Tessellation & Geometry
Shaders
Trends
Why tessellation?
Lack of geometric detail...

- Pixels are meticulously shaded, but geometric detail is modest

Image from Far Cry® 2, courtesy of Ubisoft

Henry Moreton
NVIDIA Corporation
October 13, 2011
Geometry in Film

- Pixels are meticulously shaded and geometric detail is substantial
- Tessellation + displacement mapping is the defacto standard
- Enables richer content and animation

© Disney Enterprises, Inc. and Jerry Bruckheimer, Inc. All rights reserved. Image courtesy Industrial Light & Magic.
GF100 enables much greater geometric detail

- Before GF100 - minimal progress in geometry performance
  - GeForce FX 5800 to GeForce GT200
    - >150x shading performance
    - <3x geometry performance
  - APIs unable to support a significant increase in geometry
    - Chicken & egg - really...

- GF100 — New geometry processing architecture delivers 8x performance to support DX11
Tessellation - What and Why

- **Memory footprint & BW savings**
  - Store coarse geometry, expand on-demand, keep data on die
  - Enables more complex animations

- **Scalability**
  - Dynamic LOD allows for performance/quality tradeoffs
  - Scale into the future - resolution, compute power

- **Computational efficiency**
  - Dynamic LOD
  - GPU animate and expand compact representation

- **Real geometry**
  - Dynamic shadows
  - 3D Vision™

© Kenneth Scott, id Software 2008
Tessellation - What and Why

- **Memory footprint & BW savings**
  - Store coarse geometry, expand on-demand, keep data on die
  - Enables more complex animations

- **Scalability**
  - Dynamic LOD allows for performance/quality tradeoffs
  - Scale into the future - resolution, compute power

- **Computational efficiency**
  - Dynamic LOD
  - GPU animate and expand compact representation

- **Real geometry**
  - Dynamic shadows
  - 3D Vision™

© Kenneth Scott, id Software 2008
Tessellation - What and Why

- Memory footprint & BW savings
  - Store coarse geometry, expand on-demand, keep data on die
  - Enables more complex animations

- Scalability
  - Dynamic LOD allows for performance/quality tradeoffs
  - Scale into the future - resolution, compute power

- Computational efficiency
  - Dynamic LOD
  - GPU animate and expand compact representation

- Real geometry
  - Dynamic shadows
  - 3D Vision™
Hull shader
- Runs pre-expansion
- Explicitly parallel
- Output control points and LODs

Tessellation in DirectX 11
Tessellation in DirectX 11

- Fixed function tessellation stage
- Configured by HS LOD output
- Produces U, V values
- Primitive topology
- Supports triangles and lines

From input assembly

Diagram:

- Vertex
- Patch Assembly
- Hull
- Tessellator
- Domain
- Primitive Assembly
- Geometry
Tessellation in DirectX 11

- Domain shader
  - Runs post-expansion
  - Input: LOD, (u,v), control points
  - Maps (u,v) to (x,y,z,w)
  - Attributes
  - Implicitly parallel

From input assembly

- Vertex
- Patch Assembly
- Hull
- Tessellator
- Domain
- Primitive Assembly
- Geometry

Control points
DX10 Logical Pipeline

- Primitive Distributor
- Vertex Geometry
- Vertex Geometry
- Vertex Geometry
- Vertex Geometry
- Viewport Transform
- Setup
- Raster
- Raster
- Raster
- Raster
- Raster
- Pixel
- Pixel
- Pixel
- Crossbar
- ROP
- ROP
- ROP
- ROP
- ROP
- FB
- FB
- FB
- FB
- DATA FLOW

Henry Moreton
NVIDIA Corporation
October 13, 2011
DX10 Logical Pipeline + Tessellation
DX10 + Tessellation - data expansion

API ordering means buffering before Setup
Fermi GF100 Logical Pipeline

Vertex
Hull

Vertex
Hull

Vertex
Hull

Vertex
Hull

Tessellator
Domain
Geometry
Viewport

Tessellator
Domain
Geometry
Viewport

Tessellator
Domain
Geometry
Viewport

Tessellator
Domain
Geometry
Viewport

Raster
Attribute Setup
PS

Raster
Attribute Setup
PS

Raster
Attribute Setup
PS

Raster
Attribute Setup
PS

Crossbar
ROP
FB

ROP
FB

ROP
FB

ROP
FB

DATA FLOW
Fermi GF100 Logical Pipeline

- Task Distributor
  - Task $\approx$ Hull Shader output
  - Control points + LOD
  - Pre-expansion

- Distribute tasks
  - Expand patch into primitives
  - Optional GS

- Reduced buffering
Fermi GF100 Logical Pipeline

- Polymorph Engine
- Tessellator
- Viewport Transform
- Attribute Setup
Fermi GF100 Logical Pipeline

- Parallel Rasterization
- Edge Setup, Raster & Z Cull
- Multiple primitives per clock
- Screen mapped load balancing
GF100 Block Diagram

- 512 CUDA cores
- 16 geometry units
- 4 raster units
- 64 texture units
- 48 ROP units
- 384-bit GDDR5
GF100 Scalable Parallel Implementation

The Challenge:
Sequential Rendering Semantics
Maintaining API Order

- Greater parallelism is straightforward
- API order is the challenge
- WDX - Work Distribution Crossbar
  - Between Viewport and Raster
  - Distributes work
  - Maintains order
Work Distribution Crossbar

**OWDX**
- Bounding box
- Broadcast to Raster

**SWDX**
- Reconstructs API order

Each Raster owns its pixels
- No further sorting
Screen-mapped Rasterization

- Each block is a tile of pixels
- Blocks are bound to rasterizers
- Small primitives can still straddle tiles
Load balance tension

- **Small tiles**
  - + Better pixel distribution
  - - More redundant Setup

- **Large tiles**
  - - Risk camping
  - + Not Setup limited
Questions?
Geometry Shaders - a postmortem

- Introduced as part of DX10
- Intended as a tessellation post-processor
  - Vestige of stencil shadow volumes
- Implements legacy features – sprites
- API sequential rendering semantics are costly
- Outputs are spilled to memory or buffered
Future

- More transistors
- No more watts
- More dark silicon
  - Special purpose units
    - Video encode/decode
    - Camera
    - Copy
  - Suspended cores
- Vision
  - The killer consumer application?
  - OpenCV - low level
  - Is there an API at a higher semantic level?
    - Analogous to touch....
Thanks