WebXR Layers

Immersive Web Seattle Face to Face
Feb 2020

My Motivation: Wider Graphics API support

WebGPU
WebGL 2.0 (Array Textures)

Not diving into details of multilayer*

*For the purposes of this slide deck, anyway. I still think it should happen.

Expanding on Artem's "Layers Core" proposal

Defines two basic parts:

- **Layer** types that define **How** the layer's content is shown
- **Layer Source** types that define **What** is show.

Borrows heavily from OpenXR concepts (a Good Thing)

Preserve our prescriptive rendering patterns

OpenXR relies on devs to allocate swap chains of the appropriate size/count for the device and intended use.

WebXR should do the obvious right thing whenever possible, with knobs to handle non-obvious tweaks.

Main Idea:

Create layers to define presentation in the world,
Then create a layer source which automatically allocates
the GPU resources needed to satisfy the layer’s needs.

```
// Setup
// Was: let layer = new XRWebGLLayer(xrSession, gl);

let layer = xrSession.requestProjectionLayer();
let source = new XRWebGLFramebufferLayerSource(layer, gl, {/*The usual*/});

xrSession.updateRenderState({ baseLayer: layer });

// Render Loop (same as it ever was)
gl.bindFramebuffer(gl.FRAMEBUFFER, source.framebuffer);
for (let view in xrViewerPose.views) {
  let viewport = layer.getViewSubImage(view).viewport;
  gl.viewport(viewport.x, viewport.y, viewport.width, viewport.height);
  // render
}
```

Simple WebGL content under this proposal

XRWebGLLayer Backwards Compatibility

```
interface XRWebGLLayer extends XRProjectionLayer {
  // Constructs an XRWebGLFramebufferLayerSource internally
  // and reflects all it’s methods and attributes here using
  // the same verbiage as the original XRWebGLLayer.
}
```

baseLayer now accepts any XRProjectionLayer.

```
// Setup
let layer = xrSession.requestProjectionLayer();
let source = new XRWebGL2TextureLayerSource(layer, gl2, {/*The usual*/});
let framebuffer = gl.createFramebuffer();

xrSession.updateRenderState({ baseLayer: layer });

// Render Loop (same as it ever was)
gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
for (let view in xrViewerPose.views) {
  let subImage = layer.getViewSubImage(view);
  gl.framebufferTextureLayer(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
        source.colorTexture, 0, subImage.imageIndex);
  gl.framebufferTextureLayer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
        source.depthStencilTexture, 0, subImage.imageIndex);
  let viewport = subImage.viewport;
  gl.viewport(viewport.x, viewport.y, viewport.width, viewport.height);
  // render
}
```

WebGL 2.0 Texture Array support

What about quad/equirect/etc layers?

```
let quadLayer = xrSession.requestQuadLayer();
quadLayer.referenceSpace = xrRefSpace;
quadLayer.transform = new XRRigidTransform(/*...*/);
quadLayer.width = 4;
quadLayer.height = 3;

let source = new XRWebGL2TextureLayerSource(quadLayer, gl2, {
  width: 1024, height: 768, stereo: true
});

xrSession.updateRenderState({ layers: [projectionLayer, quadLayer] });

// Render Loop
for (let eye in ["left", "right"]) {
  let subImage = quadLayer.getEyeSubImage(eye);
  /* … */
  // render
}
```

Difference with quad layer is that you need to specify dimensions/stereo-ness of the source

Sample IDL

```
partial interface XRSession {
  XRProjectionLayer? requestProjectionLayer();
  XRQuadLayer? requestQuadLayer();
  XRCylinderLayer? requestCylinderLayer();
}

interface XRSubImage {
  readonly unsigned long imageIndex;
  readonly XRViewport viewport;
}

interface XRLayer {
  XRReferenceSpace referenceSpace;
  boolean blendTextureSourceAlpha = false;
  boolean chromaticAberrationCorrection = false;
}

interface XRProjectionLayer extends XRLayer {
  XRSubImage getViewSubImage(XRView view);
}
```

```
interface XRQuadLayer extends XRLayer {
  XRSubImage getEyeSubImage(XREye eye);

  attribute XRRigidTransform transform;
  attribute float width;
  attribute float height;
}

interface XRCylinderLayer extends XRLayer {
  XRSubImage getEyeSubImage(XREye eye);

  attribute XRRigidTransform transform;
  attribute float radius;
  attribute float centralAngle;
  attribute float aspectRatio;
}

typedef (XRQuadLayer or XRCylinderLayer) XRNonProjectionLayer;
```

```
interface XRWebGLFramebufferLayerSource {
  constructor(XRProjectionLayer layer, XRWebGLRenderingContext context,
              XRWebGLLayerInit init);
  constructor(XRNonProjectionLayer layer, XRWebGLRenderingContext context,
              XRNonProjectionLayerSourceInit init);

  readonly attribute boolean antialias;
  readonly attribute boolean ignoreDepthValues;

  [SameObject] readonly attribute WebGLFramebuffer? framebuffer;
  readonly attribute unsigned long framebufferWidth;
  readonly attribute unsigned long framebufferHeight;
}

dictionary XRNonProjectionLayerSourceInit {
  unsigned int width;
  unsigned int height;
  boolean stereo = false;
  boolean depth = true;
  boolean stencil = false;
  boolean alpha = true;
  boolean ignoreDepthValues = false;
}
```

```
interface XRWebGLTextureLayerSource {
  constructor(XRProjectionLayer layer, WebGL2RenderingContext context,
              XRWebGLLayerInit init);
  constructor(XRNonProjectionLayer layer, WebGL2RenderingContext context,
              XRNonProjectionLayerSourceInit init);

  readonly attribute boolean ignoreDepthValues;

  [SameObject] readonly attribute WebGLTexture colorTexture;
  [SameObject] readonly attribute WebGLTexture? depthStencilTexture;

  readonly attribute unsigned long textureWidth;
  readonly attribute unsigned long textureHeight;
  readonly attribute unsigned long textureArraySize;
}
```

```
enum XRVideoStereoLayout {
  "mono",
  "top-bottom",
  "left-right",
}

interface XRVideoLayerSource {
  constructor(XRNonProjectionLayer layer, HTMLVideoElement video,
              XRVideoStereoLayout stereoLayout = "mono");
}
```