Fast Interactive Web Graphics

Information Architecture and Web Usability

Group 4
Michael Anderson
Jyothish Atheendran
Petra Cukrov
Filip Ljubotina

2 Dec 2025 WS 2025/2026

Graphic technologies in the web browser:

- SVG-DOM
- Canvas2D
- WebGL
- WebGPU

SVG-DOM

- SVG elements are created and inserted into the Document Object Model (DOM) using JavaScript.
- SVG elements are vector graphics: freely scalable without loss of quality. [1]
- Performance degrades with growing number of distinct SVG nodes.
- The second edition of SVG 1.1 became a Recommendation in 2011.[2]
- Work on SVG 2.0 began in 2018, however many browsers still do not support it.

---

[1] https://tapflare.com/articles/web-graphics-comparison-canvas-svg-webgl

[2] https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorials/SVG_from_scratch/Introduction#:~:text=The%20second%20edition%20of%20SVG,and%20is%20the%20current%20standard.

SVG-DOM Implementation

```
const svg = document.createElementNS("http://www.w3.org/2000/svg", "svg");
svg.setAttribute("width", canvas.width);
svg.setAttribute("height", canvas.height);
document.body.appendChild(svg);

for (const d of dataset) {
  const polyline = document.createElementNS("http://www.w3.org/2000/svg", "polyline");
  polyline.setAttribute("points", getPolylinePoints(d));
  polyline.setAttribute("stroke", "steelblue");
  polyline.setAttribute("stroke-width", "2");
  polyline.setAttribute("fill", "none");
  svg.appendChild(polyline);
}
```

[2] Example SVG-DOM code; not from the final implementation.

Canvas2D

- `canvas="2d"`.
- Immediate-mode rendering:
  - The canvas doesn't "remember" objects; every frame is redrawn.[1]
- Enables manipulating pixels directly:
  - Used in dynamic graphics, games, or animations.[2]
- Mostly CPU-based:
  - Performance degrades with complex or high-volume drawing.[3]

[1] https://si.usi.ch/assets/publications/conf/icsa/icsa2017/TaivalsaariMPS17.pdf

[2] https://aircada.com/blog/webgl-vs-canvas

[3] https://blog.pixelfreestudio.com/webgl-vs-canvas-which-is-better-for-3d-web-development

Cavnas2D Implementation

```
    function initCanvas2D(dpr: number) {
      ctx = canvasEl.getContext("2d")!;
      ctx.setTransform(dpr, 0, 0, dpr, 0, 0);
      return ctx;
    }

    function redrawCanvasLines(dataset, parcoords) {
      for (const d of dataset) {
        const pts = getPolylinePoints(d, parcoords);
        ctx.beginPath();
        ctx.moveTo(pts[0][0], pts[0][1]);
        pts.slice(1).forEach(p => ctx.lineTo(p[0], p[1]));
        ctx.lineWidth = 2;
        ctx.strokeStyle = (lineState[getLineName(d)]?.active ?? true)
        ? "rgba(0,129,175,0.5)" : "rgba(211,211,211,0.4)";
        ctx.stroke();
    }
}
```

WebGL

- Based on OpenGL ES (Embedded Systems).[1]
- Allows web pages to display interactive graphics directly in the browser.[2]
- `canvas="webgl"`:
  - Released in 2011.
  - Supported on all modern browsers.
  - Based on OpenGL ES 2.0.
- `canvas="webgl2"`:
  - Released in 2017.[3]
  - New features: 3D textures, instancing, query objects.
  - Based on OpenGL ES 3.0.

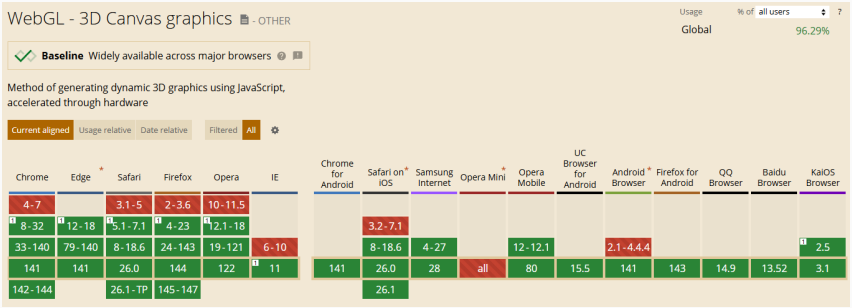
[1] `https://registry.khronos.org/webgl/specs/latest/2.0/`

[2] `https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API`

[3] `https://www.khronos.org/blog/webgl-2.0-arrives`

WebGL Browser Support

- Widely available across major browsers and devices.[1]
- Testing tool provided in browser.[2]



[1] https://caniuse.com/webgl

[2] https://get.webgl.org/

Shaders

- Both WebGL and WebGPU utilize a vertex and a fragment shader.
- Vertex shader:
  - Handles where each vertex of the triangle goes.[1]
  - Requires data in the form of 'attributes', which are sourced from GPU buffers.
  - Runs before the fragment shader.
- Fragment shader:
  - Handles what color each fragment (pixel) needs to be.[2]
  - Runs after the fragment shader.

[1] `https://webglfundamentals.org/webgl/lessons/webgl-shaders-and-glsl.html`

[2] https://www.codingwithjesse.com/blog/introduction-to-webgl-and-shaders/

Vertex Shader Example

- GPUs process vertex data to determine positions in screen space.
- Data can be passed to the fragment shader, in this example we pass a_color to the fragment shader.

```
const vertexShaderSrc = `
attribute vec2 position;
attribute vec4 a_color;
uniform vec2 resolution;
varying vec4 v_color;
void main() {
  gl_Position = vec4((position / resolution * 2.0 - 1.0) * vec2(1,-1), 0, 1);
  v_color = a_color;
}`;
```

Fragment Shader Example

- Runs once per pixel of the rendered shape.
- Outputs the final color of each pixel via gl_FragColor.

```
    const fragmentShaderSrc = `
    precision mediump float;
    varying vec4 v_color;
    void main() {
        gl_FragColor = v_color;
    }
    `;
```

WebGL Initialization

- Compiles and links vertex and fragment shaders into a GPU program.

```
function initCanvasWebGL(){
    gl = canvas.getContext("webgl");
    const vShader = createShader(gl, gl.VERTEX_SHADER, vertexShaderSrc);
    const fShader = createShader(gl, gl.FRAGMENT_SHADER, fragmentShaderSrc);
    program = createProgram(gl, vShader, fShader);

    gl.viewport(0, 0, canvasEl.width, canvasEl.height);

    vertexBuffer = gl.createBuffer();
    colorBuffer = gl.createBuffer();

    return gl;
}
```

WebGL Usage

```
function redrawWebGLLines(dataset: any[], parcoords: any) {
  gl.useProgram(program);
  gl.clear(gl.COLOR_BUFFER_BIT);

  //Loop through dataset points creating an array of vertices...

  const vertexData = new Float32Array(vertices);

  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, vertexData, gl.DYNAMIC_DRAW);
  gl.vertexAttribPointer(posLoc, 2, gl.FLOAT, false, 0, 0);

  gl.drawArrays(gl.LINES, 0, vertexData.length / 2);
}
```

WebGPU

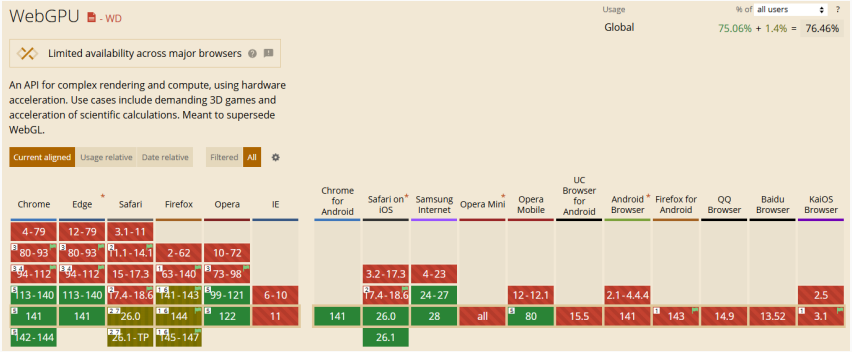
- Appeared around 2017[1] with major adoption taking place in 2025[2]
- Gives developers more control over the GPU than WebGL.
- Explicit control over GPU pipelines and memory for higher performance.

[1] https://developer.chrome.com/blog/webgpu-release

[2] https://web.dev/blog/webgpu-supported-major-browsers

WebGPU: Browser Support

- Supported in all modern browsers.
  - In some browsers WebGPU needs to be enabled.[1]



[1] https://caniuse.com/webgpu

WebGPU: Adoption

- Can convert most of WebGL2 programs to WebGPU, with limitations[1].
- 3D Graphics libraries that provide WebGPU support:
  - Three.js.
  - Pixi.js.
- Used in production[2]:
  - Google Meet background effects.
  - YouTube AR filters.
  - Figma rendering engine.

[1] https://dl.acm.org/doi/10.1145/3696410.3714785

[2] https://www.linkedin.com/pulse/webgpu-next-generation-browser-graphics-compute-4d-pipeline-5chyc/

WebGPU: Initialization

- WebGPU requires a GPU device and context to render anything.
- Every canvas must be associated with a GPU device and a pixel format.

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();

context = canvasEl.getContext("webgpu");
const canvasFormat = navigator.gpu.getPreferredCanvasFormat();
context.configure({
  device,
  format: canvasFormat,
  // the colors must have their values already multiplied by the alpha value.
  alphaMode: "premultiplied",
});
```

WebGPU: Shader

- Vertex shader transforms 2D points to clip space.
- Fragment shader outputs a uniform color per line.

```
const shaderModule = device.createShaderModule({
  code: `
  @group(0) @binding(0) var<uniform> color: vec4<f32>;
  struct VSOut { @builtin(position) position: vec4<f32> };
  @vertex
  fn vs_main(@location(0) pos: vec2<f32>) -> VSOut {
    var out: VSOut;
    out.position = vec4<f32>(pos, 0.0, 1.0);
    return out;}
  @fragment
  fn fs_main() -> @location(0) vec4<f32> { return color; }
  `
});
```

Defining a Vertex Buffer Layout in WebGPU

- A vertex buffer layout tells the GPU how to interpret vertex data.

```
const vertexBufferLayout: GPUVertexBufferLayout = {
    // Size of a single vertex in bytes (2 floats x 4 bytes each)
    arrayStride: 8,
    attributes: [
        {
            // Attribute format: 2 floats per vertex (x, y)
            format: "float32x2" as GPUVertexFormat,
            // Offset within the vertex
            offset: 0,
            // Shader location to link this attribute in the vertex shader
            shaderLocation: 0,
        } as GPUVertexAttribute,
    ],
};
```

WebGPU: Pipeline

- Pipeline defines shaders, vertex layout, primitive type, and blending.

```
const pipeline = device.createRenderPipeline({
  layout: device.createPipelineLayout({ bindGroupLayouts: [bindGroupLayout] }),
  vertex: { module: shaderModule, entryPoint: "vs_main", buffers: [vertexBufferLayout] },
  fragment: {
      module: shaderModule,
      entryPoint: "fs_main",
      targets: [{
          format: canvasFormat,
          blend: {
              color: { srcFactor: "src-alpha", dstFactor: "one-minus-src-alpha" },
              alpha: { srcFactor: "one", dstFactor: "zero" }
          }
      }]
  },
  primitive: { topology: "line-strip" }, });
```

WebGPU: Setting Up a Render Pass

- A render pass encodes commands related to controlling the vertex and fragment shader stages, as issued by a GPURenderPipeline.

```
const encoder = device.createCommandEncoder();

 const pass = encoder.beginRenderPass({
  colorAttachments: [{
      view: context.getCurrentTexture().createView(),
      loadOp: "clear",
      clearValue: { r: 0, g: 0, b: 0, a: 0 },
      storeOp: "store",
    }],
  });
```

WebGPU: Drawing Polylines

- Combine all lines into one buffer.
- Use ‘draw(vertexCount, 1, vertexOffset, 0)‘ to draw each line slice.
- Bind different colors for active/inactive lines.

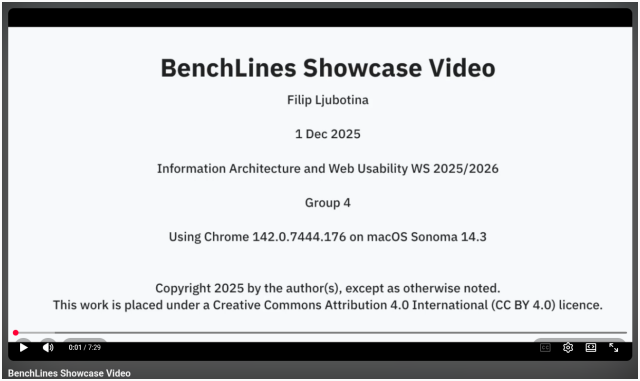
```
  pass.setPipeline(pipeline);
  pass.setVertexBuffer(0, vertexBuffer);

  let offset = 0;
  for (const line of allLines) {
    pass.setBindGroup(0, line.active ? activeBindGroup : inactiveBindGroup);
    pass.draw(line.pts.length, 1, offset, 0);
    offset += line.pts.length;
  }
  pass.end();
  device.queue.submit([encoder.finish()]);
```

Key Differences: WebGL vs WebGPU

WebGL	WebGPU
Simpler API, easier to use	More complex API, deeper control
Limited control over GPU	Explicit control over pipelines and memory management
Performance limited by older GPU paradigms	Higher, more predictable performance

BenchLines Demo



https://youtu.be/377dVewsqMc

Benchmarking Results

Technology	Dataset	Iterations	Time (ms)
SVG-DOM	20D_1000	10	15.140
WebGL-Pixi	20D_1000	10	12.440
Canvas2DPixi	20D_1000	10	9.980
WebGPU-Pixi	20D_1000	10	9.580
WebGLThree	20D_1000	10	8.810
WebGPU-Three	20D_1000	10	8.020
Canvas2D	20D_1000	10	4.990
WebGL	20D_1000	10	4.560
WebGPU	20D_1000	10	3.700

JS Graphic Libraries

- **Three.js** [3D: WebGL, WebGPU, SVG-DOM][1][2]
  https://threejs.org
- **PixiJS** [2D: WebGL, Canvas2D, WebGPU]
  https://pixijs.com

[1] https://github.com/mrdoob/three.js/releases/tag/r98

[2] https://x.com/mrdoob/status/1058022036038148096

Libraries: Three.js

- Goal: simplifying the process of creating 3D experiences for the web. [1]
- Three main components: Scene, Camera and Renderer. [2]
- Supported on all modern browsers that implement WebGL. [3]

[1] https://threejsresources.com/facts

[2] https://threejs.org/manual/#en/creating-a-scene

[3] https://threejsresources.com/facts

Libraries: Pixi.js

- Lightweight 2D library for the web.
- Offers multi-platform support.
- Support for click and touch interactions.[1]
- Ensures compatibility with older devices that cannot run WebGL:
  - Has integrated Canvas fallback.[2]

[1] https://pixijs.com/8.x/guides/getting-started/intro

[2] https://pixijs.com/8.x/guides/components/renderers

Thank You!