Render to Vertex Buffer with D3D9

Thorsten Scheuermann
Sr. Software Engineer
ATI Research
Outline

• Render to Vertex Buffer basics

• Example techniques
  – Skinned animation
  – Shadow volumes
  – Dynamic Displacement Mapping
  – Others

• Conclusion

HLSL Examples
Render To Vertex Buffer (R2VB)

- "Render to VB" = "Render to texture and re-interpret texture data as VB"
- Very general approach
  - Allows "aliasing" textures to VB and fetching 2D texture linearly
  - Can even alias data types
Data Recycling

Render to Texture

Render to Vertex Buffer

Vertex Texture Fetch

IB  VB

RS

VS

PS

Raster Backend

Render Target

Depth/Stencil

Tex

IB  VB

RS

VS

PS

Raster Backend

Render Target

Depth/Stencil

Tex

IB  VB

RS

PS

Raster Backend

Render Target

Depth/Stencil

Tex

SIGGRAPH 2006
R2VB in ATI’s D3D9 driver

• Check ATI SDK for details on enabling R2VB: http://www.ati.com/developer

• In a nutshell:
  – Create Render Target with D3DUSAGE_DMAP flag
  – Set stream source texture through DMAP sampler
  – Stride and offset set as usual with dummy VB
  – Enable R2VB settings through overloaded D3DRS_POINTSIZE render state
R2VB Hardware Support

• Supported for Radeon 9500 and up

• Radeon 9500 to Radeon X850:
  – Can bind a single R2VB buffer at a time

• Radeon X1000 series:
  – Supports binding up to five R2VB buffers simultaneously
Why R2VB?

- Process vertex data in pixel shaders
- Enables interesting class of new effects
- Efficient alternative to vertex texture fetch
- Prototype DX10-style algorithms on today’s hardware and API
  - Also for backwards-compatibility: DX10 → DX9
Why Perform Vertex Processing in Pixel Shaders?

• Radeon X1900: 48 Pixel shader ALUs, 8 Vertex shader ALUs
  – 6x computation power in PS

• Additional functionality in PS:
  – Fast filtered texture fetches from many texture formats
  – Use textures as very large constant buffers
  – Efficient dynamic branching

• Gives you an idea of what a Unified Shading Architecture will be like
Example 1: Animation
Matrix Palette Skinning

• Standard character animation technique for real-time graphics

\[
p_{\text{skinned}} = \sum_{i} w_i M_{\text{mat\_index}_i} p_{\text{base\_pose}}
\]

- \(w_i\): weights \(\sum w_i = 1\)
- \(M_i\): Transform matrices
- \(\text{mat\_index}_i\): Matrix indices for this vertex
float4x4 mBone[48];
float4x4 mVP;
float4x4 mTrans;

struct VsIn {
    float4 pos:         POSITION0;
    float3 normal:      NORMAL0;
    float2 texCoord:    TEXCOORD0;
    int4  boneIndices:  TEXCOORD1;
    float4 weights:     TANGENT;
};

struct VsOut {
    float4 pos:         POSITION;
    float2 texCoord:    TEXCOORD0;
    float3 normal:      TEXCOORD1;
};
VsOut main( VsIn inp )
{
    float4 tPos, rPos, rNormal;
    VsOut outp;

    rPos = mul(inp.pos, mBone[inp.boneIndices.x]).xyz * inp.weights.x;
    rPos += mul(inp.pos, mBone[inp.boneIndices.y]).xyz * inp.weights.y;
    rPos += mul(inp.pos, mBone[inp.boneIndices.z]).xyz * inp.weights.z;
    rPos += mul(inp.pos, mBone[inp.boneIndices.w]).xyz * inp.weights.w;
    rPos.w = 1.0f;

    [...]

    tPos = mul( mTrans, rPos );  // apply model transform
    outp.pos = mul( mVP, tPos );  // view-projection transform
    outp.normal = normalize( mul( mTrans, rNormal ).xyz );
    outp.texCoord = inp.texCoord;  // pass texture coordinates through

    return outp;
}
Animation Limitations in DX9

- Limited number of bones
  - Even worse when trying to use instancing
- Constant uploads are expensive
- Animation code in VS is executed for every render pass
Animation with R2VB

Pass 1: Animation Blending
- Animation sets
  - Matrix palette

Pass 2: Pixel Shader Skinning
- Vertex data
  - Position
  - Normal
  - ...
  - Animated model data

Pass 3: Model Rendering
- IB
- VB
- RS
sampler boneAnimation;
float3 time_interp;
float iBoneAnimationHeight;

float4 main( float2 t0: TEXCOORD0 ) : COLOR
{
  float4 a0, a1;
  float2 tc0, tc1;

  // get the four animation matrix elements of t0 frame.
  tc0 = float2( t0.x, time_interp.x * iBoneAnimationHeight );
  a0 = tex2D( boneAnimation, tc0 );

  // get the four animation matrix elements of t1 frame.
  tcl1 = float2( t0.x, time_interp.y * iBoneAnimationHeight );
  a1 = tex2D( boneAnimation, tcl1 );

  // the four animation matrix elements of current frame.
  return lerp(a0, a1, time_interp.z);
}
Pass 2: Skinning Pixel Shader (1)

```cpp
sampler skinningVertex;
sampler vertexBoneIndex;
sampler vertexWeight;
sampler boneMatrix;
float4 bias;

float4x4 getMatrix( float mi )
{
    float4x4 m;
    float4 xOff;

    xOff = mi.xxxx + bias;

    m[0] = tex2Dlod( boneMatrix, float4(xOff.x, 0, 0, 0) );
    m[1] = tex2Dlod( boneMatrix, float4(xOff.y, 0, 0, 0) );
    m[2] = tex2Dlod( boneMatrix, float4(xOff.z, 0, 0, 0) );
    m[3] = tex2Dlod( boneMatrix, float4(xOff.w, 0, 0, 0) );

    return m;
}
```
float4 main(float2 t0: TEXCOORD0) : COLOR {
    float4 outp;
    float4x4 M;

    float4 index  = tex2D( vertexBoneIndex, t0 ); // get bone index.
    float4 vertex = tex2D( skinningVertex, t0 ); // get vertex position or normal.
    float4 weight = tex2D( vertexWeight, t0 );    // get vertex weight.

    M = getMatrix( index.x ); // get bone matrix indexed by bone index 0.
    outp.xyz = mul( vertex, M ).xyz * weight.x;

    for( int i = 1; i < 4; i++ ) {
        if( weight[i] > 0.0 ) {
            M = getMatrix( index[i] ); // get matrix indexed by bone index i.
            outp.xyz += mul( vertex, M ).xyz * weight[i];
        }
    }
    outp.w = 1.0f;

    return outp;
}
VsOut main( VsIn inp )
{
    float4 tPos, rPos, rNormal;
    VsOut outp;

    // input position and normals are already skinned
    tPos = mul( mTrans, inp.pos ); // apply model transform
    outp.pos = mul( mVP, tPos );  // view-projection transform
    outp.normal = normalize( mul( mTrans, inp.normal ).xyz );
    outp.texCoord = inp.texCoord; // pass texture coordinates through

    return outp;
}
R2VB Animation Performance

- Matrix palette generation
  - ~60-80 instructions per matrix
  - Negligible performance impact due to small input data size

- 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
Example 2: Shadow Volume Extrusion
Shadow Volume Extrusion Basics

- Leave polygons facing the light in place (front cap)
- Move back-facing polygons away from light (back cap)
- Use side quads to stitch front and back caps
Problems in DX9

- Doesn’t work correctly with animated objects
  - Can’t skin face normals
- Cannot generate side triangles dynamically
- Apps are forced to perform animation and shadow volume extrusion on CPU
Shadow Volume Extrusion with R2VB

• Can’t generate unknown number of polygons with R2VB
  – Solution: use degenerate quads for all edges

• Use separate pass to re-compute face normals after animation for the extrusion
Computing Face Normals

- Need access to all three vertices of a triangle
- Encode in index texture:
  - For each vertex $v$, stores 3 indices to transformed vertex positions that make up the triangle $v$ is part of
Index Texture Layout

Skinned vertex position texture

Index texture

R2VB face normal texture

Vertex 0

Vertex 1

Vertex 2
Shadow Volume Extrusion Breakdown

Matrix palette

Vertex data
Position
Normal
...

Animated model data

Face indices for each polygon

Main Rendering pass

Face normal pass

Face normal (stored for each vertex)

Animation pass

Shadow Volume Extrusion pass
sampler vertexPos;
sampler vertexIndexMap;
float iVertexTextureWidth;

float4 main( float2 t0: TEXCOORD0 ) : COLOR
{
    float4 index = tex2D( vertexIndexMap, t0 ) * 65535.0; // convert back to short

    float4 v0 = tex2D( vertexPos, float2(index.x * iVertexTextureWidth, 0.5) );
    float4 v1 = tex2D( vertexPos, float2(index.y * iVertexTextureWidth, 0.5) );
    float4 v2 = tex2D( vertexPos, float2(index.z * iVertexTextureWidth, 0.5) );
    float3 d0 = v0 - v1;
    float3 d1 = v2 - v1;
    float3 faceNormal = cross( d1, d0 );
    faceNormal = normalize( faceNormal );

    return float4( faceNormal, 0.0 );
}
Example 3: Dynamic Displacement Mapping
Displacement Mapping

- Height map in R2VB render target
- Render footprints into height map
- Collide particles against height map
  - Deposits more snow on collision
- Use height map to displace planar triangle mesh
Data Layout for Displacement

- One vertex per heightmap texel

Height Map

Ground Mesh
Data Layout for Displacement

- One vertex per heightmap texel
Foot Prints

• Render character from below using orthographic projection
  – Store distance to ground in separate renderable texture
• Blur renderable texture for nicer slopes
• Blend with height map
  – MIN blend op
Water Simulation

- Another application for displacement mapping
Cloth Simulation

- Mass-spring system simulated in pixel shaders
Particle Simulation and Sorting

- Bitonic merge sort for rendering particles back-to-front
N-Patch Tessellation

- Pre-tesselated geometry
- Higher-order surface evaluation in pixel shader
Conclusion

• New types of algorithms and effects
• Efficient
• Example takeaway:
  – Vertex processing in pixel shaders
  – Access to mesh adjacency in pixel shaders
• Road to D3D10 and next-gen GPU architectures
  – Prototype new ideas today
  – Get a better idea of potential performance than D3D10
Acknowledgements

• ATI ISV Engineering Team
  – Emil Persson
  – Owen Wu
  – Guennadi Riguer
Questions?

thorsten@ati.com

Slides downloadable at
www.ati.com/developer