



Harnessing the Power of DirectX 10

Including DX9 porting considerations

Nicolas Thibieroz
European Developer Relations
AMD Graphics Products Group
nicolas.thibieroz@amd.com

Develop conference, 24-26 July 2007

V1.01

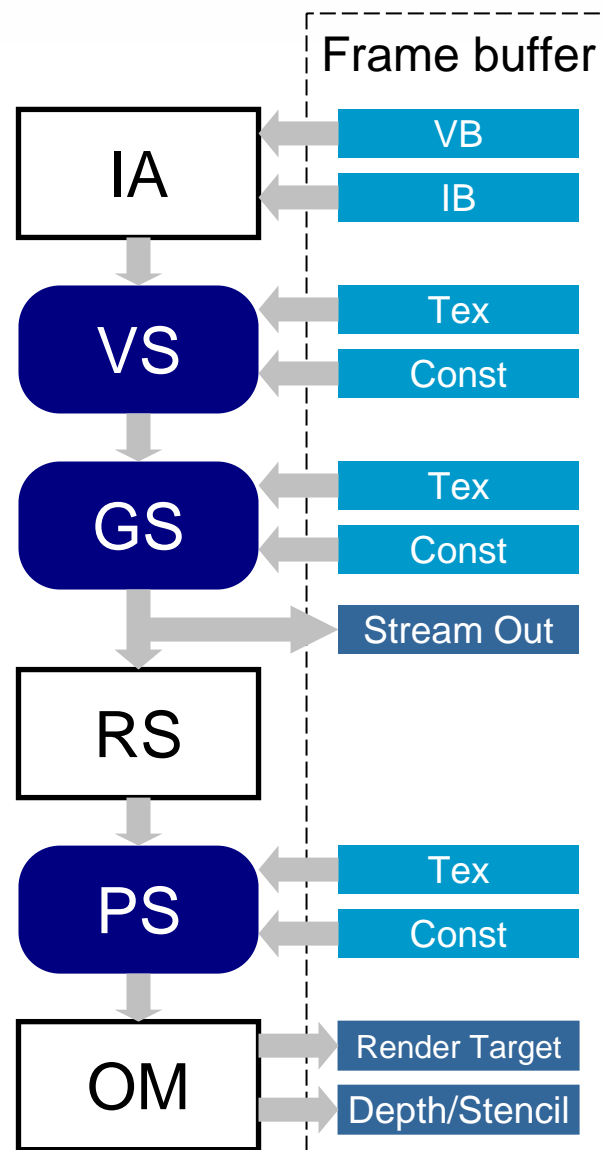
Direct3D 10 Pipeline

New pipeline in Direct3D 10

IA = Input Assembler

RS = Rasterizer Stage

OM = Output Merger



DX9 Deprecated Features

Fixed-function Transformation & Lighting (incl. clip planes)

- Use shaders for everything
- Use clip/cull distance in vertex or geometry shader

Assembly shaders

- HLSL shaders only from now on

Alpha test

- Use `discard()` or `clip()` in HLSL pixel shaders

Triangle fans

- You don't really need them

Point sprites

- Use Geometry shader to output them

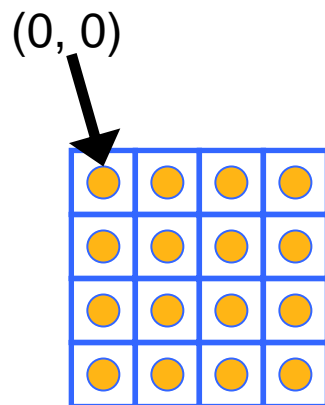
Pixel Coordinate System

Finally pixels and texels match

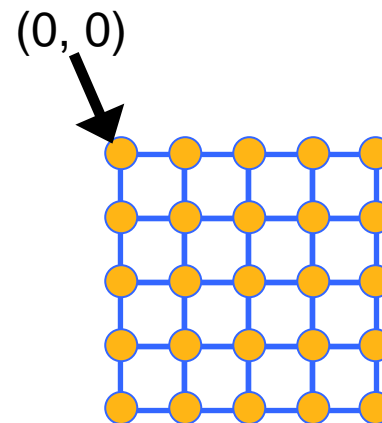
No need to offset position or texture coordinates by half a texel in DX10

This may affect some of your artwork

- Screen-space textures/shaders e.g. HUD



DirectX 9



DirectX 10

Input Layout & Signature

Input layout has to be set before each draw call

In DX10 the input layout requires both:

- Vertex layout info
- Vertex shader input signature

```
ID3D10InputLayout* pInputLayout;
```

```
pDev10->CreateInputLayout( layout, numElements,
                          pInputSignature, IASize,
                          &pInputLayout );
```

Come from vertex structure

Come from vertex shader

Can be a pain to implement from some DX9 engines

- But ultimately less error prone and more efficient

DX10 Caps

“Didn’t you say there were no caps in DX10?”

DX10 “base functionality” guaranteed on all HW

Exceptions require caps:

- Filtering capabilities for a given texture format
- Blending capabilities for a given render target format
- MSAA capabilities for a given render target format
- MSAA *resolve* capabilities for a given render target format

Use `ID3D10Device::CheckFormatSupport()` to determine capabilities

DX10 Batch Performance

The truth about DX10 batch performance

“Simple” porting job will not yield expected performance

Need to use DX10 features to yield gains:

- Geometry instancing
- Intelligent usage of state objects
- Intelligent usage of constant buffers
- Geometry shader (render target selection)
- Texture arrays

States Management

DX10 uses immutable “state objects”

- Input Layout Object
- Rasterizer Object
- DepthStencil Object
- Blend Object
- Sampler Object

DX10 State objects require a new way to manage states

- A naive DX9 to DX10 port **will** cause problems here
- Always create state objects at load-time
- Avoid duplicating state objects
- Recommendation to sort by states still valid in DX10!

States Management (2)

Example of bad porting of states from DX9 to DX10:

DX9:

```
// Render geometry in wireframe
pDev9->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME);
pDev9->DrawIndexedPrimitive(...);
```

DX10:

```
// Render geometry in wireframe
D3D10_RASTERIZER_DESC RastDesc = DefaultRSDesc;
RastDesc.FillMode = D3D10_FILL_WIREFRAME;
ID3D10RasterizerState* pMyRastState;
pDev10->CreateRasterizerState(&RastDesc, &pMyRastState);
pDev10->RSSetState(pMyRastState);
pDev10->DrawIndexed(...);
pMyRastState->Release();
```

Please don't do this!

Constant Buffers Management

Probably the #1 cause of slow performance in current DX10 apps!

Constants are declared in buffers in DX10

```

cbuffer PerFrameUpdateConstants
{
    float4x4 mView;
    float     fTime;
    float3    fWindForce;
    // etc.
};

cbuffer SkinningMatricesConstants
{
    float4x4 mSkin[64];
};
    
```

When a CB is set its *whole* contents are uploaded to the GPU

Need to strike a good balance between:

- Amount of constant data to upload
- Number of calls required to do it

Constant Buffers Management (2)

Always use a pool of constant buffers *sorted by frequency of updates*

- Don't go overboard with number of CBs to update!

Global constant buffer unlikely to yield good performance

- Especially with regard to CB contention

Group constants by access patterns in a given buffer

- This will help cache usage
- Example:

```
cbuffer PerFrameUpdateConstants
{
    float4    vLightVector;
    float4    vLoadsOfOtherStuff[32];
    float4    vLightColor;
};
```

BAD

```
cbuffer PerFrameUpdateConstants
{
    float4    vLightVector;
    float4    vLightColor;
    float4    vLoadsOfOtherStuff[32];
};
```

GOOD

Textures and Sampler States

Textures and samplers are decoupled in DX10

Can set sampler states inside shaders or from application

When sampling a texture DX10 syntax requires:

- Texture to sample from
- Sampler state to use

```
Texture2D MyTexture;  
SamplerState mySampler;  
myColor = myTexture.Sample(mySampler, Texcoord);
```

This decoupling may need serious changes to your code when porting from DX9

Geometry Shader

Geometry shader has the ability to generate and kill primitives

- Can be disabled by passing a NULL Geometry Shader
- Remember to set winding order when outputting triangles

Also allow render target index output

Shadow volumes

Point sprites

Silhouette detection

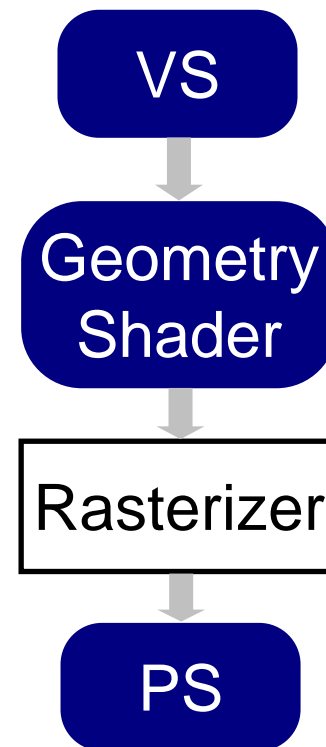
Fur/Fins generation

Render to cube map

Render to cascaded shadow maps

Global Illumination

etc.



Geometry Shader Recommendations

Use ALUs in your geometry shaders to hide latency

- This is because the GS can write data out to memory

Minimize the size of your output vertex structure

- Yields higher throughput
- Allows more output vertices (max output is 1024 floats)
- Consider adding pixel shader work to reduce output size

Cull triangles in Geometry Shader

- Backface culling / frustum culling

Also minimize the size of your input vertex structure

- Pack data and use ALU instructions to unpack it
- Combine inputs into single 4D iterator (e.g. 2 UV pairs)

Accessing Depth and Stencil

DX10 enables the depthstencil buffer to be read back as a texture

Use views to define depthstencil buffer access

- ID3D10DepthStencilView
- ID3D10ShaderResourceView

Enables features without requiring a separate depth render

- Soft particles
- Atmosphere pass
- Depth of field
- Access to stencil buffer



Not compatible with MSAA!

Depthstencil may not be bound as depth **and** texture at the same time

Depth **and** stencil cannot be accessed as SRV simultaneously

- DXGI_FORMAT_R24_UNORM_X8_TYPELESS to access depth data
- DXGI_FORMAT_X24_TYPELESS_G8_UINT to access stencil data

Shadow Mapping

Shadow mapping is pretty much the standard now

Cascaded shadow maps very popular

- Can be combined with various shadow map algorithms

No special “non-standard” code paths in DX10!

- Can bind a depth buffer with no color buffer required
- PCF texture instructions supported across all HW
 - SampleCmp and SampleCmpLevelZero

Geometry shader can be used to output to multiple cascaded shadow maps

- Use `SV_RenderTargetIndex` to select shadow map

MSAA first-class citizen in DX10

MSAA “mostly” orthogonal in DX10

- Alpha to coverage
- Multiple Render Targets
- Sample masks
- Exception: MSAA depthstencil buffer readback

Per-sample access in pixel shader allows custom resolves

- Post-Tone Mapping HDR MSAA resolves
- MSAA with Deferred Rendering
- Anti-aliased shadow maps

Post-Tone Mapping HDR MSAA Resolve

Standard MSAA resolve occurs before tone-mapping
 HDR MSAA resolve should be done **after** tone-mapping!

$$\text{ToneMap}\left(\frac{\sum_{n=1}^{\#samples} \text{Sample}(n)}{\#samples}\right) \neq \frac{\sum_{n=1}^{\#samples} \text{ToneMap}(\text{Sample}(n))}{\#samples}$$

Standard resolve



Custom resolve



Alpha to Coverage

Combine with MSAA to provide edge anti-aliasing on transparent polygons

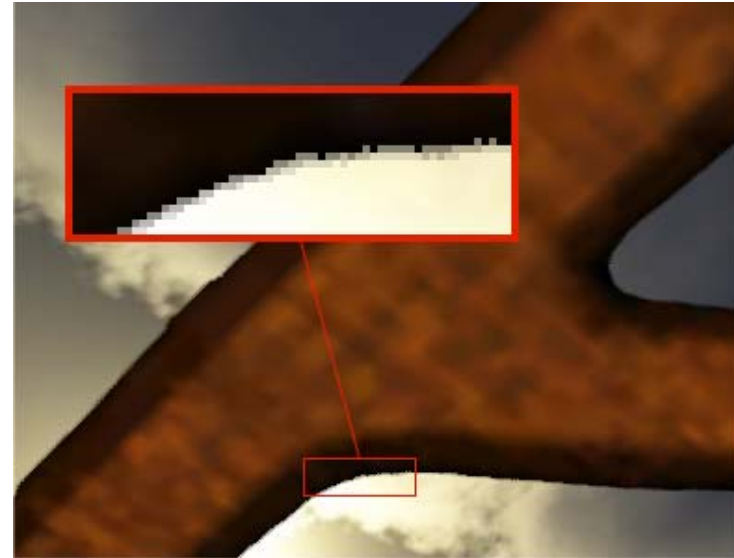
- Easy way to improve visuals without sorting geometry

Simple on/off boolean switch in Blend state object

```
typedef struct D3D10_BLEND_DESC {
    BOOL AlphaToCoverageEnable;
    ...
}
```



MSAA 6X, no alpha to coverage



MSAA 6X, alpha to coverage enabled

Other Bits and Pieces

Dual-source color blending

- Pretty cool feature rarely mentioned in DX10 materials

DepthClipEnable should be set to true

- Unless you need depth clamping (new DX10 feature)

DX10 allows the setting of several interfaces in one call

- `IASetVertexBuffers(StartSlot, NumBuffers, ...);`
- `PSSetShaderResources(StartSlot, NumViews, ...);`
- Easy to forget when porting straight from DX9 or OGL!
 - Need to go beyond the 1:1 API translation for a good port

Primitive topology is in a separate function

- `IASetPrimitiveTopology()` – don't forget about it 😊

Conclusion

Games in development should include a DX10 version

- Watch out for poor porting job though!

DX10 is a new API thus a learning curve is necessary

- DX10 very well designed API and pleasure to work with
- Can unlock the performance and visuals of your title

Use DX10 features for a real difference in your DX10 SKU

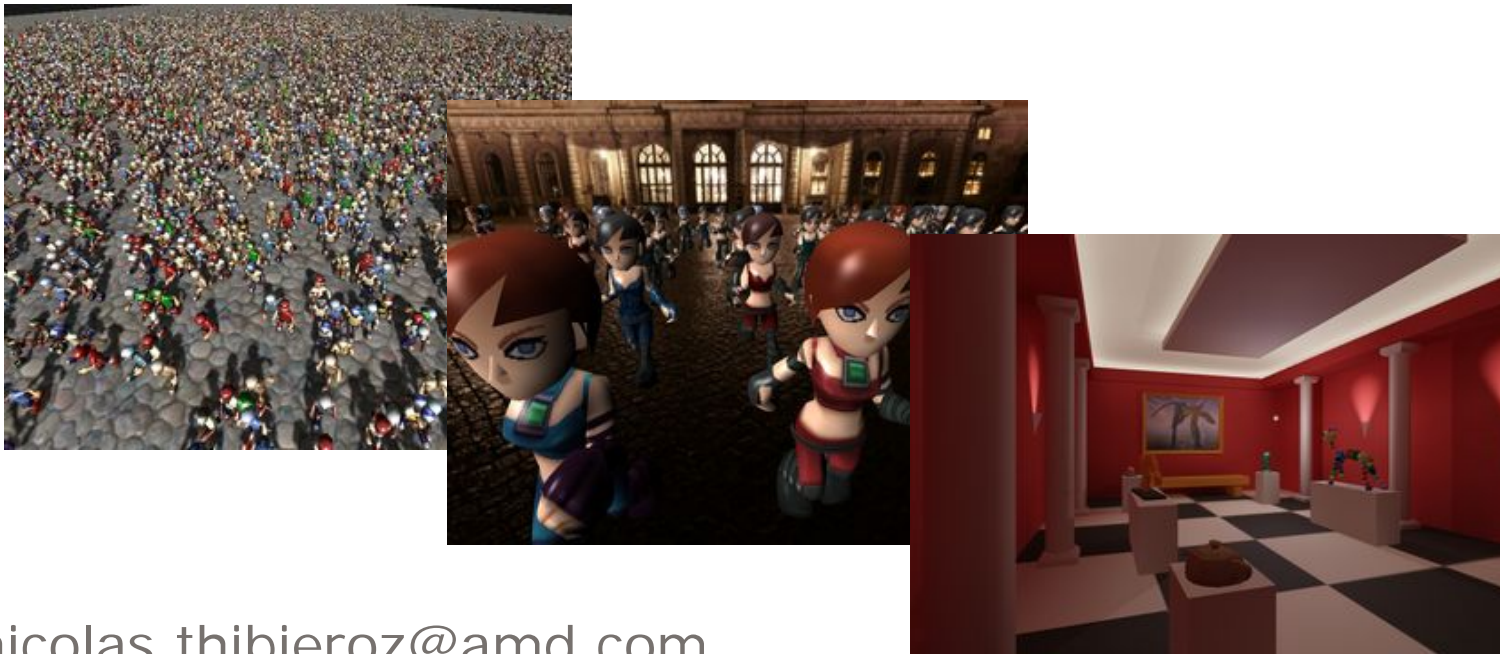
- Geometry Shader
- Unified Shader Model 4.0
- Data recirculation techniques
- Alpha to coverage
- Etc.

Questions?

Radeon DX10 SDK at:

<http://ati.amd.com/developer/radeonSDK.html>

(includes ATI HD 2000 programming guide)



nicolas.thibieroz@amd.com