Preparing A Technique For WebGPU Code Gen

Ideally, you wouldn't have to do anything special in your Gigi technique before generating code for WebGPU. Due to WebGPU limitations, you do often need to make a couple of changes, but fortunately, they are usually pretty minimal.

Setting Primary Output

When you launch the webgpu generated code, it will show the user the texture that is first in this list: Exported textures sorted by name, imported textures sorted by name, and internal textures sorted by name.

To control what the user sees, you can set the "Primary Output" in the editor's left panel.

Reading Read/Write (UAV) textures

Wherever you read from a read/write (UAV) texture, instead of reading directly, like:

```
float4 value = MyTexture[px].rgba;
```

You need to use a special token:

```
float4 value = /*$(RWTextureR:MyTexture)*/[px].rgba;
```

WebGPU is extremely limited in the texture formats it can read/write to, but the number of formats it can read only or write only is larger. Gigi WebGPU code generation will automatically split read/write texture accesses into read-only and write-only access, passing in a copy of the resource to the read-only access pin, and the original resource is passed to the write-only pin.

The RWTextureR token allows Gigi to make that read happen from the read-only resource.

Note that this can change the behavior of your technique if you expect the writes from a shader invocation to be visible to reads of that same shader invocation.

Also note: this problem with WebGPU can go away when the extension to get more read/write formats becomes available. https://github.com/gpuweb/gpuweb/pull/5160

Allowing Atomics

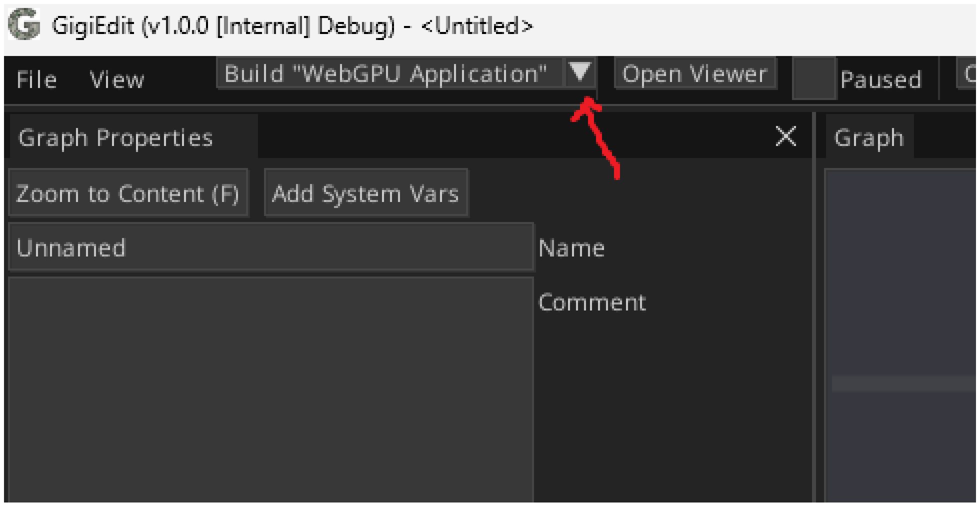
WebGPU allows atomics on buffers, but not textures, and for reasons relating to WebGPU and Slang, you have to opt into allowing atomics within Gigi. Within the shader definition in the Gigi editor, where you specify what resources a shader wants, you need to mark the "Allow Atomic Ops" checkbox to allow atomics. This could be solved within Slang, but someone would need to implement it to make hlsl atomic intrinsics work within Slang for the WGSL target.

Setting Imported Resources

Currently, imported resources are set up by writing JavaScript code within the index.html or index.js files. See the webgpu unit tests in _GeneratedCode for details of how to do this. Since this takes more than zero effort, I've found it useful to avoid imported resources when I can and instead rely on the Image2D tags, etc., that let you load textures from within shaders.

Generating Code and Running The Technique Locally

In the editor, there is a build button with a drop-down arrow to the right. Change that to "WebGPU Application" and then click the button to generate the code.



By default, it will generate the code in an out/WebGPU/ subfolder next to your .gg file.

You can run "Web_Start.bat" within that subfolder to start a local web server to host this folder using Python 3. Then, you can run "Web_Open.bat" to open your technique in a browser.

You can see which browsers support WebGPU here: https://caniuse.com/webgpu

WebGPU currently works in Chrome, but not Firefox, for example.

Enabling WebGPU Extensions (RequiredLimits / RequiredFeatures)

WebGPU is meant to work on low end hardware by default, but allows extended functionality when creating the device, by specifying extra features through the requiredFeatures array, and relaxing limitations through the requiredLimits dictionary.

Gigi allows you to opt into these things in the editor, in the Graph Properties window, under Settings -> WebGPU.

Features: https://developer.mozilla.org/en-US/docs/Web/API/GPUSupportedFeatures

Limits: https://developer.mozilla.org/en-US/docs/Web/API/GPUSupportedLimits

You can see the capabilities of your browser and machine at: https://webgpureport.org/

FAQ

The errors you see in the browser when things fail are usually not very helpful because several layers are involved: Gigi, Slang, and WebGPU. If you encounter problems, feel free to open an issue on GitHub or ask in the Discord server. This section will grow as people encounter problems not mentioned above.

How To Make BVHs

You can make BVHs to accelerate raytracing by drag/dropping a model into the viewer, clicking on the buffer resource for that model, and clicking the “Save As .BVH” button. When supplying a technique with an imported resource for raytracing geometry, there is a Boolean variable that you set to true, to let the technique know you are giving it a BVH, and not raw geometry. More information is available in the GigiViewerDX12_Documentation file.