CS7026 – HTML5

Video
Video on the Web

- Until HTML5, there has never been a standard for showing video on a web page.

- Previously, most videos were shown through a plugin (mainly Flash).

- However, not all browsers have the same plugins.
In 2007, Anne van Kesteren (Opera R&D) wrote to the Working Group:

“Opera has some internal experimental builds with an implementation of a `<video>` element. The element exposes a simple API (for the moment) much like the `audio()` object: `play()`, `pause()`, `stop()`. The idea is that it works like `<object>` except that it has special `<video>` semantics much like `<img>` has image semantics.”

While the API has increased in complexity, van Kesteren’s original announcement is now implemented in all the major browsers, including Internet Explorer 9+. 
Why do you need a `<video>` element?

- Previously, if developers wanted to include video in a web page, they had to make use of the `<object>` element, which is a generic container for “foreign objects.”

- Due to browser inconsistencies, they would also need to use the previously invalid `<embed>` element and duplicate many parameters.
Why do you need a `<video>` element?

- This resulted in code that looked much like this:

  ```html
  <object width="425" height="344">
  <param name="movie" value="http://www.youtube.com/v/9sEl1AUFJKw&hl=en_GB&fs=1"></param>
  <param name="allowFullScreen" value="true"></param>
  <param name="allowScriptAccess" value="always"></param>
  <embed src="http://www.youtube.com/v/9sEl1AUFJKw&hl=en_GB&fs=1" type="application/x-shockwave-flash" allowScriptAccess="always" allowfullscreen="true" width="425" height="344"></embed>
  </object>
  ```
Problems...

- This code is ugly and ungainly but worse than that is the fact that the browser has to pass the video off to a third-party plugin.

- The user has to have the correct version of that plugin (or the rights to download and install it and the knowledge of how to do so).
More problems...

- We then have to hope that the plugin is keyboard accessible - along with all the other unknowns involved in handing the content to a third-party application.

- Plugins can also be a significant cause of browser instability and can create worry in less technical users when they are prompted to download and install newer versions.
And another problem...

- Whenever you include a plugin in your pages, you’re reserving a certain drawing area that the browser delegates to the plugin.

- As far as the browser is concerned, the plugin’s area remains a black box - the browser does not process or interpret anything that is happening there.
And another problem...

- Normally, this is not a problem, but issues can arise when your layout overlaps the plugin’s drawing area.

- E.g., a site that contains a movie but also has JavaScript or CSS-based dropdown menus that need to unfold over the movie. By default, the plugin’s drawing area sits on top of the web page, meaning that these menus will strangely appear behind the movie.
And another problem...

- Problems and quirks can also arise if your page has dynamic layout changes.

- If the dimensions of the plugin’s drawing area are resized, this can sometimes have unforeseen effects—a movie playing in the plugin may not resize, but instead simply be cropped or display extra white space.
The Solution...

- HTML5 provides a standardised way to play video directly in the browser, with no plugins required.

- Finally, video is a full-fledged citizen on the Web. It’s no longer shunted off to the hinterland of `<object>` or the non-validating `<embed>` element.
The Solution…

- Now `<video>` elements can be styled with CSS; they can be resized on hover using CSS transitions, for example.

- They can be tweaked, controlled and redisplayed with JavaScript.
The Solution...

- Best of all, the innate hackability that open web standards provide is opened up.

- Previously, all your video data was locked away; all trapped in a box.

- With HTML5, you are free to manipulate your data any way you want.
Browser support

- The numbers in the table specify the first browser version that fully supports the `<video>` element.

<table>
<thead>
<tr>
<th>Element</th>
<th>Chrome</th>
<th>Edge</th>
<th>Firefox</th>
<th>Safari</th>
<th>Opera</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;video&gt;</code></td>
<td>4.0</td>
<td>9.0</td>
<td>3.5</td>
<td>4.0</td>
<td>10.5</td>
</tr>
</tbody>
</table>
How does it Work?

- At its simplest, including video on a page in HTML5 merely requires this code:
  `<video src="movie.ogg"></video>`

- However, this example won’t actually do anything just yet. All you can see here is the first frame of the movie.

- That’s because you haven’t told the video to play, and you haven’t told the browser to provide any controls for playing or pausing the video.
What if it doesn’t?

- Similar to `<object>`, you can put fallback markup between the tags, for older Web browsers that do not support native video.

- You should at least supply a link to the video so users can download it to their hard drives and watch it later on the operating system’s media player:

```html
<h1>Video and legacy browser fallback</h1>
<video src="movie.ogv"> Download the <a href="movie.ogv">How to play a video</a>
</video>
```
autoplay

- You can tell the browser to play the video or audio automatically.

- But many users (and particularly those using assistive technology, such as a screen reader) will find it highly intrusive.

- Users on mobile devices may not want you using their bandwidth without them explicitly asking for the video.

- Nevertheless, here’s how you do it:

  `<video src="movie.ogg" autoplay></video>`
Providing controls is waaaaay better than autoplaying your video.

You can use some simple JavaScript to write your own or you can tell the browser to provide them automatically:

```html
<video src="movie.ogv" controls></video>
```

Naturally, these differ between browsers, but you’ll find nothing too surprising. There’s a play/ pause toggle, a seek bar, and volume control.
The **poster** attribute points to an image that the browser will use while the video is downloading, or until the user tells the video to play.

It removes the need for additional tricks like displaying an image and then removing it via JavaScript when the video is started.

If you don’t use the poster attribute, the browser shows the first frame of the movie, which may not be the representative image you want to show.
height, width

- These attributes tell the browser the size in pixels of the video.

- If you leave them out, the browser uses the intrinsic width of the video resource, if that is available.

- Otherwise it is the intrinsic width of the poster frame, if that is available.

- Otherwise it is 300 pixels.
If you specify one value, but not the other, the browser adjusts the size of the unspecified dimension to preserve the video’s aspect ratio.

If you set both width and height to an aspect ratio that doesn’t match that of the video, the video is not stretched to those dimensions but is rendered “letter-boxed” inside the video element of your specified size while retaining the aspect ratio (in theory!).
The loop attribute is a Boolean attribute.

```html
<video src="movie.ogg" controls loop> Download the
<a href="movie.ogg">How to play a
video</a></video>
```

As you would imagine, it loops the media playback.
preèload

- Maybe you’re pretty sure that the user wants to activate the media (he’s drilled down to it from some navigation, for example, or it’s the only reason to be on the page), but you don’t want to use autoplay.

- If so, you can suggest that the browser preload the video so that it begins buffering when the page loads in the expectation that the user will activate the controls.

<video src="movie.ogv" controls preload> </video>
There are three spec-defined states of the preload attribute.

If you just say preload, the user agent can decide what to do.

A mobile browser may, for example, default to not preloading until explicitly told to do so by the user.
preload

1. `preload="auto"` (or just `preload`)
   - A suggestion to the browser that it should begin downloading the entire file.
   - Note “suggestion.” The browser may ignore this—perhaps because it detected very slow connection or a setting in a mobile browser “Never preload media” to save the user’s bandwidth.

2. `preload="none"`
   - This state suggests to the browser that it shouldn’t preload the resource until the user activates the controls.
3. **preload=“metadata”**

   This state suggests to the browser that it should just prefetch metadata (dimensions, first frame, track list, duration, and so on) but that it shouldn’t download anything further until the user activates the controls.
As on an `<img>`, this attribute points to the file to be displayed.

However, because not all browsers can play the same formats, in production environments you need to have more than one source file (more anon...).

Using a single source file with the `src` attribute is only really useful for rapid prototyping or for intranet sites where you know the user’s browser and which codecs it supports.
Codec Wars

- Early drafts of the HTML5 specification mandated that all browsers should at least have built-in support for multimedia in two codecs:
  - Ogg Vorbis for audio
  - Ogg Theora for video.

- *Vorbis* is a codec used by services like Spotify, among others, and for audio samples in games like Microsoft Halo, it’s often used with Theora for video and combined together in the Ogg container format.
 Codec Wars

- However, these codecs were dropped from the HTML5 spec after Apple and Nokia objected, so the spec makes no recommendations about codecs at all.

- So currently, there are 3 supported video formats for the `<video>` element: **MP4**, **WebM**, and **Ogg**:
  - **MP4** = MPEG 4 files with H.264 video codec and AAC audio codec
  - **WebM** = WebM files with VP9 video codec and Opus audio codec
  - **Ogg** = Ogg files with Theora video codec and Vorbis audio codec
Contender #1: H.264/MPEG-4 AVC (MP4)

- **Relevant Companies:** MPEG-LA (and many more…)
  - **Container:** MPEG-4

- **Overview:**
  - Dozens of major companies are weighing in on H.264 with over 1700 patent claims.
  - As the most popular and widespread, especially on the software side, they’re looking to maintain dominance and not be undercut by open source alternatives.

- **Benefits:**
  - Already very widely adopted, and is a key player in high definition.
  - Every Blu-ray player must be able to decode it. It is also widely used on streaming websites, and has native support by many browsers.
  - The video quality is said to be slightly better, although in side-by-side comparisons the advantage isn’t obvious.
Contender #1: H.264/MPEG-4 AVC

- **Drawbacks:**
  - As it is not open source, license fees are required for software distribution, and there could be limits to how quickly it can evolve without an open source community behind it.

- **Shots Fired:**
  - In early February ‘11, MPEG-LA claimed that the competitors, Google’s VP8 in particular, infringed on their patents.
  - They’re looking to be the #1 standard in HTML5, or, at minimum, to be paid patent fees by (what they consider to be) an infringing Google.
Contender #2: VP8/9

- **Relevant Companies:** Google
  - **Container:** WebM

- **Overview:**
  - Building off the bought technologies of On2, Xiph, and Matroska, VP8 and VP9 represents Google’s side of things.
  - Notably distinct as (a fairly) open source option, it is by far the newer contender.

- **Benefits:**
  - VP9 being open source means the potential for long term community development.
Contender #2: VP8/9

**Drawbacks:**

- VP8 isn’t completely open source, as while the code is available, Google releases the new versions and isn’t always quick to do so.
- It is also not as prevalent as H.264.

**Shots Fired:**

- Google announced the removal of native support for H.264 in Chrome in 2011 (however, neither actual support was removed, nor the change to this plan announced). Furthermore, YouTube, a Google property, has (April 19, 2011) transcoded their videos into the WebM file format, applying on all new videos and to the most popular current videos (on the 30% that make up 99% of views).
- That said, in the interest of peace, love, and happiness for all, YouTube has stated that it will continue to support the H.264 codec.
Contender #2: VP8/9

- Moreover, a slew of big players formed and launched the WebM Community cross-license initiative to help further support the WebM Project.

- Matt Frost, senior business product manager for the WebM Project, announced on the WebM project blog that Google, Matroska and the Xiph.Org Foundation will make the components of WebM openly available on royalty-free terms.
Contender #3: Theora

- **Relevant Companies:** Xiph.org
- **Container:** OGG

**Overview:**
- Starting off with notably poor quality video, it has since caught up.
- It was, at one point, front-runner for HTML5, but this was changed, with the heavy hitters behind the other options a likely reason.
- It’s also truly open source, unlike Google’s semi-stealy and ultimately more controlled alternative.

**Benefits:**
- It is supported by Mozilla Firefox, Google Chrome, and Safari.
Contender #3: Theora

- **Drawbacks:**
  - Without the backing of large companies, it will be difficult for Theora to be any kind of standard going forward.

- **Shots Fired:**
  - Being a bit too left behind, and not backed enough to really fire shots.
  - If anything, OGG has had to defend itself against criticism of its performance.
Video Formats and Browser Support

- This leaves us with a fragmented situation (http://en.wikipedia.org/wiki/HTML5_video).

<table>
<thead>
<tr>
<th>Browser</th>
<th>Operating System</th>
<th>Theora</th>
<th>H.264 (MP4)</th>
<th>HEVC (MP4)</th>
<th>VP8 (WebM)</th>
<th>VP9 (WebM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Chrome</td>
<td>Unix-like, Android, OS X, iOS, and Windows</td>
<td></td>
<td>Since 3.0[53][54]</td>
<td>Since 3.0[54][54]</td>
<td>No[45]</td>
<td>Since 6.0[56][57]</td>
</tr>
<tr>
<td>Internet Explorer</td>
<td>Windows</td>
<td></td>
<td>Via OpenCodecs</td>
<td>Since 9.0[60]</td>
<td>No[45]</td>
<td>Via OpenCodecs</td>
</tr>
<tr>
<td></td>
<td>Windows Phone</td>
<td>No</td>
<td>Since 9.0[61]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Windows RT</td>
<td>No</td>
<td>Since 10.0[61]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Konqueror</td>
<td>Unix-like and Windows</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mozilla Firefox</td>
<td>Windows 7+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Windows Vista</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Windows XP and N editions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Linux</td>
<td>Since 3.5[69]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Android</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OS X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Me</td>
<td>Firefox OS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opera Mobile</td>
<td>Android, iOS, Symbian, and Windows Mobile</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opera</td>
<td>OS X, Windows, Linux</td>
<td>Since 13.0</td>
<td>Since 11.50</td>
<td>No[84]</td>
<td>Since 15.0</td>
<td>Since 16.0</td>
</tr>
<tr>
<td>Safari</td>
<td>iOS</td>
<td>No</td>
<td></td>
<td></td>
<td>No[45]</td>
<td>Via a plugin[81]</td>
</tr>
<tr>
<td>Web</td>
<td>Linux and BSD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Confused?
So what do we do?

- The upshot is...

  - There is no single combination of containers and codecs that works in all HTML5 browsers.

  - This is not likely to change in the near future.

  - To make your video watchable across all of these devices and platforms, you’re going to need to encode your video more than once.
So what so we do?

For maximum compatibility, here’s what your video workflow will look like:

1. Make one version that uses Theora video and Vorbis audio in an Ogg container.

2. Make another version that uses WebM (VP9 + Opus).

3. Make another version that uses H.264 Baseline video and AAC low-complexity audio in an MP4 container.

4. Then, you tie these separate versions of the file to a single `<video>` element, and fall back to a Flash-based video player.
Multiple `<source>` elements

- Instead of using the single `src` attribute, you nest separate `<source>` elements for each encoding with appropriate type attributes inside the `<video>` element and let the browser download the format that it can display.

- In this case we do not provide a `src` attribute in the media element itself:
HTML Video - Media Types

<table>
<thead>
<tr>
<th>File Format</th>
<th>Media Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP4</td>
<td>video/mp4</td>
</tr>
<tr>
<td>WebM</td>
<td>video/webm</td>
</tr>
<tr>
<td>Ogg</td>
<td>video/ogg</td>
</tr>
</tbody>
</table>
Multiple `<source>` elements

1. `<video controls>`
2. `<source src="movie.ogv" type="video/ogg" />`
3. `<source src="movie.mp4" type="video/mp4" />`
4. `<source src="movie.webm" type="video/webm" />`
5. `<p>Your browser doesn’t support video. Please download the video in `<a href="movie.ogv">Ogg</a>` or `<a href="movie.mp4">mp4</a>` or `<a href="movie.webm">webM</a> format.</p>`
6. `</video>`
Multiple `<source>` elements

1. `<video controls>`
   `<!--Tell the browser that a video is to be inserted and to give it default controls.-->

2. `<source src="movie.ogv" type="video/ogg" />
   `<!--
   ▶ Offers an Ogg Theora video and uses the `type` attribute to tell the browser what kind of container format is used (by giving the file’s MIME type).
   ▶ If you miss out on the type attribute, the browser downloads a small bit of each file before it figures out that it is unsupported, which wastes bandwidth and could delay the media playing.
   -->`
Multiple `<source>` elements

3. `<source src="movie.mp4" type="video/mp4" />
   <!--
   Offers an H.264 video.
   -->`

4. `<source src="movie.webm" type="video/webm" />
   <!--
   We are also offering a webM video.
   -->`
Multiple `<source>` elements

5. `<p>Your browser doesn’t support video.</p>`

1. Please download the video in `<a href=movie.ogv> Ogg</a>` or `<a href=movie.mp4>mp4</a>` or `<a href=movie.webm>webM</a>` format.</p>

`<!--`

- Inside the `<video>` element is our fallback message, including links to formats for browsers that can natively deal with neither video type but which is probably on top of an operating system that can deal with one of the formats, so the user can download the file and watch it in a media player outside the browser.

`-->`
Multiple `<source>` elements

6. `</video>`

```html
<!--
  So that’s native HTML5 video for all users of modern browsers.
-->
```

- What about users of legacy browsers—including Internet Explorer 8 and older?...
Video for Legacy Browsers

- Older browsers can’t play native video or audio.

- But if you’re prepared to rely on plugins, you can ensure that users of older browsers can still experience your content in a way that is no worse than they currently get.

- Remember that the contents of the `<video>` element can contain markup, like the text and links in the previous example?

- Because the MP4 file type can also be played by the Flash player plugin, you can use the MP4 movie in combination as a fallback for Internet Explorer 8 and older versions of other browsers.
Video for Legacy Browsers

- The code for this is as hideous as you’d expect for a transitional hack, but it works everywhere a Flash Player is installed—which is almost everywhere.

- You can see this nifty technique in an article called “Video for Everybody!” by its inventor, Kroc Camen:

  - http://camendesign.com/code/video_for_everybody
Video for Legacy Browsers

Alternatively, you could host the fallback content on a video hosting site and embed a link to that between the tags of a video element:

```html
<video controls>
<source src="movie.ogv" type="video/ogg">
<source src="movie.mp4" type="video/mp4">
<embed src="http://www.youtube.com/v/cmtcc94Tv3A&hl=en_GB&fs=1&rel=0" type="application/x-shockwave-flash" allowscriptaccess="always" allowfullscreen="true" width="425" height="344">
</video>
```
HTML5 Video

- HTML5 hasn’t “killed” all plugins overnight. There are use-cases for plugins not covered by the new spec.

- HTML5 video has no provisions for streaming the video files. Users have become accustomed to being able to seek to a specific part of a video. This is something that Flash-based video players excel at, because of the amount of effort Adobe has put into Flash as a video delivery platform.

- To seek with HTML5 video, the file must be downloaded completely on browsers. This may change in time.
What HTML5 Multimedia isn’t Good For

- Copy protection is one area not dealt with by HTML5—unsurprisingly, given that it’s a standard based on openness.

- So people who need DRM are probably not going to want to use HTML5 video or audio, as they will be as easy to download to a hard drive as an `<img>` is now.

- Consider also the massive amount of content out there that will require plugins to render it for a long time to come.
What HTML5 Multimedia isn’t Good For

- Finally, and most importantly, the process of encoding videos is costly and time-consuming. The need to encode in multiple formats makes HTML5 video much less attractive.

- For that reason, you see many sites supplying video in the patent-encumbered H.264 format so that it can be played on the iPod and iPad, using a combination of the HTML5 video tag and Flash.

- These issues aren’t going to derail HTML5, but they are things to be aware of.
CS7026 – HTML5

Audio
Audio on the Web

- Until HTML5, there wasn’t a standard for playing audio on a web page.

- Today, most audio is played through a plugin (like flash). However, not all browsers have the same plugins.

- HTML5 specifies a standard way to include audio, with the audio element.

- The audio element can play sound files, or an audio stream.
Audio Formats

Simply:

- For audio: encode original, full quality audio as MP3 and Ogg.

- Most browsers and clients will accept MP3; all others will take the Ogg format.
How It Works

- To play an audio file in HTML5, this is all you need:
  ```html
  <audio src="song.ogg" controls="controls">
  </audio>
  ```

- The control attribute is for adding play, pause, and volume controls.
How It Works

- Insert content between the `<audio>` and `</audio>` tags for browsers that do not support the audio element:

  `<audio src="song.ogg" controls="controls"> Your browser does not support the audio element. </audio>`
How It Works

- To make the audio work across the browsers use the multiple source elements.

- Source elements can link to different audio files. The browser will use the first recognized format:

  <audio controls="controls">
    <source src="song.ogg" type="audio/ogg" />
    <source src="song.mp3" type="audio/mpeg" />
  </audio>

Your browser does not support the audio element.

</audio>
## All `<audio>` Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoplay</td>
<td>autoplay</td>
<td>Specifies that the audio will start playing as soon as it is ready.</td>
</tr>
<tr>
<td>controls</td>
<td>controls</td>
<td>Specifies that controls will be displayed, such as a play button.</td>
</tr>
<tr>
<td>loop</td>
<td>loop</td>
<td>Specifies that the audio will start playing again (looping) when it reaches the end</td>
</tr>
<tr>
<td>preload</td>
<td>preload</td>
<td>Specifies that the audio will be loaded at page load, and ready to run. Ignored if autoplay is present.</td>
</tr>
<tr>
<td>src</td>
<td>url</td>
<td>Specifies the URL of the audio to play</td>
</tr>
</tbody>
</table>
Audio Exercise…

- FabSoundsCo is developing a site to showcase some royalty-free audio loops for use in screencasts, and it would like to see a demo page mocked up of a single loop collection.

- When you’re done, you’ll have a list of the audio loops, and a visitor will be able to quickly audition each one.

- You don’t have to worry about finding audio loops for this project, because the client’s sound engineer has already provided us with the samples we’ll need in both MP3 and OGG formats.
Audio Exercise…

- Don’t forget to provide a fallback…
Video Exercise...

- FabVideosCo wants to showcase its new series of training videos on its website, and it wants the videos to be viewable on as many devices as possible, especially on the iPad.

- As a trial, we’ve been provided two videos in the “Photoshop Tips” series* that we’ll use to build a prototype.

- Thankfully, we’ve been given the video files in H.264, Theora, and VP8 format, so we can focus on creating the page.
Video Exercise

- Insert them into a HTML5 page using the multiple `<source>` elements.

- Save to your www folder.

- Test across different browsers. Try viewing on a mobile device.