ShaderToy

Iñigo Quilez and Pol Jeremias

ShaderToy

Shadertoy is an online shader creation tool and community, where people learn, teach, experiment and show off their skills. Based on WebGL 2 and GLSL 330, but without using polygons, Shadertoy allows and challenges people to create computer animations and demos, and encourages a procedural approach to it.

Recently Shadertoy transitioned from WebGL 1 to WebGL 2, and this is a little summary of our experience in such transition:

WebGL 2.0 Transition

The transition to WebGL 2.0 was pretty smooth for us.

- Users write their GLSL code within a custom main() function we provide which is GLSL-dialect agnostic. Therefore, user generated content didn't include references to gl_FragColor or gl_FragCoord, making our job easy - we simply replaced the true main() function that decorates user's code with a GLSL 330 version.
- We replaced all texture2D{GradEXT,LodEXT}() calls with texture{Grad,Lod}. This was an automatized problem and worked most of the time (see below for a few problems we had though)

WebGL 2.0 Transition

- Had to use InternalFormat for texture creation, which was natural for since since we are native GL programmers. The replacements are easy enough.

WebGL 2.0 Transition

- Being able to guarantee that textureLOD existed was a very awaited feature. We modified thousands of shaders to use this flavor of texture lookups for Noise, Voronoi and other LUT based primitive implementation that used texture() before and were causing long GLSL compile times in ANGLE based browsers. With textureLOD the code prevented the compiler from unrolling the loops looking for derivatives/gradients that you otherwise need for texture filtering. Luckily, most shaders utilized a couple of Noise implementations that were copy&pasted across shaders (as expected from our sharing encouraged community) and a bulk replace got the grand majority of the shaders in the database to compile faster!

WebGL 2.0 Issues

We did have however some issues:

- Many shaders used words like "sample" and "smooth" which no longer are allowed in WebGL2 for they are reserved. Had to change many shaders by hand.
- UNPACK_COLORSPACE_CONVERSION_WEBGL bit us, and LUT based shaders broke. It seems WebGL2 applies color transform to our pure-data look up tables and broke all noise based shaders. We had to force a disable.

WebGL 2.0 Issues

- Angle for WebGL 1 was creating float32 textures under the hood even when requesting float16 textures. That made some shaders unknowingly use this extra precision and then break when we got WebGL 2 and it implemented true float16 textures. We switched to requesting float32 to the API to solve the issue, although it was not the ideal solution.

WebGL 2.0 Issues

- We still got a few shaders that called texture2D() calls, now texture(), in the middle of loops. WebGL 1 seems to handle these more or less gracefully, albeit long compile times for unrolling, but WebGL 2 implementation using ANGLE seems to call the GLSL to HLSL compiler with a different set of flags that crash the browser more relatively often. These are a small percentage of the shaders in the database, and while not ideal or elegant, it's not a deal breaker.

New Features

Thanks to WebGL 2 now we have:

- Bit operations are a big deal in terms of having the procedural (non LUT based) Noise generators consistent across platforms. Before, users relied on aliasing through fract(sin(BigNumber*x)) to produce pseudo random numbers. Unfortunately, even for small BigNumber-s different platforms (mobile vs PC Windows vs PC Linux vs Mac) would produce different results and hence procedural terrains, clouds and many other Shadertoy shaders looked different in different devices. Exact bit operations has allowed for consistent content.

New Features

- accessing local arrays has been crucial on the implementation of stack based shaders who were impossible in WebGL 1 or had to be emulated with long sequences of conditional branches, which resulted in super slow shaders. Now, things are elegant and fast: https://www.shadertoy.com/view/Xds3zM
- texelFetch() greately simplified shaders that use (render to) textures to store game or simulation state. See this amazing all GLSL game: https://www.shadertoy.com/view/Xs2fWD
- integer support. Sprite/bitmap or ROM based shaders greately benefited from it. Check this out
- trunc, determinant, transpose, etc. Big win for novice users that didn't want to

Upcoming Features

Thanks to WebGL 2 we are soon going to have:

- Render to texture is guaranteed, and we'll use it. Great for deferred shaders velocity vectors, fluids syms, etc.
- 3D textures is going to be BIG soon. Users are faking it with 2D slices, but size is limited. Fast raymarching, voxel conetracing, cached irradiance, etc will be possible and we'll see it soon.
- integer textures opens the door to custom data structures too, if not more convenient game/simulation state representation.