MY.GAMES

Dmitrii Ivashchenko
Lead Software Engineer

Migration from WebGL to WebGPU

Background

- Timeline of WebGL & WebGPU
- Current State of WebGPU

Timeline of WebGL & WebGPU

WebGL 1.0

2011

OpenGL ES 2.0 (2007)

WebGL 2.0

2017

OpenGL ES 3.0 (2012)

WebGPU

Mid 2021

Vulkan (2016)
Direct3D 12 (2015)
Metal (2014)

Current State of WebGPU

Platforms

- **Chrome 113 -** ✅shipped
- **Edge 113 -** ✅shipped
- **Firefox -** 👷in development
- **Safari -** 👷in development

Web Engines

- **Babylon JS -** ✅ full WebGPU support
- **ThreeJS -** 🧪experimental support
- **PlayCanvas -** 👷in development

Game Engines

- **Cocos2d -** ✅officially supports WebGPU
- **Construct -** 🔶not all platforms are supported
- **Unity -** 🧪early experimental support in 2023.2 alpha

WebGPU core concepts

- GPUAdapter
- GPUDevice
- Features And Limits
- GPUCanvasContext
- Resource types
- Queue

GPUAdapter


```
await navigator.gpu.requestAdapter(options);

- { powerPreference: 'low-power' }

- {vendor: “nvidia”, architecture: “turing” }
```

GPUDevice


```
await adapter.requestDevice(options);
```

Features And Limits

- Roughly equivalent to WebGL's extensions
- Typically things not supported on all implementations/systems
  - `"texture-compression-bc"`
  - `"timestamp-query"`
- Adapter lists which ones are available.
- Must be specified when the requesting a Device or they won't be active.

- Numeric limits of GPU capabilities
  - `maxTextureDimension2D`
  - `maxBindGroups`
  - `maxVertexBuffers`
- Each has a baseline that all WebGPU implementations must support.
- Adapter reports the actual system limits.
- Devices will only have access to the default limits unless otherwise specified when requesting the Device.

GPUCanvasContext


```
// During initialization
const context = canvas.getContext('webgpu');
context.configure({
   device,
   format: 'bgra8unorm',
});

// During frame loop
const renderTarget = context.getCurrentTexture();
```

Resource types


```
const buffer = device.createBuffer({
   size: 2048, // Bytes
   usage: GPUBufferUsage.VERTEX |
GPUBufferUsage.COPY_DST,
});

const texture = device.createTexture({
  size: { width: 64, height: 64 },
  mipLevelCount: 4,
  format: 'rgba8unorm',
  usage: GPUTextureUsage.TEXTURE_BINDING,
});
const textureView = texture.createView({
  baseMipLevel: 1,
  mipLevelCount: 1,
});
```

```
const sampler = device.createSampler({
  magFilter: "nearest",
  minFilter: "linear",
  mipmapFilter: "linear",
  addressModeU: "repeat",
  addressModeV: "clamp-to-edge",
});
```

Queue


```
device.queue.writeBuffer(buffer, 0, typedArray);
device.queue.writeTexture({ texture: dstTexture },
                          typedArray,
                          { bytesPerRow: 256 },
                          { width: 64, height: 64 });
```

Recording GPU commands


```
const commandEncoder = device.createCommandEncoder();
commandEncoder.copyBufferToBuffer(bufferA, 0,
                                  bufferB, 0, 256);

const passEncoder = commandEncoder.beginComputePass();
passEncoder.setPipeline(pipeline);
passEncoder.setBindGroup(0, bindGroup);
passEncoder.dispatchWorkgroups(128);
passEncoder.end();

const commandBuffer = commandEncoder.finish();
device.queue.submit([commandBuffer]);
```

Passes


```
const renderPass = commandEncoder.beginRenderPass({
  colorAttachments: [{
    view: context.getCurrentTexture().createView(),
    loadOp: 'clear',
    clearValue: [0.0, 0.0, 0.0, 1.0],
    storeOp: 'store',
  }]
});
renderPass.setPipeline(renderPipeline);
renderPass.setBindGroup(0, bindGroup);
renderPass.setVertexBuffer(0, vertexBuffer);
renderPass.draw(3);
renderPass.end();

const computePass = commandEncoder.beginComputePass();
computePass.setPipeline(computePipeline);
computePass.setBindGroup(0, bindGroup);
computePass.dispatchWorkgroups(128);
computePass.end();
```

High-level Conceptual Differences

- Initialization
- Programs vs Pipelines

Initialization

WebGL

```
const gl = canvas.getContext('webgl');
```

WebGPU

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

const device = await adapter.requestDevice();

const context = canvas.getContext('webgpu');

context.configure({
  device,
  format: 'bgra8unorm',
});
```

Buffers

High-level Conceptual Differences

WebGL

```
const vertexData = new Float32Array([
   0, 1, -1,
   -1, -1, -1,
   1, -1, -1
]);

const vertexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER,
vertexBuffer);
gl.bufferData(gl.ARRAY_BUFFER, vertexData,
    gl.STATIC_DRAW);
```

WebGPU

```
const vertexData = new Float32Array([
   0, 1, 1,
   -1, -1, 1,
   1, -1, 1
]);

const vertexBuffer = device.createBuffer({
   size: vertexData.byteLength,
   usage: GPUBufferUsage.VERTEX | GPUBufferUsage.COPY_DST,
});
device.queue.writeBuffer(vertexBuffer, 0, vertexData);
```

Shaders

High-level Conceptual Differences

WebGL

```
const vertShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertShader, `
 attribute vec3 position;
 void main() {
   gl_Position = vec4(position, 1);
 }`);
gl.compileShader(vertShader);

const fragShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fragShader, `
 precision mediump float;
 void main() {
   gl_FragColor = vec4(1, 0, 0, 1);
 }`);
gl.compileShader(fragShader);
```

WebGPU

```
const shaderModule = device.createShaderModule({
 code: `
   @vertex
   fn vertexMain(@location(0) pos : vec3<f32>) ->
       @builtin(position) vec4<f32> {
     return vec4(pos, 1.0);
   }
   @fragment
   fn fragmentMain() -> @location(0) vec4<f32> {
     return vec4(1.0, 0.0, 0.0, 1.0);
   }`
});
```

Programs vs Pipelines

High-level Conceptual Differences

WebGL

```
const program = gl.createProgram();

gl.attachShader(program, vertShader);

gl.attachShader(program, fragShader);

gl.bindAttribLocation(program, 'position', 0);

gl.linkProgram(program);
```

WebGPU

```
const pipeline = device.createRenderPipeline({
 layout: 'auto',
 vertex: {
   module: shaderModule,
   entryPoint: 'vertexMain',
   buffers: [{
     arrayStride: 12,
     attributes: [{
       shaderLocation: 0, offset: 0, format: 'float32x3'
     }]
   }],
 },
 fragment: {
   module: shaderModule,
   entryPoint: 'fragmentMain',
   targets: [{ format, }],
 },
});
```

Drawing

High-level Conceptual Differences

WebGL

```
gl.clearColor(0, 0, 0, 1);
gl.clear(gl.COLOR_BUFFER_BIT);

gl.useProgram(program);

gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
gl.vertexAttribPointer(0, 3, gl.FLOAT, false, 12, 0);
gl.enableVertexAttribArray(0);

gl.drawArrays(gl.TRIANGLES, 0, 3);
```

WebGPU

```
const commandEncoder = device.createCommandEncoder();
const passEncoder = commandEncoder.beginRenderPass({
 colorAttachments: [{
   view: context.getCurrentTexture().createView(),
   loadOp: 'clear',
   clearValue: [0.0, 0.0, 0.0, 1.0],
   storeOp: 'store',
 }]
});

passEncoder.setPipeline(pipeline);
passEncoder.setVertexBuffer(0, vertexBuffer);
passEncoder.draw(3);
passEncoder.end();

device.queue.submit([commandEncoder.finish()]);
```

Uniforms

- Uniforms in WebGL 1.0 and 2.0
- Uniforms in WebGPU

Uniforms in WebGL 1.0


```
// GLSL

uniform vec3 u_LightPos;
uniform vec3 u_LightDir;
uniform vec3 u_LightColor;


// JavaScript

const location = gl.getUniformLocation(p, "u_LightPos");
gl.uniform3fv(location, [100, 300, 500]);
```

Uniforms in WebGL 2.0


```
// GLSL

layout(std140) uniform ub_Params {
   vec4 u_LightPos;
   vec4 u_LightDir;
   vec4 u_LightColor;
};
```

```
// JavaScript

gl.bindBufferBase(gl.UNIFORM_BUFFER, 1, gl.createBuffer());
```

Uniforms in WebGPU


```
// WGSL

[[block]] struct Params {
   u_LightPos : vec4<f32>;
   u_LightColor : vec4<f32>;
   u_LightDirection : vec4<f32>;
};
[[group(0), binding(0)]] var<uniform> ub_Params : Params;


// JavaScript

const buffer = device.createBuffer({
  usage: GPUBufferUsage.UNIFORM,
  size: 8
});
```

Shaders

- GLSL vs WGSL
- Comparison of Data Types
- Structures
- Function Declarations
- Built-in functions

GLSL vs WGSL

WebGL

```
sampler2D myTexture;
varying vec2 vTexCoord;
void main() {
  return texture(myTexture, vTexCoord);
}
```

WebGPU

```
[[group(0), binding(0)]] var mySampler: sampler;
[[group(0), binding(1)]] var myTexture: texture_2d<f32>;

[[stage(fragment)]]
fn main([[location(0)]] vTexCoord: vec2<f32>) ->
  [[location(0)]] vec4<f32>
{
  return textureSample(myTexture, mySampler, vTexCoord);
}
```

Comparison of Data Types

Basic Types

GLSL	WGSL
`int`	`i32`
`uint`	`u32`
`float`	`f32`
`double`	N/A

Matrix Data Types

GLSL	WGSL
`mat2`	`mat2x2<f32>`
`mat3x2`	`mat3x2<f32>`
`mat4x2`	`mat4x2<f32>`
`mat2x3`	`mat2x3<f32>`
`mat3`	`mat3x3<f32>`
`mat4x3`	`mat4x3<f32>`
`mat2x4`	`mat2x4<f32>`
`mat3x4`	`mat3x4<f32>`
`mat4`	`mat4x4<f32>`

Structures

WebGL

```
struct Light {
  vec3 position;
  vec4 color;
  float attenuation;
  vec3 direction;
  float innerAngle;
  float angle;
  float range;
};
```

WebGPU

```
struct Light {
  position: vec3<f32>,
  color: vec3<f32>,
  attenuation: f32,
  direction: vec3<f32>,
  innerAngle: f32,
  angle: f32,
  range: f32,
};
```

Function Declarations

WebGL

```
float saturate(float x) {
  return clamp(x, 0.0, 1.0);
}
```

WebGPU

```
fn saturate(x: f32) -> f32 {
  return clamp(x, 0.0, 1.0);
}
```

Built-in functions

GLSL	Stage	IO	WGSL
`gl_VertexID`	vertex	in	`vertex_index`
`gl_InstanceID`	vertex	in	`instance_index`
`gl_Position`	vertex	out	`position`
`gl_FragCoord`	fragment	in	`position`
`gl_FrontFacing`	fragment	in	`front_facing`
`gl_FragDepth`	fragment	out	`frag_depth`
`gl_LocalInvocationID`	compute	in	`local_invocation_id`
`gl_LocalInvocationIndex`	compute	in	`local_invocation_index`

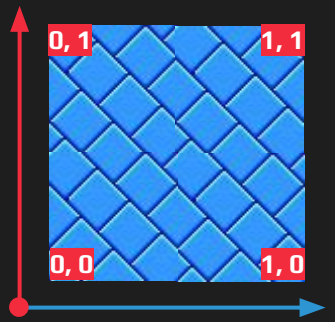
...

Convention Differences

- Texture2D
- Viewport Space
- Clip Spaces

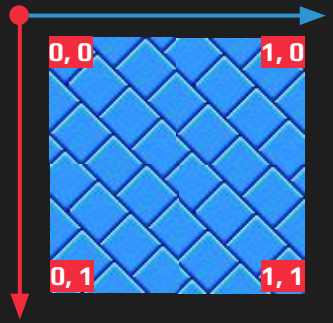
Texture2D

WebGL



The start point corresponds to the **bottom left** corner

WebGPU



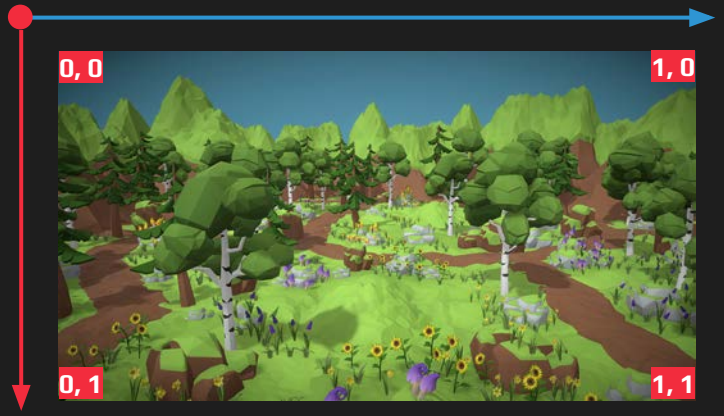
Direct3D and Metal have traditionally used the **top left** corner as the starting point for textures

Viewport Space

Convention Differences

WebGL

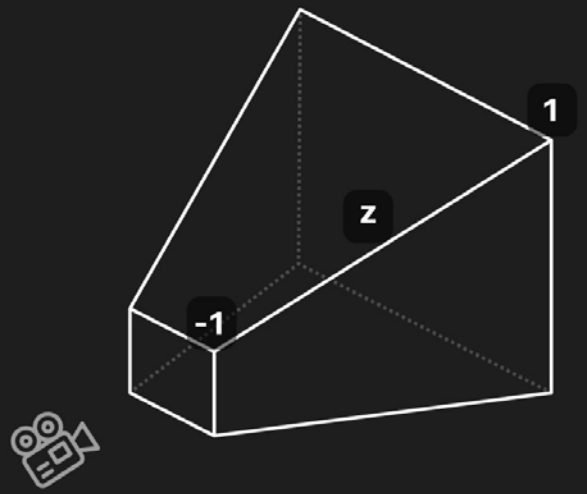
WebGPU



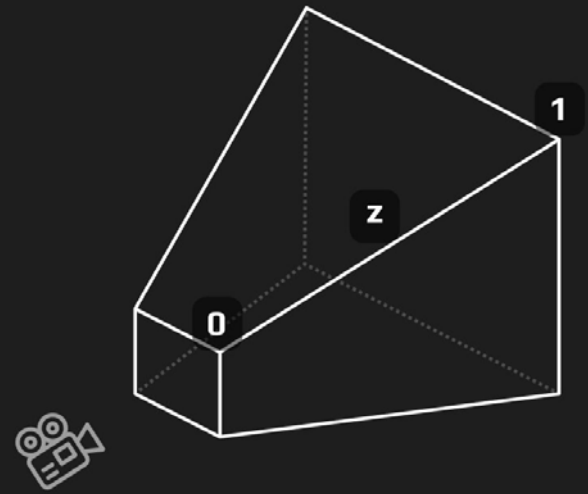
Clip Spaces

Convention Differences

WebGL



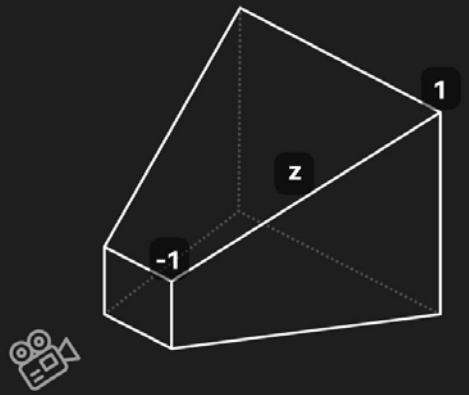
WebGPU



Clip Spaces

Convention Differences

```
if (webGPU) {
    mat4.perspectiveZO(out, Math.PI / 4, ...);
} else {
    mat4.perspective(out, Math.PI / 4, ...);
}
```



$$\text{proj}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 1 \end{bmatrix} \text{proj}$$

WebGPU tips

Best practices for performance

Tip #1

Minimize the number of pipelines you use

Tip #2

Create pipelines in advance

```
const pipeline = device.createComputePipeline({
 compute: {
   module: shaderModule,
   entryPoint: 'computeMain'
 }
});
const commandEncoder = device.createCommandEncoder();
const passEncoder = commandEncoder.beginComputePass();
passEncoder.setPipeline(pipeline);
passEncoder.setBindGroup(0, bindGroup);
passEncoder.dispatchWorkgroups(128);
passEncoder.end();
device.queue.submit([commandEncoder.finish()]);
```

Tip #2

Create pipelines in advance

```
device.createComputePipelineAsync({
 compute: {
   module: shaderModule,
   entryPoint: 'computeMain'
 }
}).then((pipeline) => {
  const commandEncoder = device.createCommandEncoder();
  const passEncoder = commandEncoder.beginComputePass();
  passEncoder.setPipeline(pipeline);
  passEncoder.setBindGroup(0, bindGroup);
  passEncoder.dispatchWorkgroups(128);
  passEncoder.end();
  device.queue.submit([commandEncoder.finish()]);
});
```

Tip #3

Use RenderBundles

```
const encoder = device.createRenderBundleEncoder({
 colorFormats: ['bgra8unorm'],
 depthStencilFormat: 'depth24plus',
});

encoder.setPipeline(pipeline);
encoder.setBindGroup(0, bindGroupA);
encoder.setVertexBuffer(0, vertexBufferA);
encoder.draw(1024);

encoder.setBindGroup(0, bindGroupB);
encoder.setVertexBuffer(0, vertexBufferB);
encoder.setIndexBuffer(indexBuffer);
encoder.drawIndexed(2048);

const renderBundle = encoder.finish();
```

Tip #3

Use RenderBundles

```
const renderPass = encoder.beginRenderPass(
 descriptor);

renderPass.setPipeline(renderPipeline);
renderPass.draw(3);

renderPass.executeBundles([renderBundle]);

renderPass.setPipeline(renderPipeline);
renderPass.draw(3);

renderPass.end();
```

Resources & Links

- WebGL + WebGPU Meetup - July 2023
- WebGPU — All of the cores, none of the canvas
- From WebGL to WebGPU in Construct
- Raw WebGPU tutorial by Alain Galvan
- WebGPU Best Practices by Brandon Jones