Tessellation of Displaced Subdivision Surfaces
in DX11

Bryan Dudash

Overview

- Motivation
- Subdivision Surfaces
  - Control point evaluation
  - Surface evaluation
- Displacement Mapping
- Content Creation

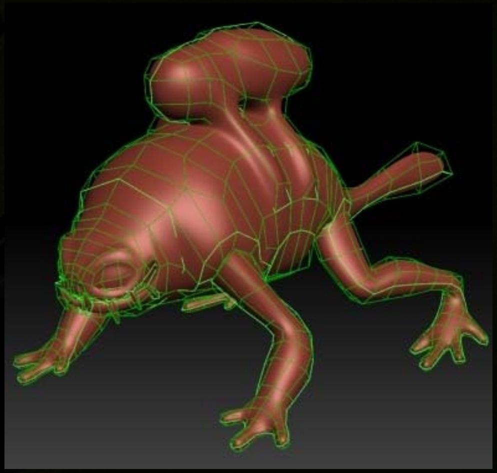
Motivation

© Kenneth Scott, id Software 2008

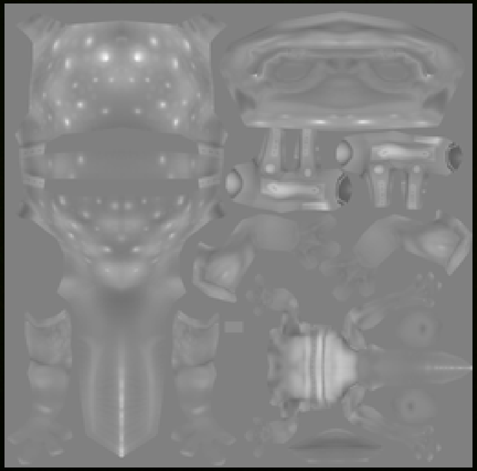
Compression

- **Save memory and bandwidth**
  - **Memory is the main bottleneck to render highly detailed surfaces**

=



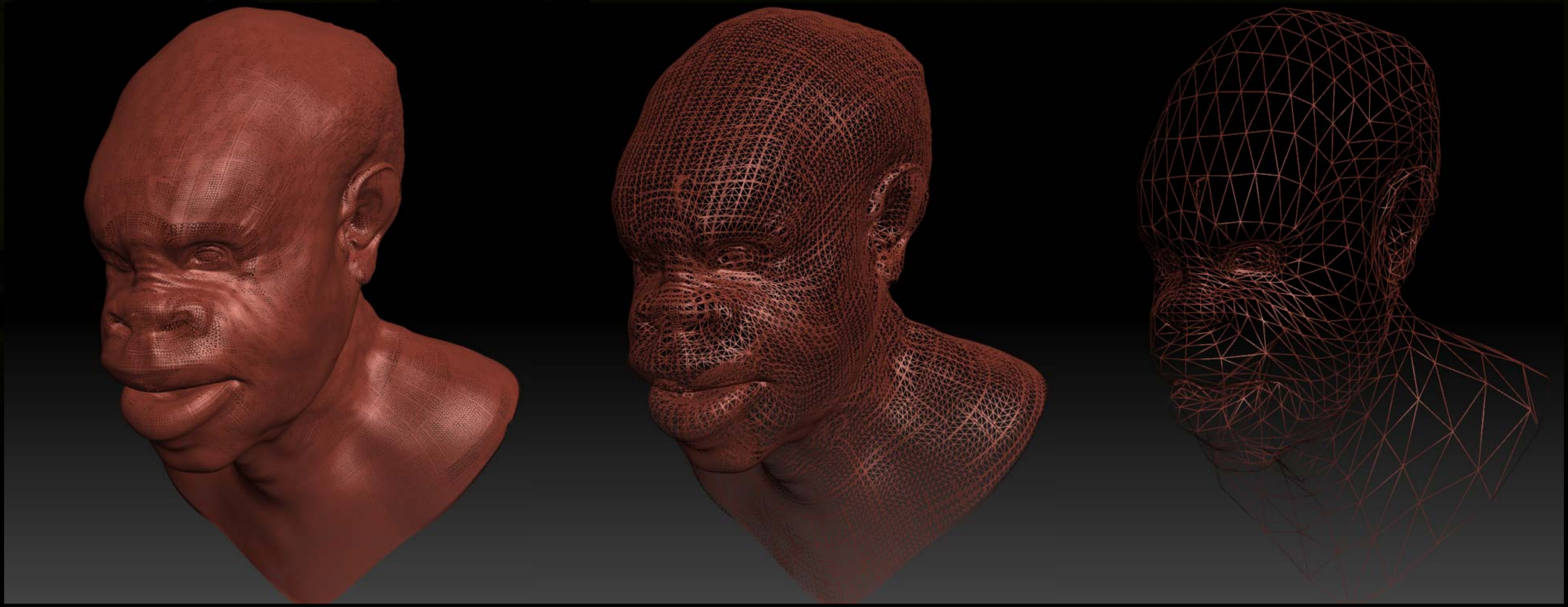
+



	Level 8	Level 16	Level 32	Level 64
Regular Triangle Mesh	16MB	59MB	236MB	943MB
Displaced Subdivision Surface	1.9MB	7.5MB	30MB	118MB

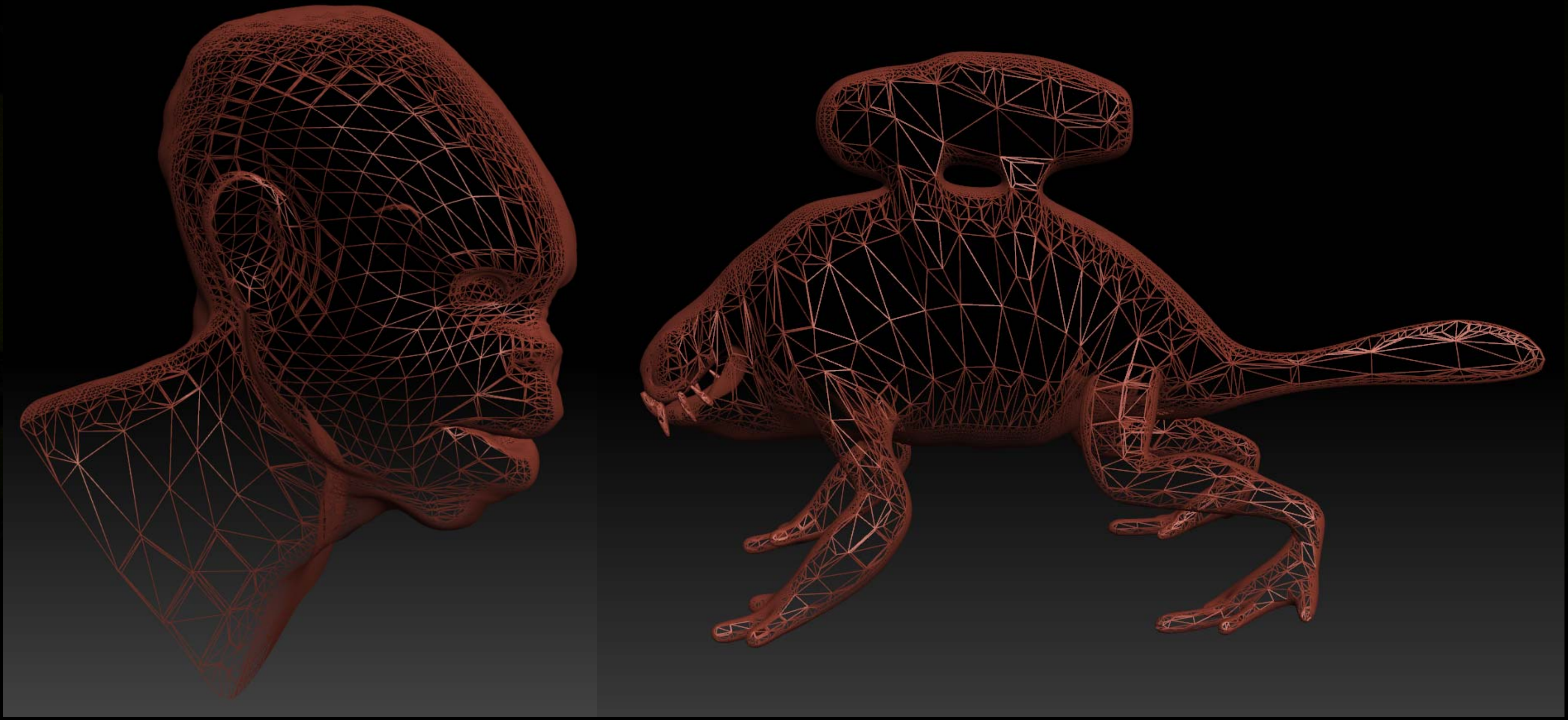
Scalability

- Continuous Level of Detail



Scalability

- View Dependent Level of Detail



Animation & Simulation

- Perform Expensive Computations at lower frequency:
  - Realistic animation: blend shapes, morph targets, etc.

- Physics, collision detection, soft body dynamics, etc.

Goal

- Enable unprecedented visuals:
  - Highly detailed characters
  - Realistic animation

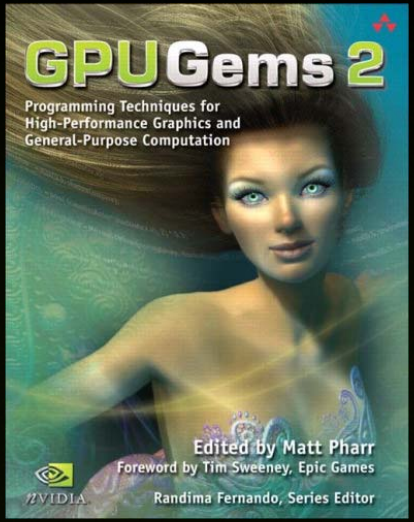
Subdivision Surfaces

Subdivision Surfaces

- Easy modeling and flexible animation
- Widespread use in the movie industry
- Readily available in modeling and sculpting tools

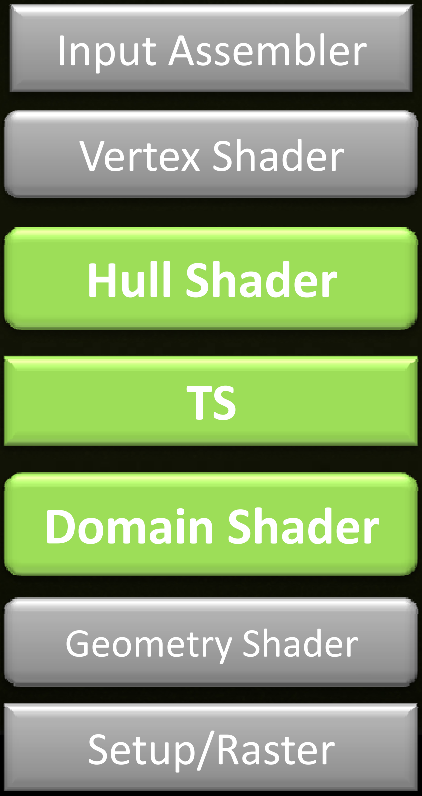
GPU Implementations

- Previous approaches on the GPU:
  - “Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping”, Michael Bunnell
  - Recursive Geometry Shader refinement
- Require multiple passes → Direct evaluation is preferred


GPU Gems 2
Programming Techniques for High-Performance Graphics and General-Purpose Computation
Edited by Matt Pharr
Foreword by Tim Sweeney, Epic Games
Randima Fernando, Series Editor
NVIDIA

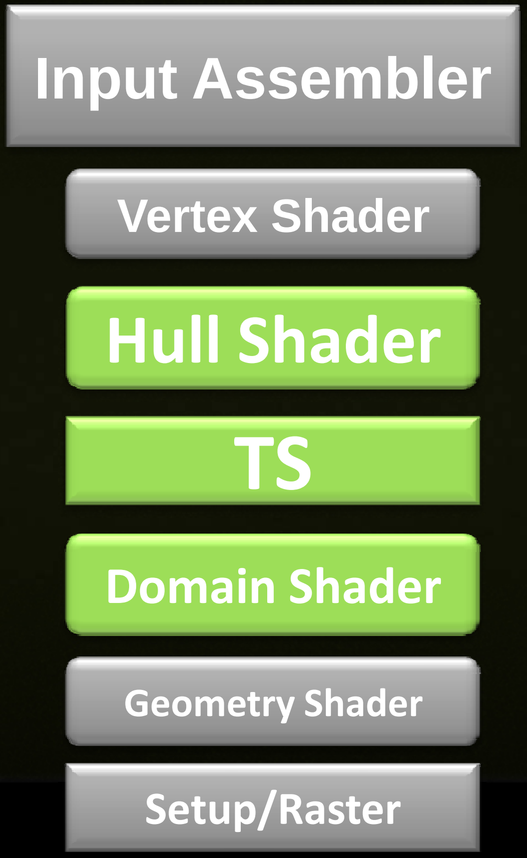
Tessellation Pipeline

- Direct3D11 extends Direct3D10 with support for programmable tessellation
- Two new shader stages:
  - Hull Shader (HS)
  - Domain Shader (DS)
- One fixed function stage:
  - Tessellator (TS)



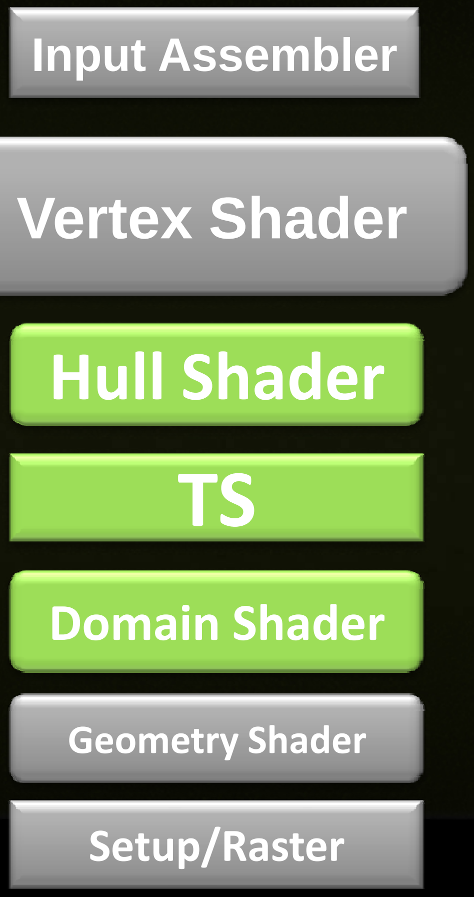
Input Assembler

- New Patch primitive type
  - Arbitrary vertex count
  - No implied topology
  - Only supported primitive when tessellation is enabled



Vertex Shader

- Transforms patch control points
- Usually used for:
  - Animation (skinning, blend shapes)
  - Physics simulation
- Allows more expensive animation at a lower frequency

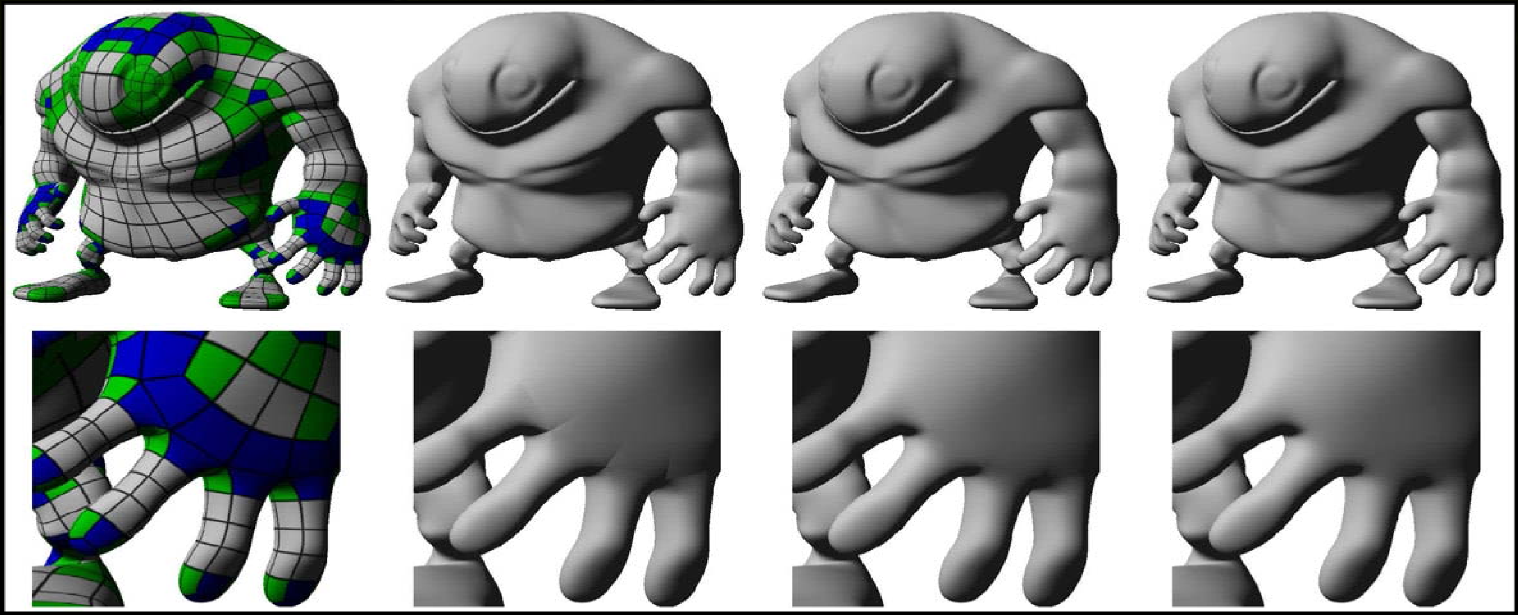


Direct Evalution of Subdiv. Surfaces

- Jos Stam: "Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values"
  - Requires extraordinary vertices to be isolated
  - Evaluation is quite expensive
- Jeff Bolz and Peter Schroeder: "Evaluation of Subdivision Surfaces on Programmable Graphics Hardware"
  - Requires pre-computed basis for each topology and each possible tessellation level

Approximating Catmull-Clark Subdivision Surfaces (ACC)

- Developed by Charles Loop and Scott Shaefer: http://research.microsoft.com/~cloop/
- Surface approximated with a Bezier patch and a pair of independent tangent patches



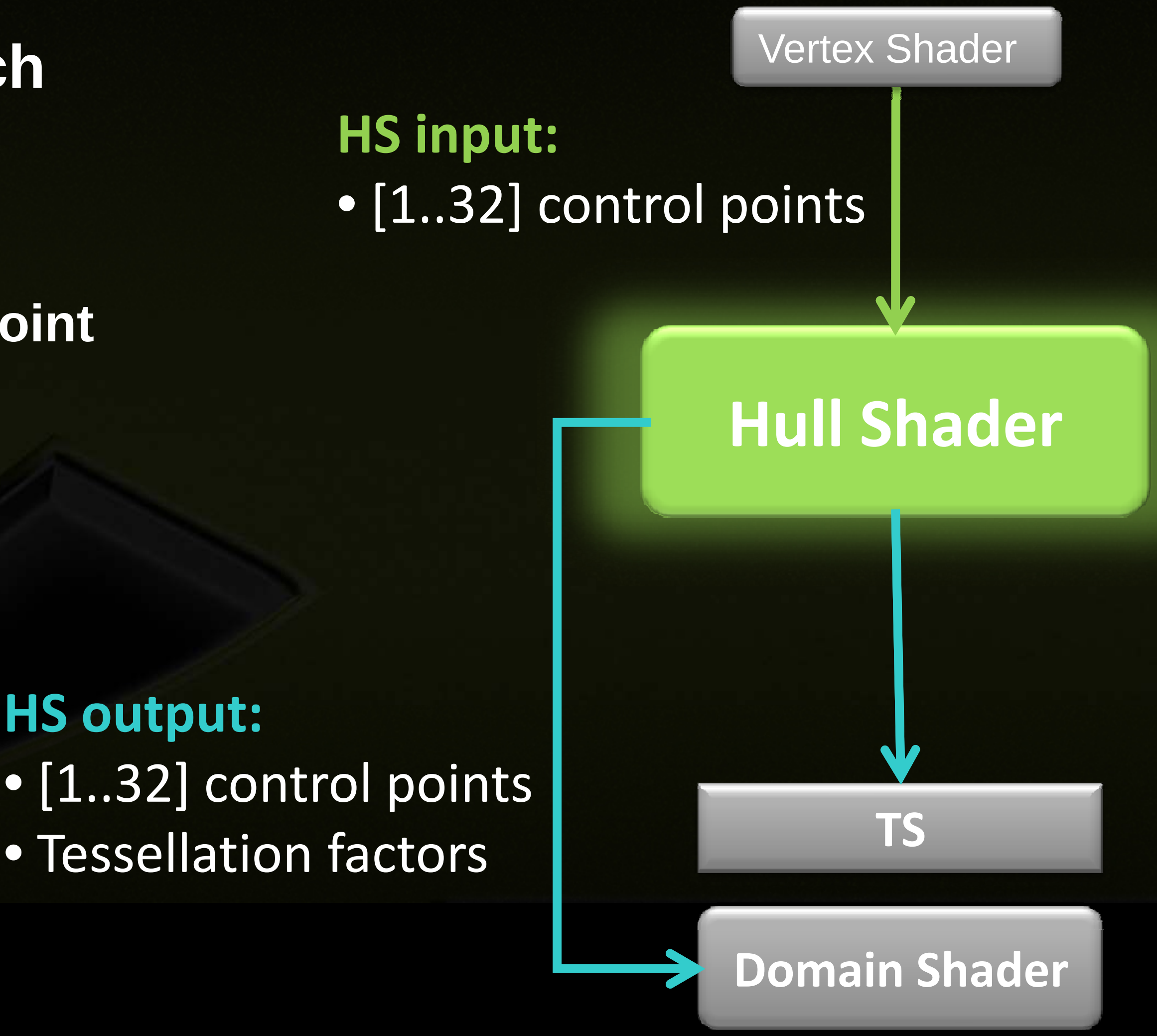
Geometry Approximation

Geometry/Tangent Approximation

Subdivision Surface

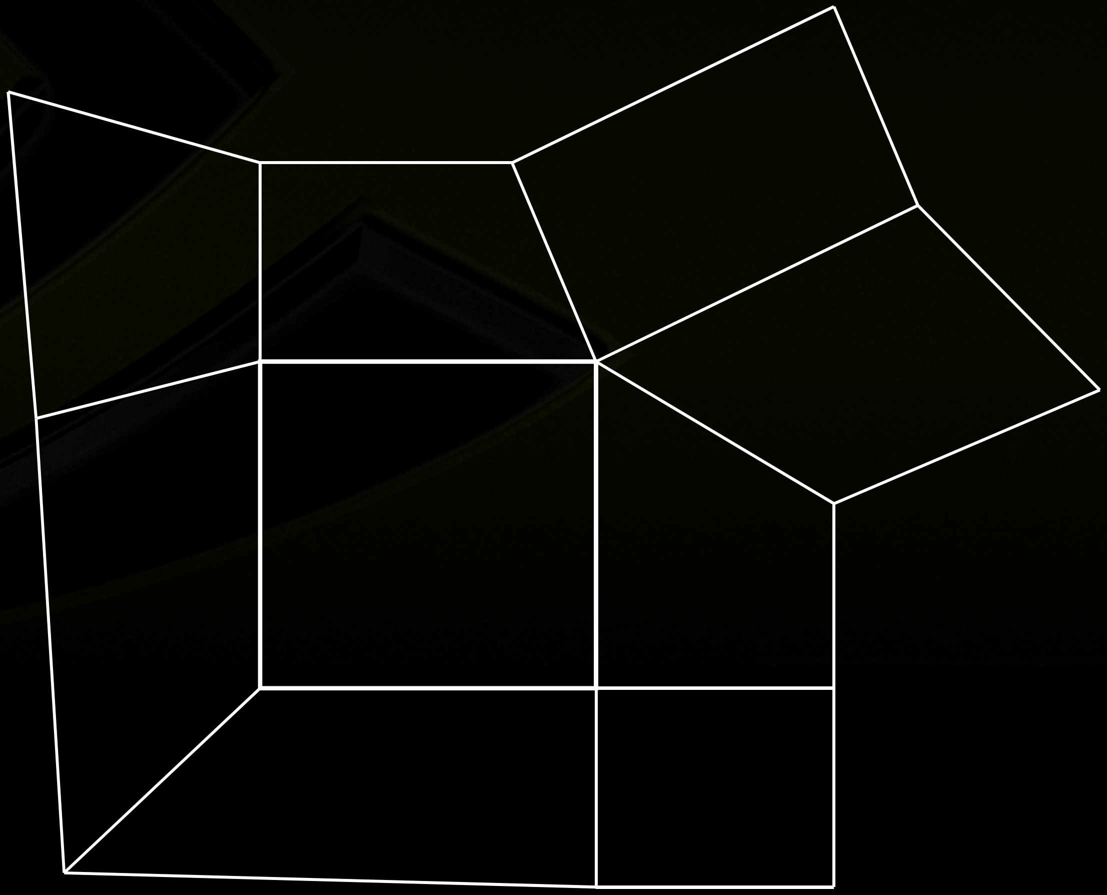
Hull Shader (HS)

- **One invocation per patch**
- **Parallelized explicitly**
  - **One thread per control point**



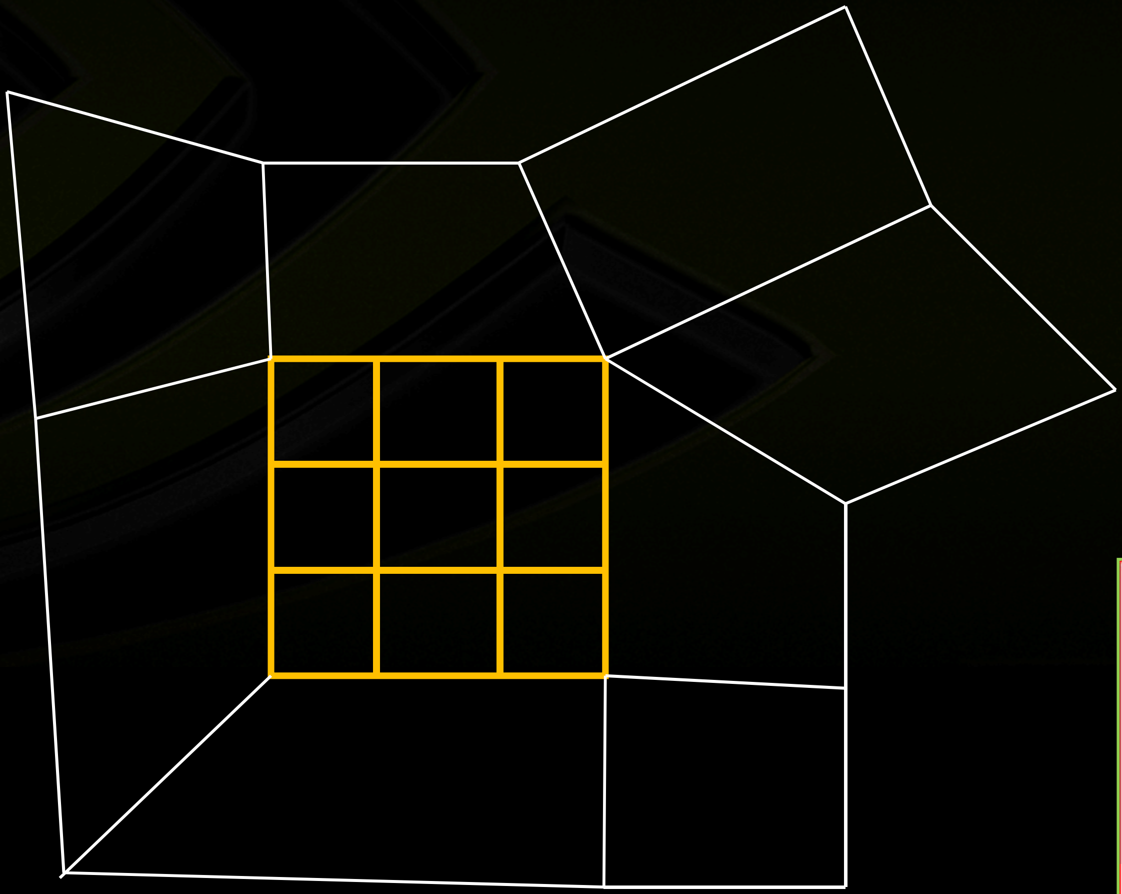
Control Point Evaluation

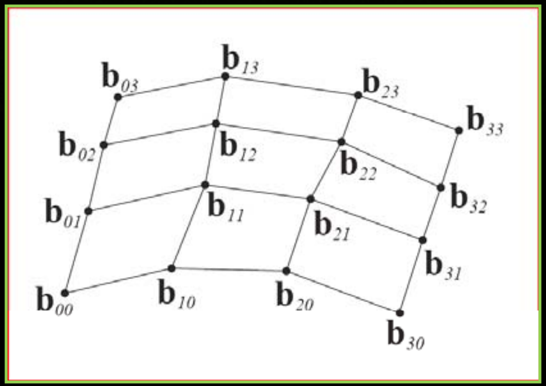
- HS input is a face and its neighborhood:



Control Point Evaluation

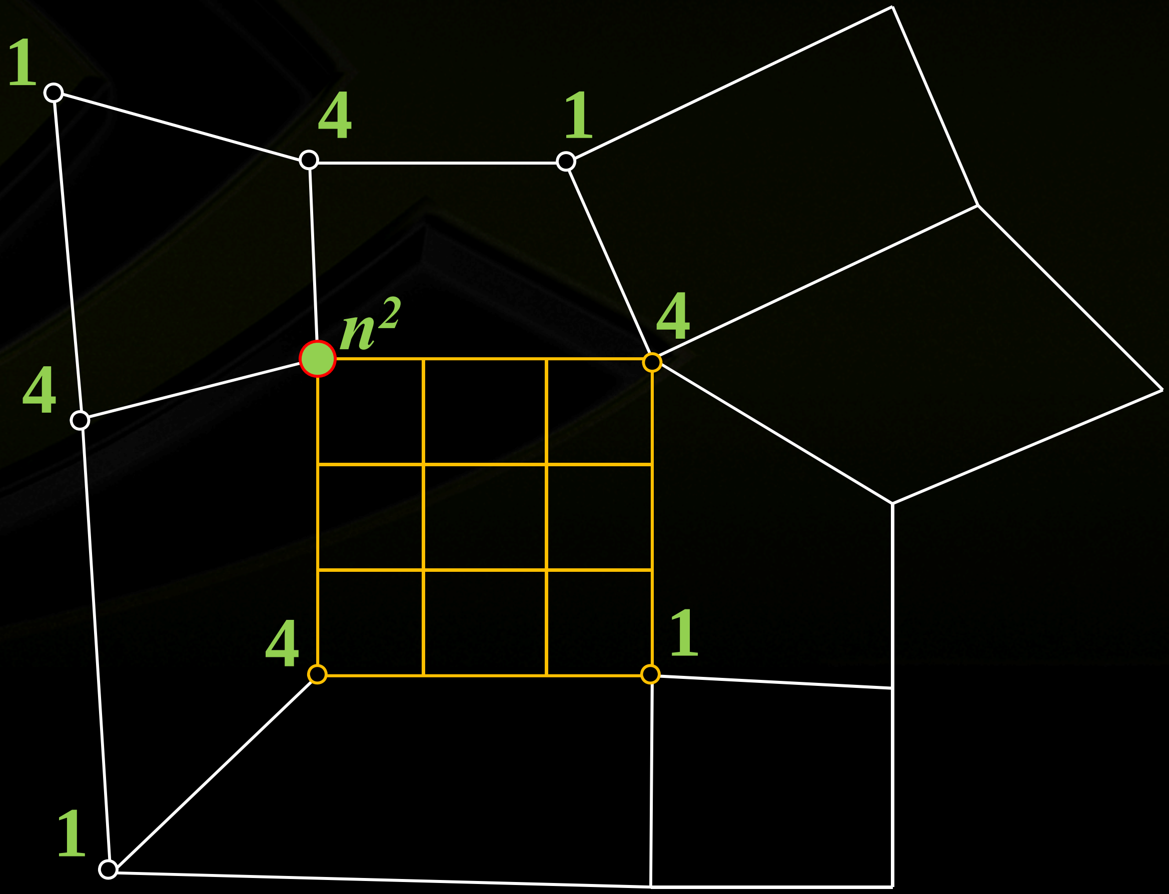
- **HS output is a regular bicubic Bezier patch:**





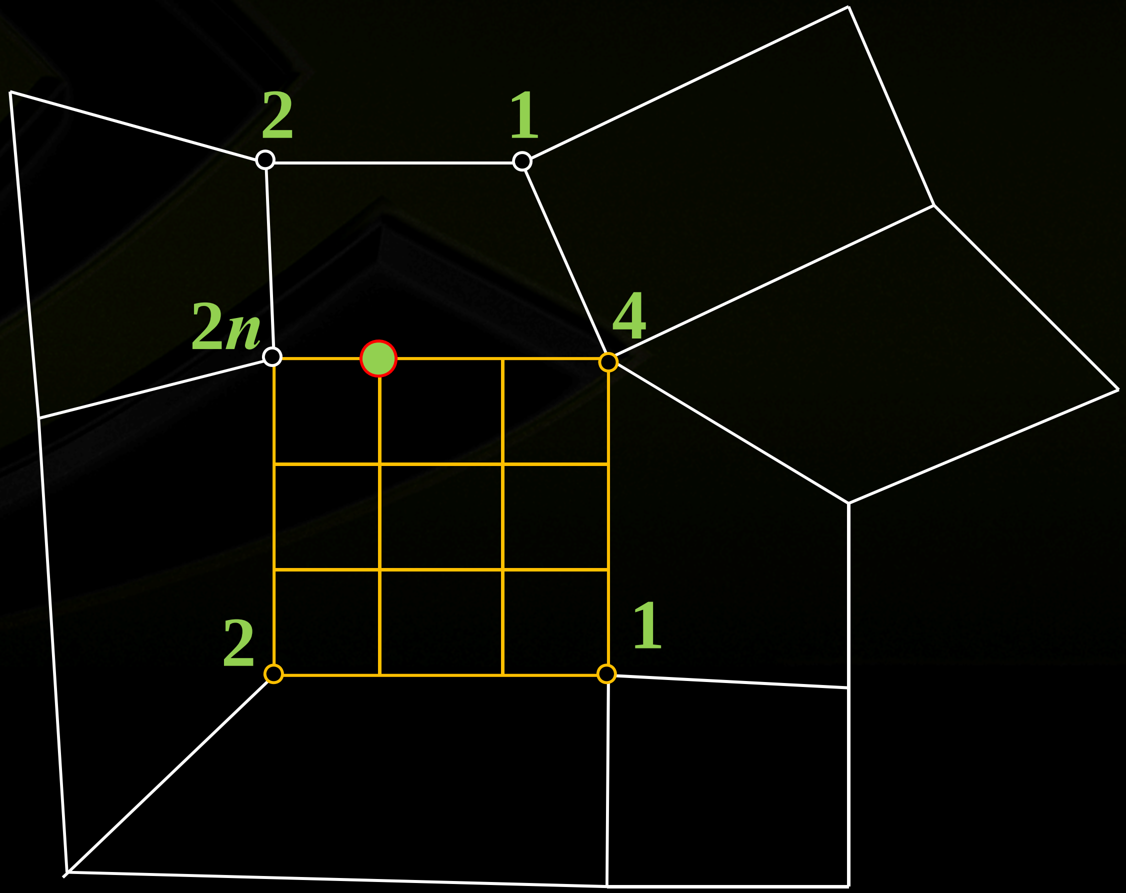
Control Point Evaluation

- Each control point is a linear combination of the neighbor vertices:



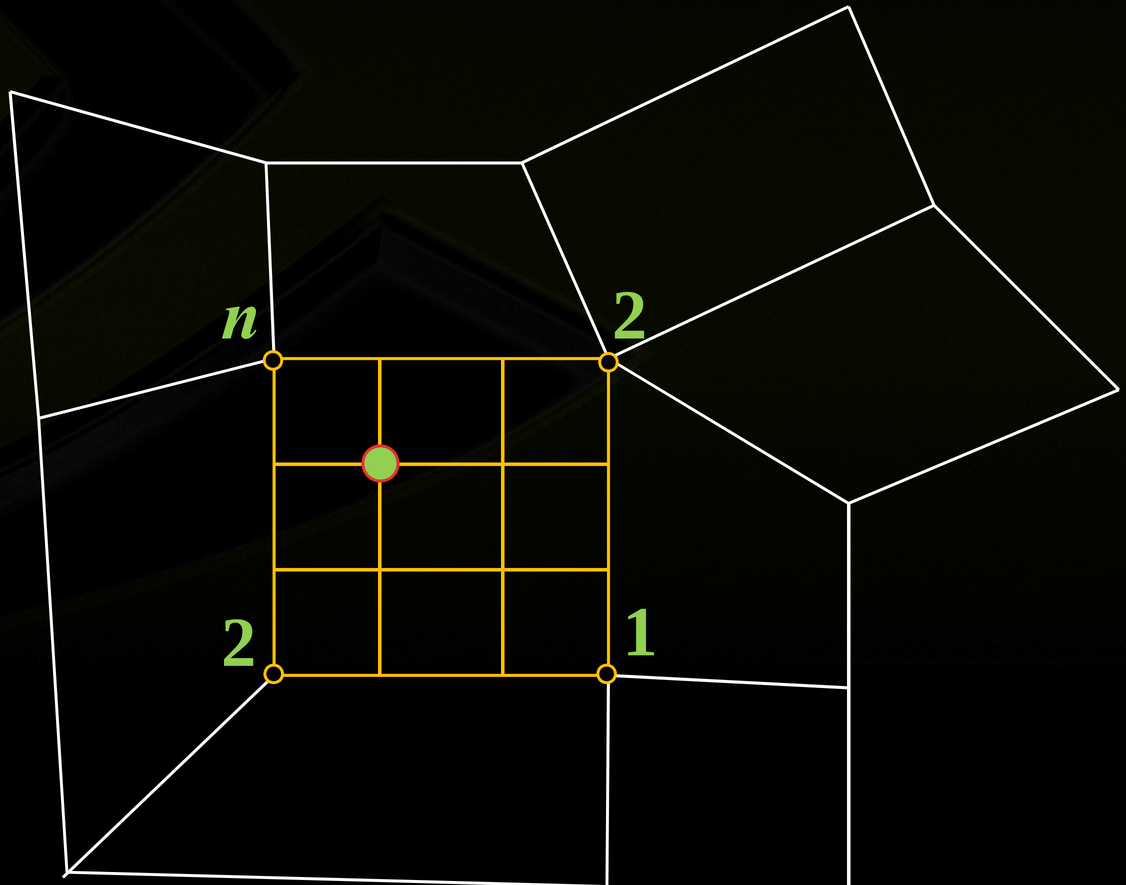
Control Point Evaluation

- Each control point is a linear combination of the neighbor vertices:



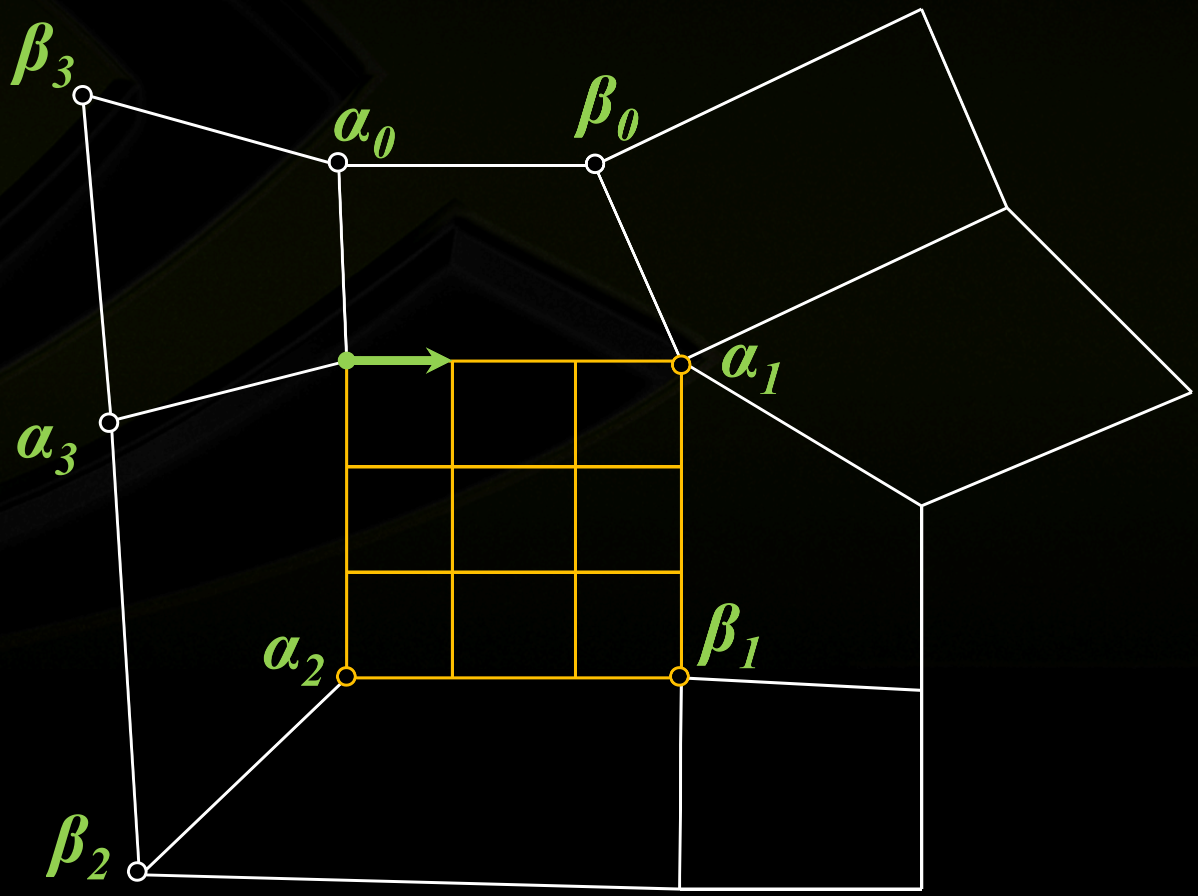
Control Point Evaluation

- **Each control point is a linear combination of the neighbor vertices:**



Control Point Evaluation

- The same is true for control tangents:



Control Point Evaluation

- In all cases we can evaluate a control point as a **weighted sum**: $P_j = \mathrm{Sum}(W_{ij} * V_i)$
- We can implement that in HS using one thread per control point:

```
global float w[K][16];     } One set of constants for each topology combination
in float3 V[K];            } Input vertices
out float3 pos[16];        } Output control points

void main() {
     float3 p = 0.0;
     for (int i = 0; i < K; i++) {
          p += V[i] * w[i][threadID];
     }
     pos[threadID] = p;
}
```

$\left.\begin{array}{l} \\ \\ \\ \end{array}\right\}$ For each input vertex V_i

Control Point Evaluation

- Pre-compute stencils for each topology combination
- Each combination rendered in a separate pass:
  - Different topologies have different number of vertices
  - One constant buffer for each set of weights
- Total number of constants depends on number of topology combinations
- It’s important to minimize total number of topology combinations

Consistent Control Point Evaluation

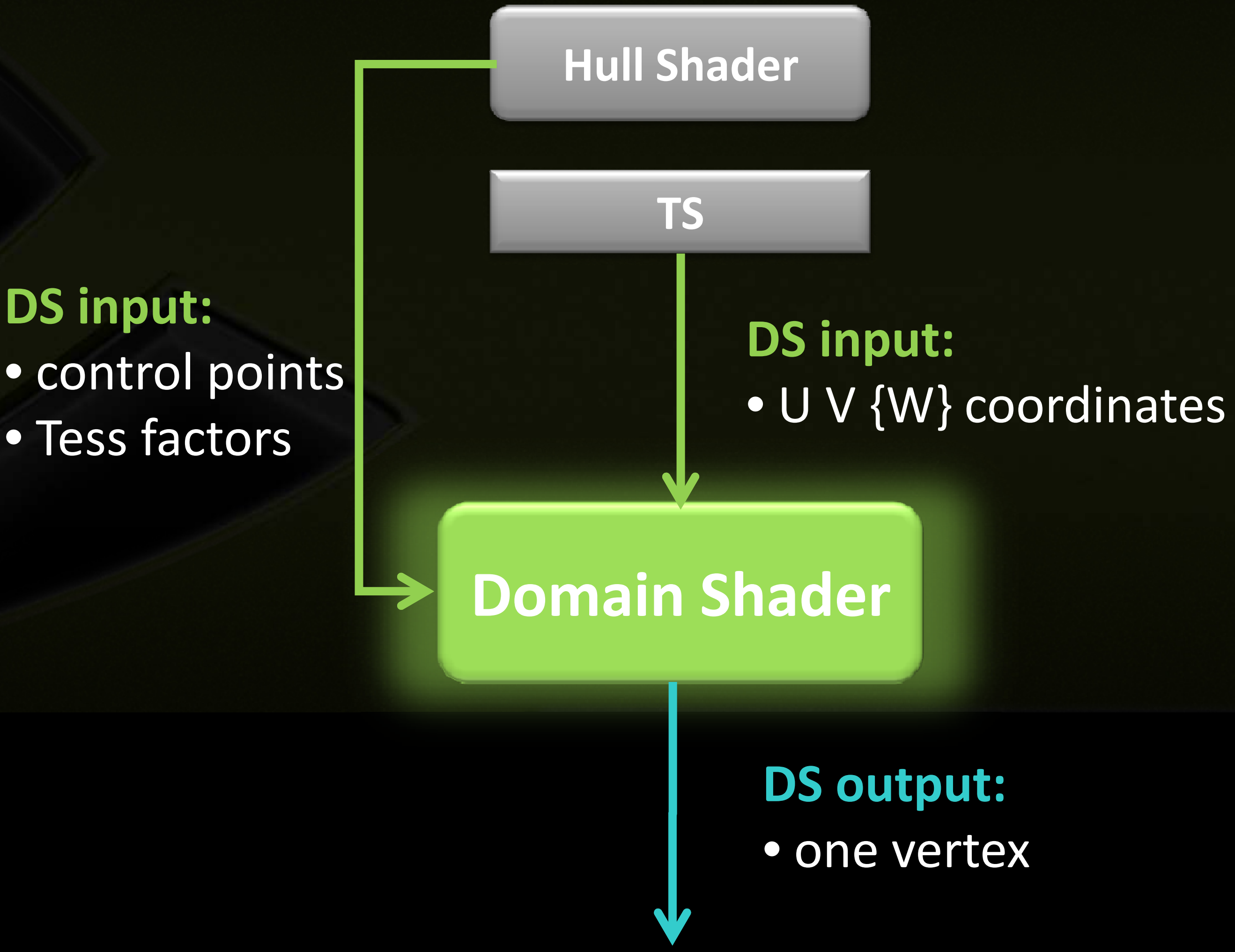
- Shared control points need to be evaluated "consistently" to avoid cracks in the mesh
  - Sum terms must be added in the same order
  - Define order globally, for example as "position of the vertex in the VB"
  - Use index array to map reordered vertices to stencils

Consistent Control Point Evaluation

```
uniform int vertexIndex[K];
global float w[K][16];
in float3 V[K];
out float3 pos[16];

void main() {
     float3 p = 0.0;
     for (int i = 0; i < K; i++) {
          int idx = vertexIndex[i];
          p += V[i] * w[idx][ThreadID];
     }
     pos[ThreadID] = p;
}
```

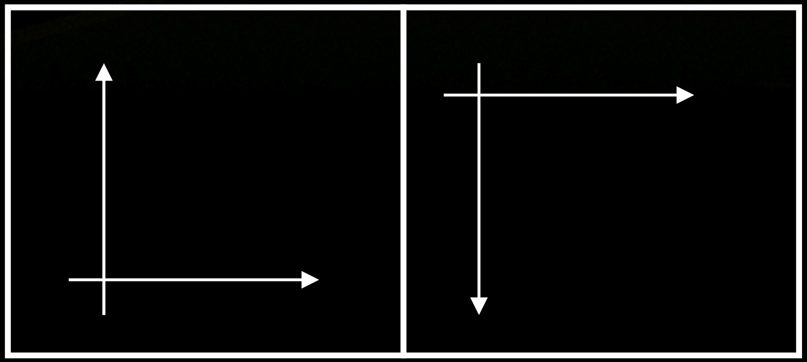
Domain Shader (DS)

- One invocation per generated vertex



Surface Evaluation

- DS shader simply evaluates the bicubic Bezier patch and the corresponding tangent fields
- Special care has to be taken to obtain watertight results (prevent cracks)
- All computations need to be symmetric along the patch edges



Floating Point Consistency

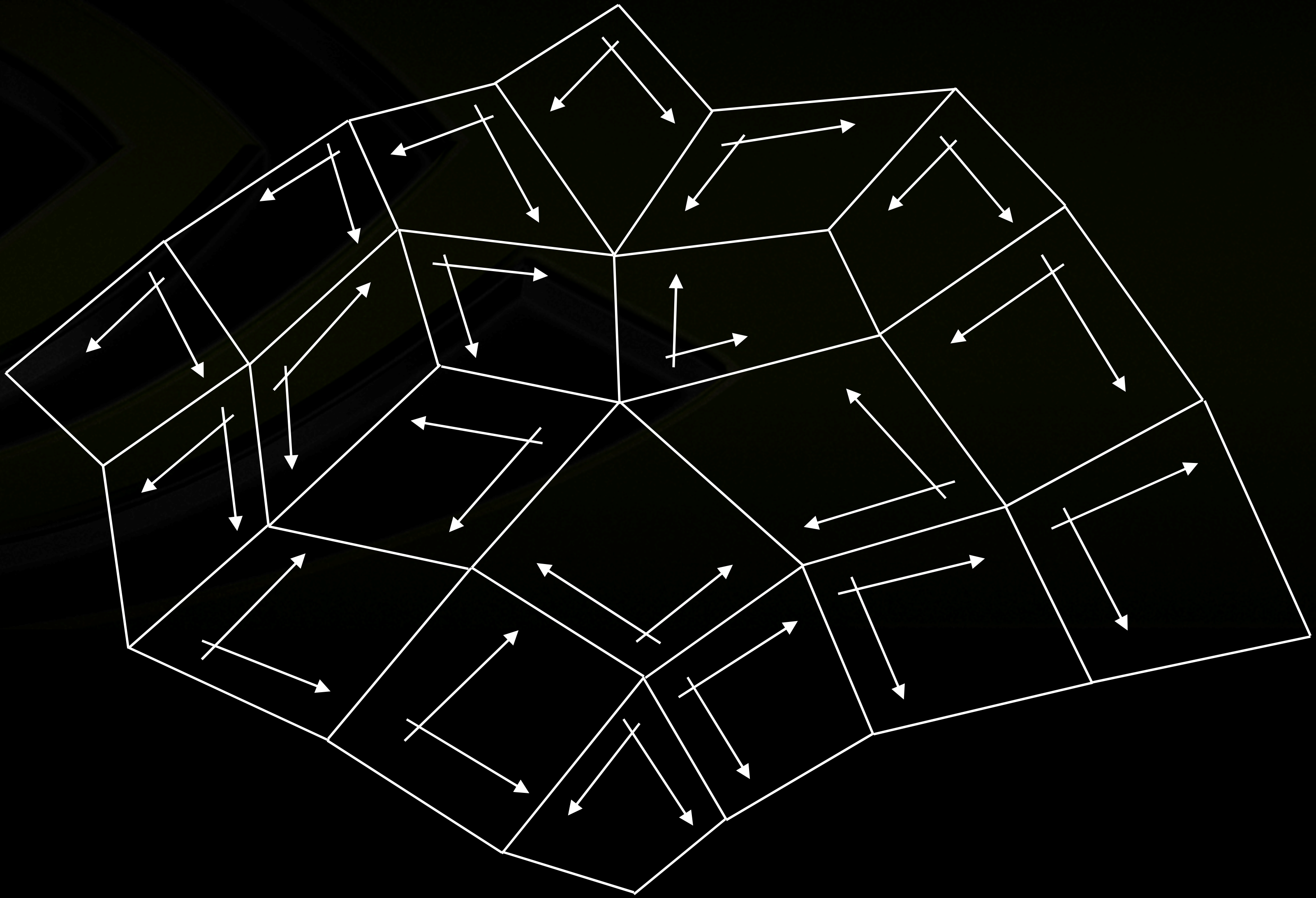
- FP addition is non commutative:
  - A + B + C + D != D + C + B + A
  - (A + B) + (C + D) == (D + C) + (B + A)
- FMA – fused multiple add (~= shader mad)
  - FMA(A,B,C) -> A+B*C
  - Not equivalent to separate MUL+ADD
    - A*a + B*b → FMA(A*a, B, b) != FMA(B*b, A, a)
- Beware of compiler optimizations
  - Use precise keyword

Surface Evaluation

- Two solutions
  - Reorder faces to make sure all edges have consistent orientations
  - Use symmetric evaluation on the boundaries

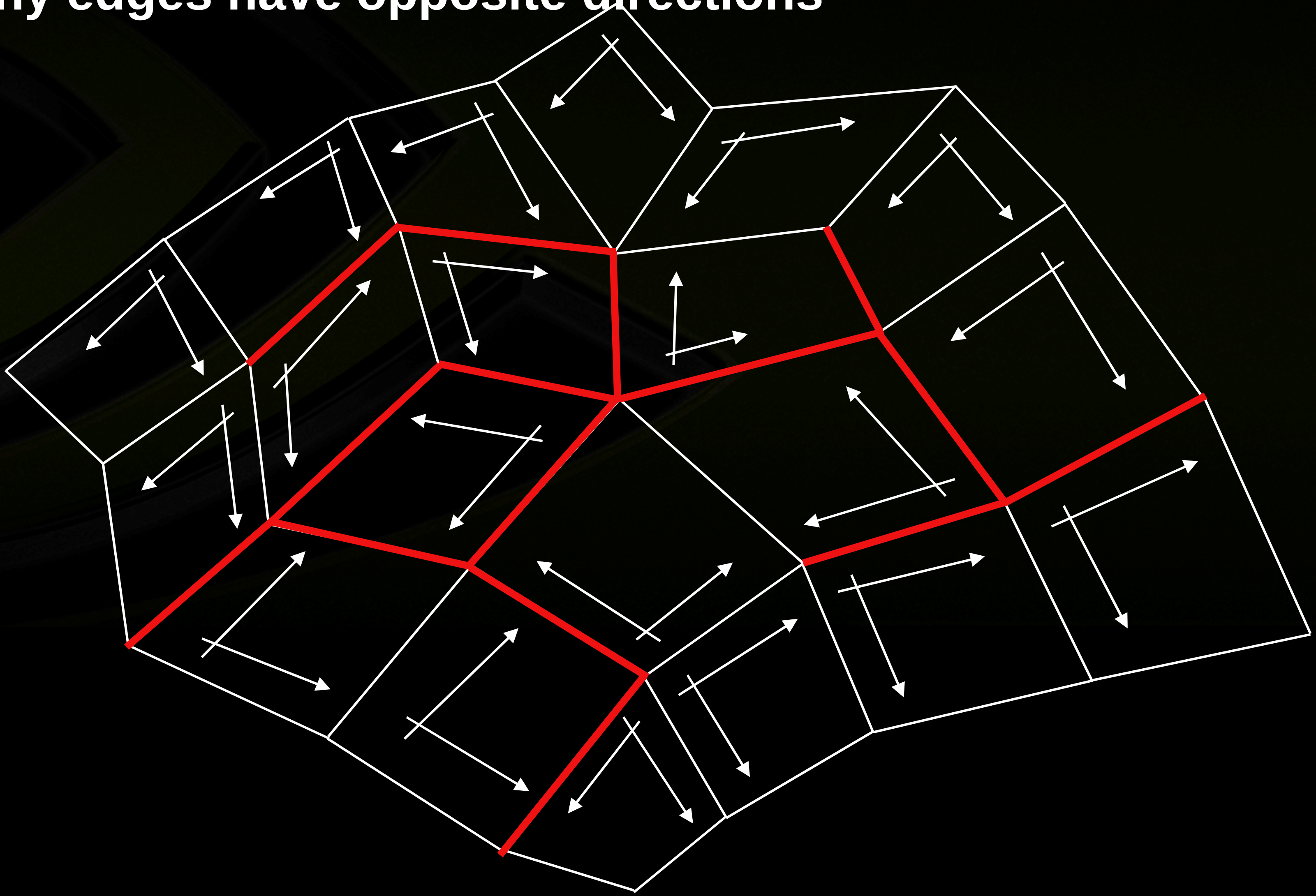
Surface Evaluation

- Faces generally have arbitrary orientations



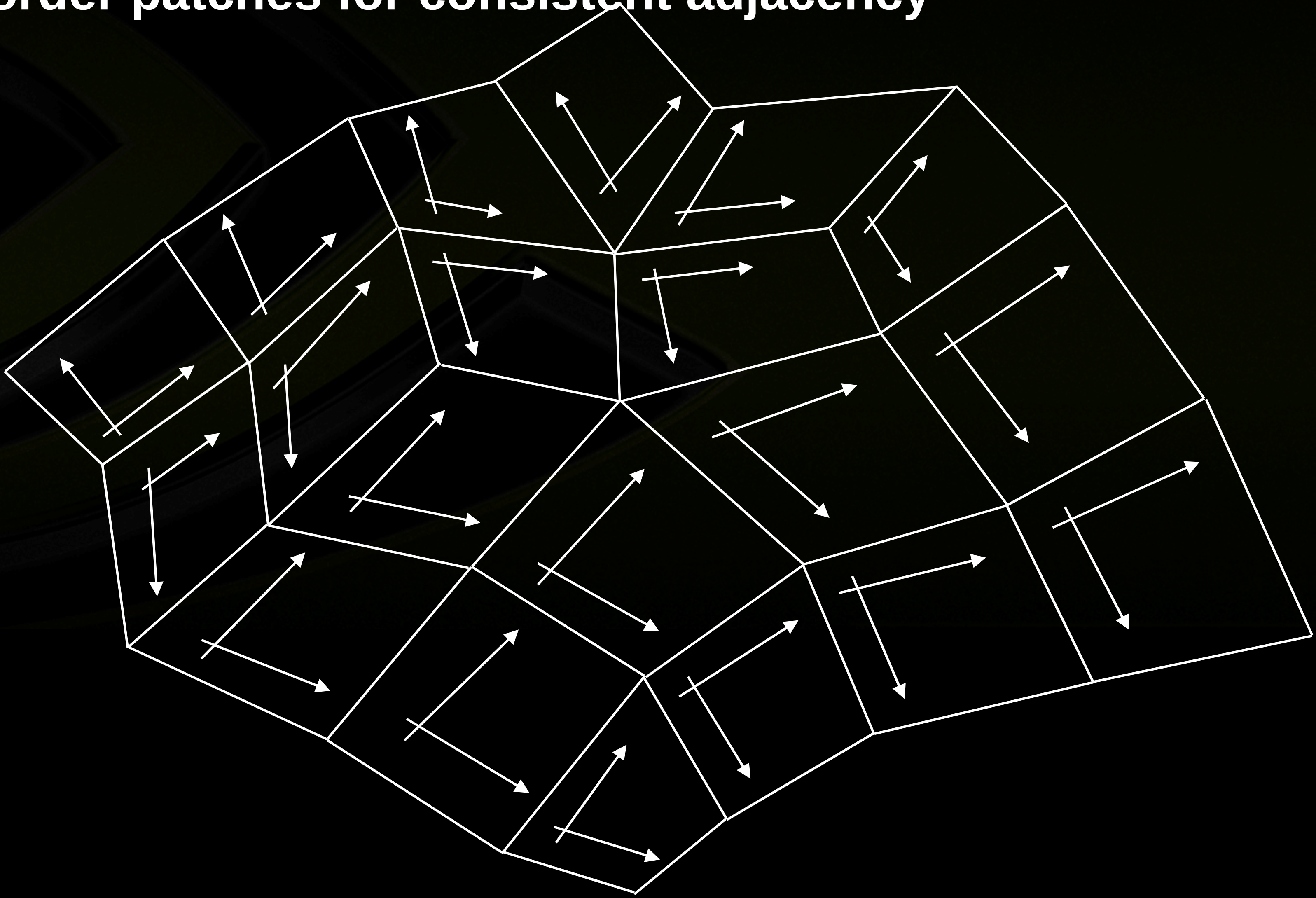
Surface Evaluation

- **Many edges have opposite directions**



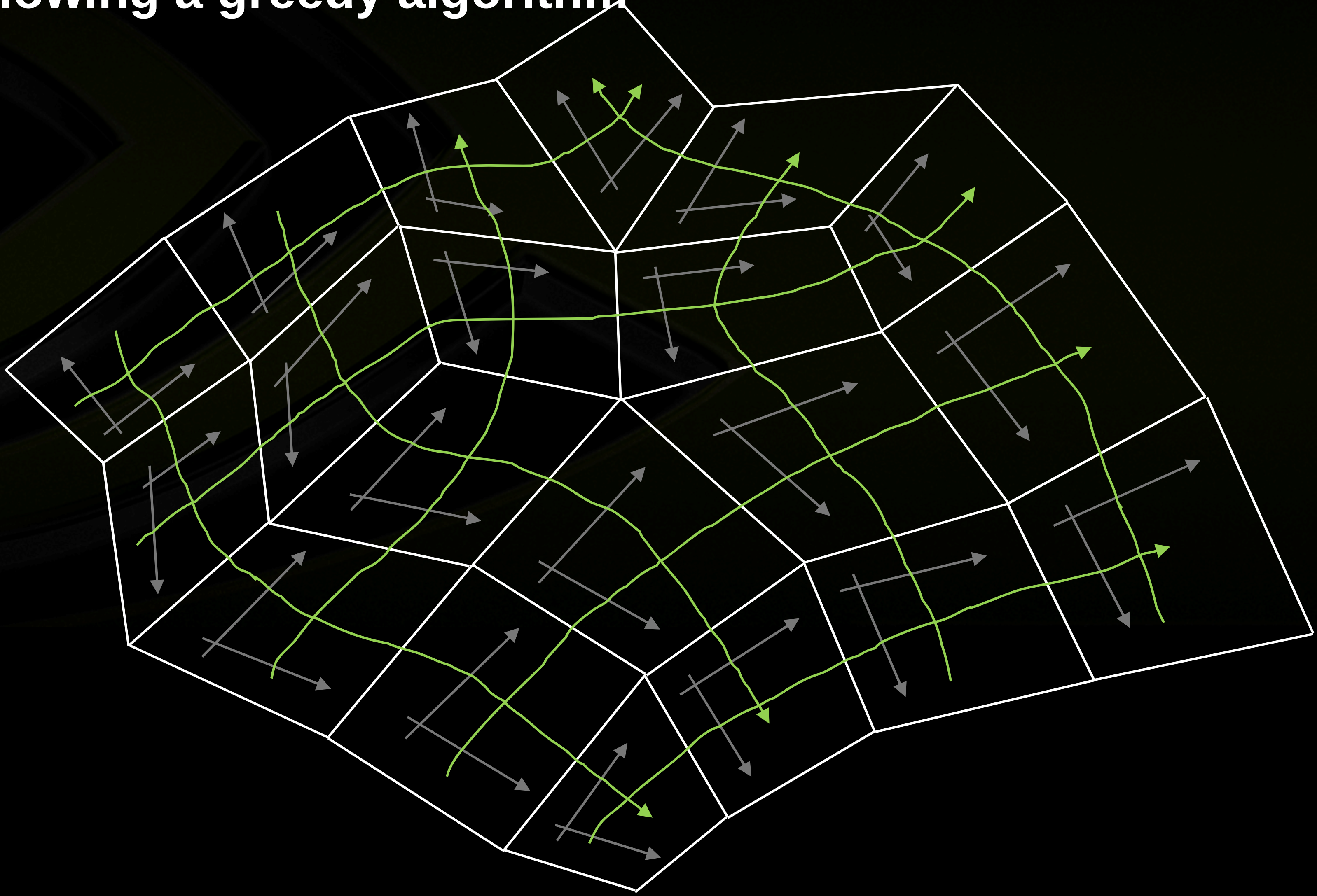
Surface Evaluation

- Reorder patches for consistent adjacency



Surface Evaluation

- Following a greedy algorithm



Bezier Evaluation

- Non-symmetric approach:

```
*--->--->--->     *
|   |   |   |     |      * = MUL
*--->--->--->     v
|   |   |   | --> |      + = ADD
*--->--->--->     v
|   |   |   |     |      > = FMA
*--->--->--->     v
```

- Evaluation order follows parametric direction
- 60 Instructions

Bezier Evaluation

- Non-symmetric approach:

```
*--->--->--->     *
|   |   |   |     |
*--->--->--->     v
|   |   |   | --> |
*--->--->--->     v
|   |   |   |     |
*--->--->--->     v
```

```
* = MUL

+ = ADD

> = FMA
```

- Evaluation order follows parametric direction
- 60 Instructions

Bezier Evaluation

- Non-symmetric approach:

```
*--->--->--->     *
|   |   |   |     |        * = MUL
*--->--->--->     v
|   |   |   | --> |        + = ADD
*--->--->--->     v
|   |   |   |     |        > = FMA
*--->--->--->     v
```

- Evaluation order follows parametric direction
- 60 Instructions

Bezier Evaluation

- Non-symmetric approach:

```
*--->--->--->      *
|   |   |   |      |
*--->--->--->      v
|   |   |   | -->  |
*--->--->--->      v
|   |   |   |      |
*--->--->--->      v
```

*	=	MUL
+	=	ADD
>	=	FMA

- Evaluation order follows parametric direction
- 60 Instructions

Bezier Evaluation

```
float3 bezierPosition(float2 uv, float3 p[16])
{
  float2 B0 =     (1 - uv) * (1 - uv) * (1 - uv);
  float2 B1 = 3 * (1 - uv) * (1 - uv) * (    uv);
  float2 B2 = 3 * (    uv) * (    uv) * (1 - uv);
  float2 B3 =     (    uv) * (    uv) * (    uv);

  return
    (B0.x * p[ 0] + B1.x * p[ 1] + B2.x * p[ 2] + B3.x * p[ 3]) * B0.y +
    (B0.x * p[ 4] + B1.x * p[ 5] + B2.x * p[ 6] + B3.x * p[ 7]) * B1.y +
    (B0.x * p[ 8] + B1.x * p[ 9] + B2.x * p[10] + B3.x * p[11]) * B2.y +
    (B0.x * p[12] + B1.x * p[13] + B2.x * p[14] + B3.x * p[15]) * B3.y;
}
```

Bezier Evaluation

- This approach only works for quad meshes
- Topology minimization constrains face orientations
- Symmetric evaluation is more flexible

Symmetric Bezier Evaluation

- Disable FMA for symmetric evaluation:

```
*-+-*-+-*-+-*     *
|   |   |   |     +
*-+-*-+-*-+-*     *
|   |   |   | --> +
*-+-*-+-*-+-*     *
|   |   |   |     +
*-+-*-+-*-+-*     *
```

```
* = MUL
+ = ADD
> = FMA
```

- 105 Instructions

Symmetric Bezier Evaluation

- Disable FMA for symmetric evaluation:

```
*-+-*-+-*-+-*     *
|   |   |   |     +          * = MUL
*-+-*-+-*-+-*     *
|   |   |   | --> +          + = ADD
*-+-*-+-*-+-*     *
|   |   |   |     +          > = FMA
*-+-*-+-*-+-*     *
```

- 105 Instructions

Symmetric Bezier Evaluation

- Disable FMA for symmetric evaluation:

```
*-+-*-+-*-+-*      *
|   |   |   |      +
*-+-*-+-*-+-*      *
|   |   |   | --> +
*-+-*-+-*-+-*      *
|   |   |   |      +
*-+-*-+-*-+-*      *
```

```
* = MUL

+ = ADD

> = FMA
```

- 105 Instructions

Symmetric Bezier Evaluation

- Use FMA, but still keep symmetry:

```
*--->-+-<---*      *
|   |   |   |      |
*--->-+-<---*      v
|   |   |   |  --> +
*--->-+-<---*      ^
|   |   |   |      |
*--->-+-<---*      *
```

```
* = MUL

+ = ADD

> = FMA
```

- 75 Instructions

Symmetric Bezier Evaluation

- Use FMA, but still keep symmetry:

```
*--->-+-<---*      *
|   |   |   |      |
*--->-+-<---*      v
|   |   |   | --> +
*--->-+-<---*      ^
|   |   |   |      |
*--->-+-<---*      *
```

```
*  =  MUL

+  =  ADD

>  =  FMA
```

- 75 Instructions

Symmetric Bezier Evaluation

- Use FMA, but still keep symmetry:

```
*--->-+-<---*     *
|   |   |   |     |
*--->-+-<---*     v
|   |   |   | --> +
*--->-+-<---*     ^
|   |   |   |     |
*--->-+-<---*     *
```

```
* = MUL
+ = ADD
> = FMA
```

- 75 Instructions

Symmetric Bezier Evaluation

- Symmetry only required on edges:

```
*--->-+-<---*     *
|   |   |   |     |
*--->--->--->     v
|   |   |   | --> +
*--->--->--->     ^
|   |   |   |     |
*--->-+-<---*     *
```

```
* = MUL

+ = ADD

> = FMA
```

- 69 Instructions

Symmetric Bezier Evaluation

- Use mad intrinsics

```
float3 w0 =
        mad(B0.x * p[ 0], B1.x, p[ 1]) +
        mad(B3.x * p[ 3], B2.x * p[ 2]);

float3 w1 = B0.x * p[ 4] + B1.x * p[ 5] + B2.x * p[ 6] + B3.x * p[ 7];
float3 w2 = B0.x * p[ 8] + B1.x * p[ 9] + B2.x * p[10] + B3.x * p[11];

float3 w3 =
        mad(B0.x * p[12], B1.x, p[13]) +
        mad(B3.x * p[15], B2.x * p[14]);

float p =
        mad(w0 * B0.y, w1, B1.y) +
        mad(w3 * B3.y, w2, B2.y);
```

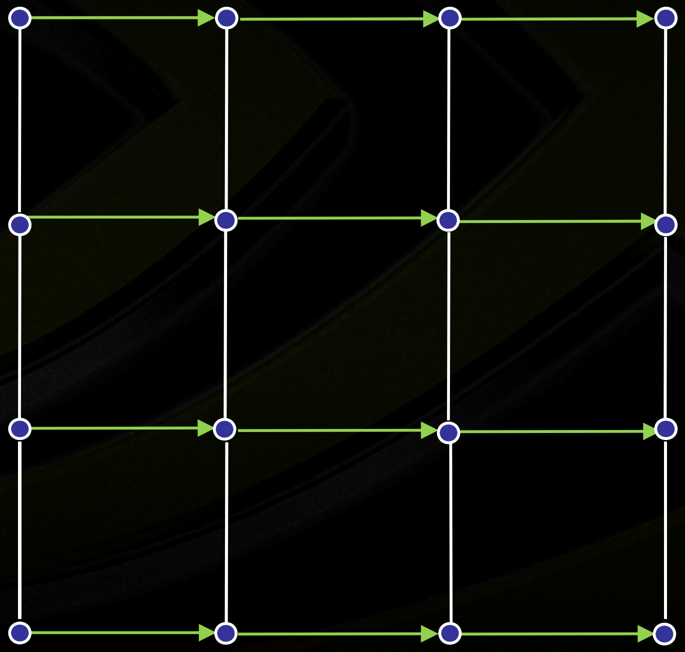
Symmetric Bezier Evaluation

- Use precise to prevent compiler optimizations

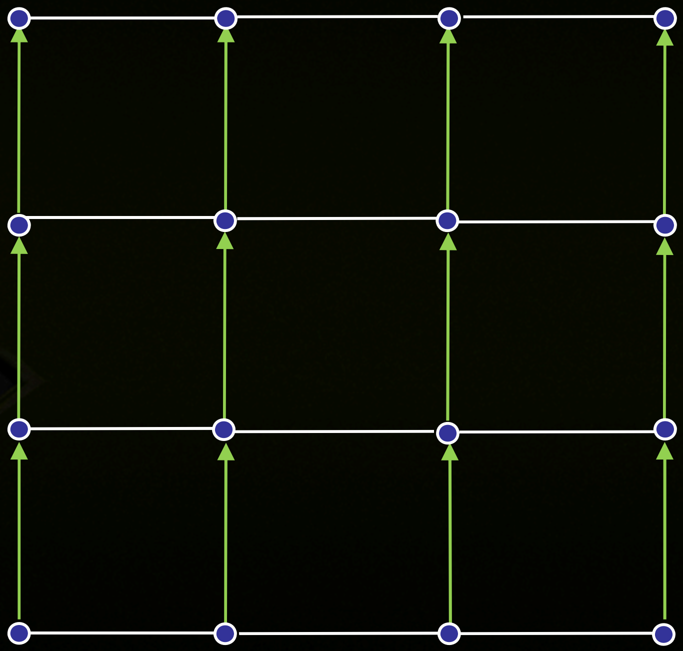
```
[precise] float3 w0 =
        mad(B0.x * p[ 0], B1.x, p[ 1]) +
        mad(B3.x * p[ 3], B2.x * p[ 2]);

float3 w1 = B0.x * p[ 4] + B1.x * p[ 5] + B2.x * p[ 6] + B3.x * p[ 7];
float3 w2 = B0.x * p[ 8] + B1.x * p[ 9] + B2.x * p[10] + B3.x * p[11];

[precise] float3 w3 =
        mad(B0.x * p[12], B1.x, p[13]) +
        mad(B3.x * p[15], B2.x * p[14]);

[precise] float p =
        mad(w0 * B0.y, w1, B1.y) +
        mad(w3 * B3.y, w2, B2.y);
```

Bezier Patches

- **Problem with Bezier Patches:**



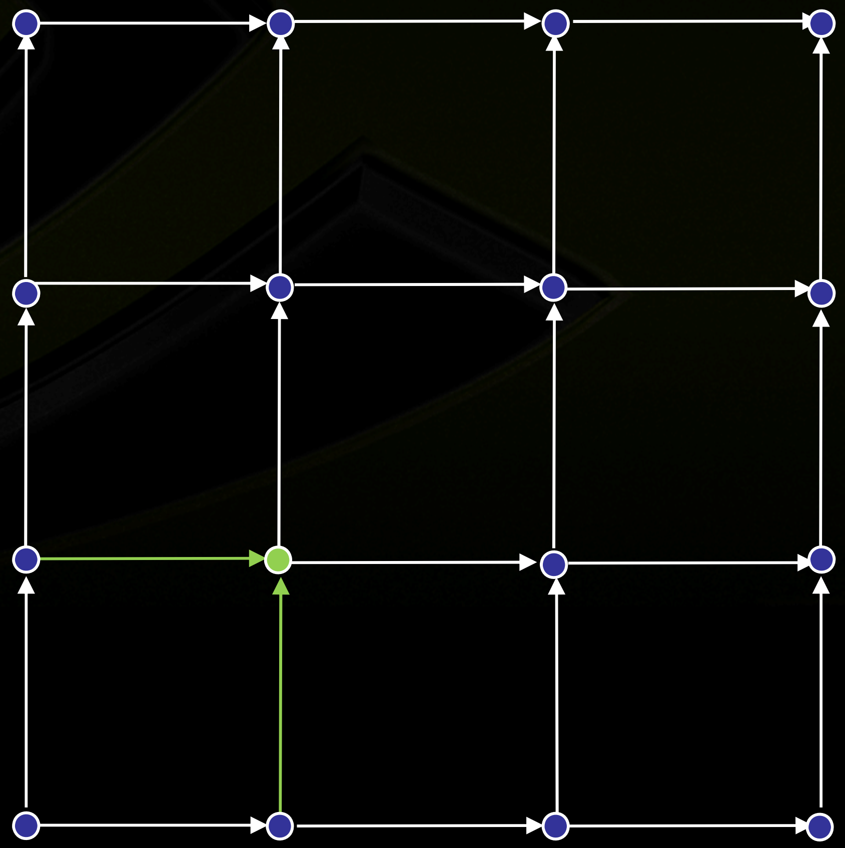
$$\frac{\partial f(u,v)}{\partial u}$$



$$\frac{\partial f(u,v)}{\partial v}$$

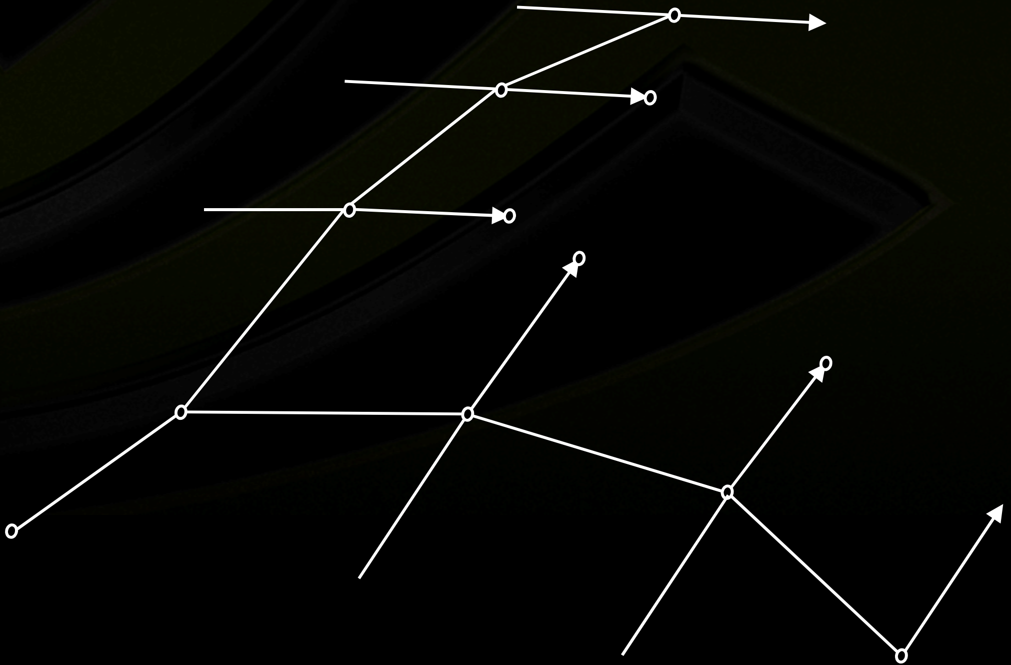
Bezier Patches

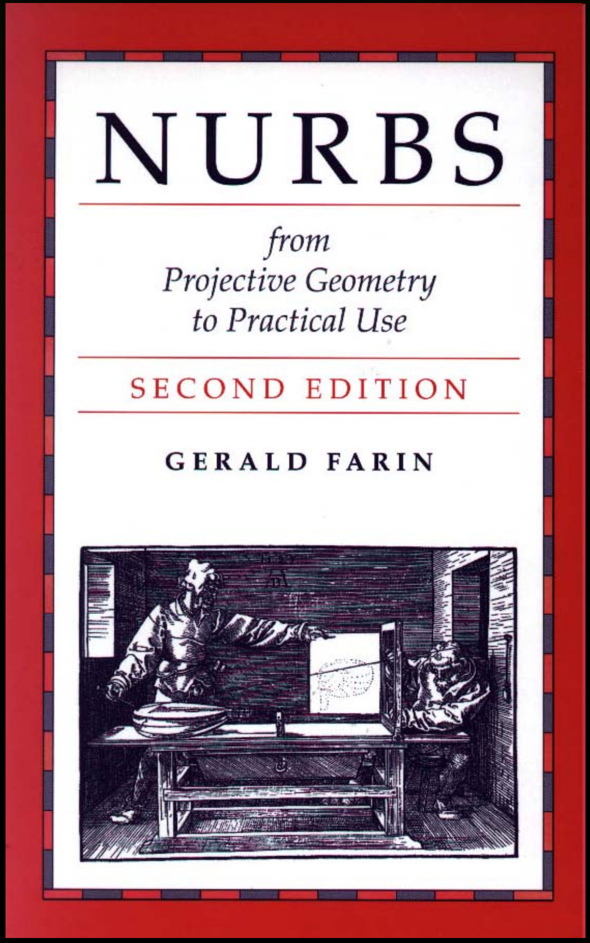
- Derivatives along the edges cannot be specified independently



Gregory Patches

- **With Bezier patches it's not possible to obtain C1 continuity across all boundaries**



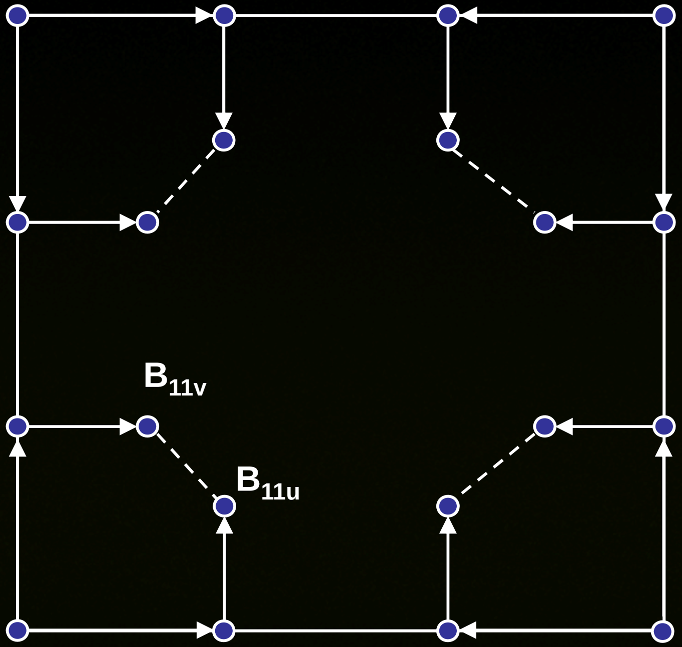


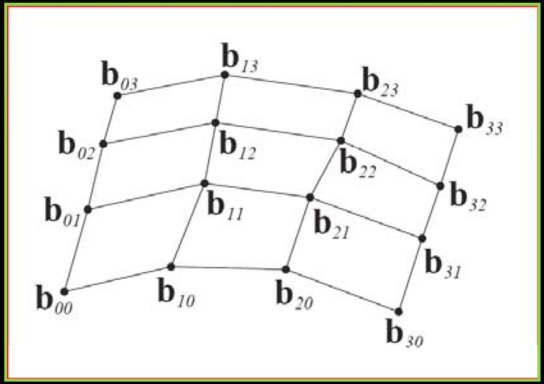
Gregory Patches

- 20 control points instead of 16
- Evaluated like Bezier where interior control point is computed as:

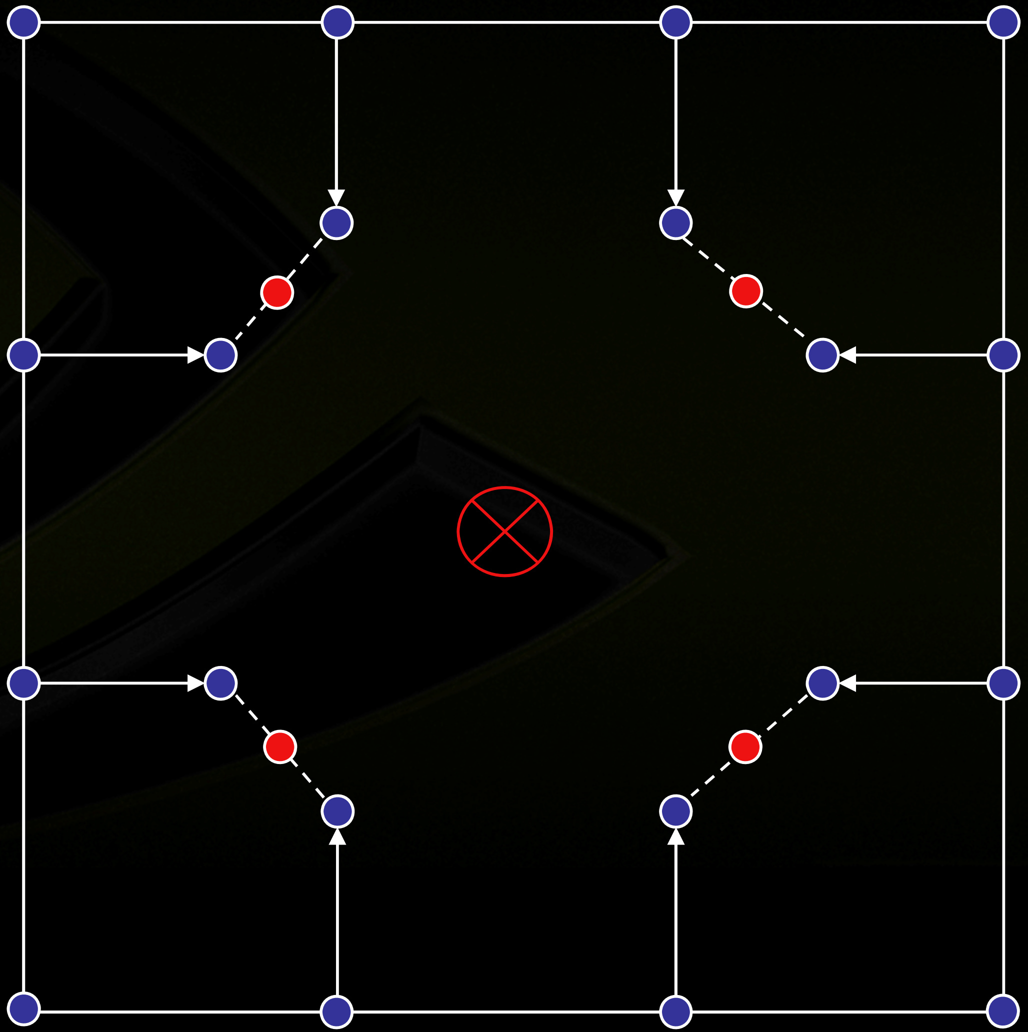
$$b_{11}(u,v)=\frac{u\,b_{11u}+v\,b_{11v}}{u+v}$$

- On regular faces Gregory patch becomes a Bezier patch

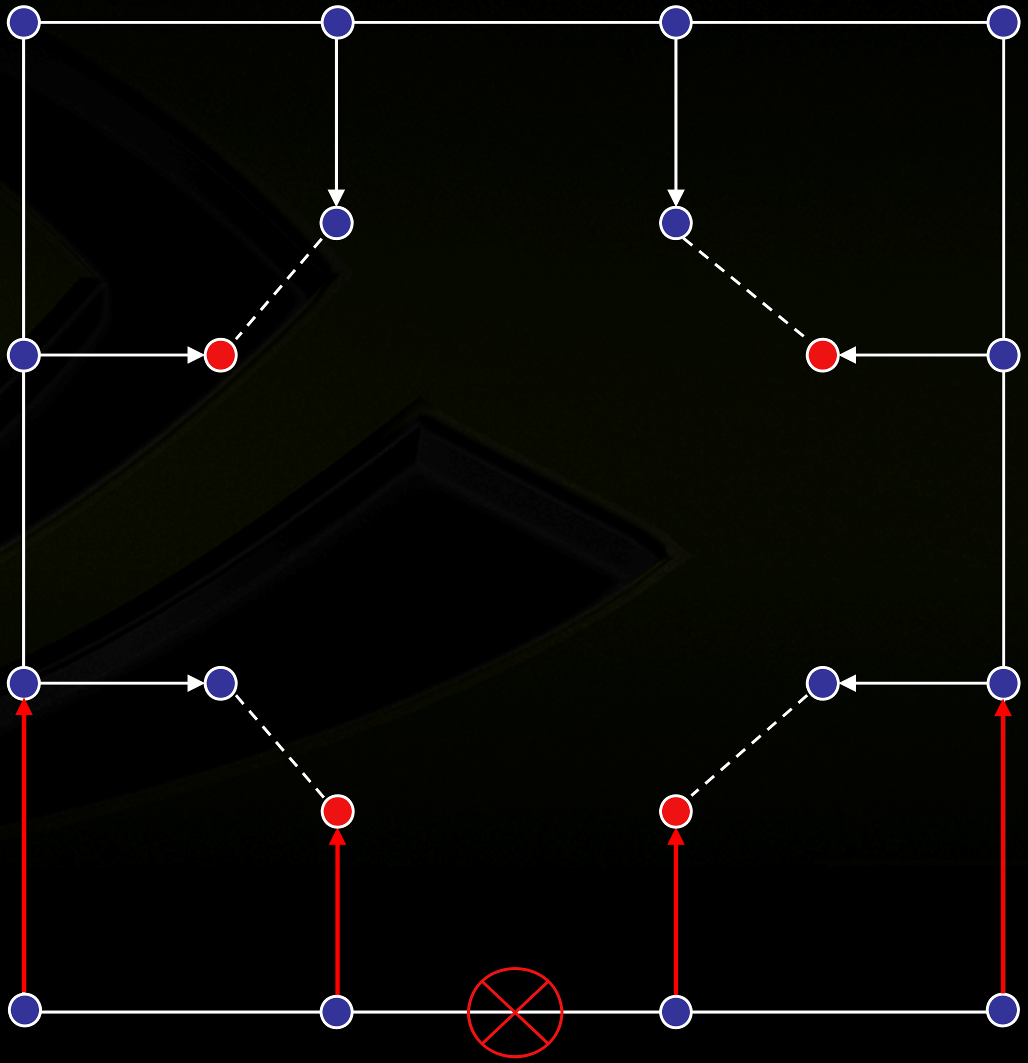




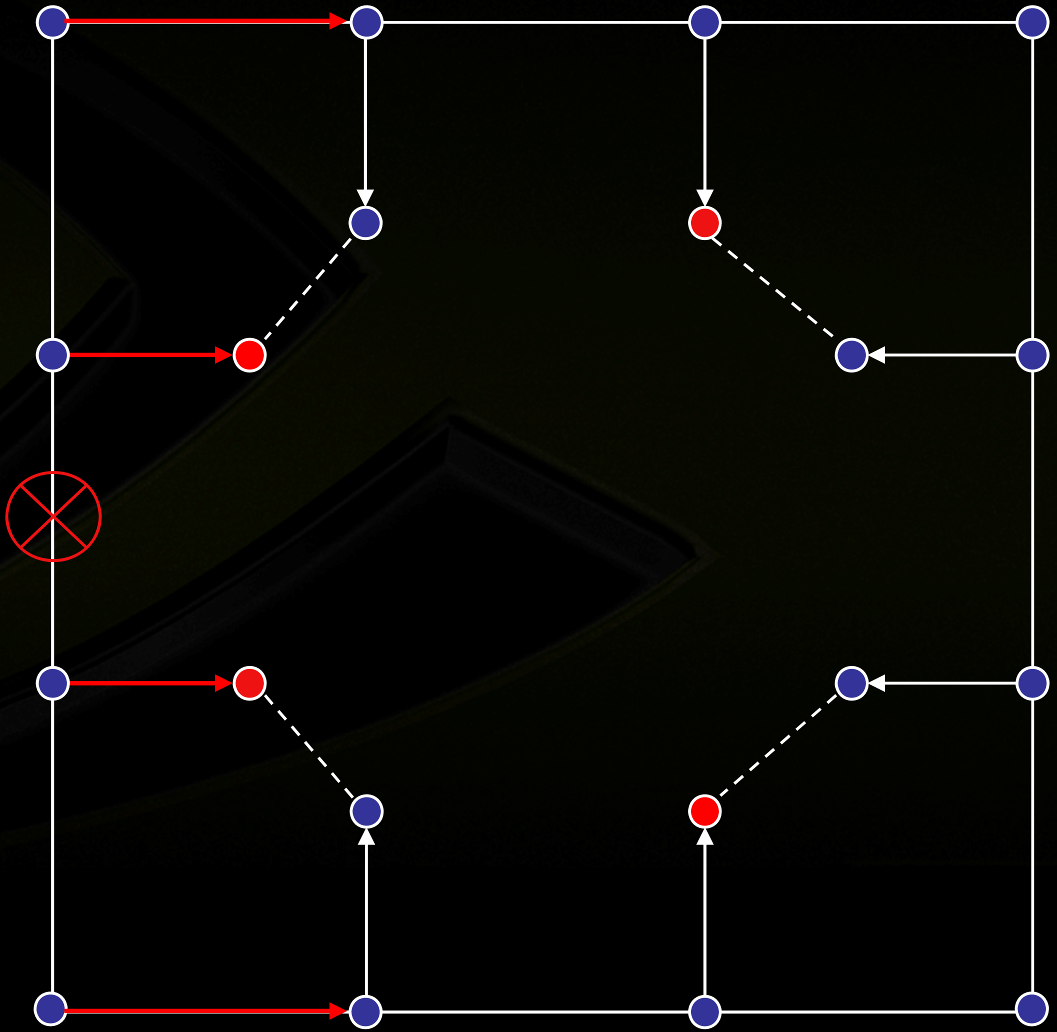
Gregory Patches



Gregory Patches

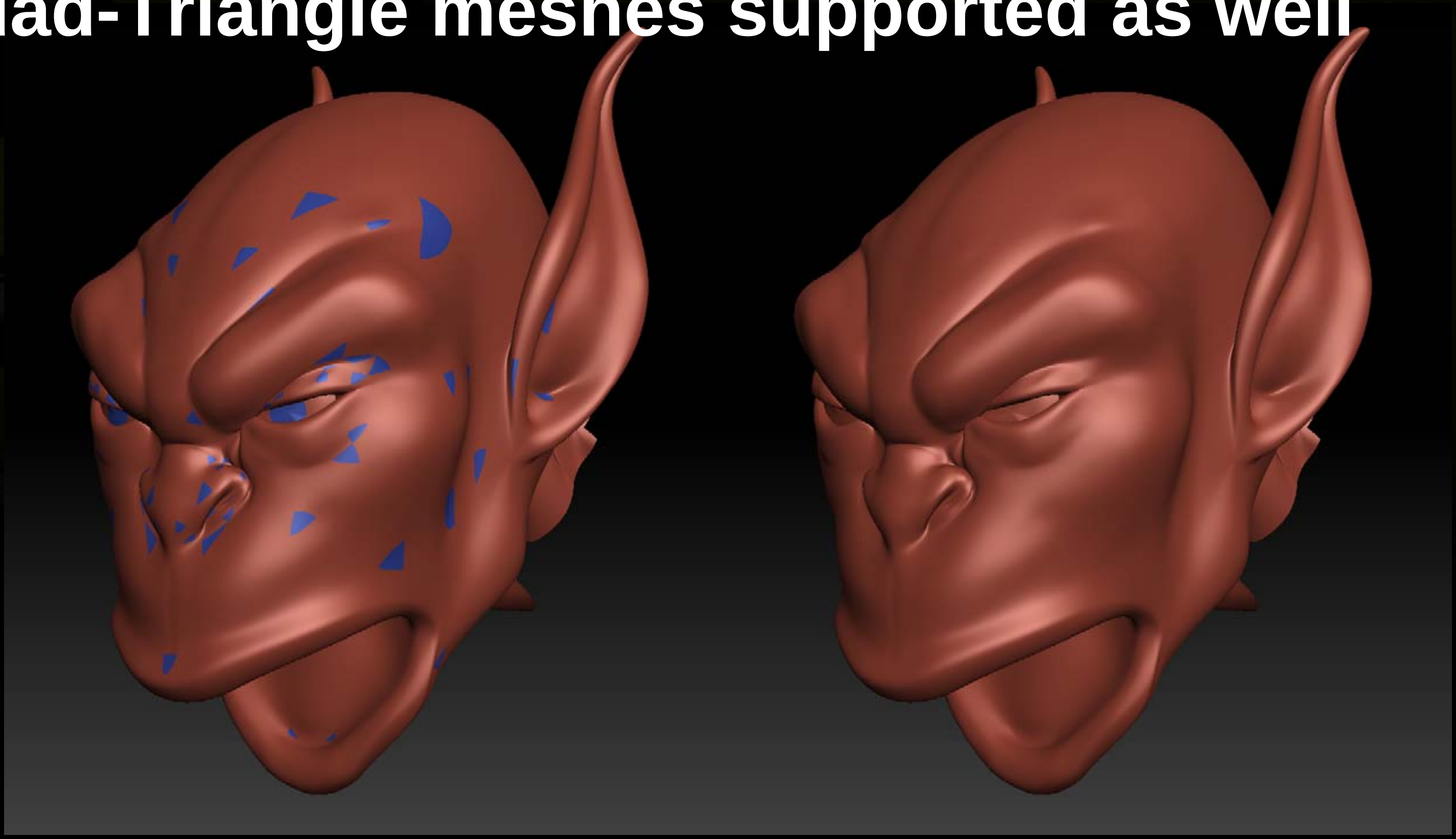


Gregory Patches



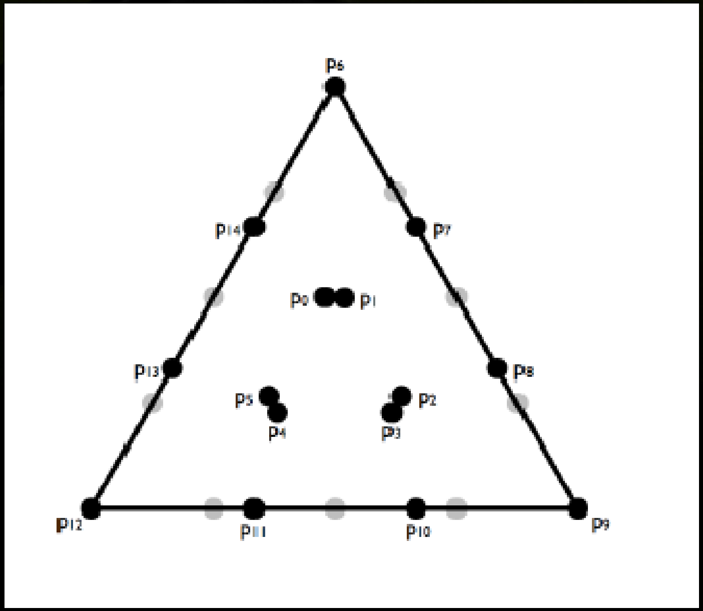
Gregory ACC

- Extended to triangular patches by Denis Kovacs
- Quad-Triangle meshes supported as well



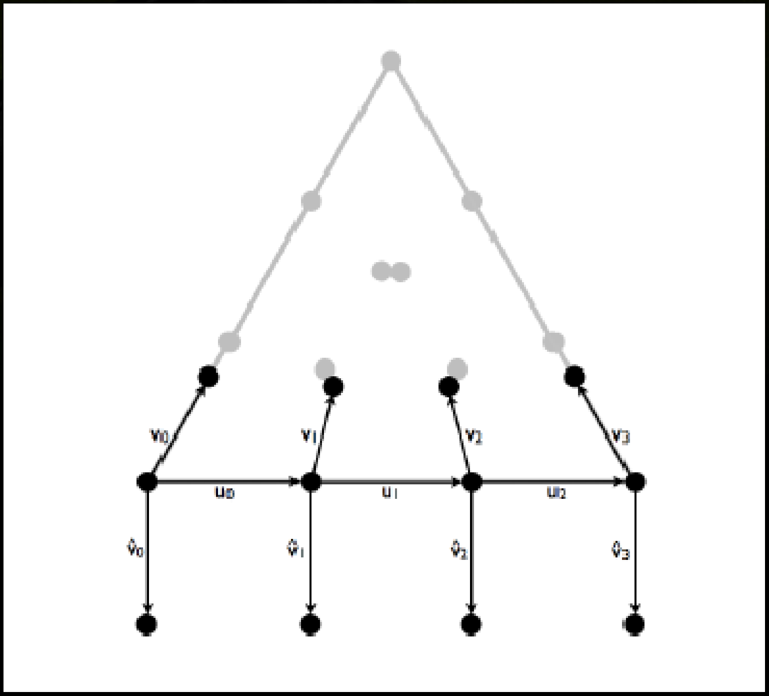
Gregory ACC

- Cubic triangle patches do not have enough degrees of freedom (only one interior control point)
- Use quartic patches with cubic boundaries



Gregory ACC

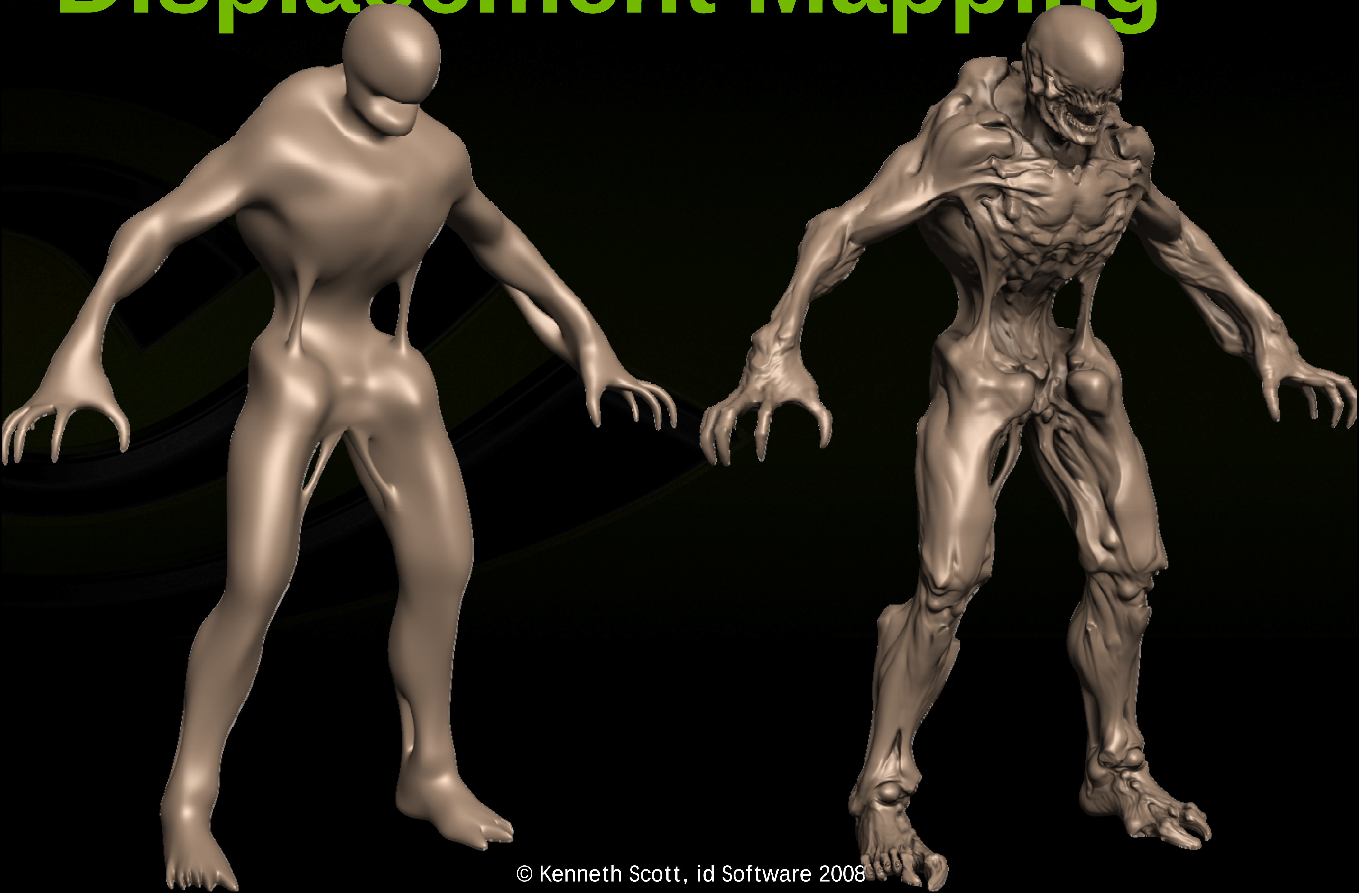
- Duplicate interior control points to satisfy the compatibility condition
- Blended according to parametric distance to edge



Other schemes

- Approximate polar subdivision
  - Prevent ripples around high-valent vertices
- C-Patches & P-Patches
  - More degrees of freedom:
    - C2 continuity
    - Sharpness adjustment
    - Triangles, Quads, and Pents
  - Evaluation is only slightly more expensive

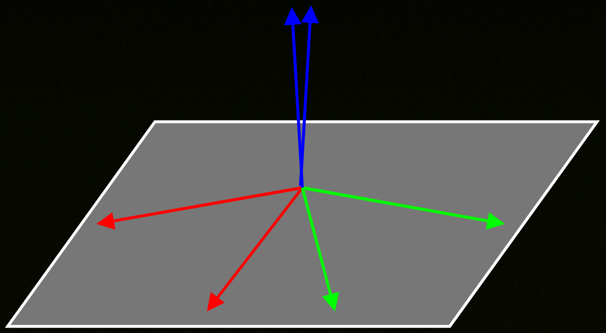
Displacement Mapping



© Kenneth Scott, id Software 2008

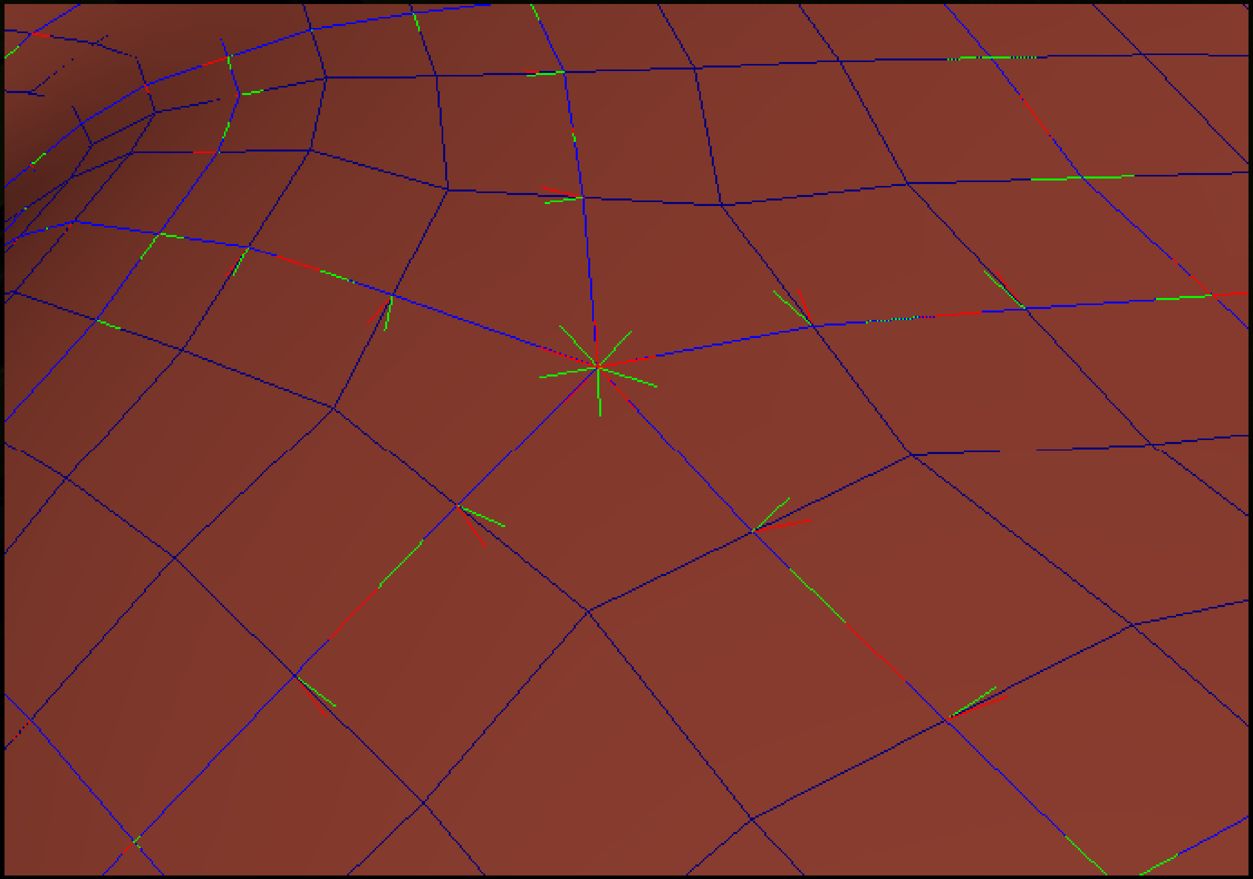
Displacement Maps

- Crack-free displacement maps
    - Consistent normal evaluation
    - Watertight texture sampling



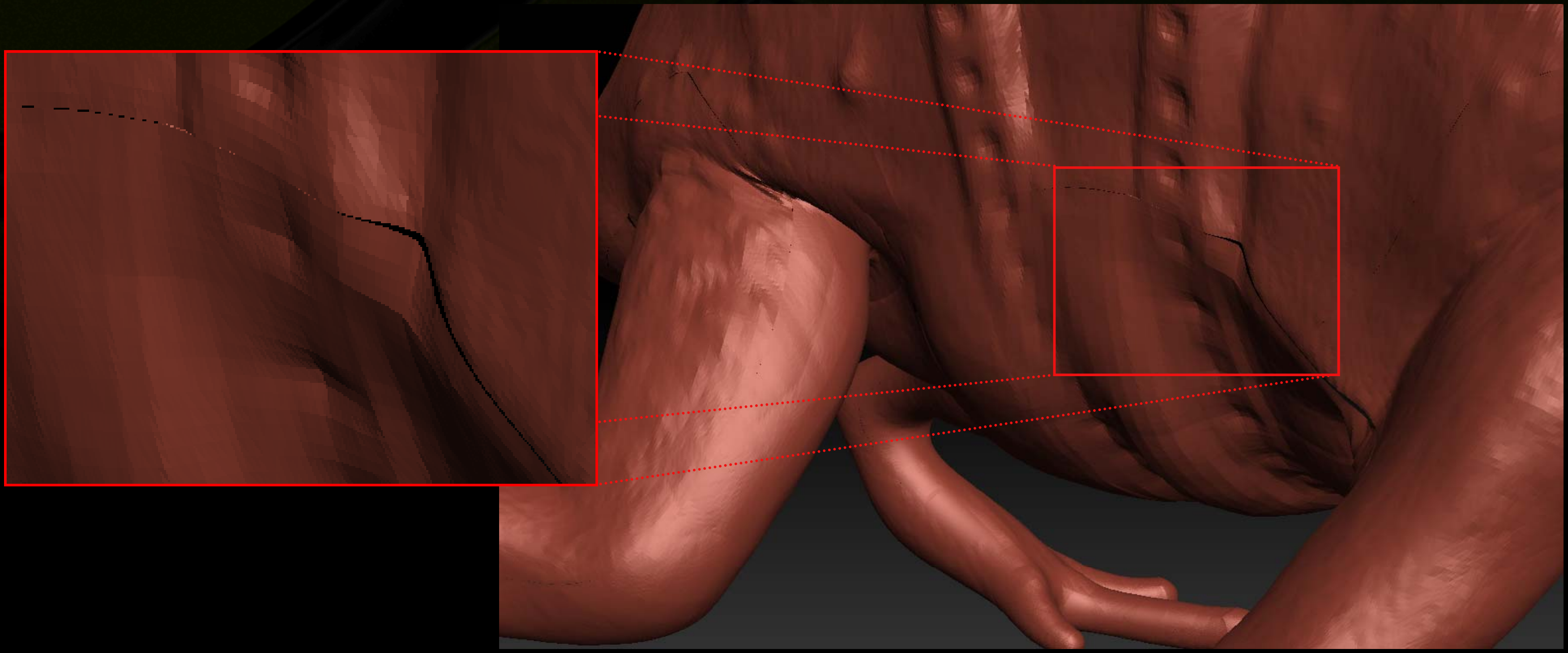
Consistent Normal Evaluation

- Control tangents along edges are not symmetric:



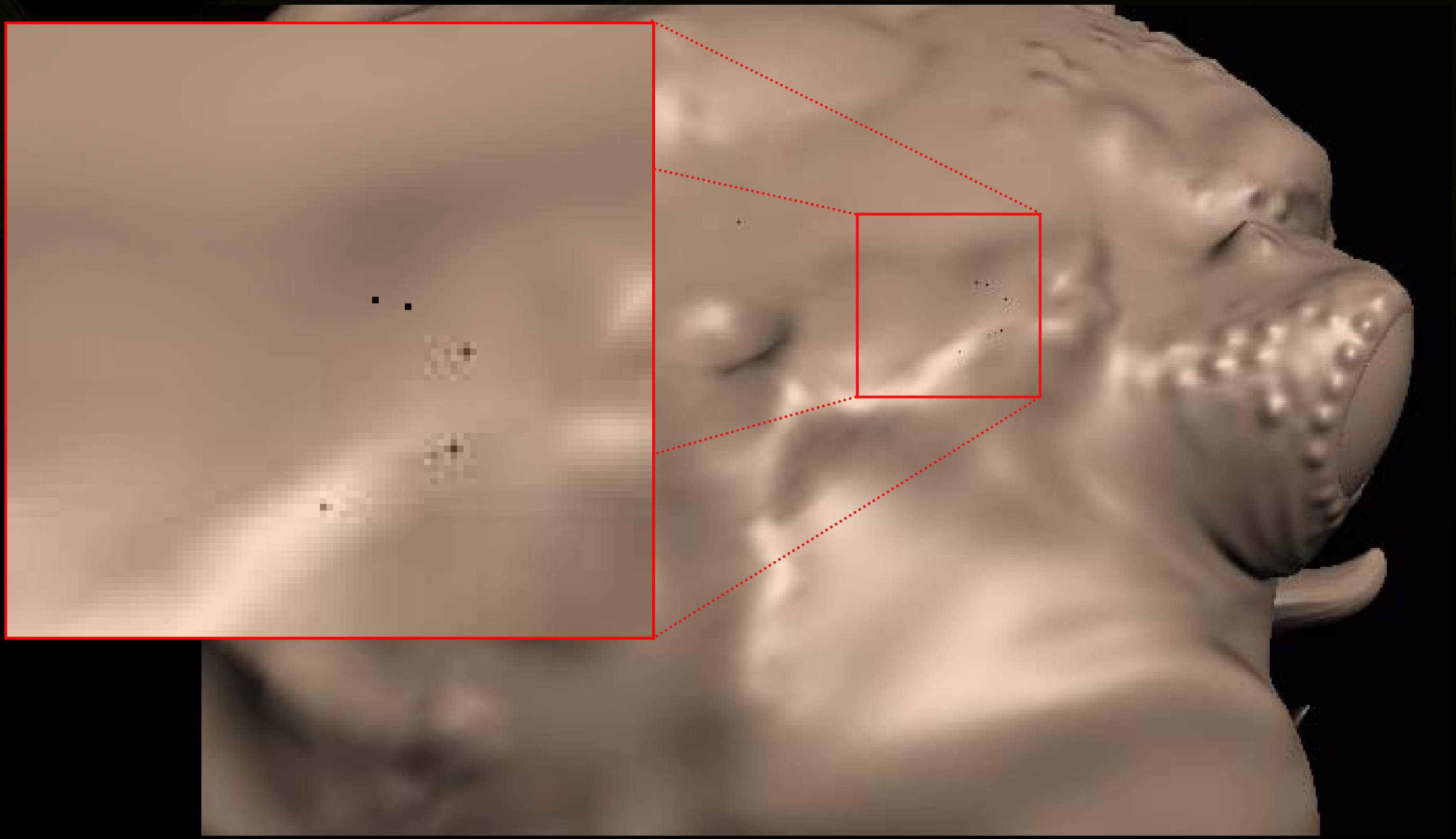
Watertight Texture Sampling

- **Texture seams cause holes in the mesh!**
  - **Due to bilinear discontinuities**
  - **Varying floating point precision on different regions of the texture map**



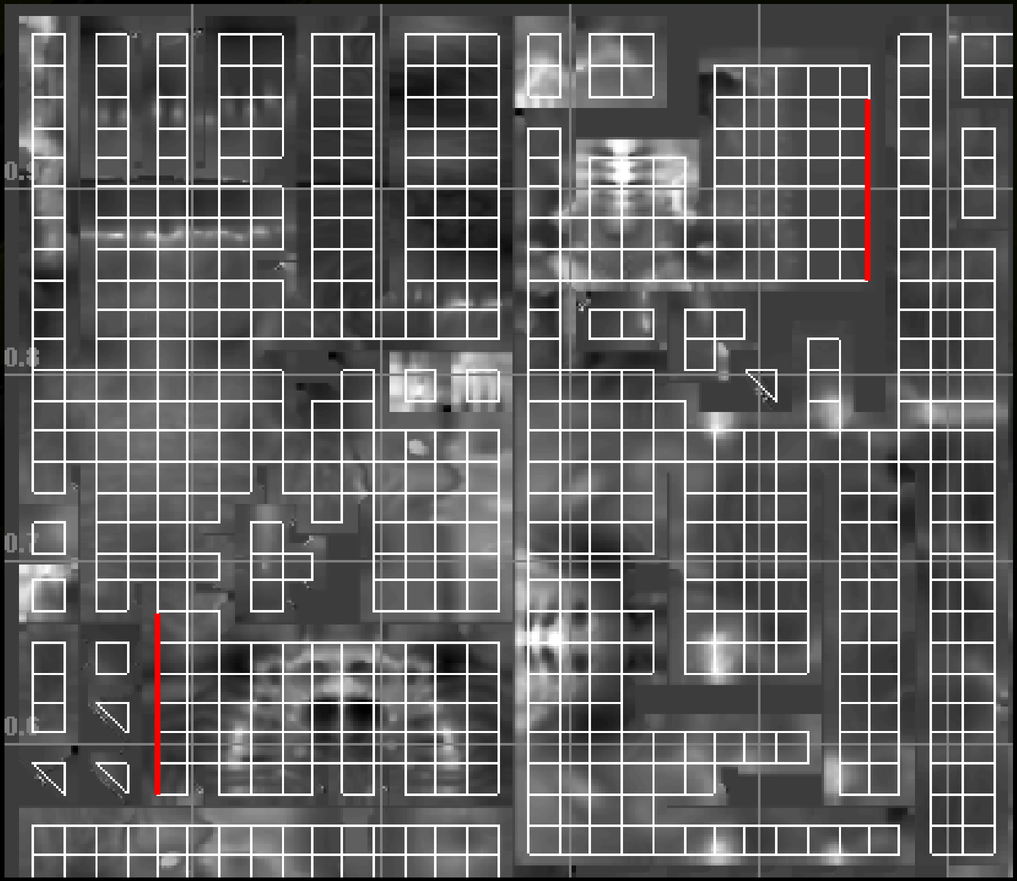
Watertight Texture Sampling

- Seamless parameterizations remove bilinear artifacts, but do not solve floating point precision issues



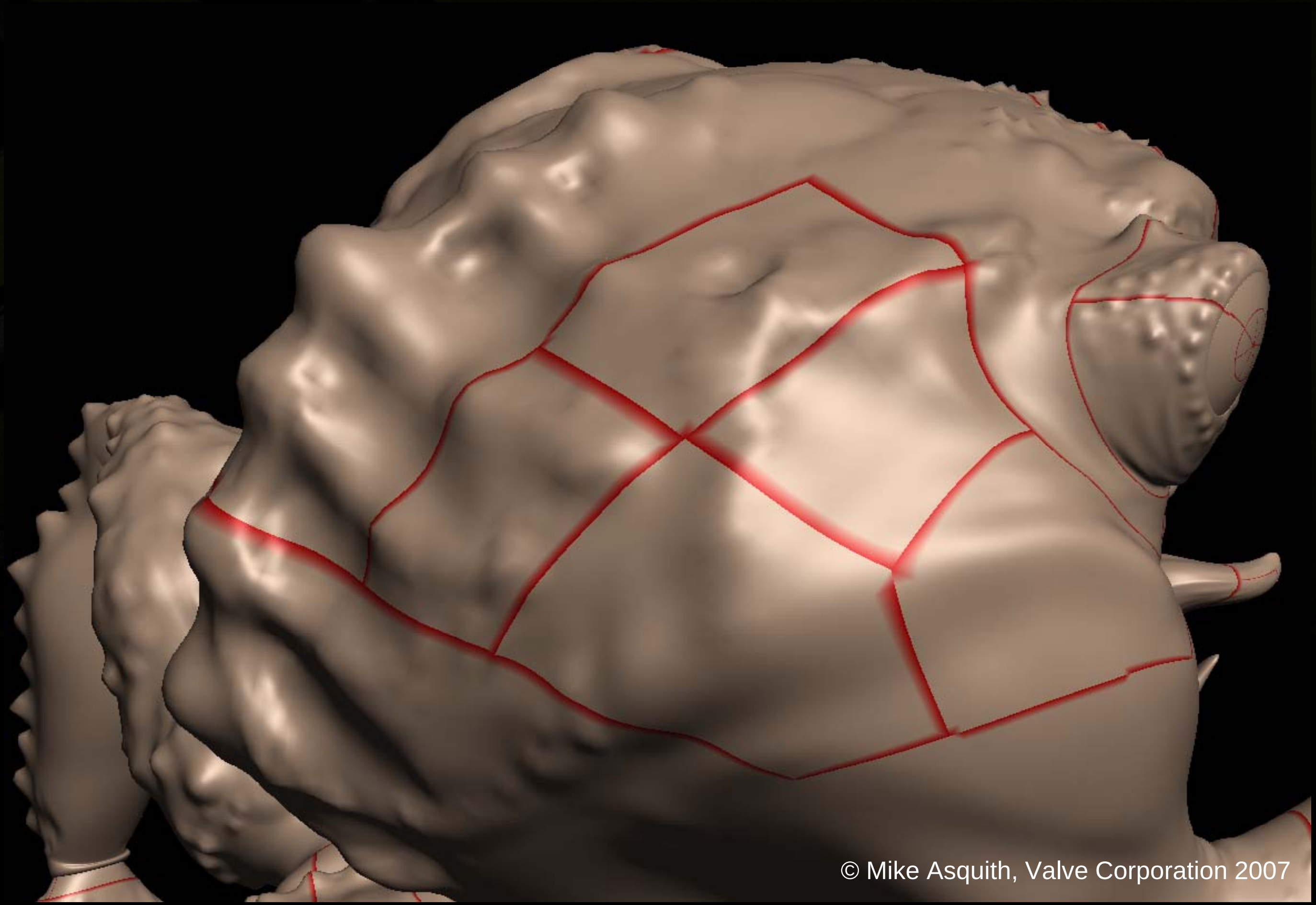
Watertight Texture Sampling

- **Texture coordinate interpolation yields different result depending on location of the seam edges:**



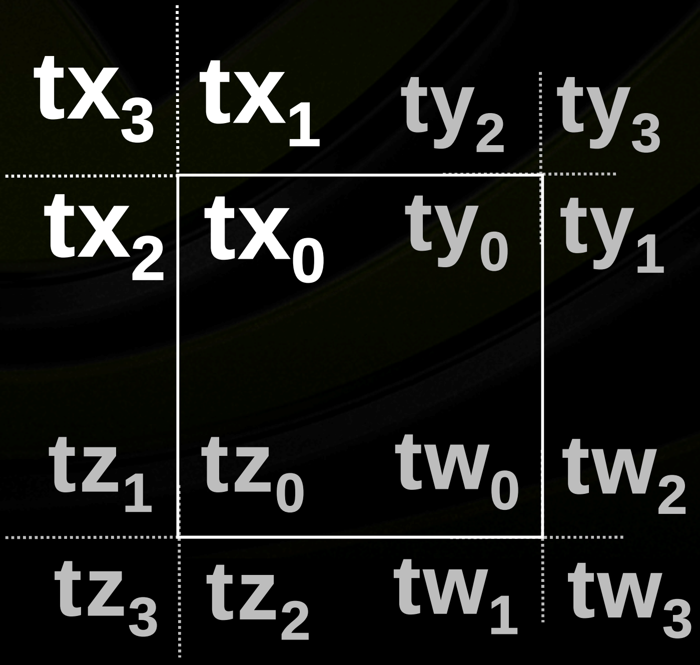
Watertight Texture Sampling

- **Solution: define edge and corner ownership**



Watertight Texture Sampling

- Store 4 texture coordinates per vertex
  - 16 per patch



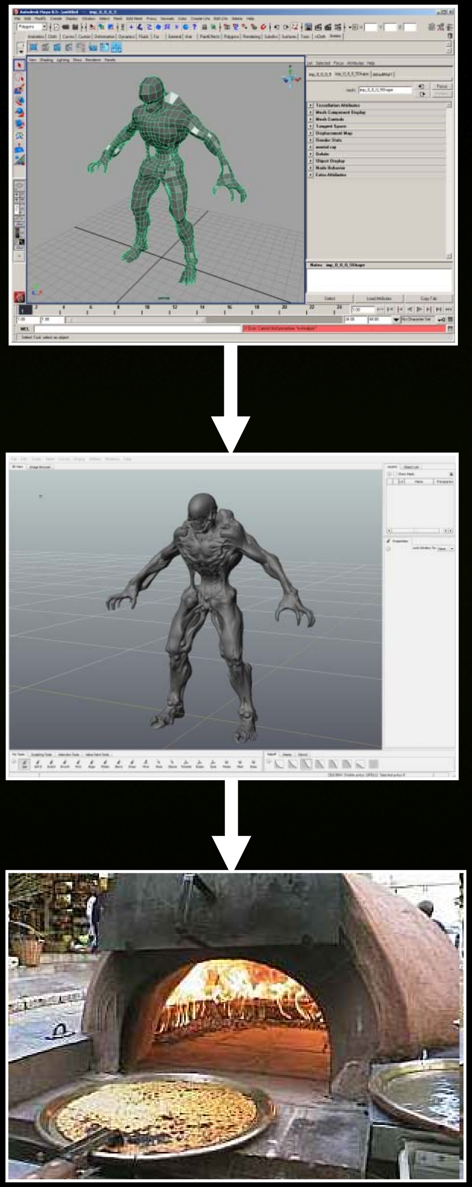
```
// float2 tx[4], ty[4], tz[4], tw[4];

int ix = 2 * (uv.x == 1) + (uv.y == 1);
int iy = 2 * (uv.y == 1) + (uv.x == 0);
int iz = 2 * (uv.x == 0) + (uv.y == 0);
int iw = 2 * (uv.y == 0) + (uv.x == 1);

float2 tc = w.x * tx[ix] +
            w.y * ty[iy] +
            w.z * tz[iz] +
            w.w * tw[iw];
```

Content Creation

© Keneth Scott, id Software 2008

Production Pipeline

- Modeling Tools
  - Base surface
- Sculpting Tools
  - Detailed mesh
- Baker Tools
  - Normal, displacement, occlusion, and other maps



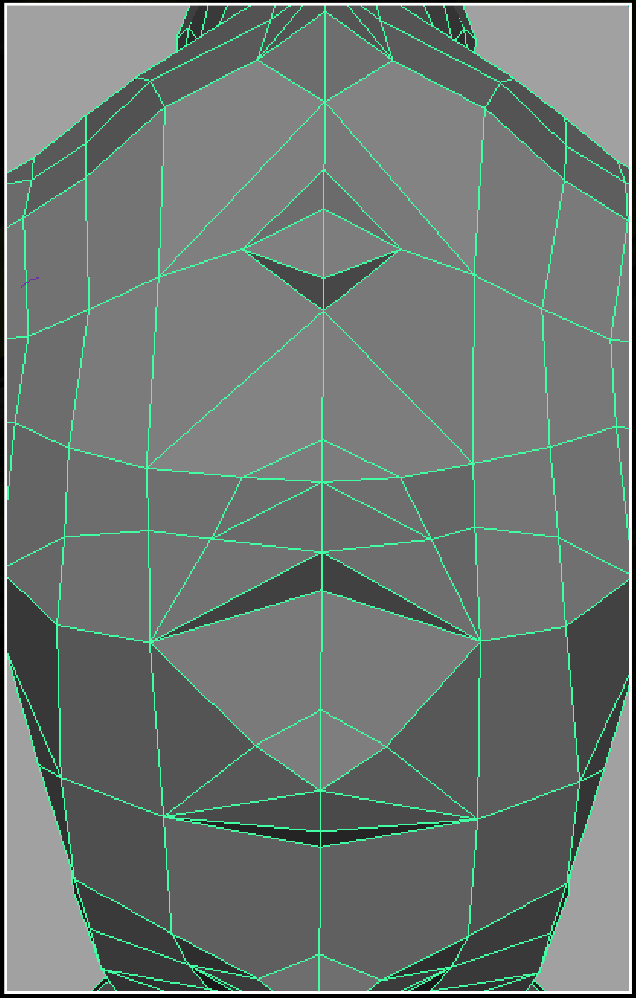
Modeling

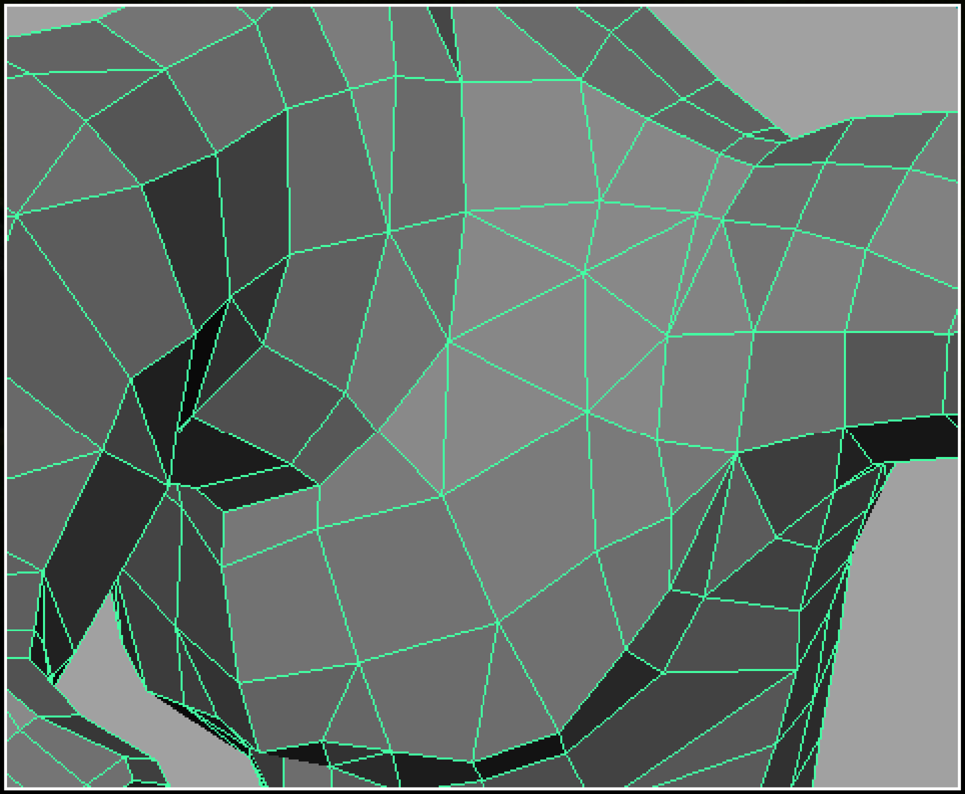
- Performance depends on number of topology combinations
- Optimization guidelines:
  - Eliminate triangles (Quad only meshes)
  - Close holes (Avoid open meshes)
  - Reduce number of extraordinary vertices
  - Decrease number of patches to the minimum

  - Try to create uniform, regular meshes

Topology Optimization

- **105 topology combinations**

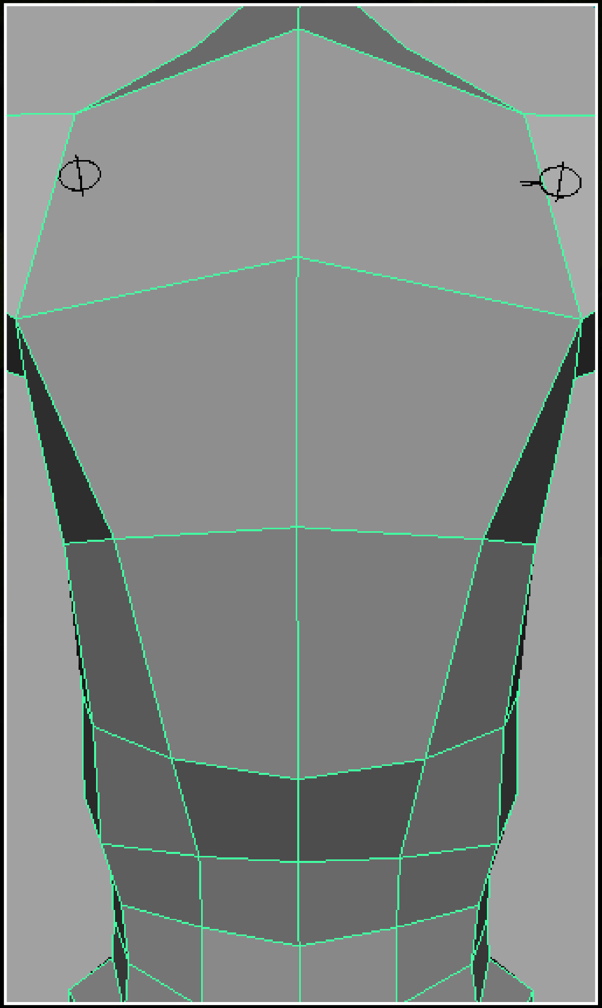


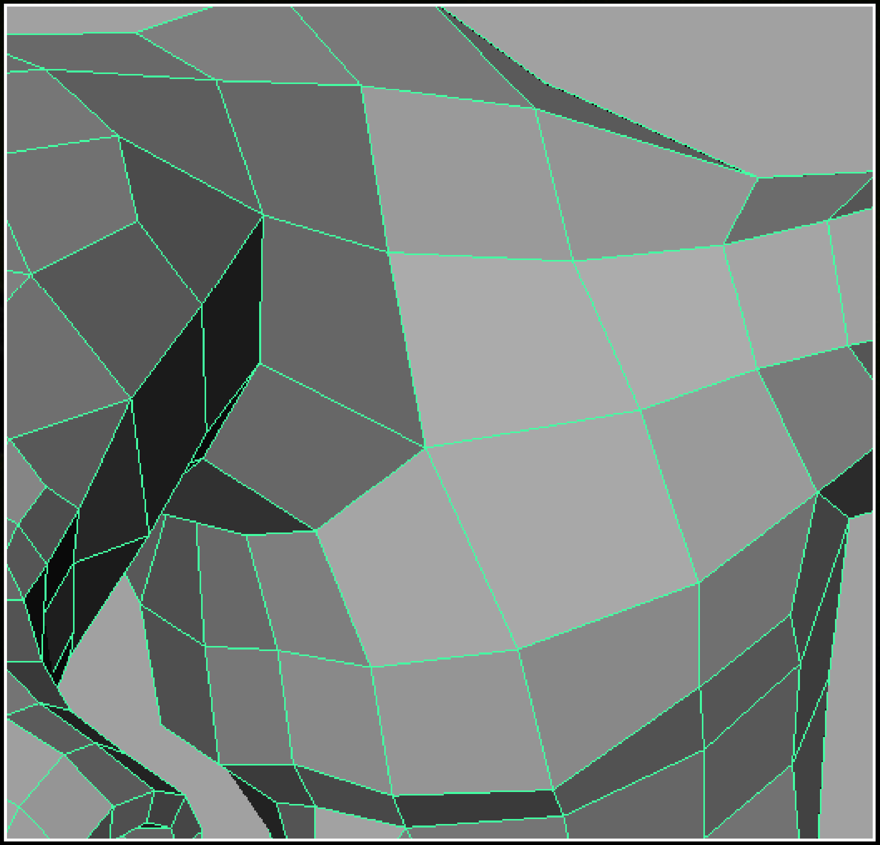


© Mike Asquith, Valve Corporation 2007

Topology Optimization

- **23 topology combinations**

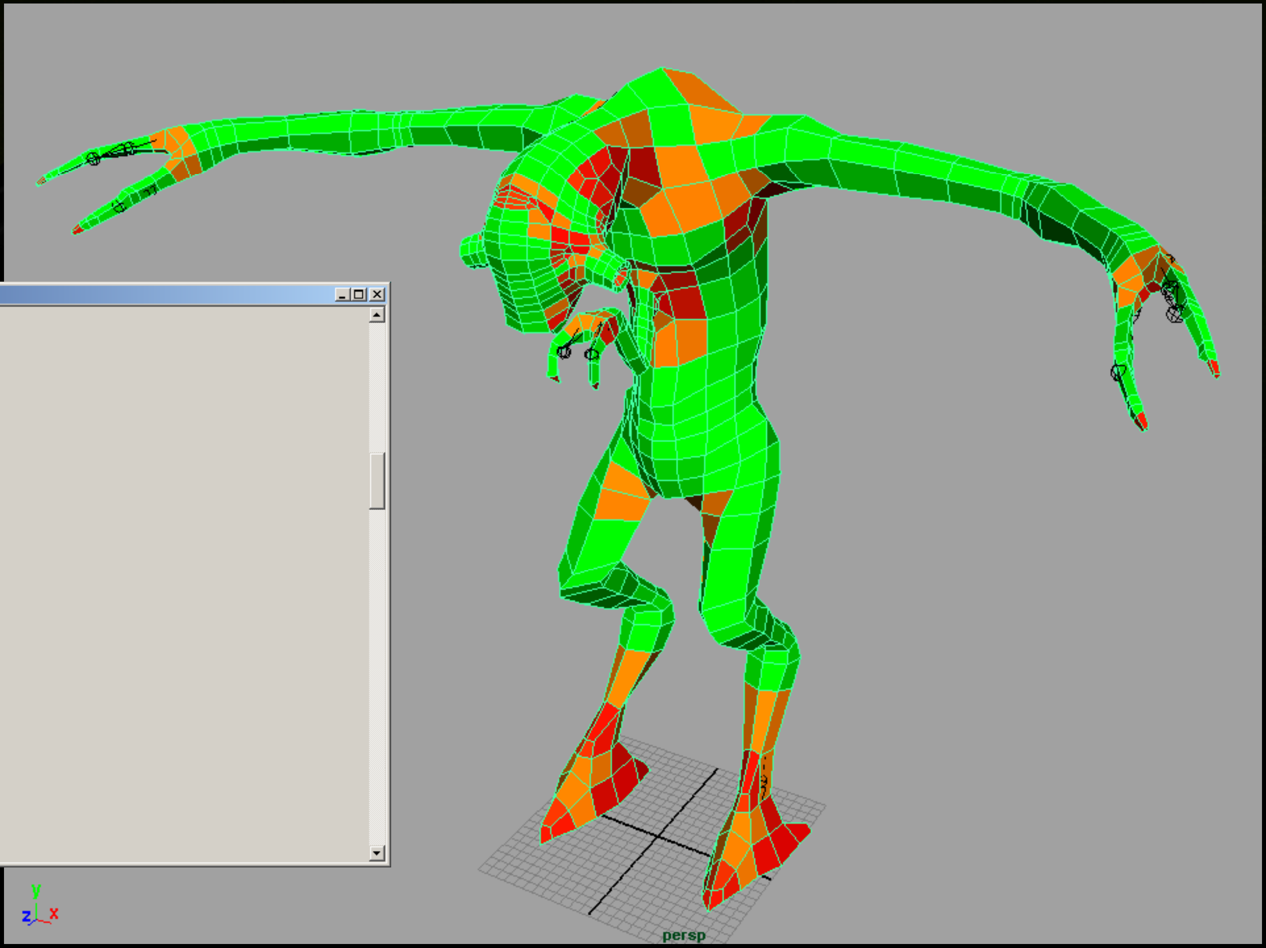


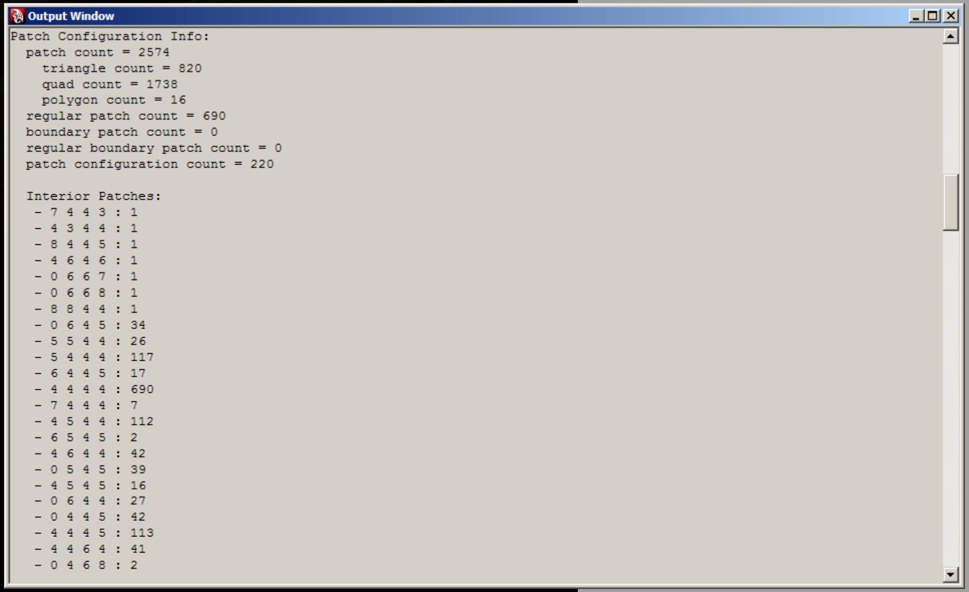


© Mike Asquith, Valve Corporation 2007

Topology Optimization

- Topology visualization tool (nvAnalyze)
  - Maya plugin that highlights faces that damage mesh quality the most





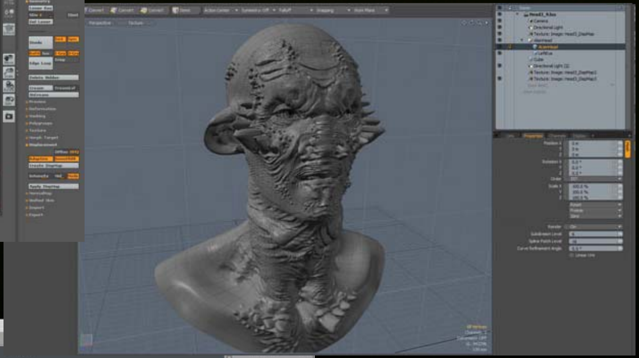
```
Patch Configuration Info:
  patch count = 2574
    triangle count = 820
    quad count = 1738
    polygon count = 16
  regular patch count = 690
  boundary patch count = 0
  regular boundary patch count = 0
  patch configuration count = 220

  Interior Patches:
   - 7 4 4 3 : 1
   - 4 3 4 4 : 1
   - 8 4 4 5 : 1
   - 4 6 4 6 : 1
   - 0 6 6 7 : 1
   - 0 6 6 8 : 1
   - 8 8 4 4 : 1
   - 0 6 4 5 : 34
   - 5 5 4 4 : 26
   - 5 4 4 4 : 117
   - 6 4 4 5 : 17
   - 4 4 4 4 : 690
   - 7 4 4 4 : 7
   - 4 5 4 4 : 112
   - 6 5 4 5 : 2
   - 4 6 4 4 : 42
   - 0 5 4 5 : 39
   - 4 5 4 5 : 16
   - 0 6 4 4 : 27
   - 0 4 4 5 : 42
   - 4 4 4 5 : 113
   - 4 4 6 4 : 41
   - 0 4 6 8 : 2
```

NVIDIA Mesh Processing Tool

- Successor of NVMeshMender and NVTriStrip but for subdivision surfaces:
  - Reorder faces for consistent adjacencies
  - Minimize topology combinations
  - Pre-compute stencils for different approximation schemes
  - Compute texture coordinates for watertight texture sampling
  - Optimize vertex and face order for best performance
  - And more!

Sculpting

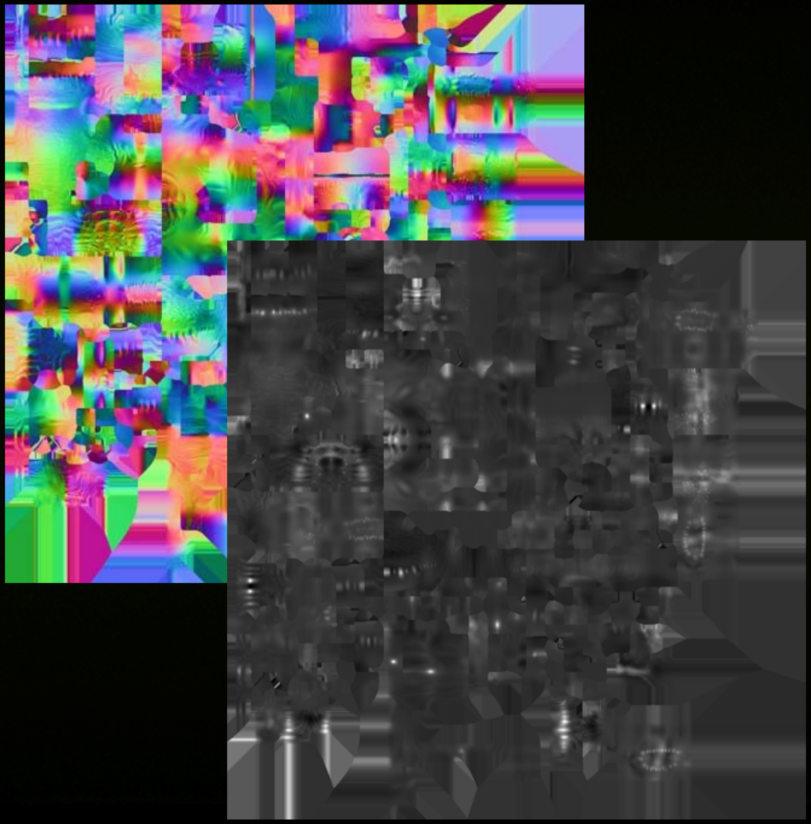
- Many tools available:
  - Autodesk® *Mudbox*™
  - Pixologic ZBrush®
  - modo™, Silo, Blender, etc.



Nevercenter.com | Model Copyright © JMichael Velasquez, 2007

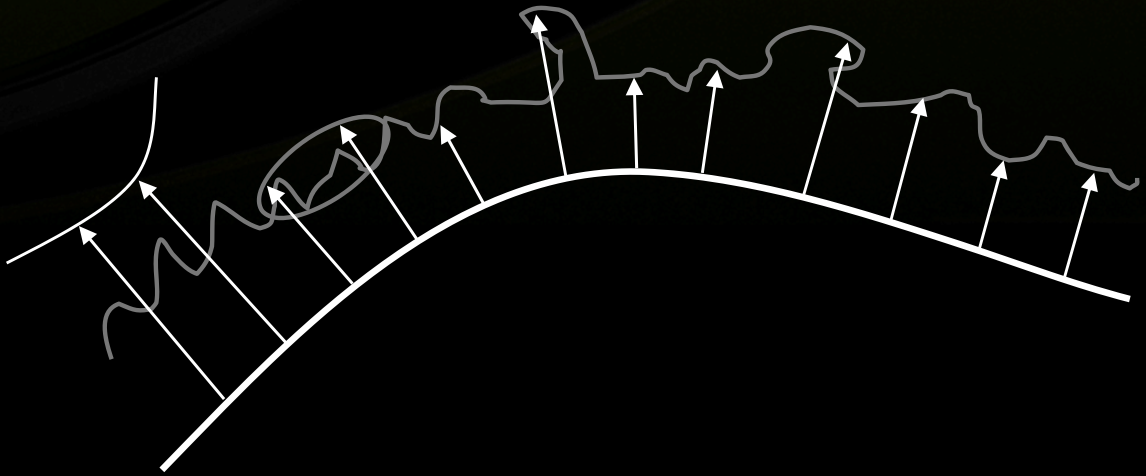
Baker Tools

- Many options:
  - xNormal™
  - Mudbox™, ZMapper
  - Melody™, etc.
  - PolyBump™, etc.
- Two approaches
  - Ray casting
  - Dual parameterization



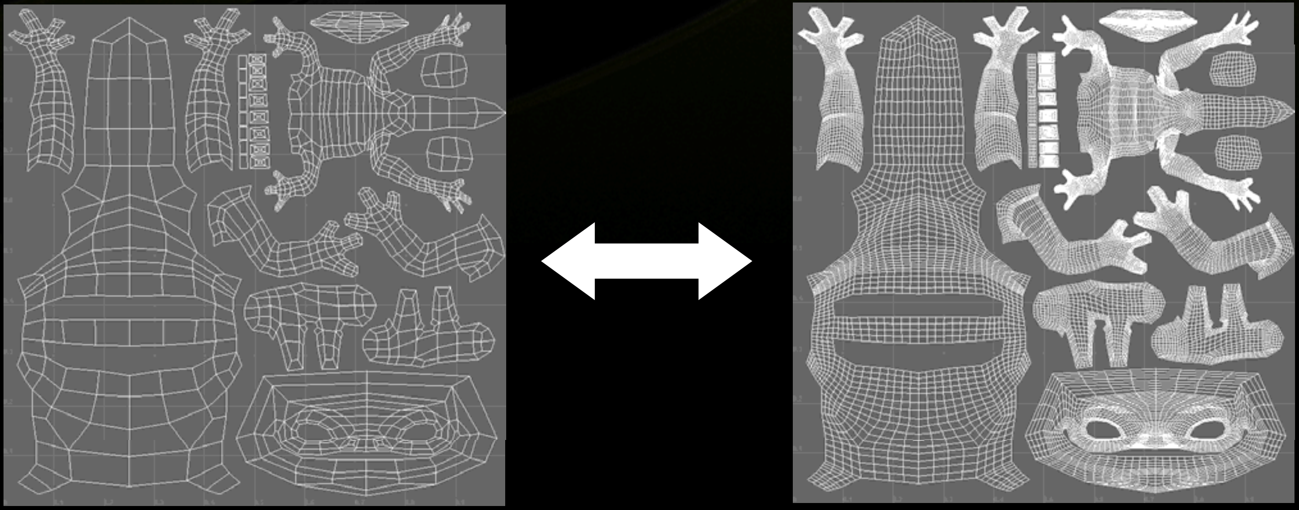
Capturing Attributes

- Ray casting
  - Can sample complex meshes made of multiple pieces
  - Produces better scalar displacements
  - Occasional artifacts (missing rays, double hits)
  - Require artist supervision and tweaking



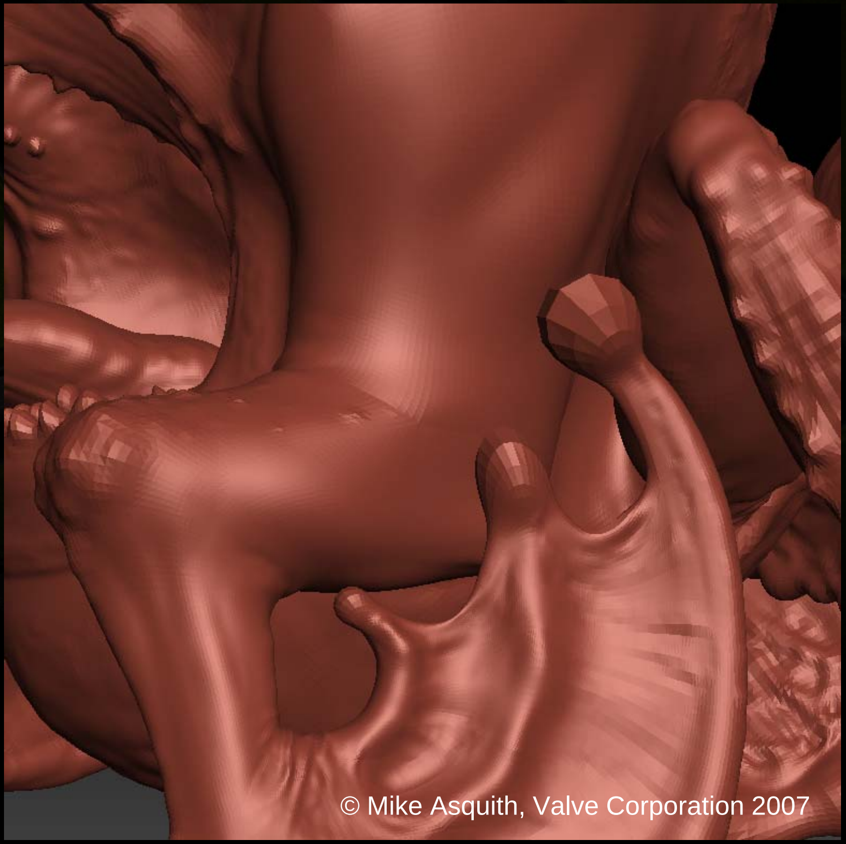
Capturing Attributes

- Dual parameterization
  - Much faster, easy to implement
  - Higher quality vector displacements
  - Artifact free, no artist supervision required
  - Inaccurate scalar displacements
  - Low and high res meshes must have same topology

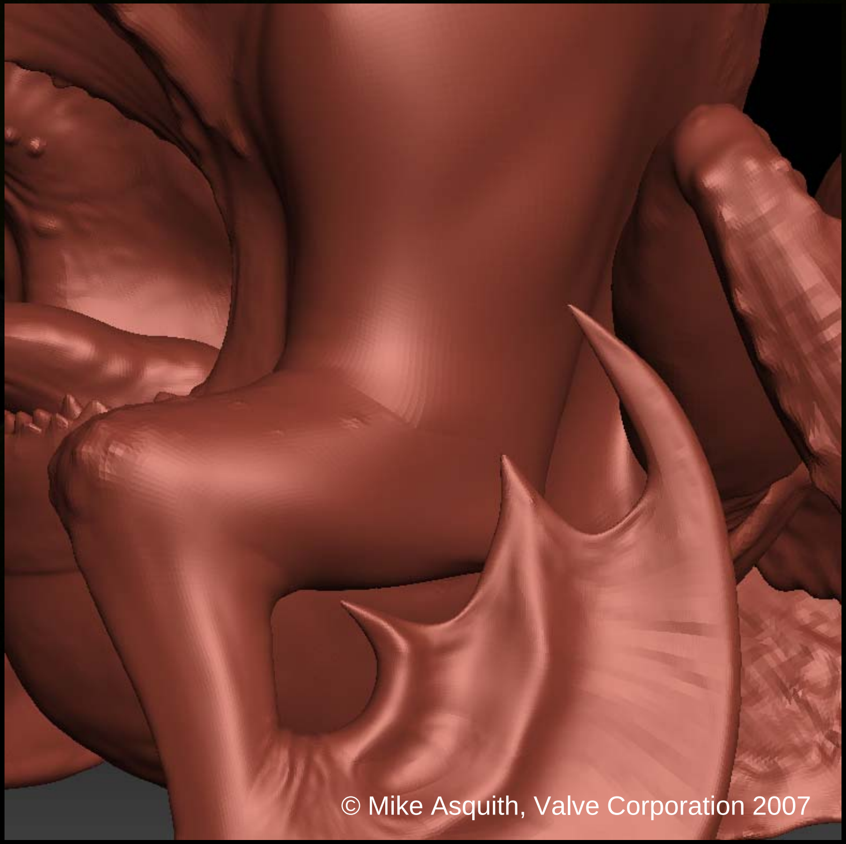


Vector Displacements

- **Native representation of most sculpting tools**



1D Displacements



3D Displacements

NVIDIA Baker Tool

- Uses dual parameterization to extract:
  - Normal and displacement maps
  - Only tool that generates vector displacements
  - Occlusion maps, and more!
- No other tool supports custom base surfaces:
  - Bezier ACC
  - Gregory ACC
  - Triangle meshes

NVIDIA Baker Tool

- Uses optimized Montecarlo Raytracer

- Can be easily extended to support:
  - Bent normals
  - Spherical harmonic PRTs
  - etc.

- Full source code will be openly available

Thanks

- Bay Raitt, Mike Asquith, Valve Corporation
- Kenneth Scott, id Software

Q & A

- Bryan Dudash bdudash@nvidia.com
- Chris Kim ckim@nvidia.com
  - Korean Devtech

Thank you!

developer.nvidia.com