Doom 3 → Dante

Performance on Mesa (i965)
(Not a Demo!)
Quick overview of Doom 3

- GPLv3+ (with additional terms) on November 22, 2011.
  - Without “Carmack's Reverse” (aka depth fail) shadows.
    Date: Fri, 25 Nov 2011 01:32:56 +0200
    Subject: [PATCH v3 1/1] renderer: added support for Carmack's Reverse (depth fail) shadows.

- OpenGL 1.x + OpenGL extensions.
  - Some of which are requirements.

- X11 and GLX.

- 8 years old.

- ARB2 backend (best backend)
  - ARB_vertex_program && ARB_fragment_program.
  - Other backends available for older hardware.

- ARB_vertex_buffer_object used when available, otherwise fallback to virtual memory.
Quick overview of Dante

- OpenGL ES2.0
  - EGL
  - GLSL primary backend
    - ARB2 backend remains for debugging on the desktop; stubbed out when compiled for ES2.0
    - Carmack's Reverse (depth fail) added back
  - VBO requirement
  - ARBvp and ARBfp programs are not part of the GPLv3+ release
    - “Clean-room” programs written in GLSL
    - Phong (rather than Blinn-Phong) shading model.
      - More computationally expensive but delivers much more realistic rendering.
    - Optional Half-Lambert lighting (see example on next slide: Phong + Half-Lambert.)

- Support for Android...
  - You'd better have a high-end device!
  - “Some” bugs and missing features...
Lambert vs Half-Lambert
Half-Lambert Gone Wrong?
Optimization on Mesa

- Unfortunately no really great tools for Mesa performance analysis...
  - i965: intel_gpu_top: works like regular `top' –
    - No support for per-frame analysis,
    - No support for pretty graphs (unless you're into ASCII art.)
  - Useful for rough estimate of GPU load.
  - Basically unusable output for game devs who haven't read and understood Intel HW docs.
    - Game devs typically don't want to read low-level HW docs...

- So, what should we do to fix this for Mesa drivers?
  - Quick example of intel_gpu_top first...
<table>
<thead>
<tr>
<th>Task</th>
<th>Percent Busy</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAM</td>
<td>68%</td>
<td>Vert fetch: 0 (0/sec)</td>
</tr>
<tr>
<td>CS</td>
<td>37%</td>
<td>Prim fetch: 0 (0/sec)</td>
</tr>
<tr>
<td>PSD</td>
<td>32%</td>
<td>GS invocations: 33076780 (1617385/sec)</td>
</tr>
<tr>
<td>DAP</td>
<td>28%</td>
<td>GS prims: 0 (0/sec)</td>
</tr>
<tr>
<td>RCPFE</td>
<td>28%</td>
<td>CL invocations: 16538390 (808744/sec)</td>
</tr>
<tr>
<td>IZ</td>
<td>28%</td>
<td>CL prims: 11324693 (689777/sec)</td>
</tr>
<tr>
<td>RCPBE</td>
<td>28%</td>
<td>PS invocations: 11415570625 (347597257/sec)</td>
</tr>
<tr>
<td>RCC</td>
<td>28%</td>
<td>PS depth pass: 11132520857 (340957947/sec)</td>
</tr>
<tr>
<td>WMFE</td>
<td>28%</td>
<td></td>
</tr>
<tr>
<td>EU 30</td>
<td>26%</td>
<td></td>
</tr>
<tr>
<td>SVG</td>
<td>26%</td>
<td></td>
</tr>
<tr>
<td>EU 10</td>
<td>25%</td>
<td></td>
</tr>
<tr>
<td>EU 00</td>
<td>25%</td>
<td></td>
</tr>
<tr>
<td>HIZ</td>
<td>25%</td>
<td></td>
</tr>
<tr>
<td>EU 20</td>
<td>25%</td>
<td></td>
</tr>
<tr>
<td>TD</td>
<td>25%</td>
<td></td>
</tr>
<tr>
<td>SVRW</td>
<td>25%</td>
<td></td>
</tr>
<tr>
<td>IC 3</td>
<td>23%</td>
<td></td>
</tr>
<tr>
<td>WMBE</td>
<td>23%</td>
<td></td>
</tr>
<tr>
<td>IC 2</td>
<td>23%</td>
<td></td>
</tr>
<tr>
<td>IC 0</td>
<td>23%</td>
<td></td>
</tr>
<tr>
<td>IC 1</td>
<td>23%</td>
<td></td>
</tr>
<tr>
<td>EU 01</td>
<td>22%</td>
<td></td>
</tr>
</tbody>
</table>

Oliver McFadden
Better debugging/analysis tools!

- AMD's gDEBugger works on GNU/Linux, but only with AMD hardware and fglrx.
  - Older pre-AMD versions used to run with Mesa, but have problems with modern glibc.
  - Proprietary tool (both pre and post-AMD versions.)
  - Basically unusable for me...

- Nvidia, SGX, ... have similar tools for their proprietary drivers.

- We don't have any great tools for Mesa...
  - But we should!
Linux kernel and `perf' system...

- https://perf.wiki.kernel.org
  - Stumbled across this by accident while looking at CPU profiling.

`perf` provides rich generalized abstractions over hardware specific capabilities. Among others, it provides per task, per CPU and per-workload counters, sampling on top of these and source code event annotation.

- `perf stat`: obtain event counts
- `perf record`: record events for later reporting
- `perf report`: break down events by process, function, etc.
- `perf annotate`: annotate assembly or source code with event counts
- `perf top`: see live event count
Kernel `perf' system and Mesa

- Possibly create infrastructure in DRM and hook into `perf' sub-system?
- Needs some cooperation with userspace:
  - Mesa should indicate frame termination without causing a stall, e.g.
    - `OUT_BATCH(SCRATCH_REG_0, 0xDEADD00D);
  - Could be done at swap buffers or more intelligently with the `GL_GREMEDY_frame_terminator` extension (with application support.)

- Userspace debugger could read the data from kernel and generate pretty graphs, suggestions, etc.
  - Interactive GUI,
  - HTML report,
  - ASCII art. ; - )

- Very much hand waving at this point. No prototype implementation.
Mesa debug output

- Mesa drivers may be able to provide “hints” for the OpenGL application:

  ```c
  if (ctx->Scissor.Enabled)
      perf_debug("Failed to fast clear depth due to scissor being enabled.
                  Possible 5% performance win if avoided."
                      );
  ```

  - 20 dwords to change surface state (disable the scissor test.)
  - How to synchronize these with the data from kernel `perf' system?
  - Possibly with a carefully managed frame counter?

- Userspace debugger could match frame counter of data fetched from `perf' system and strings fetched from `ARB_debug_output'.
  - Currently `perf_debug()` does not output to `ARB_debug_output`!

- `ARB_debug_output` works as long as the debugger and OpenGL application are in the same context...
  - But we probably do not want such a solution; it's ugly and we lose any benefits of having the debugger as a separate process.
  - Not quite sure how to handle Mesa debug output with the debugger in a separate process... Suggestions?
GLX vs EGL

- Dante (OpenGL ES2.0, X11 (XCreateWindow et al), EGL):
  - +timedemo demo1
  - vblank_mode=0
  2148 frames rendered in 64.6 seconds = 33.3 fps

  MessageBox: Time Demo Results - 2148 frames rendered in 64.6 seconds = \textbf{33.3 fps}

- Dante (OpenGL ES2.0, X11 (XCreateWindow et al), GLX):
  - +timedemo demo1
  - vbank_mode=0
  2148 frames rendered in 47.2 seconds = 45.5 fps

  MessageBox: Time Demo Results - 2148 frames rendered in 47.2 seconds = \textbf{45.5 fps}

- Mesa appears to ignore \texttt{vblank\_mode} in the EGL code...
  
  ```c
  src/egl/drivers/dri2/platform_x11.c: struct dri2_egl_surface *dri2_surf =
  src/egl/drivers/dri2/platform_x11.c- dri2_egl_surface(surf);
  src/egl/drivers/dri2/platform_x11.c- #endif
  src/egl/drivers/dri2/platform_x11.c-
  src/egl/drivers/dri2/platform_x11.c: /* XXX Check vblank\_mode here? */
  src/egl/drivers/dri2/platform_x11.c-
  src/egl/drivers/dri2/platform_x11.c- if (interval > surf->Config->MaxSwapInterval)
  src/egl/drivers/dri2/platform_x11.c- interval = surf->Config->MaxSwapInterval;
  ```
Conclusion

- Bottom line: We need better performance analysis tools.

- Intel has done work on Mesa/i965 optimization with Valve Software for their “Left 4 Dead 2” game:
  - Eric Anholt, Ian Romanick, and Ken Graunke at Valve’s headquarters in person.
  - Possible for a large game development studio,
  - Not possible for indie game developers.

- Performance tools will never be as good as experts in person, but can still be very useful.
Questions? / Comments?