OpenGL 4.0 Tessellation For Professional Applications
San Jose, CA | Sept. 21, 2010
Agenda

- Motivation
- OpenGL 4.0 Tessellation Pipeline
- Case study
  - Massive terrain rendering
Motivation

- Visual quality
- Memory Bandwidth
- Dynamic LOD
- Perform computations at lower frequency
The OpenGL 3.x pipeline

- Vertex
- Geometry
- Fragment
The OpenGL 4.x pipeline

- Vertex
- Tessellation Control
- Tessellator
- Tessellation Evaluation
- Geometry
- Fragment

ARB_tessellation_shader
The OpenGL 4.x pipeline

- 2 new Programmable stages
  - Tessellation Control Shader \((\text{GL\_TESS\_CONTROL\_SHADER})\)
  - Tessellation Evaluation Shader \((\text{GL\_TESS\_EVALUATION\_SHADER})\)

- 1 new Fixed function stage
  - tessellation primitive generator aka tessellator

- 1 new primitive type
  - Patches \((\text{GL\_PATCHES})\)
The patch primitive

- Arbitrary number of vertices (1 to 32)
  - `glPatchParameteri(GL_PATCH_VERTICES, patchVCount)`
- Only primitive type allowed when a tessellation control shader is active
- No implied geometric ordering
OpenGL Tessellation - Setup

```c
char *tcsSource;  // Null terminated string
GLuint tcs = glCreateShader(GL_TESS_CONTROL_SHADER);
glShaderSource(tcs, 1, tcsSource, NULL);
glCompileShader(tcs);
glAttachShader(program, tcs)

char* tesSource;  // Null terminated string
GLuint tes = glCreateShader(GL_TESS_EVALUATION_SHADER);
glShaderSource(tes, 1, tesSource, NULL);
glCompileShader(tes);
glAttachShader(program, tes);

glLinkProgram(program);
```
Tessellation Control Shader (TCS)

- Runs once for each vertex
- Computes LOD per patch
  - `gl_TessLevelOuter[4]`
  - `gl_TessLevelInner[2]`

- Optional
  - If not present tessellation level will be set to their default value
  - Default value can be changed using:
    - `glPatchParameterfv(GL_PATCH_DEFAULT_OUTER_LEVEL, outerLevels)`
    - `glPatchParameterfv(GL_PATCH_DEFAULT_INNER_LEVEL, innerLevels)`
Tessellation Control Shader (TCS)

- Patch discarded if
  - \( \text{gl\_TessLevelOuter}[x] \leq 0 \) (Usefull for Culling)
  - \( \text{gl\_TessLevelOuter}[x] = \text{NaN} \)
Tessellation Control Shader: Sample

```glsl
layout(vertices = 3) out;
uniform float tessLevelOuter;
uniform float tessLevelInner;
void main()
{
    gl_TessLevelOuter[0] = tessLevelOuter;
    gl_TessLevelOuter[1] = tessLevelOuter;
    gl_TessLevelOuter[2] = tessLevelOuter;
    gl_TessLevelInner[0] = tessLevelInner;
    gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;
}
```
Tessellator

- Uses tessellation levels to decompose a patch into a new set of primitive
- Each vertex is assigned a \((u, v)\) or \((u, v, w)\) coordinate

![Diagram showing tessellation levels with different outer and inner values](image.png)

- Outer = 3, Inner = 3
- Outer = 5, Inner = 5
- Outer = 7, Inner = 7
Tessellation Evaluation Shader (TES)

- Compute the position of each vertex produced by the tessellator
- Control the tessellation pattern
- Can specify orientation of generated triangles
  - ccw (default)
  - cw
- Capable of generating points instead of lines or triangles
  - point_mode
TES: Layout

- **equal_spacing**
  - \( \text{tessLevel} = \text{clamp}(\text{tessLevel}, 1, \text{maxTessLevel}) \)
  - Rounded to nearest integer

- **fractional_even_spacing**
  - \( \text{tessLevel} = \text{clamp}(\text{tessLevel}, 2, \text{maxTessLevel}) \)
  - Rounded to next even integer

- **fractional_odd_spacing**
  - \( \text{tessLevel} = \text{clamp}(\text{tessLevel}, 1, \text{maxTessLevel} - 1) \)
  - Rounded to next ... odd integer 😊
Tessellation Evaluation Shader: Sample

```glsl
layout(triangles, equal_spacing, ccw) in;

void main()
{
    gl_Position = vec4(gl_In[0].gl_Position.xyz * gl_TessCoord.x +
                       gl_In[1].gl_Position.xyz * gl_TessCoord.y +
                       gl_In[2].gl_Position.xyz * gl_TessCoord.z +
                       1.0);
}
```
Tessellation Schemes

- Flat
- PN Triangles
- Gregory Patches
Water-tight tessellation

- Cracks may occur due to floating point precision
  \[-a + b + c \neq c + b + a\]
- Use GLSL `precise` qualifier
  - Ensure computations are done in their stated order

```glsl
precise out vec4 position;
out vec4 position;
precise position; // make existing variable precise
```
Case Study: Massive Terrain Rendering

- ROAM
- Geometry Clipmaps
- Chunked LOD
- ...

- So many algorithms, why do we need tessellation?
Case Study: Massive Terrain Rendering

- **Dataset**
  - ~600MB: space required to store earth elevation data with a km precision
  - 600GB+: current publicly available resolution (30m SRTM)

- **Enhance existing algorithms**
  - “Hybrid“ Chunked LOD
    - Static chunk of terrain can be replaced by a grid of GL_PATCHES
Tessellation heuristics (TCS)

- Distance adaptive tessellation
  - Use the TCS to determine the edge tessellation level based on the distance to the viewpoint

- Orientation adaptive tessellation
  - Compute the dot product of the average patch normal with the eye vector (can be done offline, using CUDA)

- Screenspace adaptive tessellation
  - Compute edge midpoint screen space error metric
  - Use edge bounding sphere for rotation invariant heuristic
Performance Considerations

- Tessellation pipeline is not free!
- Avoid running tessellation shaders when not necessary
  - “Cache” tessellation results using Transform Feedback
    - Don’t forget to switch to GL_TRIANGLES when disabling tessellation
Frutum & Occlusion Culling

- Consider using the Tessellation Control Shader to Cull patches not in frustum
  - $\text{gl}_\text{TessLevelOuter}[x] = 0$
  - Don’t forget to take displacement into consideration

- Don’t render occluded patches
  - Use occlusion queries
Other use cases

- High quality rendering
- Vector Rendering
- Hair Rendering
Conclusion

- OpenGL tessellation pipeline can greatly enhance the visual quality of your application
- Can adapt to existing rendering pipelines
- Implement efficiently
- Great for Simulators and GIS applications
Questions?

Thank you