Subject: : AmigaOS4
Topic: : GL4ES: another OpenGL over OpenGLES2 emulation - some tech. info and porting progress
Re: GL4ES: another OpenGL over OpenGLES2 emulation - some tech. info and porting progress
Author: : Daytona675x
Date: : 2019/7/26 7:46:09
URL:

@Hans
Quote:

> it wouldn't surprise me if compiler time is dominated by the processing that takes place later.

By a factor of up to 20?! You're kidding. Unless it's written in AmigaBASIC or so it should have a similarly low footprint as the GLSL/preprocessing-compilation-part.
Okay, an alternative reason for that extreme slowness here would be your disabled optimizer. That explains it, of course.

Quote:

> Well, I looked extensively for a bug in my code.

Then, with almost absolute certainty, you simply didn't look extensively enough. I definitely second kas1e on this:

Quote:

> kas1e: And from my past expirience : if enabled optimisation (expectually -O3) show some bug, which didn't happens without optimisation, it mean that code have bug, and optimisation only help to make it visibly, so that one for sure need to be taken care of.

Your approach "I got issues with boost when optimizer turned on, so I globally disable the optimizer" is certainly not a good way of doing things.
Keep in mind that in 99% of all such situations the reason for failure is the programmer, not boost, not the stl, not the compiler (that's especially true for "it works if optimizer disabled"). Use the optimizer as a tool to validate your code, not the other way around like "disable the optimizer to hide my bugs". Failures in high optimizer levels are (almost always) indicators for subtle self-made bugs that otherwise simply are less likely to show symptoms, which is most likely what you achieved by turning off optimizations - but you didn't fix anything.

Quote:

> Using std::shared_ptr<> would be preferred.

Is there a reason why you didn't do this replacement a long time ago, especially if you believe that the problem is because of a boost-pointer-casting issue? Should be very quick and unproblematic operation.

Quote:

> All I could see, is that it was screwing up the reference counting when casting between types (meaning I end up with an extra shared_ptr, but with a different type).

Out out curiosity: how does this code snippet look like? And what means "screwing up ref counting"? One too high? One too low? Two independent ref-counters all of a sudden? Random value?

However, if you just cannot find and fix the bugs and disabling the optimizer at least improves stability to the current state then I strongly suggest to try to keep a high optimizer level for some critical units or functions. Or, maybe even easier: use -O3 and then manually remove single potentially critical optimizations again via -no-XXX etc. This could also help you to get an idea of the type of your bug. Or the other way around: use the low -O you use now and manually add the respective optimization flags until it becomes unstable. Everything is better than to fully disable optimizations!

And if at the end of the day it really should turn out that you tapped into a true boost or compiler bug, then concentrating on changing the critical construct should be the chosen approach, not global optimizer deactivation.

Quote:

> That's not so easy because more needs to be saved than just the raw shader code. There are various register settings that need to be preserved. So, I'd have to come up with another file format to save the binary.

It's an every-day serialization problem, it's certainly much less harder than you think at first glance. But yes, it isn't something with high priority. Especially if we could get some optimizer, please.