Doomsday Engine - Feature #1678

Optimize stereoscopic pixel formats

2013-12-09 11:12 - skyjake

Status:	Rejected	**Start date:**	
Priority:	Normal	**% Done:**	0%
Assignee:			
Category:	Enhancement		
Target version:			

Description

Support for a stereoscopic GL pixel format, used for instance with NVIDIA 3D Vision. This is implemented currently as VR mode 13.

The appropriate stereo format likely needs to be applied in GLFramebuffer (FBO) and/or the renderbuffers/textures attached to it (?).

Related issues:

Related to Feature #1636: Support for Oculus Rift	**Closed**	**2013-10-23**
Related to Feature #1680: Quad-buffered GL framebuffer	**Closed**	**2013-12-16**
Related to Feature #2164: Interlaced 3D mode	**Closed**	**2016-07-11**

History

#1 - 2013-12-10 17:10 - skyjake

If it is required that rendering occurs in the system framebuffer (FBO 0), we could — in VR mode 13 specifically — render the frame in depth-only mode separately for the lens flares. This won't help with postfx, though.

#2 - 2013-12-11 19:00 - cmbruns

skyjake wrote:

> If it is required that rendering occurs in the system framebuffer (FBO 0), we could — in VR mode 13 specifically — render the frame in depth-only mode separately for the lens flares. This won't help with postfx, though.

Sorry for this long winded comment. I want to get my ideas in writing while they are still fresh.

I think all stereo 3D composition could be accomplished in the system framebuffer, and that "postfx" could be performed pre-stereo-composition. This ordering is probably necessary for mode 13, and I suspect that all 3D modes could be handled the same way.

I've been thinking a lot about offscreen buffers and stereoscopic rendering. I have not looked carefully at what you have done in the lensflare branch. But I want to make some architectural comments while you still have your recent architectural considerations freshly in mind.

It sounds like you are creating an architecture for performing a series of rendering passes, using a collection of offscreen buffers. A natural friction between modularity and efficiency can arise, unless a judicious architecture is chosen. I suspect that you and I would both prefer a solution with maximum efficiency, using the most modular architecture that can support that efficiency. For maximum efficiency, it would be nice to minimize 1) the number of rendering passes, and 2) the size and number of offscreen buffers allocated. The current stereoscopic format implementations do not necessarily maximize efficiency this way. I probably won't have time to refactor the 3D modes for this release; but I do want to eventually adopt multipass-rendering-friendly behavior for all 3D modes.

Many multipass rendering pipelines make do with exactly two offscreen buffers, using the ping-pong approach: render to buffer B, using the contents of buffer A; then render to buffer A, using the contents of B, etc. I have even heard that is possible to ping pong between one offscreen buffer and a texture bound to the primary back buffer. But I also heard that this is against the OpenGL spec, so I would be queasy about taking this latter approach. For the ping-pong approach to be effective, most rendering passes would need to use the same framebuffer size. I believe the stereo 3D modes, especially Oculus Rift, could be modified to better support this same-size constraint (see below).

For a given multipass rendering pipeline, it would be most efficient for the final pass to render directly to the FBO-zero back buffer. Otherwise it might cost an extra rendering pass. Thus you might want some sort of manager class that is aware of every pass that is to be performed, and shunts the final pass to the primary framebuffer. This manager object might be well placed to either support or manage stereoscopic composition of left and right eye views.

I think all of the stereo 3D techniques could be composed in the final rendering pass to the primary frame buffer, including mode 13 (hardware stereo). The general flow would be thus:

1. Render all passes for the left eye view, with the final pass populating (some aspect of) the FBO-zero framebuffer.
2. Render all passes for the right eye view, with the final pass populating (some other aspect of) the FBO-zero framebuffer.
   In particular, the Oculus Rift mode should be refactored to perform all passes, including warping, separately for the left and right eye views.

Each stereo 3D method has a particular natural offscreen framebuffer size, relative to the displayed window/screen size. All rendering passes would

ordinarily use this natural size.

- 0.5X width: side-by-side modes, and column-interleaved mode
- 0.5X height: top/bottom mode, and row-interleaved mode
- 1.5X height; 0.5*1.5X width: Oculus Rift mode
- sqrt(2)*each dimension: checker interlaced mode
- unmodified size: all other modes (including mode 13)

3D mode details

Anaglyph modes

The anaglyph modes use glColorMask() for stereo 3D composition. This color masking could be included as an additional aspect to whatever the final rendering pass happens to be. Thus, it might be nice to include a concept of wrapping a rendering pass object inside a more complex rendering pass.

Interleaved/interlaced modes

I have not implemented these yet, in part because I encountered some trouble with the offscreen render buffer mechanics. These modes require either a stencil-based approach, or a shader-based rearrangement of pixels. I'm inclined toward the shader-based approach, which should have superior image quality, and probably superior speed performance, at the expense of an additional rendering pass.

Hardware stereo (mode 13)

Probably requires rendering to BACK_LEFT and BACK_RIGHT buffers of a specially constructed stereo-enabled primary framebuffer. This primary framebuffer probably supports both double buffering and stereo buffering, while offscreen buffers need not have those extra buffers. It's likely that offscreen buffers cannot support stereo buffering.

Oculus Rift

Oculus Rift is the most complex case. The warp shader definitely requires an additional rendering pass, to a different sized target. That target should probably be the primary framebuffer. 3D rendering passes should go use one or two enlarged offscreen buffers, depending on the number of rendering passes. It might be worth considering using one or two smaller 4:5 aspect offscreen buffers for the menu/taskbar/status/weapon items.

#3 - 2013-12-12 21:03 - skyjake

Thanks for the detailed info.

I believe that with minor modifications, the existing classes should be adequate for us. At the very least, GLFramebuffer needs to have an extra property that determines whether BACK_LEFT or BACK_RIGHT should be used.

Note that GLState now handles glColorMask, too, so it can be freely used in the rendering.

#4 - 2013-12-16 09:33 - skyjake

- Assignee set to skyjake

- Target version set to 1.13

#5 - 2013-12-16 09:34 - skyjake

- Assignee deleted (skyjake)

- Target version deleted (1.13)

#6 - 2013-12-23 12:29 - skyjake

- Subject changed from Stereoscopic pixel formats to Optimize stereoscopic pixel formats

- Target version set to 1.13

Repurposing this issue to be more about optimizing the memory use and performance of 3D stereo rendering.

#7 - 2013-12-23 12:30 - skyjake

- Target version deleted (1.13)

#8 - 2013-12-23 12:30 - skyjake

- Tags changed from Graphics, GL2, OpenGL to Graphics, GL2, OpenGL, Performance

#9 - 2013-12-23 12:31 - skyjake

- Priority changed from High to Normal

#10 - 2013-12-23 12:32 - skyjake

- % Done changed from 100 to 0

#11 - 2016-07-11 09:05 - skyjake

- Related to Feature #2164: Interlaced 3D mode added

#12 - 2019-11-29 19:23 - skyjake

- Status changed from New to Rejected