Tessellation Shaders

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 apply detailed normal maps without supplying equally detailed geometry
• You can adapt visual quality to the required level of detail
• You can create smoother silhouettes
• You can do all of this, and someone else will supply the geometric patterns for you!

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?

The Tessellation Control Shader (TCS) optionally can transform the input coordinates (but usually doesn’t). It also computes the required tessellation level based on distance to the eye, screen space spanning, hull curvature, or displacement roughness. There is one TCS execution per vertex.

The Fixed-Function Tessellation Primitive Generator (TPG) generates semi-regular u-v-w coordinates in specific patterns. (In fact, if it had been up to me, this would have been called the Tessellation Pattern Generator. Nobody asked.)

The Tessellation Evaluation Shader (TES) Turns the TPG’s u-v-w coordinates into x-y-z. It can apply displacements. There is one TES execution per generated vertex.

There is a new “Patch” primitive: `glBegin( GL_PATCHES )` followed by some number of `glVertex3f()` calls. There is no implied purpose, number of vertices, or vertex ordering – those are given by you in how you write the shader.
In the OpenGL Program

```c
GLenum tcs = glCreateShader( GL_TESS_CONTROL_SHADER );
GLenum tes = glCreateShader( GL_TESS_EVALUATION_SHADER );

Check the OpenGL extension:
"GL_ARB_tessellation_shader"

In GLSL:

```
**Examples**

In these examples:

1. I am 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 the surfaces have been tessellated.

**Examples**

Example: A Bézier Curve

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

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

Can even tessellate non-uniformly if you want, such as using more points where the curvature is higher.

The OpenGL tessellation can do 1D curves. Just set OL0 == 1.
2. The Tessellation Primitive Generator generates \( u \) values for as many subdivisions as the TCS asked for.

\[
P(u) = (1-u)^3 p_0 + 3u(1-u)^2 p_1 + 3u^2(1-u) p_2 + u^3 p_3
\]

where \( P \) is an abbreviation for \( \frac{\mathbf{v}}{u} \).
Example: A Bézier Curve

Outer1 = 5

Outer1 = 50

Example: A Bézier Surface

P_{00} \quad P_{10} \quad P_{20} \quad P_{30}

P_{01} \quad P_{11} \quad P_{21} \quad P_{31}

P_{02} \quad P_{12} \quad P_{22} \quad P_{32}

P_{03} \quad P_{13} \quad P_{23} \quad P_{33}

Bézier Surface Parametric Equations

P(u, v) = \left[ (1-u)^3 \right] \left[ 3u(1-u)^2 \right] \left[ 3u^2(1-u) \right] \left[ 1 \right]

\left[ \begin{array}{cccc}
P_{00} & P_{10} & P_{20} & P_{30} \\
4P_{00} & 4P_{10} & 4P_{20} & 4P_{30} \\
6P_{00} & 6P_{10} & 6P_{20} & 6P_{30} \\
4P_{00} & 4P_{10} & 4P_{20} & 4P_{30}
\end{array} \right]

In an OpenGL Program

```glsl
if (PatchParamType == GL_PATCH_VERTICES) {
    glBegin(GL_PATCHES);
    glVertex3f(x00, y00, z00);
    glVertex3f(x10, y10, z10);
    glVertex3f(x20, y20, z20);
    glVertex3f(x30, y30, z30);
    glVertex3f(x01, y01, z01);
    glVertex3f(x11, y11, z11);
    glVertex3f(x21, y21, z21);
    glVertex3f(x31, y31, z31);
    glVertex3f(x02, y02, z02);
    glVertex3f(x12, y12, z12);
    glVertex3f(x22, y22, z22);
    glVertex3f(x32, y32, z32);
    glVertex3f(x03, y03, z03);
    glVertex3f(x13, y13, z13);
    glVertex3f(x23, y23, z23);
    glVertex3f(x33, y33, z33);
    glEnd();
    This order is not set by OpenGL. It is set by you.
    Pick a convention yourself and stick to it!
    GLSL doesn't care as long as you are consistent.
}
```

In the .glib File

```glsl
##OpenGL GLIB
Perspective 70
Vertex                beziersurface.vert
Fragment           beziersurface.frag
TessControl beziersurface.tcs
TessEvaluation beziersurface.tes
Geometry           beziersurface.geom
Program BezierSurface uOuter02 <1 10 50>  uOuter13 <1 10 50>  uInner0 <1 10 50>  uInner1 <1 10 50>  
  uShrink <0. 1. 1.>                                                                                             
  u LightX <-10. 0. 10.>  u LightY <-10. 10. 10.>   uLightZ <-10. 10. 10. >
Color  1. 1. 0. 1.
NumPatchVertices 16
```

In the TCS Shader

```glsl
#version 400
#extension GL_ARB_tessellation_shader : enable
uniform float uOuter02, uOuter13, uInner0, uInner1;
layout( vertices = 16 )  out;
void main() {
    gl_out[ gl_InvocationID ].gl_Position = gl_in[ gl_InvocationID ].gl_Position;
    gl_TessLevelOuter[0] = gl_TessLevelOuter[2] = uOuter02;
    gl_TessLevelInner[0] = uInner0;
    gl_TessLevelInner[1] = uInner1;
    glEnd();
}
```
In the TES Shader

Computing the Normal, given a u and v

```
vec4 dpdu = dbu0 * ( bv0*p00 + bv1*p01 + bv2*p02 + bv3*p03 )
+ dbu1 * ( bv0*p10 + bv1*p11 + bv2*p12 + bv3*p13 )
+ dbu2 * ( bv0*p20 + bv1*p21 + bv2*p22 + bv3*p23 )
+ dbu3 * ( bv0*p30 + bv1*p31 + bv2*p32 + bv3*p33 );

vec4 dpdv = bu0 * ( dbv0*p00 + dbv1*p01 + dbv2*p02 + dbv3*p03 )
+ bu1 * ( dbv0*p10 + dbv1*p11 + dbv2*p12 + dbv3*p13 )
+ bu2 * ( dbv0*p20 + dbv1*p21 + dbv2*p22 + dbv3*p23 )
+ bu3 * ( dbv0*p30 + dbv1*p31 + dbv2*p32 + dbv3*p33 );

tNormals = normalize( cross( dpdu.xyz, dpdv.xyz ) );
```

Tangent Vectors obtained by differentiating the position equation with respect to u and v:

```
vect dPdu = dbu0 * ( bv0*p00 + bv1*p01 + bv2*p02 + bv3*p03 )
+ dbu1 * ( bv0*p10 + bv1*p11 + bv2*p12 + bv3*p13 )
+ dbu2 * ( bv0*p20 + bv1*p21 + bv2*p22 + bv3*p23 )
+ dbu3 * ( bv0*p30 + bv1*p31 + bv2*p32 + bv3*p33 );

vect dPdv = bu0 * ( dbv0*p00 + dbv1*p01 + dbv2*p02 + dbv3*p03 )
+ bu1 * ( dbv0*p10 + dbv1*p11 + dbv2*p12 + dbv3*p13 )
+ bu2 * ( dbv0*p20 + dbv1*p21 + dbv2*p22 + dbv3*p23 )
+ bu3 * ( dbv0*p30 + dbv1*p31 + dbv2*p32 + dbv3*p33 );
```

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

Example: Whole-Sphere Subdivision

```
Example: A Bézier Surface
```

Using the x, y, z, and w to specify the center and radius of the sphere.
Example: Whole-Sphere Subdivision

```
main() {
  vCenter = gl_Vertex.xyz;
  vRadius = gl_Vertex.w;
  gl_Position = vec4( vCenter, 0., 0., 1. );
  // doesn't matter now – we will fill in the coords later
}
```

Making the Whole-Sphere Subdivision Adapt to Screen Coverage, I

```
main() {
  gl_TessLevelInner[1]  = dmax * uDetail;
  gl_TessLevelInner[0]  = dmax * uDetail;
  gl_TessLevelOuter[3] = dmax * uDetail;
  gl_TessLevelOuter[2] = 2.;
  gl_TessLevelOuter[1] = dmax * uDetail;
  gl_TessLevelOuter[0] = 2.;
  float dmax = sqrt( dx*dx + dy*dy + dz*dz );
  float dz = distance( mz.xy, pz.xy );
  float dy = distance( my.xy, py.xy );
  float dx = distance( mx.xy, px.xy );
  pz.xy /= pz.w;
  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[1]  = uScale * tcRadius * uDetail;
  gl_TessLevelInner[0]  = uScale * tcRadius * uDetail;
  gl_TessLevelOuter[3] = uScale * tcRadius * uDetail;
  gl_TessLevelOuter[2] = 2.;
  gl_TessLevelOuter[1] = uScale * tcRadius * uDetail;
  gl_TessLevelOuter[0] = 2.;
  tcRadius = vRadius[ 0 ];
  tcCenter = vCenter[ 0 ];
  gl_out[ gl_InvocationID ].gl_Position = gl_in[ 0 ].gl_Position; // (0,0,0,1)
}
```

Example: Whole-Sphere Subdivision

```
main() {
  float PI = 3.14159265;
  out vec3          teNormal;
  patch in vec3  tcCenter;
  patch in float   tcRadius;
  layout( quads, equal_spacing, ccw)  in;
  uniform float  uScale;
  #extension GL_ARB_tessellation_shader : enable
  #version 400 compatibility
  layout( vertices = 1 )  out;
  uniform float uDetail;
  patch out vec3  tcCenter;
  patch out float   tcRadius;
  in vec3  vCenter[ ];
  #extension GL_ARB_tessellation_shader : enable
  #version 400 compatibility
  sphereadapt.tcs, I
  spheresubd.tes
  spheresubd.vert
  // doesn't matter now – we will fill in the coords later
  Using the scale and the radius to help set the tessellation detail
}
```

Making the Whole-Sphere Subdivision Adapt to Screen Coverage, II

```
main() {
  // Extreme points of the sphere in Clip space
  float dx = distance( mx.xy, px.xy );
  float dy = distance( my.xy, py.xy );
  float dz = distance( mz.xy, pz.xy );
  float dmax = sqrt( dx*dx + dy*dy + dz*dz );
  float dx = distance( mx.xy, px.xy );
  float dy = distance( my.xy, py.xy );
  float dz = distance( mz.xy, pz.xy );
  float dmax = sqrt( dx*dx + dy*dy + dz*dz );
  // Extreme points of the sphere in NDC space
  ext_topright = vec4( 1., 1., 0., 0. );
  ext_topleft = vec4( -1., 1., 0., 0. );
  ext_botright = vec4( 1., -1., 0., 0. );
  ext_botleft = vec4( -1., -1., 0., 0. );
  // How long are the lines between the extreme points?
  How long are the lines between the extreme points?
  // No longer use uDetail or tcRadius.  But, we do use uDetail to provide a way to
  // convert from NDC to Screen Space or to indicate the quality you’d like
  // (i.e., uDetail depends on how good you want the spheres to look and on how large
  // the window is in pixels.)
}
```
Making the Whole-Sphere Subdivision Adapt to Screen Coverage, III

Spherical coordinates

General idea: turn each triangle into a triangular Bezier patch. Create the Bezier control points by using the surface normals at the corner vertices. The Bezier patch equation can then be interpolated to any level of tessellation.


Example: PN Triangles

Example: PN Triangles

Example: PN Triangles

Example: PN Triangles

Notice that the number of triangles adapts to the screen coverage of each sphere, and that the size of the tessellated triangles stays about the same, regardless of radius or transformation.

Example: PN Triangles

Example: PN Triangles
Example: PN Triangles

```
#version 400 compatibility
#extension GL_gpu_shader4: enable
#extension GL_geometry_shader4: enable
uniform float   uShrink;
in vec3            teNormal[];
out float          gLightIntensity;
const vec3     LIGHTPOS = vec3( 5., 10., 10. );
vec3 V[3];
vec3 CG;
void
ProduceVertex( int v )
{
  gLightIntensity = abs( dot( normalize(LIGHTPOS - V[v]), normalize(teNormal[v]) ) );
  gl_Position = gl_ProjectionMatrix * vec4( CG + uShrink * ( V[v] - CG ), 1. );
  EmitVertex( );
}
void main( )
{
  V[0]  =   gl_PositionIn[0].xyz;
  V[1]  =   gl_PositionIn[1].xyz;
  ProduceVertex( 0 );
  ProduceVertex( 1 );
  ProduceVertex( 2 );
}
```

```
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

Notice how much improvement there is just by increasing the outer tessellation. This is because smooth shading already helps the inner parts of triangles, but does nothing for the silhouettes.

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 hedgetop 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 64x64 (the maximum allowed).
This is pretty good-looking, but doesn't come close to using the full 4096x2276 resolution available for the bump-map.