D3D12 & Vulkan: Lessons learned

Dr. Matthäus G. Chajdas
Developer Technology Engineer, AMD
D3D12 – What’s new?

- DXIL
- DXGI & UWP updates
- Root Signature 1.1
- Shader cache
- GPU validation
- PIX
D3D12 / DXIL

- DXBC gets replaced by DXIL
- Language based on LLVM IR
- New open-source frontend based on Clang (\texttt{dxc})
D3D12 / DXIL

```c
float4 PSMain(PSInput input) : SV_Target
{
    return input.color;
}
```

```c
define void @PSMain()
{
    %1 = call float @dx.op.loadInput.f32(i32 4, i32 1, i32 0, i8 0, i32 undef)
        ; LoadInput(inputSigId,rowIndex,colIndex,gsVertexAxis)
    %2 = call float @dx.op.loadInput.f32(i32 4, i32 1, i32 0, i8 1, i32 undef)
        ; LoadInput(inputSigId,rowIndex,colIndex,gsVertexAxis)
    %3 = call float @dx.op.loadInput.f32(i32 4, i32 1, i32 0, i8 2, i32 undef)
        ; LoadInput(inputSigId,rowIndex,colIndex,gsVertexAxis)
    %4 = call float @dx.op.loadInput.f32(i32 4, i32 1, i32 0, i8 3, i32 undef)
        ; LoadInput(inputSigId,rowIndex,colIndex,gsVertexAxis)
    call void @dx.op.storeOutput.f32(i32 5, i32 0, i32 0, i8 0, float %1)
        ; StoreOutput(outputSigId,rowIndex,colIndex,value)
    call void @dx.op.storeOutput.f32(i32 5, i32 0, i32 0, i8 1, float %2)
        ; StoreOutput(outputSigId,rowIndex,colIndex,value)
    call void @dx.op.storeOutput.f32(i32 5, i32 0, i32 0, i8 2, float %3)
        ; StoreOutput(outputSigId,rowIndex,colIndex,value)
    call void @dx.op.storeOutput.f32(i32 5, i32 0, i32 0, i8 3, float %4)
        ; StoreOutput(outputSigId,rowIndex,colIndex,value)
    ret void
}
```
D3D12 / Updated DXGI

- First-class support for variable refresh rate displays - *-Sync
- HDR support
D3D12 / UWP

- Lots of the initial UWP limitations have been lifted
- Capabilities now on par with normal Win32
D3D12 / Root Signature 1.1

- Allows you to tell the driver that descriptors won’t change
- Can allow some optimizations in the future
- Nice to have, but nothing critical
D3D12 / Shader cache

- The shader cache was not sufficient in D3D12 RTM
- Big improvements in the “Anniversary” edition – now usable
- Note: Drivers may have yet another shader cache!
D3D12 / GPU validation

- Checks descriptors at draw time
- Discovers various hard-to-find bugs (stale descriptors, etc.)
- Rather slow - run over night for regression testing
D3D12 / PIX

- Alternative to RenderDoc
- More than just a debugger
  - Profiling
  - Easy access to shader ISA
Vulkan – What’s new?

- KHR_maintenance1
- EXT_shader_subgroup
- KHR_get_physical_device_properties2
- KHR_shader_draw_parameters
- And many (>20) more …
Vulkan / Usability

- KHR_maintenance1
- Window origin fix (AMD_negative_viewport_height)
- Various other small fixes
Vulkan / Usability

- VK_EXT_debug_marker
- Markup scene just like in D3D with annotations
- Supported by tools!
Vulkan / Porting

- VK_AMD_draw_indirect_count
  - Multi-draw-indirect with **count from buffer**
  - Feature-parity with OpenGL
- KHR_shader_draw_parameters
  - gl_drawID, gl_BaseVertex, gl_BaseInstance
  - Again, feature parity
Vulkan / Porting

- glslang accepts HLSL now
- Already usable for many real-world shaders!
Vulkan extensions / Performance

- VK_AMD_rasterization_order
  - Relaxed rasterization order
  - A stepping stone towards more declarative rendering
What’s new – D3D12 & Vulkan

- Wave-wide
- FP16
Vulkan & D3D12 / Wave-wide

- Wave-wide instructions are now core in both APIs
  - Shader Model 6.0 for HLSL
  - SPV_KHR_shader_ballot, EXT_shader_subgroup_* for SPIR-V
- Console-like programing everywhere!
Vulkan & D3D12 / Wave-wide

Compact wave-wide using a wave-wide prefix sum: Now in SM6 and SPIR-V!
Use the right atomics at the right level

- **Wavefront** - *Intrinsic*
- **Threadgroup** - *Local memory*
- **Dispatch** - *Global memory*
Vulkan & D3D12 / Wave-wide

- Your data is wave-uniform but your shader compiler doesn`t know it
- Express it now in SM6 and SPIR-V!
  - readFirstLane
  - WaveReadFirstLane
Vulkan & D3D12 / Wave-wide

- Another typical use:
  - Iterate over light sources
  - Tell compiler light index is uniform wave-wide
  - Data goes into SGPR instead of VGPR
  - Profit!
Vulkan & D3D12 / FP16

● Same benefits on PC as on console
  ● Reduced register count (and LDS usage!)
  ● Better performance
● Simplifies porting between console & mobile
Dawn of a new era

- Peak performance requires new concepts!
Command lists

- Separate **recording** from **submission**
- Much higher throughput!
Engine evolution / Multithreading

- It’s not just Ashes any more 😊
- Engines are getting ready for massive multithreading
Engine evolution / Multithreading

● Here's Unity firing up all cores!
Engine evolution / Multithreading

• Plan for >10 threads
  • Increase draw call count – high/ultra settings on *new APIs only*
  • Or: Cut latency! Twitchy 144 fps games, anyone?
Graphics, compute, copy queues

- Schedule independent work on independent queues
- Fill up the whole GPU
Lessons learned / Copy queue

- Copy queue is low-latency, low-speed, but it’s separate hardware
  - Copy queue is optimized for transfer over PCIe®, not for GPU local copies
  - For PCIe®, it is the fastest way to transfer data
  - Avoid waiting on copy queue from graphics/compute
  - Ideal use of copy queue is streaming data over a few frames
- Some games still don’t use it ...
  - Multi-millisecond-savings are common
  - If you go from CPU to GPU or back, the copy queue is the queue of choice!
Copy queue
Lessons learned / Copy queue

- Use to it upload all your buffers (constants, index buffers, etc.)
- Use it to defragment memory
Lessons learned / Async compute

- Most games right now

<table>
<thead>
<tr>
<th>G-Buffer + Z-Buffer</th>
<th>Shadow maps</th>
<th>Shading</th>
<th>Post-Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSAO, light tile classification</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lessons learned / Async compute

- Best performance & production proven!

G-Buffer + Z-Buffer | Shadow maps | Shading
---|---|---
Post-Processing | SSAO, light tile classification |
Lessons learned / Barriers

- Barriers are **hard**
- Most issues come from **retrofitting** engines
- New engine designs handle them **much more robustly**
Lessons Learned / Barriers

- Missing barriers
  - Validation layer helps
  - No longer a big issue
- Missing batching
- The „base state“ problem
The “base state” problem
Engine evolution / Task graphs

G-Buffer fill

Shadow rendering

Shading

Post process
Engine evolution / Task graphs

Shadow rendering
Shadow map

Shading
Shadow map

Read/Write transition
Engine evolution / Task graphs

Shadow rendering

Shading

Post process

Shadow map

Passthrough

Shadow map

Alias

Tonemap target
Engine evolution / Task graphs

Shadow rendering

Draw 0  Draw 1  Draw 2  Draw 3  Draw 4

Allow out-of-order execution

Draw 0  Draw 3  Draw 1  Draw 4  Draw 2
Engine evolution / Barriers

- Manual handling *doesn't cut it any more*
- Need higher level abstractions – render graphs
  - This is happening – check out the FrameGraph presentation from Frostbite!
  - Native support in Vulkan since day 1
Engine evolution / Shaders

- Shader permutations are getting fewer
  - Doom has only a couple hundred total
  - More games are changing creation pipelines to prune variations earlier
- More high-level work (around compilers) is happening
Engine evolution / Shaders

Specialize
Engine evolution / Shaders
Engine design

- Engines moving towards more **high-level** rendering
- APIs improve to make them easier to use
- Gamers benefit!
Open topics / Scalability

- Scalability is not solved at all yet
  - Games support old and new APIs for all settings
  - Mobile $\rightarrow$ desktop increasingly important
- New APIs *only* seems to be the path forward
A new approach to APIs

- **Strong collaboration** between ISVs, IHVs and standard bodies
- **APIs evolved** along with game engines
- Loads of changes since release to make your life easier!
Conclusion

● APIs continue to change
  ● What do you need?
  ● What would make your life simpler?

● Community collaboration is critical
  ● Especially for shader language changes
  ● It’s easy to contribute – give it a shot!
Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2017 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. PCIe is a registered trademark of PCI-SIG. Other names are for informational purposes only and may be trademarks of their respective owners. Vulkan and the Vulkan logo are registered trademarks of Khronos Group Inc.