student engagement: an empirical study of MOOC videos. In Proceedings of the first ACM conference on Learning @ scale conference (L@S '14). ACM, New York, NY, USA, 41-50. DOI:http://dx.doi.org/10.1145/2556325.2566239
6. Anna Hansch, Lisa Hillers, Katherine Mcconachie, Christopher Newman, Thomas Schildhauer, and Philipp Schmidt. 2015. Video and Online Learning: Critical Reflections and Findings from the Field. SSRN Electronic Journal (March 2015). DOI:http://dx.doi.org/10.2139/ssrn.2577882
7. James Jackson. Designing a MOOC for Constrained Internet Access. 2016. Georgia Institute of Technology.

APPENDIX



```xml
<frame start="79000" end="99000">
  <localization lang="en">
    <audio
      xlink:href="/assets/kbai/audio_en/audio_im5_en.ogg" />
  </localization>
  <localization lang="es">
    <audio
      xlink:href="/assets/kbai/audio_es/tts_audio_im5_es_fast.ogg" />
  </localization>

  <image
    id="logo"
    x="0"
    y="0"
    xlink:href="/assets/kbai/images/image_5_no_text.png" />

    <text id="body" font="25px Arial" color="#6b0f82" x="240" y="65" width="420">
      <localization lang="en">Positive and negative samples force models to converge</localization>
      <localization lang="es">Muestras positivos y negativos causa los modelos que se convergen</localization>
    </text>
    <text id="subtitle" font="25px Arial" color="#000000" x="65" y="310">
      <localization lang="en">Specific</localization>
      <localization lang="es">Especifico</localization>
    </text>
    <text id="body" font="25px Arial" color="#000000" x="645" y="310">
      <localization lang="en">General</localization>
      <localization lang="es">General</localization>
    </text>
</frame>
```

This sample XML script was used to render the fifth slide of the KBAI video lecture, that is the slide starting at 79000 ms and ending at 99000 ms.