**Shader Compilation in Unity 4.5**

[By](https://blog.unity.com/author/cap-aras)

[](https://blog.unity.com/author/cap-aras)

[Aras Pranckevičius](https://blog.unity.com/author/cap-aras)

May 6, 2014 in [Technology](https://blog.unity.com/technology) | 11 min. read



Expand

Topics covered

[Graphics](https://blog.unity.com/topic/graphics)[Shader tools](https://blog.unity.com/topic/shader-tools)[Unity 4](https://blog.unity.com/topic/unity-4-0)

Share

A story in several parts. 1) how shader compilation is done in upcoming Unity 4.5; and 2) how it was developed. First one is probably interesting to Unity users; whereas second one for the ones curious on *how* we work and develop stuff.

Short summary: Unity 4.5 will have a *“wow, many shaders, much fast”* shader importing and better error reporting.

**Current state (Unity &lt;=4.3)**

When you create a new shader file (.shader) in Unity or edit existing one, we launch a “shader importer”. Just like for any other changed asset. That shader importer does some parsing, and then compiles the *whole* shader into *all* platform backends we support.

Typically when you create a simple [surface shader](https://docs.unity3d.com/Documentation/Components/SL-SurfaceShaders.html), it internally expands into 50 or so internal shader variants (classic “preprocessor driven uber-shader” approach). And typically there 7 or so platform backends to compile into *(d3d9, d3d11, opengl, gles, gles3, d3d11\_9x, flash – more if you have console licenses)*. This means, *each* time you change anything in the shader, a couple hundred shaders are being compiled. And all that assuming you have a fairly simple shader - if you throw in some [multi\_compile directives](https://docs.unity3d.com/Documentation/Components/SL-MultipleProgramVariants.html), you’ll be looking at thousands or tens of thousands shaders being compiled. Each. And. Every. Time.

Does it make sense to do that? Not really.

*Like most of “why are we doing this?” situations, this one also evolved organically, and can be explained with “it sounded like a good idea at the time” and “it does not fix itself unless someone works on it”.*

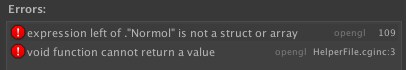
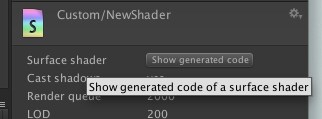
*A long time ago, Unity only had one or two shader platform backends (opengl and d3d9). And the amount of shader variants people were doing was much lower. With time, we got both more backends, and more variants; and it became very apparent that someone needs to solve this problem.*

In addition to the above, there were other problems with shader compilation, for example:

* Errors in shaders were reported, well, “in a funny way”. Sometimes the line numbers did not make any sense – which is quite confusing.
* Debugging generated surface shader code involved quite some voodoo tricks (#pragma debug etc.).
* Shader importer tried to multi-thread compilation of these hundreds of shaders, but some backend compilers (Cg) have internal global mutexes and do not parallelize well.
* Shader importer process was running out of memory for *really large* multi\_compile variant counts.

So we’re changing how shader importing works in Unity 4.5. The rest of this post will be *mostly* dumps of our internal wiki pages.

**Shader importing in Unity 4.5**

* *No runtime/platforms changes compared to 4.3/4.5 – all changes are editor only*.
* *No shader functionality changes compared to 4.3/4.5*.
* **Shader importing is *much* faster**; especially complex surface shaders (Marmoset Skyshop etc.).
  + Reimporting all shaders in graphics tests project: 3 minutes with 4.3, 15 seconds with this.
* **[](https://blog-api.unity.com/sites/default/files/2014/05/shaders-errors.png)Errors in shaders are reported on correct lines**; errors in shader include (.cginc) files are reported with the filename & line number correctly.
  + Was mostly “completely broken” before, especially when include files came into play.
  + On d3d11 backend we were reporting error *column* as the line, hah. At some point during d3dcompiler DLL upgrade it changed error printing syntax and we were parsing it wrong. Now added unit tests so hopefully it will never break again.
* **[](https://blog-api.unity.com/sites/default/files/2014/05/shaders-surface.png)Surface shader debugging workflow is much better**.
  + No more “add #pragma debug, open compiled shader, remove tons of assembly” nonsense. Just one button in inspector, “Show generated code”.
  + Generated surface shader code has some comments and better indentation. It is actually readable code now!
* Shader inspector improvements:
  + Errors list has scrollview when it’s long; can double click on errors to open correct file/line; can copy error text via context click menu; each error clearly indicates which platform it happened for.
  + Investigating compiled shader is saner. One button to show compiled results for currently active platform; another button to show for all platforms.
* Misc bugfixes
  + Fixed multi\_compile preprocessor directives in surface shaders sometimes producing very unexpected results.
  + UTF8 BOM markers in .shader or .cginc files don’t produce errors.
  + Shader include files can be at non-ASCII folders and filenames.

**Overview of how it works**

* Instead of compiling all shader variants for all possible platforms at import time:
  + Only do minimal processing of the shader (surface shader generation etc.).
  + Actually compile the shader variants only when needed.
  + Instead of typical work of compiling 100-1000 internal shaders at import time, this usually ends up compiling just a handful.
* At player build time, compile all the shader variants for that target platform
  + Cache identical shaders under Library/ShaderCache.
  + So at player build time, only not-yet-ever-compiled shaders are compiled; and always only for the platforms that need them. If you never ever use Flash, for example, then none of shaders will be compiled for Flash *(as opposed to 4.3, where all shaders are compiled to all platforms, even if you never ever need them)*.
* Shader compiler (CgBatch) changes from being invoked for each shader import, into being run as a “service process”
  + Inter-process communication between compiler process & Unity; using same infrastructure as for VersionControl plugins integration.
  + At player build time, go wide and use all CPU cores to do shader compilation. Old compiler tried to internally multithread, but couldn’t due to some platforms not being thread-safe. Now, we just launch one compiler process per core and they can go fully parallel.
  + Helps with out-of-memory crashes as well, since shader compiler process never needs to hold bazillion of shader variants in memory all at once - what it sees is one variant at a time.

**How it was developed**

This was mostly a one-or-two person effort, and developed in several “sprints”. For this one we used our internal wiki for detailed task planning (Confluence “task lists”), but we could have just as well use Trello or something similar. Overall this was *probably* around two months of actual work – but spread out during much longer time. Initial sprint started in 2013 March, and landed in a “we think we can ship this tomorrow” state to 4.5 codebase just in time for 1st alpha build (2013 October). Minor tweaks and fixes were done during 4.5 alpha & beta period. *Should ship anyday now, fingers crossed!*

Surprisingly (or perhaps not), largest piece of work was around “how do you report errors in shaders?” area. Since now shader variants are imported only *on demand*, that means some errors can be discovered only “some time after initial import”. This is a by-design change, however - as the previous approach of “let’s compile all possible variants for all possible platforms” clearly does not scale in terms of iteration time. However, this “shader seemed like it did not have any errors, but whoops now it has” is clearly a potential downside. Oh well; as with almost everything there are upsides & downsides.

Most of development was done on a Unity 4.3-based branch, and after something was working we were sending off custom “4.3 + new shader importer” builds to the beta testing group. We were doing this before any 4.5 alpha even started to get early feedback. Perhaps the nicest feedback I ever got:

*I’ve now used the build for about a week and I’m completely blown away with how it has changed how I work with shaders.*

*I can try out things way quicker.  
I am no longer scared of making a typo in an include file.  
These two combine into making me play around a LOT more when working.  
Because of this I found out how to do fake HDR with filmic tonemapping [on my mobile target].*

*The thought of going back to regular beta without this [shader compiler] really scares me.*

Anyhoo, here’s a dump of tasks from our wiki (all of them had little checkboxes that we’d tick off when done). As usual, “it basically works and is awesome!” was achieved after first week of work (1st sprint). What was left after that was “fix all the TODOs, do all the boring remaining work” etc.

**2013 March Sprint:**

* Make CgBatch a DLL
  + Run unit tests
  + Import shaders from DLL
  + Don’t use temp files all over the place
* Shader importer changes
  + Change surface shader part to only generate source code and not do any compilation
  + Make a “Open surface compiler output” button
  + At import time, do surface shader generation & cache the result (serialize in Shader, editor only)
  + Also process all CGINCLUDE blocks and actually do #includes at import time, and cache the result (after this, left with CGPROGRAM blocks, with no #include statements)
  + ShaderLab::Pass needs to know it will have yet-uncompiled programs inside, and able to find appropriate CGPROGRAM block:
    - Add syntax to shaderlab, something like Pass { GpuProgramID int }
    - Make CgBatch not do any compilation, just extract CGPROGRAM blocks, assign IDs to them, and replace them with “GpuProgramID xxx”
    - “cache the result” as editor-only data in shader: map of snippet ID -> CGPROGRAM block text
  + CgBatch, add function to compile one shader variant (cg program block source + platform + keywords in, bytecode + errors out)
  + Remove all #include handling from actual shader compilers in CgBatch
  + Change output of single shader compilation to not be in shaderlab program/subprogram/bindings syntax, but to produce data directly. Shader code as a string, some virtual interface that would report all uniforms/textures/… for the reflection data.
* Compile shaders on demand
  + Data file format for gpu programs & their params
  + ShaderLab Pass has map: m\_GpuProgramLookup (keywords -> GPUProgram).
  + GetMatchingSubProgram:
    - return one from m\_GpuProgramLookup if found. Get from cache if found
    - Compile program snippet if not found
    - Write into cache

**2013 July Sprint:**

* Pull and merge last 3 months of trunk
* Player build pipeline
  + When building player/bundle, compile all shader snippets and include them
  + exclude\_renderers/include\_renderers, trickle down to shader snippet data
  + Do that properly when building for a “no target” (everything in) platforms
    - Snippets are saved in built-in resource files (needed? not?)
  + Make building built-in resource files work
    - DX11 9.x shaders aren’t included
    - Make building editor resource file work
  + Multithread the “missing combinations” compilation while building the player.
    - Ensure thread safety in snippet cache
* Report errors sensibly
* Misc
  + Each shader snippet needs to know keyword permutation possibly needed: CgBatch extracts that, serialized in snippet (like vector< vector >)
  + Fix GLSLPROGRAM snippets
  + Separate “compiler version” from “cgbatch version”; embed compiler version into snippet data & hash
  + Fix UsePass

**2013 August Sprint:**

* Move to a 4.3-based branch
* Gfx test failures
  + Metro, failing shadow related tests
  + Flash, failing custom lightmap function test
* Error reporting: Figure out how to deal with late-discovered errors. If there’s bad syntax, typo etc.; effectively shader is “broken”. If a backend shader compiler reports an error:
  + Return pink “error shader” for all programs ­ i.e. if any of vertex/pixel/… had an error, we need to use the pink shaders for all of them.
  + Log the error to console.
  + Add error to the shader, so it’s displayed in the editor. Can’t serialize shader at that time, so add shaders to some database under Library (guid­>errors).
    - SQLite database with shader GUID -> set of errors.
  + Add shader to list of “shaders with errors”; after rendering loop is done go over them and make them use pink error shader. (Effectively this does not change current (4.2) behavior: if you have a syntax error, shader is pink).
* Misc
  + Fix shader Fallback when it pulls in shader snippets
  + “Mesh components required by shader” part at build time - need to figure them out! Problem; needs to compile the variants to even know it.
  + Better #include processing, now includes same files multiple times
* Make CgBatch again into an executable (for future 64 bit mac…)
  + Adapt ExternalProcess for all communication
  + Make unit tests work again
  + Remove all JobScheduler/Mutex stuff from CgBatch; spawn multiple processes instead
  + Feels like is leaking memory, have to check
* Shader Inspector
  + Only show “open surface shader” button for surface shaders
  + “open compiled shader” is useless now, doesn’t display shader asm. Need to redo it somehow.

**2013 September Sprint:**

* Make ready for 4.5 trunk
  + Merge with current trunk
  + Make TeamCity green
  + Land to trunk!
* Make 4.3-based TeamCity green
  + Build Builtin Resources, fails with shader compiler RPC errors GL-only gfx test failures (CgProps test)
  + GLSLPROGRAM preprocessing broken, add tests
  + Mobile gfx test failures in ToonyColors
* Error reporting and #include handling
  + Fixing line number reporting once and for all, with tests.
  + Report errors on correct .cginc files and correct lines on them
  + Solve multiple includes & preprocessor affecting includes this way: at snippet extraction time, do not do include processing! Just hash include contents and feed that into the snippet hash.
  + UTF8 BOM in included files confusing some compilers
  + Unicode paths to files confusing some compilers
  + After shader import, immediately compile at least one variant, so that any stupid errors are caught & displayed immediately.
* Misc
  + Make flags like “does this shader support shadows?” work with new gpu programs coming in
  + Check up case 550197
  + multi\_compile vs. surface shaders, fix that
* Shader Inspector
  + Better display of errors (lines & locations)
  + Button to “exhaustively check shader” - compiles all variants / platforms.
  + Shader snippet / total size stats

**What’s next?**

Some more in shader compilation land will go into Unity 5.0 and 5.x. Outline of our another wiki page describing 5.x related work:

* 4.5 fixes “compiling shaders is slow” problem.
* Need to fix “New standard shader produces very large shader files” *(due to lots of variants - 5000 variants, 100MB)* problem.
* Need to fix “how to do shader LOD with new standard shader” problem.

[By](https://blog.unity.com/author/cap-aras)

[](https://blog.unity.com/author/cap-aras)

[Aras Pranckevičius](https://blog.unity.com/author/cap-aras)

May 6, 2014 in [Technology](https://blog.unity.com/technology) | 11 min. read