Kernel GPU API and radeon KMS status

Jerome Glisse

February 2010

Outline

Big fella

Doesn't do coffee yet ...

- 70 kloc (2 times bigger than Intel)
- Biggest driver in the whole Linux kernel tree ?
- More than 9 differents GPU family (R1XX, R2XX, R3XX, RS4XX, R4XX, R5XX, RS6XX, R6XX, RS7XX, R7XX, RS8XX)
- Several different ASIC per family
- Everybody mess with the GPU (BIOS, GPU BIOS, ACPI, ...)

Small computer

More complexe than other piece of hw

- Memory controller
- Command processor
- Clock
- GART
- PLL
- Encoder (LVDS, DVI, HDMI, DisplayPort, ...)
- ...

Future features

New features

- Support for unmappable VRAM (PCI BAR & CPU)
- R8xx family support (Evergreen)
- Use of PM ops for better suspend/hibernate support
- Power management
- HDMI audio for R7XX/R8XX ?
- Improved GPU lockup recording

Code cleanup

- Better fence for improved lockup detection
- Use union to separate asic specific datas
- Better message print for multi-gpu configuration

Issues

COMPUTER READOUT
COMPUTER READOUT
PREMIER 1981 EDITION $1.95
BUYERS GUIDE To Small Business and Hobby Computer Systems
We PEEK at PERIPHERALS
WORD PROCESSING for Profit
Facts on FLOPPIES
Inside WINCHESTER Disk Drives
SOFTWARE Directory for Business, School, Home
MICROCOMPUTERS Profiles on Atari, Heath, Sinclair & more!
by the Editors of Science & Electronics

Most common issues

- Mode detection issue (no EDID or broken EDID)
- PLL issues especialy laptop
- Suspend/Resume issues
- ACPI/BIOS interactions issues
- Memory fragmentation & cs issues
- AGP related issues
- GPU lockup

GPU lockup

Several possible root

- Timing issue with the bus
- Access to invalid address on the bus
- Invalid/Incorrect command stream issues
- Overheating leading to GPU lockup ?
- Bad VRAM ... memtest for VRAM

Hard to fix

- Capture invalid/incorrect command stream. Detecting it ?
- Find errata for timing issue ? Issue with hw manufacturer ...
- Better testing of memory controller setup so we can understand how invalid bus address happens (corrupted GART entry ?).

radeon command processing overview

command processor

- reads command from the ring buffer
- each entry follow a common format: packet (PKT)
- a special packet allow the cp to fetch command from another buffer called indirect buffer (IB)
- in the end all command can be derived into register write (CP translates packet into register write)

radeon command processing overview

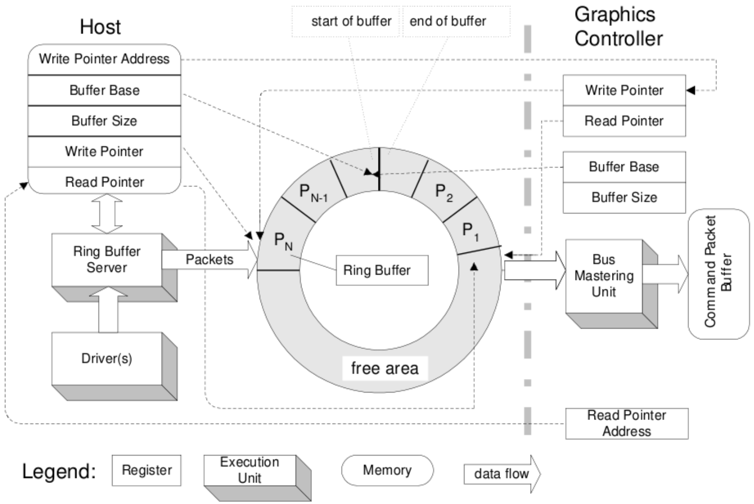


Figure: Ring Buffer and its Control Structure

Packet

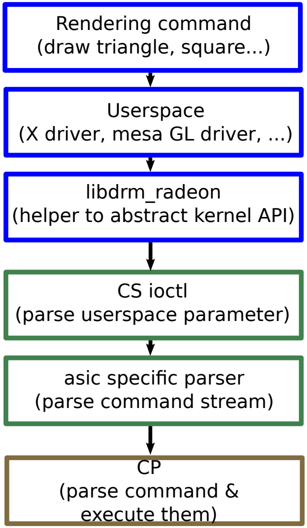
Packet type

- Type 0 write contiguous registers
- Type 1 weird registers write
- Type 2 NOP
- Type 3 special packet

Packet type 3

- each op is dedicated to one specific aspect of the states
- packet3 are translated into register write by the CP
- easier to check, fixed ordering of values

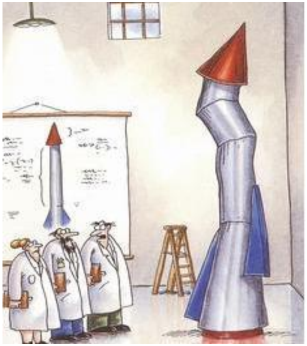
KMS userspace command submission API: cs



Command stream or cs

- buffer filled by userspace with packets
- kernel parse this buffer for security purpose and relocation
- if buffer is valid kernel schedule the buffer as an IB by writting the special packet 3 into the ring

Building packet is not rocket science

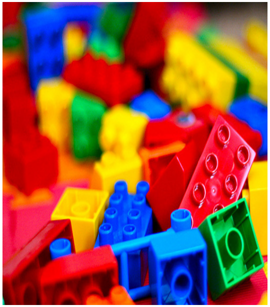


It's time we face reality, my friends... We're not exactly rocket scientists

Bottom line

- Packet packing is simple (bytes shifting & oring)
- Core driver don't need to bother
- Duplicate work packing in userspace, unpacking in kernel space
- Kernel can do it

States split up



Breaking into pieces

- Grouping GPU states into pieces
- Try matching the hw grouping (registers holding different states together)
- Try to match Gallium3D states (src/gallium/include/p_states.h)

Gallium3D theory

Create, bind, profit, unbind

- States atom regroup different states related to the same aspect (blending, zbuffer, stencil, ...)
- Create a states atom
  texture, vbo, blend configuration, ...
- Bind a group of states & perform rendering
- Unbind, rebind, render and profit

Why it's good ?

- Most of the states stays alive during long period of time
- Would be nice to validate once and reuse

Surface states atom

Surface

- size width & height
- layout (tiling, ...)
- usage (texture, renderbuffer, vertex, index ...)

```
struct pipe_surface
{
        struct pipe_reference reference;
        enum pipe_format format;      /**< PIPE_FORMAT_x */
        unsigned width;               /**< logical width in pixels */
        unsigned height;              /**< logical height in pixels */
        unsigned layout;              /**< PIPE_SURFACE_LAYOUT_x */
        unsigned offset;              /**< offset from start of buffer, in byte
        unsigned usage;               /**< PIPE_BUFFER_USAGE_*  */
        unsigned zslice;
        struct pipe_texture *texture; /**< texture into which this is a view  *
        unsigned face;
        unsigned level;
};
```

Blending states atom

Blending

- logcial operation
- dithering
- per render target blend information

```
struct pipe_rt_blend_state {
        unsigned blend_enable:1;
        unsigned rgb_func:3;          /**< PIPE_BLEND_x */
        unsigned rgb_src_factor:5;    /**< PIPE_BLENDFACTOR_x */
        unsigned rgb_dst_factor:5;    /**< PIPE_BLENDFACTOR_x */
        unsigned alpha_func:3;        /**< PIPE_BLEND_x */
        ... };
struct pipe_blend_state {
        unsigned independent_blend_enable:1;
        unsigned logicop_enable:1;
        unsigned logicop_func:4;      /**< PIPE_LOGICOP_x */
        unsigned dither:1;
        struct pipe_rt_blend_state rt[PIPE_MAX_COLOR_BUFS];
};
```

Vertex buffer states atom

Vertex buffer

- num of vertices
- stride between element
- one pipe_vertex_buffer per attributes (pos, color, textures, ...)

```
struct pipe_vertex_buffer {
        unsigned stride;    /**< stride to same attrib in next vertex, in bytes
        unsigned max_index;   /**< number of vertices in this buffer */
        unsigned buffer_offset;  /**< offset to start of data in buffer, in byt
        struct pipe_buffer *buffer;  /**< the actual buffer */
};
```

"True" GPU states

Close to metal ... or silicium

- Use GPU register packing
- Field related to same aspect are often grouped tightly in a small number of registers.
- For instance color control states :

```
struct drm_r600_cb_cntl {
        u32 cb_target_mask;
        u32 cb_shader_mask;
        u32 cb_clrcmp_control;
        u32 cb_clrcmp_src;
        u32 cb_clrcmp_dst;
        u32 cb_clrcmp_msk;
        u32 cb_color_control;
};
```

States atom creation

Building it for the GPU

- Each states atom is selfcontained (all information necessary are provided at creation)
- Perform various check :
  - size of buffer used (for render buffer with * height * bpp, ...)
  - is configuration is legal for the GPU
  - ...
- Build packets which correspond to the states
- Save the packets so later on batch scheduler can use them

Batch

How to render ?

- Submit a batch which has ID of all GPU states

```
struct drm_r600_batch {
        struct radeon_atom *vs_constants;
        struct radeon_atom *ps_constants;
        struct radeon_atom *blend;
        struct radeon_atom *cb;
        struct radeon_atom *cb_cntl;
        struct radeon_atom *pa;
        struct radeon_atom *tp;
        struct radeon_atom *vport;
        struct radeon_atom *db;
        struct radeon_atom *db_cntl;
        struct radeon_atom *vgt;
        struct radeon_atom *spi;
        struct radeon_atom *sx;
        struct radeon_bo *indices;
        unsigned num_indices;
};
```

Batch scheduler

Avoiding reprogramming states

- Program a given states only if GPU was using different one
- Do buffer validation and relocation
- Fill command buffer
- Flush command buffer once no more batch can be queue
- Compute how much GPU memory the command buffer need
- Flush command buffer when there isn't enough GPU space for next batch
- For programming GPU use the packets build at states atom creation

States atom, adding fields

There is no trick ...

- Each states atom has a uniq type ID
- Add a new type ID
- Old userspace can continue to use previous states atom type
- Could mix old states atom types & new one
- Mixing render less efficient same states detections

```
struct drm_r600_cb_cntl_tropbientropnouveau {
        u32 cb_target_mask;
        u32 cb_shader_mask;
        ...
        u32 unknown_register_qui_tue;
};
```

Where should we do that ?

- States atom creation & validation can only happen in kernel

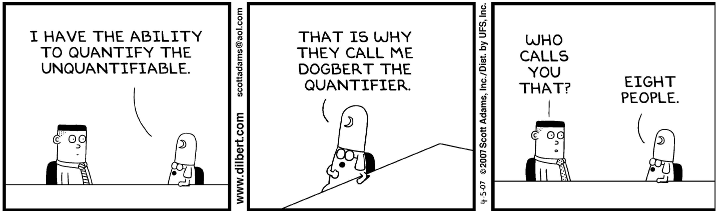
How to store states ?

- It can takes a lot of memory.
- Memory should be accounted as process's memory
- Forbid process to overwrite it

Plans

- Building kernel API is time consuming
- Kernel API are frozen once released
- We want somethings now that can be refined before going into the kernel

Doing it on top of cs ioctl


I HAVE THE ABILITY TO QUANTIFY THE UNQUANTIFIABLE.
THAT IS WHY THEY CALL ME DOGBERT THE QUANTIFIER.
WHO CALLS YOU THAT?
EIGHT PEOPLE.
www.dilbert.com scottadams@aol.com
4-5-07 © 2007 Scott Adams, Inc./Dist. by UFS, Inc.

What's good for

- No ugly command size prediction like in current mesa driver
- No cumberstone BO accounting for flush
- Core driver can focus on building GPU states

Command size prediction

Command size prediction today

- Cumberstone
- Easily breakable when adding or moving states
- Driver writer has to worry about it
- Hard for the driver to avoid state reprogramming duplication

Command size prediction tomorrow

- Each batch has a know command size
- Avoid reprogramming same state in same cs
- Easy to remove a batch from a cs to another cs (forced flush)
- Core driver don't worry about that, it builds states and send batch

Buffer object accounting & flushing

Buffer object accounting & flushing today

- Cumberstone
- Driver has to check over and over if it needs to flush cs

Buffer object accounting & flushing tomorrow

- Easy for the batch scheduler to count size needed for all the buffer of a cs
- Easy for the batch scheduler to flush and put next batch into a new cs

So you get free ticket to the moon ?

Divide and conquer

States atom and batch allow to divide the problem

- Core driver build GPU states
- Batch scheduler deals with flushing and states emissions optimization

Each part has its own problem to solve: easier, simpler, cleaner

Drawbacks/shortcoming ?

- None, i am biased ;)

But R100/R200 states thingy was a disaster !

Too much too often

- All the states were in a single structure
- Had to reemit all the states all the times (each ioctl)
- Each draw command needed full states reemissions

Not the same

States atom presented here doesn't follow this broken design.

- Small number of states per atom
- Each states atom as a uniq ID allow :
  - States atom easily replaced by new one (adding new fields)
- Batch several rendering in one ioctl while allowing changing as little states as possible between the differents rendering

What's coming next ?

Plans

- Finish up r600winsys (nearly fully done)
- Do a quick gallium3d skeleton driver able to render tri-flat
- Plugin a shader compiler -> glxgears stepstone
- Plugin texture & sampler -> quake3 stepstone
- Refine states split up, try to optimize things a bit
- Starts playing with kernel implemented states and benchmark to see if it is worth to put it in the kernel for performances
- Refine the Gallium3D driver to support all the features (multi-buffer rendering, hyperz, tiling, ...)
- Do coffee somewhere between those steps ...

That's all Folks