Tessellation in a Low Poly World

Bill Bilodeau, AMD         Peter Lohrmann, AMD
Overview

• Part 1 - Rendering with Tessellation (Bill)
  – Background
  – Tessellation on the GPU
  – GPU Tessellation Demo

• Part 2 - Content Creation (Peter)
  – GPU MeshMapper Demo
  – How GPU MeshMapper Works
  – Mesh Authoring Tips for Tessellation

• Questions
Part 1
Rendering With Hardware Tessellation
Bill Bilodeau, AMD
AMD has added a hardware tessellator to its new ATI graphics cards.
High order surfaces can be rendered using tessellation.

Complex surfaces need to be broken up into many triangles for rendering on 3D hardware.

Triangle counts can be “dialed in” by adjusting the tessellation level.
Hardware tessellation allows you to render more polygons for better silhouettes.

Initial concept artwork from Bay Raitt, Valve

No Tessellation

With Tessellation and Displacement

Initial concept artwork from Bay Raitt, Valve
Surface control cages are easier to work with than individual triangles.

Artists prefer to create models this way.

Animations are simpler on a control cage.

Control cage can be animated on the GPU, then tessellated in a second pass.
Hardware tessellation is a form of compression.

Smaller footprint – you only need to store the control cage and possibly a displacement map.

Improved bandwidth – less data to transfer across the bus.
Three types of primitives, or “superprims”, are supported.

- Triangles
- Quads
- Lines
There are two tessellation modes

- Continuous

- Adaptive
Continuous Tessellation

Specify floating point tessellation level per-draw call
- Tessellation levels range from 1.0 to 14.99

Eliminates popping as vertices are added through tessellation
Continuous Tessellation

Specify floating point tessellation level between per-draw call
- Tessellation levels range from 1.0 to 14.99

Eliminates popping as vertices are added through tessellation
Adaptive allows different levels of tessellation within the same mesh.
Adaptive tessellation can be done in real-time using multiple passes.
The ATI Tessellation Library provides a clean interface that hides implementation details.

ATI Tessellation Library
(DirectX or OpenGL)

Graphics API: DX9, DX10, OpenGL    OS: XP, Vista, Mac

Hardware with Tessellator

Hardware without Tessellator
Code Example: Continuous Tessellation

// Enable tessellation:
TSSetTessellationMode( pd3dDevice, TSMD_ENABLE_CONTINUOUS );
// Set tessellation level:
TSSetMaxTessellationLevel( pd3dDevice, sg_fMaxTessellationLevel );
// Select appropriate technique to render our tessellated objects:
sg_pEffect->SetTechnique( "RenderTessellatedDisplacedScene" );

// Render all passes with tessellation
V( sg_pEffect->Begin( &cPasses, 0 ) );
   for ( iPass = 0; iPass < cPasses; iPass++ )
   {
      V( sg_pEffect->BeginPass( iPass ) );
      V( TSDrawMeshSubset( sg_pMesh, 0 ) );
      V( sg_pEffect->EndPass() );
   }
V( sg_pEffect->End() );

// Disable tessellation:
TSSetTessellationMode( pd3dDevice, TSMD_DISABLE );
The vertex shader is used as an **evaluation shader**.

Super-prim Mesh → **Tessellator** → Tessellated Mesh → (Evaluation Shader) → **Vertex Shader** → **Sampler** → **Tessellated and Displaced Mesh** → **Displacement Map**
Example Code: Evaluation Vertex Shader

```cpp
struct VsInputTessellated
{
    // Barycentric weights for this vertex
    float3 vBarycentric: BLENDWEIGHT0;

    // Data from superprim vertex 0:
    float4 vPositionVert0 : POSITION0;
    float2 vTexCoordVert0 : TEXCOORD0;
    float3 vNormalVert0 : NORMAL0;

    // Data from superprim vertex 1:
    float4 vPositionVert1 : POSITION4;
    float2 vTexCoordVert1 : TEXCOORD4;
    float3 vNormalVert1 : NORMAL4;

    // Data from superprim vertex 2:
    float4 vPositionVert2 : POSITION8;
    float2 vTexCoordVert2 : TEXCOORD8;
    float3 vNormalVert2 : NORMAL8;
};
```
Example Code: Evaluation Vertex Shader

VsOutputTessellated VSRenderTessellatedDisplaced( VsInputTessellated i )
{
    VsOutputTessellated o;
    // Compute new position based on the barycentric coordinates:
    float3 vPosTessOS = i.vPositionVert0.xyz * i.vBarycentric.x + i.vPositionVert1.xyz * i.vBarycentric.y + i.vPositionVert2.xyz * i.vBarycentric.z;
    // Output world-space position:
    o.vPositionWS = vPosTessOS;
    // Compute new normal vector for the tessellated vertex:
    o.vNormalWS = i.vNormalVert0.xyz * i.vBarycentric.x + i.vNormalVert1.xyz * i.vBarycentric.y + i.vNormalVert2.xyz * i.vBarycentric.z;
    // Compute new texture coordinates based on the barycentric coordinates:
    o.vTexCoord = i.vTexCoordVert0.xy * i.vBarycentric.x + i.vTexCoordVert1.xy * i.vBarycentric.y + i.vTexCoordVert2.xy * i.vBarycentric.z;
    // Displace the tessellated vertex (sample the displacement map)
    o.vPositionWS = DisplaceVertex( vPosTessOS, o.vTexCoord, o.vNormalWS );
    // Transform position to screen-space:
    o.vPosCS = mul( float4( o.vPositionWS, 1.0 ), g_mWorldViewProjection );
    return o;
}  // End of VsOutputTessellated VSRenderTessellatedDisplaced(....)
What if you want to do more?

DirectX 9 has a limit of 15 `float4` vertex input components – High order surfaces need more inputs.

`TSToggleIndicesRetrieval()` allows you to fetch the super-prim data from a vertex texture.

\[
P(u, v) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_{0,0} \\ P_{1,0} \\ P_{2,0} \\ P_{3,0} \end{bmatrix} \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ v \\ v^2 \\ v^3 \end{bmatrix}
\]
Other Tessellation Library Functions

TSDrawIndexed(…)
- Analogous to DrawIndexedPrimitive(…)

TSDrawNonIndexed(…)
- Needed for adaptive tessellation, since every edge needs its own tessellation level.

TSSetMinTessellationLevel(…)
- Sets the minimum tessellation level for adaptive tessellation.

TSComputeNumTessellatedPrimitives(…)
- Calculates the number of tessellated primitives that will be generated by the tessellator.
Displacement mapping alters tangent space.

To do normal mapping we need to rotate tangent space.

Alternatively, use world space normal maps.

- Doesn’t work with animation or tiling.
Displacement map lighting

Use the displacement map to calculate the per-pixel normal.

- Central differencing with neighboring displacements can approximate the derivative

Light with the computed normal

No need to use a normal map
Catmull-Clark subdivision surfaces can be approximated by Bezier patches.

Widely adopted by existing tools’ pipelines

Loop and Schaefer algorithm uses very few Bezier patches to closely approximate Catmull-Clark surface

For each quad face of the control mesh, construct a geometry patch and a pair of tangent patches which are used to tessellate the surface

Figure 1: a) The patch structure of a Catmull-Clark subdivision surface. The grey patches only contain vertices of valence 4, green have one extraordinary vertex and blue have more than one extraordinary vertex. b) Our approximation to the Catmull-Clark subdivision surface using geometry patches and c) our final approximation using geometry and tangent patches compared with d) the actual Catmull-Clark limit surface.

[Loop Schaefer 2007]
## Terrain Rendering: Performance Results

<table>
<thead>
<tr>
<th></th>
<th>Low Resolution with Tessellation</th>
<th>High Resolution, No Tessellation</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-disk model polygon count (pre-tessellation)</td>
<td>840 triangles</td>
<td>1,280,038 triangles</td>
</tr>
<tr>
<td>Original model rendering cost</td>
<td></td>
<td>1210 fps (0.83 ms)</td>
</tr>
<tr>
<td>Actual rendered model polygon count</td>
<td>1,008,038 triangles</td>
<td>1,280,038 triangles</td>
</tr>
<tr>
<td>VRAM Vertex buffer size</td>
<td>70 KB</td>
<td>31 MB</td>
</tr>
<tr>
<td>VRAM Index buffer size</td>
<td>23 KB</td>
<td>14 MB</td>
</tr>
<tr>
<td>Rendering time</td>
<td>821.41 fps (1.22 ms)</td>
<td>301 fps (3.32 ms)</td>
</tr>
</tbody>
</table>

Both use the same displacement map (2K x 2K) and identical pixel shaders. Rendering with tessellation is > 6X faster and provides memory savings over 44MB! Subtracting the cost of shading.
Terrain Tessellation Sample

[Image: Diagram of a terrain tessellation sample, showing a green wireframe model on the left and a realistic image of a snowy mountain range on the right.]

[Logo: AMD Smarter Choice]
Advantages of the Tessellator

• Saves memory bandwidth and reduces memory footprint

• Flexible support for displacement mapping and many kinds of high order surfaces.

• Easier content creation – artists and animators only need to work with low resolution geometry

• Continuous LOD avoids unnecessary triangles

• The tessellator is available now on the Xbox 360 and the latest ATI Radeon and FireGL graphics cards.
  – PC Driver available with March Catalyst release.
Acknowledgements

Natasha Tatarchuk, AMD
- Tessellator Library, Documentation, and Samples
- Slides from previous talks

Richard Huddy, Nick Thibieroz, AMD ISV Relations
Part 2
Creating Content for Hardware Tessellation
Peter Lohrmann, AMD
Content Creation

Overview

• GPU MeshMapper
  – Demo
  – Map Generation Tips

• Introduction to Mesh Mapping
  – How GPU MeshMapper Works

• Mesh Authoring Tips
  – Quality Displacement Maps
  – Take advantage of Tessellation
Prefix

• I am by no means an artist

• Soldier Model
  – Crowds Demo (2004)
  – Not authored for displacement
AMD GPU MeshMapper

New tool for generate normal, displacement, and ambient occlusion maps from hi-res and low-res mesh pairs
GPU MeshMapper: Key Features

• Integrated displacement and tessellation preview
  – See the results before integrating into your engine
  – Available on HD 2000 and HD 3000 series hardware

• Includes MapComposer tool
  – Combine maps for sub-components of model
  – Dynamically adjusts scale and bias

• Output scale and bias values
  – Removes guesswork from applying vertex displacement
  – Takes advantage of full dynamic range
Displacement Scale and Bias

GPU MeshMapper provides scale and bias for displacement maps

- Available in MapViewer, MapComposer, and in txt files associated with saved displacement maps

- Other tools don’t provide this, leaving the developers to guess values, and this can lead to surfaces looking bloated

- To calculate displacement in your shaders:

  \[
  \text{FinalDisplacement} = (\text{SampledDisplacement} \times \text{Scale}) + \text{Bias}
  \]
Map Generation: Review

- Use same low-resolution mesh as you will use for rendering
- Separate model into pieces
- Use highest resolution maps possible
- Use proper scale and bias values for displacement
- Not much gained by using 16-bits over 8-bits for displacement maps
Introduction to Mesh Mapping

Finding correlation between surfaces on low-resolution mesh and those on a high-resolution mesh

• For each point on a low-res surface
  – Travel along normal until intersect with high-res surface
  – Collect desired data about intersection
    • Normal
    • Displacement
    • AO
Mesh Authoring: Smooth Normals

Normals indicate which direction to displace along

- Surfaces displace in different directions
- No longer a smooth surface

Per-Face Normals

Rendered w/ Displacement
Mesh Authoring: Smooth Normals

Displacement after smoothed normals
Mesh Authoring: UV Seam Placement

In areas of high displacement
- Don’t place seams along steep slope
- Preferably put vertices along both high and low points
  - Allows textures to capture the sloping surface
    - Avoiding the artifacts from texture stretching
  - Avoids cracks along the slope

High-Res Mesh

Tessellated, viewed from side

Tessellated, viewed from front
(Cracks are visible from some viewpoints)
Mesh Authoring: Surface Proximity

Multiple high-res surfaces are in close proximity to one another

- Try to keep low-res surfaces close to their high-res counterpart
- Decrease your “Max Distance”
- Change “Map Correlation Settings”
Mesh Authoring: Avoid Pockets

“Pockets” are areas where the low-res model includes a volume where there is no high-res geometry.
Mesh Authoring: Fixing Pockets

Remove the pockets by making the coarse geometry fit within the high-resolution mesh.

- Fingers may still be webbed, but look much nicer

Dedicate a few triangles for each extremity

- The thumb is its own finger!
Mesh Authoring: “Enough” Triangles

Tessellation can turn each triangle into \(\sim 411\) triangles

- Place more geometry where you want more fine details to appear post-tessellation
- Too few triangles (or too small UV space) results in texture stretching
Mesh Authoring: Review

- Smoothed Normals
- Consider UV Seam Placement
- Consider Surface Proximity
- Avoid Pockets
- Use “Enough” Triangles
Summary

• Part 1 - Rendering with Tessellation (Bill)
  – Background
  – Tessellation on the GPU
  – GPU Tessellation Demo

• Part 2 - Content Creation (Peter)
  – GPU MeshMapper Demo
  – How GPU MeshMapper Works
  – Mesh Authoring Tips for Tessellation

http://ati.amd.com/developer/tools.html
QUESTIONS?

Ask now, or email:
bill.bilodeau@amd.com
peter.lohrmann@amd.com

Visit our developer web site at:
http://ati.amd.com/developer/