Low-Overhead Rendering with Direct3D

Evan Hart
Principal Engineer - NVIDIA
Ground Rules

● No DX9
● Need to move fast
  ● Big topic in 30 minutes
  ● Assuming experienced audience
● Everything is a tradeoff
  ● These are suggestions if an app is SW limited
Why care about overhead?

- Overhead translates to draws per second
- Everyone wants more draws
  - More detailed/interesting world
  - Further draw distance
  - More shadows
Do I have a problem?

- Optimizing the wrong thing is harmful
- All games are CPU limited somewhere
  - Also likely GPU-bound somewhere
- Need to find the tradeoff for your game
- Genre/style can influence balance
  - Online / strategy can be worse
Game / Driver Interaction

Game Thread

Driver Thread

SetVB  SetSRV  SetCB  Draw  SetVB  SetSRV  SetCB  Draw
Game/Driver Interaction

- Driver thread does most real API work
  - Not easily visible in app threads
- App threads just queue commands
  - Typically very fast
- Deferred context is slightly different
  - Driver work is done on app thread
  - Work is kept minimal though
State of the World

- 5 Million draws / second is feasible
  - >50k draws / scene
- 290K draws / second w/ state changes
  - Change most relevant state
- Drawing is cheap
- Object references are expensive
“Full” Draw Call

Ctx->IASetInputLayout( ... );
Ctx->IASetVertexBuffer( ... );
Ctx->IASetIndexBuffer( ... );
Ctx->VSSetShader( ... );
Ctx->Map( ... );
Ctx->VSSetConstantBuffers( ... );
Ctx->PSSetShader( ... );
Ctx->Map( ... );
Ctx->PSSetConstantBuffers( ... );
Ctx->PSSetShaderResources( ... );
Ctx->PSSetSamplers( ... );
Ctx->DrawIndexed( ... );
Cost of the “Full” Draw Call

- Binding 5+ GPU memory objects
  - Vertices, Indices, Constants, Textures
- Two memory management operations
  - Map + DISCARD
- Whole bunch of indirections and cache misses
User costs for “Full” draw call

- ~14 COM calls
- Copying of data tied to buffer update
- Gathering of any data needed to make the calls
- Map probably looks the most expensive
  - Requires driver to provide an immediate response
Driver costs for a “Full” draw

- Every object ref potentially requires
  - Pointer indirection
  - Object lifetime management
  - Object residency management

- Map + Discard requires
  - Rename active object
  - Manage memory pool
What can be done?

- Parallelize the load
- Reduce non-draw functions
- Reduce draw calls
  - Seems counter-intuitive to our goals
Deferred Contexts

- Can allow the load to be spread out
- Biggest help is Map related to constants
  - 30-50% faster isn’t unreasonable
    - On drivers that support them natively
- Not a panacea as the driver thread still does a lot of serial work
Reducing Draw Setup

- Übershaders
- Optimize Constant Buffers
- Sub-allocate Vertex and Index Buffers
- Managing SRVs
- Managing Samplers
Ubershaders

- Changing shaders snowballs costs
  - Shader is connected to everything
- Conditionals are fairly cheap
  - `if (hasSpec) / for( numLayers )`
- Can have a negative impact on GPU costs
  - Secondary factors like register pressure
Reducing Constant Buffer Costs

- Per-Frame Constants
- Per-View Constants
- Per-Draw Constants

Pixel Shader | Vertex Shader
PSSetConstantBuffers1(
  // Parameters from older methods
  UINT StartSlot,
  UINT NumBuffers,
  ID3DBuffer *const *Buffers,

  // Offset in number of constants (16 bytes each)
  const UINT* FirstConstant,

  // Size of the block in constants ( Num % 16 == 0)
  const UINT* NumConstants
)
Shared Constant Buffers DX 11

// Vertex Shader

// VS constants at the beginning
float4x4 WVP_matrix : packoffset( c0 );
float4x4 W_matrix : packoffset( c4 );
...

// Pixel Shader

// PS constants later
float4 Ambient : packoffset( c16 );
...
Vertex and Index Suballocation

- Simple concept
  - Multiple objects can share the same buffer
- Most games standardize vertex formats
  - Handful of configurations
- DrawIndexed offers BaseIndex/BaseVertex
  - Vastly cuts down on # of IASetXXX calls
SRV Management Basics

- Assign slots for “global” textures
  - Shadow maps, environment maps
- Don’t “Clean-up” slots
  - Binding NULL does have a cost
- Group SRVs that change together
  - Albedo and normal
SRV Management Advanced

- Suballocate from texture arrays
  - Sizes and format are typically quasi-standard
  - Terrain tiles are a fantastic example
  - Just need to add an extra index to CB

- Need to take some care
  - Allocating the max array for all combinations probably doesn’t make sense
Sampler Management

- Repeat SRV basic advice
- Samplers don’t change much
  - Fairly limited set of common choices
Revised Draw Call

// Setup:
//  ubershader constants
//  texture array indices
//  normal constants, like transform
Ctx->Map( ... );

// Don’t need to rebind CBs

// Offsets in multiuse IB/VB
//  StartIndexLocation -> offset to IB
//  VertexOffset -> offset for vertices
Ctx->DrawIndexed( ... );
DX 11.1 is better still

- Use one Map() for batch of draw calls
- Change offset into constant buffer per draw
Reducing draw calls

● You’ve heard it for years
  ● Instancing

● Nearly every game could benefit
  ● Same model drawn N times in a frame

● There are practical concerns
  ● Depth sorting
  ● Constant buffer size
Wrap-up

- Analyze your situation
- Consider API options
- Get the engine out of the way
  - See appendix slides online
Thanks

NVIDIA Devtech team
NVIDIA D3D Driver Performance Team
Bryan Dudash
Dan Baker