**Motivation**

- Save memory and bandwidth
  - Memory is the main bottleneck to render highly detailed surfaces

**Scalability**

- Continuous Level of Detail

**Compression**

- Continuous Level of Detail

**Scalability**

- View Dependent Level of Detail

**Agenda**

- Motivation
- Tessellation Basics
- DX11 Tessellation Pipeline
- Instanced Tessellation in DX10
- Displacement Mapping
- Content Creation
Animation & Simulation

- Perform Expensive Computations at lower frequency:
  - Realistic animation: blend shapes, morph targets, etc.
  - Physics, collision detection, soft body dynamics, etc.

Goal

- Enable unprecedented visuals:
  - Highly detailed characters
  - Realistic animation

Tessellation Basics

- A new primitive called patch
  - Patch is defined by a set of control points

Tessellation Concept

- A new type of operation called refinement
  - Generate a number of triangles from a patch

Transforming control points

- Animation can be performed at a lower rate

Per-patch Operations

- LOD computation
- Transform control points to different basis
- Computes edge tessellation levels
Generating Topology

- Generate a set of (u,v)-points in the tessellation domain

Evaluating Points

Example: Catmull-Clark Subdivision

- Most popular scheme
  - Based on B-spline subdivision
  - Evaluation through recursive refinement
    - Add one new vertex for every face and every edge

Example: Catmull-Clark Subdivision
Catmull-Clark Subdivision

- Resulting mesh only approximates the limit subdivision surface
- Halstead et al show that it’s possible to project the vertices to the limit surface
  - Efficient, Fair Interpolation using Catmull-Clark Surfaces

GPU Implementations

- Previous approaches on the GPU:
  - "Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping", Michael Bunnell
  - Recursive Geometry Shader refinement

  • Require multiple passes → Direct evaluation is preferred

DX11 Tessellation Pipeline

Tessellation Pipeline

- Direct3D11 extends Direct3D10 with support for programmable tessellation
  - Two new shader stages:
    - Hull Shader (HS)
    - Domain Shader (DS)
  - One fixed function stage:
    - Tessellator (TS)

Input Assembler

- New Patch primitive type
  - Arbitrary vertex count (up to 32)
  - No implied topology
  - Only supported primitive when tessellation is enabled

Vertex Shader

- Transforms patch control points
- Usually used for:
  - Animation (skinning, blend shapes)
  - Physics simulation
- Allows more expensive animation at a lower frequency
**Hull Shader (HS)**
- One invocation per patch
- Parallelized explicitly. Code split to multiple threads by compiler
  - One thread per output control point
  - One thread per patch constant

**Domain Shader (DS)**
- One invocation per generated vertex
- **Input:**
  - U V W coordinates
- **Output:**
  - One vertex

**Surface Evaluation**
- **Input:**
  - [1..32] control points
  - Tessellation factors
- **Output:**
  - Bicubic Bezier patch
- Special care has to be taken to obtain watertight results (prevent cracks)
- All computations need to be symmetric along the patch edges

**Instanced Tessellation**
- We can approximate the tessellation pipeline with DirectX 10 API
- Instancing can be used to replicate patches

**Instanced Tessellation in DX10**

**GS is not for tessellation**
- GS outputs triangles serially
GS is not for tessellation

- Limited output size (maximum 1024 scalars) is not always enough
- If each vertex is 4 float's, you can only tessellate up to 16x16

Use instancing instead

- Render pre-tessellated patch with instancing
- Set the entire mesh as instance data

Use instancing instead

- Render pre-tessellated patch with instance count equal to patch count in the mesh
- Pre-tessellated patch represents results of tessellating every input patch

Use instancing instead

- Compute refined vertex position in the vertex shader using chosen evaluation algorithm

Input attributes limitation

- Maximum VS input size is not enough to fit all data required for point evaluation
- Instead all data can be stored in buffers bound as shader resources
- Use Load() instruction to fetch this data
Using Load()
- Store mesh data in vertex buffers
- Bind these buffers as shader resources

Load() and InstanceID
- Use SV_InstanceID as an index to the patch buffer

Computing U and V coords
- Use SV_VertexID to compute U and V coordinates for the current vertex
  \[ U = \text{VertexID} \mod \text{LoD} \]
  \[ V = \text{VertexID} \div \text{LoD} \]

Application integration
- Without tessellation
- With tessellation

Results of refinement
- Before
- After
Tips

- Pack all data in float4-buffers to use Load() more efficiently
- Use $2^n \times 2^n$ tessellation and bitwise operations to compute U and V from Vertex_ID
  - Integer division is slow!

Displacement Maps

- Crack-free displacement maps
  - Consistent normal evaluation
  - Watertight texture sampling

Consistent Normal Evaluation

- Control tangents along edges are not symmetric:

Watertight Texture Sampling

- Texture seams cause holes in the mesh!
  - Due to bilinear discontinuities
  - Varying floating point precision on different regions of the texture map

Watertight Texture Sampling

- Seamless parameterizations remove bilinear artifacts, but do not solve floating point precision issues
Watertight Texture Sampling

- Texture coordinate interpolation yields different result depending on location of the seam edges:

Content Creation

Production Pipeline

- Modeling Tools
  - Base surface
- Sculpting Tools
  - Detailed mesh
- Baker Tools
  - Normal, displacement, occlusion, and other maps

Modeling

- Performance depends on number of topology combinations
- Optimization guidelines:
  - Eliminate triangles (Quad only meshes)
  - Close holes (Avoid open meshes)
  - Reduce number of extraordinary vertices
  - Decrease number of patches to the minimum
  - Try to create uniform, regular meshes

Topology Optimization

- 105 topology combinations

Topology Optimization

- 23 topology combinations

© 2008 NVIDIA Corporation. All rights reserved.
Topology Optimization

- Topology visualization tool (nvAnalyze)
  - Maya plugin that highlights faces that damage mesh quality the most

NVIDIA Mesh Processing Tool

- Successor of NVMeshMender and NVTriStrip but for subdivision surfaces:
  - Reorder faces for consistent adjacencies
  - Minimize topology combinations
  - Pre-compute stencils for different approximation schemes
  - Compute texture coordinates for watertight texture sampling
  - Optimize vertex and face order for best performance
  - And more!

Sculpting

- Many tools available:
  - Autodesk® Mudbox™
  - Pixologic ZBrush®
  - modo™, Silo, Blender, etc.

Baker Tools

- Many options:
  - xNormal™
  - Mudbox™, ZMapper
  - Melody™, etc.
  - PolyBump™, etc.

  - Two approaches
    - Ray casting
    - Dual parameterization

Vector Displacements

- Native representation of most sculpting tools

NVIDIA Baker Tool

- Uses dual parameterization to extract:
  - Normal and displacement maps
  - Only tool that generates vector displacements
  - Occlusion maps, and more!

- No other tool supports custom base surfaces:
  - Bezier ACC
  - Gregory ACC
  - Triangle meshes
NVIDIA Baker Tool

- Uses optimized Montecarlo Raytracer
- Can be easily extended to support:
  - Bent normals
  - Spherical harmonic PRTs
  - etc.
- Full source code will be openly available

Thanks

- Bay Raitt, Mike Asquith, Valve Corporation
- Kenneth Scott, id Software

Questions?
calin@nvidia.com