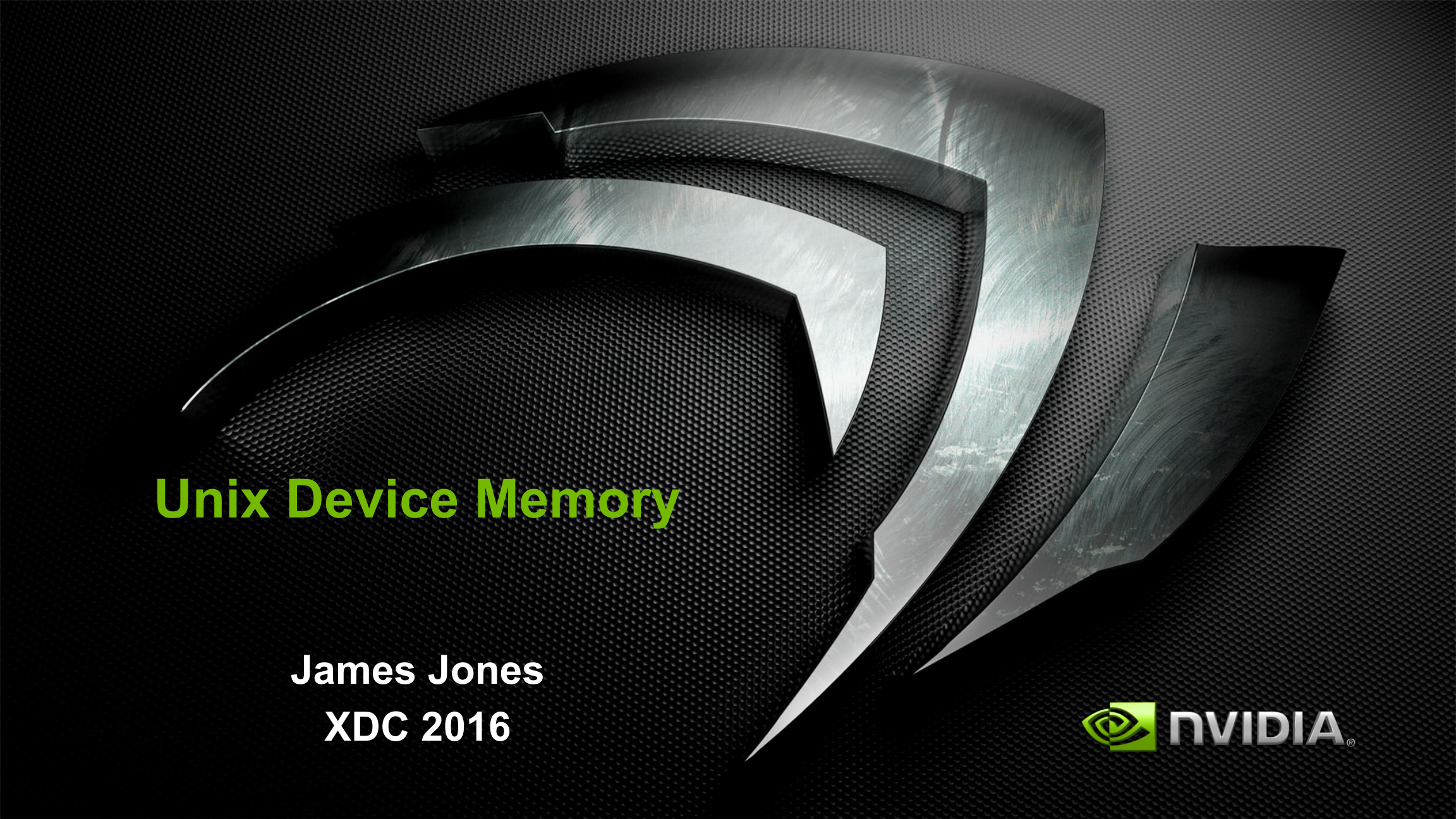

Unix Device Memory
James Jones
XDC 2016
NVIDIA

Background

- Started with a Weston patch proposal
- Many strong views
- Much time invested in current software and APIs
- Thank you for keeping discussions civil
- Many areas for improvement identified

Problem Space

- Device-accessible Surface Allocation in Userspace
- Surface Handles
- Surface State/Layout Management
- Synchronization

Goals

- Consensus-based, forward-looking APIs
- Window System, Kernel, and Vendor Agnostic
- Minimal, Optimal driver interface
- Final destination: Optimized scene graphs for every frame

Prior Art: GBM

- Provides: Allocation, Arbitration, Handles
- Benefits:
  - Incorporated in many codebases now
  - Widely deployed and well exercised
  - Minimal API & implementation
  - Allocation-time usage specification for supported usages
- Current Shortcomings:
  - Process-local handles only. Can import external handles, but not export
  - Currently very GPU-focused
  - Arbitration is within device scope

Prior Art: Chrome OS/Freon

- Attempted to add surface state management to GBM/EGLImage
- Failed to reach consensus optimal design
- Major point of contention: Level of abstraction.

Prior Art: Gralloc

- Provides: Allocation, Arbitration, Handles
  - Synchronization via Android/Linux fence FDs
  - Out-of-process handles require other components
- Benefits:
  - Deployed, proven in field
  - Allocation-time usage specification
  - Support for non-graphics usage
- Current Shortcomings:
  - No explicit surface state management
  - Limited, usage-flag-based arbitration abilities
  - Open Source, but proprietary API

Prior Art: EGLStream

- Provides: Allocation, Arbitration, Handles, State Management, Synchronization
- Benefits:
  - Deployed, proven in field
  - Portable
  - Comprehensive feature set and extensible
- Current Shortcomings:
  - Open standard, but single vendor implementation in practice
  - No cross-device support
  - It is EGL-based
  - Too much encapsulation
  - Behavior loosely defined or undefined in some cases

Prior Art: DMA-BUF

- **Provides: Handles**

- **Benefits:**
  - Supported by non-graphics devices
- **Current Shortcomings:**
  - No centralized userspace allocation API
  - Linux-only
  - Does not describe content layout
  - No arbitration
  - Limited or no allocation-time usage specification

Prior Art: Vulkan

- **Provides: Allocation, Detailed Usage, State Management, Synchronization**

- **Benefits:**
  - Allocation-time usage specification for graphics/compute
  - Image state management
  - Extensible
  - Portable
- **Current Shortcomings:**
  - No Unix cross-process/cross-API/cross-device handles or arbitration
  - Graphics/Compute and Display only

Important features identified

- Minimalism
- Portability
- Support for non-graphics devices
- Optimal performance in steady state
- Allocation-time usage specification
- Driver-negotiated image capabilities
- Good performance during usage transitions
- Multiple usages per image without reallocation
- Image layout transitions

Path Forward

- Suggest a focus on solving problems, rather than picking a winner from existing APIs
- Focus on cross-driver, cross-engine, cross-device image/texture arbitration first
  - This has historically been where everything falls apart
  - Simpler cases fall out naturally from this
  - State transitions are also easier with well-described end points
- Also, Jason Ekstrand has put together some proposals for this

Assumptions

For the sake of simplifying initial discussions:

1. Assume we are designing an ideal allocation API from scratch

2. Think in terms of userspace API first

3. Both API elegance and hardware capabilities are important

Image Sharing Proposals

- Define extensible capability descriptor lists
  - Similar concept to Khronos data-format spec, but describing properties other than sub-pixel data layout and interpretation
- Lists of capabilities could be queried from each “driver”
  - List could be large. Some filtering mechanism would be employed
- Centralized mechanism mutexes the capability namespaces
  - Could be a file in a git repo, Khronos, etc. Anything authoritative
- Image creation function intersects capabilities of relevant drivers

Proposal: How are capabilities filtered?

- Describe the desired usage
  - Examples of usage: Format, operations, dimensions
- Leads to next question: How is usage described?
  - Make use of Khronos data format spec for formats
  - Some usage data, such as width/height have obvious representations
  - Other data lend themselves to boolean flags, like those in Gralloc
  - Some usage is specific to certain devices or engines
  - Each driver ignores usages targeted only at other drivers
  - Special device/engine target for basic usage properties: ALL

Proposal: How are capabilities intersected?

- First pass: Each driver eliminates incompatible capabilities
  - Unrecognized or vendor-specific capabilities are inherently incompatible
  - E.g., Intel driver would trivially eliminate all NVIDIA tiling formats
- Second pass: Sort the remaining capabilities
  - Correct sorting is implementation and usage dependent
  - Therefore, must be done by a driver, not common framework
  - Which driver? Straw-man proposal: Let the app decide.

Proposal: Describing allocation result

- After an image is created, its chosen properties must be described
- Can chosen capability data double as property definitions?

Image Capabilities Vs. Memory Capabilities

- Thus far, focused on image-level capabilities
- What about memory level capabilities?
  - e.g., contiguous requirement
- Image capability mechanism should generalize to describe these
- Might be a separate but symmetric step in allocation machine

Questions?

Backup Slides

Code: Capabilities and Usage Structure

```
#define VENDOR_BASE 0x0000
// Remaining Vendor Namespace: 0x0001-0xFFFF

typedef struct header {
    uint16_t vendor;
    uint16_t property_name;
    uint32_t length_in_words;
};

typedef struct header capability_header_t;
typedef struct header usage_header_t;
```

Code: Capabilities

```
#define CAP_BASE_PITCH_LINEAR 0x0000 // upstream-controlled namespace
typedef struct capability_pitch_linear {
    capability_header_t header; // { VENDOR_BASE, CAP_BASE_PITCH_LINEAR, 1 }
    uint32_t min_stride_in_bytes;
} capability_pitch_linear_t;

#define CAP_NVIDIA_TILED 0x0000 // NV-specific namespace
typedef struct capability_nvidia_tiled {
    capability_header_t header; // { VENDOR_BASE, CAP_NVIDIA_TILED, 1 }
    uint16_t tile_width;
    uint16_t tile_height;
} capability_nvidia_tile_format_t;

#define CAP_NVIDIA_COMPRESSED 0x0001 // NV-specific namespace
typedef struct capability_nvidia_compressed {
    capability_header_t header; // { VENDOR_BASE, CAP_NVIDIA_COMPRESSED, 1 }
    uint32_t compressed;
} capability_nvidia_compressed_t;
```

Code: Usage

```
#define USAGE_BASE_TEXTURE 0x0000 // upstream-controlled namespace
typedef struct usage_texture {
    usage_header_t header; // { VENDOR_BASE, USAGE_BASE_TEXTURE, 0 }
} usage_texture_t;

#define USAGE_BASE_DISPLAY 0x0001 // upstream-controlled namespace
typedef struct usage_display {
    usage_header_t header; // { VENDOR_BASE, USAGE_BASE_DISPLAY, 0 }
} usage_display_t;

#define USAGE_NVIDIA_DISPLAY 0x0000 // NV-specific namespace
typedef struct usage_nvidia_display {
    usage_header_t header; // { VENDOR_NVIDIA, USAGE_NVIDIA_DISPLAY, 1 }
    uint32_t rotation;
} usage_nvidia_display_t;
```

Code: App-supplied usage lists


```
typedef void* device_t;
typdef struct usage {
    device_t dev;
    const usage_header_t usage;
} usage_t;
```

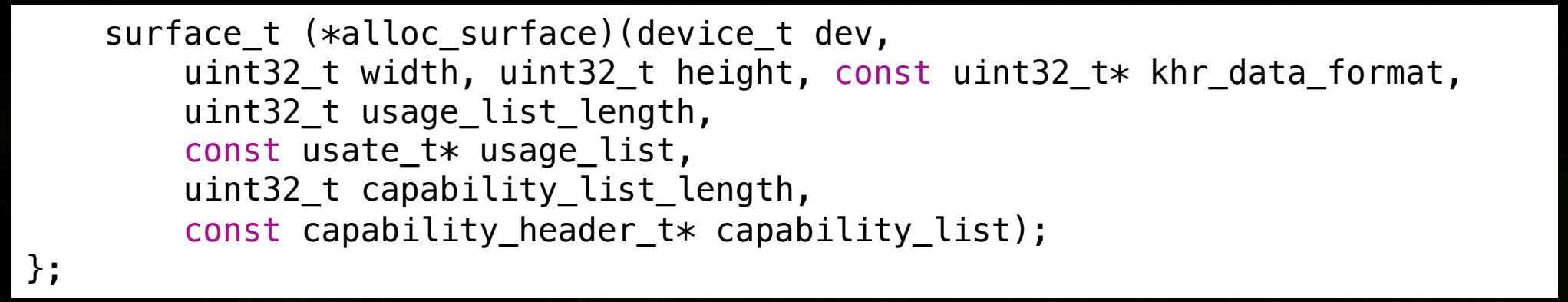
Code: Application Usage

```
typedef void* surface_t;

// Application-facing
AllocSurface(device_t primary_device,
             uint32_t width,
             uin32_t height,
             const void* khr_data_format,
             uint32_t usage_list_length,
             const usage_t *usage_list
             surface_t* surface_out);
```

Code: Driver-side Usage


```
typedef struct driver_api {
    void (*get_capabilities)(device_t dev,
        uint32_t width, uint32_t height, const uint32_t* khr_data_format,
        uint32_t usage_list_length,
        const usate_t* usage_list,
        uint32_t* capability_list_length_out,
        capability_header_t** capability_list_out);

    const capability_header_t* (*filter_capabilities)(device_t dev,
        uint32_t width, uint32_t height, const uint32_t* khr_data_format,
        uint32_t usage_list_length,
        const usate_t* usage_list,
        uint32_t capability_list_length_in,
        const capability_header_t* capability_list_in,
        uint32_t* capability_list_length_out,
        capability_header_t** capability_list_out);
```

Code: Driver-side Usage (cont.)



```
    surface_t (*alloc_surface)(device_t dev,
        uint32_t width, uint32_t height, const uint32_t* khr_data_format,
        uint32_t usage_list_length,
        const usate_t* usage_list,
        uint32_t capability_list_length,
        const capability_header_t* capability_list);
};
```