Tessellation Performance

Jon Story, AMD
Cem Cebenoyan, NVIDIA
Legend

- **AMD specific**
- **NVIDIA specific**
Why Tessellate?

- Enables substantial geometric fidelity
  - GPU side expansion very efficient
- Scale performance and quality
  - Programmatic LOD control
- Compute at lower, adaptively chosen, frequency
Geometric Realism With Tessellation
Generating Geometry On-Chip

- Coarse data read through IA
  Compact representation

- Hull shader controls expansion

- Domain shader evaluates surface
Geometry Data Flow

- Read coarse model data in VS
  - Take advantage of this!
    - Optimize models for post-transform cache
    - Do transformations and animation
    - Prepare all other per vertex attributes

VS Output

DS Output
Is Tessellation Free?

- No!
- If adding more geometry was free, we would have been doing this along time ago...😊
- Tessellation should be used where it will benefit image quality the most
- So tessellate wisely...
How Many Triangles?

<table>
<thead>
<tr>
<th>Tess Factor</th>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>37</td>
</tr>
<tr>
<td>7</td>
<td>73</td>
</tr>
<tr>
<td>9</td>
<td>121</td>
</tr>
<tr>
<td>11</td>
<td>181</td>
</tr>
<tr>
<td>13</td>
<td>253</td>
</tr>
<tr>
<td>15</td>
<td>337</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>64</td>
<td>~6000</td>
</tr>
</tbody>
</table>
Tessellation Factor ~ 1

- Mesh would look identical if rendered non-tessellated
- Using 3 additional pipeline stages needlessly
  - Total waste of GPU resources
- Use mesh bounding volume to calculate the average tessellation factor on the CPU
  - If ~1 render non-tessellated
Use Occlusion & Culling

- Don’t render occluded meshes!
  - Even more important for tessellated meshes
  - Consider using occlusion queries or predicated rendering

- Use the HS to cull patches outside the view frustum
  - Set tessellation factor to 0
  - ~30% speed up in one application
Use Adaptive Tessellation Techniques Aggressively

- Consider using a combination of these techniques in your HS:
  - Distance Adaptive
  - Orientation Adaptive
  - Density Adaptive
  - Screen Space Adaptive

- Select the combination that yields the biggest win in your app

- Over-tessellation will impact both frontend and backend performance
Distance Adaptive Tessellation

- Use the HS to determine the edge tessellation factors based on distance from view point
- If using a CPU check on the bounding volume to switch tessellation off:
  
  HS should use the same falloff values to avoid tessellation popping
Distance Adaptive Tessellation: ON
Orientation Adaptive Tessellation

- Compute dot product of average edge normal with view vector
- Back facing patches either:
  - Use lower tessellation factors
  - Get culled altogether
- Silhouette patches use higher factors
  
  \[ \text{EdgeScale} = 1.0f - \text{abs}( \text{dot}( N, V ) ) \]

- Perfect for PN-Triangles
  
  \~3x\ gain at tessellation factor 9
Orientation Adaptive Tessellation: ON
Density Adaptive Tessellation

- Pre-compute tessellation factors from displacement maps
- Calculate gradients from height values
- Create a buffer of patch edge tessellation factors
- Sample buffer in your HS to determine tessellation factor
Density Adaptive Tessellation: OFF

Density Adaptive Tessellation: ON
Screen Space Adaptive Tessellation

- Triangles under 8 pixels are not efficient
- Consider limiting the global maximum TessFactor by screen resolution
- Consider the screen space patch edge length as a scaling factor
  - Watch out for patches at a skew angle to the camera
  - May need to tweak how this works
Draw Tessellated Meshes Together

<table>
<thead>
<tr>
<th>Bad</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Draw</td>
</tr>
<tr>
<td>Draw_Tessellated</td>
<td>Draw</td>
</tr>
<tr>
<td>Draw</td>
<td>Draw_Tessellated</td>
</tr>
<tr>
<td>Draw_Tessellated</td>
<td>Draw_Tessellated</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Toggling tessellation is a large state change
- Minimize these transitions
Consider Using Stream Out

- If you render tessellated meshes multiple times consider streaming out the tessellated mesh
  - Shadow map slices
  - Lighting passes
- Then render multiple times through the non-tessellated pipeline
  - Make sure you measure, this may not help performance!
General Rules

- Compute complex things as early in the pipeline as possible
  - VS possible?
  - HS possible?
  - DS possible?
  - If not, then PS

- Try to minimize number of attributes coming to PS stage
Hull Shader Tips : 1

- A long HS can affect performance at low tessellation factors
  Keep simple
  Move work to the VS if possible
- Minimize vertex data passed from the VS
- Minimize data passed to the DS
- Specify maxtessfactor() with HS
  May help the driver to optimize the workload
Hull Shader Tips : 2

Use a PASS-THROUGH Control Point Phase

Only requires 1 HW thread
Shading in the Domain Shader

- Can hoist lower-frequency computation from PS to DS
  E.g. ambient/volumetric lighting
  Test to see if this is a performance win – it may well not be!

- This often works best with uniform sampling of surface
  Tessellation with uniform screen space triangle sizes
  Aim for rasterizer “sweet spot”
Shading in the Domain Shader

Example: underwater caustics
Shading in the Domain Shader

Example: Fourier Opacity Maps
Domain Shader Tips

- A long DS can dominate performance at high tessellation factors
  - Keep simple
- Calculate mip level for sampling displacement maps
  - Avoid thrashing the texture cache
- Minimize data passed to GS / PS
Summary

- Tessellation can be a big quality and performance win
- Use occlusion & culling
- Disable tessellation if not needed
- Aggressively use adaptive tessellation techniques
- Keep HS and DS stages as simple as possible
- Use this killer DX11 feature to make games look awesome...😊
Questions?

jon.story@amd.com

cem@nvidia.com