Direct3D 11 Tutorial: Tessellation

Bill Bilodeau
ISV Relations Group
AMD
bill.bilodeau@amd.com

V1.0

Topics Covered in This Session

- Overview of Direct3D tessellation
- Design of Hull and Domain Shaders
  - Two simple examples
- How to write efficient tessellation code
  - An advanced example

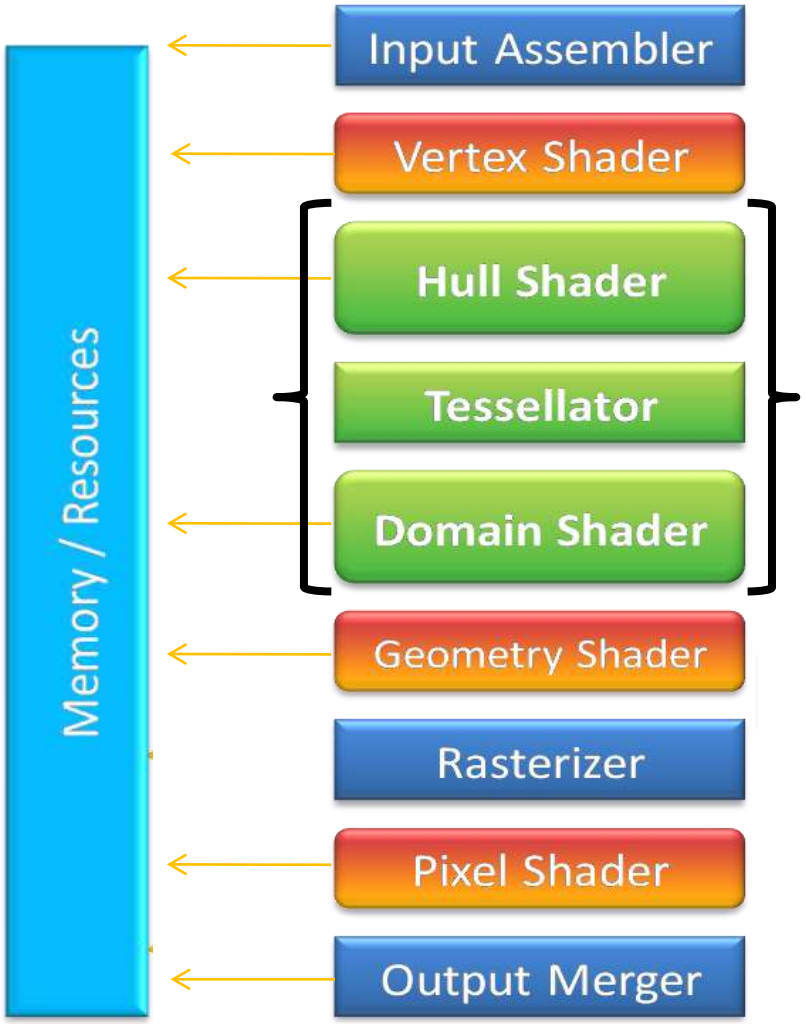
There are many really good reasons for using tessellation

- Compact representation
- Real-time, continuous level of detail
- Better looking silhouettes
- Direct rendering of high-order surfaces created by modeling software
- Faster – Animations can be done on the lower resolution mesh

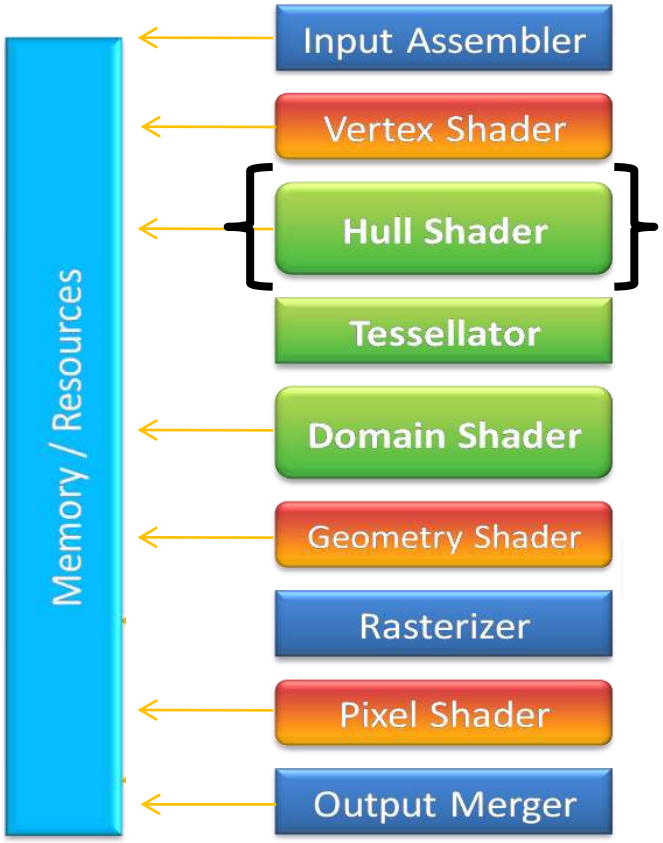
.. and one really bad reason for using it.

- Tessellating unnecessarily!
  - Triangles not seen
  - Triangles too small

Direct3D 11 Tessellator Stages



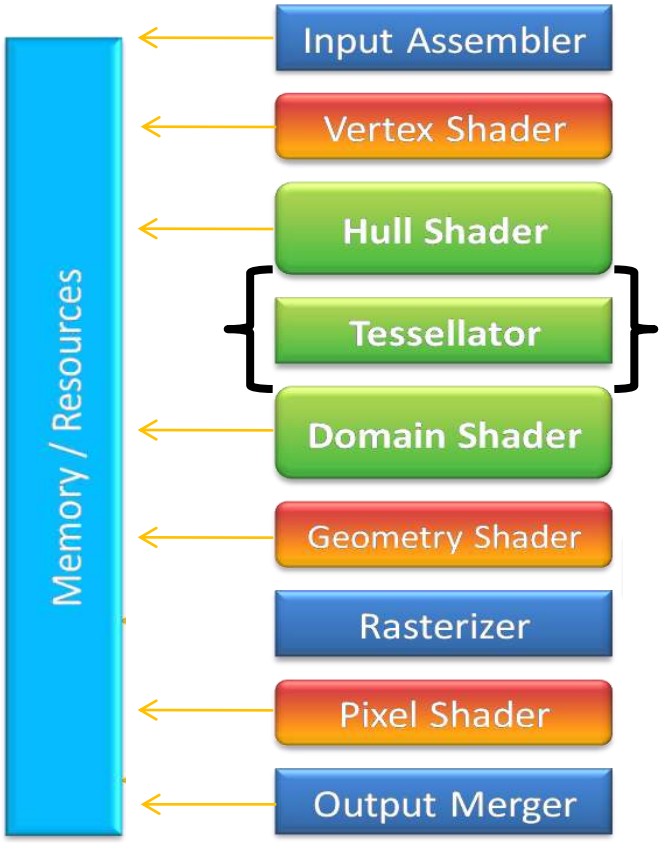
Direct3D 11 Tessellator Stages



Hull Shader

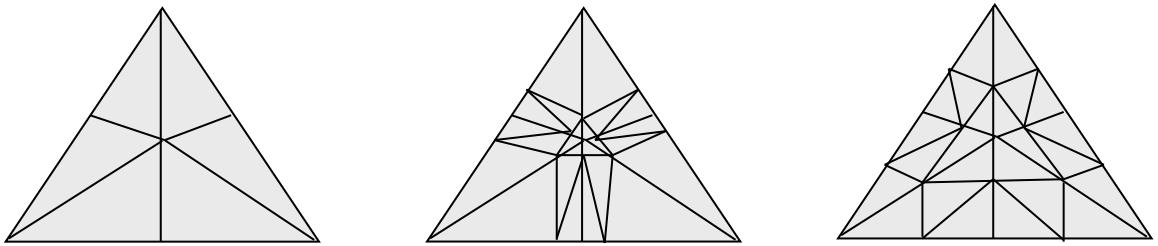
- Control point phase
  - Runs once per control point
- Patch constant phase
  - Runs once per input primitive
  - Returns tessellation factors

Direct3D 11 Tessellator Stages



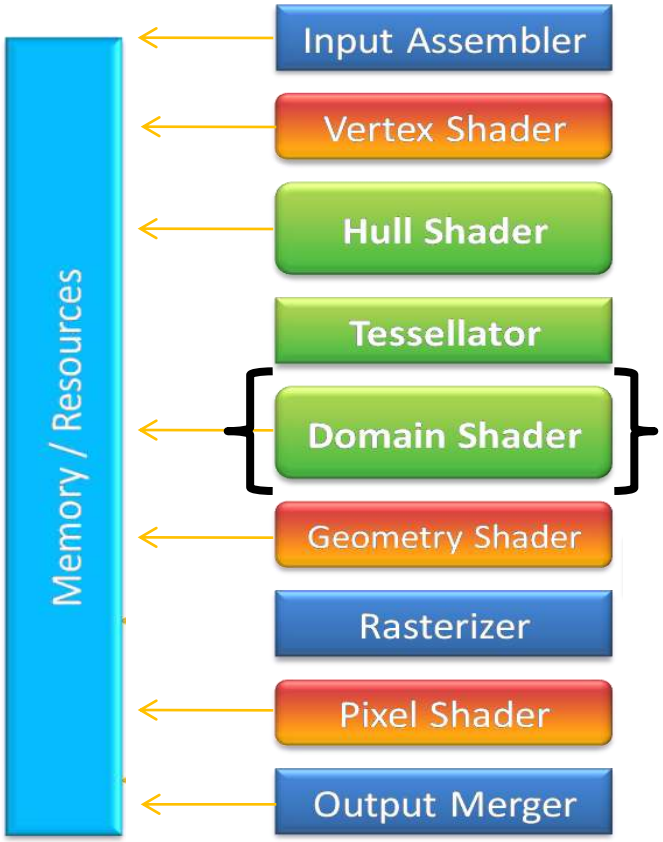
Tessellator Stage

- Generates the new vertices
- Higher tessellation factors mean more triangles generated.



Level 1.0 Level 1.5 Level 3.0

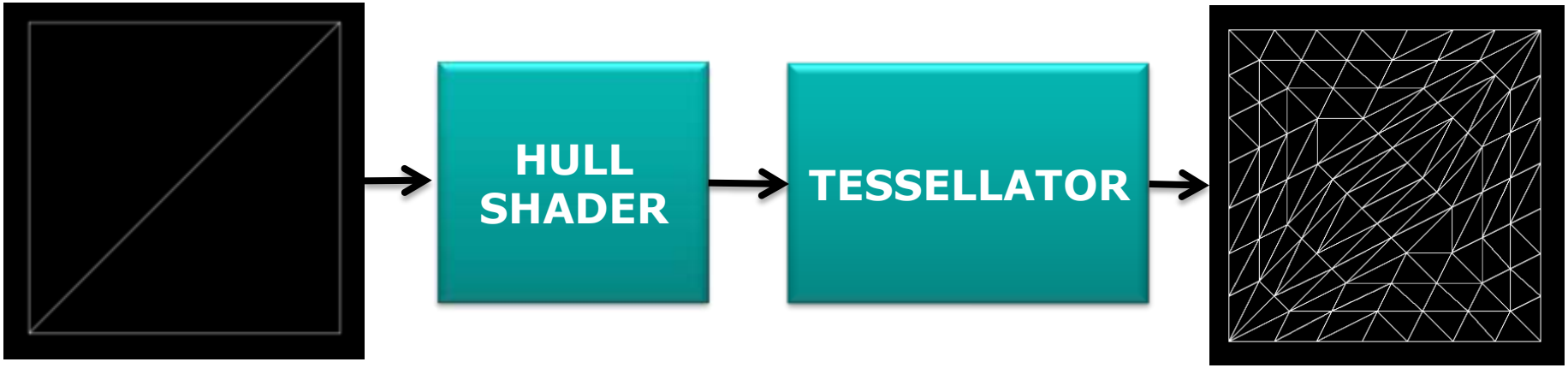
Direct3D 11 Tessellator Stages



Domain Shader

- Runs once per vertex for surface evaluation at each vertex
- Vertex data passed in as parametric coordinates

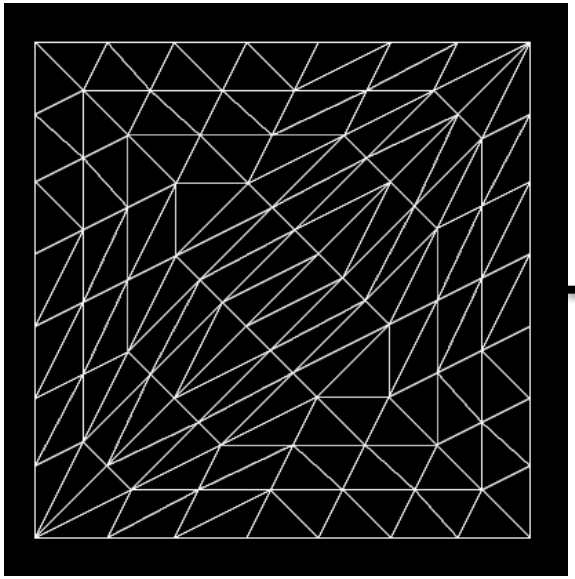
A simple example of tessellation (part 1)



Triangle Patch Mesh

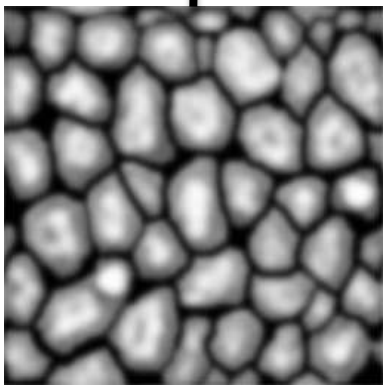
Tessellated Mesh

A simple example of tessellation (part 2)

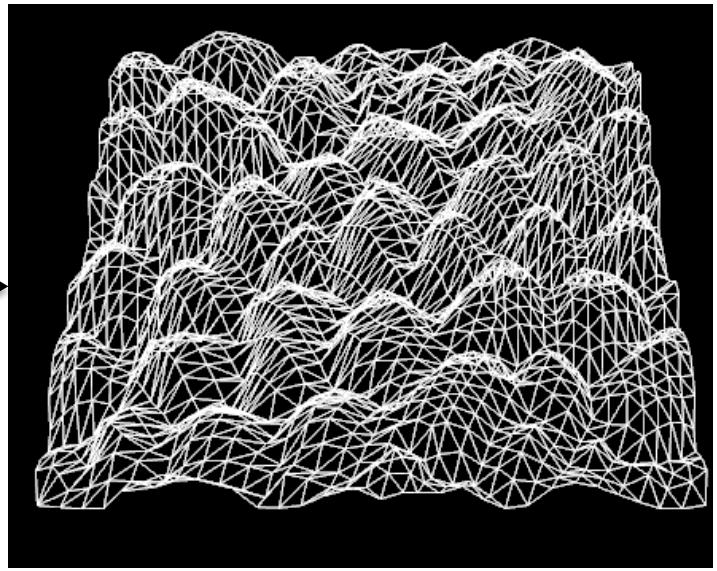


Tessellated Mesh

DOMAIN SHADER

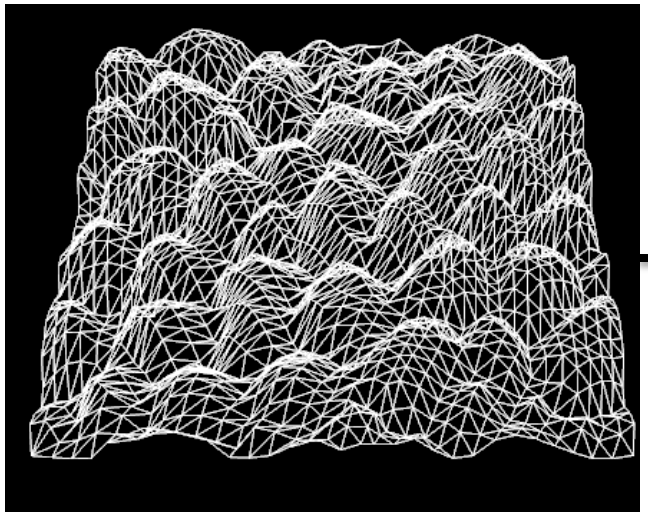


Displacement Map



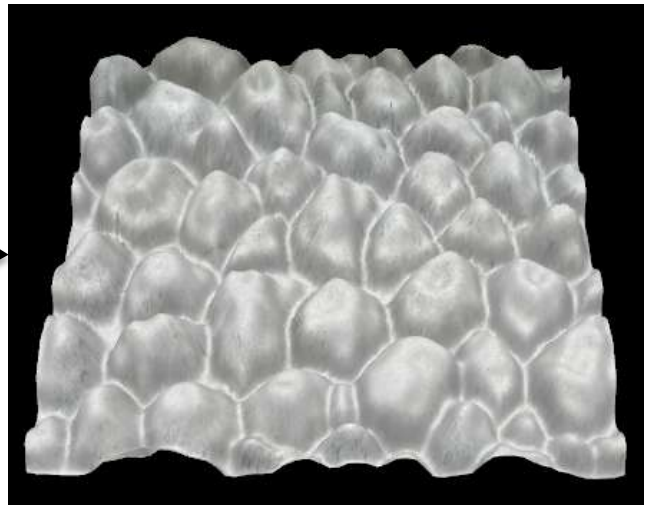
Displaced Mesh

A simple example of tessellation (part 3)



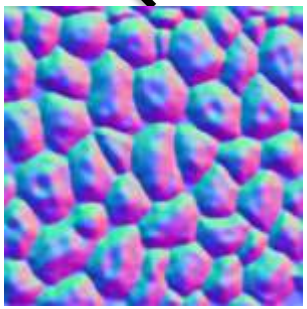
Displaced Mesh

PIXEL SHADER



Final Image

Diffuse Texture



Normal Map

C++ Code (part 1)

```
// set the shaders - note the addition of hull and domain shaders
pd3dImmediateContext->VSSetShader( g_pTessellationVS, NULL, 0 );
pd3dImmediateContext->HSSetShader( g_pTessellationHS, NULL, 0);
pd3dImmediateContext->DSSetShader( g_pTessellationDS, NULL, 0);
pd3dImmediateContext->GSSetShader( NULL, NULL, 0 );
Pd3dImmediateContext->PSSetShader( g_pSimplePS, NULL, 0 );
// set the vertex buffer which contains the triangle patches
UINT stride = sizeof(Vertex);
UINT offset = 0;
pd3dImmediateContext->IASetVertexBuffers( 0, 1, &g_pMeshVB, &stride,
   &offset );

// set input layout
pd3dImmediateContext->IASetInputLayout( g_pVertexLayout );

// set the primitive topology - note the topology is a control point
// patch list, which enables tessellated drawing
pd3dImmediateContext->IASetPrimitiveTopology(
               D3D11_PRIMITIVE_TOPOLOGY_3_CONTROL_POINT_PATCHLIST);
// Draw the model - same as drawing without tessellation
pd3dImmediateContext->Draw( g_SceneModel.triCount * 3, 0 );
```

Simple Hull Shader (control point phase)

```
// Called once per control point
[domain("tri")] // indicates a triangle patch (3 verts)
[partitioning("fractional_odd")] // fractional avoids popping
// vertex ordering for the output triangles
[outputtopology("triangle_cw")] [outputcontrolpoints(3)]
// name of the patch constant hull shader
[patchconstantfunc("ConstantsHS")]
[maxtessfactor(7.0)] //hint to the driver - the lower the better
// Pass in the input patch and an index for the control point
HS_CONTROL_POINT_OUTPUT HS( InputPatch<VS_OUTPUT_HS_INPUT, 3>
inputPatch, uint uCPID : SV_OutputControlPointID )
{
    HS_CONTROL_POINT_OUTPUT Out;

    // Copy inputs to outputs - "pass through" shaders are optimal
    Out.vWorldPos = inputPatch[uCPID].vPosWS.xyz;
    Out.vTexCoord = inputPatch[uCPID].vTexCoord;
    Out.vNormal = inputPatch[uCPID].vNormal;
    Out.vLightTS = inputPatch[uCPID].vLightTS;

    return Out;
}
```

Simple Hull Shader (patch constant phase)

```
//Called once per patch. The patch and an index to the patch (patch
// ID) are passed in
HS_CONSTANT_DATA_OUTPUT ConstantsHS( InputPatch<VS_OUTPUT_HS_INPUT, 3>
   p, uint PatchID : SV_PrimitiveID )
{
    HS_CONSTANT_DATA_OUTPUT Out;

   // Assign tessellation factors - in this case use a global
   // tessellation factor for all edges and the inside. These are
   // constant for the whole mesh.
   Out.Edges[0] = g_TessellationFactor;
   Out.Edges[1] = g_TessellationFactor;
   Out.Edges[2] = g_TessellationFactor;
   Out.Inside   = g_TessellationFactor;
   return Out;
}
```

Simple Domain Shader (part 1)

```
// Called once per tessellated vertex
[domain("tri")] // indicates that triangle patches were used
// The original patch is passed in, along with the vertex position in barycentric
coordinates, and the patch constant phase hull shader output (tessellation factors)
DS_VS_OUTPUT_PS_INPUT DS( HS_CONSTANT_DATA_OUTPUT input,
              float3 BarycentricCoordinates : SV_DomainLocation,
              const OutputPatch<HS_CONTROL_POINT_OUTPUT, 3>
              TrianglePatch )
{
    DS_VS_OUTPUT_PS_INPUT Out;
    // Interpolate world space position with barycentric coordinates
    float3 vWorldPos =
        BarycentricCoordinates.x * TrianglePatch[0].vWorldPos +
        BarycentricCoordinates.y * TrianglePatch[1].vWorldPos +
        BarycentricCoordinates.z * TrianglePatch[2].vWorldPos;
    // Interpolate texture coordinates with barycentric coordinates
    Out.vTexCoord =
       BarycentricCoordinates.x * TrianglePatch[0].vTexCoord + ...
    // Interpolate normal with barycentric coordinates
    float3 vNormal =
       BarycentricCoordinates.x * TrianglePatch[0].vNormal + ...
    // Interpolate light vector with barycentric coordinates
    Out.vLightTS =
       BarycentricCoordinates.x * TrianglePatch[0].vLightTS +
```

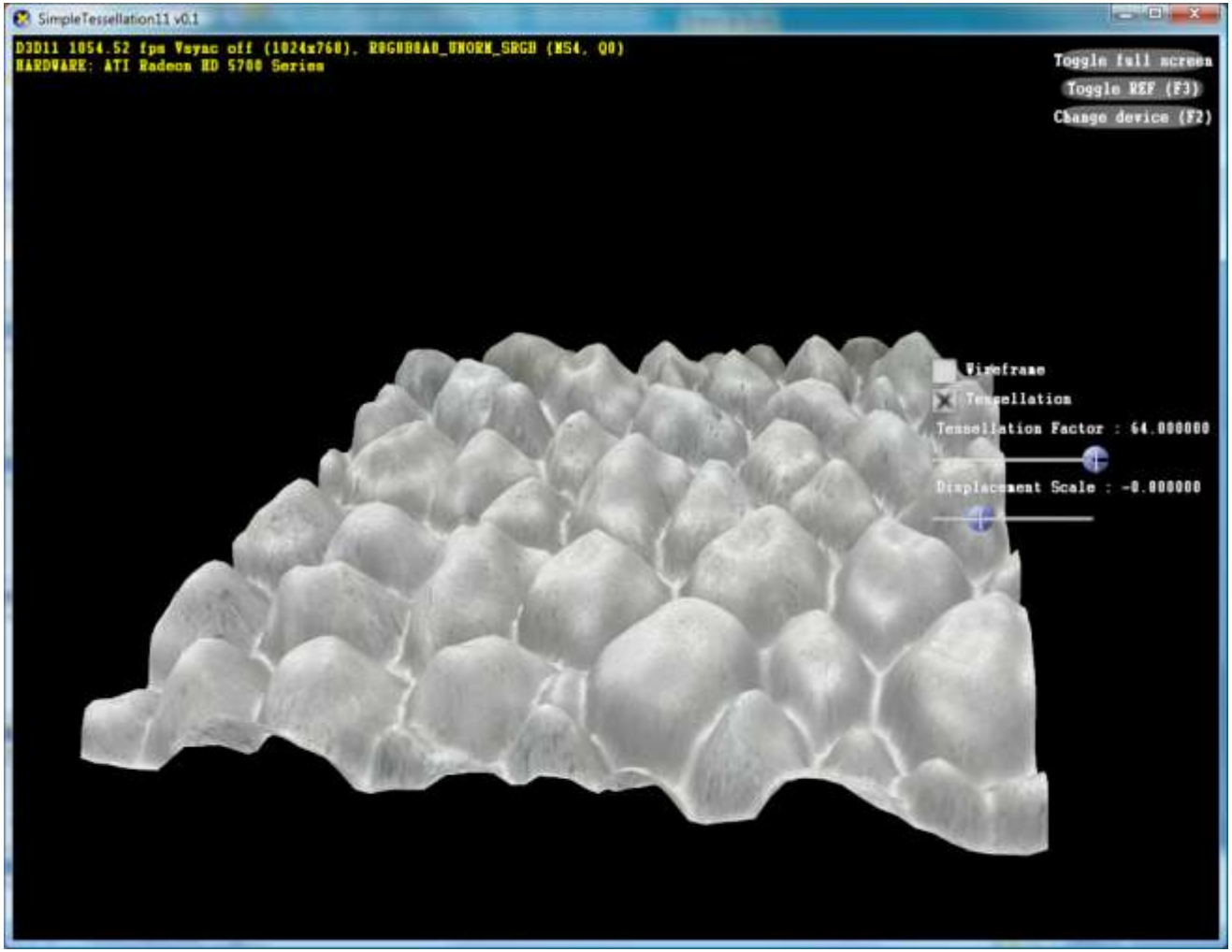
Simple Domain Shader (part 2)

```
    // sample the displacement map for the magnitude of displacement
    float fDisplacement = g_DisplacementMap.SampleLevel(
                                g_sampleLinear, Out.vTexCoord.xy, 0 ).r;
    fDisplacement *= g_Scale;
    fDisplacement += g_Bias;
    float3 vDirection = -vNormal; // direction is opposite normal

    // translate the position
    vWorldPos += vDirection * fDisplacement;

    // transform to clip space
    Out.vPosCS = mul( float4( vWorldPos.xyz, 1.0 ),
                                g_mWorldViewProjection );

    return Out;
} // end of domain shader
```

Simple Pixel Shader

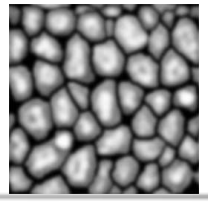
```
float4 PS( DS_VS_OUTPUT_PS_INPUT i ) : SV_TARGET
{
    float3 vLight;
    float3 vNormal;

    // Get the normal
    vNormal = normalize( (g_NormalMap.Sample(
                        g_sampleLinear, i.vTexCoord ).rgb) * 2 - 1 );
    vLight = normalize(i.vLightTS);

    // get base color
    float4 vBaseColor = float4(g_BaseMap.Sample( g_sampleLinear,
                        i.vTexCoord).rgb, 1.0);

    // light the pixel
    float diffuse = saturate( dot( vNormal, vLight ) );
    float4 color = (vBaseColor * diffuse) + (vBaseColor *
                        g_vAmbientColor);
    return color;
}
```

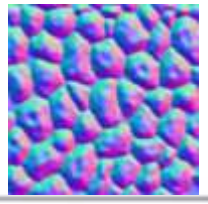
Demo: Simple Tessellation



Some things to remember about this example

- Simple but powerful
  - A few hours of coding but can be used for something as useful as terrain rendering
- Minimal work in the Hull Shader
  - Driver recognizes the "Pass Through" shader and optimizes for it
- Normal Map Lighting
  - Lighting a displaced surface can be difficult because the normals change!
  - Normal map lighting avoids this since normal map is paired with the displacement map:

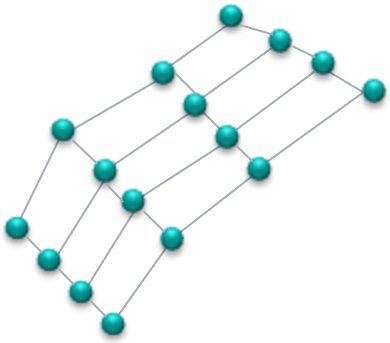




Parametric Surfaces

- Hardware tessellation can be used to render 3D parametric surfaces directly
- Cubic Bezier surfaces are easy to render
  - Can convert many types of surfaces to Bezier patches in the Hull Shader
- Domain Shader evaluates the surface at the vertex

Bicubic Bezier Patch Example

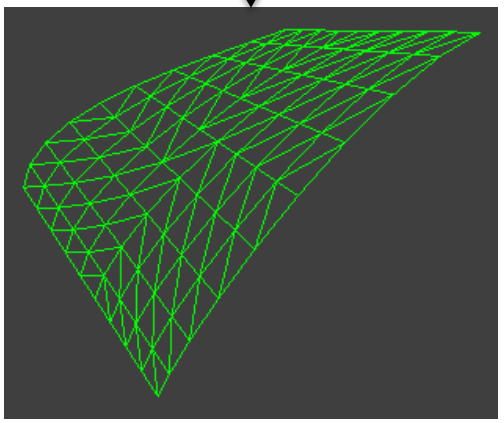


Bezier Control Points

$$Pos(u,v) = \sum_{m=0}^{3} \sum_{n=0}^{3} B_m(u)\, B_n(v)\, G$$

Where: $B_0 = (1-t)^3$
$B_1 = 3t(1-t)^2$
$B_2 = 3t^2(1-t)$
$B_3 = t^3$

and $G = \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$



Tessellated Bezier Patch

Bezier Patch C++ Code

```
// Layout for control points
const D3D11_INPUT_ELEMENT_DESC patchlayout[] =
 {
  {"POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0,
                                          D3D11_INPUT_PER_VERTEX_DATA, 0 },
 };

// ...

// Control points are stored in the vertex buffer
pd3dImmediateContext->IASetVertexBuffers(0,1,&g_pControlPointVB,
                                                 &Stride, &Offset );
// Primitive Topology - note 16 control points this time
pd3dImmediateContext->IASetPrimitiveTopology(
              D3D11_PRIMITIVE_TOPOLOGY_16_CONTROL_POINT_PATCHLIST );

 // Draw the mesh
 pd3dImmediateContext->Draw( g_vertexCount, 0 );
```

Bezier Patch Domain Shader Helper functions (part 1)

```
//Evaluates the Bernstein basis for a given parameter t
float4 BernsteinBasis(float t)
{
    float invT = 1.0f - t;

    return float4( invT * invT * invT,          // (1-t)^3
                   3.0f * t * invT * invT,      // 3t(1-t)^2
                   3.0f * t * t * invT,         // 3t^2(1-t)
                   t * t * t );                 // t^3
}

// Derivative of the basis functions for computing tangents
float4 dBernsteinBasis(float t)
{
    float invT = 1.0f - t;

    return float4( -3 * invT * invT,                    // -3(1-t)^2
                   3 * invT * invT - 6 * t * invT,      // 3(1-t)-6t(1-t)
                   6 * t * invT - 3 * t * t,            // 6t(1-t) - 3t^2
                   3 * t * t );                         // 3t^2
}
```

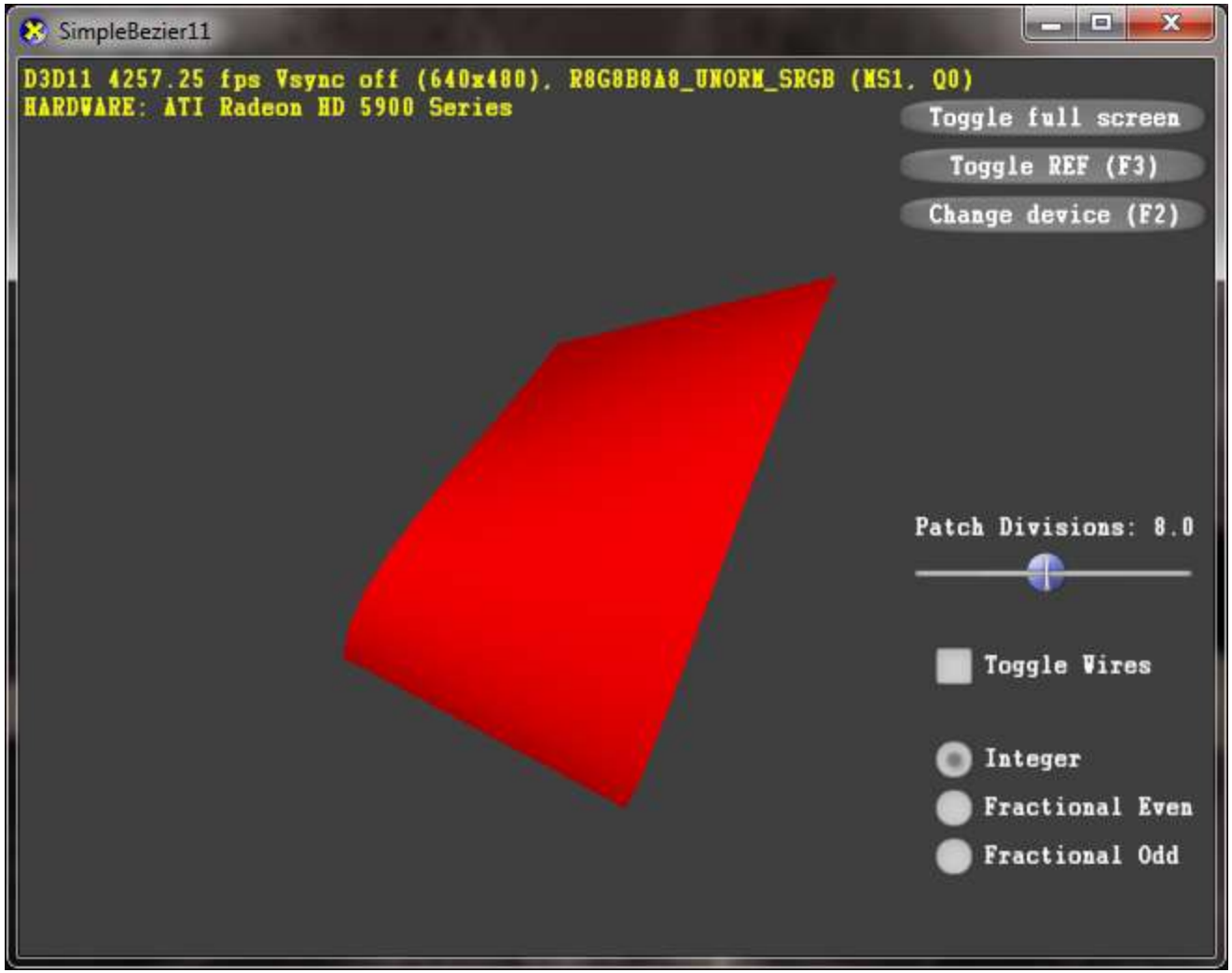
Bezier Patch Domain Shader Helper functions (part 2)

```
float3 EvaluateBezier( const OutputPatch<HS_OUTPUT, OUTPUT_PATCH_SIZE>
                       bezpatch, float4 BasisU, float4 BasisV )
{
  // This function essentially does this: Value(u,v) = Σ Σ Bm(u) Bn(v) G

  float3 Value = float3(0,0,0);
  Value  = BasisV.x * ( bezpatch[0].vPosition * BasisU.x +
           bezpatch[1].vPosition * BasisU.y + bezpatch[2].vPosition *
           BasisU.z + bezpatch[3].vPosition * BasisU.w );
  Value += BasisV.y * ( bezpatch[4].vPosition * BasisU.x +
           bezpatch[5].vPosition * BasisU.y + bezpatch[6].vPosition *
           BasisU.z + bezpatch[7].vPosition * BasisU.w );
  Value += BasisV.z * ( bezpatch[8].vPosition * BasisU.x +
           bezpatch[9].vPosition * BasisU.y + bezpatch[10].vPosition *
           BasisU.z + bezpatch[11].vPosition * BasisU.w );
  Value += BasisV.w * ( bezpatch[12].vPosition * BasisU.x +
           bezpatch[13].vPosition * BasisU.y + bezpatch[14].vPosition *
           BasisU.z + bezpatch[15].vPosition * BasisU.w );

    return Value;
}
```

Bezier Patch Domain Shader

```
[domain("quad")] // indicates that the patches are quads
DS_OUTPUT BezierDS( HS_CONSTANT_DATA_OUTPUT input,
// Since this is a quad, tessellated vertex positions are given with
// two (u,v) parameters
           float2 UV : SV_DomainLocation,
           const OutputPatch<HS_OUTPUT, OUTPUT_PATCH_SIZE> bezpatch )
{    // Evaluate the basis functions at (u, v)
     float4 BasisU = BernsteinBasis( UV.x );
     float4 BasisV = BernsteinBasis( UV.y );
     float4 dBasisU = dBernsteinBasis( UV.x );
     float4 dBasisV = dBernsteinBasis( UV.y );
     // Evaluate the surface position for this vertex
     float3 WorldPos = EvaluateBezier( bezpatch, BasisU, BasisV );
     // Evaluate the tangent space for this vertex (using derivatives)
     float3 Tangent = EvaluateBezier( bezpatch, dBasisU, BasisV );
     float3 BiTangent = EvaluateBezier( bezpatch, BasisU, dBasisV );
     float3 Norm = normalize( cross( Tangent, BiTangent ) );
     DS_OUTPUT Output;
     Output.vPosition = mul( float4(WorldPos,1), g_mViewProjection );
     Output.vWorldPos = WorldPos;
     Output.vNormal = Norm;
     return Output;
}
```

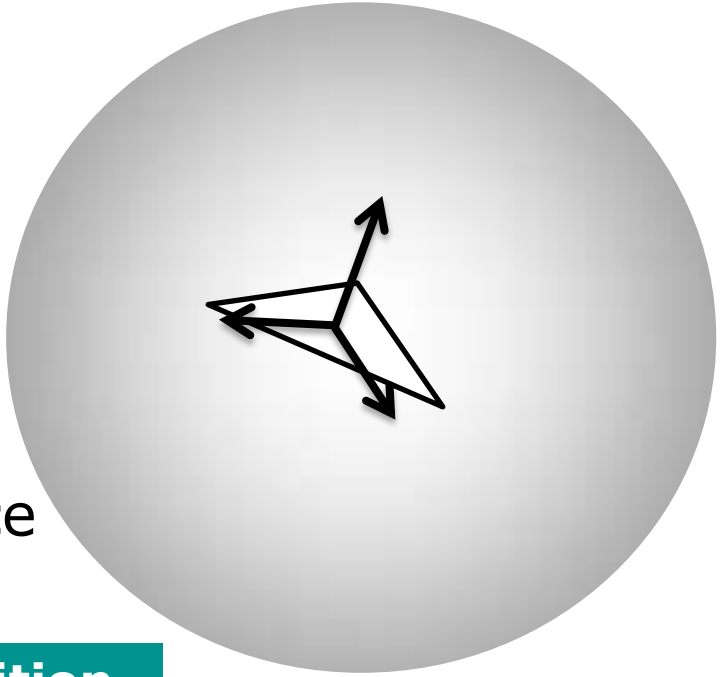
Bezier Patch Demo



Advanced Tessellation Example: Decal Tessellation

- Apply a displacement decal to any mesh, then draw with tessellation to accurately render the displacement
- Tessellates arbitrary scene meshes without modifications to the art pipeline
- Use the hull shader to control tessellation levels and improve performance

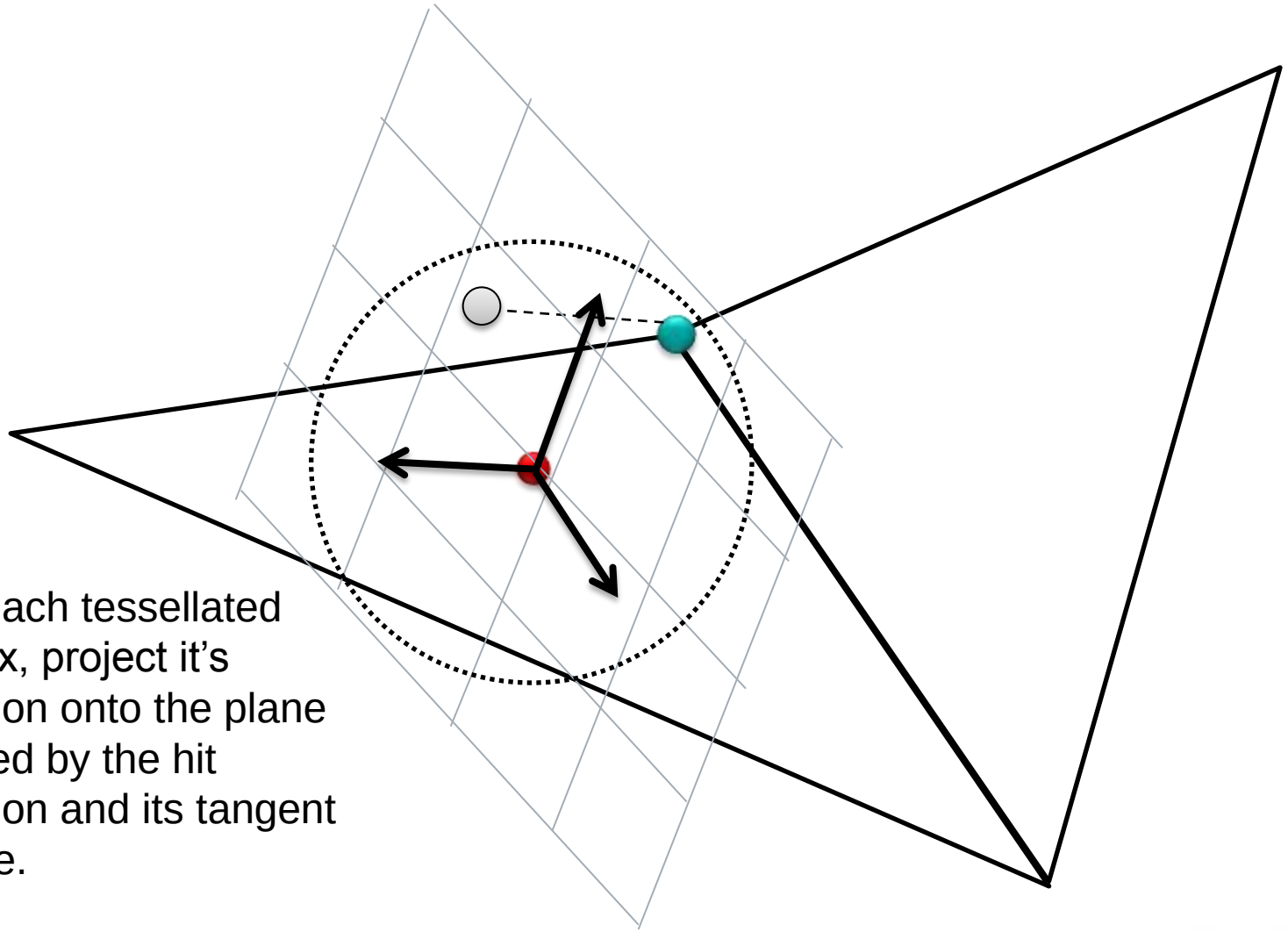
Ray cast to find the hit location



Store hit location tangent space into a constant buffer

Normal	Binormal	Tangent	Position
N_0	B_0	T_0	P_0
N_1	B_1	T_1	P_1
N_2	B_2	T_2	P_2

Compute the displacement map coordinates in the Domain Shader



For each tessellated vertex, project it's position onto the plane formed by the hit location and its tangent space.

Decal Tessellation Domain Shader (part 1)

```
// See if this vertex is affected by any of the decals
for (int i = 0; i < MAX_DECALS; i++)
{
  // get the hit location
  float3 vHitLocation = g_HitLocation[i];

  // find the distance from the current vertex to the hit location
  float distanceToHit = distance(vWorldPos, vHitLocation.xyz);

  // check if the vertex is within the decal radius
  if (distanceToHit <= decalRadius)
  {
     // translate the position to a coordinate space
     // with the hit location as the origin
     float3 vWorldPosTrans = vWorldPos - vHitLocation.xyz;

     // create the decal tangent space matrix
     float3x3 mWorldToTangent = float3x3( g_vTangent[i].xyz,
                                          g_vBinormal[i].xyz, g_vNormal[i].xyz );
```

Decal Tessellation Domain Shader (part 2)

```
// Transform the position into decal tangent space to
// get the displacement map texture coordinate.
float3 vWorldPosTrans = vWorldPos - vHitLocation.xyz;
float3 vDMTexCoord = mul( mWorldToTangent,vWorldPosTrans);

// normalize coordinate to values between 0 and 1
vDMTexCoord /= decalRadius* 2;
vDMTexCoord += 0.5;

// project displacement map coordinate onto the x,y plane
vDMTexCoord.z = 1; // z = 0 tells pixel shader this is invalid

// sample the displacement map
float fDisplacement = g_DisplacementMap.SampleLevel(
                        g_sampleLinear, vDMTexCoord.xy, 0 ).r;
fDisplacement *= g_Scale;
fDisplacement += g_Bias;

// hit direction is opposite of tangent space normal
float3 vDirection = -g_vNormal[i].xyz;
```

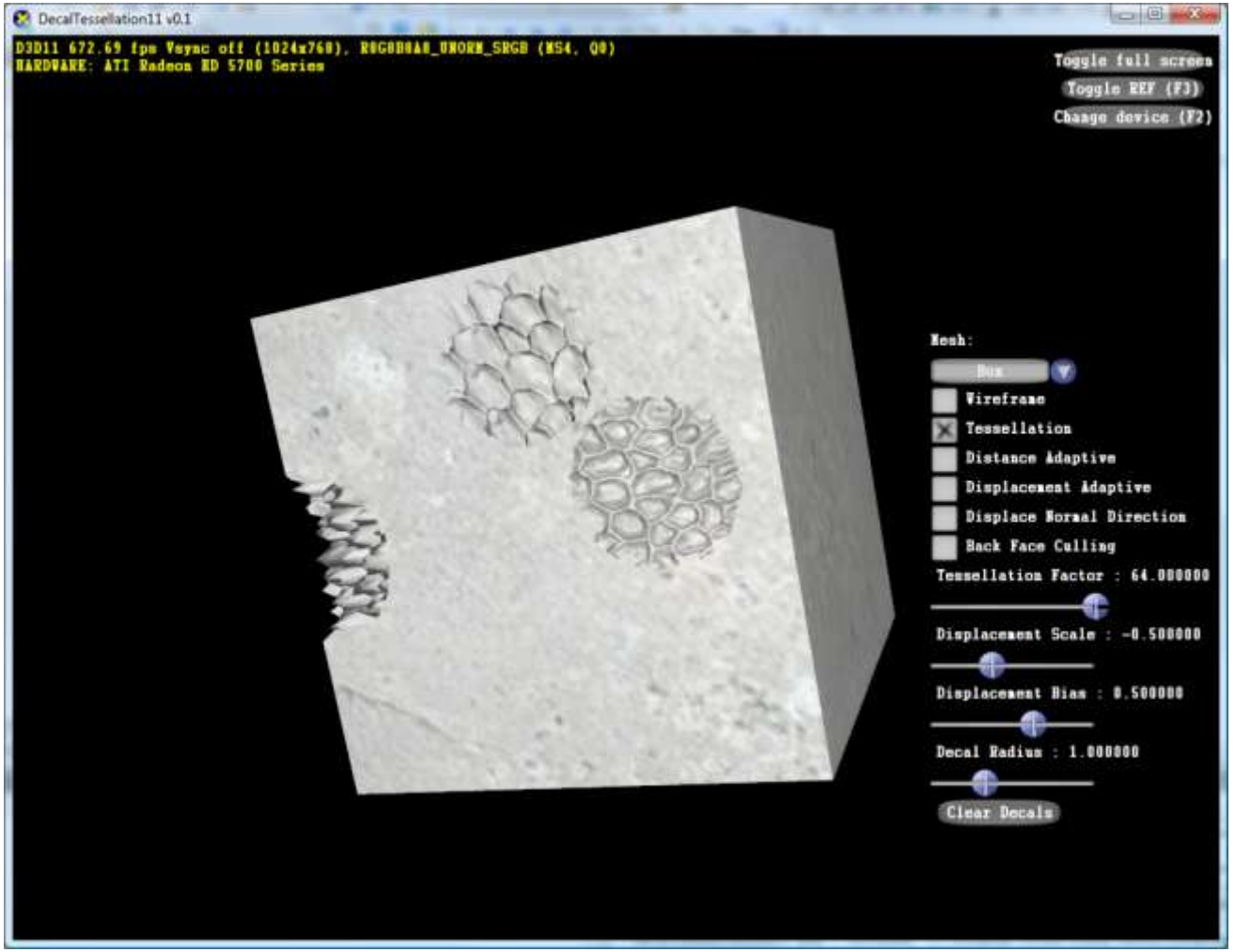
Decal Tessellation Domain Shader (part 3)

```
        // Displace the vertex
        vWorldPos += vDirection * fDisplacement;
        // Create the light vector
        float3 vLightWS = g_vLightPosition.xyz - vWorldPos;
        // transform the light vector into tangent space
        Out.vLightTS = mul( mWorldToTangent, vLightWS );
        // Use the displacement map coord for the normal map coord
        Out.vNMTexCoord = vDMTexCoord;
        break;
      } // end of "if (distanceToHit <= decalRadius)"
    } // end of "for (int i = 0; i < MAX_DECALS; i++)"

    Out.vPosCS = mul( float4( vWorldPos.xyz, 1.0 ),
                      g_mWorldViewProjection );
    return Out;
} // end of domain shader
```

Decal Tessellation Pixel Shader

```
// Get the Normal and Light vectors for lighting

// If vNMTexCoord.z = 1, then this pixel is part of the decal
if (i.vNMTexCoord.z < 1)
{
     // Not part of the decal, just use the interpolated normal.
     vNormal = normalize(i.vNormal);
     vLight = normalize(i.vLightWS);
}
else
{
     // Part of the decal, get the normal from the decal normal map.
     vNormal = normalize( (g_NormalMap.Sample( g_sampleLinear,
                    i.vNMTexCoord ).rgb) * 2 - 1 );
     vLight = normalize(i.vLightTS);
}
...
```

Demo: Decal Tessellation



Making it faster

- Analysis
  - Expensive Domain Shader
  - Small triangles are inefficient
    - Hardware wants triangles to be at least 8 pixels
- Solution
  - Reduce the number of tessellated vertices
    - Fewer executions of the Domain Shader
  - Hull Shader can adjust the tessellation factors

Distance Adaptive Tessellation

- Adjust the tessellation factors globally for the object based on distance

- Simple LERP done outside of shader code:

  TF = (1 – t) maxTF + t

  where: t = distance / maxTessellationDistance

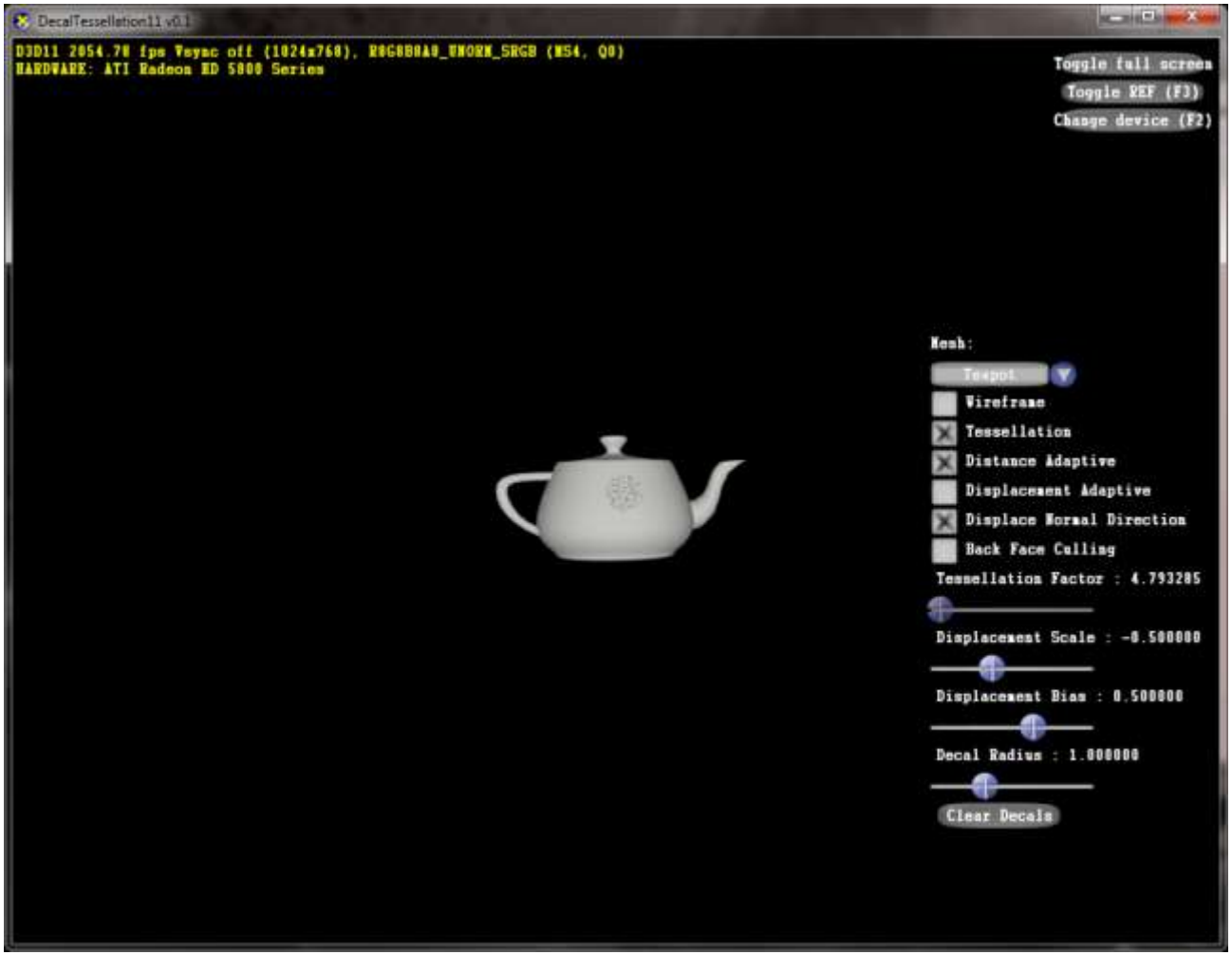
- Or, use Hull Shader to calculate screen space edge length

- Use an impostor when far away (just normal map)

Distance Adaptive Tessellation Demo (1)

Distance adaptive up close: 33 FPS

Distance Adaptive Tessellation Demo (2)



Distance adaptive far away: 2054 FPS!

Distance Adaptive Results

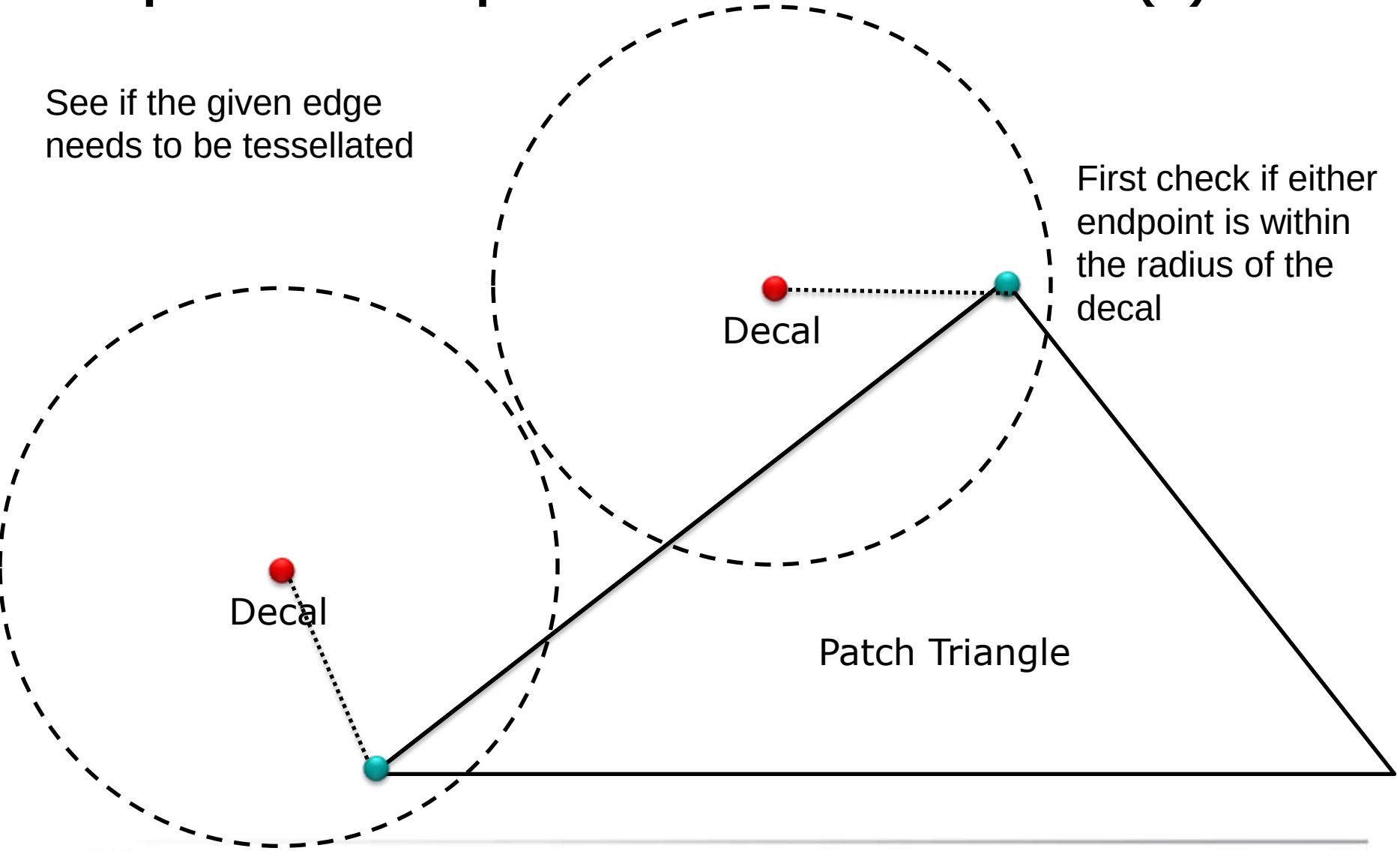
- Two orders of magnitude improvement for when object is far away!
- Doesn’t help much for objects close up
- Very little code changes

Displacement Adaptive

- Use the displacement map to determine how much to tessellate
- High frequency of displacements (jagged) require more tessellation
- Low Frequency of displacements (smooth) require fewer tessellation

Displacement Adaptive with Decal Tessellation (1)

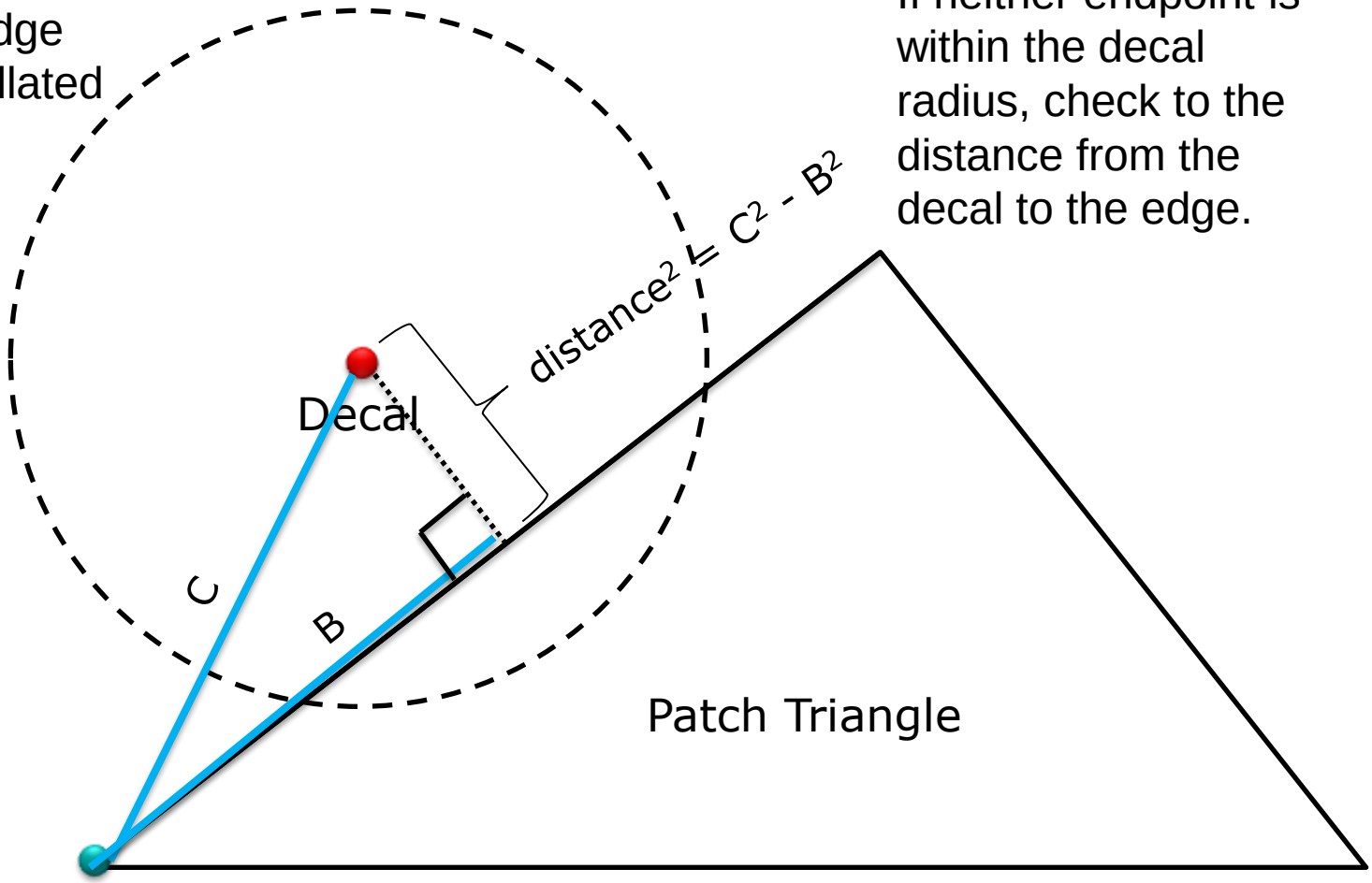
See if the given edge needs to be tessellated



Displacement Adaptive with Decal Tessellation (2)

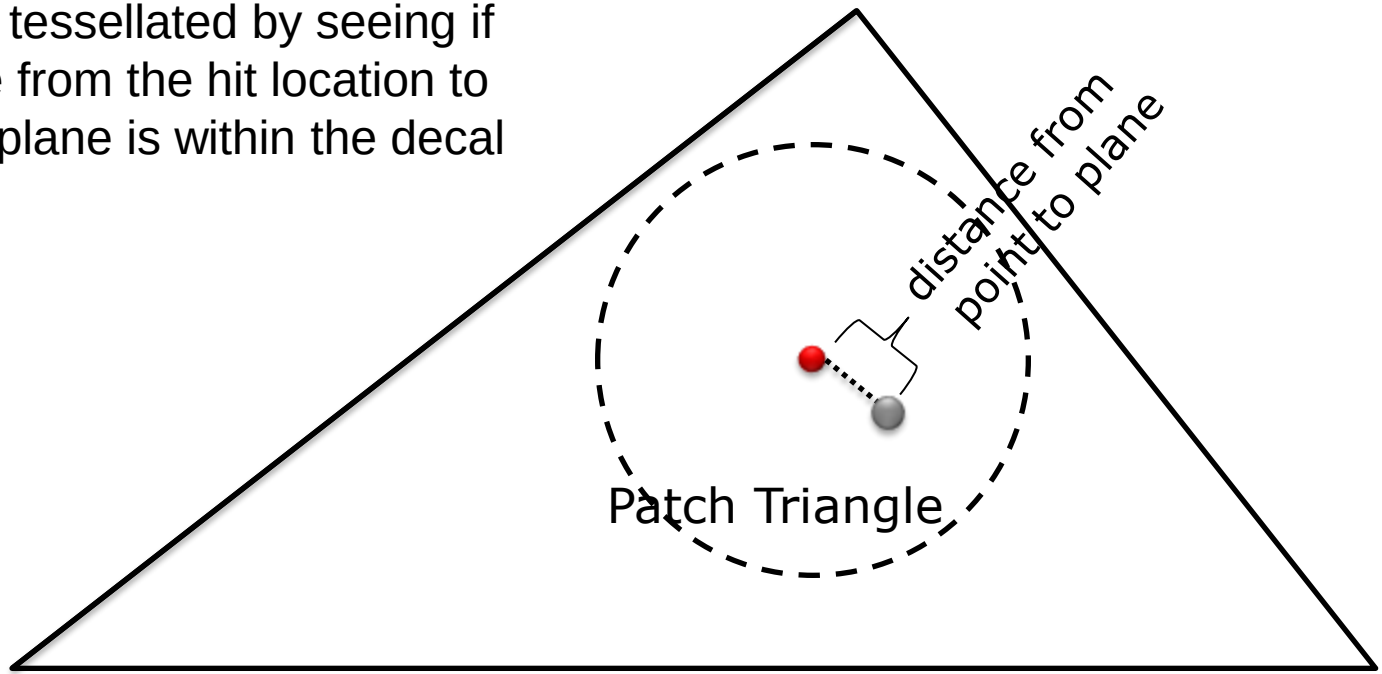
See if the given edge needs to be tessellated

If neither endpoint is within the decal radius, check to the distance from the decal to the edge.



Displacement Adaptive with Decal Tessellation (3)

See if the *inside* of the triangle needs to be tessellated by seeing if the distance from the hit location to the triangle plane is within the decal radius.



Displacement Adaptive Hull Shader Pseudo Code

```
for (each decal in the list)
{
    if ( either vertex of Edge 0 is within the decal's radius)
    {
                tessellate this edge;
    }
    else
    {
        if ( (distance from the decal to Edge 0 is within the radius)
                                        && (decal intersects the line segment) )
        {
                tessellate this edge;
        }
    }

    Do the same for Edge 1 and Edge 2 …

    if ( any edge is tessellated )
        || ( ( distance from decal to triangle plane is within radius)
            && ( decal projected onto triangle plane is within triangle) )
    {
        tessellate the inside
    }
}
```

Displacement Adaptive Hull Shader (part 1)

```
 // default tessellation factors
Out.Edges[0] = Out.Edges[1] = Out.Edges[2] = Out.Inside = 1;

// Iterate over all decals to see if tessellation is needed
for (int i = 0; i < MAX_DECALS; i++)
{
     edgeTessellated = false;
     float3 vDecalLocation = g_DecalLocation[i];
     // Edge 0
     vDecalToPos0 = vDecalLocation - vPos0;
     magSquared0 = dot( vDecalToPos0 , vDecalToPos0 );
     vDecalToPos1 = vDecalLocation - vPos1;
     magSquared1 = dot( vDecalToPos1, vDecalToPos1 );
     // See if the distance to either vertex < decal radius
     if ((magSquared0 <= decalRadiusSquared) ||
                    (magSquared1 <= decalRadiusSquared))
     {
            Out.Edges[0] = g_TessellationFactor;
            edgeTessellated = true;
     }
```

Displacement Adaptive Hull Shader (part 2)

```
else
{
  // project line onto triangle patch edge
  vEdge0 = vPos1 – vPos0;
  magSquaredEdge0 = dot(vEdge0, vEdge0);
   vProjected = (dot(vDecalPos0,vEdge0)/magSquaredEdge0) * vEdge0;
  magSquaredProj = dot(vProjected, vProjected);

  // Use the Pythagorean theorem to find the squared distance.
  distanceSquared = magSquared0 - magSquaredProj;

  if ((distanceSquared <= decalRadiusSquared) &&  // within radius
    (dot(vProjected,vEdge0) >= 0) && // within edge line segment
    (magSquaredProj <= magSquaredEdge0))
        {
                Out.Edges[0] = g_tessellationFactor;
                edgeTessellated = true;
        }
}
// Do the same thing for Edge 1 and Edge 2 ...
```

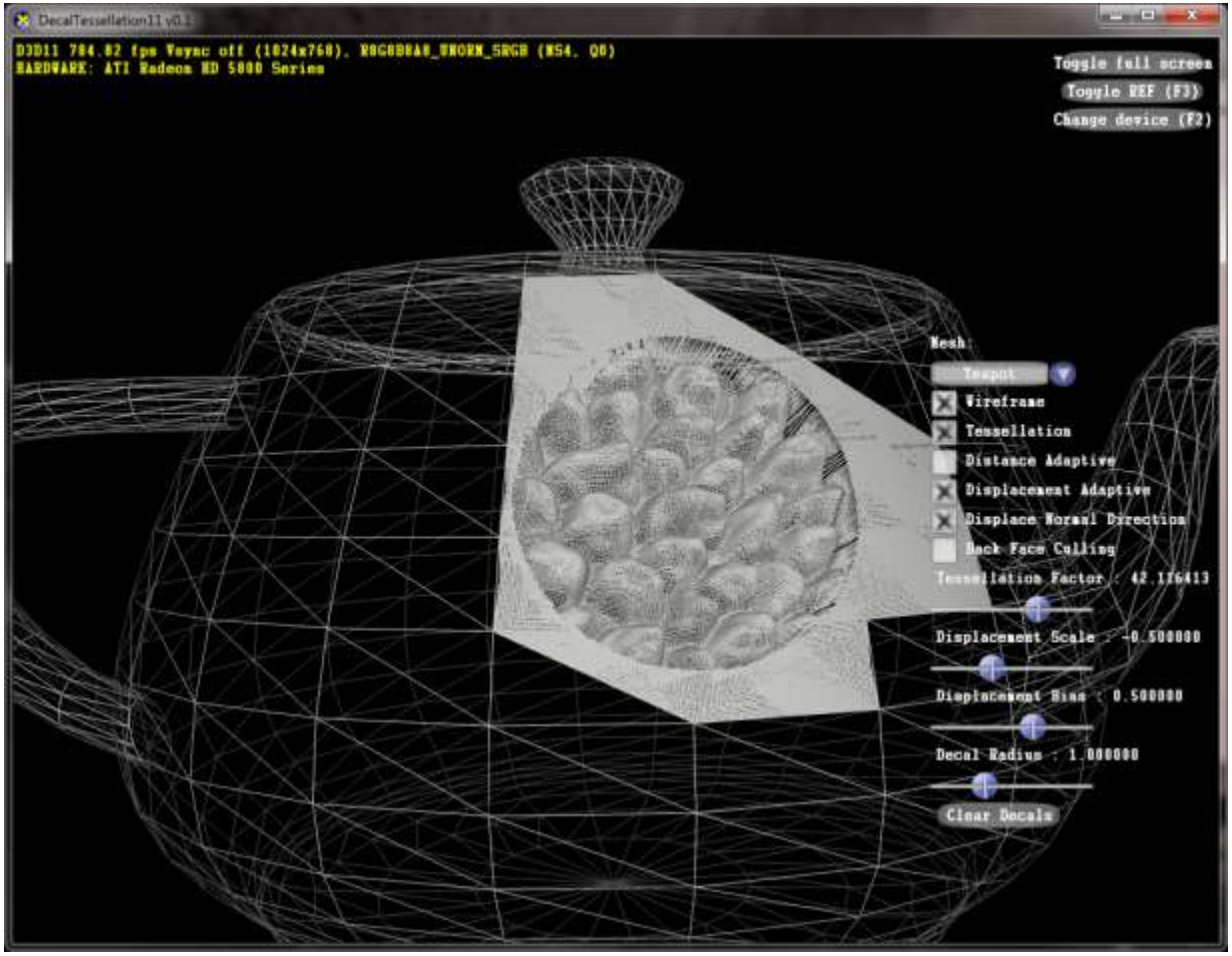
Displacement Adaptive Hull Shader (part 3)

```
  // Inside
  float3 vPlaneNormal = normalize( cross(vEdge0, -vEdge2) );
  // Use the dot product to find distance between point and plane
  float distanceToPlane = abs (dot(vPlaneNormal, vDecalToPos0));
  if (distanceToPlane <= decalRadius)
  {
    // Point in triangle test using barycentric coordinates
    float dotAA = dot(vEdgeA, vEdgeA);
    float dotAB = dot(vEdgeA, vEdgeB);
    float dotBB = dot(vEdgeB, vEdgeB);
    float invDenom = 1.0 / (dotAA * dotBB - dotAB * dotAB);
    float dotAHit = dot(vEdgeA, vHitEdge0);
    float dotBHit = dot(vEdgeB, vHitEdge0);
    float u = (dotBB * dotAHit - dotAB * dotBHit) * invDenom;
    float v = (dotAA * dotBHit - dotAB * dotAHit) * invDenom;
    if ( ((u > 0) && (v > 0) && ((u + v) < 1)) || edgeTessellated )
    {
         Out.Inside = tessellationFactor;
    }
  }
} // end of for loop
```

Displacement Adaptive Tessellation Demo (1)

Displacement adaptive up close: 1434 FPS!

Displacement Adaptive Tessellation Demo (2)



Wireframe view showing tessellation only on affected patches

Displacement Adaptive Results

- Two orders of magnitude improvement in performance when object is close to camera!
- Combined with Distance Adaptive, almost as fast as without tessellation when viewed from far away.
- But wait there’s more …

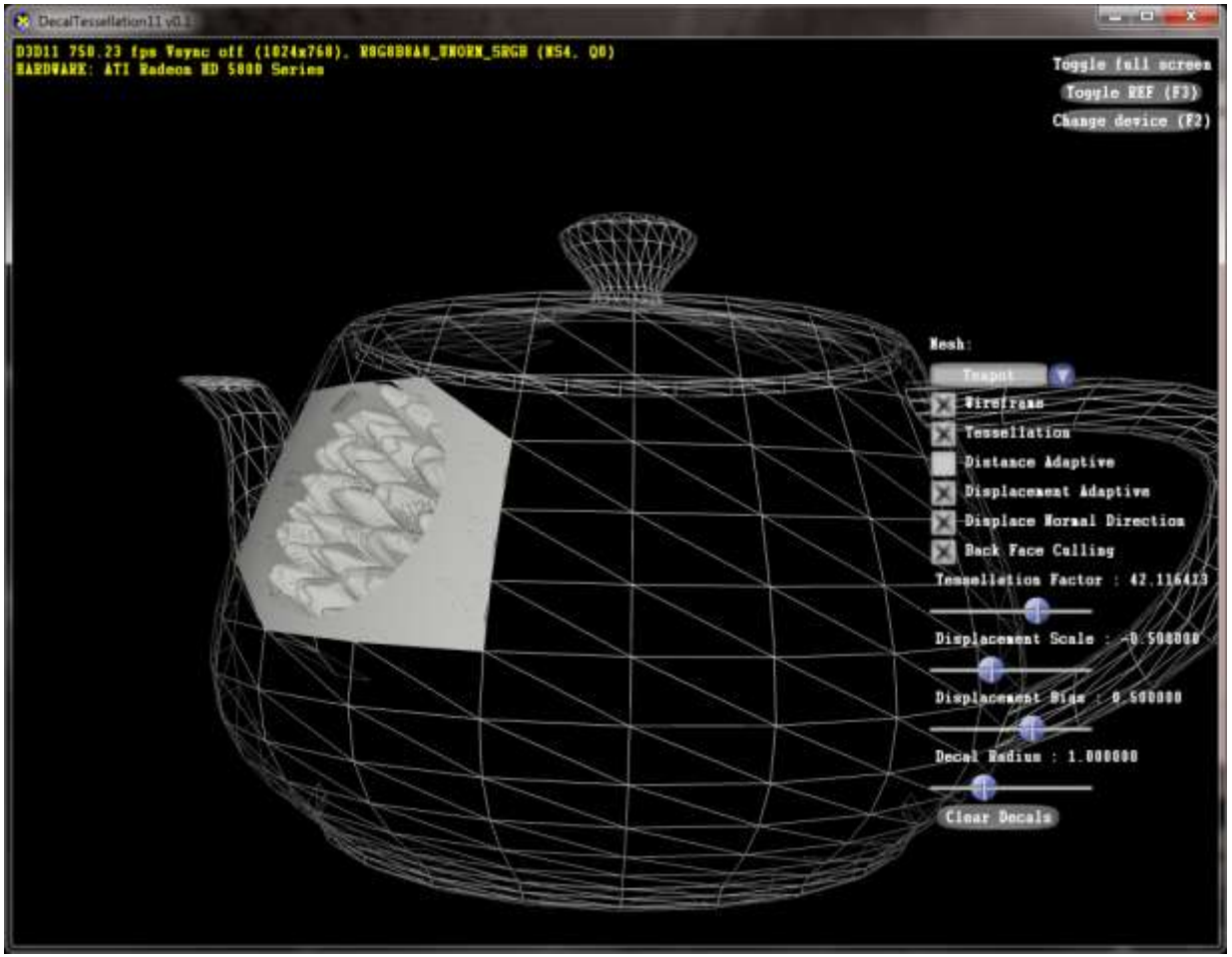
Back-Face Culling

- Why tessellate (or even draw) triangles facing away from the camera that you can't see?
- Do back-face culling in the Hull Shader
- Patches are not drawn if all of the tessellation factors are set to zero.

Back-Face Culling in the Hull Shader

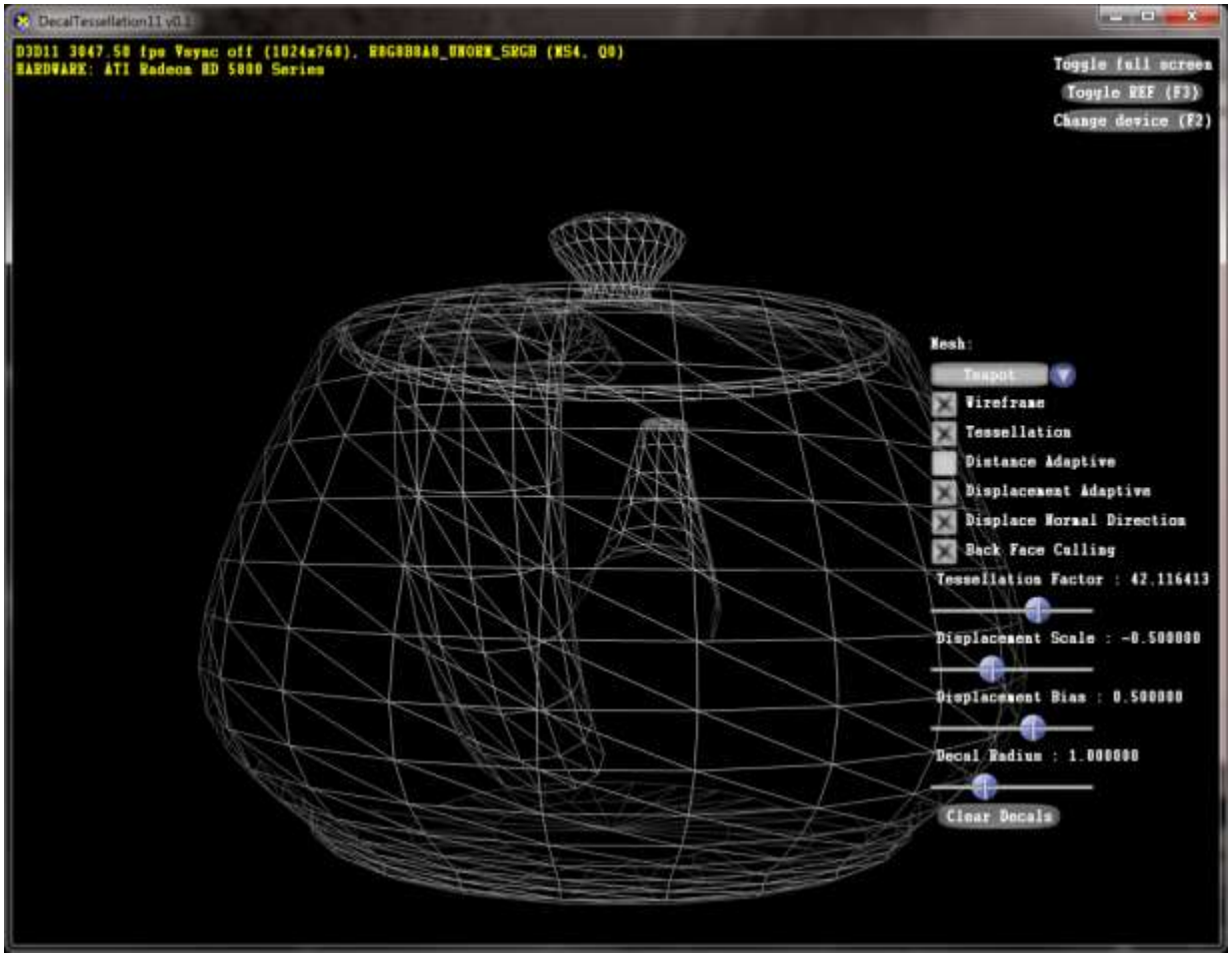
```
// find two triangle patch edges
float3 vEdge0 = vPos1 - vPos0;
float3 vEdge2 = vPos2 - vPos0;

// Create the normal and view vector
float3 vFaceNormal = normalize( cross(vEdge2,vEdge0) );
float3 vView = normalize( vPos0 - g_vEyePosition );

// A negative dot product means facing away from view direction.
// Use a small epsilon to avoid popping, since displaced vertices
// may still be visible with dot product = 0.
if ( dot(vView, vFaceNormal) < -0.25 )
{
     // Cull the triangle by setting the tessellation factors to 0.
     Out.Edges[0] = 0;
     Out.Edges[1] = 0;
     Out.Edges[2] = 0;
     Out.Inside   = 0;
     return Out; // early exit
}
```

Back-Face Culling Demo (1)



Wireframe view with back face culling

Back-Face Culling Demo (2)



Rotated wireframe view with back face culling showing culled patches

Back-Face Culling Results

- About 40% improvement when one decal is culled.
- Easy to implement – only a few lines of code
- Make cull test with a small epsilon to avoid popping

Other Optimizations to Consider

- Frustum Culling
- Group tessellated draw call together
- Orientation Adaptive Tessellation
  - Use dot(V, N) to find silhouette patches
- Hull Shader
  - Try to reduce data passed in and out
- Domain Shader
  - Try to move work to Pixel Shader
  - Reduce data passed out
- Geometry Shader
  - Use Stream Out to avoid re-tessellating an object

Questions ?

bill.bilodeau@amd.com