

1

Tessellation Shaders

This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#)

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

Four images showing different tessellation results: a sphere, a cube, a sphere with a grid, and a sphere with a grid.

2

Tessellation Fun Facts

I once won an OpenGL t-shirt at the SIGGRAPH conference by knowing how to correctly spell "tessellation". (It's 2 s's and 2 l's.)

The only reason I knew it was that the week before I had been experimenting with the newly-released tessellation shaders for the first time and my program wouldn't compile because I had misspelled tessellation. The week of the conference, it was still fresh in my mind how to spell it correctly.

Please don't tell anyone. Spelling correctly undermines one's CS credibility... ©

3

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 ($\approx$ 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)

4

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

Legend:
 = Fixed Function
 = Programmable

Annotations:
 - If Tessellation Shaders are in use, they are *always* the first stop after the Vertex Shader.
 - If Geometry Shaders are in use, they are *always* the last stop before the Rasterizer.

5

Tessellation Shader Organization

Flow:
 Transformed xyz Patch Vertices from the Vertex Shader → **Tessellation Control Shader** → **Tessellation Primitive Generator** → **Tessellation Evaluation Shader** → **Primitive Assembly** → xyz vertices

Annotations:
 - **Tessellation Control Shader:** One call per output vertex. Consumes the entire patch. Determines how much to tessellate.
 - **Tessellation Primitive Generator:** One call per patch. Tessellates the curve or surface into uvw coordinates.
 - **Tessellation Evaluation Shader:** One call per generated uvw vertex. Evaluate the curve or surface. Possibly apply a displacement map.

6

Tessellation Shader Organization

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 7

```

glPatchParameteri( GL_PATCH_VERTICES, num ); // # of vertices in each patch
glBegin( GL_PATCHES );
    glVertex3f( ... );
    glVertex3f( ... );
glEnd();

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

Check the OpenGL extension:
"GL_ARB_tessellation_shader"

In GLSL:
#version 400
#extension GL_ARB_tessellation_shader : enable

```

These have no implied topology – they will be given to you in an array. It's up to your shader to interpret the order

 mp - December 18, 2024

TCS Inputs 8

`gl_in[ ]` is an array of structures:

```

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

```

`gl_InvocationID` tells you which vertex you are working on. This is 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)

 mp - December 18, 2024

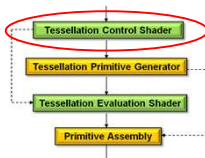
TCS Outputs 9

`gl_out[ ]` is an array of structures:

```

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

```



`layout( vertices = n ) out;` Used to specify the number of vertices sent to the TPG

`gl_TessLevelOuter[4]` is a built-in array containing up to 4 outside edges of tessellation levels

`gl_TessLevelInner[2]` is a built-in array containing up to 2 inside edges of tessellation levels

 mp - December 18, 2024

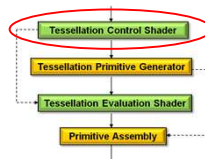
In the TCS 10

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 use:

`layout( vertices = 16 ) out;`



 mp - December 18, 2024

TES Inputs 11

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

User-defined variables defined per-vertex are qualified as "out"

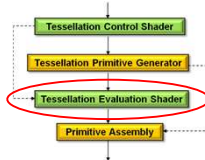
User-defined variables defined per-patch are qualified as "patch out"

`gl_in[ ]` is an array of structures coming from the TCS:

```

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

```

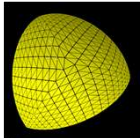


`layout( {triangles, quads, isolines}, {equal_spacing, fractional_even_spacing, fractional_odd_spacing}, {ccw, cw}, point_mode ) in;`

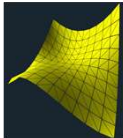
 mp - December 18, 2024

Tessellation Primitive Pattern Generator (TPG) 12

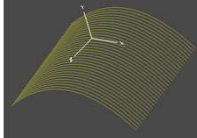
- The TPG is "fixed-function", i.e., you can't change its operation except by setting parameters
- The TPG consumes all vertices from the TCS and emits vertices for the **triangles, quads, or isolines** patterns
- The TPG outputs a series of vertices as coordinates in barycentric, i.e., in terms of the parameters (u,v,w)
- Really, only (u,v) are unique: for triangles $w = 1. - u - v$
- Just (u,v) are used for quads and isolines



Triangle pattern



Quad pattern



Isoline pattern

 mp - December 18, 2024

TES Output Topologies: the Quad Pattern 13

`gl_TessLevelOuter[4]` is an array containing up to 4 outside edges of tessellation levels.
`gl_TessLevelInner[2]` is an array containing up to 2 inside edges of tessellation levels.

Oregon State University
Computer Graphics

rjb - December 18, 2024

TES Output Topologies: the Isolines Pattern 14

`gl_TessLevelOuter[4]` is an array containing up to 4 outside edges of tessellation levels.
`gl_TessLevelInner[2]` is an array containing up to 2 inside edges of tessellation levels.

Oregon State University
Computer Graphics

rjb - December 18, 2024

TES Output Topologies: the Triangle Pattern 15

`gl_TessLevelOuter[4]` is an array containing up to 4 outside edges of tessellation levels.
`gl_TessLevelInner[2]` is an array containing up to 2 inside edges of tessellation levels.

Oregon State University
Computer Graphics

rjb - December 18, 2024

Examples 16

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.

Oregon State University
Computer Graphics

rjb - December 18, 2024

Example: A Bézier Curve 17

$$P(u) = (1-u)^3 P_0 + 3u(1-u)^2 P_1 + 3u^2(1-u) P_2 + u^3 P_3$$

Need to pass 4 points in to define the curve. Need to pass N points out to draw the curve as a line strip.

Oregon State University
Computer Graphics

rjb - December 18, 2024

Example: A Bézier Curve 18

$$P(u) = (1-u)^3 P_0 + 3u(1-u)^2 P_1 + 3u^2(1-u) P_2 + u^3 P_3$$

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

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`.

Oregon State University
Computer Graphics

rjb - December 18, 2024

Example: A Bézier Curve

2. The Tessellation Primitive Generator generates u values for as many subdivisions as the TCS asked for.

Oregon State University
Computer Graphics

19

Example: A Bézier Curve

3. The Tessellation Evaluation Shader computes the x, y, z coordinates based on the TPG's u values and P_0, P_1, P_2 , and P_3 .

$$P(u) = (1-u)^3 P_0 + 3u(1-u)^2 P_1 + 3u^2(1-u) P_2 + u^3 P_3$$

where P is an abbreviation for $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$

Oregon State University
Computer Graphics

20

In an OpenGL Program

```

In InitGraphics( ):
Pattern.Init( );
Pattern.Create( "pattern.vert", "pattern.tcs", "pattern.tes", "pattern.frag" );
Pattern.SetUniformVariable( "uOuter0", 20 );
Pattern.SetUniformVariable( "uOuter1", 10 );

In Display( ):
glPatchParameteri( GL_PATCH_VERTICES, 4 );

glBegin( GL_PATCHES );
glVertex3f( x0, y0, z0 );
glVertex3f( x1, y1, z1 );
glVertex3f( x2, y2, z2 );
glVertex3f( x3, y3, z3 );
glEnd( );

```

Oregon State University
Computer Graphics

21

In a .glsl File

```

##OpenGL GLSL
Perspective 70

Vertex      beziercurve.vert
Fragment    beziercurve.frag
TessControl  beziercurve.tcs
TessEvaluation beziercurve.tes
Program BezierCurve uOuter0 <0 1 5> uOuter1 <3 5 50>

Color 1. .5 0. 1.

NumPatchVertices 4
glBegin gl_patches
glVertex 0. 0. 0.
glVertex 1. 1. 1.
glVertex 2. 1. 0.
glVertex 3. 0. 1.
glEnd

```

Oregon State University
Computer Graphics

22

In the TCS Shader

```

#version 400
#extension GL_ARB_tessellation_shader: enable

uniform int uOuter0, uOuter1;

layout( vertices = 4 ) out;

void main( )
{
    gl_out[ gl_InvocationID ].gl_Position = gl_in[ gl_InvocationID ].gl_Position;

    gl_TessLevelOuter[0] = float( uOuter0 );
    gl_TessLevelOuter[1] = float( uOuter1 );
}

```

Oregon State University
Computer Graphics

23

In the TES Shader

```

#version 400
#extension GL_ARB_tessellation_shader: enable

layout( isolines, equal_spacing ) in;

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

Oregon State University
Computer Graphics

24

Example: A Bézier Curve 25

Outer1 = 5

Outer1 = 50

Oregon State University Computer Graphics

mp - December 18, 2024

Example: A Bézier Surface 26

Oregon State University Computer Graphics

mp - December 18, 2024

Bézier Surface Parametric Equations 27

$$P(u, v) = \begin{bmatrix} (1-u)^3 & 3u(1-u)^2 & 3u^2(1-u) & u^3 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} \begin{bmatrix} (1-v)^3 \\ 3v(1-v)^2 \\ 3v^2(1-v) \\ v^3 \end{bmatrix}$$

Oregon State University Computer Graphics

mp - December 18, 2024

In an OpenGL Program 28

```
glPatchParameteri( GL_PATCH_VERTICES, 16 );

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.

Oregon State University Computer Graphics

mp - December 18, 2024

In the .glib File 29

```
##OpenGL GLIB
Perspective 70

Vertex      beziensurface.vert
Fragment    beziensurface.frag
TessControl beziensurface.tcs
TessEval    beziensurface.tes
Geometry    beziensurface.geom
Program BezierSurface uOuter02 <1 10 50> uOuter13 <1 10 50> ulnner0 <1 10 50> ulnner1 <1 10 50> \
    uShrink <0.1 1> uLightX <-10.0 10.0> uLightY <-10.10 10.10> uLightZ <-10.10 10.10>

Color 1.1 0.1.

NumPatchVertices 16

glBegin gl_patches
glVertex 0.2 0.
glVertex 1.1 0.
glVertex 2.1 0.
glVertex 3.2 0.

glVertex 0.1 1.
glVertex 1.-2.1.
glVertex 2.1 1.
glVertex 3.0 1.

glVertex 0.0 2.
glVertex 1.1 2.
glVertex 2.0 2.
glVertex 3.-1.2.

glVertex 0.0 3.
glVertex 1.1 3.
glVertex 2.-1.3.
glVertex 3.-1.3.
glEnd
```

Oregon State University Computer Graphics

mp - December 18, 2024

In the TCS Shader 30

```
#version 400
#extension GL_ARB_tessellation_shader : enable

uniform float uOuter02, uOuter13, ulnner0, ulnner1;

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_TessLevelOuter[1] = gl_TessLevelOuter[3] = uOuter13;
    gl_TessLevelInner[0] = ulnner0;
    gl_TessLevelInner[1] = ulnner1;
}
```

Oregon State University Computer Graphics

mp - December 18, 2024

In the TES Shader 31

```
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

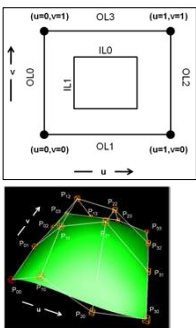
layout( quads, equal_spacing, ccw ) in;

out vec3 teNormal;

void main( )
{
    vec4 p00 = gl_in[ 0 ].gl_Position;
    vec4 p10 = gl_in[ 1 ].gl_Position;
    vec4 p20 = gl_in[ 2 ].gl_Position;
    vec4 p30 = gl_in[ 3 ].gl_Position;
    vec4 p01 = gl_in[ 4 ].gl_Position;
    vec4 p11 = gl_in[ 5 ].gl_Position;
    vec4 p21 = gl_in[ 6 ].gl_Position;
    vec4 p31 = gl_in[ 7 ].gl_Position;
    vec4 p02 = gl_in[ 8 ].gl_Position;
    vec4 p12 = gl_in[ 9 ].gl_Position;
    vec4 p22 = gl_in[ 10 ].gl_Position;
    vec4 p32 = gl_in[ 11 ].gl_Position;
    vec4 p03 = gl_in[ 12 ].gl_Position;
    vec4 p13 = gl_in[ 13 ].gl_Position;
    vec4 p23 = gl_in[ 14 ].gl_Position;
    vec4 p33 = gl_in[ 15 ].gl_Position;

    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;

    // Assigning the intermediate p_i's is here to make the code more
    // readable. From what I've seen, the compiler will optimize this away.
}
```



Computer Graphics

In the TES Shader – 32

```
// 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;

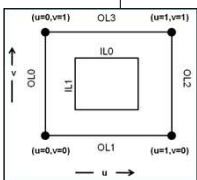
float dbu0 = -3. * (1.-u) * (1.-u);
float dbu1 = 3. * (1.-u) * (1.-3.*u);
float dbu2 = 3. * u * (2.-3.*u);
float dbu3 = 3. * u * u;

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 )
```



mp – December 18, 2024

In the TES Shader –
Computing the Normal, given a u and v 33

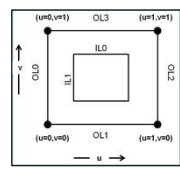
Tangent Vectors obtained by differentiating the position equation with respect to u and v:

```
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 );

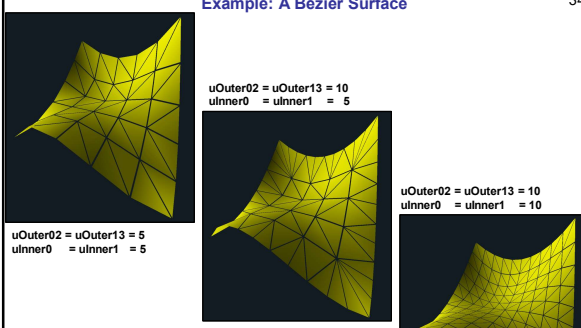
teNormal = normalize( cross( dpdu.xyz, dpdv.xyz ) );
```

Vector cross product to get the perpendicular normal to the two tangent vectors.



University of Oregon
Computer Graphics

Example: A Bézier Surface 34



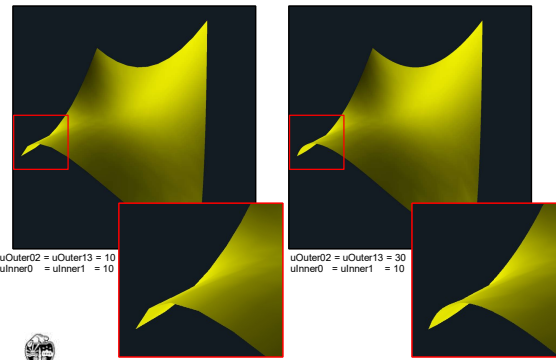
uOuter2 = uOuter13 = 5
ulnner0 = ulnner1 = 5

uOuter2 = uOuter13 = 10
ulnner0 = ulnner1 = 10

uOuter2 = uOuter13 = 10
ulnner0 = ulnner1 = 10

University of Oregon
Computer Graphics

Tessellation Levels and Smooth Shading 35



uOuter2 = uOuter13 = 10
ulnner0 = ulnner1 = 10

uOuter2 = uOuter13 = 30
ulnner0 = ulnner1 = 10

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

University of Oregon
Computer Graphics

Example: Whole-Sphere Subdivision 36

```
spheresubd.glsl

##OpenGL GLSL

Vertex    spheresubd.vert
Fragment  spheresubd.frag
TessControl spheresubd.tcs
TessEvaluation spheresubd.tes
Geometry  spheresubd.geom

Program SphereSubd
    uDetail <1 30 200>
    uScale <0.1 1. 10.>
    uShrink <0. 1. 1.>
    uFlat <false>
    uColor {1. 0. 0.}
    uLightX <-10. 5. 10.> uLightY <-10. 10. 10.> uLightZ <-10. 10. 10.>

Color 1. 1. 0.

NumPatchVertices 1

glBegin gl_patches
    glVertex 0. 0. 0. .2
    glVertex 0. 1. 0. .3
    glVertex 0. 0. 1. .4
glEnd
```

Using the x, y, z, and w to specify the center and radius of the sphere

University of Oregon
Computer Graphics

Example: Whole-Sphere Subdivision

37

```

spheresubd.vert
#version 400 compatibility
out vec3 vCenter;
out float vRadius;

void main()
{
    vCenter = gl_Vertex.xyz;
    vRadius = gl_Vertex.w;

    gl_Position = vec4( 0., 0., 0., 1. );
    // doesn't matter now - we will fill in the coords later
}

```

Using the x, y, z, and w to specify the center and radius of the sphere

Oregon State University
Computer Graphics

mp - December 18, 2024

Example: Whole-Sphere Subdivision

38

```

spheresubd.tcs
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

in float vRadius[ ];
in vec3 vCenter[ ];

patch out float tcRadius;
patch out vec3 tcCenter;

uniform float uDetail;
uniform float uScale;

layout( vertices = 1 ) out;

void main()
{
    gl_out[ gl_InvocationID ] gl_Position = gl_in[ 0 ].gl_Position; // (0.0,0.1)

    tcCenter = vCenter[ 0 ];
    tcRadius = vRadius[ 0 ];

    gl_TessLevelOuter[0] = 2.;
    gl_TessLevelOuter[1] = uScale * tcRadius * uDetail;
    gl_TessLevelOuter[2] = 2.;
    gl_TessLevelOuter[3] = uScale * tcRadius * uDetail;
    gl_TessLevelInner[0] = uScale * tcRadius * uDetail;
    gl_TessLevelInner[1] = uScale * tcRadius * uDetail;
}

```

Using the scale and the radius to help set the tessellation detail

Outer[0] and Outer[2] are the number of divisions at the poles. Outer[1] and Outer[3] are the number of divisions at the vertical seams. Inner[0] and Inner[1] are the inside sphere detail.

Oregon State University
Computer Graphics

mp - December 18, 2024

Example: Whole-Sphere Subdivision

39

```

spheresubd.tes
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

uniform float uScale;

layout( quads, equal_spacing, cw ) in;

patch in float tcRadius;
patch in vec3 tcCenter;

out vec3 teNormal;

const float PI = 3.14159265;

void main()
{
    vec3 p = gl_in[0].gl_Position.xyz;

    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;

    float phi = PI * (u - .5);
    float theta = 2. * PI * (v - .5);

    float cosphi = cos(phi);
    vec3 xyz = vec3( cosphi*cos(theta), sin(phi), cosphi*sin(theta) );
    teNormal = xyz;

    xyz *= (uScale * tcRadius);
    xyz += tcCenter;

    gl_Position = gl_ModelViewMatrix * vec4( xyz, 1. );
}

```

Turning u and v into spherical coordinates

$-\frac{\pi}{2} \leq \phi \leq +\frac{\pi}{2}$
 $-\pi \leq \theta \leq +\pi$

Oregon State University
Computer Graphics

mp - December 18, 2024

Example: Whole-Sphere Subdivision

40

Oregon State University
Computer Graphics

mp - December 18, 2024

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

41

```

spheresubd.tcs, I
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

in float vRadius[ ];
in vec3 vCenter[ ];

patch out float tcRadius;
patch out vec3 tcCenter;

uniform float uDetail;

layout( vertices = 1 ) out;

void main()
{
    gl_out[ gl_InvocationID ] gl_Position = gl_in[ 0 ].gl_Position; // (0.0,0.1)

    tcCenter = vCenter[ 0 ];
    tcRadius = vRadius[ 0 ];

    vec4 mx = vec4( vCenter[0] - vec3( vRadius[0], 0., 0. ), 1. );
    vec4 px = vec4( vCenter[0] + vec3( vRadius[0], 0., 0. ), 1. );
    vec4 my = vec4( vCenter[0] - vec3( 0., vRadius[0], 0. ), 1. );
    vec4 py = vec4( vCenter[0] + vec3( 0., vRadius[0], 0. ), 1. );
    vec4 mz = vec4( vCenter[0] - vec3( 0., 0., vRadius[0] ), 1. );
    vec4 pz = vec4( vCenter[0] + vec3( 0., 0., vRadius[0] ), 1. );
}

```

Extreme points of the sphere

Oregon State University
Computer Graphics

mp - December 18, 2024

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

42

```

spheresubd.tcs, II
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

in float vRadius[ ];
in vec3 vCenter[ ];

patch out float tcRadius;
patch out vec3 tcCenter;

uniform float uDetail;

layout( vertices = 1 ) out;

void main()
{
    gl_out[ gl_InvocationID ] gl_Position = gl_in[ 0 ].gl_Position; // (0.0,0.1)

    tcCenter = vCenter[ 0 ];
    tcRadius = vRadius[ 0 ];

    vec4 mx = vec4( vCenter[0] - vec3( vRadius[0], 0., 0. ), 1. );
    vec4 px = vec4( vCenter[0] + vec3( vRadius[0], 0., 0. ), 1. );
    vec4 my = vec4( vCenter[0] - vec3( 0., vRadius[0], 0. ), 1. );
    vec4 py = vec4( vCenter[0] + vec3( 0., vRadius[0], 0. ), 1. );
    vec4 mz = vec4( vCenter[0] - vec3( 0., 0., vRadius[0] ), 1. );
    vec4 pz = vec4( vCenter[0] + vec3( 0., 0., vRadius[0] ), 1. );

    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 );

    gl_TessLevelOuter[0] = 2.;
    gl_TessLevelOuter[1] = dmax * uDetail;
    gl_TessLevelOuter[2] = 2.;
    gl_TessLevelOuter[3] = dmax * uDetail;
    gl_TessLevelInner[0] = dmax * uDetail;
    gl_TessLevelInner[1] = dmax * uDetail;
}

```

Extreme points of the sphere in Clip space

Extreme points of the sphere in NDC space

How long are the lines between the extreme points?

We no longer use uScale 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.)

Oregon State University
Computer Graphics

mp - December 18, 2024

Making the Whole-Sphere Subdivision Adapt to Screen Coverage, III 43

```

spheradapt.tes
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

layout( quads, equal_spacing, ccw ) in;

patch in float  tcRadius;
patch in vec3  tcCenter;

out vec3      teNormal;

const float PI = 3.14159265;

void main()
{
    vec3 p = gl_in[0].gl_Position.xyz;

    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;
    float w = gl_TessCoord.z;

    float phi = PI * ( u - .5 );
    float theta = 2 * PI * ( v - .5 );

    float cosphi = cos(phi);
    vec3 xyz = vec3( cosphi*cos(theta), sin(phi), cosphi*sin(theta) );
    teNormal = xyz;

    xyz *= tcRadius;
    xyz += tcCenter;

    gl_Position = gl_ModelViewMatrix * vec4( xyz, 1. );
}

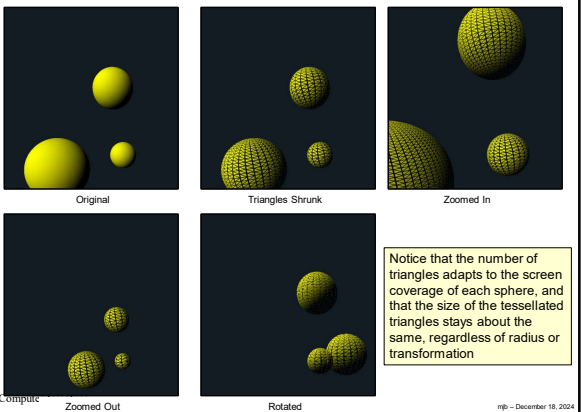
```

$-\frac{\pi}{2} \leq \phi \leq \frac{\pi}{2}$
 $-\pi \leq \theta \leq +\pi$

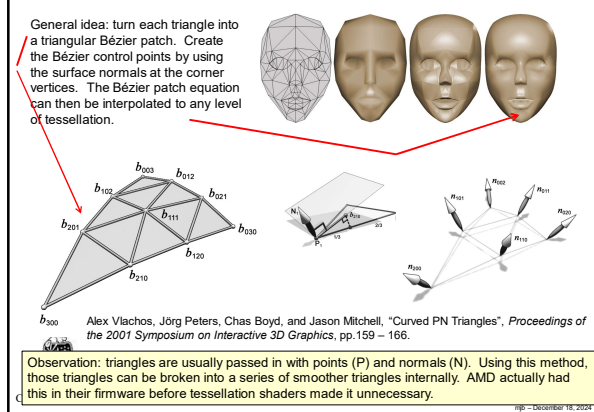
Spherical coordinates

No longer uses uScale

Making the Whole-Sphere Subdivision Adapt to Screen Coverage, IV 44



Example: PN Triangles 45



Example: PN Triangles 46

```

pntriangles.vert
#version 400 compatibility
uniform float uScale;
out vec3 vNormal;
void main()
{
    vec3 xyz = gl_Vertex.xyz;
    xyz *= uScale;
    gl_Position = gl_ModelViewMatrix * vec4( xyz, 1. );
    vNormal = normalize( gl_NormalMatrix * gl_Normal );
}

pntriangles.tcs
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable
uniform int uOuter, uInner;
uniform float uScale;
layout( vertices = 3 ) out;
in vec3 vNormal[ ];
out vec3 tcNormals[ ];

void main()
{
    tcNormals[ gl_InvocationID ] = vNormal[ gl_InvocationID ];
    gl_out[ gl_InvocationID ].gl_Position = gl_in[ gl_InvocationID ].gl_Position;

    gl_TessLevelOuter[0] = uScale * float(uOuter);
    gl_TessLevelOuter[1] = uScale * float(uOuter);
    gl_TessLevelOuter[2] = uScale * float(uOuter);
    gl_TessLevelInner[0] = uScale * float(uInner);
}

```

Example: PN Triangles 47

```

pntriangles.tes, I
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable
in vec3 tcNormals[ ];
out vec3 teNormal;
layout( triangles, equal_spacing, ccw ) in;

void main()
{
    vec3 p1 = gl_in[0].gl_Position.xyz;
    vec3 p2 = gl_in[1].gl_Position.xyz;
    vec3 p3 = gl_in[2].gl_Position.xyz;

    vec3 n1 = tcNormals[0];
    vec3 n2 = tcNormals[1];
    vec3 n3 = tcNormals[2];

    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;
    float w = gl_TessCoord.z;

    vec3 b300 = p1;
    vec3 b030 = p2;
    vec3 b003 = p3;

    float w12 = dot( p2 - p1, n1 );
    float w21 = dot( p1 - p2, n2 );
    float w13 = dot( p3 - p1, n1 );
    float w31 = dot( p1 - p3, n3 );
    float w23 = dot( p3 - p2, n2 );
    float w32 = dot( p2 - p3, n3 );
}

```

Example: PN Triangles 48

```

pntriangles.tes, II
vec3 b210 = ( 2*p1 + p2 - w12*n1 ) / 3.;
vec3 b120 = ( 2*p2 + p1 - w21*n2 ) / 3.;
vec3 b021 = ( 2*p2 + p3 - w32*n3 ) / 3.;
vec3 b012 = ( 2*p3 + p2 - w32*n3 ) / 3.;
vec3 b102 = ( 2*p3 + p1 - w31*n3 ) / 3.;
vec3 b201 = ( 2*p1 + p3 - w13*n1 ) / 3.;

vec3 ee = ( b210 + b120 + b021 + b012 + b102 + b201 ) / 6.;
vec3 vv = ( p1 + p2 + p3 ) / 3.;
vec3 b111 = ee + ( ee - vv ) / 2.;

vec3 xyz = 1.*b300*w*w*w + 1.*b030*u*u*u + 1.*b003*v*v*v +
3.*b210*u*w*w + 3.*b120*u*u*w + 3.*b201*v*w*w +
3.*b021*u*u*v + 3.*b102*v*v*w + 3.*b012*u*v*v +
6.*b111*u*v*w;

float v12 = 2. * dot( p2-p1, n1+n2 ) / dot( p2-p1, p2-p1 );
float v23 = 2. * dot( p3-p2, n2+n3 ) / dot( p3-p2, p3-p2 );
float v31 = 2. * dot( p1-p3, n3+n1 ) / dot( p1-p3, p1-p3 );

vec3 n200 = n1;
vec3 n020 = n2;
vec3 n002 = n3;
vec3 n110 = normalize( n1 + n2 - v12*(p2-p1) );
vec3 n011 = normalize( n2 + n3 - v23*(p3-p2) );
vec3 n101 = normalize( n3 + n1 - v31*(p1-p3) );
Normal = n200*w*w + n020*u*u + n002*v*v +
n110*w*w + n011*u*u + n101*v*v;

gl_Position = vec4( xyz, 1. );
}

```


Example: PN Triangles

49

pntriangles.geom

```
#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;
    V[2] = gl_PositionIn[2].xyz;
    CG = ( V[0] + V[1] + V[2] ) / 3.;

    ProduceVertex( 0 );
    ProduceVertex( 1 );
    ProduceVertex( 2 );
};
```

Oregon State University
Computer Graphics

18, 2024

Example: PN Triangles

50

pntriangles.frag

```
#version 400 compatibility
in float gLightIntensity;

const vec3 COLOR = vec3( 1., 1., 0. );

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

Oregon State University
Computer Graphics

18, 2024

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

51

uOuter = 1, ulnner = 1

uOuter = 2, ulnner = 1

uOuter = 2, ulnner = 2

uOuter = 2, ulnner = 2

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.

Oregon State University
Computer Graphics

18, 2024

The Difference Between Tessellation Shaders and Geometry Shaders

52

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.

Oregon State University
Computer Graphics

18, 2024

Demonstrating the Limits of Tessellation Shaders

53

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.

Oregon State University
Computer Graphics

18, 2024