Why do we need a Tessellation step right in the pipeline?

- You can perform adaptive subdivision based on a variety of criteria (size, curvature, etc.)
- You can provide coarser models, but have finer ones displayed (∝ geometric compression)
- You can apply detailed displacement maps without supplying equally detailed geometry
- You can create smoother silhouettes
- You can do all of this, and someone else will supply the patterns!

What built-in patterns can the Tessellation shaders produce?

- Lines
- Triangles
- Quads (subsequently broken into triangles)

The Tessellation Shaders: Where Do they Fit in the Pipeline?

- Tessellation Shader Organization

In the OpenGL Program

```glsl
#version 400
extension GL_ARB_tessellation_shader : enable

GLuint tcs = glCreateShader( GL_TESS_CONTROL_SHADER );
GLuint tes = glCreateShader( GL_TESS_EVALUATION_SHADER );

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

Check the OpenGL extension:

- "GL_ARB_tessellation_shader"

In GLSL:

```glsl
#extension GL_ARB_tessellation_shader : enable
```
TCS Inputs

`gl_in` is an array of structures:

```
struct {
    vec4 gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[6];
} gl_in[n];
```

`gl_InvocationID` tells you which output vertex you are working on. This must be the index into the `gl_in` 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 the last ` glBegin()` (the first one is #0).

```
layout( vertices=n ) out;
```

Used to specify the number of vertices output to the TPG.

```
struct {
    vec4 gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[6];
} gl_out[n];
```

In the TCS

User-defined variables defined per-vertex are qualified as 'out'.
User-defined variables defined per-patch are qualified as 'patch out'.

Defining how many vertices this patch will output:

```
layout( vertices=16 ) out;
```

TCS Outputs

`gl_out` is an array of structures:

```
struct {
    vec4 gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[6];
} gl_out[n];
```

All invocations of the TCS have read-only access to all the output information.

```
layout( vertices=n ) out;
```

TES Inputs

Reads one vertex of 0 <= (u,v,w) <= 1 coordinates in variable vec3 `gl_TessCoord`.

User-defined variables defined per-vertex are qualified as 'out'.
User-defined variables defined per-patch are qualified as 'patch out'.

```
struct {
    vec4 gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[6];
} gl_in[n];
```

TES Output Topologies: the Quad Pattern

```
layout( triangles | quads | isolines ) in;
```

- The TPG is “fixed-function”, i.e., you can’t change its operation except by setting parameters.
- Consumes all vertices from the TCS and emits vertices for the triangles, quads, or isolines patterns.
- TPG outputs a series of vertices as coordinates in barycentric (u,v,w) parametric space.
- All three coordinates (u,v,w) are used for triangles.
- Just (u,v) are used for quads and isolines.

```
layout( tessels = 4 ) in;
```

- `gl_TessLevelOuter[4]` is an array containing up to 4 edges of tessellation levels.
- `gl_TessLevelInner[2]` is an array containing up to 2 edges of tessellation levels.
**Examples**

In these examples:

1. We are using glman to run them. The only necessary input files are the glman .glib file and the shader files. If you aren’t using glman, you can do this from a full OpenGL program.

2. All of the surface examples use the Geometry Shader triangle-shrink shader. This isn’t necessary, but is educational to really see how much and where the surfaces have been tessellated.

Example: A Bézier Curve

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

1. You program the Tessellation Control Shader to decide how much to tessellate the curve based on screen area, curvature, etc.

   You can even tessellate non-uniformly if you want.

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

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

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

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

\[ P(u) = u^3(-P_0 + 3P_1 - 3P_2 + P_3) + u^2(3P_0 - 6P_1 + 3P_2 - P_3) + u(-3P_0 + 3P_1 - P_2) + P_3 \]

In an OpenGL Program

```glsl
glPatchParameteri(GL_PATCH_VERTICES, 4);
gBegin(GL_PATCHES);
    glVertex3f(x0, y0, z0);
    glVertex3f(x1, y1, z1);
    glVertex3f(x2, y2, z2);
    glVertex3f(x3, y3, z3);
gEnd();
```

In a .glib File

```glsl
#version 400
#extension GL_ARB_tessellation_shader: enable
uniform int uOuter0, uOuter1;
layout(vertices = 4) out;
void main()
{
    vec4 p0 = gl_in[0].gl_Position;
    vec4 p1 = gl_in[1].gl_Position;
    vec4 p2 = gl_in[2].gl_Position;
    vec4 p3 = gl_in[3].gl_Position;
    float u = gl_TessCoord.x;
    // the basis functions:
    float b0 = (1.-u) * (1.-u) * (1.-u);
    float b1 = 3. * u * (1.-u) * (1.-u);
    float b2 = 3. * u * u * (1.-u);
    float b3 = u * u * u;
    gl_Position = b0*p0 + b1*p1 + b2*p2 + b3*p3;
}
```

Assigning the intermediate pi's is here to make the code more readable. From what I have seen, the compiler will optimize this away.

Example: A Bézier Curve

\[ Outer1 = 5 \]

\[ Outer1 = 50 \]
Example: A Bézier Surface

Bézier Surface Parametric Equations

In an OpenGL Program

In the .glib File

In the TCS Shader

In the TES Shader
In the TES Shader – Computing the Position, given a u and v

```glsl```
// the basis functions:
float bu0 = (1.-u) * (1.-u) * (1.-u);
float bu1 = 3. * u * (1.-u) * (1.-u);
float bu2 = 3. * u * u * (1.-u);
float bu3 = u * u * u;
float dbu0 = -3. * (1.-u) * (1.-u);
float dbu1 =  3. * (1.-u) * (1.-3.*u);
float dbu2 =  3. * u *      (2.-3.*u);
float dbu3 =  3. * u *      u;

float bv0 = (1.-v) * (1.-v) * (1.-v);
float bv1 = 3. * v * (1.-v) * (1.-v);
float bv2 = 3. * v * v * (1.-v);
float bv3 = v * v * v;
float dbv0 = -3. * (1.-v) * (1.-v);
float dbv1 =  3. * (1.-v) * (1.-3.*v);
float dbv2 =  3. * v *      (2.-3.*v);
float dbv3 =  3. * v *      v;

// finally, we get to compute something:
gl_Position = bu0 * ( bv0*p00 + bv1*p01 + bv2*p02 + bv3*p03 )
       + bu1 * ( bv0*p10 + bv1*p11 + bv2*p12 + bv3*p13 )
       + bu2 * ( bv0*p20 + bv1*p21 + bv2*p22 + bv3*p23 )
       + bu3 * ( bv0*p30 + bv1*p31 + bv2*p32 + bv3*p33 );
```
Example: Whole-Sphere Subdivision

```
sphereadapt.tcs

#version 400 compatibility
in float uDetail;  
uniform float uScale;  
patch in vec3  tcCenter;  
patch in float   tcRadius;  
in vec3  vCenter[],  
in float   vRadius[];  
layout( quads, equal_spacing, ccw)  in;

const float PI = 3.14159265;
out vec3          teNormal;  
patch in vec3  tcCenter;  
patch in float   tcRadius;  
layout( quads, equal_spacing, ccw)  in;

void main( )
{  
    gl_TessLevelInner[] = dmax * uDetail;  
    gl_TessLevelOuter[] = 2.;  
    gl_TessLevelOuter[] = dmax * uDetail;  
    gl_TessLevelOuter[] = 2.;  
    float dmax = sqrt( dx*dx + dy*dy + dz*dz );  
    float dy = distance( my.xy, py.xy );  
    float dx = distance( mx.xy, px.xy );  
    py.xy /= py.w;  
    my.xy /= my.w;  
    px.xy /= px.w;  
    mx.xy /= mx.w;  
    pz = gl_ModelViewProjectionMatrix * pz;  
    mz = gl_ModelViewProjectionMatrix * mz;  
    py = gl_ModelViewProjectionMatrix * py;  
    my = gl_ModelViewProjectionMatrix * my;  
    px = gl_ModelViewProjectionMatrix * px;  
    mx = gl_ModelViewProjectionMatrix * mx;  
    gl_TessLevelInner[] = uScale * tcRadius * uDetail;  
    gl_TessLevelOuter[] = uScale * tcRadius * uDetail;  
    gl_TessLevelOuter[] = 2.;  
    gl_TessLevelOuter[] = uScale * tcRadius * uDetail;  
    gl_TessLevelOuter[] = 2.;  
    tcRadius = vRadius[0];  
    tcCenter = vCenter[0];  
    gl_out[ gl_InvocationID ].gl_Position = gl_in[0].gl_Position;  
    xyz = vec3( cosphi*cos(theta), sin(phi), cosphi*sin(theta) );  
    float cosphi = cos(phi);  
    float phi = PI * ( u - .5 );  
    float w = gl_TessCoord.z;  
    float v = gl_TessCoord.y;  
    float u = gl_TessCoord.x;  
    vec3 p = gl_in[0].gl_Position.xyz;  
    vec4 mz = vec4( vCenter[0] - vec3( 0., 0., vRadius[0] ), 1. );  
    vec4 py = vec4( vCenter[0] + vec3( 0., vRadius[0], 0. ), 1. );  
    vec4 my = vec4( vCenter[0] - vec3( 0., vRadius[0], 0. ), 1. );  
    vec4 px = vec4( vCenter[0] + vec3( vRadius[0], 0., 0. ), 1. );  
    teNormal = vec3( cosphi*cos(theta), sin(phi), cosphi*sin(theta) );  
}
```

Example: Whole-Sphere Subdivision

```
sphereadapt.tes

#version 400 compatibility
uniform float  uScale;  
patch in vec3  tcCenter;  
patch in float   tcRadius;  
layout( quads, equal_spacing, ccw)  in;

const float PI = 3.14159265;
out vec3          teNormal;  
patch in vec3  tcCenter;  
patch in float   tcRadius;  
layout( quads, equal_spacing, ccw)  in;

void main( )
{  
    gl_TessLevelInner[] = dmax * uDetail;  
    gl_TessLevelOuter[] = 2.;  
    gl_TessLevelOuter[] = dmax * uDetail;  
    gl_TessLevelOuter[] = 2.;  
    float dmax = sqrt( dx*dx + dy*dy + dz*dz );  
    float dy = distance( my.xy, py.xy );  
    float dx = distance( mx.xy, px.xy );  
    py.xy /= py.w;  
    my.xy /= my.w;  
    px.xy /= px.w;  
    mx.xy /= mx.w;  
    pz = gl_ModelViewProjectionMatrix * pz;  
    mz = gl_ModelViewProjectionMatrix * mz;  
    py = gl_ModelViewProjectionMatrix * py;  
    my = gl_ModelViewProjectionMatrix * my;  
    px = gl_ModelViewProjectionMatrix * px;  
    mx = gl_ModelViewProjectionMatrix * mx;  
    gl_TessLevelInner[] = uScale * tcRadius * uDetail;  
    gl_TessLevelOuter[] = uScale * tcRadius * uDetail;  
    gl_TessLevelOuter[] = 2.;  
    gl_TessLevelOuter[] = uScale * tcRadius * uDetail;  
    gl_TessLevelOuter[] = 2.;  
    tcRadius = vRadius[0];  
    tcCenter = vCenter[0];  
    gl_out[ gl_InvocationID ].gl_Position = gl_in[0].gl_Position;  
    xyz = vec3( cosphi*cos(theta), sin(phi), cosphi*sin(theta) );  
    float cosphi = cos(phi);  
    float theta = 2. * PI * ( v - .5 );  
    float w = gl_TessCoord.z;  
    float v = gl_TessCoord.y;  
    float u = gl_TessCoord.x;  
    vec3 p = gl_in[0].gl_Position.xyz;  
    vec4 mz = vec4( vCenter[0] - vec3( 0., 0., vRadius[0] ), 1. );  
    vec4 py = vec4( vCenter[0] + vec3( 0., vRadius[0], 0. ), 1. );  
    vec4 my = vec4( vCenter[0] - vec3( 0., vRadius[0], 0. ), 1. );  
    vec4 px = vec4( vCenter[0] + vec3( vRadius[0], 0., 0. ), 1. );  
    teNormal = vec3( cosphi*cos(theta), sin(phi), cosphi*sin(theta) );  
}
```
Making the Whole-Sphere Subdivision Adapt to Screen Coverage

General idea: turn each triangle into a triangular Bézier patch. This is done by using the surface normals at the corner vertices. The Bézier patch equation can then be interpolated to any level of tessellation.

Example: PN Triangles

Observation: triangles are usually passed in with points (P) and normals (N). Using this method, those triangles can be broken into a series of smoother triangles internally. AMD actually had this in their drivers before tessellation shaders made it unnecessary.

Example: PN Triangles

Example: PN Triangles

Example: PN Triangles

Example: PN Triangles
Example: PN Triangles

```glsl
#version 400

in float gLightIntensity;
const vec3 COLOR = vec3(1., 1., 0.);

void main()
{
    gl_FragColor = vec4(gLightIntensity*COLOR, 1.);
}
```

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

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 an input topology into a different output topology, such as in the silhouette and hedgehog shaders (triangles → lines) or the explosion shader (triangles → points).
2. You need some sort of geometry processing to come after the Tessellation Shader (such as how the shrink shader was used).

**Use a Tessellation Shader when:**
1. One of the built-in tessellation patterns will suit your needs.
2. You need more than 6 input vertices to define the surface being tessellated.
3. You need more output vertices than a Geometry Shader can provide.

Demonstrating the Limits of Tessellation Shaders

This tessellation is using 8464 (the maximum allowed).
This is pretty good-looking, but doesn't come close to using the full 4096x2276 resolution available for the bump-map.