Motivation

- Direct3D 10 is Microsoft’s next graphics API
  - Driving the feature set of next generation GPUs

- Many new features
  - New programmability, generality

- New driver model
  - Improved performance

- Cleaned up API
  - Improved state handling. Almost no caps bits!
Agenda

- Short review of DX10 pipeline and features
- Effect Case Studies
  - Curves
  - Silhouette detection
  - Metaballs
- Conclusions
Direct3D 10 Features Overview

- Common shader core
  - Integer operations in shaders
- Geometry shader
- Stream out
- Texture arrays
- Generalized resources
- Improved instancing support
Geometry Shader

- Brand new programmable stage
- Allows GPU to create (or destroy) geometry
- Run after vertex shader, before setup
- Input: point, line or triangle
  - Also new primitive types with adjacency information
- Output: points, line strips or triangle strips
  - Can output multiple primitives
- Allow us to offload work from CPU
Geometry Shader Applications

- Shadow volume generation
- Fur / fin generation
- Render to cubemap
- GPGPU
  - enables variable number of outputs from shader
Silhouette Detection

- Calculate geometric normal of centre triangle and adjacent triangles
- Calculate dot products between normals and view direction
- If centre triangle is facing towards viewer and adjacent triangle is facing away, edge must be on silhouette
Silhouhette Detection

- Can be used for cartoon shading
- Same basic technique used for stencil shadow volumes extrusion
Bezier Curve Tessellation

Input:
- Line with adjacency
- 4 control vertices
Bezier Curve HLSL Code

```hlsl
[maxvertexcount(10)]
void bezier_GS(lineadj float4 v[4],
    inout LineStream<float4> stream,
    uniform int segments = 10)
{
    float4x4 bezierBasis = {  
        { 1, -3, 3, -1 },
        { 0, 3, -6, 3 },
        { 0, 0, 3, -3 },
        { 0, 0, 0, 1 }
    };

    for(int i=0; i<segments; i++) {
        float t = i / (float) (segments-1);
        float4 tvec = float4(1, t, t*t, t*t*t);
        float4 b = mul(bezierBasis, tvec);
        float4 p = v[0]*b.x + v[1]*b.y + v[2]*b.z + v[3]*b.w;
        stream.Append(p : SV_POSITION)
    }
    CubeMapStream.RestartStrip();
}
```
Beziers Curve Tessellation

Output:
Line strip
Fur Generation

1\textsuperscript{st} pass
- Generate lines with adjacency from triangles
- Use barycentric coords
- Direction based on tangent vectors
- Use noise texture to perturb directions

2\textsuperscript{nd} pass
- Generate curves from lines
Fur Generation

Light using anisotropic lighting model
Stream Out

- Allows storing output from geometry shader to buffer
- Enables multi-pass operations on geometry, e.g.
  - Recursive subdivision
  - Store results of skinning to buffer, reuse for multiple lights
- Can use DrawAuto() function to automatically draw correct no. of primitives
  - No CPU intervention required
Geometry Shader Tips

- GS is not designed for large-scale tessellation
- Output limited to 1024 float values
- Try to minimize output size
- Output order is guaranteed
- Prefer multi-pass algorithms using stream-out to single pass with large output
- Do as much as possible in vertex shader
  - Run once per vertex, rather than per primitive vertex
- No quad input type
  - can use lines with adjacency instead (4 vertices)
Texture Arrays

- Array of 1D or 2D textures
- Indexable from shader
- Slices must be same size and format
- Arrays of cubemaps not supported (until DX10.1)
- Removes need for texture atlases
  - Useful for instancing, terrain texturing
Terrain using Texture Arrays
Rendering to Texture Arrays

- Can select destination slice from GS
  - Write to one of more layers

- Contrast to MRT
  - Writes to all render targets

- Can be used for single-pass render-to-cubemap
  - Read input triangle
  - Output to 6 cube map faces, transformed by correct face matrix
  - Simple culling may help
Two new floating point HDR formats

- **R9G9B9E5_SHAREDEXP**
  - 9 bit mantissa, shared 5 bit exponent
  - Very similar to Radiance RGBE format (R8G8B8E8)
  - Cannot be used for render targets (would be lossy)
  - Good for storing emissive textures (sky boxes etc.)

- **R11G11B10_FLOAT**
  - Each component has own 5 bit exponent (like fp16 numbers)
  - RGB components have 6, 6, 5 bits of mantissa each (vs. 10 bit mantissa for fp16)
  - No sign bit, all values must be positive
  - Can be used for render targets

No sign bits, all values must be positive
Case Study: GS Metaballs
What are Isosurfaces?

Consider a function $f(x, y, z)$
- Defines a *scalar field* in 3D-space
- Can come from procedural function, or 3D simulation

**Isosurface** $S$ is a set of points for which

$$f(x, y, z) = \text{const}$$

- Can be thought of as an *implicit* function relating $x$, $y$ and $z$
- Sometimes called *implicit* surfaces
Metaballs

- A particularly interesting case
- Use implicit equation of the form
  \[ \sum_{i=1}^{N} \frac{r_i^2}{\|x - p_i\|^2} = 1 \]
- Gradient can be computed directly
  \[ \nabla f = -\sum_{i=1}^{N} \frac{2 \cdot r_i^2}{\|x - p_i\|^4} \cdot (x - p_i) \]
- Soft/blobby objects that blend into each other
  - Perfect for modelling fluids, explosions in games
The Marching Cubes Algorithm

A well-known method for scalar field polygonization

Sample \( f(x, y, z) \) on a cubic lattice

For each cubic cell:

- Estimate where isosurface intersects cell edges by linear interpolation
- Tessellate depending on values of \( f() \) at cell vertices
The marching cubes algorithm

- Each vertex can be either “inside” or “outside”
- For each cube cell there are 256 ways for isosurface to intersect it
- Can be simplified down to 15 unique cases
Geometry shaders in DX10

From CPU

Vertex Shader

Geometry Shader

Raster

Pixel Shader

Stream Out

To Framebuffer

Triangles with adjacency:
- Vertices 0, 1, 2
- Edges (0, 1), (1, 2), (2, 0)

Lines with adjacency:
- Vertices 0, 1, 2, 3
- Edges (0, 1), (1, 2), (2, 3), (3, 0)
Implementation - Basic Idea

- App feeds a GPU with a grid of vertices
- VS transforms grid vertices and computes $f(x, y, z)$, feeds to GS
- GS processes each cell in turn and emits triangles
A problem...

- Topology of GS input is restricted
  - Points
  - Lines
  - Triangles
  - with optional adjacency info
- Our “primitive” is a cubic cell
  - Need to input 8 vertices to a GS
  - A maximum we can input is 6 (with triangleadj)
Solution

- First, note that actual input topology is irrelevant for GS
  - E.g. lineadj can be treated as quad input

- Work at tetrahedra level
  - Tetrahedron is 4 vertices - perfect fit for lineadj!

- We’ll subdivide each cell into tetrahedra
Marching Tetrahedra (MT)

- Tetrahedra are easier to handle in GS
  - No ambiguities in isosurface reconstruction
  - Always output either 1 or 2 triangles
Generating a sampling grid

- There’s a variety of ways to subdivide
  - Along main diagonal into 6 tetrahedra – MT6
  - Tessellate into 5 tetrahedra – MT5
  - Body-centered tessellation – CCL

- Can also generate tetrahedral grid directly
  - AKA simplex grid
  - Doesn’t fit well within rectilinear volume
Sampling grids

MT5

MT6

CCL
## Sampling grids comparison

<table>
<thead>
<tr>
<th></th>
<th>Generation Complexity</th>
<th>Sampling effectiveness</th>
<th>Regularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT5</td>
<td>Med</td>
<td>Med</td>
<td>Low</td>
</tr>
<tr>
<td>MT6</td>
<td>Low</td>
<td>Med</td>
<td>Low</td>
</tr>
<tr>
<td>CCL</td>
<td>High</td>
<td>High</td>
<td>Med</td>
</tr>
<tr>
<td>Simplex</td>
<td>Low</td>
<td>Med</td>
<td>High</td>
</tr>
</tbody>
</table>
// Grid vertex
struct SampleData
{
    float4 Pos : SV_POSITION; // Sample position
    float3 N : NORMAL; // Scalar field gradient
    float Field : TEXCOORD0; // Scalar field value
    uint IsInside : TEXCOORD1; // “Inside” flag
};

// Surface vertex
struct SurfaceVertex
{
    float4 Pos : SV_POSITION; // Surface vertex position
    float3 N : NORMAL; // Surface normal
};

VS/GS Input/output
// Metaball function
// Returns metaball function value in .w
// and its gradient in .xyz

float4 Metaball(float3 Pos, float3 Center, float RadiusSq)
{
    float4 o;

    float3 Dist = Pos - Center;
    float InvDistSq = 1 / dot(Dist, Dist);

    o.xyz = -2 * RadiusSq * InvDistSq * InvDistSq * Dist;
    o.w = RadiusSq * InvDistSq;

    return o;
}
```glsl
#define MAX_METABALLS 32

SampleData VS_SampleField(float3 Pos : POSITION,
uniform float4x4 WorldViewProj,
uniform float3x3 WorldViewProjIT,
uniform uint NumMetaballs, uniform float4 Metaballs[MAX_METABALLS])
{
    SampleData o;
    float4 Field = 0;

    for (uint i = 0; i<NumMetaballs; i++)
        Field += Metaball(Pos, Metaballs[i].xyz, Metaballs[i].w);

    o.Pos = mul(float4(Pos, 1), WorldViewProj);
    o.N = mul(Field.xyz, WorldViewProjIT);
    o.Field = Field.w;
    o.IsInside = Field.w > 1 ? 1 : 0;

    return o;
}
```
Geometry Shader

// Estimate where isosurface intersects grid edge
SurfaceVertex CalcIntersection(SampleData v0, SampleData v1)
{
    SurfaceVertex o;

    float t = (1.0 - v0.Field) / (v1.Field - v0.Field);

    o.Pos = lerp(v0.Pos, v1.Pos, t);
    o.N = lerp(v0.N, v1.N, t);

    return o;
}
Geometry Shader

[MaxVertexCount(4)]

void GS_TessellateTetrahedra(lineadj SampleData In[4],
    inout TriangleStream<SurfaceVertex> Stream)
{
    // construct index for this tetrahedron
    uint index =
        (In[0].IsInside << 3) | (In[1].IsInside << 2) |
        (In[2].IsInside << 1) | In[3].IsInside;

    const struct { uint4 e0; uint4 e1; } EdgeTable[] = {
        { 0, 0, 0, 0, 0, 0, 0, 1 },  // all vertices out
        { 3, 0, 3, 1, 3, 2, 0, 0 },  // 0001
        { 2, 1, 2, 0, 2, 3, 0, 0 },  // 0010
        { 2, 0, 3, 0, 2, 1, 3, 1 },  // 0011 - 2 triangles
        { 1, 2, 1, 3, 1, 0, 0, 0 },  // 0100
        { 1, 0, 1, 2, 3, 0, 3, 2 },  // 0101 - 2 triangles
        { 1, 0, 2, 0, 1, 3, 2, 3 },  // 0110 - 2 triangles
        { 3, 0, 1, 0, 2, 0, 0, 0 },  // 0111
        { 0, 2, 0, 1, 0, 3, 0, 0 },  // 1000
        { 0, 1, 3, 1, 0, 2, 3, 2 },  // 1001 - 2 triangles
        { 0, 1, 0, 3, 2, 1, 2, 3 },  // 1010 - 2 triangles
        { 3, 1, 2, 1, 0, 1, 0, 0 },  // 1011
        { 0, 2, 1, 2, 0, 3, 1, 3 },  // 1100 - 2 triangles
        { 1, 2, 3, 2, 0, 2, 0, 0 },  // 1101
        { 0, 3, 2, 3, 1, 3, 0, 0 }    // 1110
    };
}
const struct { uint4 e0; uint4 e1; } EdgeTable[] = {
    // ...
    { 3, 0, 3, 1, 3, 2, 0, 0 }, // index = 1
    // ...
};

Index = 0001, i.e. vertex 3 is “inside”
Geometry Shader

// ... continued
// don't bother if all vertices out or all vertices in
if (index > 0 && index < 15)
{
    uint4 e0 = EdgeTable[index].e0;
    uint4 e1 = EdgeTable[index].e1;

    // Emit a triangle
    Stream.Append(CalcIntersection(In[e0.x], In[e0.y]));
    Stream.Append(CalcIntersection(In[e0.z], In[e0.w]));
    Stream.Append(CalcIntersection(In[e1.x], In[e1.y]));

    // Emit additional triangle, if necessary
    if (e1.z != 0)
        Stream.Append(CalcIntersection(In[e1.z], In[e1.w]));
}
}
Respect your vertex cache!

- $f(x, y, z)$ can be arbitrary complex
  - E.g., many metaballs influencing a vertex

- Need to be careful about walk order
  - Worst case is 4x more work than necessary!
  - Straightforward linear work is not particularly cache friendly either

- Alternatively, can pre-transform with StreamOut
Tessellation space

Object space
- Works if you can calculate BB around your metaballs

View space
- Better, but sampling rate is distributed inadequately
Tessellation in post-projection space

- View-space
- Post-projective space

- Probably the best option
- We also get LOD for free!
Problems with current approach

- Generated mesh is over-tessellated
  - General problem with MT algorithms
- Many triangles end up irregular and skinny
  - Good sampling grid helps a bit

(a) MT, smooth  (b) MT, triangles
Possible enhancements

- Regularized Marching Tetrahedra (RMT)
  - Vertex clustering prior to polygonization
  - Generated triangles are more regular
  - For details refer to [2]

- Need to run a pre-pass at vertex level, looking at immediate neighbors
  - For CCL, each vertex has 14 neighbors
  - GS input is too limited for this 😞
Conclusion

- Direct3D 10 is a major discontinuity in graphics hardware functionality
- Enables new effects and better performance
- Start redesigning your game engine now
Questions?

sgreen@nvidia.com
Buffer Resources

- Input assembler accepts
  - Vertex buffer
  - Index buffer
  - Buffer resource

- Buffer resource can only be rendered to
  - And limited to 8k elements at a time

- Multiple passes can get you a R2VB
Respect your vertex cache!

- Can use space-filling fractal curves
  - Hilbert curve
  - Swizzled walk
- We’ll use swizzled walk
- To compute swizzled offset, just interleave x, y and z bits
Linear walk vs swizzled walk

Linear walk

Swizzled walk