Deep Dive on HDR, WCG and Linear Rendering in WebGL and WebGPU

Jeff Gilbert, Mozilla
Ken Russell, Google

Background

- WebGL and the forthcoming WebGPU provide GPU-accelerated, immediate mode graphics rendering to the web
- Native game engines and image processing applications have pushed the forefront of rendering fidelity
- The web must keep up to stay competitive

Linear vs. Non-Linear Gamma

- Most game engines today operate in linear-gamma space to support physically based rendering (PBR)
  - Real-time PBR is one of the major innovations in 3D graphics in recent years
- The Unity Engine is an important customer publishing games to the web
- Unity documents their linear vs. non-linear gamma workflows
- All of Unity's Tiny examples use the linear workflow
  - Running in WebAssembly / JavaScript
  - Rendering with WebGL
- For example, Tiny Racing:

API Support for Linear Rendering

- Both WebGL and WebGPU support the “sRGB-encoded” texture formats needed for an 8-bit-per-channel linear rendering workflow
- However, currently WebGL always uses an 8-bit normalized backbuffer format, which is unsuitable for linear rendering
- Therefore, working in the sRGB-encoded formats needed for linear rendering currently imposes a full-canvas blit at the end of each frame!
    - Prohibitively expensive on many GPUs, both desktop and mobile

Extraneous Full-Canvas Copies

```
// This is what's necessary today.
// OpenGL ES 3.0 spec forbids converting/resolving in one step.

// "multisampledFramebuffer" is the application's antialiased,
// linear-gamma rendering output - SRGB8_ALPHA8 renderbuffer attachment.
// "resolvedFramebuffer" has a (single-sampled) SRGB8_ALPHA8 texture as
// its color attachment.
gl.bindFramebuffer(gl.READ_FRAMEBUFFER, multisampledFramebuffer);
gl.bindFramebuffer(gl.DRAW_FRAMEBUFFER, resolvedFramebuffer);
gl.blitFramebuffer(0, 0, width, height, ...);

// Next, copy to WebGL's (RGBA8) drawingBuffer:
gl.bindFramebuffer(gl.READ_FRAMEBUFFER, resolvedFramebuffer);
gl.bindFramebuffer(gl.DRAW_FRAMEBUFFER, null);
gl.blitFramebuffer(0, 0, width, height, ...);
```

Without extraneous Full-Canvas Copies

```
// Possible soon!
// Resolve directly to drawing buffer, converting/resolving in one step.

// "multisampledFramebuffer" is the application's antialiased,
// linear-gamma rendering output - SRGB8_ALPHA8 renderbuffer attachment.
// "resolvedFramebuffer" has a (single sampled) SRGB8_ALPHA8 texture as
// its color attachment.
gl.bindFramebuffer(gl.READ_FRAMEBUFFER, multisampledFramebuffer);
gl.bindFramebuffer(gl.DRAW_FRAMEBUFFER, resolvedFramebuffer);
gl.blitFramebuffer(0, 0, width, height, ...);

// Next, copy to WebGL's (RGBA8) drawingBuffer:
gl.bindFramebuffer(gl.READ_FRAMEBUFFER, resolvedFramebuffer);
gl.bindFramebuffer(gl.DRAW_FRAMEBUFFER, null);
gl.blitFramebuffer(0, 0, width, height, ...); // Resolve directly!
```

WebGL API Proposal

- Allow drawingBuffer's storage format to be changed:

```
const gl = canvas.getContext('webgl2');
gl.drawingBufferStorage(
    gl.SRGB8_ALPHA8, canvas.width, canvas.height);
```

- Default is RGB[A]8 depending on `alpha:true/false`
- Alternatives considered:
  - Choose storage format at context creation
  - Rejected as too inflexible; doesn't work with WebGL 1.0, where extensions are needed

WebGPU API Proposal

- WebGPU already supports 8-bit-per-channel linear-gamma rendering via GPUSwapChainDescriptor
- Per Chris Cameron (Google)'s comprehensive description:

```
const gpuPresentContext = canvas.getContext('gpupresent');
const swapChain = gpuPresentContext.configureSwapChain({
      device: device,
      format: 'rgba8unorm-srgb'});
```

Wide Color Gamut Support

- Applications need to be able to specify the color space of WebGL's and WebGPU's rendering results. Work-in-progress here.
- WebGL: expose colorSpace as a mutable attribute on the context:

```
const gl = canvas.getContext('webgl2');
gl.colorSpace = 'display-p3';
```

- WebGPU: allow configuration of colorSpace on swap chain:

```
const format = gpuPresentContext.getSwapChainPreferredFormat(adapter);
const swapChain = gpuPresentContext.configureSwapChain({
      device: device,
      format: format,
      colorSpace: 'display-p3'});
```

High Dynamic Range Support

- Will be common across WebGL and WebGPU
- Want web apps to take advantage of increasingly widespread HDR displays
- HDR consumes more power, so must be opt-in by application
- Currently being discussed in Chris Cameron's HDR proposal

High Dynamic Range Support

- One possible direction for WebGL support:

```
const gl = canvas.getContext('webgl2');
gl.drawingBufferStorage(gl.RGBA16F, canvas.width, canvas.height);
gl.colorSpace = 'srgb-linear';
canvas.configureHDR('extended'); // Discussing where this belongs!
gl.clearBufferfv(gl.COLOR, 0, [2,0,0,1]); // Unclamped super-red!
```

- WebGPU support expected to be similar to WebGL:

```
const gpuPresentContext = canvas.getContext('gpupresent');
const swapChain = gpuPresentContext.configureSwapChain({
      device: device,
      format: 'rgba16float',
      colorSpace: 'srgb-linear'});
canvas.configureHDR('extended'); // Discussing where this belongs!
```

Summary

- These new proposals for linear-gamma rendering, extended color spaces, and high dynamic range enable a new level of rendering quality in WebGL and WebGPU
- Under active development and collaboration here, so please join the discussions and contribute!
- The work is ongoing in the ColorWeb CG, WebGL WG, and WebGPU CG.

Acknowledgements

- Special thanks to Chris Cameron (Google) for driving many of these discussions and proposals

Thanks!