Fast Interactive Web Graphics with WebGPU

```
// Survey Presentation
const group1_members: [&str; 3] = [
    “Thomas Pinheiro de Souza”,
    “Stefan Schintler”,
    “Andreas Steinkellner”,
];
const presentation_date: &str = “2022-11-29”;
```

// Information Architecture and Web Usability, WS 2022

Web Graphics

Web Graphics - Introduction

Main categories:

- 2D Graphics
  - Canvas2D
  - SVG

→ 🐌 slow & simple 🌝

- 2D & 3D Graphics
  - WebGL
  - WebGPU

→ 🚀 fast execution & complicated 😵‍💫

WebGL *#[deprecated]*

- Wrapper (abstraction layer) over OpenGL[1].
- Developed by the Khronos Group[2] and Mozilla[3].
- Project start: 2009, based on OpenGL ES 3.0[4].
- Developed slowed down in 2017 (slow transition to WebGPU).

1: https://www.opengl.org/
2: https://www.khronos.org/
3: https://www.mozilla.org/
4: https://registry.khronos.org/webgl/specs/latest/2.0/

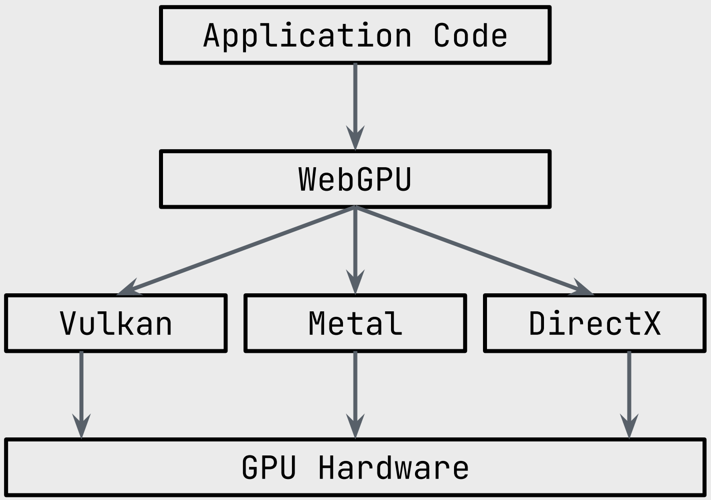
WebGL::Evolve() → WebGPU

- WebGPU is the natural successor to WebGL.
- Native support for compute shaders.
- Steady development.
- Standardization is already in polishing phase.
- Expected release of V1.0 in Q4, 2022.

Source: WebGL+WebGPU Meetup on 2022-10-04
Slides available at: https://www.khronos.org/assets/uploads/developers/presentations/WebGL__WebGPU_Updates_October_2022.pdf

WebGPU *#[experimental]*

- Abstraction layer that drives Vulkan[1], Metal[2] or DirectX 12[3].
- First prototype in 2017.
- Not yet enabled in major browsers:
  - Chrome: Origin Trial / feature flag in Beta & Canary
  - Firefox: Feature flag in Nightly
  - Safari: Experimental in Safari Technology Preview



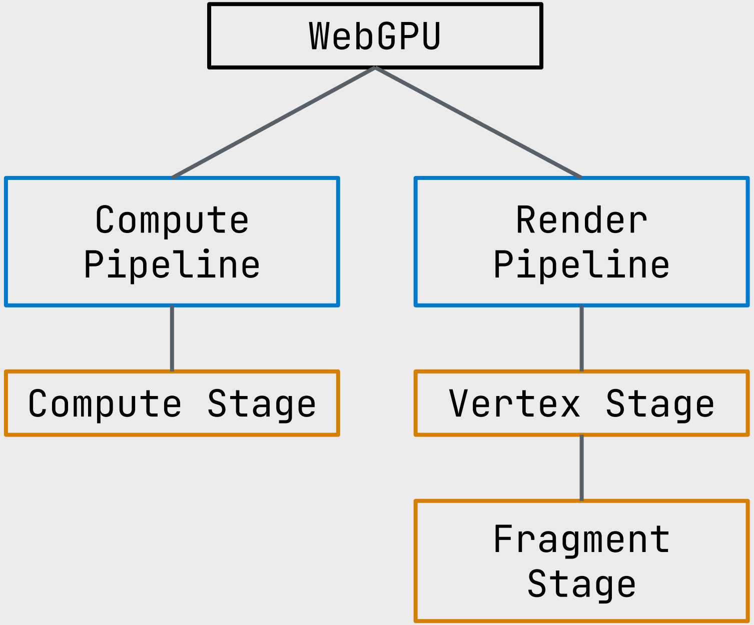
1: https://www.vulkan.org/
2: https://developer.apple.com/metal/
3: http://msdn.microsoft.com/de-de/directx/

🪜

Fundamentals

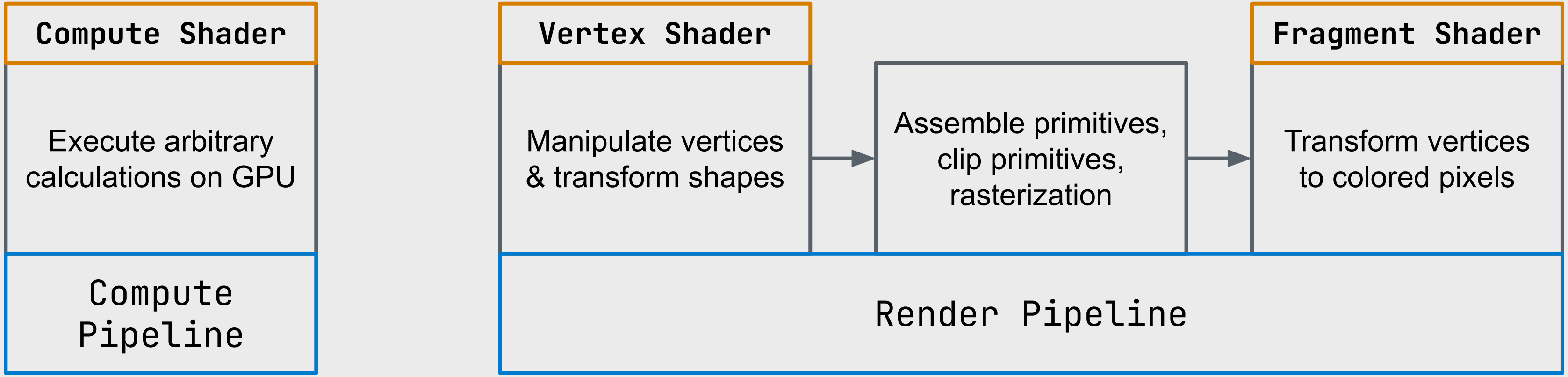
WebGPU::Overview()

- Supports different pipelines.
- Each **pipeline** consists of one or more stages.
- Each **stage** contains a shader (WGSL) and an entry point.



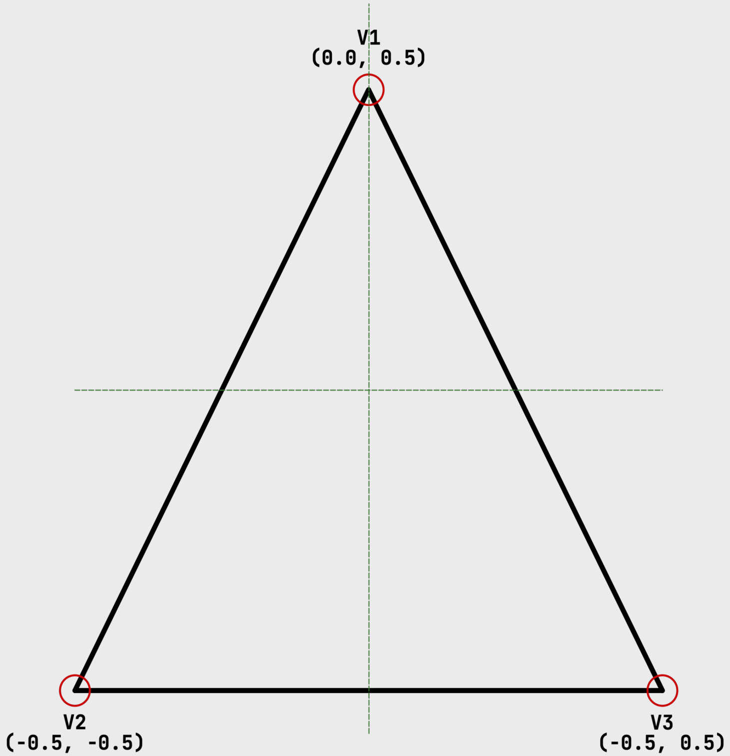
Shaders - Introduction

- **Programmable** stage of the render pipeline.
- Main types:



Vertex Shader

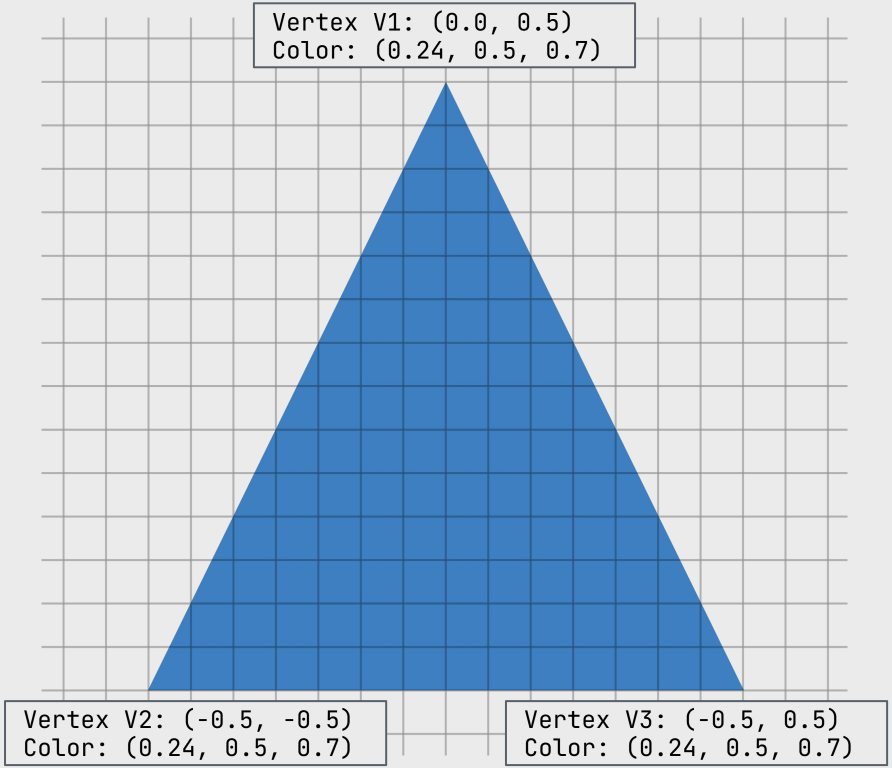
- **Goal**: Define all vertices of a desired primitive.
- Vertex is defined by a position and a set of attributes.
- Some output values of vertex shader are passed into fragment shader.



Vertex shader of a triangle - example.
Created by the authors

Fragment Shader

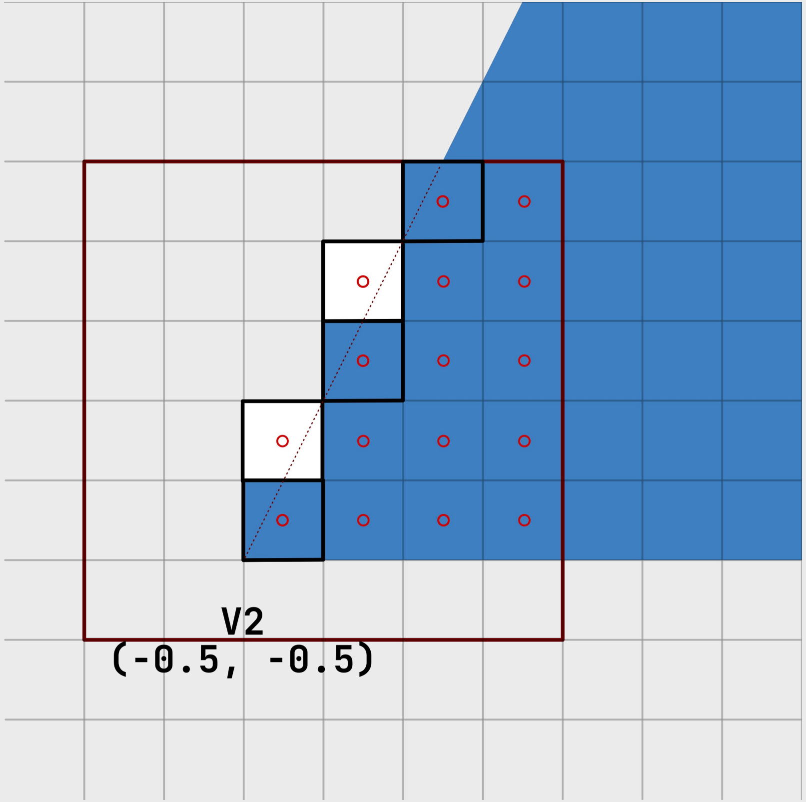
- **Goal**: Transform vertices to colored pixels on the screen.
- Each vertex has a defined color value, the rest is interpolated.
- **Output**: One fragment per rasterization point, runs in **parallel.**
- Challenges:
  - Rasterization (pixel grid)
  - Interpolation between vertices



Desired output for the specified vertices V1 - V3.
Created by the authors

Rasterization

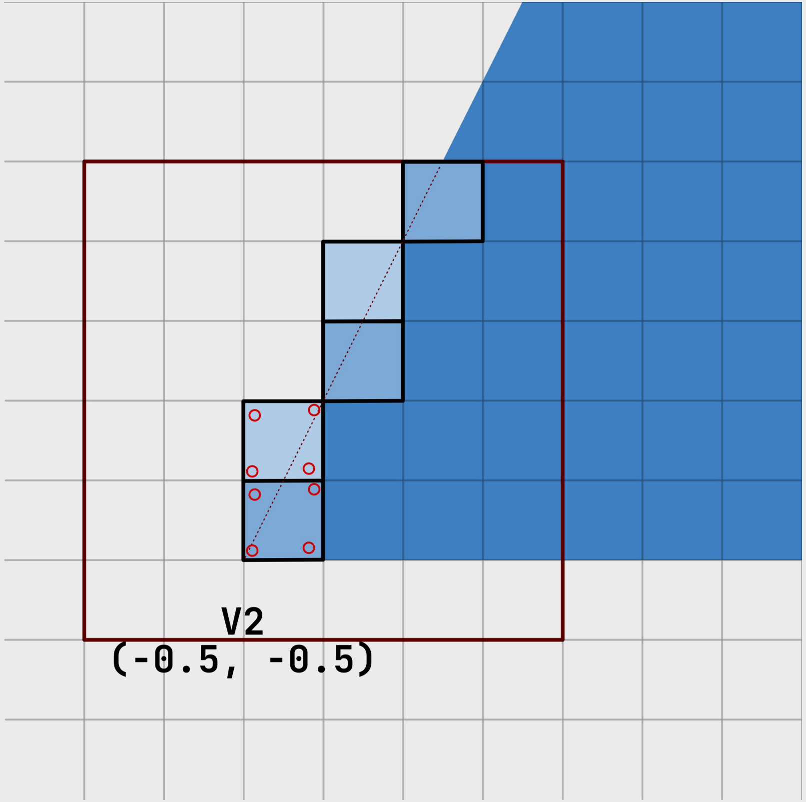
- **Goal**: Transform vertex information into rasterized points.
- Determines the set of pixels for a given primitive.
- Challenges
    - Culling: Evaluate front- and back-facing polygons, discard obstructed ones
    - Aliasing



Triangle rasterization with a single sample point in the center.
Created by the authors

Multisampling

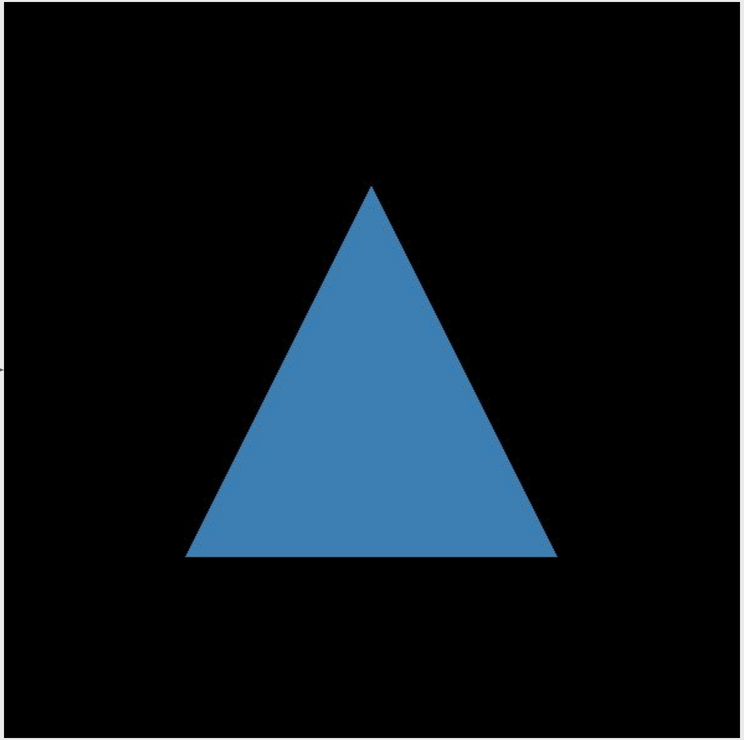
- Each pixel is evaluated on multiple points.
- Points are placed near the edge and create a **sample mask.**
- Interpolate final pixel value between all samples.



Triangle rasterization with four sample points for each pixel. (mutlisampling). Created by the authors

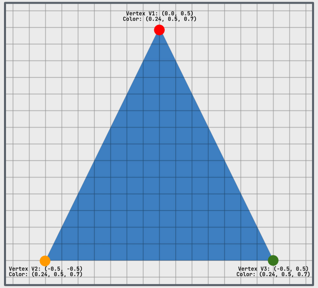
WebGPU::isComplicated = true

```
export function createBuffer(
  device: GPUDevice,
  data: number[],
  usage: GPUBufferUsageFlags
): GPUBuffer {
  // Align to 4 bytes
  let desc: GPUBufferDescriptor = {
    size: (data.length * 4 + 3) & ~3,
    usage,
    mappedAtCreation: true,
  };
  let buffer = device.createBuffer(desc);

  const writeArray = new Float32Array(buffer.getMappedRange());

  writeArray.set(data, 0);
  buffer.unmap();
  return buffer;
}

export type Coordinates = {
  x: number;
  y: number;
};

export type Color = [number, number, number];

You, vor 8 Minuten | 1 author (You)
export class Vertex {
  x: number; // 4 byte
  y: number; // 4 byte
  color: [number, number, number]; // 12 bytes

  static byteSize = 4 + 4 + 12;

  static webGPU_attributes: GPUVertexAttribute[] = [
    {
      // vec2<f32> position
      shaderLocation: 0,
      format: "float32x2",
      offset: 0,
    },
```

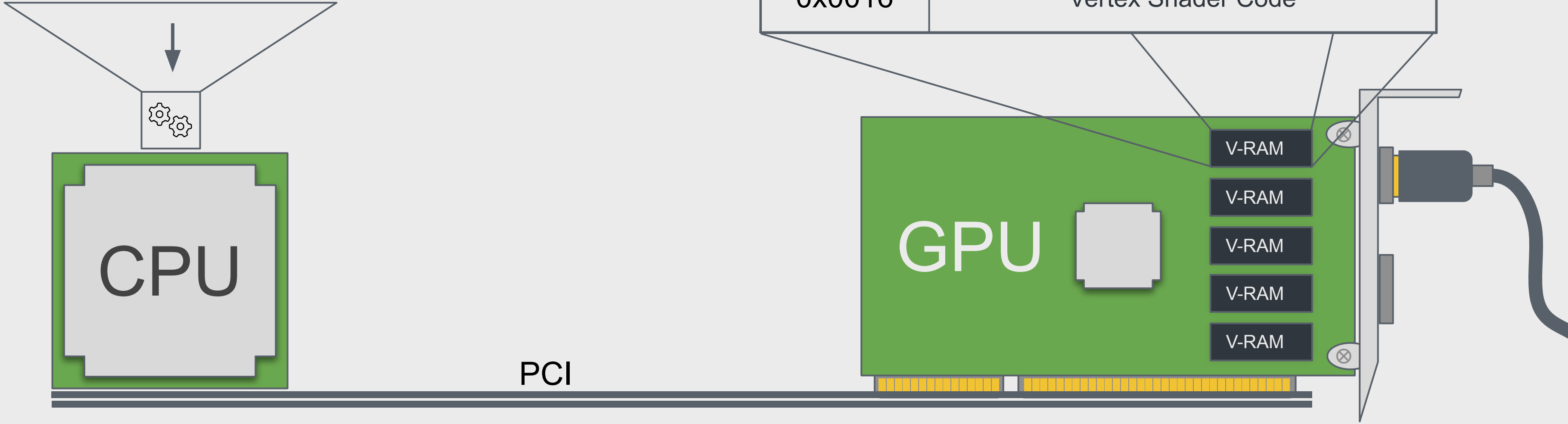


Minimal WebGPU example. Created by the authors.
Code available at: https://github.com/steschi/iaweb2022g1-survey

WebGPU::Explain()



Address	0	1	2	3	4	5
0x0000	Vertex 1 0	0,5 Position	0	0,24	0,5 Color	0,7
0x0006	1 Opacity	Vertex 2 -0,5	-0,5 Position	0	0,24	0,5
0x000B	0,7	1	Vertex 3 -0,5	0,5 Position	0	0,24
0x0010	0,5	0,7	1	Fragment Shader Code		
0x0016	Vertex Shader Code					



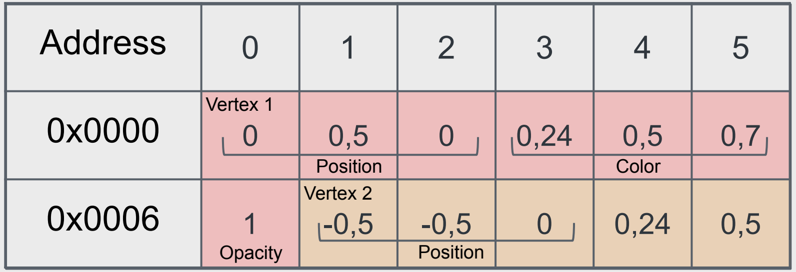
WebGPU::Step1()

```
export function createBuffer(
  device: GPUDevice, data: number[],
  usage: GPUBufferUsageFlags): GPUBuffer {

  let desc: GPUBufferDescriptor = {
     // Align to 4 bytes
     size: (data.length * 4 + 3) & ~3,
     usage, mappedAtCreation: true };
  let buffer = device.createBuffer(desc);

  const writeArray = new
           Float32Array(buffer.getMappedRange());
  writeArray.set(data, 0);
  buffer.unmap();
  return buffer;
}
```

```
const encodedData = encodeVertices(Triangle);
const gpu = navigator.gpu;
const canvas = document.getElementById("canvas");
const canvasContext = canvas.getContext("webgpu")!;
const adapter = await gpu.requestAdapter();
const device = await adapter!.requestDevice();
const buffer = createBuffer(
                            device,
                            encodedData,
                            GPUBufferUsage.VERTEX
                           );
```

WebGPU::Step2()

```
const pipeline = device.createRenderPipeline({
  layout: "auto",
  vertex: {
    module: device.createShaderModule({ code: shaderCode }),entryPoint: "vertex_main",
    buffers: [{ arrayStride: Vertex.byteSize, attributes: [
          { shaderLocation: 0, format: "float32x3", offset: 0 },          // vec3<f32> position
          { shaderLocation: 1, format: "float32x4", offset: 4 + 4 + 4 },  // vec4<f32> color
    ]}],
  },
  fragment: {
    module: device.createShaderModule({ code: shaderCode }),
    entryPoint: "fragment_main",
    targets: [{ format: gpu.getPreferredCanvasFormat() }],
  },
  primitive: { topology: "triangle-list" },
});
```



Address	0	1	2	3	4	5
0x0000	Vertex 1 0	0,5 Position	0	0,24	0,5 Color	0,7
0x0006	1 Opacity	Vertex 2 -0,5	-0,5 Position	0	0,24	0,5

WGSL::Explain() // WebGPU Shading Language

- Shading language for WebGPU.
- Syntactically a mix between Rust and C / C++.
- More explicit than GLSL (WebGL shader).
- Can easily be converted to other shader languages with Naga[1] or Tint[2].

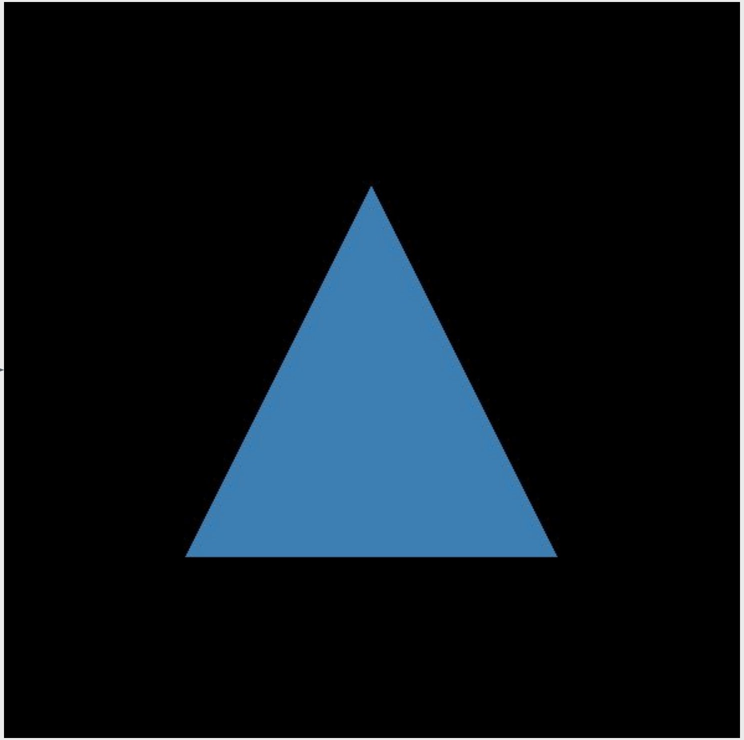
1: https://github.com/gfx-rs/naga
2: https://dawn.googlesource.com/tint

WebGPU::Step3() // WebGPU Shading Language

```
struct VSOutput { @builtin(position) position: vec4<f32>, @location(0) color: vec4<f32> };

@vertex
fn vertex_main(@location(0) position: vec3<f32>, @location(1) color: vec4<f32>) -> VSOutput {
    var output: VSOutput;
    output.position = vec4<f32>(position, 1.0);
    output.color = color;
    return output;
}

@fragment
fn fragment_main(@location(0) color: vec4<f32>) -> @location(0) vec4<f32> {
    return color;
}
```

WebGPU::isComplicated == true

```
export function createBuffer(
  device: GPUDevice,
  data: number[],
  usage: GPUBufferUsageFlags
): GPUBuffer {
  // Align to 4 bytes
  let desc: GPUBufferDescriptor = {
    size: (data.length * 4 + 3) & ~3,
    usage,
    mappedAtCreation: true,
  };
  let buffer = device.createBuffer(desc);

  const writeArray = new Float32Array(buffer.getMappedRange());

  writeArray.set(data, 0);
  buffer.unmap();
  return buffer;
}

export type Coordinates = {
  x: number;
  y: number;
};

export type Color = [number, number, number];

You, vor 8 Minuten | 1 author (You)
export class Vertex {
  x: number; // 4 byte
  y: number; // 4 byte
  color: [number, number, number]; // 12 bytes

  static byteSize = 4 + 4 + 12;

  static webGPU_attributes: GPUVertexAttribute[] = [
    {
      // vec2<f32> position
      shaderLocation: 0,
      format: "float32x2",
      offset: 0,
    },
```



Minimal WebGPU example. Created by the authors.
Code available at: https://github.com/steschi/iaweb2022g1-survey

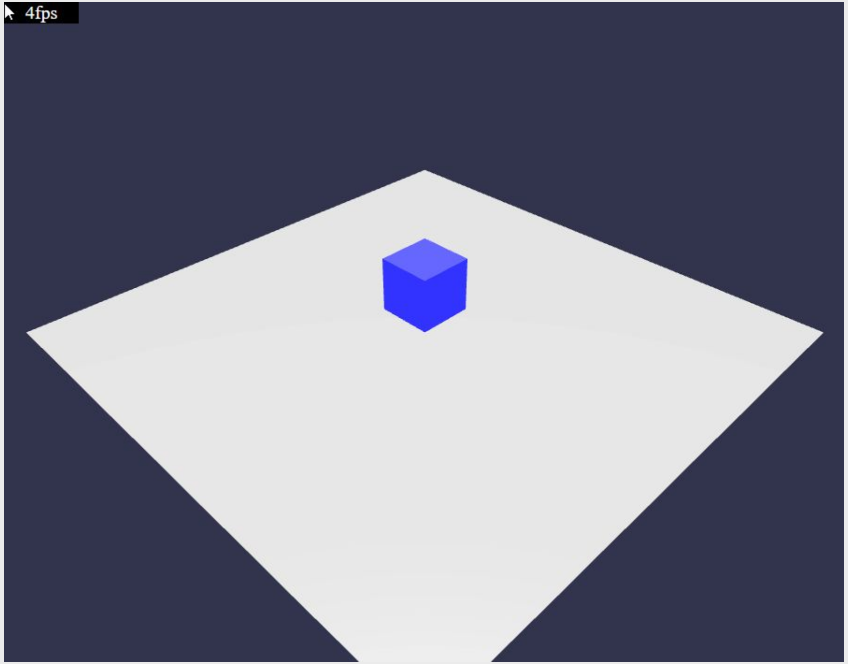
🔨

Libraries

Babylon.js

Babylon.js test scene (WebGL vs WebGPU):
https://youtu.be/eYgkDymaNr8?t=4

- Started support process in 2019.
- Support for WebGPU with Babylon 5.0.
- Feature parity with WebGL since January 2022.



Babylon.js interactivity example. Created by the authors.
Code available at: https://github.com/steschi/iaweb2022g1-survey

Two.js / Three.js

Two.js

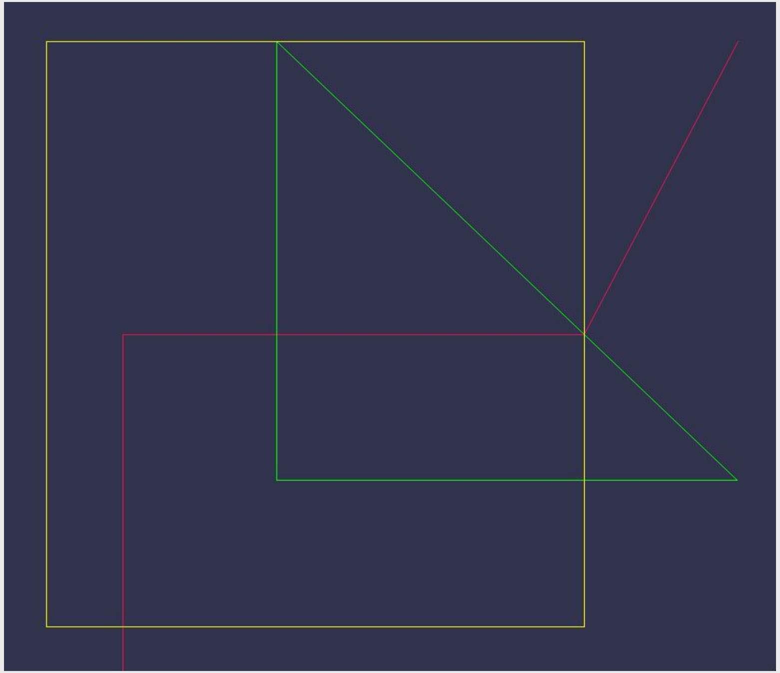
- WebGPU support is planned in the future.
- “… a WebGPU renderer seems like the natural evolution of renderers in Two.js.”

Three.js

- Basic WebGPU support implemented.
- Still some issues.
- Several examples already online.

Three.js Interactivity Example

- Hover needs to be implemented manually through JavaScript.
- Interactivity through raycasting.
  - Obtain list of intersected objects
- Ray intersection radius can be modified.



Three.js interactivity example. Created by the authors.
Code available at: https://github.com/steschi/iaweb2022g1-survey

🖥️

Cross-Platform

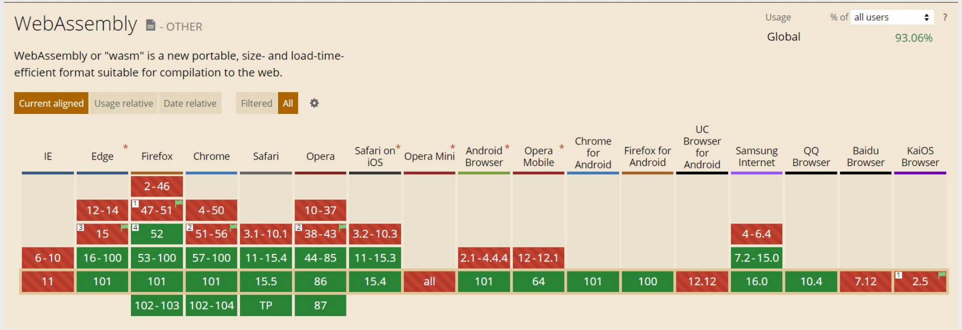
WebAssembly

Facts:

- Modern binary instruction format.
- Load-time-efficient virtual stack machine.
- Sandboxed and memory-safe.

Wasm allows to:

- Compile shader code for the browser without Javascript.
- Use existing native libraries within the browser.



WebAssembly - OTHER

WebAssembly or "wasm" is a new portable, size- and load-time-efficient format suitable for compilation to the web.

Usage % of all users

Global 93.06%

IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
		2 - 46														
	12 - 14	[1] 47 - 51	4 - 50		10 - 37											
	[3] 15	[4] 52	[2] 51 - 56	3.1 - 10.1	[2] 38 - 43	3.2 - 10.3							4 - 6.4			
6 - 10	16 - 100	53 - 100	57 - 100	11 - 15.4	44 - 85	11 - 15.3		2.1 - 4.4.4	12 - 12.1				7.2 - 15.0			
11	101	101	101	15.5	86	15.4	all	101	64	101	100	12.12	16.0	10.4	7.12	[1] 2.5
		102 - 103	102 - 104	TP	87											

Screenshot taken from https://caniuse.com/wasm

Engines

WebGPU support planned

Godot[1] `C/C++` `WebAssembly`

- Will drop WebGL for WebGPU.

PlayCanvas[2] `JavaScript`

- Major refactor to support WebGPU.

Unity[3] `C/C++` `WebAssembly`

- Looking for people to work on WebGPU.

No WebGPU support

Unreal[4] `C/C++`

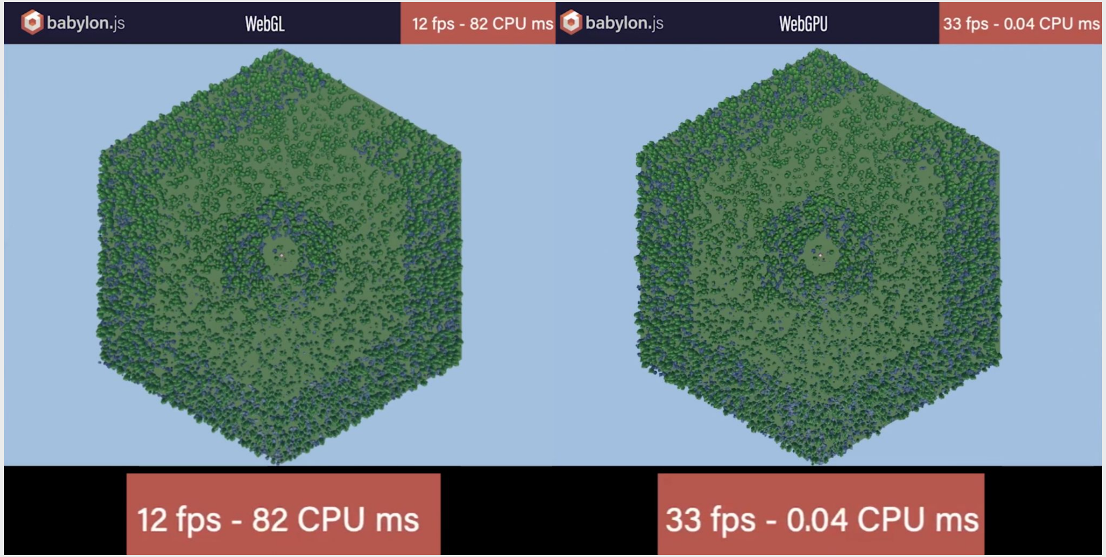
- No WebGPU and dropped WebGL support → native Vulkan

1: https://godotengine.org/
2: https://playcanvas.com/
3: https://unity.com/
4: https://www.unrealengine.com/

🚀

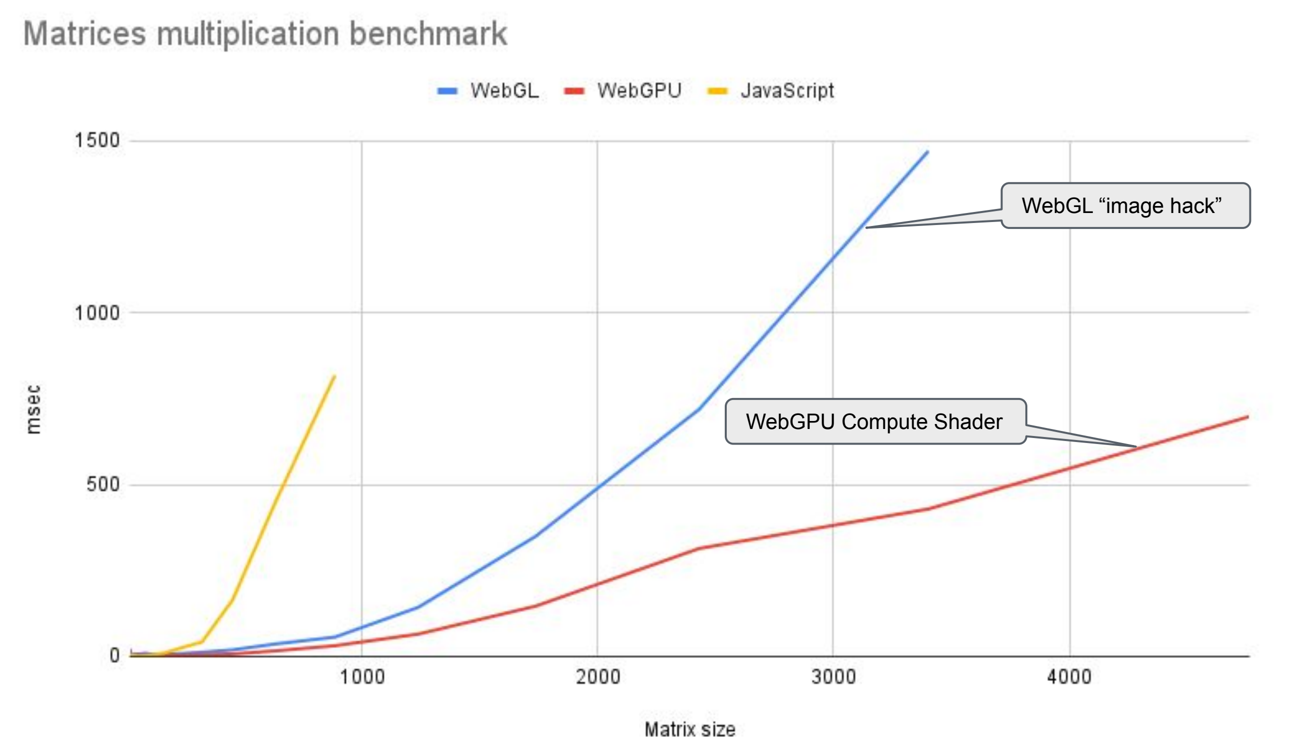
Performance

WebGL vs WebGPU (Babylon.js)



Screenshot taken from Babylon.js tech demo (WebGL vs WebGPU).
Full video on YouTube: https://youtu.be/eYgkDymaNr8
Copyright © Babylon.js. Used under §42f.(1) of Austrian copyright law.

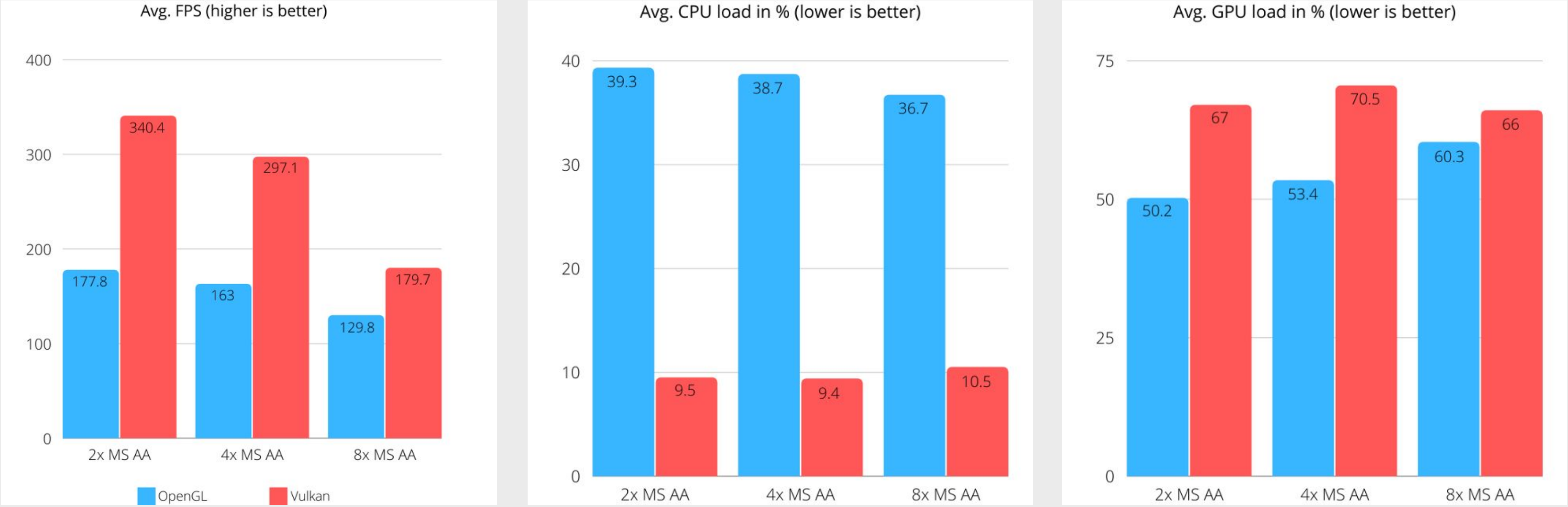
Matrix Multiplication (Compute)



Matrix multiplication, WebGL vs. WebGPU vs. Javascript. Image taken from: https://pixelscommander.com/javascript/webgpu-computations-performance-in-comparison-to-webgl/

OpenGl vs Vulkan

Multisampling Anti-Aliasing (MS AA) Test scene from Oreon Engine:
https://www.youtube.com/watch?v=hvdAVsjrQRM



Performance comparison between OpenGL and Vulkan. Images taken from:
https://eytanmanor.medium.com/the-story-of-webgpu-the-successor-to-webgl-bf5f74bc036a
Copyright © Eytan Manor. Used under §42f.(1) of Austrian copyright law.

Summary

- Writing native WebGPU code is complicated.
- WebGPU specification (types, documentation) in libraries & tools is still lacking.
- Currently best support with Babylon.js or Three.js.
- Recommendation: Wait for a stable release before adoption.

👏

Cheers.