Developing The Northlight Engine: Lessons Learned

Ville Timonen
Graphics Programmer
Remedy Entertainment

Contents

- Northlight overview
- DX12 porting checklist
- Northlight & DX12
    - Resource transitions
    - Drawing
    - Threading
- Conclusion

Northlight

- Northlight is Remedy’s in-house engine
- Origins in Alan Wake
- Now used in Quantum Break

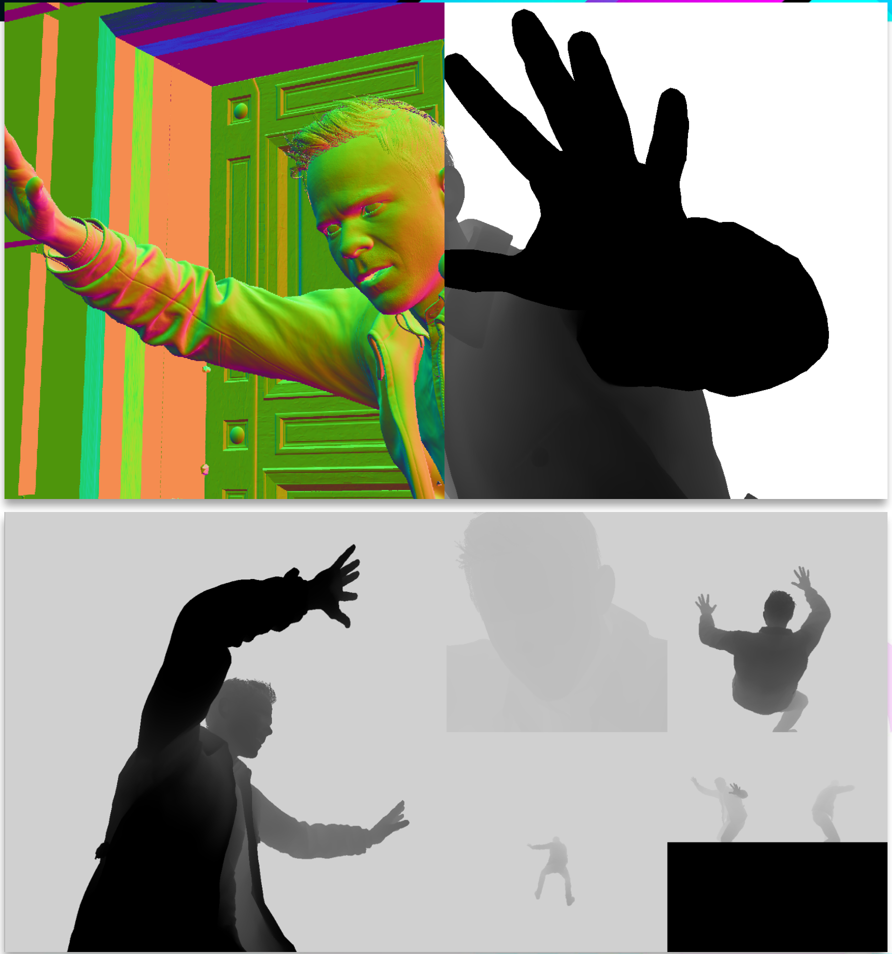
Quantum Break, 3rd person cinematic action game,
Out on Xbox One and Windows 10 on April 5th!

Northlight rendering pipe

1. GBuffer, Velocity, Shadow passes (threaded)
2. Full-screen shadowing
3. Full-screen lighting
4. Primary, transparent passes (threaded)
5. Post-processing

Northlight rendering pipe

1. **GBuffer, Velocity, Shadow passes (threaded)**
2. Full-screen shadowing
3. Full-screen lighting
4. Primary, transparent passes (threaded)
5. Post-processing



Northlight rendering pipe

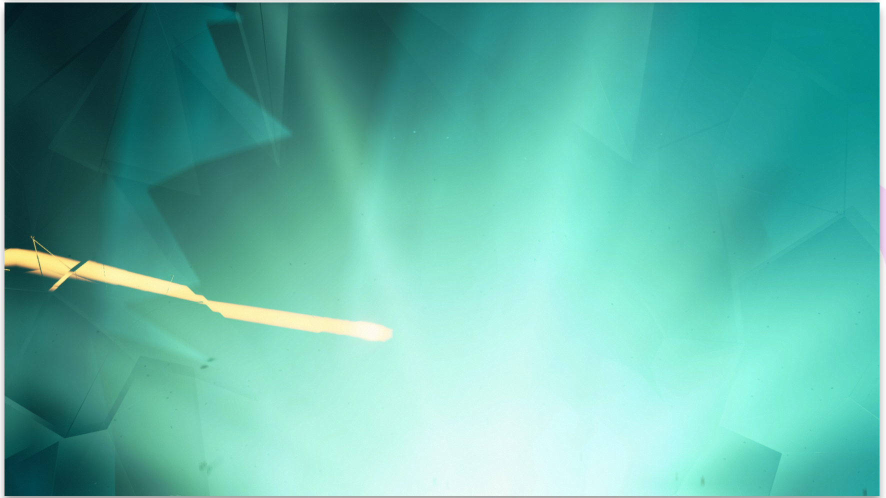
1. GBuffer, Velocity, Shadow passes (threaded)
2. **Full-screen shadowing**
3. Full-screen lighting
4. Primary, transparent passes (threaded)
5. Post-processing

Northlight rendering pipe

1. GBuffer, Velocity, Shadow passes (threaded)
2. Full-screen shadowing
3. **Full-screen lighting**
4. Primary, transparent passes (threaded)
5. Post-processing

Northlight rendering pipe

1. GBuffer, Velocity, Shadow passes (threaded)
2. Full-screen shadowing
3. Full-screen lighting
4. **Primary, transparent passes (threaded)**
5. Post-processing



Northlight rendering pipe

1. GBuffer, Velocity, Shadow passes (threaded)
2. Full-screen shadowing
3. Full-screen lighting
4. Primary, transparent passes (threaded)
5. **Post-processing**

DX12 checklist (decreasing order of headache)

1. Descriptor tables
2. Dynamic resources
3. Pipeline state objects
4. CommandLists/Allocators
5. Resource transitions
6. Staging resources
7. Small resources
8. GenerateMips
9. Null resources
10. Counted/Append buffers
11. Queries

DX12 checklist

1. Descriptor tables

- A table holds descriptors to all resources that any shader stage might use
- Each draw call needs a table, can be reused once the draw is done on the GPU

DX12 checklist

2. Dynamic resources

- No such thing in DX12
- Manage versioning/renaming/rotation yourself
- Write once (CPU), read once (GPU): upload heap ring buffer

DX12 checklist

3. Pipeline state objects (part 1/2)

- Creation is the problematic part
- Ideally output in export pipeline, load at game start
- We create them when we first encounter them
- CS PSOs can be generated at CS load
- ~500 unique graphics PSOs take ~200ms to generate

DX12 checklist

3. Pipeline state objects (part 2/2)

- Root signature (resource layout)
- Shader code
- Vertex shader input layout (not vertex/index buffers)
- Primitive type, blend, raster states, MSAA mode
- Render target and depth-stencil formats (not resources)

DX12 checklist

4. CommandLists/Allocators

- Immediate/deferred contexts in DX11
- Allocator owns the memory

DX12 checklist

5. Resource transitions

- Driver doesn’t track usage anymore
- Have to manually transition to correct state before usage
  - Shader resource
  - Render/depth target
  - Copy source/destination
  - UAV
  - Present

DX12 checklist

6. Staging resources/UpdateSubresource

- No dynamic resources: heavier use of staging resources
- On-demand from ring buffer or persistent
- No UpdateSubresource
  - Can’t rely on the emulated d3dx12.h version
- No staging textures, emulate via staging buffers

DX12 checklist

7. Small resources

- CreateCommittedResource allocates in 64kB pages
- Will not fly for small resources
    - Ideally suballocate all resources in defragged heaps
    - Or special-case small resources

DX12 checklist

8. GenerateMips

- No such thing in DX12
- Write e.g. a compute shader for it
  - We found manual implementations to outperform DX11
  - But need to handle many different cases (2D/3D/arrays/color spaces)

DX12 checklist

9. Null resources

- Can’t just bind nullptr anymore
- Need to have null resources for 1D/2D/3D textures, buffers, UAV textures/buffers, CBVs, samplers
    - Might have to lift null binding higher up in your abstractions to know the type

DX12 checklist

10. Counted/Append buffers

- No such thing in DX12
- Have a separate count buffer that you atomic increment

DX12 checklist

11. Queries

- Yet another easy-to-forget aspect that needs attention
- Manage/rotate query heaps
- Consolidate resolves
- Read back in full heaps

DX11 to DX12

- You're the driver now
- Be mindful of memory usage & performance
- Focus optimizations on bottlenecks
- Think in separate CPU & GPU timelines

Northlight & DX12

Northlight & DX12

- DX12 alongside DX11 path
- Went XBox One first

Northlight & DX12 / ResourceBarriers

Northlight & DX12 / ResourceBarriers

- Do resource transitions automatically in main thread:
  - When binding RT
  - Setting resources in descriptor tables
  - Copying
- Other async render threads aren't allowed to transition
  - Manually make sure resources (mainly the RT) are in their correct state before executing the command list

Northlight & DX12 / ResourceBarriers

- Spamming them unnecessarily might kill your GPU perf
  - Depends on HW
- Use UAV barriers only when necessary, they force GPU to go idle in between dispatches (DX11 style)

Northlight & DX12 / Drawing

Northlight & DX12 / Drawing

- Traverse your draws, catch DX11-style Set* calls
- Keep track of previous values
- If PSO changed, mark it dirty
    - Hash at draw if dirty, fetch PSO from map
- Set Index/Vertex buffers, RT/DS and descriptor heaps only if changed
    - Sets are cheap on CPU but cause HW context rolls

Northlight & DX12 / Drawing

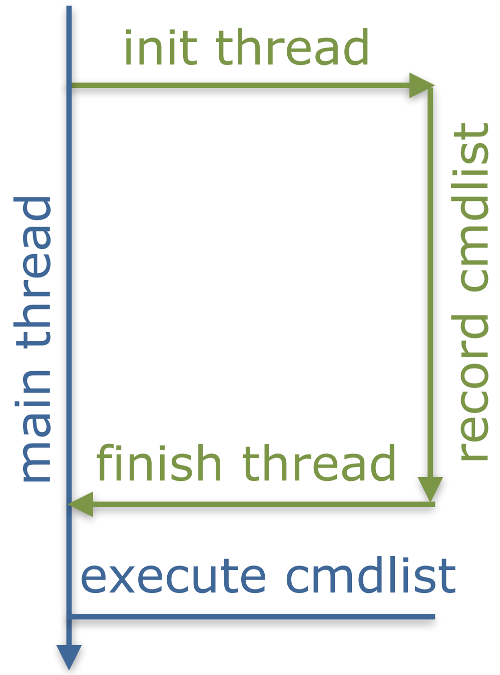
- PSOs are read-only, bind and forget
- Rotate into a free (GPU) descriptor table every draw call
- Reuse descriptor tables once the command list is executed on the GPU

Northlight & DX12 / Threading

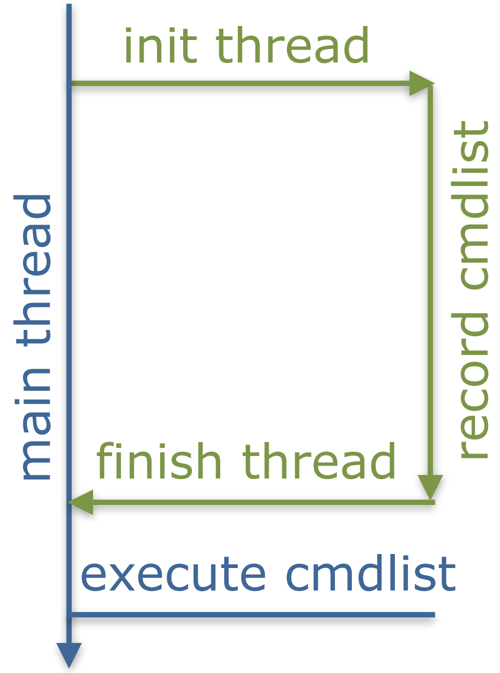
Northlight & DX12 / Threading

- No intermediate/deferred context separation
- Record command lists on any thread, submit from one

Northlight & DX12 / Threading

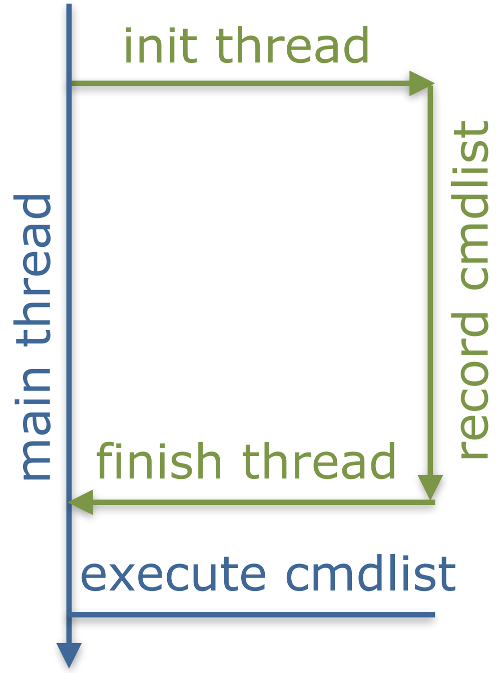


Northlight & DX12 / Threading



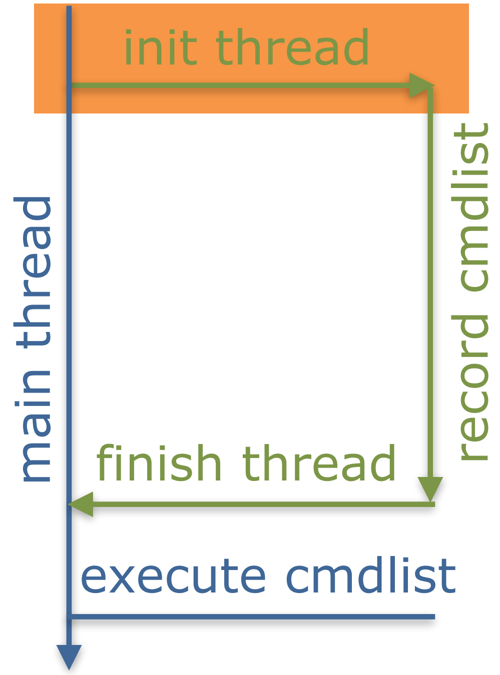
- Pool your:
  - Descriptor table managers: handles table rotation
  - Descriptor table manager GPU fences: lets you know when tables can be reused

Northlight & DX12 / Threading



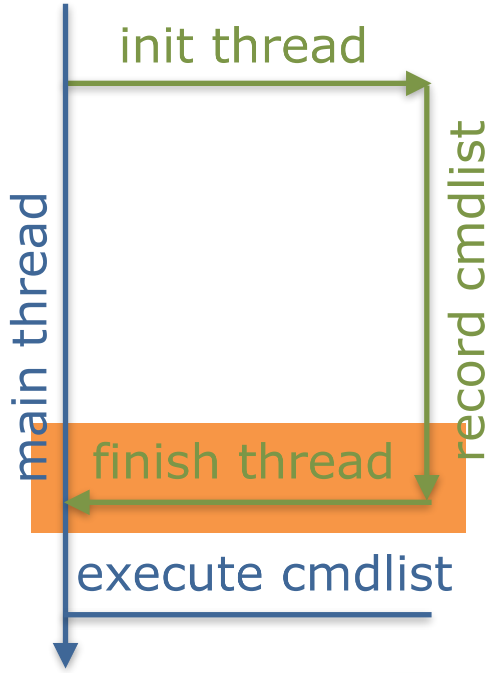
- Pool your:
  - Command lists: can reuse when GPU has finished executing
  - Command allocators: can be used for multiple command lists

Northlight & DX12 / Threading



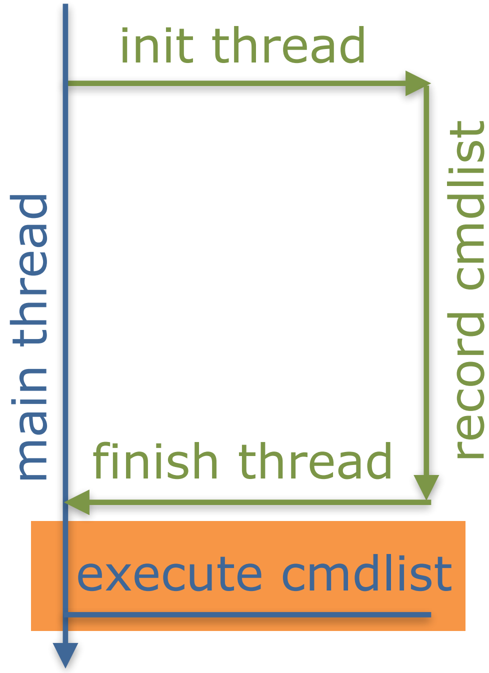
- Acquire:
  - Descriptor manager
  - Descriptor manager fence
  - Command allocator
  - Command list

Northlight & DX12 / Threading



- Release CPU reusable:
  - Descriptor manager
  - Descriptor manager fence
  - Command allocator
  - Command list

Northlight & DX12 / Threading



- Release GPU reusable:
  - Descriptor manager
  - Descriptor manager fence
  - Command allocator
  - Command list

Conclusion

Conclusion

- GPU perf: Do things right, match DX11
    - Not trivial on all architectures
    - Messing up GPU mem mgmt can be costly
- CPU perf: Easy to outperform DX11
    - But are you really API overhead bound?
    - Instancing, LODding, good culling: You're not swamping the driver with draws.

Thank you!
Questions?

www.remedygames.com
@remedygames