

The Rebuilt Linux Desktop

Keith Packard

keith.packard@intel.com



The Rebuilt Linux Desktop

- Recap of last year's LCA presentation
- A tour of the 2008 developments
- Unfinished business



Where were we last year

- No composited 3D
- No monitor auto-plug
- No 2D/3D/Media shared objects
- No kernel mode setting
- No kernel-based 2D drawing



What was holding us back?



No memory manager.



Kernel memory management

- Persistent objects
 - Preserved across context switch
 - No backup copy required
- Global names
 - Sharing across applications
 - Sharing across APIs
- Pageable contents
 - “Infinite” storage
 - No user-space GPU/CPU swapping



GEM: The Graphics Execution Manager


- Kernel memory manager
- Integrated into Linux 2.6.28
- Initial development done in April



GEM Process

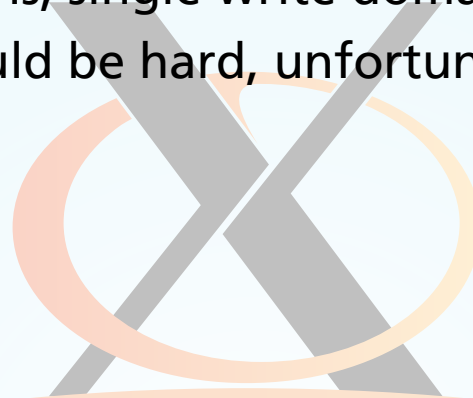


Calendar of GEM Events

- March: Eric and Keith agree to try a new approach
 - April: First rough implementation produced
 - May 13: Announced on LKML
 - June/July: GEM becomes usable
 - July 30: GEM development moved from mesa to kernel
 - August/Sept: 3D/2D user mode stabilizes
 - October 17: GEM merged to mainline
 - December 24: 2.6.28 released
- 

GEM Architecture In One Slide

- Use SHMFS for object allocation
 - Unpinned objects can be paged out
 - SHMFS handles read/write/mmap
 - Sharing code is almost always good
- Focus on cache management using “memory domains”
 - Moving pages between cache domains is expensive
 - Applications must know what caches each operation uses
 - Multiple read domains, single write domain
 - We thought this would be hard, unfortunately we were right



Ok, One More GEM Slide

- Refuse to expose physical object addresses to user space
 - Kernel completely manages graphics aperture
 - User space provides “relocations” to reference objects
 - Ensures that the full aperture is always available
- Expose batch buffers as the fundamental scheduling unit
 - It's all about execution management
 - Objects are mapped to the GTT for execution
 - Relocations are evaluated
 - Objects remain mapped until execution finishes



Developing GEM in the Open

- Less than two months from start to announce
- Intense feedback was critical to final architecture
- Substantial core kernel changes adopted for GEM
 - SHMFS export for allocation (shmem_file_setup)
 - io_mapping infrastructure added



GEM Nuggets

- Flushing CPU caches is slow
 - Map pages to GTT yields WC mapping
 - Must still cflush the first time
- Non-atomic page mapping is slow
 - `kmap_atomic_prot_pfn` to the rescue
 - But, that's not "legal" on IO region
 - Invent `io_mapping` API



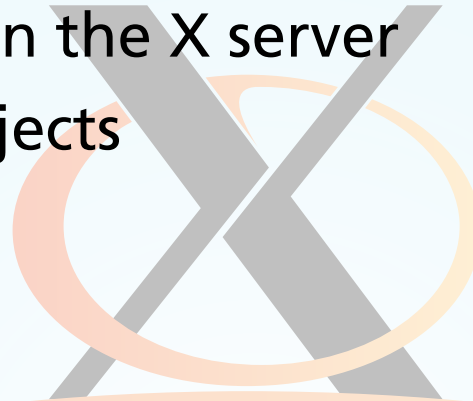
So, what does GEM replace?

- Balkanized memory
 - Static cursors, overlays, 2D state, etc.
 - Fixed-size, shared front/back/depth/stencil
 - X pixmaps
 - Shared textures, invalidated on task switch
- Pages allocated and bound to GTT at X startup
- Lots of data copying through WC aperture



DRI Re-Invented

- DRI2 is a new X extension
- Replaces DRI extension (oddly enough)
- Three basic requests
 - Connect
 - GetBuffers
 - CopyRegion
- No shared memory area
- Buffer swaps done in the X server
- Buffers are GEM objects



Differences between DRI and DRI2

- DRI used shared front/back/depth/stencil
- Small shared texture space, no changes saved
- No support for FBOs or other mutable objects
- Cannot perform in-place texture-from-pixmap



KMS integrated

- Depends on GEM
- Full mode setting API to user space
- Provides fbdev interface too
- Non-root X server has been demonstrated
- Wayland window system demonstrates fast user switching



Multi-master DRM

- Under DRI 1, only one user got DRM
- Everyone else got unaccelerated 3D
- With DRI2 and KMS, no more privileged user
- Also allows for non-X GPU usage, including GPGPU



UXA: GEM-centric 2D

- Most of EXA deals with migration
- GEM makes migration unnecessary
- UXA shares EXA acceleration code
- But, without migration or explicit syncs



Whither UXA?

- XAA accelerates lots of core X
- EXA/UXA do only fills, blts, composite
- Switching between HW and SW is expensive
- What else to accelerate?
 - Core text (until emacs ships)
 - 0-width lines
- Move UXA back to the X server



Stuff is still broken



Tearing: The Problem

- This seems simple.
 - Draw stuff off-screen
 - Wait for retrace
 - Copy to screen
- But
 - We don't want to stop the GPU



Tearing: A Work in Progress

- Goal:
 - Queue copy to kernel
 - Executed in response to interrupt
 - Never stall the hardware
- Issues:
 - Clip lists change
 - Switching HW contexts



Tearing: Some Solutions

- Clip list changes
 - Invalidate pending blts on window move
 - Regenerate and re-queue them
 - May miss frame, oh well
- Switching Contexts
 - Intel-specific problem
 - Old hardware has high-priority queue
 - New hardware has hw contexts
 - GM965/G965 has neither. ☹️



Tearing Will Be Fixed Soon



Multiple frame buffers (shatter)

- Xinerama-lite
- Multiple surfaces per pixmap
- Same acceleration hardware
- Need to integrate DMX-like layer for multiple cards
- Go see Adam's talk



2008: Lots of Changes

- LCA last year:
 - Full of promises
- LCA this year:
 - Full of code



Desktop Talks at LCA

- From click to pixel (Carl, Wed 11:40)
- Xi2 (Peter, Wed 13:50)
- HW-Accelerated HD Video (Nanhai, Wed 16:00)
- Shatter (Adam, Fri 14:50)
- Clutter (Rob, Fri 16:00)

