Using D3D10 Now

Kevin Myers
NVIDIA

D3D10 Is Ready

- Hardware is available
- OS is shipping
- SDKs and tools are here
- Adoption has already begun…

Where few Devs have gone before

- Worked with several over last year
- Learn from their trials and tribulations
- D3D9 -> D3D10
- Case study
  - Flagship Studios
    - *Hellgate London*

Hellgate London

Initial Issues

- Not as simple as D3D8 to D3D9
    - Can’t program D3D9 class device with D3D10
- No fixed function
- State management very different

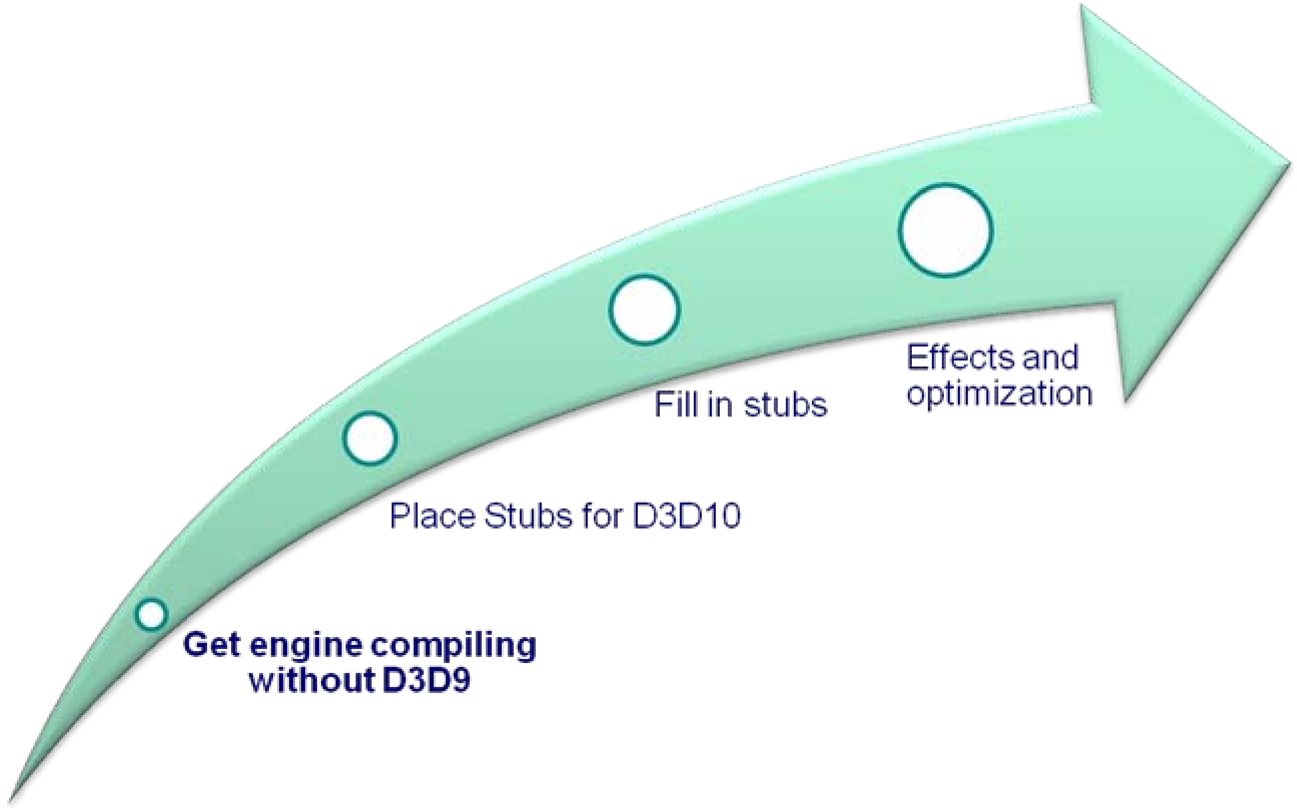
What’s Good

- HLSL Syntax is the same
- Most D3DX Functionality is there
    - Math lib the same
- More orthogonal
    - Cleaner abstractions

First some choices

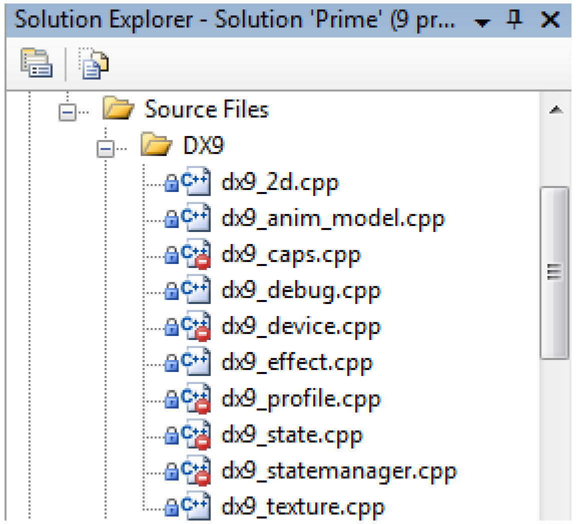
- Hellgate binary is API specific
  - Pros
    - Leverage Dx9 similarities
    - Header file magic
    - Dead code removal
  - Cons
    - Maintenance of separate compile targets
- Using FX system
  - Leverage cross API support

Hellgate Timeline



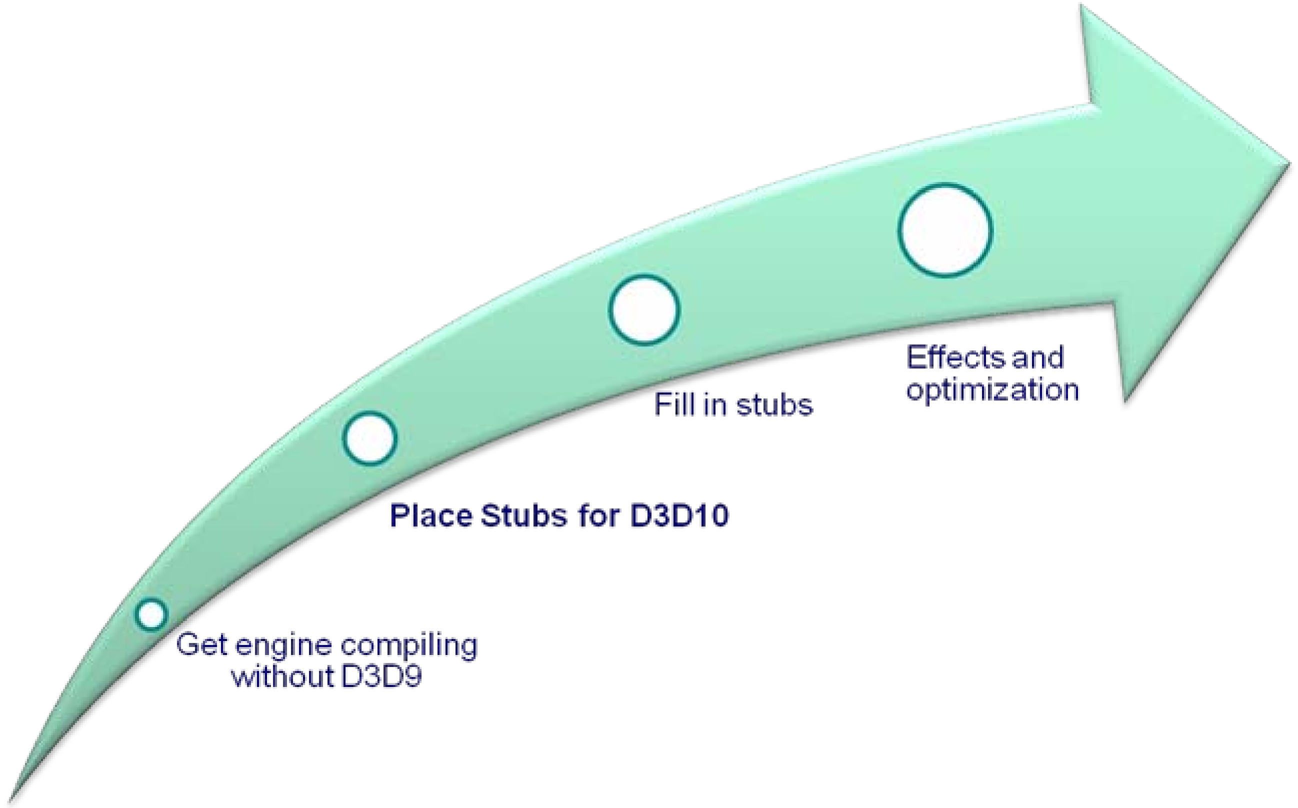
Get engine compiling

- Remove headers
- Disable source files



Source Files
DX9
dx9_2d.cpp
dx9_anim_model.cpp
dx9_caps.cpp
dx9_debug.cpp
dx9_device.cpp
dx9_effect.cpp
dx9_profile.cpp
dx9_state.cpp
dx9_statemanager.cpp
dx9_texture.cpp

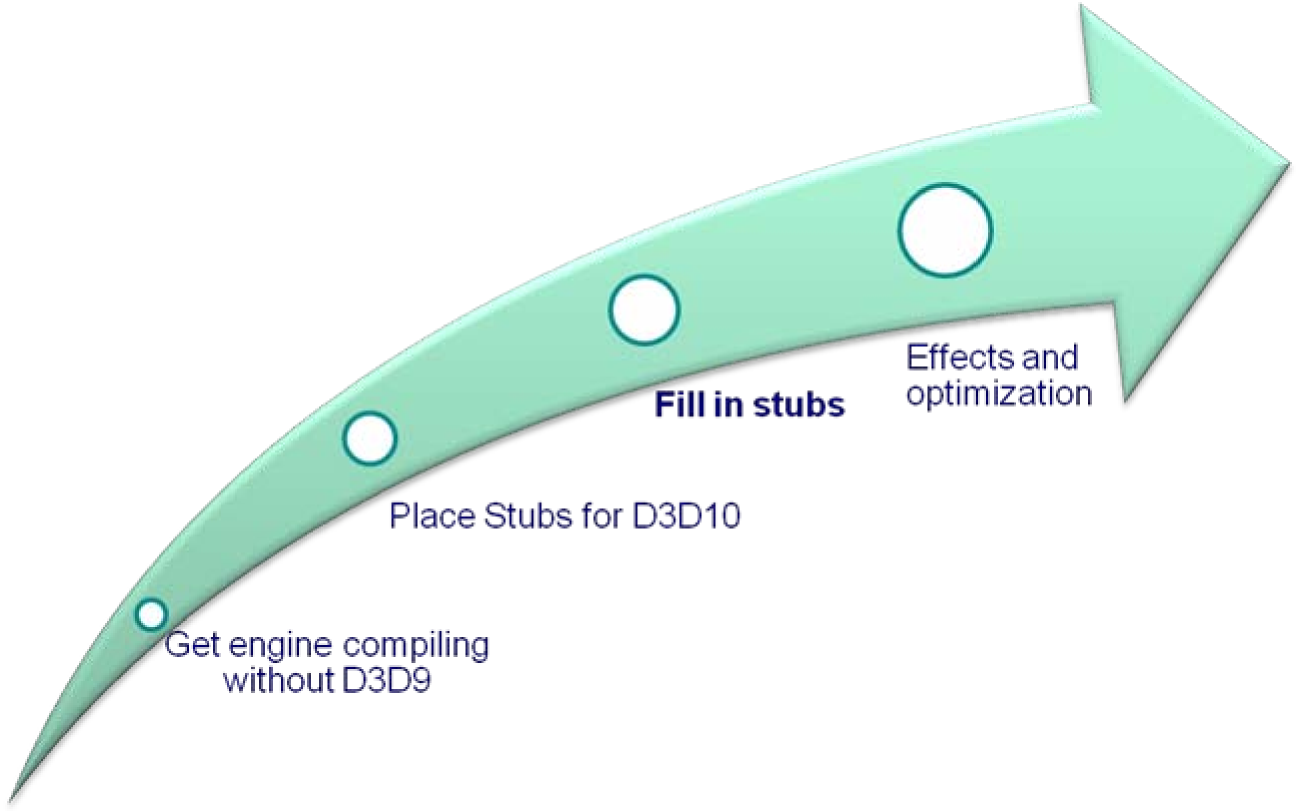
Hellgate Timeline



Stub Code

```
HRESULT dxC_Create2DTexture(…)
{
#ifdef ENGINE_TARGET_DX9

        return D3DXCreateTexture(…);

#elif defined(ENGINE_TARGET_DX10)

       ASSERTX( 0, "D3D10 TODO: Implement texture creation" );
       return E_FAIL;

#endif
}
```

Hellgate Timeline



Annoyingly Similar

- Many common structs are similar but not

```
typedef struct D3DLOCKED_RECT {
        INT Pitch;
        void * pBits;
        }

typedef struct D3D10_MAPPED_TEXTURE2D {
        void *pData;
        UINT RowPitch;
        }
```

- Same usage in both APIs

Annoyingly Similar

- Hellgate remaps member names
- Allows old API syntax with new structs

```
struct D3DLOCKED_RECT : public D3D10_MAPPED_TEXTURE2D
{
public:
          remapperVar<UINT> Pitch;
          remapperVar<void*> pBits;
          D3DLOCKED_RECT()
          {
                    Pitch.set( &RowPitch );
                    pBits.set( &pData );
          }
};
…
D3DLOCKED_RECT rect;
rect.pBits = NULL; //pBits points to pData using operator overloading
```

Remapper Class

```
template <typename mappedType> class remapperVar
{
private:
        mappedType* hidden;
public:
        void set( mappedType* map )
        {
                hidden = map;
        }
        mappedType operator=( const mappedType &input )
        {
                return *hidden = input;
        }
        operator const mappedType() const
        {
                return *hidden;
        }
};
```

Annoyingly Similar

- Hellgate remaps member names
- Allows old API syntax with new structs

```
struct D3DLOCKED_RECT : public D3D10_MAPPED_TEXTURE2D
{
public:
          remapperVar<UINT> Pitch;
          remapperVar<void*> pBits;
          D3DLOCKED_RECT()
          {
                    Pitch.set( &RowPitch );
                    pBits.set( &pData );
          }
};
…
D3DLOCKED_RECT rect;
rect.pBits = NULL; //pBits points to pData using operator overloading
```

Fixed Function Emulation

- Hellgate – Emulated for UI
- Don’t bother
    - Just move over to HLSL
- If you must
    - The MS SDK has a good example

Hellgate State Management

- Hybrid state control
  - Blend mode in the FX File
  - Setrenderstate for cullmode
- Most controlled from application
  - Using ID3DXEffectStateManager
- Problem
  - D3D10 Must use stateblocks

First Idea

- Emulate DX9 Behavior
  - Fastest path to something running
- Effect stateblocks with global variables
  - Straightforward
  - Potential for setting state in effect and engine

First Idea

```
//Blend state
shared cbuffer cbBlend{
        bool g_bAlphaToCoverageEnable;
        bool g_bBlendEnable[8];
        int g_iSrcBlend;
        ...

shared BlendState GlobalBlend{
        AlphaToCoverageEnable = g_bAlphaToCoverageEnable;
        BlendEnable[0] = g_bBlendEnable[0];
        BlendEnable[1] = g_bBlendEnable[1];
        ...

// To use global blend state
// SetBlendState(GlobalBlend, g_fBlendFactor, g_iSampleMask);
```

First Idea - Problems

- /Gdp disable performance mode
- Per batch constant updates
    - ~60% of these weren’t updated per batch
- Doesn’t really get you anything
    - Just construct state blocks in engine

Second Idea

- All state set in technique
- Works well if *all* state is defined in shader
    - Engine control allowed with globals
        - Engine must set global always
- State blocks *defined* for DX9

Second Idea

```
#ifdef ENGINE_TARGET_DX10
      #define DXC_BLEND_RGB_SRCALPHA_INVSRCALPHA_ADD \
       SetBlendState( dx10block, … )
#else
      #define DXC_BLEND_RGB_SRCALPHA_INVSRCALPHA_ADD \
      AlphaBlendEnable = TRUE; \
      COLORWRITEENABLE = BLUE | GREEN | RED; \
      BlendOp = ADD; \
      SrcBlend = SRCALPHA ; \
      DestBlend = INVSRCALPHA );

#endif
```

Second Idea - Problems

- Hellgate sets some state from engine
- Required touching more of the engine
- Defining DX9 blocks was painful
    - A lot of nasty macro work

Final Idea

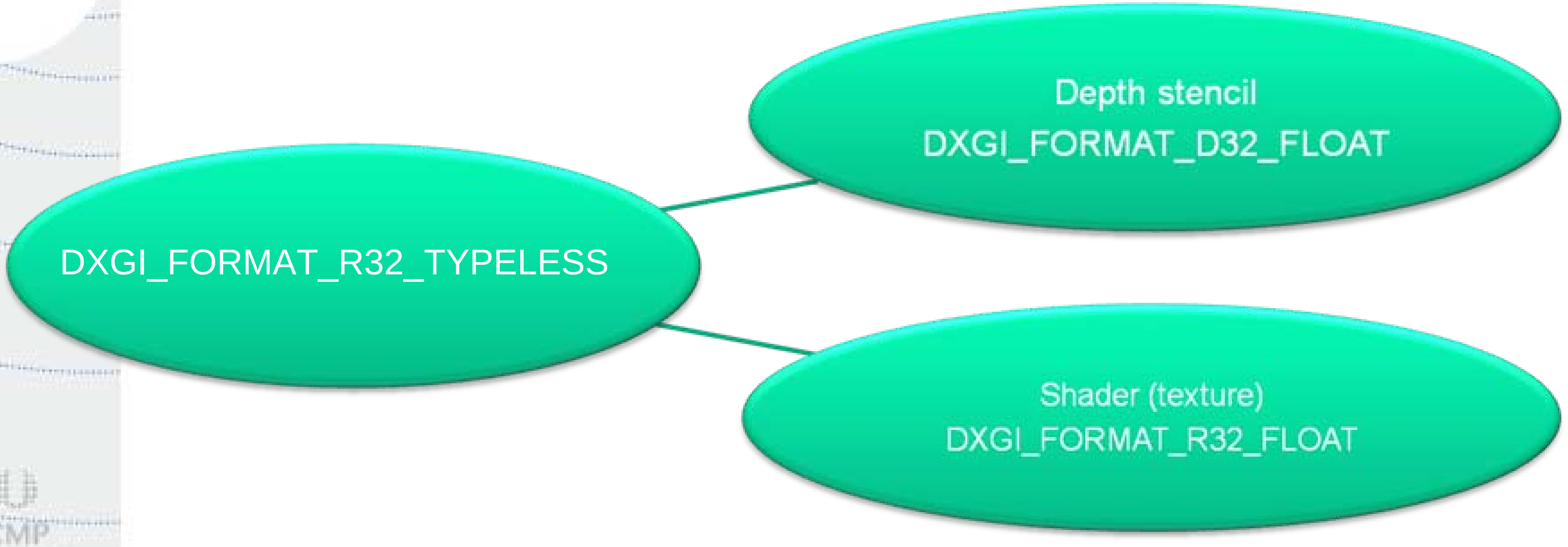
- Just construct stateblocks in engine
- SetRenderState call collects engine calls
- Constructs state blocks on the fly
  - Hash ensures created once
  - D3D10 won't duplicate anyways
- Improvements
  - Profile hash see how many permutations we need
    - Only 8 or so blend states

Statemanager

- ID3DXEffectStateManager
  - Allows engine to filter state calls
- WARNING: Not in D3DX10
- Implement our own D3D10StateManager
  - Collects engine state calls

Formats and Resource Views

- Most resources are typeless
- Not a simple DX9 type == DX10 type
  - Usage defines type



Formats and Resource Views

```
struct D3D_BASIC_TEXTURE
{
…
#ifdef ENGINE_TARGET_DX9
        SPD3DCTEXTURE2D          pD3DTexture;
#elif defined( ENGINE_TARGET_DX10 )
        SPD3DCTEXTURE2D          pD3DTexture;
        SPD3DCSHADERRESOURCEVIEW  pD3D10ShaderResourceView;
#endif

        DX9_BLOCK( SPD3DCTEXTURE2D )
        DX10_BLOCK( SPD3DCSHADERRESOURCEVIEW )
        GetShaderResourceView(UINT iLevel = 0)
        {
#ifdef ENGINE_TARGET_DX9
          return pD3DTexture;
#elif defined(ENGINE_TARGET_DX10)
          if( !pD3D10ShaderResourceView && pD3DTexture)
           CreateSRVFromTex2D( pD3DTexture, &pD3D10ShaderResourceView );

          return pD3D10ShaderResourceView;
        }
#endif
…
```

Formats and Resource Views

- Cubemaps are texture arrays in Dx10
  - Texture array support needed off the bat
- Vertex formats
  - `D3DDECLTYPE_SHORT2 == DXGI_FORMAT_R16G16_SINT`
  - No integral float conversion

Resources

- D3D9 managed resources virtualized
- D3D10 ***all*** resources virtualized
- Historically POOL associated with usage
  - D3D10 everyone shares the “POOL”
- Usage and CPU Access
  - Helps runtime and GPU

What’s my usage? D3D9

- D3D9 MANAGED
  - Don’t have to worry about restoring buffers
  - CPU copy == perf advantage w/ many updates
- D3D9 DEFAULT
  - Destroyed on device reset
  - Live only in vidmem
  - | DYNAMIC
    - GPU keeps data close
    - Dynamic means we plan to update often

What’s my usage? D3D10

- STAGING
    - CPU Copy used to update GPU
- DYNAMIC
    - Lots of updates (Mappable or lockable)
- DEFAULT
    - Few updates (Not-Mappable)
- IMMUTABLE
    - No Updates

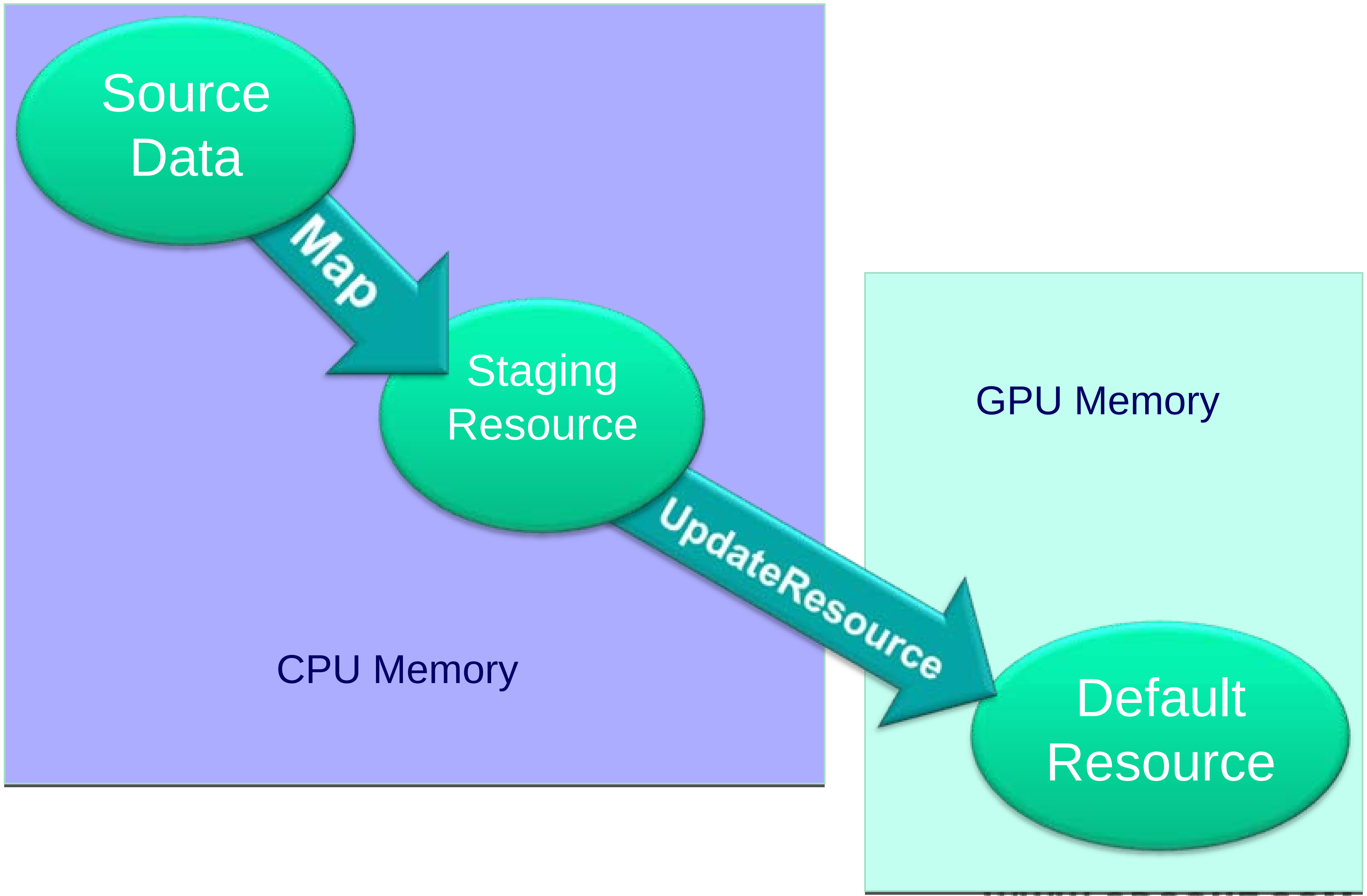
What’s my buffer usage?

- In Hellgate most buffers managed yet static
  - D3D10 usage == DEFAULT
- Particles and UI default dynamic
  - D3D10 usage == DYNAMIC
- Some caveats to choices
  - DEFAULT means no Map (lock)
  - DYNAMIC means no UpdateSubResource

What's my texture usage?

- Hellgate most textures are managed yet static
  - Usage DEFAULT
- D3D9 Lock is the primary update tool
  - Hellgate had lots of locking to update textures
    - Locks don't occur during runtime
    - Want usage DEFAULT for perf
  - Solution use staging buffer

Updating A Default Resource



Hellgate Staging Buffer

```
#ifdef ENGINE_TARGET_DX10
struct D3D_MIRRORED_TEXTURE
{
            SPD3DCTEXTURE2D pGPUTexture;
            SPD3DCTEXTURE2D pCPUTexture;

            D3D_MIRRORED_TEXTURE( const LPD3DCTEXTURE2D& input )
            {
                        pGPUTexture = input;
                        pCPUTexture = NULL;
            }

…

            LPD3DCTEXTURE2D operator =( const LPD3DCTEXTURE2D& input )
            {
                        return pGPUTexture = input;
            }

            LPD3DCTEXTURE2D* operator &()
            {
                        return &pGPUTexture.p;
            }
…
};
#elif defined( ENGINE_TARGET_DX9)
typedef LPD3DCTEXTURE2D D3D_MIRRORED_TEXTURE;
#endif
```

Hellgate Staging Buffer

```
HRESULT dxC_MapManagedTexture( D3D_MIRRORED_TEXTURE& … )
{
        if( !pD3DTexture.pCPUTexture )
        {
                dx10_CreateCPUTex( &pD3DTexture);
        }

        pD3DTexture.pCPUTexture->Map(…);

        return S_OK;
}
```

Making Usage Work

- Creation functions take DXC_USAGE enum
  - **`DXC_USAGE_RENDER_TARGET`**
  - **`DXC_USAGE_BUFFER_DEFAULT`**
- API code can decipher
  - D3D9
    - **`dx9_GetPool( DXC_USAGE usage )`**
    - **`dx9_GetUsage(…)`**
  - D3D10
    - **`dx10_GetBindFlags(…)`**
    - **`dx10_GetCPUAccess(…)`**
    - **`dx10_GetUsage(…)`**

Getting Usage D3D9

```
D3DPOOL dx9_GetPool( D3DC_USAGE usage )
{
       switch( usage )
       {
       case D3DC_USAGE_2DTEX:
          return D3DPOOL_MANAGED; break;
       …
       }
}
DWORD dx9_GetUsage( D3DC_USAGE usage )
{
       switch( usage )
       {
       case D3DC_USAGE_2DTEX:
          return 0x00000000; break;
       …
       }
}
```

Getting Usage D3D10

```
D3D10_USAGE dx10_GetUsage( D3DC_USAGE usage )
{
        switch( usage )
        {
        case D3DC_USAGE_2DTEX:
           return D3D10_USAGE_DYNAMIC; break;
        …
        }
}
D3D10_BIND_FLAG dx10_GetBindFlags( D3DC_USAGE usage )
{
        switch( usage )
        {
        case D3DC_USAGE_2DTEX:
           return D3D10_BIND_SHADER_RESOURCE; break;
        …
        }
}
```

Getting Usage D3D10

```
D3D10_CPU_ACCESS_FLAG dx10_GetCpuAccess( D3DC_USAGE usage )
{
        switch( usage )
        {
        case D3DC_USAGE_2DTEX:
           return D3D10_CPU_ACCESS_WRITE; break;
        …
        }
}
```

- CPU Access pre-declared here
  - Must specify if you need READ

Constant Buffers - Hellgate

- Allow FX system to handle CB usage
- Split CBs based on update frequency
    - CBs are shared through FX pool

```
shared cbuffer changing
{
    float4x4 mWorld;
    float4x4 mView;
    …
}

shared cbuffer permanent
{
    float4x4 mProj;
       …
}
```

FX Pool

- More robust than D3D9
  - Master “effect”
  - Pool can be accessed as FX
- Hellgate keeps CBs and shared samplers

Note about Constant Buffers

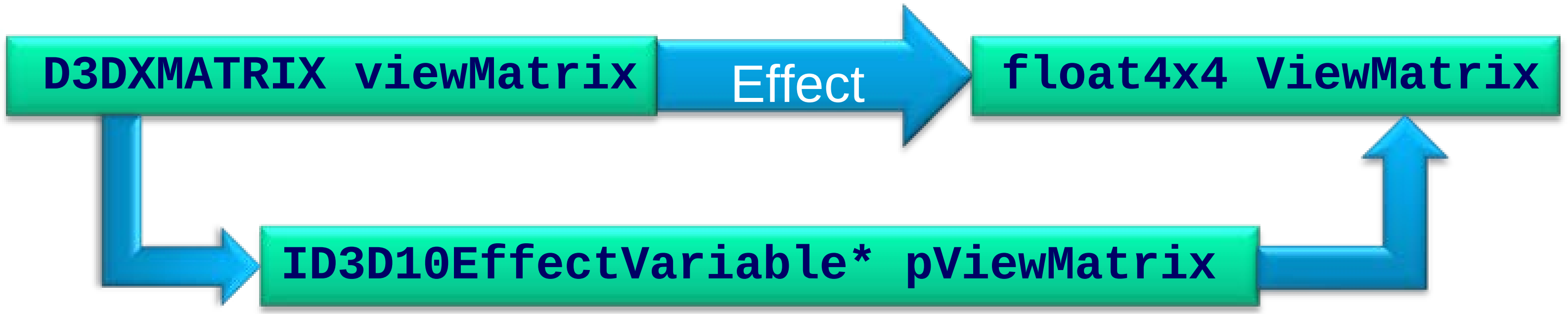
- Packing is on a float4 basis
  - Adjacent two float2's -> Single float4
  - "Packing Rules for Constant Variables"
- Example

```
float1 val1;
float1 val2;  // This is packed with the previous
float3 val3;  // This starts a new vector
```

FX Variable Handles

- D3D9 used strings to access by name
  - D3DXHANDLE
- D3D10 uses proper interfaces
  - ID3D10EffectVariable
    - D3D10EffectScalarVariable
    - D3D10EffectMatrixVariable
    - …

ID3D10EffectVariable



D3D9 Way

```
pEffect->SetMatrix(“ViewMatrix”, &viewMatrix );
```

D3D10 Way

```
ID3D10EffectVariable* pViewMatrix =
pEffect->GetVariableByName(“ViewMatrix”);
pViewMatrix->AsMatrix()->SetMatrix( &viewMatrix );
```

Hellgate Param

```
#ifdef INCLUDE_EFFECTPARAM_ENUM
#define DEFINE_PARAM(codename,fxname) codename,
#else
#define DEFINE_PARAM(codename,fxname) fxname,
#endif


DEFINE_PARAM( EFFECT_PARAM_SPECULAR, "gbSpecular“ )
DEFINE_PARAM( EFFECT_PARAM_NORMALMAP, "gbNormalMap" )
…
```

- Params defined in header
  - Run through all definitions
  - If variable is used by shader save interface

IsParameterUsed

- Hellgate needed IsParameterUsed (Dx9)
- Shader reflection gives us this
    - Check all constant buffers
    - Check all resource views

ID3D10ShaderReflection

- Allows inspection of FX Code
- Created from shader byte code
    - - Input structures (VS/GS/PS)
    - - CB usage
    - - SRV usage

Using The Reflection API

```
float4x4 shadMatrix;

Texture2D DiffuseTex;

struct PS_Input
{
        float4 pos : SV_Position;
        float2 tex : TEXCOORD0;
};

float4 SimplePS( PS_Input input ) : SV_Target0
{
   float3 shadTex = mul( input.worldPos, shadMatrix );
   return DiffuseTex.SampleCmp( PCFSampler, shadTex );
}
```

$$\begin{tikzcd}\text{ID3D10VertexShader* pVertexShader} \arrow[r, "\text{GetShaderDesc}"] & \text{D3D10\_EFFECT\_SHADER\_DESC}\end{tikzcd}$$

Using The Reflection API

```
float4x4 shadMatrix;

Texture2D DiffuseTex;

struct PS_Input
{
        float4 pos : SV_Position;
        float2 tex : TEXCOORD0;
};

float4 SimplePS( PS_Input input ) : SV_Target0
{
   float3 shadTex = mul( input.worldPos, shadMatrix );
   return DiffuseTex.SampleCmp( PCFSampler, shadTex );
}
```

D3D10_EFFECT_SHADER_DESC

D3D10ReflectShader

Using The Reflection API

```
float4x4 shadMatrix;

Texture2D DiffuseTex;

struct PS_Input
{
        float4 pos : SV_Position;
        float2 tex : TEXCOORD0;
};

float4 SimplePS( PS_Input input ) : SV_Target0
{
   float3 shadTex = mul( input.worldPos, shadMatrix );
   return DiffuseTex.SampleCmp( PCFSampler, shadTex );
}
```

```
ID3D10ShaderReflection*
pShaderReflection
```

`GetInputParameterDesc`

`GetResourceBindingDesc`

`GetConstantBufferByIndex`

Code to get ID3D10ShaderReflection

```
D3D10_PASS_SHADER_DESC passShaderDesc;
ID3D10ShaderReflection* pReflect = NULL;
pEffectPass->GetVertexShaderDesc( &passShaderDesc );
D3D10_EFFECT_SHADER_DESC shadDesc;
passShaderDesc.pShaderVariable->GetShaderDesc( 0, &shadDesc );
//shadDesc.pBytecode ← Byte code is here
```

Hellgate Shaders

- Using FX and SM 3.0
- /Gec Compatibility mode works!
- Technique – deprecated
  - Technique10 – use macro for DX10
  - Remove any state setting
  - Compile syntax is different

New technique

```
technique CrapTasticD3D9
{
    pass p0
    {
        VertexShader = compile vs_3_0 VS();
        PixelShader = compile ps_3_0 PS();
    }
}

Technique10 DaBombD3D10
{
    pass p0
    {
        SetVertexShader( CompileShader( vs_4_0, VS()));
        SetGeometryShader( NULL );
        SetPixelShader( CompileShader( ps_4_0, PS()));
    }
}
```

Samplers and textures

- /Gec Shaders use old syntax
  - Sampler has to be declared in FX file
    - Warning: Problem if using runtime to set sampler
- D3D10 Decouples sampler from texture
  - `myTexture.Sample( PT_SMPLR, texCoor )`
- D3D10 Shaders use new syntax

Cross API Sampler

```
sampler LightMapSampler
{
        Texture = (tLightMap);

#ifdef ENGINE_TARGET_DX10
        Filter = MIN_MAG_MIP_LINEAR;
#else
        Filter = Linear;
#endif
        AddressU = CLAMP;
        AddressV = CLAMP;
};
```

Input Assembler Objects

- AKA Vertex Declarations
- Creation requires signature from Vshader

  Must be created after FX

  Can be reused with other shaders

  - Creation signature must match other shader

  Warning: IA Must be created with complete sig for reuse

Matching Signatures

```
struct VS_Input
{
    float4 pos : SV_Position;
    float2 tex : TEXCOORD;
};
```

```
struct VS_InputNormal
{
    float4 pos : SV_Position;
    float2 tex : TEXCOORD;
    float3 norm : NORMAL;
};
```

```
const D3D10_INPUT_ELEMENT_DESC layout[] =
{
  { " SV_Position ",  0, DXGI_FORMAT_R32G32B32_FLOAT,…},
  { " TEXCOORD ",    0, DXGI_FORMAT_R32G32_FLOAT, …},
  { " NORMAL ",  0, DXGI_FORMAT_R32G32B32_FLOAT, … },
};
```

Matching Signatures

```
struct VS_Input
{
    float4 pos : SV_Position;
    float2 tex : TEXCOORD;
};
```

```
struct VS_InputNormal
{
    float4 pos : SV_Position;
    float2 tex : TEXCOORD;
    float3 norm : NORMAL;
};
```

```
const D3D10_INPUT_ELEMENT_DESC layout[] =
{
  { " SV_Position ",  0, DXGI_FORMAT_R32G32B32_FLOAT,…},
  { " TEXCOORD ",    0, DXGI_FORMAT_R32G32_FLOAT, …},
  { " NORMAL ",  0, DXGI_FORMAT_R32G32B32_FLOAT, … },
};
```

Input Assembler Objects - Tip

- Pain to match up between Dx9/Dx10/FX
- Ended up with single header for all three

```
#define FLOAT2_32 DECLARE_FORMAT \
( float2, D3DDECLTYPE_FLOAT2, DXGI_FORMAT_R32G32_FLOAT, D3DXVECTOR2 )
```

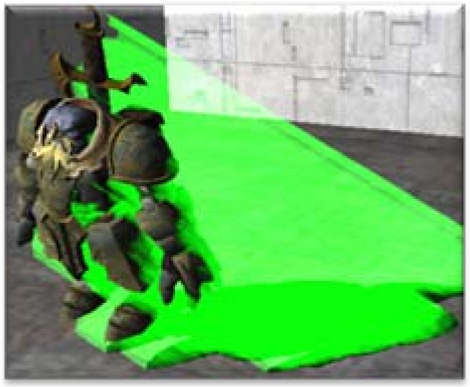
- Formats declared for all three
  - Final format used to get size/offset
- …or use ShaderReflection

Making Friends with GS

- GS - Geometry Shader
  - New pipeline stage
  - Per primitive shader
  - Limited data expansion
- Powerful when properly used
  - Respect the Max Output Decl
    - Static allocation – crucial to performance
- D3D10_QUERY_SO_STATISTICS
  - Predict real usage
  - Often lower than you think

GS Data Expansion

- Useful for point sprites
  - Points->Tris
- SSVs
  - GPU Extrusion



Optimize Output

- Max 1024 floats
  - 512 better, 256 better…
- Optimize for minimal output

```
//Typical VS structure
//36 floats for a sprite
struct VS_OUTPUT
{
		float4 pos : SV_Position;
		float2 tex : TEXCOORD0;
		float3 norm : NORMAL
};
```

```
//Packed GS struct
//32 floats for a sprite
struct VS_OUTPUT
{
		float4 pos : SV_Position;
		float2 tex : TEXCOORD0;
		float2 norm : NORMAL;
};
```

Hellgate GS

- Process silhouette
  - 1 Triangle -> 3 points
- Simulate
  - 1 Point -> 1 Point
- Draw
  - 1 Point -> 2 Triangles (4 vertices)

Instancing

- API greatly simplified in D3D10
  - Now “built in” to all draw calls
- New input classification for IA
  - D3D10_INPUT_PER_INSTANCE_DATA
- New system generated value for instancing SV_InstanceID

SV_InstanceID

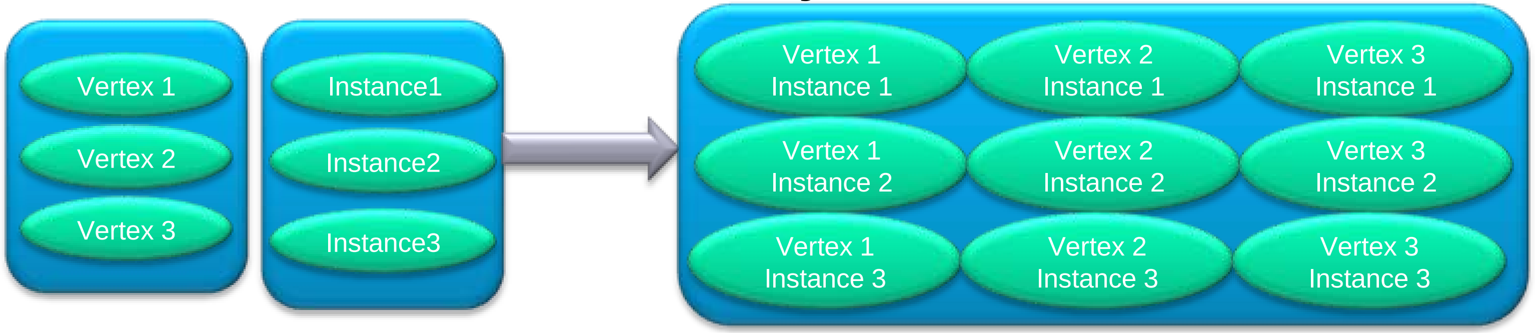
- System value available all the time
  - Even when not drawing instanced
- Monotonically increasing per instance
  - Starts at 0
- Resets every batch
  - Draw, DrawInstanced, DrawIndexed, DrawIndexedInstanced, DrawAuto
- Very handy to index into constant mem
  - Or generate VTF address

Instancing

- Only goal of Instancing
  - ***Reduce CPU load and bandwidth***
    - By allowing the GPU to iterate over like objects
- Little difference between
  - 5000 objects with 1 call
  - 5000 objects with 25 calls
    - *Still better than 5000 calls!!*
- Indexing into constant memory is faster than reading per instance attributes from second stream… but why?

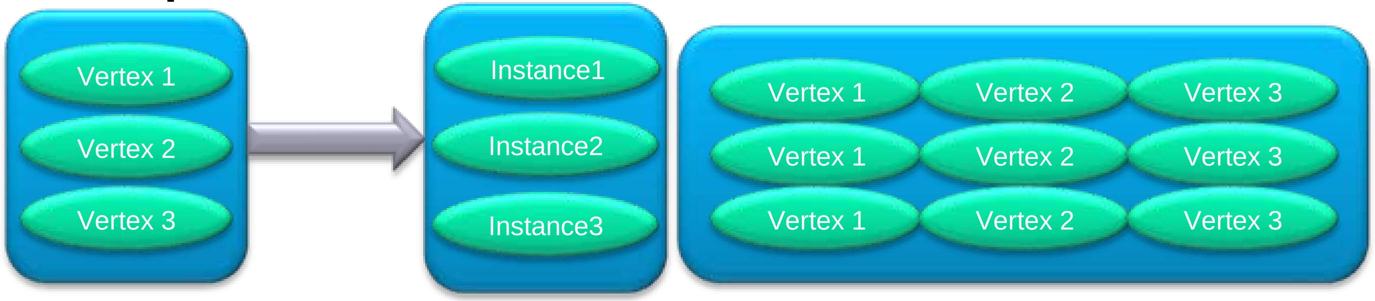
2 Stream Instancing

- Adding a 2nd stream of per-instanced attributes bloats the vertex size
  - Each vertex contains redunant instance information
- Unlimited # of instances per draw
- Lower cache efficiency



Constants Instancing

- SV_InstanceID to index into banks of constants
- Store instance data using FX structs in constants.
- # of max instances per constant buffer depends on instance data



Example of Constants Instancing

- Using the following per instance data

```
struct PerInstanceData
{
  float4 world1;
  float4 world2;
  float4 world3;  // Translation encoded in the .w components
  float4 color;
};

cbuffer cInstanceData
{
  PerInstanceData g_Instances[MAX_INSTANCE_CONSTANTS];
}
```

- Can store 4096 / 4 = 1024 in a single buffer.
- 10K instances in 10 draw calls. Not bad!

Debugging Tips

- Debug device enabled with creation flag
  - D3D10_CREATE_DEVICE_DEBUG
- ID3D10Debug
  - SetFeatureMask
    - D3D10_DEBUG_FEATURE_FINISH_PER_RENDER_OP
      - App waits for GPU to finish rendering
    - Helpful for tracking down that rare driver bug
  - Validate
    - Check that current device state works

Debugging Tips

- ID3D10InfoQueue
  - PushStorageFilter
    - Filter debug Spam

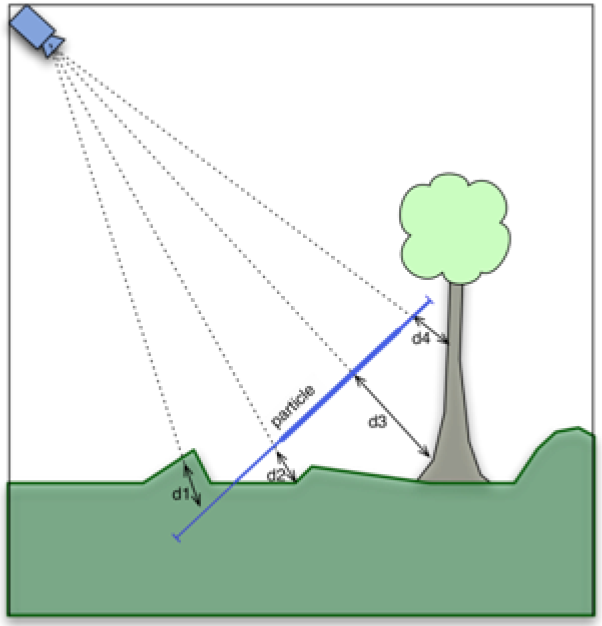
```
ID3D10InfoQueue* pD3D10InfoQue;
…
pd3dDevice->QueryInterface( __uuidof( ID3D10InfoQueue ),
reinterpret_cast<void**>(&pD3D10InfoQue) )
D3D10_INFO_QUEUE_FILTER filtList;
ZeroMemory( &filtList, sizeof( D3D10_INFO_QUEUE_FILTER ) );
D3D10_MESSAGE_ID idList[…] =  {…};
filtList.DenyList.NumIDs = …;
filtList.DenyList.pIDList = idList;
pD3D10InfoQue->PushStorageFilter( &filtList );
```

Hellgate Effects

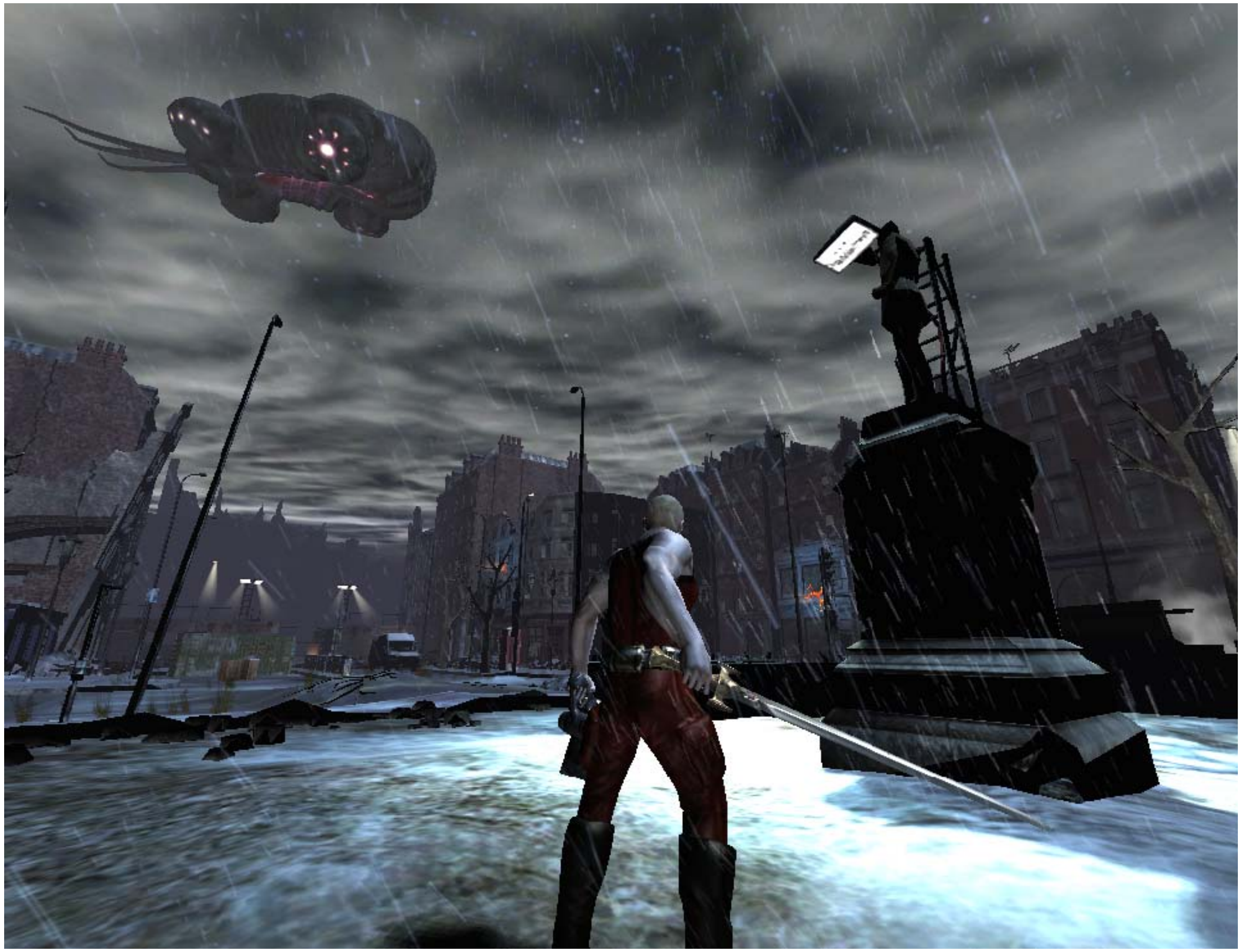
- Effects implemented in Game

Hellgate Soft Particles

- Avoid particle opaque intersection
- D3D10
    - Sample zBuffer



Hellgate Rain



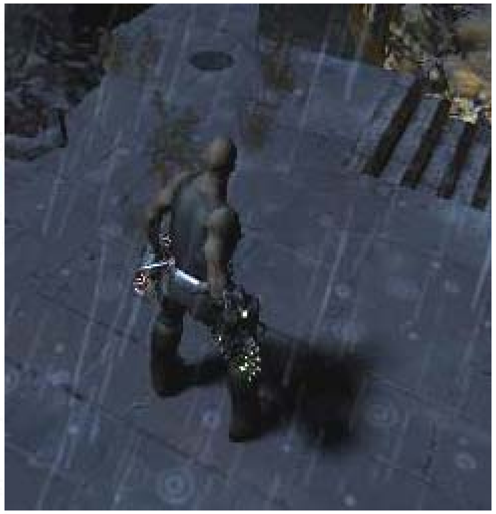
Hellgate Rain

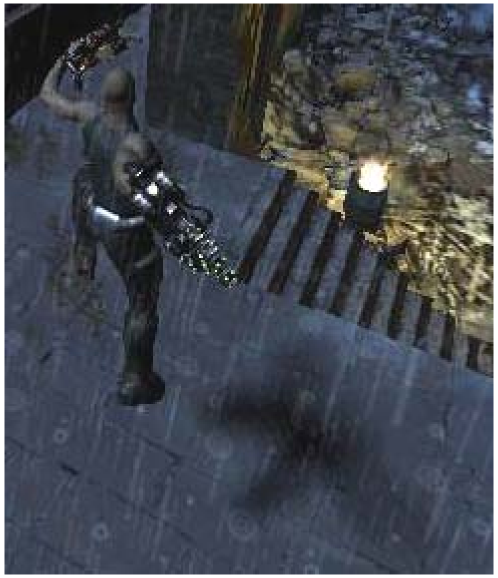
- GPU Rain simulation
- Lit by directional lights
- D3D10
  - Uses GS
  - Uses texture arrays

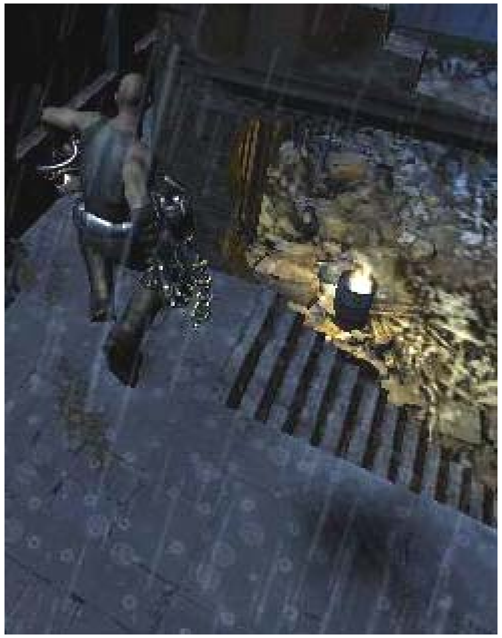
Hellgate Rain

- Texture array database
  - Indexed by 2 angles
    - Light
    - Eye

Hellgate PCSS





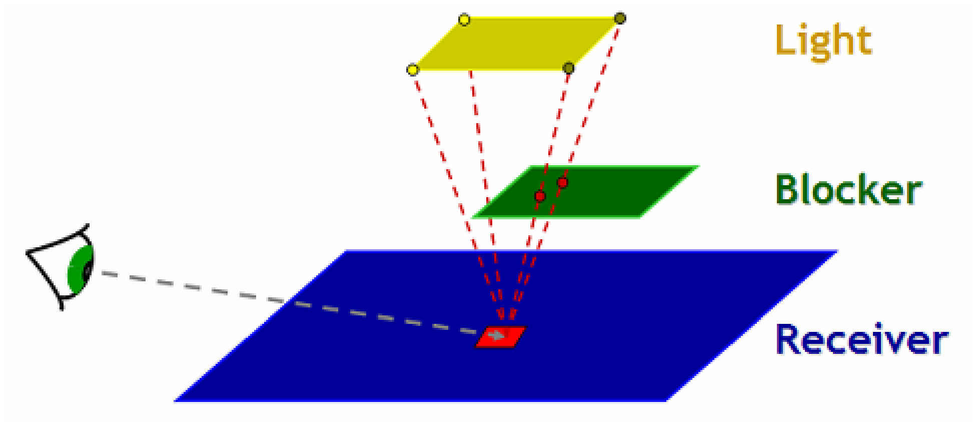


Hellgate PCSS

- Randy Fernando NV 2005
- Perceptually accurate soft shadows
- D3D10
    - Uses direct zBuffer access
    - Uses point and PCF filtering

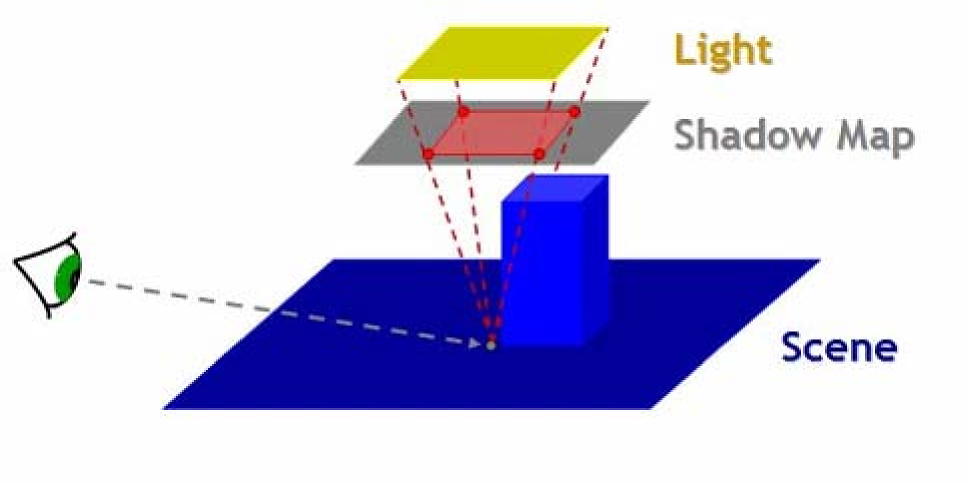
Hellgate PCSS

- Blocker search uses point sampling

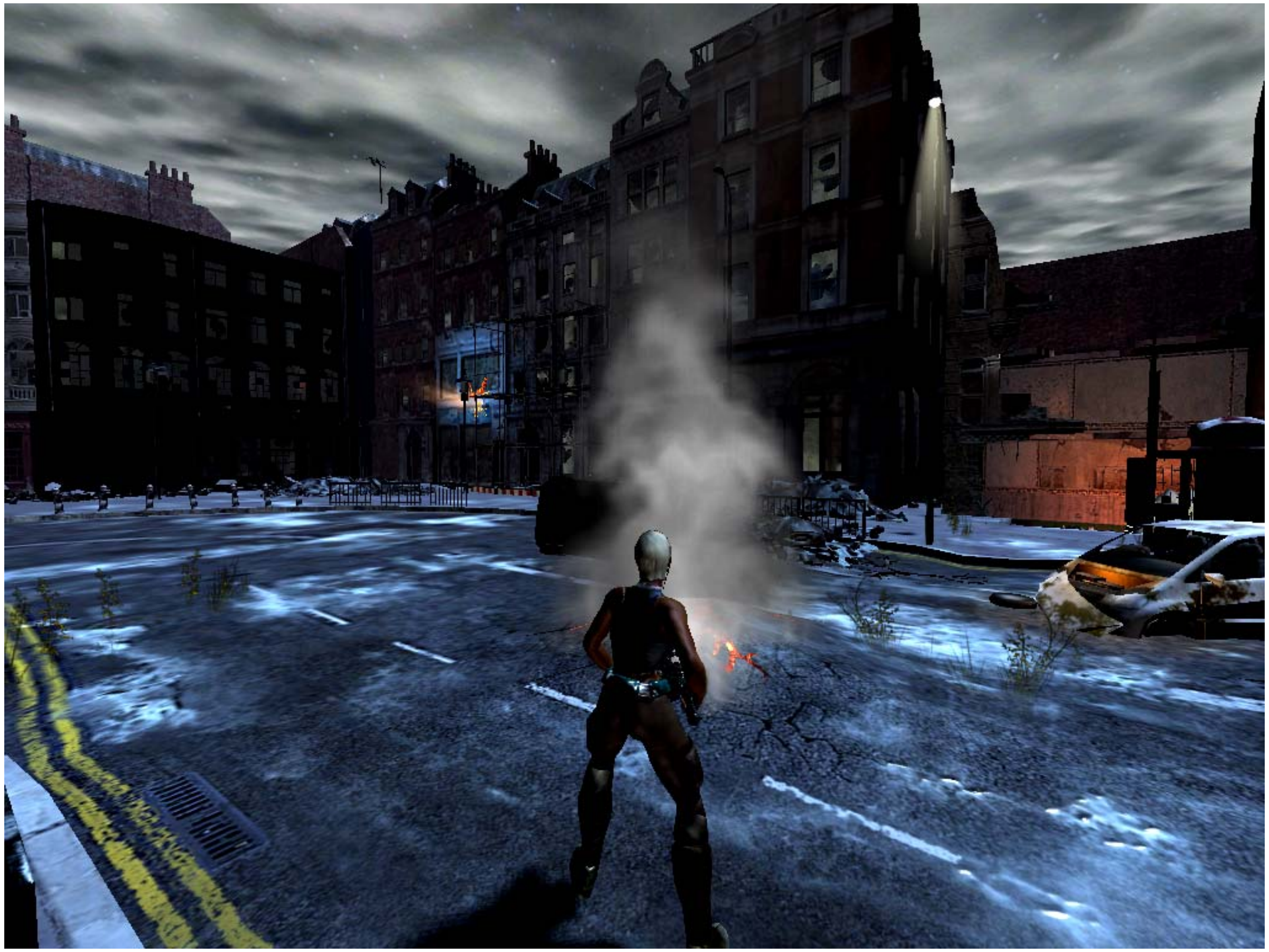


Hellgate PCSS

- Penumbra estimate uses PCF



Hellgate Smoke



Hellgate - Smoke

- Volumetric smoke
- Smoke simulation reacts to character
- D3D10
    - Uses render to 3D texture
    - Read zBuffer
- Sarah Tariq explains all at 5 PM today

Conclusion

- Think about state management
    - Where is my state set?
    - Can I make D3D9 use blocks?
- Nail down usage
- Don't be afraid to recycle your shaders
- Use the Reflection API

Kevin Myers
NVIDIA
kmyers@nvidia.com