title	date	updated	tags	published	by
Choosing a Path Forward for IIIF Audio and Video	2017-02-02 21:00 -0500	2017-02-02 21:00 -0500	iiif, video, audio, api	false	Jason Ronallo

IIIF is working to bring AV resources into IIIF. I have been thinking about how to bring to AV resources the same benefits we have enjoyed for the IIIF Image and Presentation APIs. The initial intention of IIIF, especially with the IIIF Image API, was to meet a few different goals to fill gaps in what the web already provided for images. I want to consider how video works on the web and what gaps still need to be filled for audio and video.

(readmore)

This is a draft and as I consider the issues more I will make changes to better reflect my current thinking.

Images

When images were specified for the web the image formats were not chosen, created, or modified with the intention of displaying and exploring huge multi-gigabit images. Yet we have high resolution images that users would find useful to have in all their detail. So the first goal was to improve performance of delivering high resolution images. The optimization that would work for viewing large high resolution images was already available; it was just done in multiple different ways. Tiling large images is the work around that has been developed to improve the performance of accessing large high resolution images. If image formats and/or the web had already provided a solution for this challenge, tiling would not have been necessary. When IIIF was being developed there were already tiling image servers available. The need remained to create standardized access to the tiles to aid in interoperability. IIIF accomplished standardizing the performance optimization of tiling image servers. The same functionality that enables tiling can also be used to get regions of an image and manipulate them for other purposes. In order to improve performance smaller derivatives can be delivered for use as thumbnails on a search results page.

The other goal for the IIIF Image API was to improve the sharing of image resources across institutions. The situation before was both too disjointed for consumers of images and too complex for those implementing image servers. IIIF smoothed the path for both. Before IIIF there was not just one way of creating and delivering tiles, and so trying to retrieve image tiles from multiple different institutions could require making requests to multiple different kinds of APIs. IIIF solves this issue by providing access to technical information about an image through an info.json document. That information can then be used in a standardized way to extract regions from an image and manipulate them. The information document delivers the technical properties necessary for a client to create the URLs needed to request the given sizes of whole images and tiles from parts of an image. Having this standard accepted by many image servers has meant that institutions can have their choice of image servers based on local needs and infrastructure while continuing to interoperate for various image viewers.

So it seems as if the main challenges the IIIF Image API were trying to solve were about performance and sharing. The web platform had not already provided solutions so they needed to be developed. IIIF standardized the pre-existing performance optimization pattern of image tiling. Through publishing information about available images in a standardized way it also improved the ability to share images across institutions.

> What other general challenges were trying to be solved with the IIIF Image API?

Video and Audio

The challenges of performance and sharing are the ones I will take up below with regards to AV resources. How does audio and video currently work on the web? What are the gaps that still need to be filled? Are there performance problems that need to be solved? Are there challenges to sharing audio and video that could be addressed?

AV Performance

The web did not gain native support for audio and video until later in its history. For a long time the primary ways to deliver audio and video on the web used Flash. By the time video and audio did become native to the web many of the performance considerations of media formats already had standard solutions. Video formats have such advanced lossy compression that they can sometimes even be smaller than an image of the same content. (Here is an example of a screenshot as a lossless PNG being much larger than a video of the same page including additional content.) Tweaks to the frequency of full frames in the stream and the bitrate for the video and audio can further help improve performance. A lot of thought has been put into creating AV formats with an eye towards improving file size while maintaining quality. Video publishers also have multiple options for how they encode AV in order to strike the right balance for their content between compression and quality.

Progressive Download

In addition video and audio formats are designed to allow for progressive download. The whole media file does not need to be downloaded before part of the media can begin playing. Only the beginning of the media file needs to be downloaded before a client can get the necessary metadata to begin playing the video in small chunks. The client can also quickly seek into the media to play from any arbitrary point in time without downloading the portions of the video that have come before or after. Segments of the media can be buffered to allow for smooth playback. Requests for these chunks of media can be done with a regular HTTP web server like Apache or Nginx using byte range requests. The web server just needs minimal configuration to allow for byte range requests that can deliver just the partial chunk of bytes within the requested range. Progressive download means that a media file does not have to be pre-segmented--it can remain a single whole file--and yet it can behave as if it has been segmented in advance. Progressive download effectively solves many of the issues with the performance of the delivery of very long media files that might be quite large in size. Media files are already structured in such a way that this functionality of progressive download is available for the web. Progressive download is a performance optimization similar to image tiling. Since these media formats and HTTP already effectively solve the issue of quick playback of media without downloading the whole media file, there is no need for IIIF to look for further optimizations for these media types. Additionally there is no need for special media servers to get the benefits of the improved performance.

Quality of Service

While progressive download solves many of the issues with delivery of AV on the web based on how the media files are constructed, it is a partial solution. The internet does not provide assurances on quality of service. A mobile device at the edge of the range of a tower will have more latency in requesting each chunk of content than a wired connection at a large research university. Even over the same stable network the time it takes for a segment of media to be returned can fluctuate based on network conditions. This variability can lead to media playback stuttering or stalling while retrieving the next segment or taking too much time to buffer enough content to achieve smooth playback. There are a couple different solutions to this that have been developed.

With only progressive download at your disposal one solution is to allow the user to manually select a rendition to play back. The same media content is delivered as several separate files at different resolutions and/or bitrates. Lower resolutions and bitrates mean that the segments will be smaller in size and faster to deliver. The media player is given a list of these different renditions with labels and then provides a control for the user to choose the version they prefer. The user can then select whether they want to watch a repeatedly stalling, but high quality, video or would rather watch a lower resolution video playing back smoothly. Many sites implement this pattern as a relatively simple way to take into account that different users will have different network qualities. The problem I have found with this solution for progressive download video is that I am often not the best judge of network conditions. I have to fiddle with the setting until I get it right if I ever do. I can set it higher than it can play back smoothly or select a much lower quality than what my current network could actually handle. I have also found sites that set my initial quality level much lower than my network connection can handle which results in a lesser experience until I make the change to a higher resolution version. That it takes me doing the switching is annoying and distracting from the content.

Adaptive Bitrate Formats

To improve the quality of the experience while providing the highest quality rendition of the media content that the network can handle, other delivery mechanisms were developed. I will cover in general terms a couple I am familiar with, that have the largest market share, and that were designed for delivery over HTTP. For these formats the client measures network conditions and delivers the highest quality version that will lead to smooth playback. The client monitors how long it takes to download each segment as well as the duration of the current buffer. (Sometimes the client also measures the size of the video player in order to select an appropriate resolution rendition.) The client can then adapt on the fly to network conditions to play the video back smoothly without user intervention. This is why it is called "smooth streaming" in some products.

For adaptive bitrate formats like HLS and MPEG-DASH what gets initially delivered is a manifest of the available renditions/adaptations of the media. These manifests contain pointers for where (which URL) to find the media. These could be whole media files for byte range requests, media file segments as separate files, or even in the case of HLS a further manifest/playlist file for each rendition/stream. While the media is often referred to in a manifest with relative URLs, it is possible to serve the manifest from one server and the media files (or further manifests) from a different server like a CDN.

How the media files are encoded is important for the success of this approach. For these formats the different representations can be pre-segmented into the same duration lengths for each segment across all representations. In a similar way they can also be carefully generated single files that have full frames relatively close together within a file and all have these full frames synchronized between all the renditions of the media. For instance all segments could be six seconds with an iframe every 2 seconds. This careful alignment of segments allows for switching between representations without having glitchy moments where the video stalls, without the video replaying or skipping ahead a moment, and with the audio staying synchronized with the video.

It is also possible in the case of video to have one or more audio streams separate from the video streams. Separate audio streams aligned with the video representations will have small download sizes for each segment which can allow a client to decide to continue to play the audio smoothly even if the video is temporarily stalled or reduced in quality. One use case for this audio stream performance optimization is the delivery of alternative language tracks as separate audio streams. The video and audio bitrates can be controlled by the client independently.

In order for adaptive formats like this to work all of the representations need to have the next required segment ready on the server in case the client decides to switch up or down bitrates. While cultural heritage use cases that IIIF considers do not include live streaming broadcasts, the number of representations that all need to be encoded and available at the same time effects the "live edge"--how close to real-time the stream can get. If segments are available in only one high bitrate rendition then the client may not be able to keep up with a live broadcast. If all the segments are not available for immediate delivery then it can lead to playback issues.

The manifests for adaptive bitrate formats also include other helpful technical information about the media. (For HLS the manifest is called a master playlist and for MPEG-DASH a Media Presentation Description.) Included in these manifests can be the duration of the media, the maximum/minimum height and width of the representations, the mimetype and codecs (including MP4 level) of the video and audio, the framerate or sampling rate, and lots more. Most importantly for quality of experience switching, each representation includes a number for its bandwidth. There are cases where content providers will deliver two video representations with the same height and width and different bitrates to switch between. In these cases it is a better experience for the user to maintain the resolution and switch down a bandwidth than to switch both resolution and bandwidth. The number of representations--the ladder of different bandwidth encodes--can be quite extensive for advanced cases like Netflix over-the-top (OTT aka internet) content delivery. These adaptive bitrate solutions are meant to scale for high demand use cases. The manifests can even include information about sidecar or segmented subtitles and closed captions. (One issue with adaptive formats is that they may not play back across all devices, so many implementations will still provide progressive download versions as a fallback.) Manifests for adaptive formats include the kind of technical information that is useful for clients.

Because there are existing standards for the adaptive bitrate pattern that have broad industry and client support, there is no need to attempt to recreate these formats.

AV Performance Solved

All except the most advanced video on demand challenges have current solutions through ubiquitous video formats and adaptive bitrate streaming. As new formats like VP9 increase in adoption the situation for performance will improve even further. These formats have bitrate savings through more advanced encoding that greatly reduces file sizes while maintaining quality. This will mean that adaptive bitrate formats are likely to require fewer renditions than are typically published currently. Note though that in some cases smaller file sizes and faster decoding comes at the expense of much slower encoding when trying to keep a good quality level.

There is no need for the cultural heritage community to try to solve performance challenges when the expert AV community and industry has developed advanced solutions.

Parameterized URLs and Performance

One of the proposals for providing a IIIF AV API alongside the Image API involves mirroring the existing Image API by providing parameters for segmenting and transforming of media. I will call this the "parameterized approach." One way of representing this approach is this URL:

```
http://server/prefix/identifier/timeRegion/spaceRegion/timeSize/spaceSize/rotation/quality.format
```

You can see more about this type of proposal here and here. The parameters after the identifier and before the quality would all be used to transform the media.

For the Image API the parameterized approach for retrieving tiles and other derivatives of an image works as an effective performance optimization for delivery. In the case of AV having these parameters does not improve performance. It is already possible to seek into progressive download and adaptive bitrate formats. There is not the same need to tile or zoom into a video as there is for a high definition image. A good consumer monitor will show you as full a resolution as you can get out of most video.

And these parameters do not actually solve the most pressing media delivery performance problems. The parameterized approach probably is not optimizing for bitrate which is one of the most important settings to improve performance. Having a bitrate parameter within a URL would be difficult to implement well. Bitrate could significantly increase the size of the media or increase visible artifacts in the video or audio beyond usability. Would the audio and video bitrates be controlled separately in the parameterized approach? Bitrate is a crucially important parameter for performance and not one I think you would put into the hands of consumers. It will be especially difficult as bitrate optimization for video on demand is slow and getting more complicated. In order to optimize variable bitrate encoding 2-pass encoding is used and slower encoding settings can further improve quality. With new formats with better performance for delivery, bitrate is reduced for the same quality while encoding is much slower. Advanced encoding pipelines have been developed that perform metrics on perceptual difference so that each video or even section of a video can be encoded at the lowest bitrate that still maintains the desired quality level. Bitrate is where performance gains can be made.

The only functionality proposed for IIIF AV that I have seen that might be helped by the parameterized approach is download of a time segment of the video. This is specific to download of just that time segment. Is this use case big enough to be seriously considered for the amount of complexity it adds? Why is download of a time segment crucial? Why would most cases not be met with just skipping to that section to play? Or can the need be met with downloading the whole video in those cases where download is really necessary? If needed any kind of time segment download use case could live as a separate non-IIIF service. Then it would not have any expectation of being real-time. I doubt most would really see the need to implement a download service like this if the need can be met some other way. In those cases where real-time performance to a user does not matter those video manipulations could be done outside of IIIF. For any workflow that needs to use just a portion of a video the manipulation could be a pre-processing step. In any case if there is really the desire for a video transformation service it does not have to be the IIIF AV API but could be a separate service for those who need it.

Most of the performance challenges with AV have already been solved via progressive download formats and adaptive bitrate streaming. Remaining challenges not fully solved with progressive download and adaptive bitrate formats include live video, server-side control of quality of service adaptations, and greater compression in new codecs. None of these are the types of performance issues the cultural heritage sector ought to try to take on, and the parameterized approach does not contribute solutions to these remaining issues. Beyond these rather advanced issues, performance is a solved problem that has had a lot of eyes on it.

If the parameterized approach is not meant to help with optimizing performance what problem is it trying to solve? The community would be better off steering clear of this trap of trying to optimize for performance and instead focus on problems that still need to be solved. The parameterized approach is sticking with a performance optimization pattern that does not add anything for AV. It has a detrimental fixation on the bitstream that does not work for AV especially as adaptive bitrate segmented formats are concerned. It appears motivated by some kind of purity of approach rather than taking into account the unique attributes of AV and solving these particular challenges well.

AV Sharing

The other challenge a standard can help with is sharing of AV across institutions. If the parameterized approach does not solve a performance problem, then what about sharing? If we want to optimize for sharing and have the greatest number of institutions sharing their AV resources, then there is still no clear benefit for the parameterized approach. What about this parameterized approach aids in sharing? It seems to optimize for performance, which as we have seen above is not needed, at the expense of the real need to improve and simplify sharing. There are many unique challenges for sharing video across institutions on the web that ought to be considered before settling on a solution.

One of the big barriers to sharing is the complexity of AV. Compared to delivery of still images video is much more complicated. I have talked to a few institutions that have digitized video and have none of it online yet because of the hurdles. Some of the complication is technical, and because of this institutions are quicker to use easily available systems just to get something done. As a result many fewer institutions will have as much control over AV as they have over images. It will be much more difficult to gain that kind of control. For instance with some media servers they may not have a lot of control over how the video is served or the URL for a media file.

Video is expensive. Even large libraries often make choices about technology and hosting for video based on campus providing the storage for it. Organizations should be able to make the choices that work for their budget while still being able to share in as much as they desire and is possible.

One argument made is that many institutions had images they were delivering in a variety of formats before the IIIF Image API, so asking for similar changes to how AV is delivered should not be a barrier to pursuing a particular technical direction. The difficulty of institutions in dealing with AV can not be minimized in this way as any kind of change will be much greater and asking much more. The complexity and costs of AV and the choices that forces should be taken into consideration.

An important question to ask is who you want to help by standardizing an API for sharing? Is it only for the well-resourced institutions who self-host video and have the technical expertise? If it is required that resources live in a particular location and only certain formats be used it will lead to fewer institutions gaining the sharing benefits of the API because of the significant barriers to entry. If the desire is to enable wide sharing of AV resources across as many institutions as possible, then that ought to lead to a different consideration of the issues of complexity and cost.

One issue that has plagued HTML5 video from the beginning is the inability of the browser vendors to agree on formats and codecs. Early on open formats like WebM with VP8 were not adopted by some browsers in favor of MP4 with H.264. It became common practice out of necessity to encode each video in a variety of formats in order to reach a broad audience. Each source would be listed on the page (on a source element within a video element) and the browser picks which it can play. HTML5 media was standardized to use a pattern to accommodate the situation where it was not possible to deliver a single format that could be played across all browsers. It is only recently that MP4 with H.264 has been able to be played across all current browsers. Only after Cisco open sourced its licensed version of H.264 was this possible. Note while the licensing situation for playback has been improved there are still patent/licensing issues which mean that some institutions still will not create or deliver any MP4 with H.264.

But now even as H.264 can be played across all current browsers, there are still changes coming that mean a variety of formats will be present in the wild. New codecs like VP9 that provide much better compression are taking off and have been adopted by most, but not all, modern browsers. The advantages of VP9 are that it reduces file size such that storage and bandwidth costs can be reduced significantly. Encoding time is increased while performance is improved. And still other new, open formats like AV1 using the latest technologies are being developed. Even audio is seeing some change as Firefox and Chrome are implementing FLAC which will make it an option to use a lossless codec for audio delivery.

As the landscape for codecs continues to change the decision on which formats to provide should be given to each institution. Some will want to continue to use a familiar H.264 encoding pipeline. Others will want to take advantage of the cost savings of new formats and migrate. There ought to be allowance for each institution to pick which formats best meet their needs. Since sources in HTML5 media can be listed in order of preference, in as much as is possible a standard ought to support the ability of a client to respect the preferences of the institution for these reasons. So if WebM VP9 is the first source and the browser can play that format it should play it even if an MP4 H.264 is available which it can also play. The institution may make decisions around the quality to provide for each format to optimize for their particular content and intended uses.

Then there is the choice to implement adaptive bitrate streaming. Again institutions could decide to implement these formats for a variety of reasons. Delivering the appropriate adaptation for the situation has benefits beyond just enabling smooth playback. By delivering only the segment size a client can use based on network conditions and sometimes player size, the segments can be much smaller lowering bandwidth costs. The institution can make a decision depending on their implementation and use patterns whether their costs are more with storage or bandwidth and use the formats that work best for them. It can also be a courtesy to mobile users to deliver smaller segment sizes. Then there are delivery platforms where an adaptive bitrate format is required. Apple requires iOS applications to deliver HLS for any video over ten minutes long. Any of these types of considerations might nudge an AV provider to use ABR formats. They add complexity but also come with attractive performance benefits.

Any solution for an API for AV media should not try to pick winners among codecs or formats. The choice should be left to the institution while still allowing them to share the media in these formats with other institutions. It should allow for sharing AV in whatever formats an institution chooses. An approach which restricts which codecs and formats can be shared does harm and closes off important considerations for publishers. Asking them to deliver too many duplicate versions will also mean forcing certain costs. Will this variety of codecs allow for complete interoperability from every institution to every other institution and user? Probably not, but the tendency will be for institutions to do what is needed to support a broad range of browsers while optimizing for their particular needs.

Guidelines and evolving best practices can also be part of any community built around the API. A standard for AV sharing should not shut off options while allowing for a community of practice to develop.

Simple API

If an institution is able to deliver any of their video on the web, then that is an accomplishment. What could be provided to allow them to most easily share their video with other institutions? One simple approach would be for them to create a URL where they can publish information about the video. Some JSON with just enough technical information could map to the properties an HTML5 video player uses. Since it is still the case that many institutions are publishing multiple versions of each video in order to cover the variety of new and old browsers and mobile devices, it could include a list of these different video sources in a preferred order. Preference could be given to an adaptive bitrate format or newer, more efficient codec like VP9 with an MP4 fallback further down the list. Since each video source listed includes a URL to the media, the media file(s) could live anywhere. Hybrid delivery mechanisms are even possible where different servers are used for different formats or the media are hosted on different domains or use CDNs.

This ability to just list a URL to the media would mean that as institutions move to cloud hosting or migrate to a new video server, they only need to change a little bit of information in a JSON file. This greatly simplifies the kind of technical infrastructure that is needed to support the basics of video sharing. The JSON information file could be a static file. No need even for redirects for the video files since they can live wherever and change location over time.

Here is an example of what part of a typical response might look like where a WebM and an MP4 are published:

```
{
  "sources": [
    {
      "id": "https://iiif-staging02.lib.ncsu.edu/iiifv/pets/pets-720x480.webm"
      "format": "webm",
      "height": 480,
      "width": 720,
      "size": "3360808",
      "duration": "35.627000",
      "type": "video/webm; codecs=\"vp8,vorbis\"",
    },
    {
      "id": "https://iiif-staging02.lib.ncsu.edu/iiifv/pets/pets-720x480.mp4"
      "format": "mp4",
      "frames": "1067",
      "height": 480,
      "width": 720,
      "size": "2924836",
      "duration": "35.627000",
      "type": "video/mp4; codecs=\"avc1.42E01E,mp4a.40.2\"",
    }
  ]
}
```

You can see an example of this "sources" approach here.

An approach that simply lists the available sources an institution makes available for delivery ought to be easier for more institutions over other options for sharing AV. It would allow them to effectively share the whole range of the types of audio and video they already have no matter what technologies they are currently using. In the simplest cases there would be no need for even redirects. If you are optimizing for widest possible sharing from the most institutions, then an approach along these lines ought to be considered.

Straight to AV in the Presentation API?

One interesting option has been proposed for IIIF to move forward with supporting AV resources. This approach is presented in What are Audio and Video Content APIs?. The mechanism is to list out media sources similar to the above approach but on a canvas within a Presentation API manifest. The pattern appears clear for how to provide a list of resources in a manifest in this way. It would not require a specific AV API that tries to optimize for the wrong concerns. The approach still has some issues that may impede sharing.

Requiring an institution to go straight to implementing the Presentation API means that nothing is provided to share AV resources outside of a manifest or a canvas that can be referenced separate from a Presentation manifest. Not every case of sharing and reuse requires the complexity of a Presentation manifest in order to just play back a video. There are many use cases that do not need a sequence with a canvas with media with an annotation with a body with a list of items--a whole highly nested structure, just to get to the AV sources needed to play back some media. This breaks the pattern from the Image API where it is easy and common to view an image without implementing Presentation at all. Only providing access to AV through a Presentation manifest lacks simplicity which would allow an institution to level up over time. What is the path for an institution to level up over time and incrementally adopt IIIF standards? Even if a canvas could be used as the AV API as a simplification over a manifest, requiring a dereferenceable canvas would further complicate what it takes to implement IIIF. Even some institutions that have implemented IIIF and see the value of a dereferenceable canvas have not gotten that far yet in their implementations.

One of the benefits I have found with the Image API is the ability to view images without needing to have the resource described and published to the public. This allows me to check on the health of images, do cache warming to optimize delivery, and use the resources in other pre-publication workflows. I have only implemented manifests and canvases within my public interface once a resource has been published, so would effectively be forced to publish the resource prematurely or otherwise change the workflow. I am guessing that others have also implemented manifests in such a way that is tied to their public interfaces.

Coupling of media access with a manifest has some other smaller implications. Requiring a manifest or canvas leads to unnecessary boilerplate when an institution does not have the information yet and still needs access to the resources to prepare the resource for publication. For instance a manifest and a canvas MUST have a label. Should they use "Unlabeled" in cases where this information is not available yet?

In my own case sharing with the world is often the happy result rather than the initial intention of implementing something. For instance there is value in an API that supports different kinds of internal sharing. Easy internal sharing enables us to do new things with our resources more easily regardless of whether the API is shared publicly. That internal sharing ought to be recognized as an important motivator for adopting IIIF and other standards. IIIF thus far has enabled us to more quickly develop new applications and functionality that reuse special collections image resources. Not every internal use will need or want the features found in a manifest, but just need to get the audio or video sources to play them.

If there is no IIIF AV API that optimizes for the sharing of a range of different AV formats and instead relies on manifests or canvases, then there is still a gap that could be filled. For at least local use I would want some kind of AV API in order to get the technical information I would need to embed in a manifest or canvas. This seems like it could be a common desire to decouple technical information about video resources from the fuller information needed for a manifest including attributes like labels needed for presentation with context to the public. Coupling AV access too tightly to Presentation does not help to solve the desire to decouple these technical aspects. It is a reasonable choice to consider this technical information a separate concern. And if I am already going through the work to create such an internal AV API, I would like to be able to make this API available to share my AV resources outside of a manifest or canvas.

Then there is also the issue of AV players. In the case of images many pan zoom image viewers were modified to work with the Image API. One of the attractions to delivery images via IIIF or adopting a IIIF image server is that there is choice in viewers. Is the expectation that any AV players would need to read in a Presentation manifest or canvas in order to support IIIF and play media? The complexity of the manifest and canvas documents may hinder adoption IIIF in media players. These are rather complicated documents that take some time to understand. A simpler API than Presentation may have a better chance to be more widely adopted for players and easier to maintain. We only have the choice of a couple featureful client side applications for presenting manifests (UniversalViewer and Mirador), but we already have many basic viewers for the Image API. Even though not all of those basic viewers are used within the likes of UniversalViewer and Mirador, the simpler viewers have still been of value for other use cases. For instance a simple image viewer can be used in a metadata management interface where UniversalViewer features like the metadata panel and download buttons are unnecessary or distracting. Would the burden of maintaining plugins and shims for various AV players to understand a manifest or canvas rest with the relatively small IIIF community rather than with the larger group of maintainers of AV players? Certainly having choice is part of the benefit of having the Image API supported in many different image viewers. Would IIIF still have the goal of being supported by a wide range of video players? This ability to have broad support within some of the foundational pieces like media players allows for better experimentation on top of it.

My own implementation of the Image API has shown how having a choice of viewers can be of great benefit. When I was implementing the IIIF APIs I wanted to improve the viewing experience for users by using a more powerful viewer. I chose UniversalViewer even though it did not have a very good mobile experience at the time. We did not want to give up the decent mobile experience we had previously developed. Moving to only using UV would have meant giving up on mobile use. So that we could still have a good mobile interface while UV was in the middle of improving its mobile view, we also implemented a Leaflet-based viewer alongside UV. We toggled each viewer on/off with CSS media queries. This level of interoperability at this lower level in the viewer allowed us to take advantage of multiple viewers while providing a better experience for our users. You can read more about this in Simple Interoperability Wins with IIIF. As AV players are uneven in their support of different features this kind of ability to swap out one player for another, say based on video source type, browser version, or other features, may be particularly useful. We have also seen new tools for tasks like cropping grow up around the Image API and it would be good to have a similar situation for AV players.

So while listing out sources within a manifest or canvas would allow for institutions with heterogeneous formats to share their distributed AV content, the lack of an API that covers these formats results in some complication, open questions, and less utility.

Conclusion

IIIF ought to focus on solving the right challenges for audio and video. There is no sense in trying to solve the performance challenges of AV delivery. That work has been well done already by the larger AV community and industry. The parameterized approach to an AV API does not bring significant delivery performance gains though that is the only conceivable benefit to the approach. The parameterized approach does not sufficiently help make it easier for smaller institutions to share their video. It does not provide any help at all to institutions that are trying to use current best practices like adaptive bitrate formats.

Instead IIIF should focus on achieving ubiquitous sharing of media across many types of institutions. The focus on solving the challenges with sharing media and the complexity and costs with delivering AV resources leads to meeting institutions more where they are at. A simple approach to an AV API that lists out the sources would more readily solve the challenges institutions will face with sharing.

Optimizing for sharing leads to different conclusions than optimizing for performance.