GPU kernel and userspace border

Jerome Glisse

September 2011
Outline

The bad

The Ugly

The good
No common low level API!

- Problem number one with GPU & multimedia hw
- Complex memory requirement (tiling, interleaving page btw different bank, ...)
- Complex command stream (especially 3D)
Aim for opensource driver

Conflicting balance

- Secure API (can’t trust the userspace)
- Efficient, low overhead
Today hardware

The bad

- No memory protection
- No level of execution
- GART might be reprogrammable through 2D/3D engine
- can’t rely on IOMMU
Future hardware (or today for the lucky few)

- memory management unit
- memory protection
- run level
Sorry can’t trust userspace

What level of abstraction ... If any ...

- OpenGL or alike?
- Register level?
- Middle ground?
OpenGL in kernel?

▶ Not gonna happen too complex
▶ Too much management
▶ Probably inefficient
Register level

- Need to go over all register write (CPU intensive)
- Frozen API
- Too much register to get it right on first revision
Frozen API, welcome to revision hell

- Can’t change list of allowed register
- Missing regs for some features
- Some regs can be program in different way by kernel and userspace
Sorry kernel too old ...

- Userspace have to conditionally enable feature based on kernel version
Middle ground

- Not as complex as OpenGL
- Low level enough
- Straightforward translation to register
- Gallium like but no cso
Surface & memory layout

Surface

- Corner stone of graphic
- Common API across different hw
- Better knowledge of usage in kernel thus better memory management choice

Memory layout

- Shader input
- Vertex buffer object
- Anything that is not surface (raw memory, vertex attributes, shader constant buffer, ...)

Jerome Glisse - GPU kernel and userspace border 13/17
Factor duplicate userspace code
Less duplicate code btw GL & DDX driver
Common userspace driver for different hw?
No complex command stream checking
struct pipe_viewport_state
{
    float scale[4];
    float translate[4];
};

struct pipe_scissor_state
{
    unsigned minx:16;
    unsigned miny:16;
    unsigned maxx:16;
    unsigned maxy:16;
};
struct pipe_rasterizer_state
{
    unsigned flatshade:1;
    unsigned light_twoside:1;
    unsigned front_ccw:1;
    unsigned cull_face:2;    /**< PIPE_FACE_x */
    unsigned fill_front:2;    /**< PIPE_POLYGON_MODE_x */
    unsigned fill_back:2;    /**< PIPE_POLYGON_MODE_x */
    unsigned offset_point:1;
    unsigned offset_line:1;
    unsigned offset_tri:1;
    unsigned scissor:1;
    unsigned poly_smooth:1;
    unsigned poly_stipple_enable:1;
    unsigned point_smooth:1;
    unsigned sprite_coord_enable:PIPE_MAX_SHADER_OUTPUTS;
    unsigned sprite_coord_mode:1;    /**< PIPE_SPRITE_COORD_ */
    unsigned point_quad_rasterization:1;    /**< points rasterized as quads or points */
    unsigned point_size_per_vertex:1;    /**< size computed in vertex shader */
    unsigned multisample:1;    /* XXX maybe more ms state in future */
    unsigned flatshade_first:1;
    unsigned gl_rasterization_rules:1;
    float line_width;
    float point_size;        /**< used when no per-vertex size */
    float offset_units;
    float offset_scale;
};
That’s all Folks