Tesselation Shaders

Thanks to Mike Bailey (OSU)

Why a tessellation shader

- You can perform adaptive subdivision based on a variety of criteria (size, curvature, etc.)
- Your application can provide coarser models (+ geometric compression)
- You can apply detailed displacement maps without supplying equally detailed geometry
- You can adapt visual quality to the required level of detail
- You can create smoother silhouettes

What patterns can the Tessellation shaders produce?

- Lines
- Triangles
- Quads (output as triangles)

Another view of the Tess Shader

Figure 8.1: Schematic of OpenGL tessellation
**In the OpenGL Program**

```
if (you have a TCS, you must also have a Vertex Shader)

// Triangle Quad Isolines

TCS Inputs

• `gl_in[]` is an array of structures containing:
  - `gl_Position`
  - `gl_PointSize`
  - `gl_ClipDistance[]`

• `gl_invocationID` tells you which output vertex you are working on. This must be the index into the `gl_out[]` array.

• `gl_PatchVerticesIn` is the number of vertices in each patch and the dimension of `gl_in[]`

• `gl_PrimitiveID` is the number of primitives since last `glBegin()` (the first one is #0)

• `barrier()` causes all instances of TCS’s to wait here

TCS Outputs

• `gl_out[]` is an array of structures containing:
  - `gl_Position`
  - `gl_PointSize`

• All invocations of the TCS have read-only access to all the output information. `barrier()` causes all instances of TCS’s to wait here

• `layout(vertices = n; out; Used to specify the number of vertices output to the TPS)`

• Defining how many vertices this patch will output:
  ```
  layout(vertices = 16; out);
  ```

• `gl_TessLevelOuter[4]` is an array containing up to 4 edges of tessellation levels

• `gl_TessLevelInner[4]` is an array containing up to 2 edges of tessellation levels

• User-defined variables defined per-vertex are qualified as “out”

• User-defined variables defined per-patch are qualified as “patch out”

Tessellation Primitive Generator

• Is “fixed-function”, i.e., you can’t change its operation except by setting parameters

• Consumes all vertices from the TCS and emits tessellated triangles, quads, or lines

• Outputs positions (vertices) as coordinates in barycentric (u,v,w)

• All three coordinates (u,v,w) are used for triangles

• Just (u,v) are used for quads and isolines

```
**TES Output Topologies: the Quad Pattern**

```
GLuint glTessLevelInner[0] = 2;
GLuint glTessLevelInner[1] = 1;
GLuint glTessLevelOuter[0] = 2;
GLuint glTessLevelOuter[1] = 3;
GLuint glTessLevelOuter[2] = 2;
GLuint glTessLevelOuter[3] = 4;
```
Demo tessmodes

TES subdivision

- layout(triangles, equal_spacing, ccw) in;
- equal_spacing means that the triangle edges will be subdivided into segments with equal lengths (according to the TLs).
- fractional_even_spacing means if there is a fractional portion based on TLs, it is evenly split between the ends.
- fractional_odd_spacing means if there is a fractional portion based on TLs, it is not evenly split between the ends.

Demo tesssubdivisionmodes

Example: A Bézier Curve

\[ P(u) = (1-u)^3 P_0 + 3u(1-u)^2 P_1 + 3u^2(1-u)P_2 + u^3 P_3 \]
Example: A Bézier Curve

\[ P(u) = \binom{3}{0}u^0(1-u)^3P_0 + \binom{3}{1}u^1(1-u)^2P_1 + \binom{3}{2}u^2(1-u)P_2 + \binom{3}{3}u^3P_3 \]

1. The Tessellation Control Shader figures how much to tessellate the curve based on screen area, curvature, etc. Can tessellate non-uniformly if desired.

The OpenGL tessellation can also do 3D curves. Just set GL_E = 1.

2. The Tessellation Primitive Generator generates \( u, v, w \) values for as many subdivisions as the TCS asked for.

Example: A Bézier Curve

\[ P(u) = \binom{3}{0}(-u^3P_0 - 3u^2P_1 - 3uP_2 - uP_3) + \binom{3}{1}u^3P_0 + \binom{3}{2}u^2P_1 + \binom{3}{3}uP_2 \]

3. The Tessellation Evaluation Shader computes the \( x, y, z \) coordinates based on the TPG's \( u \) values.

In the OpenGL Program

```glsl
#version 430

in vec3 InPos;

out vec3 OutPos;

void main() {
    OutPos = InPos;
    // ... shader code...
}
```

In the TCS Shader

```glsl
#version 430

in vec3 InPos;

out vec3 OutPos;

void main() {
    OutPos = InPos;
    // ... shader code...
}
```

In the TES Shader

```glsl
#version 430

in vec3 InPos;

out vec3 OutPos;

void main() {
    OutPos = InPos;
    // ... shader code...
}
```

Assigning the intermediate pins is here to make the code more readable. We assume that the compiler will optimize this away.
Example: A Bézier Curve

\[
P(t) = \left(\begin{array}{c} 3(1-t)^3 \quad 3t(1-t)^2 \quad 3t^2(1-t) \quad 3t^3 \\
(1-t)^3 \quad 3t^2(1-t) \quad 3t(1-t)^2 \quad t^3 
\end{array} \right) \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}
\]

Outer 1 = 5

Bézier Surface Parametric Equations

In the OpenGL Program

```c
// In the OpenGL Program

glInit(16); 

gBegin(GL_TRIANGLES); 

gVertex3f(x0, y0, z0); 

gVertex3f(x1, y1, z1); 

gVertex3f(x2, y2, z2); 

gVertex3f(x3, y3, z3); 

gEnd(); 
```

In the TCS Shader

```c
// In the TCS Shader

uniform float u; 

tex_coord = vec4(u, 0, 1, 1); 
```

In the TES Shader

```c
// In the TES Shader

vertex = vec3(u, 0, z0); 
```

In the TES Shader – Computing the Position

```c
// In the TES Shader – Computing the Position

final_pos = gl_Position; 
```

Assigning the information psi is here to make this code more noticeable. We assume that the compiler will optimize this away.
In the TES Shader – Computing the Normal

Example: A Bézier Surface

Smoothing edge boundaries is one of the reasons that you can set Outer and Inner tessellation levels separately.

Example: Whole-Sphere Subdivision

Example: Whole-Sphere Subdivision

Using the sphere_subdiv entry in the shader parameters, you can control the number of segments around the equator of the sphere.
Example: PN Triangles

General idea: trim each triangle into a triangular wedge patch
Create the triangle centers by using the surface normals at the corner vertices. The triangular patch equation can then be discretized
at any level of tessellation.

The Cow's Tail is a Good Example of using PN Triangles

Demonstrating the Limits of Tessellation Shaders

Demo Displacement

The Difference Between Tessellation Shaders and Geometry Shaders

By now, you are probably confused about when to use a Geometry Shader and when to use a Tessellation Shader. Both are capable of creating new geometry from existing geometry. See if this helps.

Use a Geometry Shader when:
1. You need to convert geometry topologies, such as the silhouette and headlight shapes. Changes - lines to triangles, triangles to points.
2. You need some sort of geometry processing to come after the Tessellation Shader (such as how the inlay shader was used here).

Use a Tessellation Shader when:
1. You need to generate many new vertices and one of the tessellation topologies will suit your needs.
2. Use a Tessellation Shader when you need to generate more than 6 input vertices to define the surface being tessellated.