Efficient rendering in The Division 2

Calle Lejdfors – Technical Director, Massive – A Ubisoft Studio
Raul Aguaviva – Developer Technology Engineer, Advanced Micro Devices, Inc.
Focus of talk

Efficient submission of GPU workloads
Deferred command lists
Asynchronous compute
(Raul) AMD DevTech and collaboration
Overall pipeline

- CPU/render/GPU work interleaved
- Submit early, submit often
  - No render graph or *a priori* knowledge of frame layout
- Automatic resource transition tracking
  - But with opt out (untracked)
Interesting frame numbers

- 50-60 submits
- 200 transitions/100 barriers
- 3-6K draws
- 3-6M primitives
- (Some vendors) More time spent submitting than building immediate command lists
Render core

- Handles non-command list operations
  - Resource creation:
    - Buffers, pixel storages, textures, RTs, ...
  - Render state/PSO
- Manages render contexts
  - Graphics/compute/deferred/DMA
Render contexts

- DirectX® 11 like API for command list operations
- Resource binding based around enum’d slots
- Keeps internal cached state
- Each public context is paired with a worker thread + task queue
- Rendering is “just” posting tasks to the appropriate task queue
Rendering objects

- Encapsulated into *render queues*
  - Templatified on sorting strategy
- Can do three things
  - Prepare – sort and group instances
  - Flush – render objects
  - Reset
Filling a render queue

- Culling outputs a 32-bit mask for where to draw each object
  - Z-prepass, gbuffer, CSMs, ...
- Bitmask + object flags decide render queues
- We have 30+ render queues
Flush a queue

- Setup render state for the entire queue
- Upload per-instance (4 uints) data to GPU
  - Single copy per render queue
- For each instance group:
  - Set PSO/buffers/VBs/IBs/…
  - DrawIndexedInstanced
Updating buffers

- All transient data
  - Copy into upload buffers
  - Then, copy into GPU-local buffer
- Shaders only read from GPU-local buffers!

**Not faster, but more stable frame**
Deferred command lists

- Handled by thread local *deferred render contexts*
- Recording done without transitions tracking
  - Liberal use of asserts on transitions/barriers to trap misuse
- Buffer uploads goes straight to DMA without waiting for the execute
Deferred command list example

Deferred ctx: Record

Immediate ctx: ECL
Deferred command list example

Deferred ctx

Immediate ctx

Record

STALL

ECL

Not free!!!
Command list chaining

- Available on some consoles
- Allows executing a command list while it is being recorded
- Minimize risk of CPU stalling CPU and GPU
- Not available in DirectX® 12... 😞
Solution: Emulate!!!

- Enter Queue Manager
- Handles command list operations
  - ExecuteCommandLists, Close, Reset
  - Hides CPU cost of those operations
- Has its own worker thread and task queues
- In effect: a custom driver thread 😊
Queue manager example
Queue manager

- Queue manager submits what it can
- Atomics to track command list state
  - Recording, open
- One queue per context
- Round-robbins executes in priority order
  - Compute, Graphics, DMA
Queue manager details

Deferred ctx

Setup earlier in frame

Immediate ctx

Queue manager

When idle
Queue manager comparison

Naive

Deferred ctx

Immediate ctx

Queue mgr

Deferred ctx

Immediate ctx

Queue mgr
Queue manager

- Eliminates most CPU stalls
- Speculatively prepares command lists
  - Avoids command list create/reset stalls
- Elide superfluous signal/waits
Async compute

- Submits also handled by the queue manager
- 2 types of compute workloads
  - Dependent on gfx state
  - Independent
- Used for workloads that do not need to finish soon
Async compute examples

- Depth downsampling and light culling
- Fog and volumetrics
- Rain/snow GPU particles
- Sky coverage sampling
- Grass/vegetation updates
- Shadows (variable penumbra pre-calc)
- GI relighting
- Async compute is stalling the gfx pipe!
- Can result in GPU under-utilization
- On consoles: limit async compute occupancy
- Not current possible on PC 😞
- D3D12_COMMAND_QUEUE_PRIORITY
Key takeaways

- Check time spent inside DirectX® 12
  - Maybe you need a driver thread too?
- Experiment with buffer upload patterns
- Look at your async compute behaviour!
  - Would low-prio workloads help you?
  - If so, help us push Microsoft + IHVs 😊
AMD DevTech

- Helping devs get the most out of:
  - Tools
  - Driver
  - Hardware
  - Shader Compiler
Existing optimizations in Snowdrop

- Use SGPRs
- Optimized LODs
- Sorting by state
- Batching barriers
- Root signature order
- Use of async compute
Better async

- Async is awesome
- Can we do better?
- Typical usage is:
  - Graphics queue, for what I need ASAP
  - Async queue, for not time critical
- Problem: Async and Graphics queue may compete
Better async

- Competing for execution resources
Better async

- Competing for cache
Better async

- Solution: Parallelize unalike workloads

<table>
<thead>
<tr>
<th>Memory dominated</th>
<th>Shader Throughput</th>
<th>Geometry dominated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shadow Mapping</td>
<td>Deferred lighting (usually)</td>
<td>Rendering highly detailed modules</td>
</tr>
<tr>
<td>ROP heavy workloads</td>
<td>Many Postprocessing effects</td>
<td></td>
</tr>
<tr>
<td>Many Gbuffer operations</td>
<td>Many Postprocessing effects</td>
<td></td>
</tr>
<tr>
<td>DMA operations:</td>
<td>Most compute tasks:</td>
<td></td>
</tr>
<tr>
<td>- Texture upload</td>
<td>- Texture compression</td>
<td></td>
</tr>
<tr>
<td>- Heap defrag</td>
<td>- Physics</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Simulation</td>
<td></td>
</tr>
</tbody>
</table>
Ongoing research

- Interest from several developers
- Expose a way to slow down the async pipe
- Still experimenting...
- Results are so far are exciting!
- PC is tricky
Ongoing research

- Competing for cache
Disclaimer and Attribution

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. This document may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

© 2019 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, Radeon and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.