Pipe Driver Overview

Keith Whitwell

Nov 13, 2009

Virtual Machine Group

Gallium3D Interface Overview

Key aspects:

- Contexts and Screens
- Constant state objects
- Further immediate state
- Assembly-style tokenized shaders
- Opaque, immutable texture objects
- Render targets as views into texture objects
- Buffer objects with map/unmap semantics
- Vertex, Index and Constant buffer objects
- Small set of drawing commands
- The backend winsys interface.

Screens and Contexts

Screen:

- Explicit object to manage shared resources.
    - Textures, surfaces and buffers
    - Contexts

Contexts:

- Familiar concept of a rendering context.
- Rasterization and vertex state
- Shaders
- Drawing commands.

Screen: Textures and Buffers

```
struct pipe_screen {
...
   /**
    * Create a new texture object, using the given template info.
    */
   struct pipe_texture * (*texture_create)(struct pipe_screen *,
                                           const struct pipe_texture *);

   struct pipe_buffer *(*buffer_create)( struct pipe_screen *screen,
                                         unsigned alignment,
                                         unsigned usage,
                                         unsigned size );
...
};
```

Created and managed by the pipe_screen object.
Shareable with all rendering contexts created from that screen.
Current screen interface has grown a large number of texture/surface/buffer functions.
Time is ripe for a cleanup of these interfaces.

Screen: Surfaces

```
struct pipe_screen {
...
   struct pipe_surface *(*get_tex_surface)(struct pipe_screen *,
                                           struct pipe_texture *,
                                           unsigned face,
                                           unsigned level,
                                           unsigned zslice,
                                           unsigned usage );
...
};

struct pipe_framebuffer_state
{
   unsigned width, height;
   unsigned nr_cbufs;
   struct pipe_surface *cbufs[PIPE_MAX_COLOR_BUFS];
   struct pipe_surface *zsbuf;
};
```

Surfaces are views into textures
Bind to contexts as render targets or depth-stencil buffers

Screen: Contexts

```
struct pipe_screen {
...
   /* Where is the (*create_context)() entrypoint?
    */
...
};
```

Gallium is not without its own historical oddities.
Conceptually, contexts should be created directly through the pipe_screen interface.
For historical reasons:
- the create_context interface has ended up in the winsys backend interface,
- and has yet to be moved somewhere sensible.

Context: Shaders

```
VERT1.1
DCL IN[0]
DCL IN[1]
DCL OUT[0], POSITION
DCL OUT[1], COLOR
DCL CONST[0..3]
DCL TEMP[0]
  0: MUL TEMP[0], IN[0].xxxx, CONST[0]
  1: MAD TEMP[0], IN[0].yyyy, CONST[1], TEMP[0]
  2: MAD TEMP[0], IN[0].zzzz, CONST[2], TEMP[0]
  3: MAD OUT[0], IN[0].wwww, CONST[3], TEMP[0]
  4: MOV OUT[1], IN[1]
  5: END
```

```
FRAG1.1
DCL IN[0], COLOR, LINEAR
DCL OUT[0], COLOR
  0: MOV OUT[0], IN[0]
  1: END
```

Shaders delivered to driver as stream of tokens.
Vertex shader inputs matched to context state by number.
Vertex outputs matched to fragment inputs by semantic tags.
Fragment inputs include interpolation information.
Fragment outputs matched to fixed-function blending behaviour by semantic tags.
Gallium shader representation was fairly complicated – ongoing simplification.

Context: Constant state objects

```
struct pipe_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 */
   unsigned alpha_src_factor:5;  /**< PIPE_BLENDFACTOR_x */
   unsigned alpha_dst_factor:5;  /**< PIPE_BLENDFACTOR_x */
   unsigned logicop_enable:1;
   unsigned logicop_func:4;      /**< PIPE_LOGICOP_x */
   unsigned colormask:4;         /**< bitmask of PIPE_MASK_R/G/B/A */
   unsigned dither:1;
};
```

Create/Bind/Destroy lifecycle.
Helper module (CSO) for state-trackers manages this through hash table.
Single level save/restore in CSO module to assist beyond-api driver tasks, like quad blitters.

Context: Immediate state

```
struct pipe_vertex_element
{
   unsigned src_offset;
   unsigned vertex_buffer_index:8;
   unsigned nr_components:8;
   enum pipe_format src_format;
};


struct pipe_blend_color
{
   float color[4];
};
```

Passed directly to context in state setting functions.
Always possible to debate which state should be immediate and which CSO.
Try not to deviate too far from other APIs.

Context: Drawing

```
struct pipe_context {
...
   boolean (*draw_range_elements)( struct pipe_context *pipe,
                                   struct pipe_buffer *indexBuffer,
                                   unsigned indexSize,
                                   unsigned minIndex,
                                   unsigned maxIndex,
                                   unsigned mode,
                                   unsigned start,
                                   unsigned count);
...
};
```

Small number of fairly uncontroversial drawing calls.
Will necessarily expand as we add functionality such as instanced drawing for GL3.

Winsys Interface

```
/**
 * This is the interface that llvmpipe expects any window system
 * hosting it to implement.
 */
struct llvmpipe_winsys
{
...
   struct llvmpipe_displaytarget *
   (*displaytarget_create)( struct llvmpipe_winsys *ws,
                            enum pipe_format format,
                            unsigned width, unsigned height,
                            unsigned alignment,
                            unsigned *stride );
...
};
```

Each driver chooses its own winsys interface.
Must be implemented on each platform that supports the driver.
Typical functions include allocating video memory and executing command buffers.
This component has evolved to become a resource manager abstraction.