How to Work on Next Gen Effects Now: Bridging DX10 and DX9

Guennadi Riguer
ATI Technologies
Overview

- New pipeline and new cool things
- Simulating some DX10 features in DX9
- Experimental techniques
Why This Talk?

- DX10 – really powerful toolset
- No good way to start experimenting
  - HW isn’t quite ready yet
  - RefRast is too slow, not representative of real HW performance
- Need a way to start experimenting right now to be ready when HW comes
  - Hopefully will make you think about new cool techniques
New Pipeline

- Orthogonalized FB
- Geometry shader
- Stream out
Orthogonalized FB

- All resources are just memory arrays
- Multiple “views” provide attachment points to different stages in the pipeline
- Can re-interpret resource data
- Allows recirculation of data
- Texture fetch in VS, GS and PS
Constant Buffers

- Much bigger constant store
- Faster constant updates
- Ability to group consts into multiple buffers for more efficient updates
- No more DX9 limits!
New Cool Things You Can Do

- More efficient multi-pass rendering
  E.g. animate once, render many times
- Displacement mapping
- Neat tricks with re-interpreting data
- GPGPU driven effects
Geometry Shader

- Operates on primitives
- Can change topology
  - Generate new primitives
  - Remove existing primitives
- Can have access to adjacency information

Triangle with adjacency
Stream Out

- Allows outputting GS results
  - Works with variable number of output primitives
- All topologies are converted to lists on output
- Alternative to render to texture/VB
New Cool Things You Can Do

- Compute per-primitive entities
  - E.g. face normals
- Shadow volume extrusion
- Not really designed for high-performance tessellation, but…
Simulating DX10 with DX9

- Can’t really simulate API
- Only some of the features can be used
- Useful to try high-level concepts
- Useful for “guesstimating” DX10 performance
- Faster than RefRast
Simulating DX10 with DX9

- Idea: re-circulate data
  Use multi-pass approach
- Can somewhat simulate GS + SO
  Can’t generate variable number of primitives
- Features available in DX9
  Vertex texture fetch (Nvidia)
  Render to VB (ATI)
Multi-pass with Vertex Texture Fetch
Render to VB

IB | VB
---|---
VS
RS
PS
Raster Backend

Text
Render Target
Depth/Stencil
Render to VB in DX10

- Multiple ways to produce output
  - Stream out
  - Render targets
- Ideal rendering method in DX10 – combination of GS, stream out and render to VB
Experimental Techniques

- Animation
- Shadow volume extrusion
- Tessellation
- Displacement mapping
- Sort on GPU
- Uber-shaders
- GPGPU
Animation

Problems in DX9

- Limited number of bones
  - Even worse when trying to use instancing

Constant uploads are expensive

Multiple animation shader execution for multi-pass rendering could be expensive
Animation

- Solution in DX10
  - Constant buffers provide large storage
  - Constant buffer updates should be cheaper
    - Could preload all animation data – no transfer
  - Stream out animated data for multi-pass
- Other possible improvements
  - Move animation blending to GPU
- Used by other techniques
Constant Buffers

- Fake with textures in DX9 for simulation
- Was used with render to VB animation

- DX9 VS constant update \(~400\) Mb/s
- DX9 dynamic texture update \(~2.5\) Gb/s
  - Updates batched into a large texture to reduce overhead of texture locking
Streamed Out Animation in DX10

- **Pass 1**
  - Draw list of points (verts) for animation and stream out

- **Pass 2**
  - Reinterpret animated verts with proper index information for final rendering
Streamed Out Animation in DX9

Fake stream out with render to VB
Render to VB
Animation Performance

- Matrix palette generation
  ~60-80 instructions per matrix
  Overall negligible performance impact

- Animation in PS
  ~80-100 instructions/vertex max depending on shader complexity
  Mostly texture fetch bound for the bone matrices
  90-275 Mvert/s (depends on number of bones)
Solving Batching Problem

- Also solves batching problems
- Can batch transformations from multiple objects
- Simulate DX10 texture arrays using texture atlases
- Demo renders up to 4096 objects in one draw call
Animation Example
Shadow Volume Extrusion

- **Problems in DX9**
  - Doesn’t work correctly with animated objects
  - Games are forced to perform animation and extrusion on CPU

- **Solution in DX10**
  - Perform animation on GPU, stream it out
  - Perform extrusion in GS, generate sides of shadow volume
Shadow Volume Extrusion in DX10

- Compute face normal in GS
- Leave polygons facing the light in place (front cap)
- Move back-facing polygons away from light (back cap)
- Create side quads to stitch front and back caps
Shadow Volume Extrusion in DX10

- Side filling quads
- Front cap
- Back cap

Light direction
Shadow Volume Extrusion in DX9

- Can’t generate unknown number of polygons with render to VB
- Solution: use degenerate quads for all edges
  - Some already use in DX9 for static objects
- Use separate pass to re-compute face normals after animation for the extrusion
Shadow Volume Extrusion in DX9

- Matrix palette
- Vertex data: Position, Normal, ...
- Animated model data
- Face indices for each polygon
- Face normal pass
- Face normal (stored for each vertex)
- Extrusion pass

Animation pass
Shadow Volume Extrusion Performance

- Assumptions for extrusion in DX10 (with GS)
  Animation – 60 instr/vertex
  Average extrusion in GS – 40-50 instr/tri

- Assumptions for extrusion in DX9
  Animation – 60 instr/vertex
  Face normal calculation – 20 instr/vertex
  Extrusion in VS – 10 instr/vertex

- Estimates: DX9 is ~20% slower than GS
  For 1K model with moderate vertex reuse
  Extra memory bandwidth isn’t taken into account
Tessellation

- Could use render to VB to generate fixed number of vertices/polygons
- Can simulate fixed level tessellation
- GS isn’t really designed for tessellation, but might be worth trying anyway
N-patches with Render to VB

ése Back to the future 😊
N-patches

- Cubic interpolation for the position
  - Uses triangular Bezier patch
- Quadratic interpolation for the normal
  - Different control mesh from position
- Linear interpolation for texture coordinates
N-patches

Original geometry

Triangular Bezier patch

Normal control points
DX9 Implementation Details

- Render quad with as many pixels as final number of vertices
- For each rendered pixel:
  - Fetch 3 original control points
  - Get barycentric coordinate of this vertex
  - Compute new position, normal and tex coord
- Use pre-computed barycentric coordinates to compute new positions, normals, etc.
  - Store in (N x 1) texture, N – verts per patch
- Finally render new mesh
N-patch Performance

- DX9 performance
  All control points are re-computed for every vertex – a bit wasteful
  Tessellation shader – 112 SM 3.0 instructions
  Tessellation rate – 60-80 Mverts/sec

- DX10 improvements
  GS removes redundant control point computations
Displacement Mapping

- Displace vertices with VTF
- Could use with or without tessellation
- Usage examples
  - Geometry compression technique
  - Procedural or dynamic displacement
  - Creation of unique objects
  - Intersection / collision detection
Geometry Compression

- Can create huge landscapes
  - Use really small footprint formats
- E.g. can store hightmap of California in 222Mb with 30 x 30 m resolution using BC4_UNORM format
  - 24:1 compression ratio
Dynamic Displacement

- Compute displacement/heightmap
- Re-compute normals in texture space
  Use Sobel filter (could be done with 4 bilinear fetches)

\[
\begin{array}{ccc}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1 \\
\end{array}
\quad
\begin{array}{ccc}
-1 & -2 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1 \\
\end{array}
\]
Dynamic Displacement
Snow Example

- “Physically correct” snow accumulation
  See Nvidia’s whitepaper and sample
- Making imprints in texture space
  Render parts of objects intersecting snow
  Blur imprints to create more natural slope
  Combine new imprints with previous layer of snow
- Varying depth of imprints based on snow depth
Automatic Object Placement

- Automatically put objects (trees, grass, etc.) on the terrain
  - Use displacement map to find vertical displacement
- Works with procedurally generated landscapes
- Works perfectly for objects where CPU doesn’t need to know real position
  - Just keep that data on GPU
Creating Unique Objects

- Make all your objects look different using the same mesh
- Displace basic mesh using collection of displacement maps
  
  E.g. no 2 trees in the forest are the same
Sorting on GPU

- Order independent transparency is a huge problem
- Could sort alpha blended particles (or other polygons, objects, etc.) on GPU
- Works perfectly with particle systems implemented on GPU
Bitonic Sort

- Very hardware friendly and has excellent parallelism
  - Can be implemented without recursion
- Performance: $\Theta(n \ (\log_2 n)^2)$
  - 16K particles at over 40 fps on X1900
- 2D texture used to store and sort more than 4096 particles
Bitonic Sort Example

3 1 7 2 4 9 8 5

1 3 7 2 4 9 8 5

1 2 3 7 9 8 5 4

1 2 3 4 5 7 8 9
2D Bitonic Sort

- Combination of vertical and horizontal sort passes

Last 4 sort passes
Implementation Tricks – Part 1

- Use 2 textures due to number of data components
  Position, distance, particle ID
- Compare-and-exchange operation
  Twice the workload
  Example:
  - Compare A and B, smaller goes into A
  - … then B and A, greater goes into B
Implementation Tricks – Part 2

- Use texture coordinate “magic” to select direction of pixel offset
- Use square wave with +1 and -1 values to drive direction of the offset and sign of comparison
Bitonic Sort Shader (VS)

```cpp
struct VsOut {
    float4 pos: POSITION;
    float4 texCoord: TEXCOORD0;
    float3 direction: TEXCOORD1;
};

float2 halfPixel, select;
float step0, step1, step2;

VsOut main(float4 pos: POSITION){
    VsOut Out;
    Out.pos = pos;
    Out.texCoord.xy = pos.xy * float2(0.5,-0.5) + 0.5 + halfPixel;
    Out.texCoord.zw = 0;
    // Select sample direction
    float r = dot(Out.texCoord, select);
    // Sample direction and frequencies
    Out.direction = float3(r*step0, r*step1, Out.texCoord.y*step2);
    return Out;
}
```
Bitonic Sort Shader (PS)

```c
struct PsOut {
    float4 pos: COLOR0; // particle position
    float4 dist: COLOR1; // distance used for sorting
};

sampler2D Tex;
sampler2D Dist;

// Distance between pixels to compare.
float4 offset;

PsOut main(float4 texCoord: TEXCOORD0, float3 direction: TEXCOORD1){
    PsOut Out;
    // Sample current value
    float4 d0 = tex2D(Dist, texCoord.xy);
    // dir.x = Sample and comparison direction.
    // dir.y = Horizontal comparison frequency.
    // dir.z = Vertical comparison frequency.
    float3 dir = (frac(direction) < 0.5)? 1 : -1;
    // Select other value to compare to
    float4 texCoord1 = texCoord + dir.x * offset;
    float4 d1 = tex2D(Dist, texCoord1.xy);
    ...
```

Game Developers Conference
Bitonic Sort Shader (PS)

...
Mixing Materials

- Often alpha blended particles have different textures or even materials
- In DX10 use texture arrays and uber-shaders to render all in one pass
  
  Flow control in uber-shader will have good coherency

- GS can even generate different geometry for different types of particles
Uber-shader Example

- Grass – static particles with different textures
- Fires – vertically rotating billboards with procedural fire (generated in PS)
- Smoke – camera oriented particles
Sorting Demo
GPGPU

- Physics on GPU allows keeping data on the card all the time
  - No extra data transfers
  - Free up CPU for more important tasks
- The best solution for GPU generated particle systems, etc.
  - E.g. smoke interaction with the ground
Questions