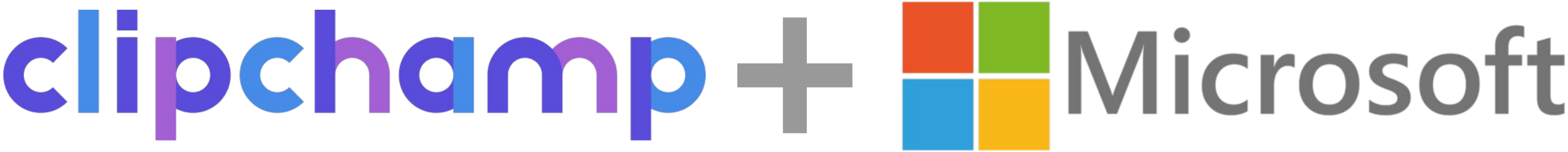
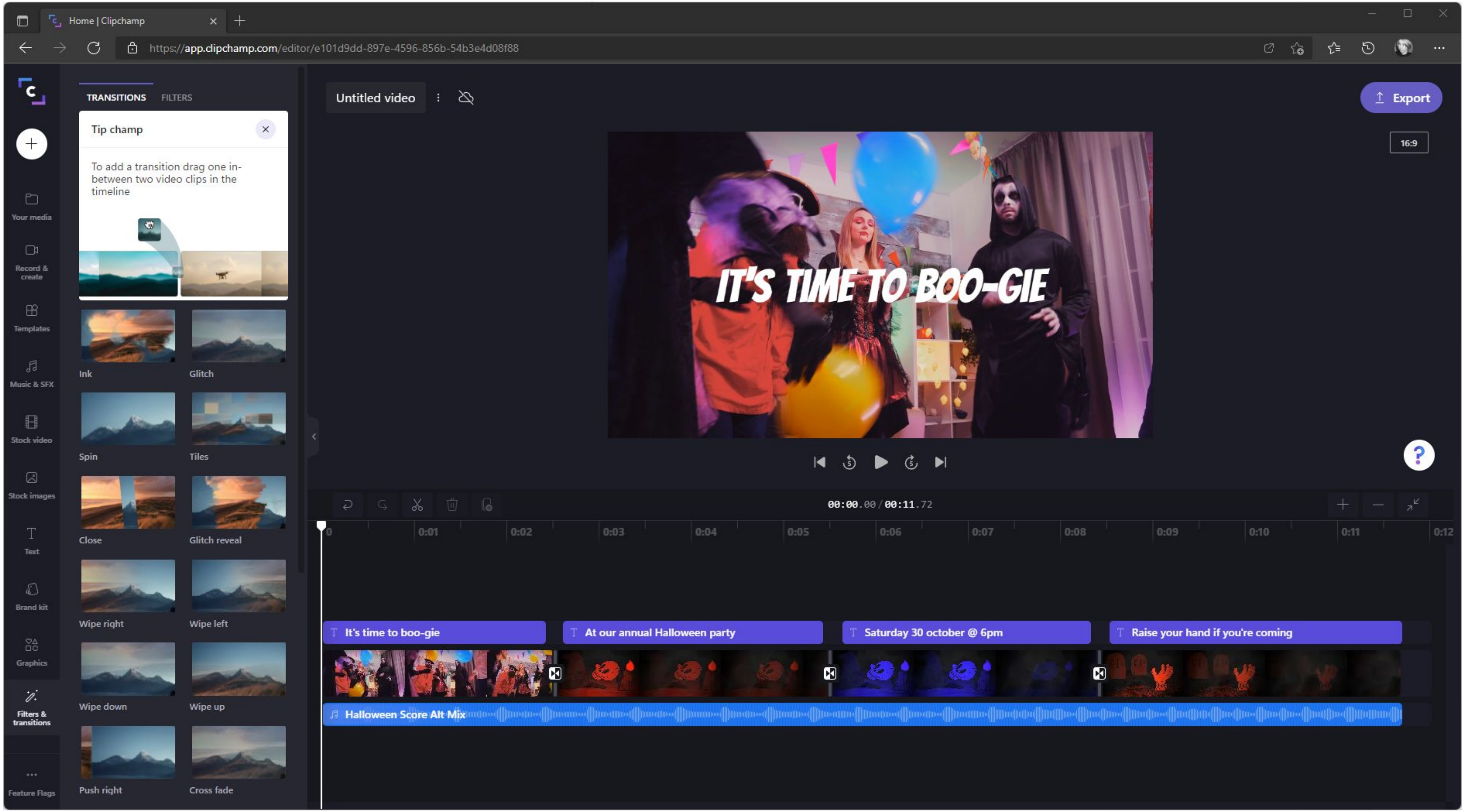
WebCodecs in an in-browser video editor

Sören Balko
October 2021


clipchamp
+
Microsoft

Our mission is to
empower anyone
to tell stories
worth sharing


Home | Clipchamp

https://app.clipchamp.com/editor/e101d9dd-897e-4596-856b-54b3e4d08f88

Untitled video

Tip champ

To add a transition drag one in-between two video clips in the timeline

00:00.00 / 00:11.72

It's time to boo-gie

At our annual Halloween party

Saturday 30 october @ 6pm

Raise your hand if you're coming

Halloween Score Alt Mix

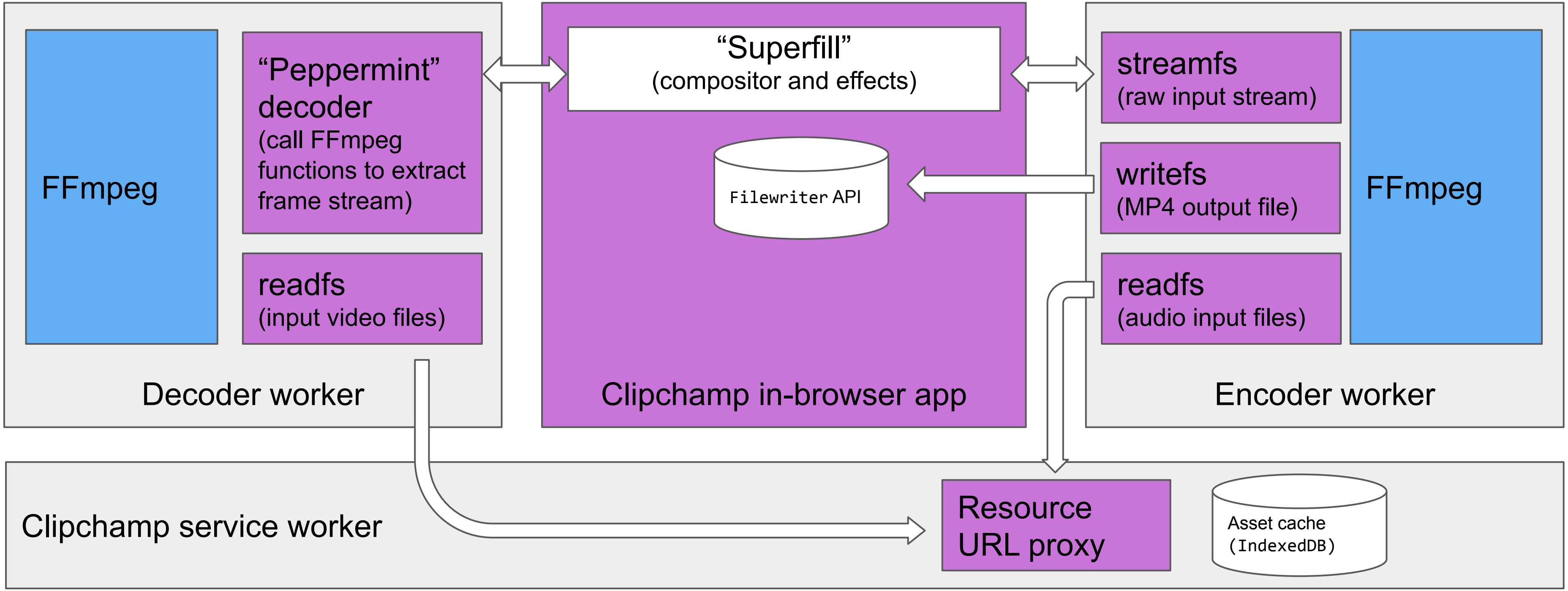
Only an in-browser platforms offers the convenience of a cloud service combined with the speed of a desktop application.

Clipchamp’s secret sauce

- Clipchamp’s full video production pipeline runs “in browser”
    - Great user experience (no upload of user media)
    - Near zero runtime cost
    - Perfect privacy for user media

But...

- The browser remains a challenging platform for fully featured in-browser apps:
    - Resource allocation (memory, storage)
    - Performance
    - Access to low-level hardware capabilities
    - Cross-browser woes, buggy browsers, etc.


FFmpeg
"Peppermint" decoder
(call FFmpeg functions to extract frame stream)
readfs
(input video files)
Decoder worker
"Superfill"
(compositor and effects)
Filewriter API
Clipchamp in-browser app
streamfs
(raw input stream)
writefs
(MP4 output file)
readfs
(audio input files)
FFmpeg
Encoder worker
Clipchamp service worker
Resource URL proxy
Asset cache
(IndexedDB)

Integrating WebCodecs API into Clipchamp's export pipeline

- Combine WebAssembly build of FFmpeg (for de/muxing, file I/O, software codec fallbacks, filters) with WebCodecs API
- Introduce new FFmpeg codec "stubs" for VP8, VP9, H.264, AV1
- Codec calls out to Javascript for WebCodecs interactions (to initialise/configure, push frames, pull encoded packets, close down)
- Generate `VideoEncoder` configuration from FFmpeg's internal data structures

Gotchas

- Had to create a "preflight" dry-run of VideoEncoder to generate codec extradata (eg. H.264 SPS/PPS NALUs) to satisfy FFmpeg's need to have that available during codec initialization.
- A WebAssembly build of FFmpeg is inherently synchronous, whereas the WebCodecs API is asynchronous - we had to break FFmpeg up into per-frame asynchronous calls.

Our wishlist for WebCodecs 2.0

- A big THANK YOU to the people who have pushed the WebCodecs standard!
- Where we are hoping to see further improvements:
  - Active encoder back pressure detection
  - Quality control “tuning knob” (other than bitrate)
  - Support for HDR, HEVC decoding
  - Synchronous flavor of WebCodecs API inside workers