Evolved Video Encoding with WebCodecs Breakout Session September 25, 2024

Erik Språng, Eugene Zemtsov

TPAC 2024
Anaheim CA, USA
hybrid meeting

23–27 SEPTEMBER 2024

W3C Code of Conduct

- This meeting operates under W3C Code of Ethics and Professional Conduct

- We're all passionate about improving WebRTC and the Web, but let's all keep the conversations cordial and professional

Safety Reminders

While attending TPAC, follow the health rules:

- Masks and daily testing are left to individual choice

Please be aware of and respect the personal boundaries of your fellow participants

For information about mask and test availability, what to do if you are not feeling well, and what to do if you test positive for COVID, see:

https://www.w3.org/2024/09/TPAC/health.html

Virtual Meeting Tips (Zoom)

- Both local and remote participants need to be on irc.w3.org channel #evolved-webcodecs.
- Use “+q” in irc to get into the speaker queue and “-q” to get out of the speaker queue.
- Please use headphones when speaking to avoid echo.
- Please wait for microphone access to be granted before speaking.
- Please state your full name before speaking.

Agenda

- Goals
  - Why reference frame control?
- Initial Proposal
  - Minimum viable change, but that is still useful.
- Querying Capabilities
  - How to reason about what the codecs can do.

As time permits:

- Spatial Scalability
  - Dealing with multiple encodings per temporal unit.
- Rate Control
  - Issues with CBR in layered modes.
- Other Useful Features
  - Speed control, content hint, segmentation…

Goals

"Be able to implement **any reference structure** using a **minimal set of tools** in a **codec agnostic** way."

- The encoder only needs to know *what* and *how* to encode
- Use as little abstractions, complexity and state as possible

Goals - The How

Implement explicit **reference frame control**.

- All modes in w3.org/TR/webrtc-svc/ are possible, as are many modes currently not in the spec (e.g. quality layers)
- Enables structures that can't be expressed by a simple "mode", for instance different flavours of LTR
- No need for "dependency descriptor"-style feedback

Define a toolset that maps to all major codec types (H26x, VPx, AVx)

Initial Proposal

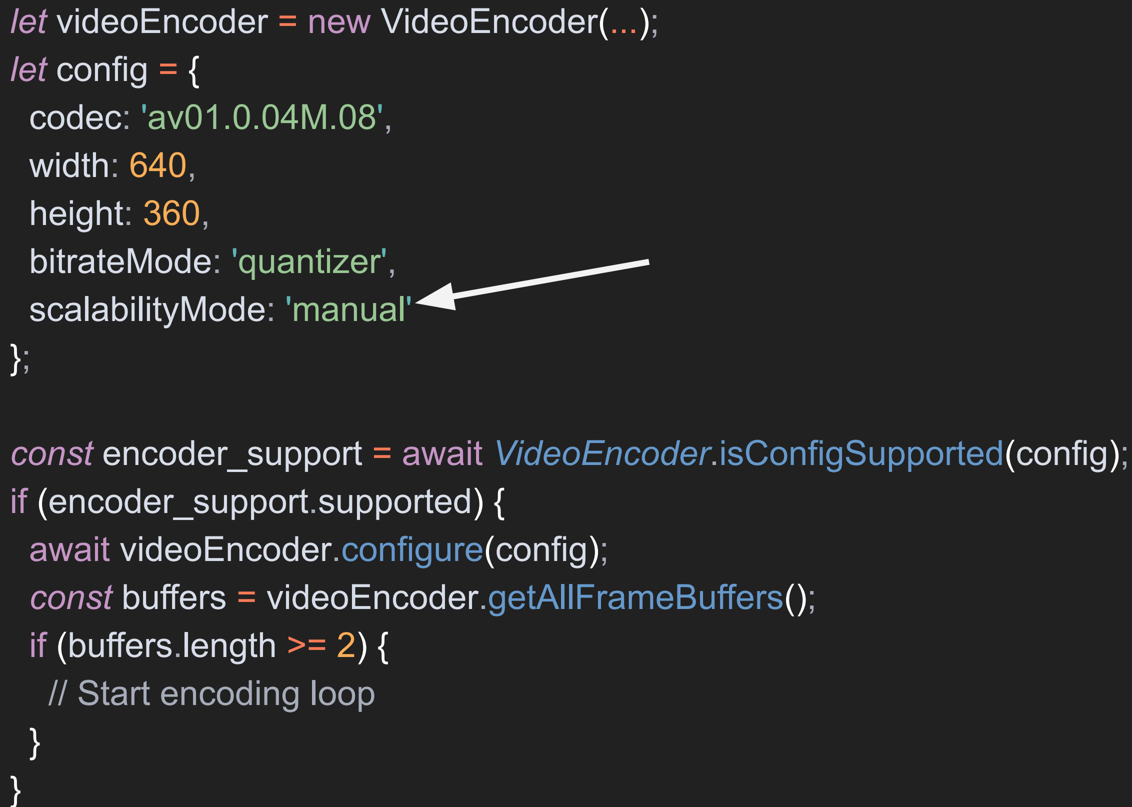
Available in Chromium behind the feature flag:
`WebCodecsVideoEncoderBuffers`

```
partial dictionary VideoEncoderConfig {
  DOMString scalabilityMode;  // New value "manual".
};

// This interface can't be constructed, it can only be obtained via calls to VideoEncoder.
interface VideoEncoderBuffer {
  DOMString id;
};

partial interface VideoEncoder {
  // Get a list of all buffers that can be used while encoding.
  sequence<VideoEncoderBuffer> getAllFrameBuffers();
};

partial dictionary VideoEncoderEncodeOptions {
  // Buffers that can be used for inter-frame prediction while encoding a given
  // frame. If this array is empty we basically ask for an intra-frame.
  sequence<VideoEncoderBuffer> referenceBuffers;

  // A buffer where the encoded frame should be saved after encoding.
  VideoEncoderBuffer updateBuffer;
};
```

W3C 1994 → 2024 TPAC 2024

Initial Proposal

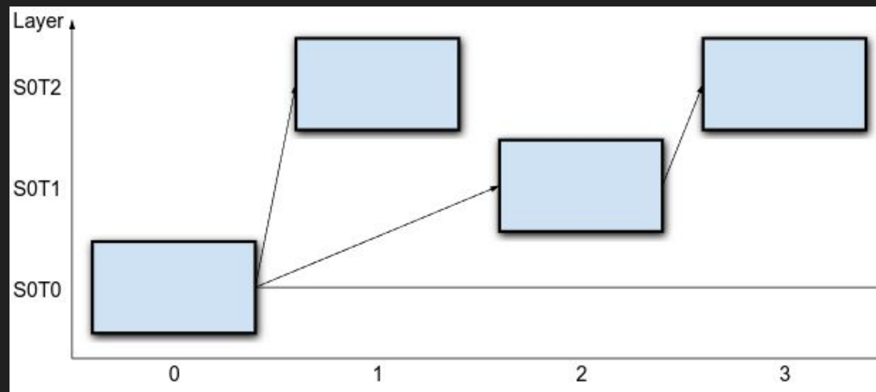
Example: Implement L1T3



Initial Proposal

Example: Implement L1T3

```
const pattern_index = frame_num % 4;
let encode_options = …;

switch (pattern_index) {
 case 0:
  encode_options.referenceBuffers = [buffers[0]];
  encode_options.updateBuffer = buffers[0];
  break;
 case 1:
  encode_options.referenceBuffers = [buffers[0]];
  break;
 case 2:
  encode_options.referenceBuffers = [buffers[0]];
  encode_options.updateBuffer = buffers[1];
  break;
 case 3:
  encode_options.referenceBuffers = [buffers[1], buffers[0]];
  break;
}

videoEncoder.encode(frame, encode_options);
```



Initial Proposal

Trade-offs and simplifications were made:

- Only CQP for now
  - Avoids bitrate-per-layer problems
- Number of buffers & references only limited by spec
  - Let's us postpone capability querying
- Only a single encoding per temporal unit
  - No SVC or simulcast support yet
- Only a single buffer can be updated
  - Multiple buffer updates only useful when pipelining & frame dropping is enabled
- Only model “long-term” style reference buffers
  - Limits to a common denominator: H26x can use both short-term and long-term references, VPx and AV1 support only long-term.
- Limited frame dropping support
  - Hard to reason about behavior with pipelining async encoders

Initial Proposal

Questions / feedback on initial proposal?

Please also provide feedback on the WebCodecs Github issue: #285

Querying Capabilities

Problem: *VideoEncoder*.isConfigSupported() does not scale well

Many encoder implementations have constraints, such as:

- Min / max resolution
- Resolution alignment (e.g. must be divisible by 16)
- QP ranges
- Number of buffers may be limited
- Number of references per frame may be limited
- Spatial layer count may be limited
- Reference frame scaling may be limited
- …

Suppose we add minimum number of buffers and references to config, then if supported == *false* is returned, it's nearly impossible to understand why. Trying all limits lead to combinatorial explosion.

Querying Capabilities

Illustrative example of `VideoEncoderCapabilities`:

```
dictionary VideoEncoderInputConstraints {
  unsigned long maxWidth;
  unsigned long maxHeight;
  unsigned long pixelAlignment;
};

dictionary VideoEncoderPredictionConstraints {
  unsigned long maxNumberOfSpatialLayers;
  unsigned long numberOfReferenceBuffers;
  unsigned long maxReferencesPerFrame;
};

dictionary VideoEncoderPerformanceCharacteristics {
  boolean isHardwareAccelerated;
};

dictionary VideoEncoderCapabilities {
  VideoEncoderInputConstraints inputConstraints;
  VideoEncoderPredictionConstraints predictionConstraints;
  VideoEncoderPerformanceCharacteristics performanceCharacteristics;
};
```

W3C 1994 → 2024 TPAC 2024

Querying Capabilities

Two possible ways of acquiring the capabilities:

- **Add enumeration of devices and capabilities**
  - **Makes it possible to choose between multiple implementations for the same codec type (e.g. discreet vs built-in GPU)**
  - **All necessary information from a single call**
- **Return capabilities as part of** VideoEncoderSupport **returned from** isConfigSupported**()**
  - **Only basic information in** VideoEncoderConfig
  - **If config is supported,** VideoEncoderCapabilities **tells you how to us it**
  - **Some iterations required, but no explosion**

Querying Capabilities

Illustrative example of encoder enumeration

```
// Uniquely identifies an encoder implementation.
interface VideoEncoderIdentifier {
  DOMString id;                        // Unique identifier for this entry.
  DOMString codecName;                 // e.g. "av01.0.04M.08"
  DOMString implementationName;  // e.g. "libaom"
};

partial interface VideoEncoder {
  // Get a map containing all the available video encoder implementations and their respective capabilities.
  static record<VideoEncoderIdentifier, VideoEncoderCapabilities> enumerateAvailableImplementations();
};

// When creating a VideoEncoder instance, pass the id of the implementation you want to use.
partial dictionary VideoEncoderInit {
  optional DOMString encoderId;  // Matches id of a VideoEncoderIdentifier.
};
```

Querying Capabilities

Illustrative example of capabilities as part of support

```
partial dictionary VideoEncoderSupport {
  VideoEncoderCapabilities? encoderCapabilities;  // Only set if supported == true.
};
```

To find the capabilities of available encodes:

```
const codecs = ["avc1.42001E", "vp8", "vp09.00.10.08", "av01.0.04M.08"];
const accelerations = ["prefer-hardware", "prefer-software"];
for (const codec of codecs) {
  for (const acceleration of accelerations) {
    // Test if supported, and check capabilities for this combo.
  }
}
```

Querying Capabilities

Fingerprinting Concerns

Exposing the capabilities may *seem* like it's adding a large fingerprinting surface. In fact, it does not.

- **All capabilities follow directly from the implementation in use.**
- The implementation can already be deduced
  - SW implementation defined by browser vendor + version
  - HW implementation defined by GPU
    - Can be found e.g. via WebGL or WebGPU
  - Once instantiated, an encoder can be easily identified based on the produced bitstream.

So the capabilities can in fact already be deduced, but we want an API that makes that information available in a useful and structured way.

Querying Capabilities

Questions / feedback on capability querying?

Spatial Scalability

How to support SVC / Simulcast

Two main features needed:

1. Ability to specify the **resolution** in the encode options
2. A **sequence** of encode options per input frame

The current interface:

```
partial interface VideoEncoder : EventTarget {
  undefined encode(VideoFrame frame, optional VideoEncoderEncodeOptions options = {});
};

dictionary VideoEncoderEncodeOptions {
  boolean keyFrame = false;
};
```

Spatial Scalability

How to support SVC / Simulcast

Proposed API:

```
partial interface VideoEncoder : EventTarget {
  undefined encode(VideoFrame frame, sequence<VideoEncoderEncodeOptions> options);
};

partial dictionary VideoEncoderEncodeOptions {
  boolean keyFrame = false;

  unsigned long spatialLayerId;
  unsigned long? width;
  unsigned long? height;
};

partial dictionary SvcOutputMetadata {
 unsigned long spatialLayerId;
};
```

Spatial Scalability

How to support SVC / Simulcast

With this API, any spatial scalability mode can be achieved. That includes all of the modes specified in https://www.w3.org/TR/webrtc-svc/ as well as quality-layers (inter-layer dependencies but no scale factors), keyframe-less resolution switching, switch-frames, and more.

It does come with some detailed requirements however:

- Each entry in options must use a distinct spatialLayerId.
- If a resolution isn't specified, the currently configured resolution of the encoder is assumed.
- The resolution specified must <= the configured resolution.
- The number of entries in the options list must not exceed the number of spatial layers supported by the encoder implementation.
- Used and updated buffers must be in accordance with support for reference frame scaling of the encoder implementation.
- If more than one spatial layer has keyFrame set to true, care must be taken to ensure that buffer management is done in accordance with the buffer space type of the encoder implementation.
- If inter-layer dependencies are used, the order of the encode options must be strict

Spatial Scalability

How to support SVC / Simulcast

Some special considerations:

- Not all encoders support reference frame scaling
  - …and even some that do only support e.g 2:1 and 1:2 scaling.
  - The user needs to understand what the encoder can do (through capability querying) and what content is in each buffer in order to create valid encode options.
- When dealing with *independent* spatial layers (simulcast or “SxTx”)
  - Some encoders have complete & separate sets of buffers for each spatial layer
  - Others use a “shared” buffer space. Users must make sure never to cross the buffers.
- When dealing with reference frame scaling, some encoders allow referencing any buffer while others (notably H26x) can only use reference frame scaling within the same temporal unit

Spatial Scalability

How to support SVC / Simulcast

Questions / feedback on spatial scalability?

Rate Control

Issues with CBR and layered encoding

Using external rate control, with encoder running in a simple CQP mode makes the API quite straightforward.

If we want to use CBR it becomes more tricky as the rate control needs to be layer aware:

- A target rate is needed for each spatial & temporal layer
- The each output frame (encode option) we need:
  - The spatial & temporal layer
  - The timestamp of the frame
  - The duration of the frame (might not be 1/fps)

Open question: do we need knobs to tune target/maximum delay values and other parameters for the rate controllers?

Rate Control

Issues with CBR and layered encoding

A common feature in rate control for RTC in frame dropping. That does however lead to difficulties reasoning about behavior. We suggest not supporting frame dropping when using reference frame control, instead:

- Have the application update the reference buffers such that the previous state is not lost.
  - I.e. never update the same buffer you reference
- This allows the app to either drop immediately or even re-encode with a high QP in a “semi-two-pass” fashion

Rate Control

Issues with CBR and layered encoding

Questions / feedback on rate control?

Other Useful Features

Speed control, content hint, segmentation…

Many other features could be implemented. Examples:

- Speed control (aka effort level, complexity)
  - Different implementations have different number of possible speed settings, with varying defaults.
  - Optimally configured on a per-frame basis
- Content type (e.g. camera vs screencast)
  - Already exists in the main config, but the type can change quickly within a single stream
- Segmentation
  - Useful for ROI coding, cyclic refresh, improved rate control
  - How do we expose it in a way that generalizes well?

W3C 1994 → 2024 TPAC 2024

Other Useful Features

Speed control, content hint, segmentation…

What's on *your* wish-list?