Game Developers Conference®

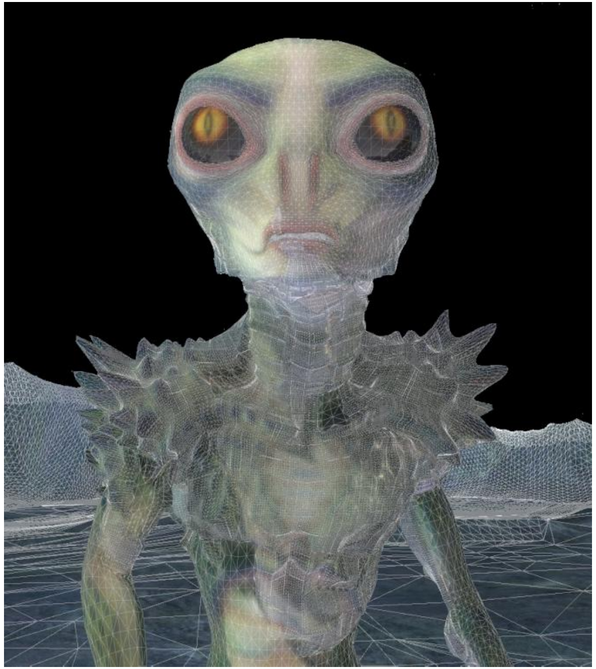
February 28 - March 4, 2011
Moscone Center, San Francisco

www.GDConf.com

GDC 25

UBM

Tessellation on Any Budget

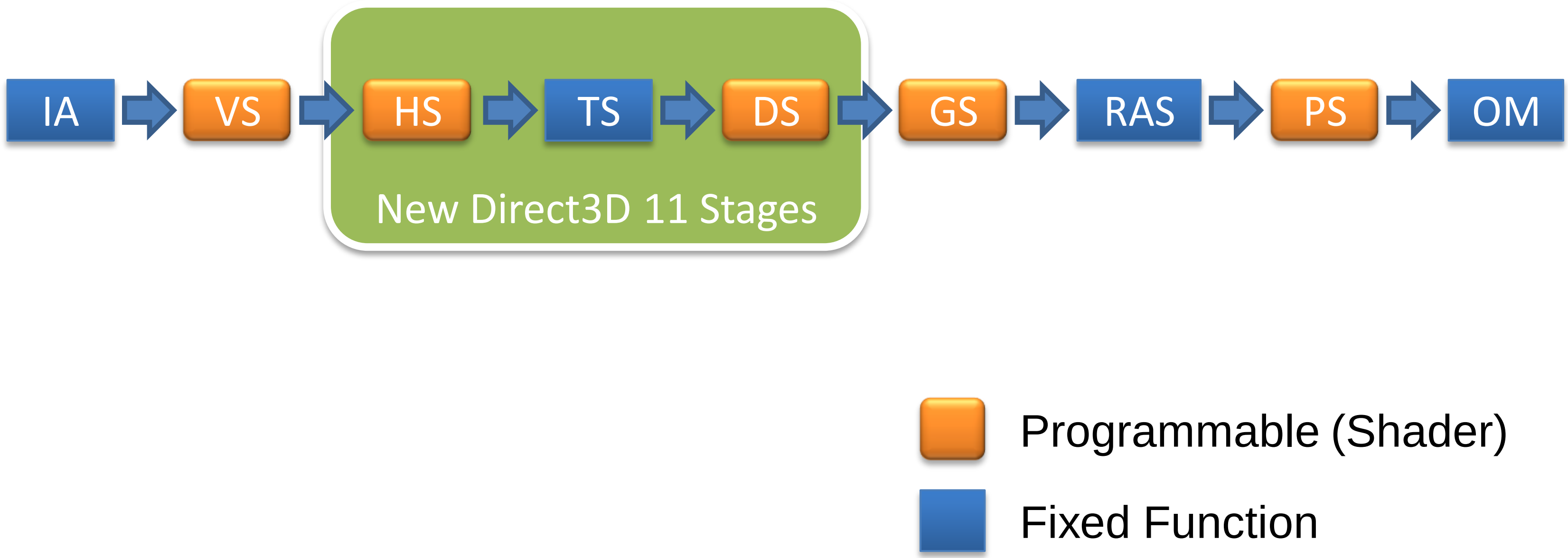


John McDonald
Developer Technology
NVIDIA Corporation

Topics Covered

- Canonical Work Breakdown
- Techniques
- Debugging
- Optimizing

Brief Recap


IA
VS
HS
TS
DS
GS
RAS
PS
OM
New Direct3D 11 Stages
Programmable (Shader)
Fixed Function

Canonical Work Breakdown

- VS: conversion to camera space, control point animation
- HS (CP): Compute Control Point locations, compute per-control point culling info
- HS (PC): Use info from HS (CP) to compute per-edge LOD; cull patches outside frustum

Canonical Work Breakdown cont'd

- DS: Merge Data from HS, TS. Transform from eye to clip space.

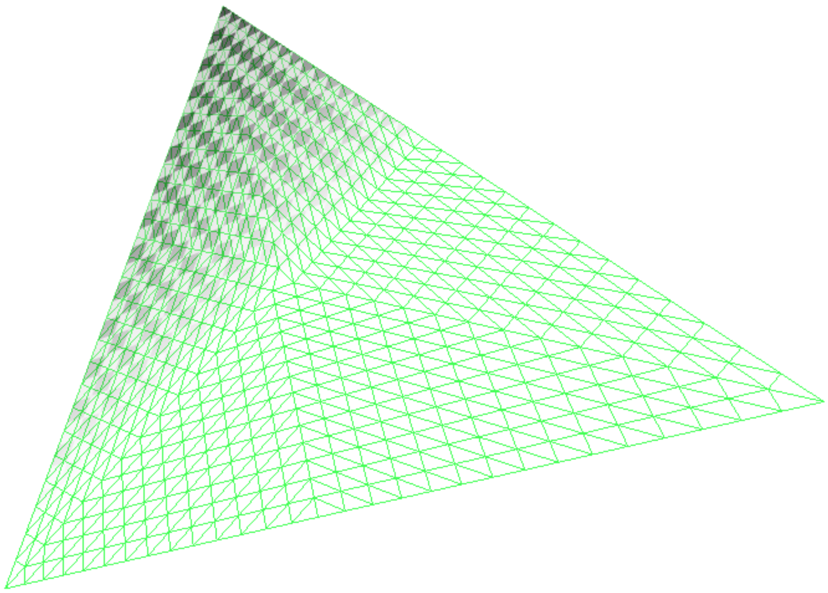
Techniques

- All techniques (except vanilla Flat Dicing) will improve silhouettes and lighting
- But don’t incur the corresponding increase in memory consumption
- And continuous LOD!

Flat Dicing

- Simplest form of tessellation
- Merely adds triangles where fewer were previously
- Does not improve silhouettes alone
  - Usually paired with displacement mapping
  - Can also be used to reduce PS complexity

Flat Dicing



Flat Dicing Code (HS-CP)

```
HS_CPOut HS_FlatTriangles(
  InputPatch<HS_RenderSceneIn, 3> I,
  uint uCPID : SV_OutputControlPointID )
{
  HS_CPOut O = (HS_CPOut)0;
  const uint NextCPID = uCPID < 2 ? uCPID + 1 : 0;

  O.f3ViewPos     = I[uCPID].f3ViewPos;
  O.f3ViewNormal  = I[uCPID].f3ViewNormal;
  O.f2TexCoord    = I[uCPID].f2TexCoord;

  O.fClipped = ComputeClipping(
    g_f4x4Projection,
    O.f3ViewPos
  );

  O.fOppositeEdgeLOD = ComputeEdgeLOD(
    g_f4x4Projection,
    O.f3ViewPos,
    I[NextCPID].f3ViewPos
  );

  return O;
}
```

Flat Dicing Code (HS-PC)

```
HS_ConstOut HS_ConstantFlat(
  const OutputPatch<HS_CPOut, 3> I )
{
  HS_ConstOut O = (HS_ConstOut)0;

  O.fTessFactor[0] = I[1].fOppositeEdgeLOD;
  O.fTessFactor[1] = I[2].fOppositeEdgeLOD;
  O.fTessFactor[2] = I[0].fOppositeEdgeLOD;

  O.fInsideTessFactor[0] = max(
    max(O.fTessFactor[0],
        O.fTessFactor[1]),
    O.fTessFactor[2]);

  if (I[0].fClipped && I[1].fClipped &&
      I[2].fClipped)
  {
    O.fTessFactor[0] = 0;
    O.fTessFactor[1] = 0;
    O.fTessFactor[2] = 0;
  }

  return O;
}
```

Flat Dicing Code (DS)

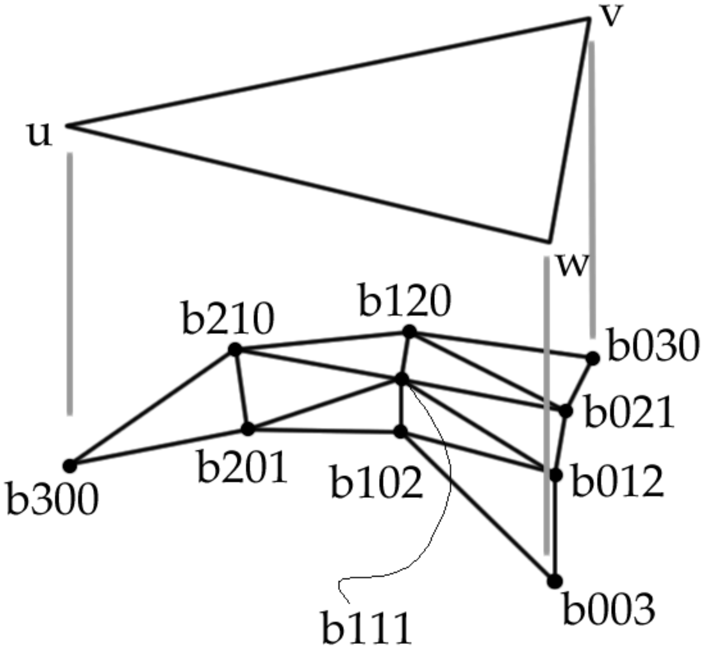
```
DS_Out DS_Flat( HS_ConstOut cdata,
  const OutputPatch<HS_CPOut, 3> I,
  float3 f3BarycentricCoords : SV_DomainLocation )
{
  DS_Out O = (DS_Out)0;

  float fU = f3BarycentricCoords.x;
  float fV = f3BarycentricCoords.y;
  float fW = f3BarycentricCoords.z;

  float3 f3EyePos = I[0].f3ViewPos * fU
                  + I[1].f3ViewPos * fV
                  + I[2].f3ViewPos * fW;
  O.f4ClipPos = ApplyProjection(g_f4x4Projection,
    f3EyePos);

  O.f3Normal = I[0].f3ViewNormal * fU
             + I[1].f3ViewNormal * fV
             + I[2].f3ViewNormal * fW;
  O.f3Normal = normalize( O.f3Normal );

  O.f2TexCoord = I[0].f2TexCoord * fU
               + I[1].f2TexCoord * fV
               + I[2].f2TexCoord * fW;

  return O;
}
```

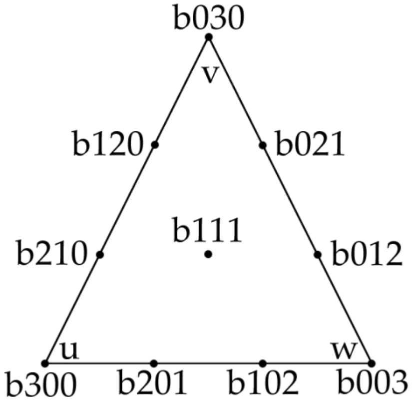
PN

- Originally proposed by Alex Vlachos, et al, in *Curved PN Triangles*.
- Treats primitives as descriptions of Bezier Surfaces, using the location as a position and normal as a description of tangent of surface at that position

PN Details

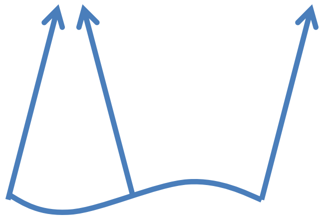
- Uses existing vertex and index buffer without modification





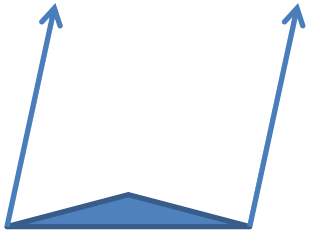
PN Modifications

- PN calls for quadratic interpolation of normals
- This allows for inflection points while lighting curved triangles
- The inflection points **will** show up geometrically
- Skip the quadratic normal for lighting



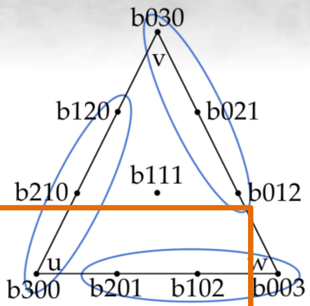
Quadratic Normals

- Per-pixel lighting would require quadratic tangents and binormals as well
  - Lots of extra math
  - Potential for gimbal lock in lighting!



- While correct according to the surface, this does not match artist intent for lighting

PN Code (HS-CP)

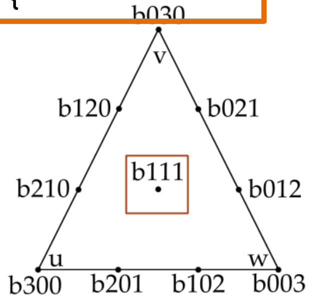



```
HS_CPOut HS_PNTriangles(
  InputPatch<HS_RenderSceneIn, 3> I,
  uint uCPID : SV_OutputControlPointID )
{
  HS_CPOut O = (HS_CPOut)0;
  const uint NextCPID = uCPID < 2 ? uCPID + 1 : 0;

  O.f3ViewPos[0]      = I[uCPID].f3ViewPos;
  O.f3ViewNormal      = I[uCPID].f3ViewNormal;
  O.f2TexCoord        = I[uCPID].f2TexCoord;

  O.f3ViewPos[1] = ComputeCP(I[uCPID].f3ViewPos,
    I[NextCPID].f3ViewPos, I[uCPID].f3ViewNormal);
  O.f3ViewPos[2] = ComputeCP(I[NextCPID].f3ViewPos,
    I[uCPID].f3ViewPos, I[NextCPID].f3ViewNormal);

  O.fClipped = ComputeClipping(
    g_f4x4Projection, O.f3ViewPos[0],
    O.f3ViewPos[1], O.f3ViewPos[2]
  );

  O.fOppositeEdgeLOD = ComputeEdgeLOD(
    g_f4x4Projection, O.f3ViewPos[0],
    O.f3ViewPos[1], O.f3ViewPos[2],
    I[NextCPID].f3ViewPos
  );

  return O;
}

float3 ComputeCP(float3 pA, float3 pB, float3 nA) {
  return (2 * pA + pB - (dot((pB - pA), nA) * nA))
    / 3.0f;
}
```

New Code (as compared to Flat Dicing HS-CP)

PN Code (HS-PC)

```
HS_ConstOut HS_ConstantShared(
  const OutputPatch<HS_CPOut, 3> I )
{
  HS_ConstOut O = (HS_ConstOut)0;
  float3 f3B300 = I[0].f3ViewPos[0],
         f3B210 = I[0].f3ViewPos[1],
         f3B120 = I[0].f3ViewPos[2],
         f3B030 = I[1].f3ViewPos[0],
         f3B021 = I[1].f3ViewPos[1],
         f3B012 = I[1].f3ViewPos[2],
         f3B003 = I[2].f3ViewPos[0],
         f3B102 = I[2].f3ViewPos[1],
         f3B201 = I[2].f3ViewPos[2];

  O.fTessFactor[0] = I[1].fOppositeEdgeLOD;
  O.fTessFactor[1] = I[2].fOppositeEdgeLOD;
  O.fTessFactor[2] = I[0].fOppositeEdgeLOD;
  O.fInsideTessFactor[0] = max(
    max(O.fTessFactor[0],
        O.fTessFactor[1]),
        O.fTessFactor[2]);

  float3 f3E = (f3B210 + f3B120 + f3B021 + f3B012 +
                f3B102 + f3B201) / 6.0f;
  float3 f3V = (f3B003 + f3B030 + f3B300) / 3.0f;
  O.f3ViewB111 = f3E + ((f3E - f3V) / 2.0f);

  float fB111Clipped = IsClipped(
   ApplyProjection(g_f4x4Projection, O.f3ViewB111));

  if (I[0].fClipped && I[1].fClipped &&
      I[2].fClipped && fB111Clipped) {
    O.fTessFactor[0] = 0;
    O.fTessFactor[1] = 0;
    O.fTessFactor[2] = 0;
  }
  return O;
}
```



New Code (as compared to Flat Dicing HS-PC)

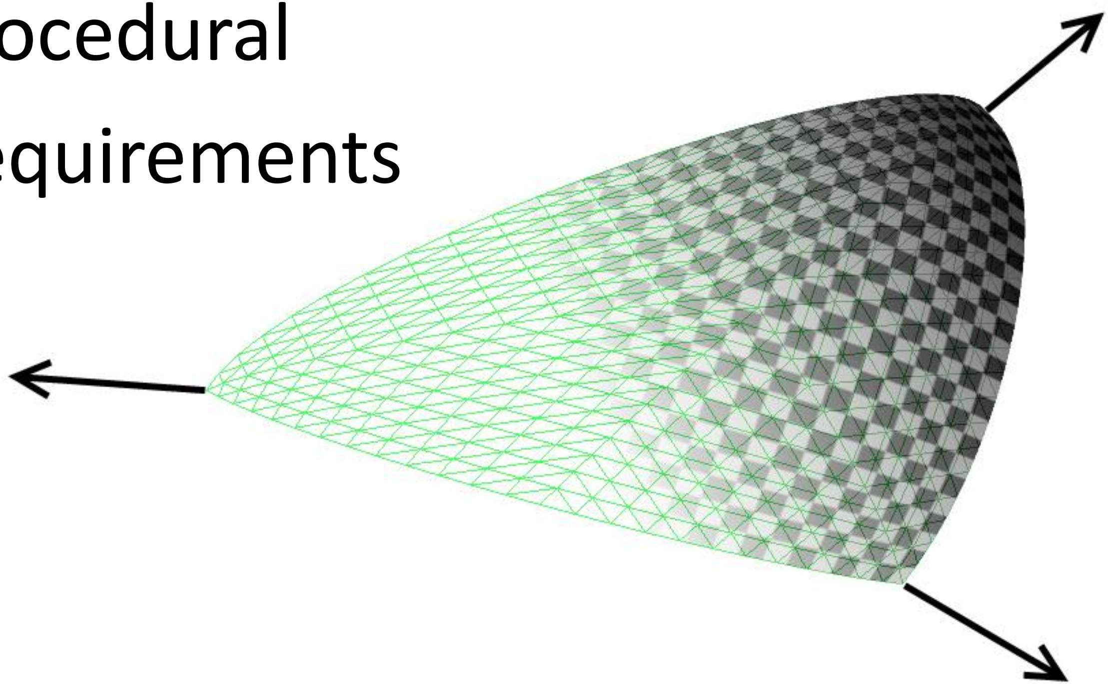
PN Code (DS)

```
DS_Out DS_Shared( HS_ConstOut cdata,
  const OutputPatch<HS_CPOut, 3> I,
  float3 f3BarycentricCoords : SV_DomainLocation )
{
  DS_Out O = (DS_Out)0;
  float fU = f3BarycentricCoords.x;
  float fV = f3BarycentricCoords.y;
  float fW = f3BarycentricCoords.z;
  float fUU = fU * fU;
  float fVV = fV * fV;
  float fWW = fW * fW;
  float fUU3 = fUU * 3.0f;
  float fVV3 = fVV * 3.0f;
  float fWW3 = fWW * 3.0f;

  float3 f3EyePosition =
    I[0].f3ViewPos[0] * fUU * fU
    + I[1].f3ViewPos[0] * fVV * fV
    + I[2].f3ViewPos[0] * fWW * fW
    + I[0].f3ViewPos[1] * fUU3 * fV
    + I[0].f3ViewPos[2] * fVV3 * fU
    + I[1].f3ViewPos[1] * fVV3 * fW
    + I[1].f3ViewPos[2] * fWW3 * fV
    + I[2].f3ViewPos [1] * fWW3 * fU
    + I[2].f3ViewPos [2] * fUU3 * fW
    + cdata.f3ViewB111 * 6.0f * fW * fU * fV;
  O.f4ClipPos = ApplyProjection(g_f4x4Projection,
    f3EyePos);

  O.f3Normal = I[0].f3ViewNormal * fU
             + I[1].f3ViewNormal * fV
             + I[2].f3ViewNormal * fW;
  O.f3Normal = normalize( O.f3Normal );

  O.f2TexCoord = I[0].f2TexCoord * fU
               + I[1].f2TexCoord * fV
               + I[2].f2TexCoord * fW;
  return O;
}
```

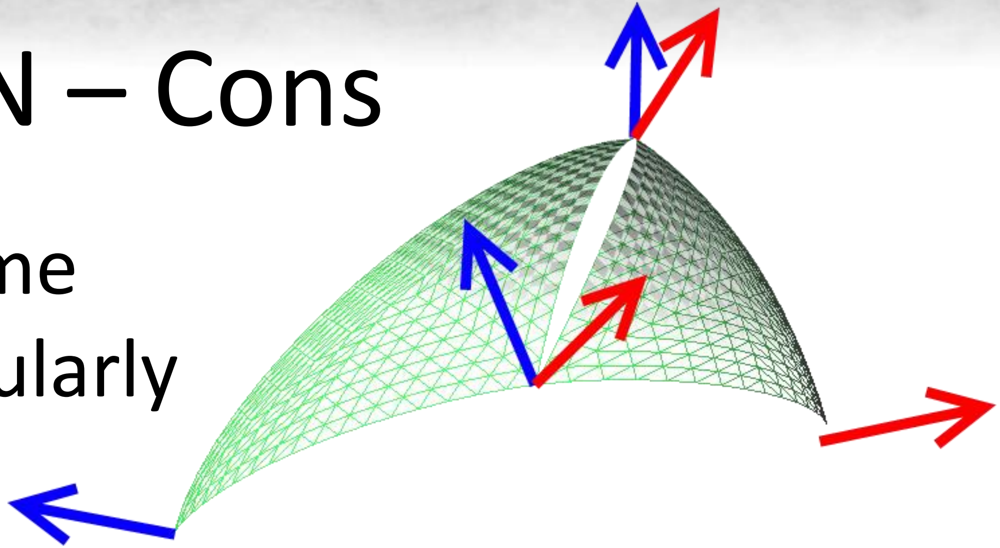
New Code (as compared to Flat Dicing DS)

PN – Pros

- Completely procedural
- No memory requirements



PN – Cons



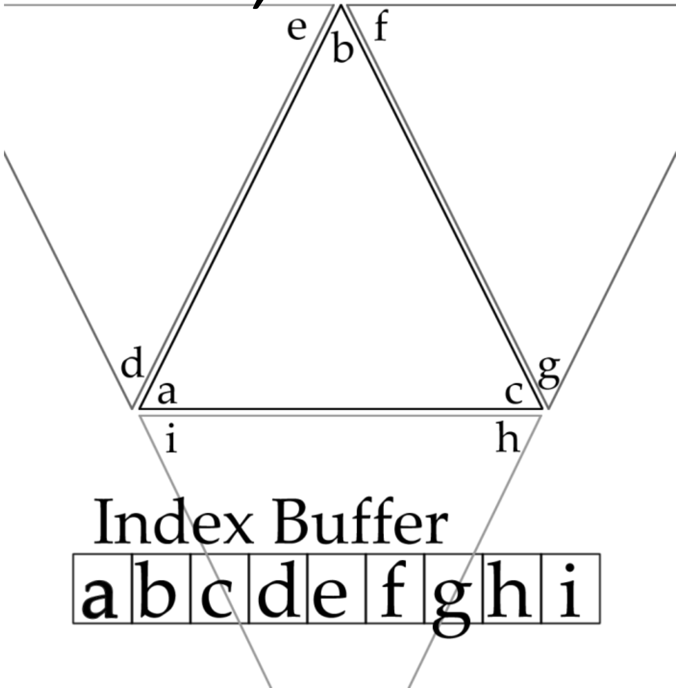
- Meshes can become "Stay Puft", particularly around the feet
- C1 discontinuities (same position, different normal) in input result in C0 discontinuity!
- Fixing discontinuities requires artist involvement

PN-AEN

- Proposed by John McDonald and Mark Kilgard in *Crack-Free Point-Normal Triangles Using Adjacent Edge Normals*
- Uses PN with a twist—neighbor information determined during a preprocess step to avoid cracking

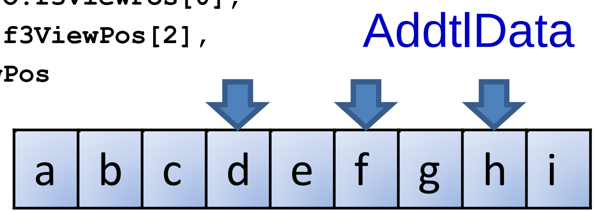
PN-AEN Details

- Uses existing VB without modification, but a second IB must be generated
- Tool available from NVIDIA to generate second IB automatically (works for all vendors)



PN-AEN Code (HS-CP)

```
HS_CPOut HS_PNTriangles(
  InputPatch<HS_RenderSceneIn, 9> I,
  uint uCPID : SV_OutputControlPointID )
{
  HS_CPOut O = (HS_CPOut)0;
  const uint NextCPID = uCPID < 2 ? uCPID + 1 : 0;
  const uint AddtlData = 3 + 2 * uCPID;
  const uint NextAddtlData = AddtlData + 1;

  O.f3ViewPos[0]       = I[uCPID].f3ViewPos;
  O.f3ViewNormal       = I[uCPID].f3ViewNormal;
  O.f2TexCoord         = I[uCPID].f2TexCoord;
  float3 myCP, otherCP;
  myCP = ComputeCP(I[uCPID].f3ViewPos,
    I[NextCPID].f3ViewPos, I[uCPID].f3ViewNormal);
  otherCP = ComputeCP(I[AddtlData].f3ViewPos,
    I[NextAddtlData].f3ViewPos,
    I[AddtlData].f3ViewNormal);
  O.f3ViewPos[1] = (myCP + otherCP) / 2;
  myCP = ComputeCP(I[NextCPID].f3ViewPos,
    I[uCPID].f3ViewPos, I[NextCPID].f3ViewNormal);
  otherCP = ComputeCP(I[NextAddtlData].f3ViewPos,
    I[AddtlData].f3ViewPos,
    I[NextAddtlData].f3ViewNormal);
  O.f3ViewPos[2] = (myCP + otherCP) / 2;

  O.fClipped = ComputeClipping(
    g_f4x4Projection, O.f3ViewPos[0],
    O.f3ViewPos[1], O.f3ViewPos[2]
  );

  O.fOppositeEdgeLOD = ComputeEdgeLOD(
    g_f4x4Projection, O.f3ViewPos[0],
    O.f3ViewPos[1], O.f3ViewPos[2],
    I[NextCPID].f3ViewPos
  );

  return O;
}
```



New Code (as compared to PN HS-CP)

PN-AEN Code (HS-PC)

```
HS_ConstOut HS_ConstantShared(
  const OutputPatch<HS_CPOut, 3> I )
{
  HS_ConstOut O = (HS_ConstOut)0;
  float3 f3B300 = I[0].f3ViewPos[0],
         f3B210 = I[0].f3ViewPos[1],
         f3B120 = I[0].f3ViewPos[2],
         f3B030 = I[1].f3ViewPos[0],
         f3B021 = I[1].f3ViewPos[1],
         f3B012 = I[1].f3ViewPos[2],
         f3B003 = I[2].f3ViewPos[0],
         f3B102 = I[2].f3ViewPos[1],
         f3B201 = I[2].f3ViewPos[2];

  O.fTessFactor[0] = I[1].fOppositeEdgeLOD;
  O.fTessFactor[1] = I[2].fOppositeEdgeLOD;
  O.fTessFactor[2] = I[0].fOppositeEdgeLOD;
  O.fInsideTessFactor[0] = max(
    max(O.fTessFactor[0],
        O.fTessFactor[1]),
        O.fTessFactor[2]);

  float3 f3E = (f3B210 + f3B120 + f3B021 + f3B012 +
                f3B102 + f3B201) / 6.0f;
  float3 f3V = (f3B003 + f3B030 + f3B300) / 3.0f;
  O.f3ViewB111 = f3E + ((f3E - f3V) / 2.0f);

  float fB111Clipped = IsClipped(
    ApplyProjection(g_f4x4Projection, O.f3ViewB111));

  if (I[0].fClipped && I[1].fClipped &&
        I[2].fClipped && fB111Clipped) {
      O.fTessFactor[0] = 0;
      O.fTessFactor[1] = 0;
      O.fTessFactor[2] = 0;
  }
  return O;
}
```

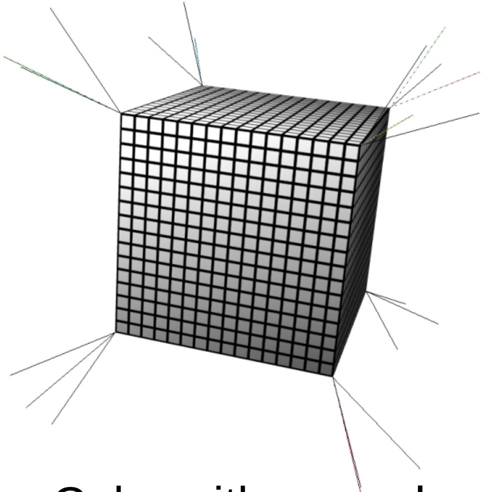
New Code (as compared to PN HS-PC)

PN-AEN Code (DS)

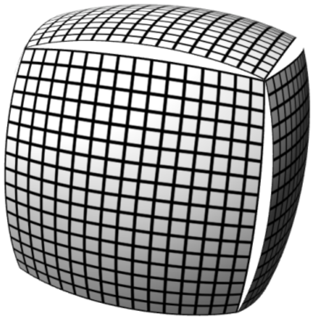
```
DS_Out DS_Shared( HS_ConstOut cdata,
  const OutputPatch<HS_CPOut, 3> I,
  float3 f3BarycentricCoords : SV_DomainLocation )
{
  DS_Out O = (DS_Out)0;
  float fU = f3BarycentricCoords.x;
  float fV = f3BarycentricCoords.y;
  float fW = f3BarycentricCoords.z;
  float fUU = fU * fU;
  float fVV = fV * fV;
  float fWW = fW * fW;
  float fUU3 = fUU * 3.0f;
  float fVV3 = fVV * 3.0f;
  float fWW3 = fWW * 3.0f;

  float3 f3EyePosition =
    I[0].f3ViewPos[0] * fUU * fU
    + I[1].f3ViewPos[0] * fVV * fV
    + I[2].f3ViewPos[0] * fWW * fW
    + I[0].f3ViewPos[1] * fUU3 * fV
    + I[0].f3ViewPos[2] * fVV3 * fU
    + I[1].f3ViewPos[1] * fVV3 * fW
    + I[1].f3ViewPos[2] * fWW3 * fV
    + I[2].f3ViewPos [1] * fWW3 * fU
    + I[2].f3ViewPos [2] * fUU3 * fW
    + cdata.f3ViewB111 * 6.0f * fW * fU * fV;
  O.f4ClipPos = ApplyProjection(g_f4x4Projection,
    f3EyePos);

  O.f3Normal = I[0].f3ViewNormal * fU
             + I[1].f3ViewNormal * fV
             + I[2].f3ViewNormal * fW;
  O.f3Normal = normalize( O.f3Normal );

  O.f2TexCoord = I[0].f2TexCoord * fU
               + I[1].f2TexCoord * fV
               + I[2].f2TexCoord * fW;
  return O;
}
```

PN-AEN – Pros

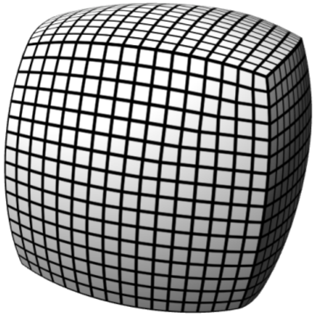
- Completely Procedural
- Small memory overhead



Cube with normals



PN



PN-AEN

PN-AEN – Cons

- More expensive (runtime cost) than PN
- Still can result in some ‘Stay Pufting’ of meshes
- No artist involvement means less artistic control
- Requires second index buffer, 9 indices pp

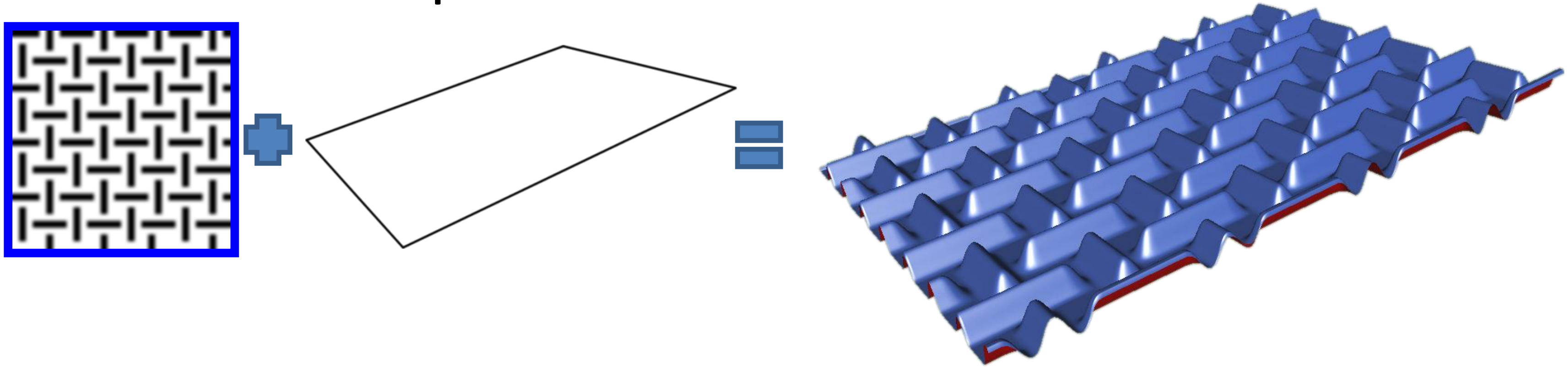
Displacement Mapping

- Used together with another tessellation technique (often Flat Dicing)
- Texture controls displacement at each generated vertex during Domain Shading

Wretch used courtesy Epic Games, Inc

Displacement Mapping Details

- Requires displacement map to be authored
  - Although tools exist to generate from normal maps



Displacement Mapping – Pros

- High impact silhouette and lighting adjustments
- “Pay as you go”: Easy to add displacement to “key” assets without adding to all assets

Displacement Mapping – Cons

- Care must be taken to avoid:
  - Mesh swimming when LOD changes
  - Cracks between patches
- Artist involvement means money being spent

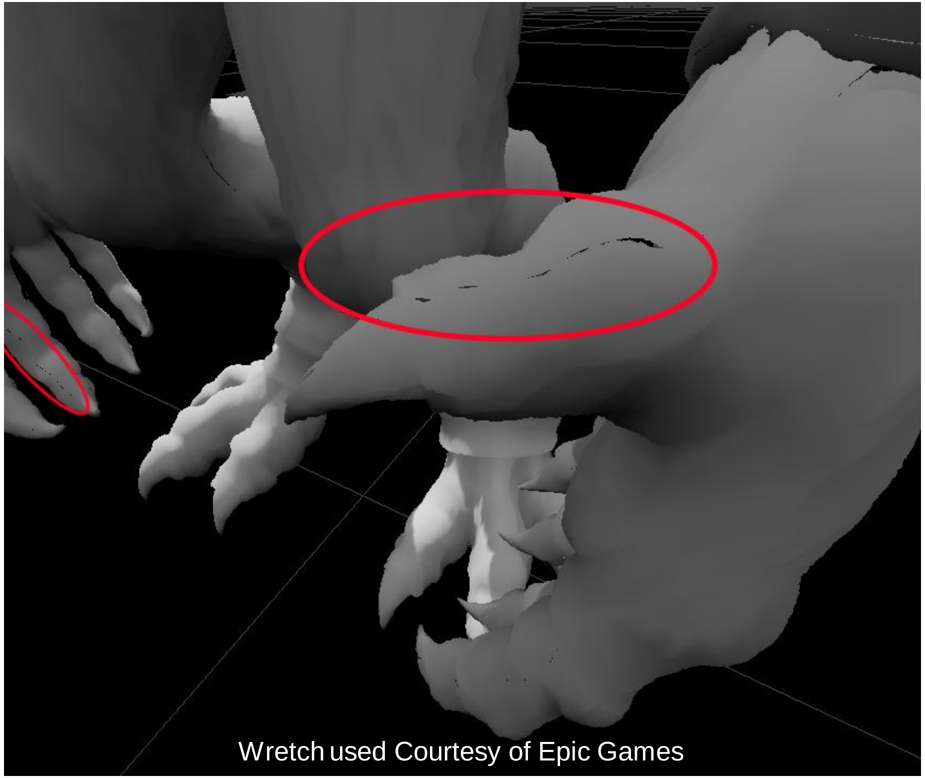
Continuity

- Games have had continuity errors forever
- Normal/lighting discontinuities break C1.
- Tessellation, particularly displacement mapping, makes breaking C0 easy.
- This is undesirable

Discontinuity

- What causes discontinuities?
  - Vertices with same position pre-tessellation, but different position post-tessellation
  - Math Errors (Edge LOD calculation for triangles)
  - Displacing along normals when normals are disjoint
  - Sampling Errors

Discontinuity



Sampling Errors?!

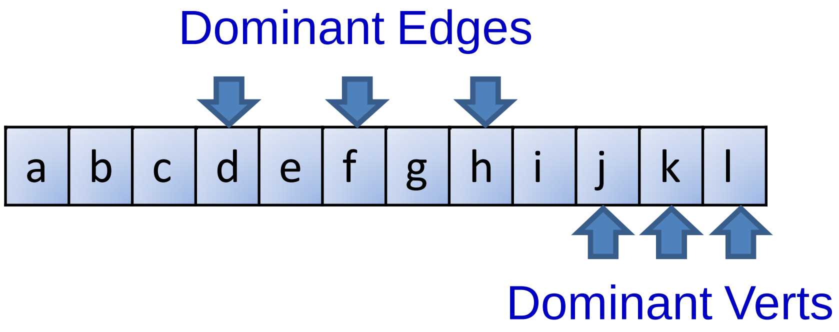
- Impossible to ensure bit-accurate samples across texture discontinuities
- With normal maps, this causes a lighting seam
- With tessellation, this causes a surface discontinuity

Discontinuity Solution

- For each patch, store dominant edge/dominant vertex information
- Detect that you’re at an edge or corner
- If so, sample UVs from dominant information instead of self.
- Everyone agrees, cracks are gone!

Discontinuity Solution cont'd

- Orthogonal to choice of tessellation (works everywhere!)



Discontinuity Code (HS-CP)

```
HS_CPOut HS_FlatTrianglesCrackFree(
  InputPatch<HS_RenderSceneIn, 3+9> I,
  uint uCPID : SV_OutputControlPointID )
{
  HS_CPOut O = (HS_CPOut)0;
  const uint NextCPID = uCPID < 2 ? uCPID + 1 : 0;
  const uint DomEdge = uCPID * 2 + 3;
  const uint DomVert = uCPID + 9;

  O.f3ViewPos     = I[uCPID].f3ViewPos;
  O.f3ViewNormal  = I[uCPID].f3ViewNormal;
  O.f2TexCoord    = I[uCPID].f2TexCoord;

  O.f2DomEdgeTC[0] = I[DomEdge].f2TexCoord;
  O.f2DomEdgeTC[1] = I[DomEdge+1].f2TexCoord;
  O.f2DomVertTC    = I[DomVert].f2TexCoord;

  O.fClipped = ComputeClipping(
    g_f4x4Projection,
    O.f3ViewPos
  );

  O.fOppositeEdgeLOD = ComputeEdgeLOD(
    g_f4x4Projection,
    O.f3ViewPos,
    I[NextCPID].f3ViewPos
  );

  return O;
}
```

New Code (as compared to Flat Dicing HS-CP)

Flat + Displacement (DS)

```
DS_Out DS_FlatDisplace( HS_ConstOut cdata,
  const OutputPatch<HS_CPOut, 3> I,
  float3 f3BarycentricCoords : SV_DomainLocation )
{
  DS_Out O = (DS_Out)0;

  float fU = f3BarycentricCoords.x;
  float fV = f3BarycentricCoords.y;
  float fW = f3BarycentricCoords.z;

  float3 f3EyePos = I[0].f3ViewPos * fU
                  + I[1].f3ViewPos * fV
                  + I[2].f3ViewPos * fW;

  O.f3Normal = I[0].f3ViewNormal * fU
             + I[1].f3ViewNormal * fV
             + I[2].f3ViewNormal * fW;
  O.f3Normal = normalize( O.f3Normal );

  O.f2TexCoord = I[0].f2TexCoord * fU
               + I[1].f2TexCoord * fV
               + I[2].f2TexCoord * fW;

  f3EyePos += g_txDisplace.Sample( s_Displace,
    O.f2TexCoord ) * O.f3Normal;

  O.f4ClipPos = ApplyProjection(g_f4x4Projection,
    f3EyePos);

  return O;
}
```

New Code (as compared to Flat Dicing DS)

Discontinuity Code (DS)

```
{
  float fU = f3BarycentricCoords.x;
  float fV = f3BarycentricCoords.y;
  float fW = f3BarycentricCoords.z;
  // ...
  float
    uCorner =  (fU == 1 ? 1:0),
    vCorner =  (fV == 1 ? 1:0),
    wCorner =  (fW == 1 ? 1:0),
    uEdge =    (fU == 0 && fV * fW ? 1:0),
    vEdge =    (fV == 0 && fU * fW ? 1:0),
    wEdge =    (fW == 0 && fU * fV ? 1:0),
    interior = (fU * fV * fW) ? 1:0;

  float2 f2DisplaceCoord =
      uCorner * I[0].f2DomVertTC
    + vCorner * I[1].f2DomVertTC
    + wCorner * I[2].f2DomVertTC
    + uEdge * lerp(I[1].f2DomEdgeTC[0],
                   I[1].f2DomEdgeTC[1], fV)
    + vEdge * lerp(I[2].f2DomEdgeTC[0],
                   I[2].f2DomEdgeTC[1], fW)
    + wEdge * lerp(I[0].f2DomEdgeTC[0],
                   I[0].f2DomEdgeTC[1], fU)
    + interior * O.f2TexCoord;

  f3EyePos += g_txDisplace.Sample( s_Displace,
    f2DisplaceCoord ) * O.f3Normal;

  // ...
  return O;
}
```

New Code (as compared to Flat + Displacement DS)

Other Tessellation Techniques

- Phong Tessellation
  - Works with existing assets
  - No artist intervention required
  - Suffers same issue as PN (C1 input discontinuity result in C0 output discontinuity)

Other Tessellation Techniques

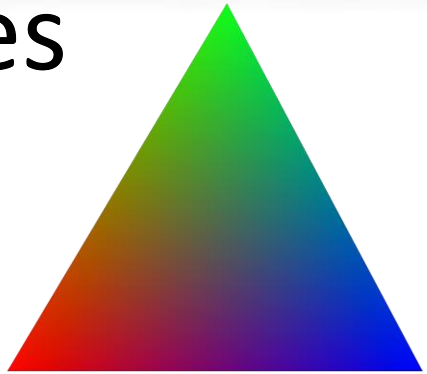
- Catmull Clark sub-d surfaces
  - Great for the future
  - Not great for fitting into existing engines
- NURBS
  - Not a great fit for the GPU
  - Definitely not a good fit for existing engines

Summary / Questions

Technique	Production Cost	Runtime Cost	Value Add
Flat Dicing	Free	~Free	May improve perf, good basis for other techniques
PN	May require art fixes	Small runtime overhead	Improved silhouettes/lighting
PN-AEN	Free	Additional indices pulled versus PN	Crack free, better silhouettes/lighting, preserve hard edges
Displacement	Requires art, but pay as you go	1 texture lookup	Works with other techniques, allows very fine detail

Debugging Techniques

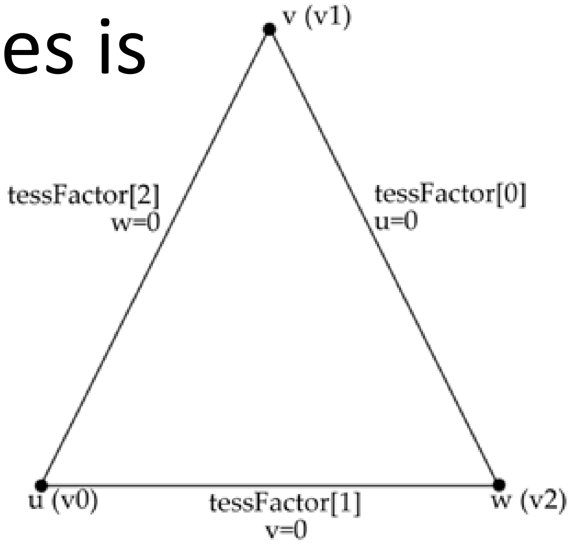
- Verify your conventions
  - Output Barycentric coordinates as diffuse color
- Reduce shader to flat tessellation, add pieces back
- Remove clipping, clever optimizations



Barycentric Coordinates as colors

Debugging Techniques cont'd

- Edge LOD specification for triangles is surprising
- Parallel nSight – Version 1.51 is available for free



Optimization Strategies

- Work at the lowest frequency appropriate
- Be aware that with poor LOD computation, DS could run *more* than the PS.
- Shade Control Points in Vertex Shader to leverage V$
- Clipping saves significant workload

Optimization Strategies cont'd

- Code should be written to maximize SIMD parallelism
- Prefer shorter Patch Constant shaders (only one thread per patch)

Optimization Strategies cont’d

- Avoid tessellation factors <2 if possible
  - Paying for GPU to tessellate when expansion is very low is just cost—no benefit
- Avoid tessellation factors that would result in triangles smaller than 3 pix/side (PS will still operate at quad-granularity)

Questions?

- jmcdonald at nvidia dot com

NVIDIA @ GDC 2011

CAN'T GET ENOUGH? MORE WAYS TO LEARN:

NVIDIA GAME TECHNOLOGY THEATER

Wed, March 2nd and Fri, March 4th @ NVIDIA Booth

Open to all attendees. Featuring talks and demos from leading developers at game studios and more, covering a wide range of topics on the latest in GPU game technology.

NVIDIA DEVELOPER SESSIONS

All Day Thurs, March 3rd @ Room 110, North Hall E

Open to all attendees. Full schedule on www.nvidia.com/gdc2011

MORE DEVELOPER TOOLS & RESOURCES

Available online 24/7 @ developer.nvidia.com

WE'RE HIRING

More info @ careers.nvidia.com

NVIDIA Booth
South Hall #1802
Details on schedule and to download copies of presentations visit
www.nvidia.com/gdc2011

Appendix

```
float4 ApplyProjection(float4x4 projMatrix,
                       float3 eyePosition)
{
  float4 clipPos;
  clipPos[0] = projMatrix[0][0] * eyePosition[0];
  clipPos[1] = projMatrix[1][1] * eyePosition[1];
  clipPos[2] = projMatrix[2][2] * eyePosition[2] + projMatrix[3][2];
  clipPos[3] = eyePosition[2];

  return clipPos;
}

float2 ProjectAndScale(float4x4 projMatrix, float3 inPos)
{
  float4 posClip = ApplyProjection(projMatrix, inPos);
  float2 posNDC = posClip.xy / posClip.w;
  return posNDC * g_f4ViewportScale.xy / g_f4TessFactors.z;
}
```

Appendix

```
float ComputeClipping(float4x4 projMatrix, float3 cpA, float3 cpB, float3 cpC)
{
  float4 projPosA = ApplyProjection(projMatrix, cpA),
         projPosB = ApplyProjection(projMatrix, cpB),
         projPosC = ApplyProjection(projMatrix, cpC);

  return min(min(IsClipped(projPosA), IsClipped(projPosB)), IsClipped(projPosC));
}
```

Note: This isn't quite correct for clipping—it will clip primitives that are so close to the camera that the control points are all out of bounds. The correct clipping would actually store in-bounds/out-of bounds for each plane, then determine if all points failed any one plane.

Appendix

```
float ComputeEdgeLOD(float4x4 projMatrix, float3 cpA, float3 cpB, float3 cpC, float3 cpD)
{
  float2 projCpA = ProjectAndScale(projMatrix, cpA).xy,
         projCpB = ProjectAndScale(projMatrix, cpB).xy,
         projCpC = ProjectAndScale(projMatrix, cpC).xy,
         projCpD = ProjectAndScale(projMatrix, cpD).xy;

  float edgeLOD = distance(projCpA, projCpB)
                + distance(projCpB, projCpC)
                + distance(projCpC, projCpD);

  return max(edgeLOD, 1);
}
```

References

- Vlachos, Alex, Jorg Peters, Chas Boyd, and Jason L. Mitchell. "Curved PN Triangles." *Proceedings of the 2001 Symposium on Interactive 3D Graphics* (2001): 159-66. Print.
- McDonald, John, and Mark Kilgard. "Crack-Free Point-Normal Triangles Using Adjacent Edge Normals." *developer.nvidia.com*. NVIDIA Corporation, 21 Dec. 2010. Web. 25 Feb. 2011.