

Optimizations and Star Wars Galaxies

Jeff Grills

Sony Online Entertainment

Technical Director

jgrills@soe.sony.com

What is SWG?

- MMORPG developed by Sony Online Entertainment and published by LucasArts
- Shipped in summer 2003
- Uses D3D9 interfaces, PS/VS support
- Ground-based game at launch
- Space expansion in the works

Challenges for SWG

- Seamless 16 km by 16 km planets
- 2 km view distance
- Both first and third person camera modes
- Can see the horizon
- Players control the scene content, not developers
- Server can and will send down object creation messages to clients anytime

Graphics layer DLLs

- Graphics code that interfaces with D3D is loaded from a DLL
 - FFP only DLL
 - VS/PS only DLL
 - Deprecated mixed DLL supporting FFP & VS/PS
 - All built from same source using `#ifdef`
- Launcher program detects hardware and sets up DLL configuration (among other things)

DPVS

- Licensed 3rd party library from Hybrid to do visibility determination
- Excellent for SWG cities, especially player cities where we couldn't have pre-computed visibility offline
- Wonderful "occluder fusion"
- We currently do not use terrain chunks as occluders
- Non-deterministic rendering can make profiling more challenging

Typical number of draw calls and triangles per frame

Type	Draws	Primitives
Environment	15	10000
Terrain	50	7000
Creatures	55	8000
Static objects	150	18000
UI	45	1800

Batching

- It's a necessary evil on today's hardware
- Sometimes worth using a more expensive effect across an entire object rather than using two effects
- Terrain is batched by textures with little regard given for lighting

Vertex shaders

- Both ASM and HLSL vertex shaders are used, mix of both
- FFP has more convenient flexibility when it comes to texture coordinate set binding
- SWG compiles vertex shaders on the fly from the source language using #defines to implement the same flexibility
- Compiled to the max version supported by the hardware
- May be able to use multiple streams with same vertex buffer bound using different decls to achieve the same flexibility and compile the shaders offline

Pixel shaders

- Both ASM and HLSL pixel shaders are used. High percentage still ASM, especially ps_1*. ps_2* shaders are much more likely to be in HLSL
- Unlike vertex shaders, these are precompiled in an offline tool
- Effort underway to add cool new effects targeting PS 2.0

Setting vertex shader constants 1

- Depends primarily on the number of constants, not the number of calls
- SWG does 3x SetVertexShaderConstants calls as draw calls
- Optimized to track dirty regions and do a single update at draw time
 - Then ~0.80 SVSC calls per draw call
 - Added 3% wasted constant sets
 - Total time spent in SVSC did not budge
 - Reverted the optimization

Setting vertex shader constants 2

- Still, function calls aren't free. I've also been told the per-constant overhead will likely be improved
- SWG has a hard-coded register layout
 - Group constants changed at similar frequency
- For VS/PS hardware, don't tie yourself down to FFP structures like D3DMATERIAL9
 - Your lighting model may not use emissive
 - Specular color and specular power are not together in memory, so you can't set them in a single call to a pixel shader if you need those values there as well

HLSL optimization that wasn't

```
max(dot(normal_w, light.direction), 0.0)
```

VS

```
saturate(dot(normal_w, light.direction), 0.0)
```

- PS supports `_sat` modifier, but VS doesn't. Difficult to write code that compiles optimally for both
- Understanding of the underlying target language is still beneficial to writing optimal HLSL
- At least compare assembly output instruction length after making changes

Don't resend the same state

- Most games have some high level state management
- Typically don't check matrices for difference because they have too many elements to compare efficiently
- Discussing state changes with Valve, they suggested setting up as much world state as possible and then drawing as much without changing state
- SWG uses the identity matrix for terrain. But it is getting resent to the card for each terrain draw call
- Other games that can preprocess their geometry may be able to use this optimization a lot more by converting multiple objects into the same space

SWG Characters

- Highly customizable
 - Each character has unique geometry built from morph targets
 - Some textures baked out, primarily faces, but most not
- Equipping a character
 - Hair, chest piece from one armor set, pants from another, shoes from yet another
 - Two hue rendering
- Many draw calls (15+) per fully equipped character
- Entire character is rendered using a single dynamic VB for performance
 - Tried separate static + dynamic VB, but it was slower
 - Recently discovered characters aren't optimized for post-transform cache like our static geometry
 - Could be because of input vertex cache misses
- Low LOD batcher renders multiple skinned objects at once using just vertex colors based off the average texture color

Examine a whole frame

- Examine entire beginning state every call to D3D for an entire frame
- It's very easy to have a rogue state that may impact performance
 - We just recently found a cut & paste bug that enabled 2d texture transforms for all FFP hardware for basically every triangle
 - Also would have found the terrain resending the object-to-world transform identity matrix repeatedly

Choose optimizations wisely

- Make sure you are optimizing things that are impacting performance
 - Spent 4 days adding in vs_2_0 support for static conditionals to disable lights that weren't on for a demo. No performance gain – the hardware had more vertex processing power than we were using
- Don't optimize for uncommon case
 - Not worth optimizing terrain system for stationary camera

The disk is your enemy

- Some games may be able to preload all the data for a level ahead of time
- Asynchronously load data from another thread
- If that thread stalls waiting for data from disk, at least it's not your main game
- Until the data is loaded, an object isn't rendered. Players can't tell the difference between that and the server sending a create message down slightly later.
- Integrated relatively late in SWG, so the integration had to work around assumptions that meshes were already loaded

Profile of SWG in VTune

- Where does time go?
 - Application 50-70%
 - Driver 25-50%
 - OS 5-10%
- Can be dependent upon in-game options like stencil shadows (which can peg the driver)
- Changing window size can help identify fill rate/pixel shader bound problems

Time spent in D3D 1

Class	Calls	Time (s)	Avg (s)
presentTime	1	3.7882E-04	3.7882E-04
beginSceneTime	1	3.9111E-06	3.9111E-06
drawTime	327	9.1716E-04	2.8048E-06
setVertexShaderConstantsTime	833	2.0053E-03	2.4073E-06
setViewportTime	4	9.2190E-06	2.3048E-06
setScissorRectTime	4	9.2190E-06	2.3048E-06

Time spent in D3D 2

Class	Calls	Time (s)	Avg (s)
endSceneTime	1	1.9556E-06	1.9556E-06
setPixelShaderTime	34	5.1962E-05	1.5283E-06
setVertexShaderTime	41	6.0064E-05	1.4650E-06
setIndicesTime	146	1.8438E-04	1.2629E-06
setStreamSourceTime	149	1.8326E-04	1.2300E-06
setRenderStateTime	83	1.0085E-04	1.2151E-06
setTextureTime	437	5.2856E-04	1.2095E-06
setSamplerStateTime	50	5.9505E-05	1.1901E-06
setPixelShaderConstantsTime	745	8.3670E-04	1.1231E-06

Misc

- Keep sort flexible to accommodate changing hardware and runtime costs
- Pre-pack scalar constants into 4d vectors to be sent to D3D together
- Use of function pointers / pointer to member functions to avoid conditional branches
- Only update the D3D mouse cursor when it has changed
- Prevent users from dragging game window onto other monitors
 - IDirect3D9::GetAdapterMonitor
 - GetMonitorInfo

Questions?

Jeff Grills
Sony Online Entertainment
Technical Director
jgrills@soe.sony.com