Answer from helioviewer.org :

Roman,

In nominal operations, we have one AIA image in each of the higher temperature channels (94, 131, 171, 193, 211, 304, 335) once every 36 seconds.  We have AIA 4500 images once every hour.  We have AIA 1600 and 1700 images once every 36 seconds, very roughly speaking.  HMI continuum and magnetogram images are about once every 45 seconds each.

We have 57 TB of AIA images and 8 TB of HMI images.

Let me know if you would like any more information.

Thanks,

Jack

<https://helgeklein.com/blog/2017/12/browser-video-codecs-formats-hardware-acceleration/>

<https://opensensorhub.org/2016/09/16/video-decoding-in-osh-js-toolkit/>

Wikipedia bitrate:

* 400 kbit/s [YouTube](https://en.wikipedia.org/wiki/YouTube) 240p videos (using [H.264](https://en.wikipedia.org/wiki/MPEG-4_AVC))[[20]](https://en.wikipedia.org/wiki/Bit_rate#cite_note-youtube-20)
* 750 kbit/s [YouTube](https://en.wikipedia.org/wiki/YouTube) 360p videos (using [H.264](https://en.wikipedia.org/wiki/MPEG-4_AVC))[[20]](https://en.wikipedia.org/wiki/Bit_rate#cite_note-youtube-20)
* 128 kbit/s for 128p !

<https://stackoverflow.com/questions/10328401/html5-how-to-stream-large-mp4-files>

* Having so many small files worries me. Maybe large videos with ~~byte range requests~~ ?

<https://flussonic.com/post/2016-12-13-html5-streaming>

* ~~MPEG-Dash ?~~

MOVIE\_LENGTH 64 with 30fps. Movie is too short…

<https://bitmovin.com/mp4box-dash-content-generation-x264/>

ceil(1920 / 256) x ceil(1200 / 256) = max 40 streaming tiles

YouTube has ~2MB chunk size for 360p video playback.

Conclusions:

* Media Source Extensions with custom streaming logic (zoom & playback speed). Hardware accelerated decoding there.
* Videos:
  + Bitrate ~~1024kbit/s for 128px~~ 🡪 ~240kbit/s for 256p
  + Longer durations: 2 secs 🡪 6secs).
  + Optional: Many black videos, 4kb vs 200+kb. Longer videos based on content?
  + Grayscale
* Don’t use any adaptive bitrate logic. Just upscale low res.

# Was wurde gemacht

Von Helioviewer.org wurde mithilfe eines Jupyter Notebook zwei sets von Bilder heruntergeladen (256 Bilder und 2048 Bilder) und jeweils als jp2, Bitmap und jpg file abgespeichert.

In einem zweiten Schritt wurden die Bitmap-Files mit einer C#-Anwendung vorskaliert. Diese skalierten Bilder wurden verwendet um mithilfe von ffmpeg in einen Video-tree mit dem h.264codec zu konvertieren.

Anschliessend wurde eine kleine Website implementiert, welche sechs dieser Videos gleichzeitig decodiert und mithilfe von webgl abspielt. Dabei wurde der Broadwaydecoder (<https://github.com/mbebenita/Broadway>) verwendet. Die videos werden dabei im voraus decodiert und jedes Frame wird im Memory zwischengespeichert.

Der verbrauchte Speicherplatz ist im Dokument data.xlsx ersichtlich.

## Video-tree

128x128 grosse kacheln (jeweils für verschiedene Auflösungen)

64 Frames pro video.

Räumliche Auflösung: je ein tree für verschiedene Auflösungen (128, 256, 512, 1024, 2048 und 4096)

Zeitliche Auflösung: je ein video für jedes Bild, jedes zweite, jedes vierte, usw..

# Feststellungen

* Um die beiden 128x128 video-tree abzuspeichern ist ca. Acht mal so viel Speicherplatz nötig wie von den jp2-Dateien verbraucht wird. Dieser Faktor kann sich jedoch ändern, da nur mit einer kleinen Datenmenge getested wurde. (22h Sonnenbilder mit je ein Bild alle 36 Sekunden, entspricht 2048 Bildern).
* Das erstellen des video-trees dauert lange (22h laufzeit auf einem Laptop, um den 22h-tree herzustellen). Der Code kann vermutlich optimiert werden.
* Das Abspielen mehrerer Kacheln gleichzeitig mit webgl fällt nicht auf und ist technisch möglich.

# Offene Punkte

## Andere Auflösungen testen

Die Anwendung wurde nur mit 128x128 pixel grossen Segmente getestet. Eventuell ist eine andere oder eine variable Auflösung besser.

## Andere Videolänge testen

Es wurden nur videos mit je 64 Frames erstellt.

## H264 Codec licensing abklären

Die lizensierung muss genau abgeklärt werden, da h.264 patentiert ist und der verwendete Broadway.js decoder nicht lizensiert ist. Alternativ kann ein lizensierter decoder verwendet werden.

Siehe github issue für Beispielfall:  
<https://github.com/mbebenita/Broadway/issues/124>

## Andere Bitrate testen

Getestet wurde mit einer hohen Bitrate. Dadurch bestehen die videos mehrheitlich aus P-Frames und sind dementsprechend mehrheitlich gleich gross. (durchschnittlich 200KB/video, wobei die meisten ca. 250KB gross sind)

## Andere Codecs abklären

Es wurde nur der h.264 codec getestet

# ****Issues****

## Memory Leak

Die aktuelle Implementierung hat ein memory leak bezüglich GPU-Memory. Die Texturen werden in jeden Frame neu erstellt und nie mehr freigegeben. Dies führt dazu das die Anwendung nach einiger Zeit Abstürzt.

Vorschläge:

* Texturen pro frame erstellen und Referenz dazu behalten
* Eine Textur pro Segment erstellen und den Inhalt der Textur neuberechnen

## Lange Creation time

Den video-tree herzustellen nimmt viel Zeit in Anspruch und wurde nicht spezielll optimiert.

Ca. 21h um 2048 Bilder zu konvertieren und in 128x128 grosse videos umzuwandeln.

## Kein Internet Explorer support

Die Testanwendung läuft nicht auf IE11, da dieser keine javascript-klassen unterstützt. Microsoft Edge wird jedoch unterstützt.