WebXR+WebGPU Projection Matrices

Projection Matrices: The Problem

WebGL’s Normalized Device Coordinates (NDC) transforms everything into a -1 to 1 range for depth values, using OpenGL’s convention.

WebGPU’s NDC uses a 0 to 1 range for depth values, using Vulkan, Metal, and D3Ds convention.

This means that if a WebGL projection matrix is used for WebGPU content, content will generally appear correct but you’ll lose half your depth range. (Could lead to Z fighting on WebGPU that wasn’t present on WebGL.)

A note about viewports

Viewports ALSO have this problem:
WebGL's viewports have an origin in the bottom left with +Y axis going up.
WebGPU's viewports have an origin in the top left with +Y axis going down.

But viewport are already queried with an graphics API-specific entry point, the layer, so we can just return the right thing for the layer type! (Thanks Nik for pointing this out!)

Proposed solution

Add a (nullable?) `projectionMatrix` attribute to `XRSubImage`.

```
[Exposed=Window] interface XRSubImage {
  [SameObject] readonly attribute XRViewport viewport;
  readonly attribute Float32Array? projectionMatrix;
};
```

Somewhat awkward caveat: It would be null for non-projection layers (quad, cylinder, etc.)

Alternative solution

Add a `getViewProjectionMatrix()` method to `XRGPUBinding` and `XRWebGLBinding`.

```
partial interface XRGPUBinding {
  // ...
  XRGPUSubImage getViewSubImage(XRProjectionLayer layer, XRView view);
  Float32Array getViewProjectionMatrix(XRView view);
};
```

Alternative “solution”

- Leave it alone and tell developers the math to fix it: 😒

```
mtxGLToGPU = [      // Not real JS
  1, 0,   0, 0,
  0, 1,   0, 0,
  0, 0, 0.5, 0,
  0, 0, 0.5, 1,
];
projMatGPU = view.projectionMatrix * mtxGLToGPU;
```