Video Rendering on Frontend and Backend
Lightricks is the creator of numerous award-winning photo and video editing apps. Our goal is to build fun and powerful tools that reinvent the way content is created all over the world.

Established 2013
HQ in Jerusalem, branches in Haifa & London
~400 Employees
12 Apps
We’re a Unicorn!
Boosted Web
The Playloop

- Decode frame
- Interpolate values: \[centerX = 0.5 + \text{time} \times 0.2\]
- Render frame
- Display
- Encode to video file

* Links to more info about value interpolation are in the last slide
Can we do that in the browser?

Decode:
  HTMLVideoElement
Interpolate:
  pure logic
Render:
  WebGL
Encode:
  Software codec
But...

- HTMLVideoElement
  - Can’t give source frame rate
  - Can’t seek to frame
  - Is geared for real-time playing
    - will skip frames during heavy workload

- We can code around it
  - Working against the intended usage
  - Will work on some browsers
Alternative

- Software emulation
- FFmpeg library
  - either back-and-forth in backend
  - or large import in frontend
  - Both are slow
Let’s split!

Browser:
- Decode frame with HTMLVideoElement
- Render frame
- Display

Server:
- Decode frame with FFmpeg
- Render frame
- Encode to video file
# Stacks

<table>
<thead>
<tr>
<th></th>
<th>Browser</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Javascript</td>
<td>C++</td>
</tr>
<tr>
<td>Decoding</td>
<td>HTMLVideoElement</td>
<td>FFmpeg</td>
</tr>
</tbody>
</table>
Rendering

OpenGL / WebGL as API

GL – Graphics Library

Same functionality, different structure

OpenGL – global state machine

WebGL – object oriented
## Stacks

<table>
<thead>
<tr>
<th></th>
<th>Browser</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Javascript</td>
<td>C++</td>
</tr>
<tr>
<td>Decoding</td>
<td>HTMLVideoElement</td>
<td>FFmpeg</td>
</tr>
<tr>
<td>Rendering</td>
<td>WebGL</td>
<td>OpenGL</td>
</tr>
</tbody>
</table>
# Stacks

<table>
<thead>
<tr>
<th></th>
<th>Browser</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Javascript</td>
<td>C++</td>
</tr>
<tr>
<td>Decoding</td>
<td>HTMLVideoElement</td>
<td>FFmpeg</td>
</tr>
<tr>
<td>Rendering</td>
<td>WebGL</td>
<td>OpenGL</td>
</tr>
<tr>
<td>Text Shaping</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
## Stacks

<table>
<thead>
<tr>
<th>Language</th>
<th>Browser</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Javascript</td>
<td></td>
<td>C++</td>
</tr>
<tr>
<td>Decoding</td>
<td>HTMLVideoElement</td>
<td>FFmpeg</td>
</tr>
<tr>
<td>Rendering</td>
<td>WebGL</td>
<td>OpenGL</td>
</tr>
<tr>
<td>Text Shaping</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>And more</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
WebAssembly

WASM in short

Available in browsers

Standard for secure, performant, cross-platform computing

Compiled from any language

* Links to more info about WebAssembly are in the last slide
Emscripten – compiler toolchain

Compile code to WASM

Create JavaScript bindings

Abstract interfaces to use JavaScript objects

OpenGL + Cpp = WASM + JavaScript + WebGL
## Stacks

<table>
<thead>
<tr>
<th></th>
<th>Browser</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Javascript &amp; C++</td>
<td>C++</td>
</tr>
<tr>
<td>Decoding</td>
<td>HTMLVideoElement</td>
<td>FFmpeg</td>
</tr>
<tr>
<td>Rendering</td>
<td></td>
<td>OpenGL</td>
</tr>
<tr>
<td>Text Shaping</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>And more</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Let’s combine code!

Browser:

Server:

Decode frame with HTMLVideoElement

Render frame

Display

Encode to video file
Story Time!
WebAssembly — What does security mean?

- WASM has separate memory
- Can’t access runtime memory
- WASI — WebAssembly System Interface

This means that everything must be copied.
Loading frames to textures

Has API for fast loading of textures from HTMLVideoElement

Textures represented by opaque JS objects

What's a HTMLVideoElement?

Textures represented by IDs
The Problem

renderer code uses OpenGL, frontend code uses WebGL

How to efficiently pass the frame data to WASM

Frame data — copy to WASM’s memory

Copy to WebGL texture — not available in C++
Restating the Problem

Data in JavaScript, needs to be consumed in WASM  Data copy is expensive  Abstraction layer separates between WASM and JavaScript  Can’t use the same objects on both sides
Misusing Emscripten for fun & profit

Emscripten calls WebGL with OpenGL code

Emscripten gives OpenGL texture identifiers to WebGL textures

**getNewId** — a function that inserts to array, and returns index as identifier

WebGL objects

Frame buffers

Textures
Solution

Create textures → Create texture identifier using Emscripten’s getNewId → Load frame data → Explicitly remove texture identifier

Use texture identifier
All’s well that ends well

We’ve reached 60 FPS  We wrote the engine once, used it twice  The bridging code is minimal
Is This Smart?

Relying on internal API which might change — Bad

Bridging abstraction gap — Necessary

Manually Managing texture lifetime isn’t great

Final score: 😞😞😞😞
Bottom line

We can write C++ code and run it everywhere

Emscripten bridges a lot of platform differences

Some hacking might still be required

Everything is more complicated when rendering :(
Links!

→ Check out the app
→ My talk about value interpolation
→ Lin Clark’s excellent intro to WebAssembly
→ My talk about the history of efficient computing on the web
→ Contact me