NVIDIA.

SIGGRAPH2008

Next-Generation Rendering of Subdivision Surfaces

SIGGRAPH2008

Ignacio Castaño

Developer Technology - NVIDIA

Outline

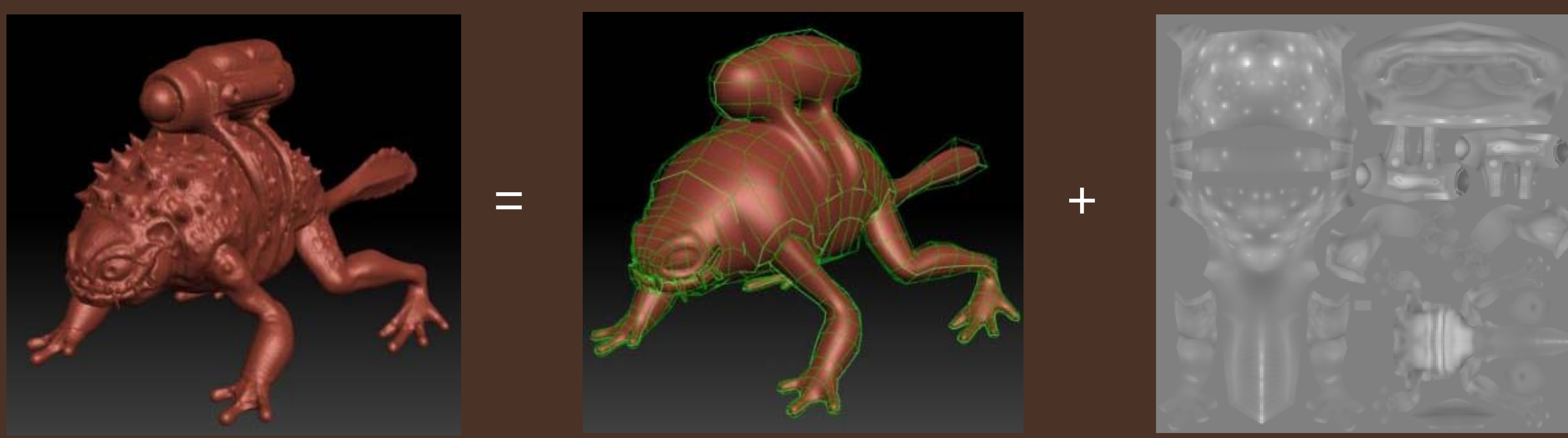
- Motivation
- Displaced Subdivision Surfaces
  - Background
  - Direct Evaluation Methods
  - Implementation on next-gen GPUs
  - Implementation on current GPUs
- Content Creation
  - Tools and guidelines

Motivation

© Kenneth Scott, id Software 2008

Compression

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

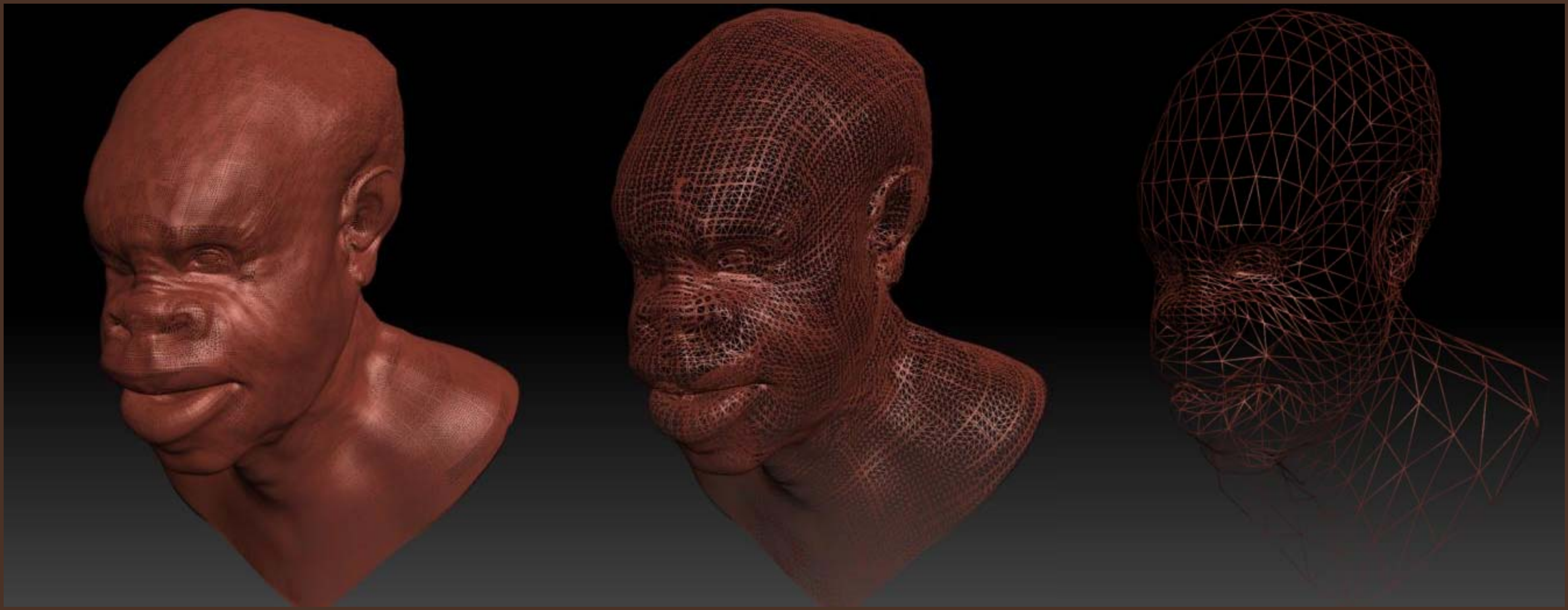


© Bay Raitt

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

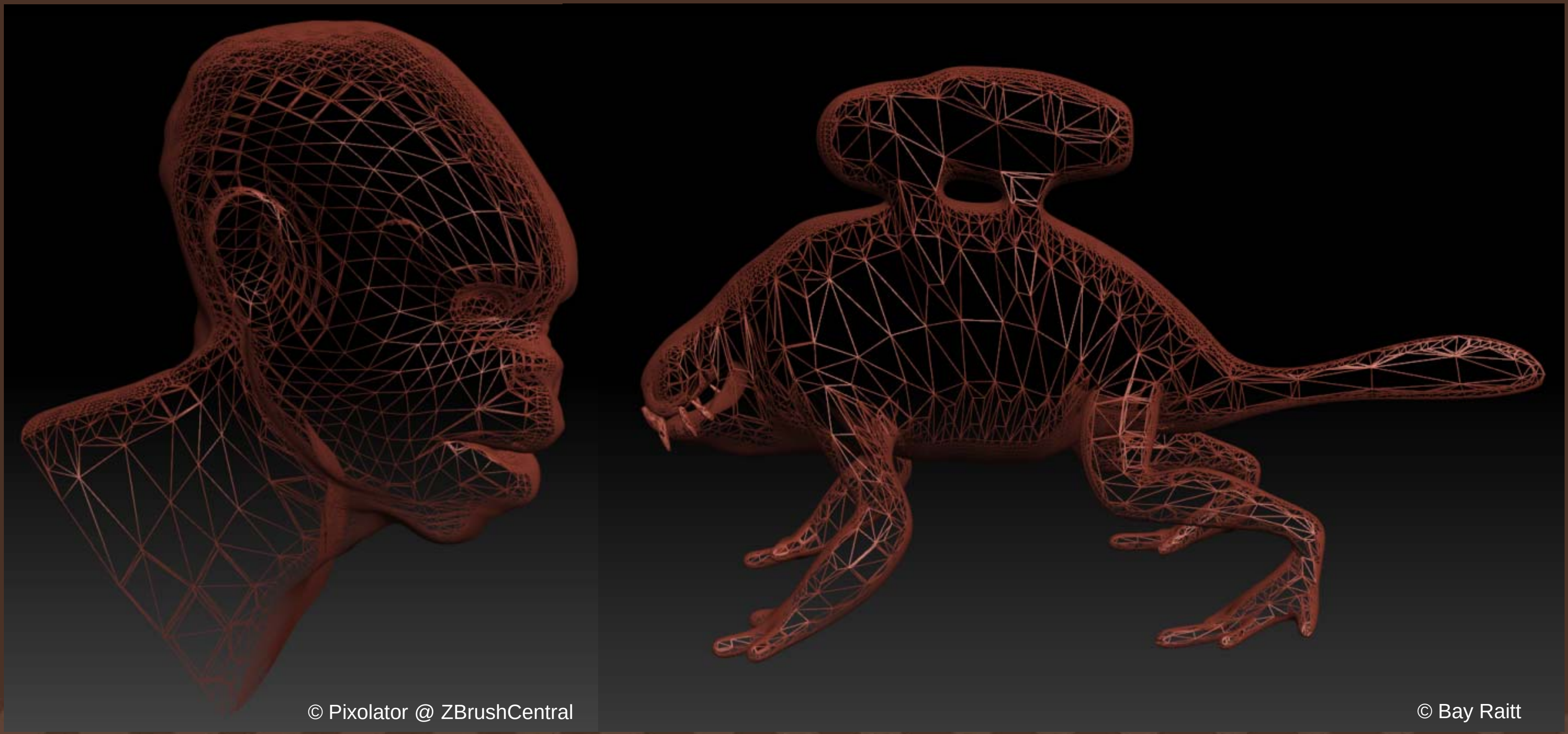
Scalability

- Continuous Level of Detail



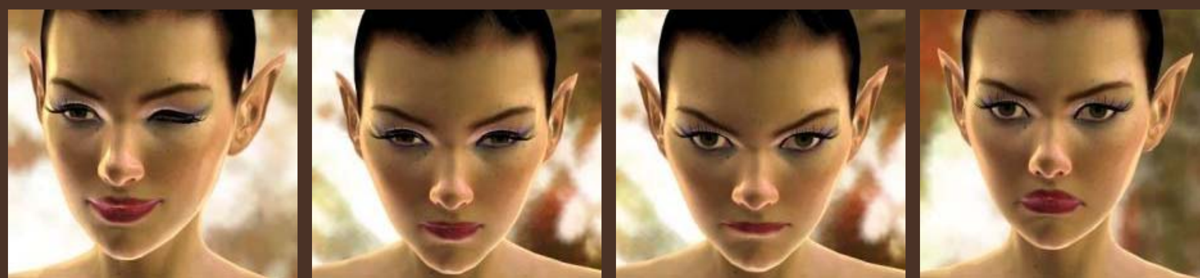
© Pixolator @ ZBrushCentral

Scalability



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

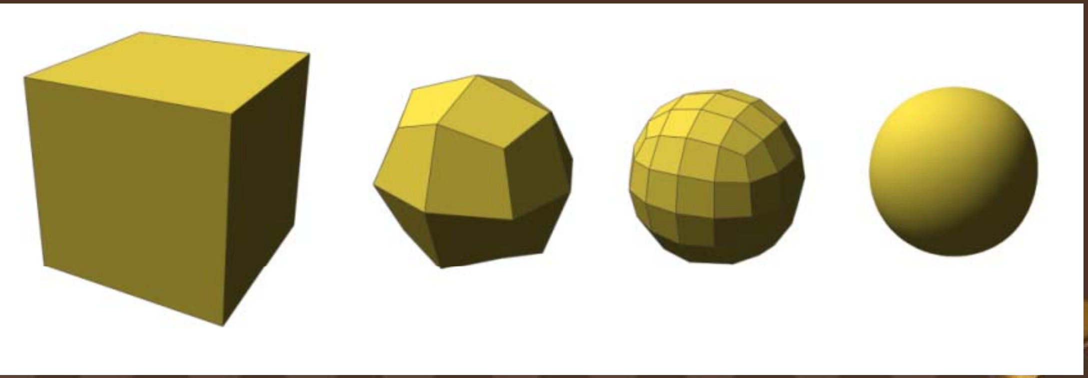
- Well understood, mature technology
- Widespread use in the movie industry
- Readily available modeling and sculpting tools

Geri's Game © Pixar Animation Studios 1998

SIGGRAPH2008

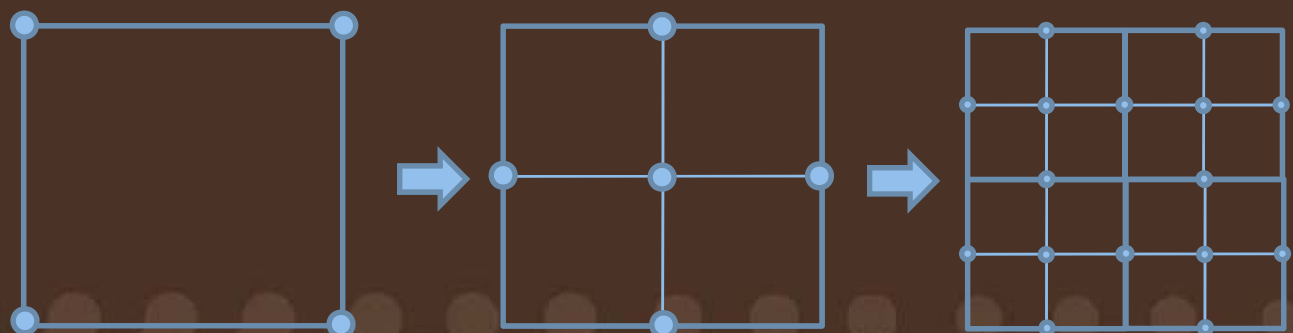
Subdivision Surfaces

- Overcome deficiencies of previous methods
- Limit of an infinite recursive refinement process
  - Not limited to rectangular grids of control nets
  - Represent arbitrary surface topologies
- Ease of manipulation is their main advantage

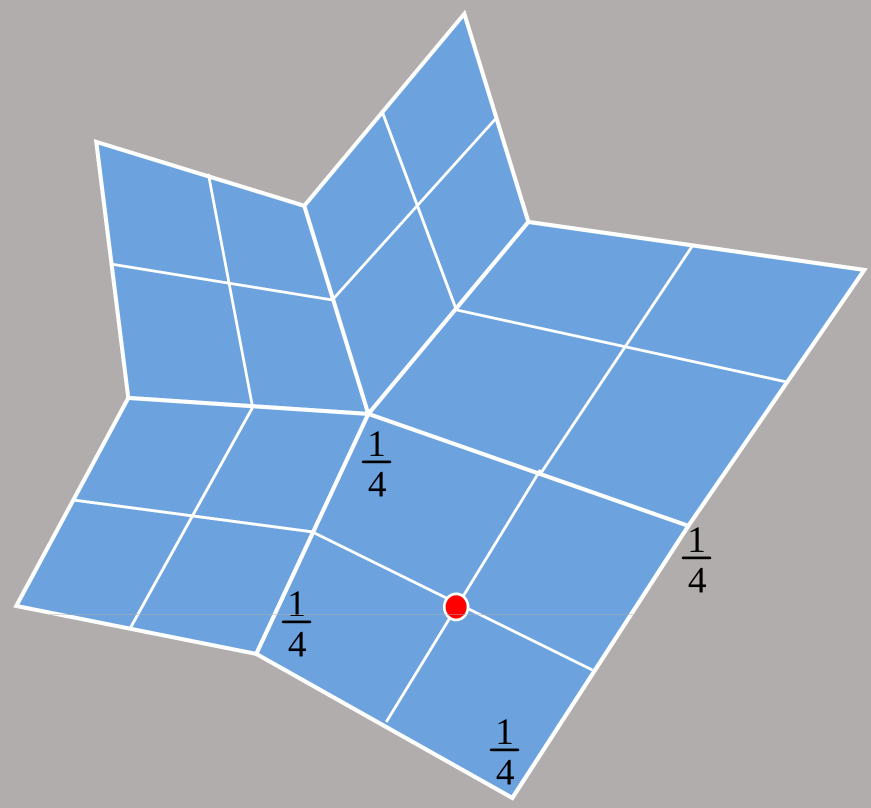


Catmull-Clark Subdivision

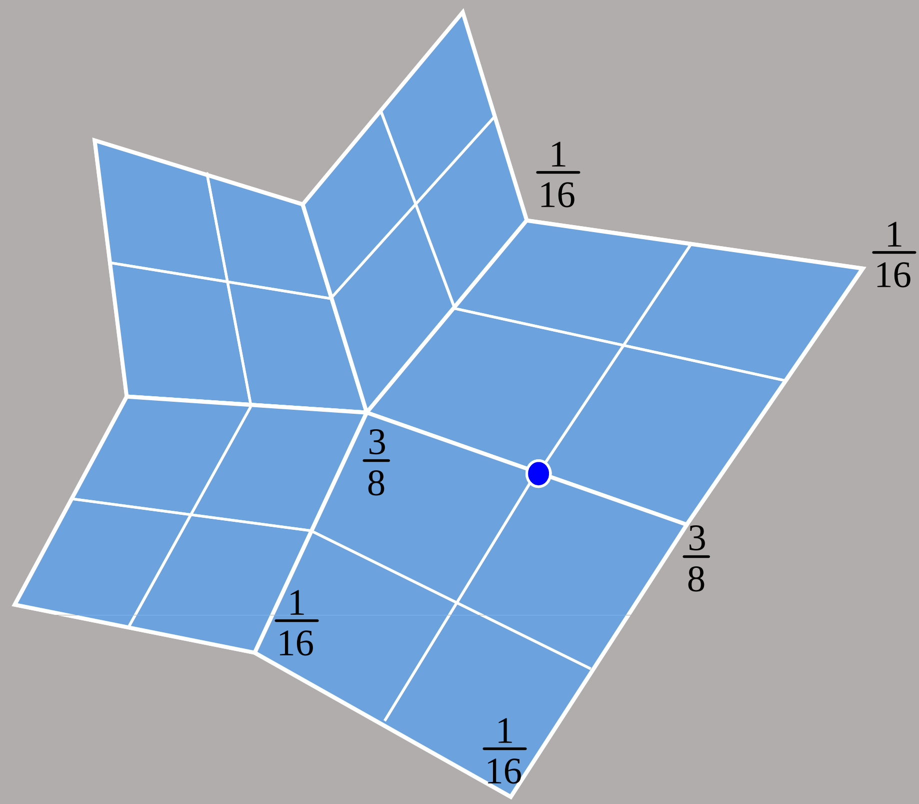
- Most popular scheme
  - Based on B-Spline subdivision
- Evaluation through recursive refinement
  - Add one new vertex for every face and every edge



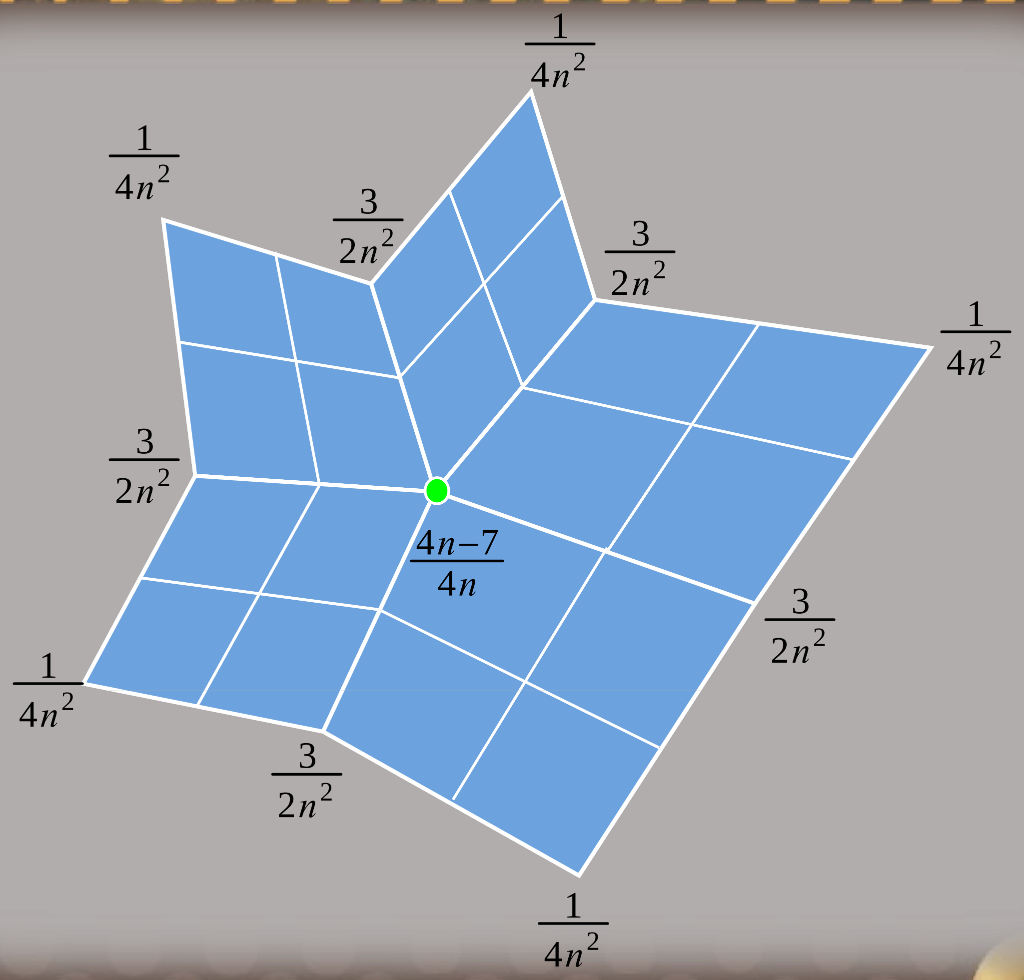
Catmull-Clark Subdivision



Catmull-Clark Subdivision



Catmull-Clark Subdivision

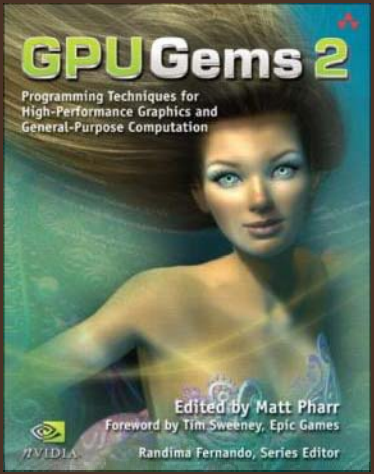


Catmull-Clark Subdivision

- Resulting mesh only approximates the limit subdivision surface
- Halstead et al show that it’s possible to project the vertices to the limit surface
  - Efficient, Fair Interpolation using Catmull-Clark Surfaces
    http://graphics.cs.uiuc.edu/~jch/cs497jch/halstead.pdf

GPU Implementations

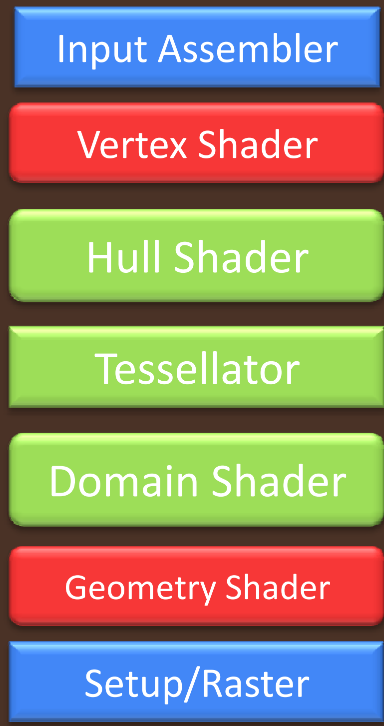
- Previous approaches on the GPU:
  - “Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping”, Michael Bunnell (Render to Texture)
  - Recursive Geometry Shader refinement (Stream Out)



- Require multiple passes:
  - High bandwidth requirement
  - Direct evaluation is preferred

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)

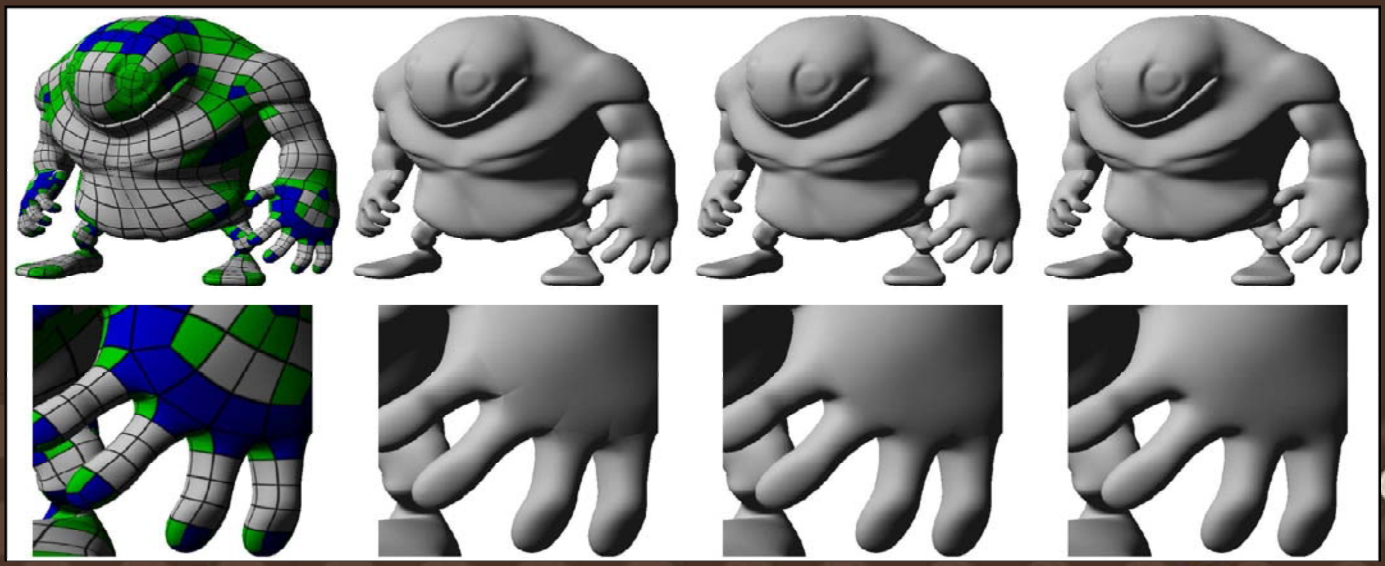


Direct Evalution of Subdiv. Surfaces

- 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-tabulated basis for each topology and each possible tessellation level

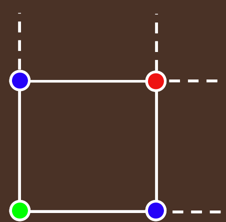
Approximating Catmull-Clark Subdivision Surfaces (ACC)

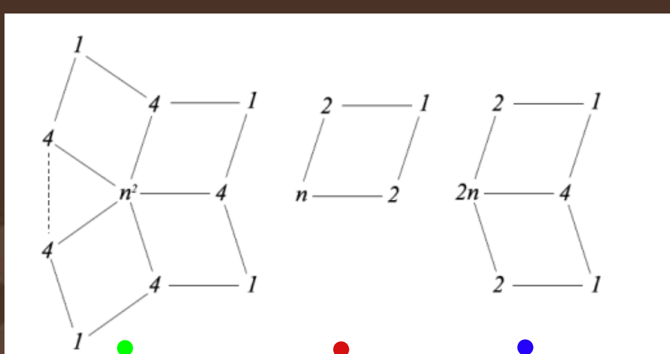
- Developed by Charles Loop and Scott Schaefer: http://research.microsoft.com/~cloop/
- Surface approximated with a Bezier patch and a pair of independent tangent patches



Approximating Catmull-Clark Subdivision Surfaces (ACC)

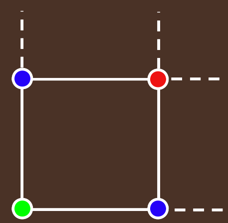
- Corner control point is vertex projected to limit surface

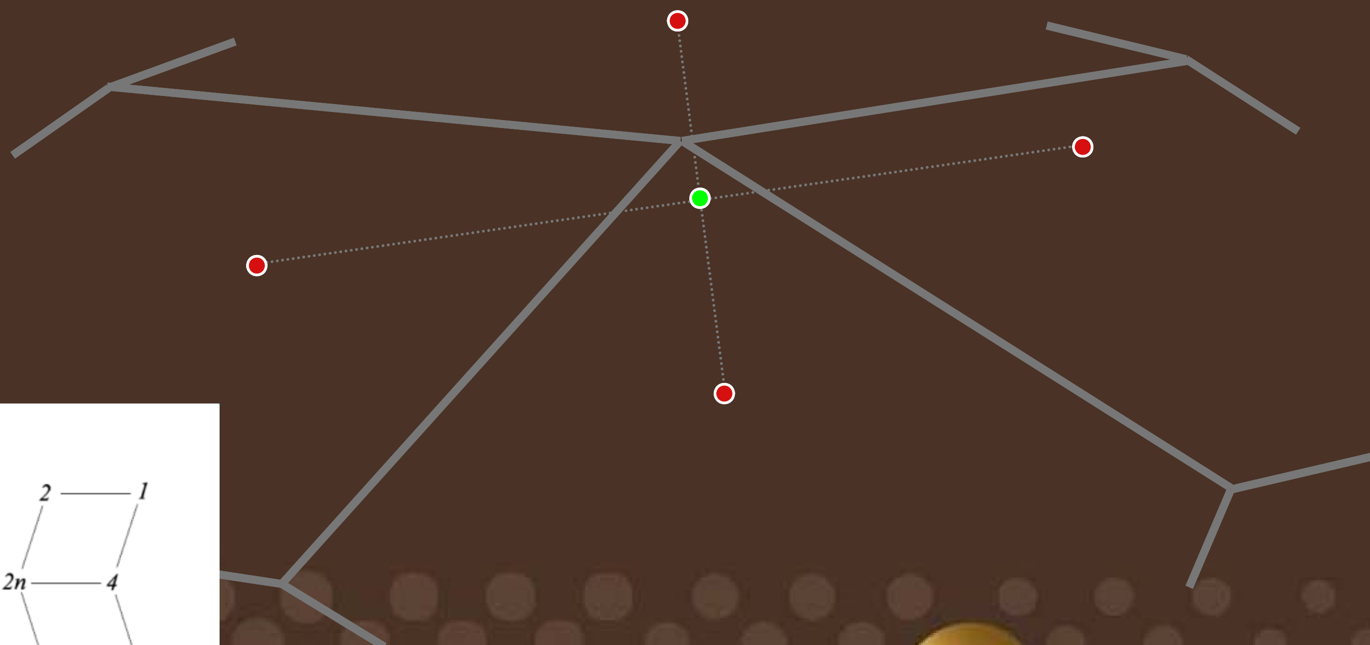


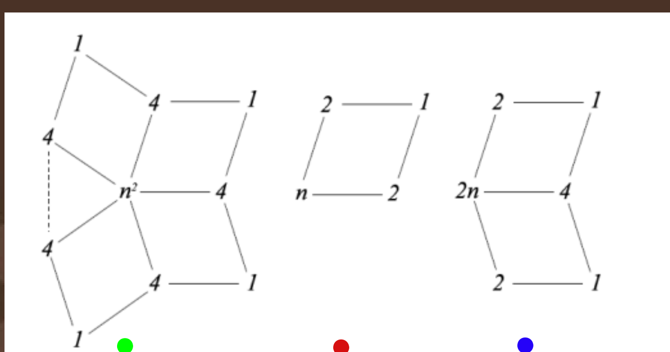


Approximating Catmull-Clark Subdivision Surfaces (ACC)

- Centroid of interior control points equal to limit position of corner vertex



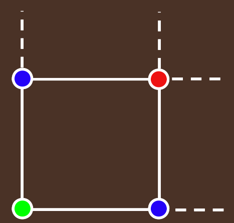


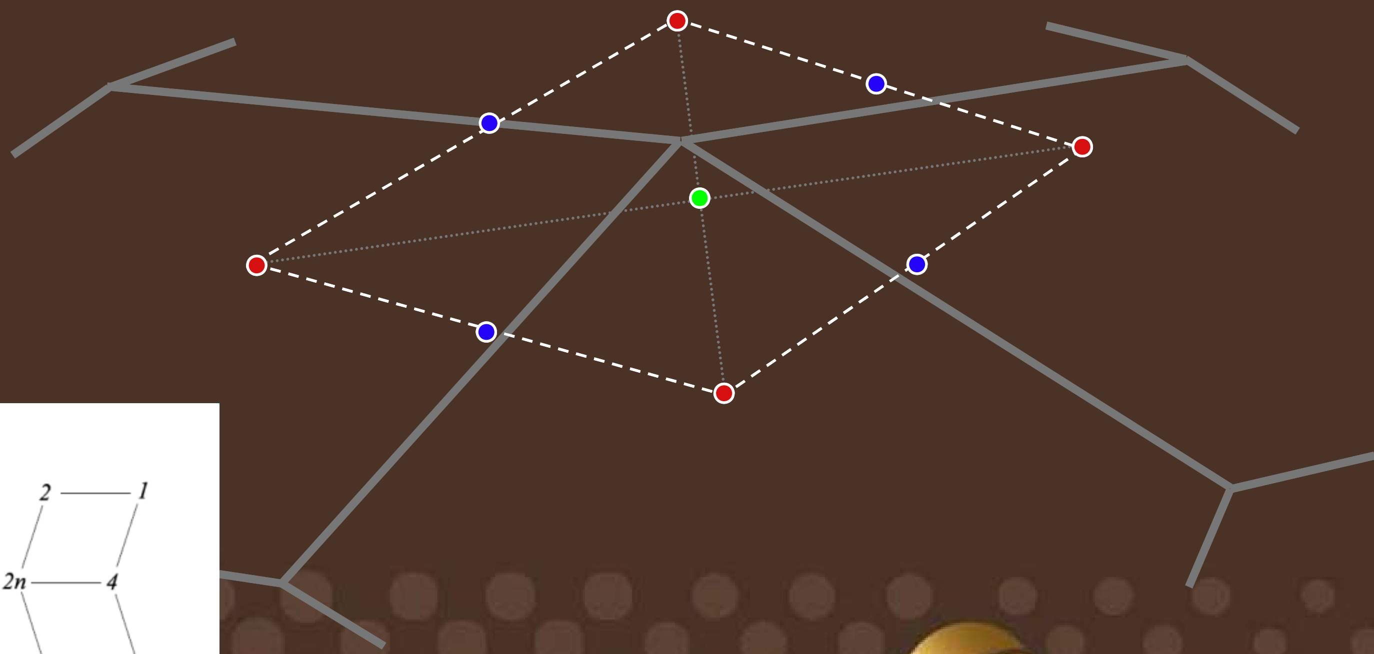


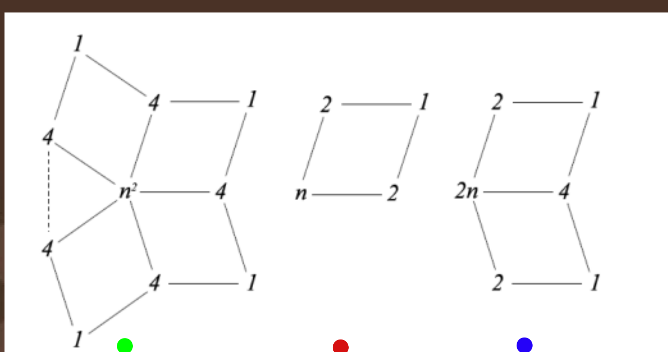
SIGGRAPH2008

Approximating Catmull-Clark Subdivision Surfaces (ACC)

- Edge control point midpoint of two interior control points







Approximating Catmull-Clark Subdivision Surfaces (ACC)

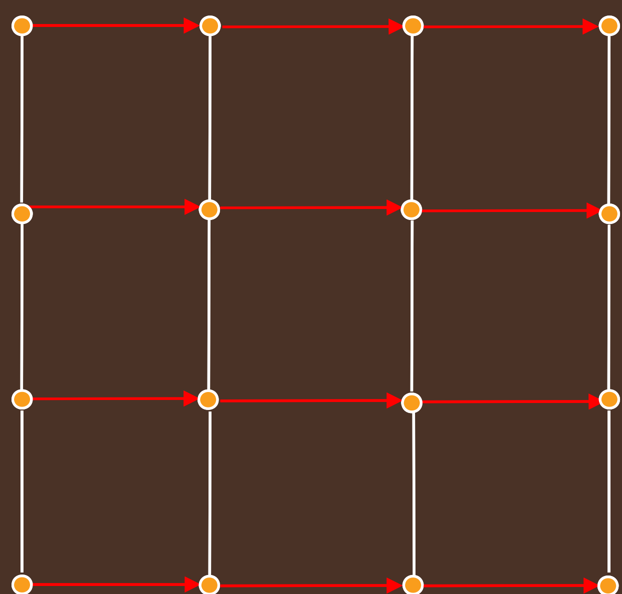
- This construction is exact on regular patches
  - Equivalent to B-Spline to Bezier conversion
- Approximate on irregular patches
  - Edges around extraordinary vertices only have C0 continuity

Approximating Catmull-Clark Subdivision Surfaces (ACC)

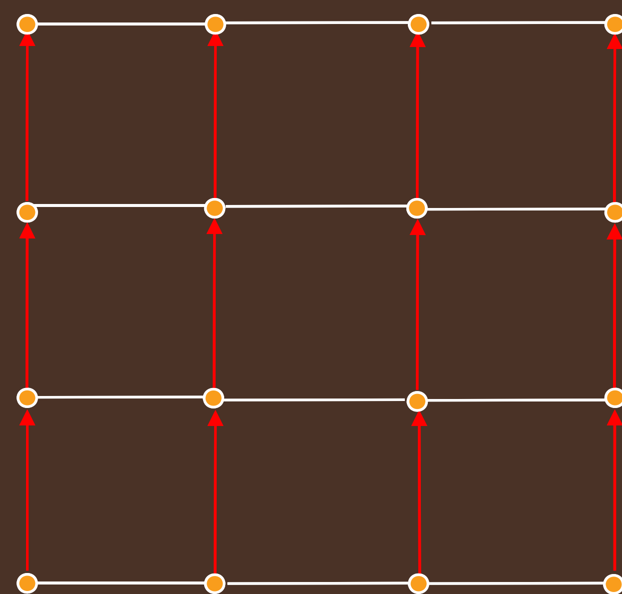
- Use separate tangent field for shading in order to hide tangent discontinuities
- Control tangents are computed similarly:
  - Corner tangents are tangents of the limit surface
  - Boundary tangents are constructed to satisfy continuity condition

Bezier Patches

- Problem with Bezier Patches:



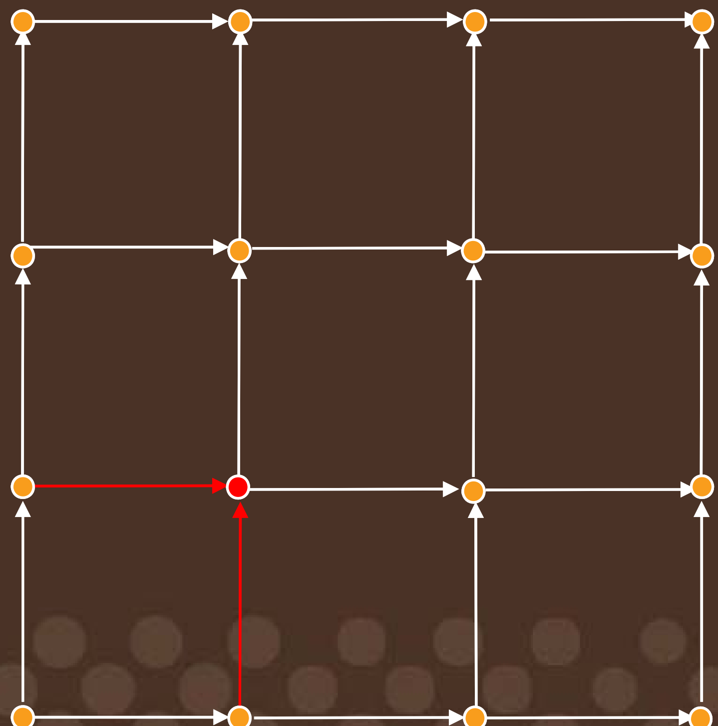
$$\frac{\partial f(u,v)}{\partial u}$$



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

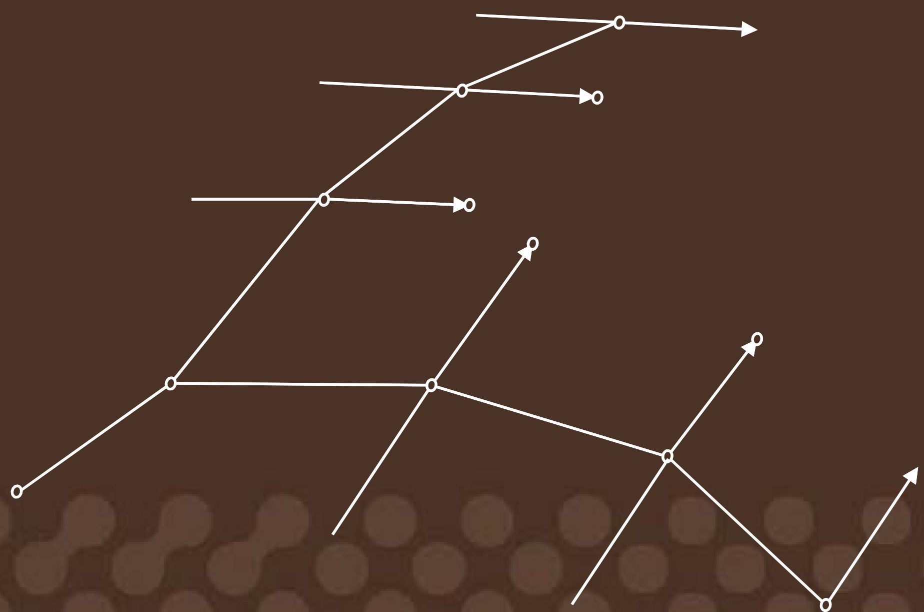
Bezier Patches

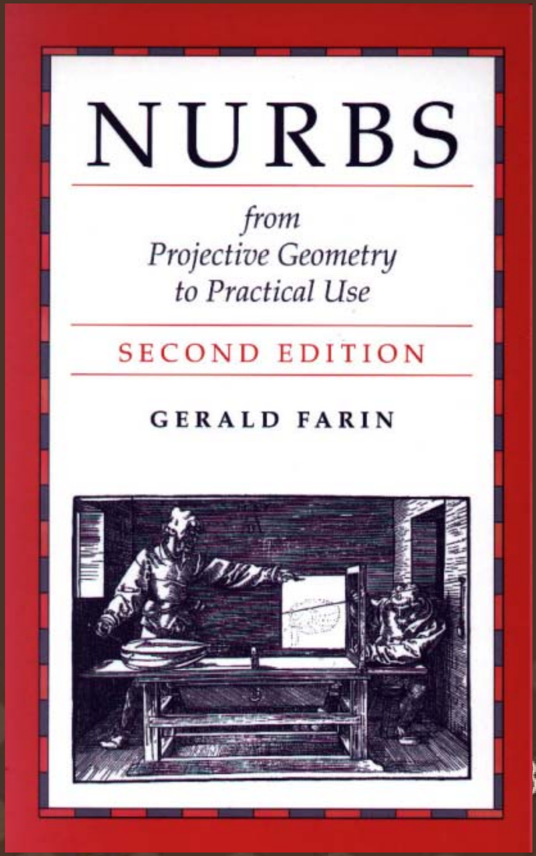
- Derivatives along the edges cannot be specified independently



Gregory Patches

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

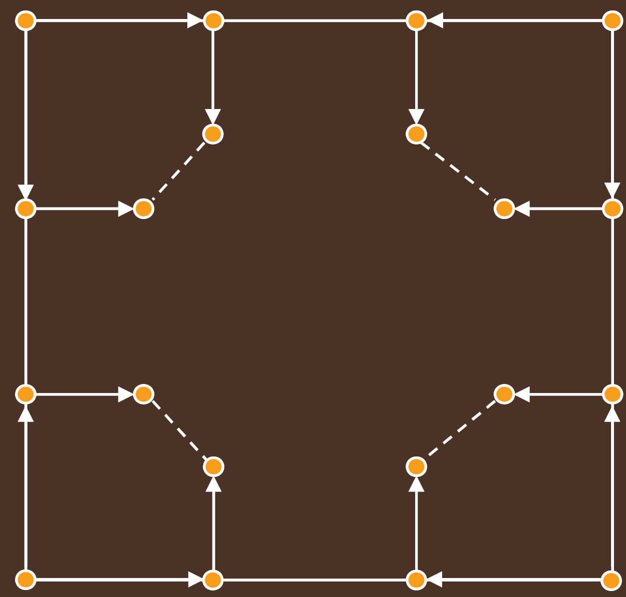




Gregory Patches

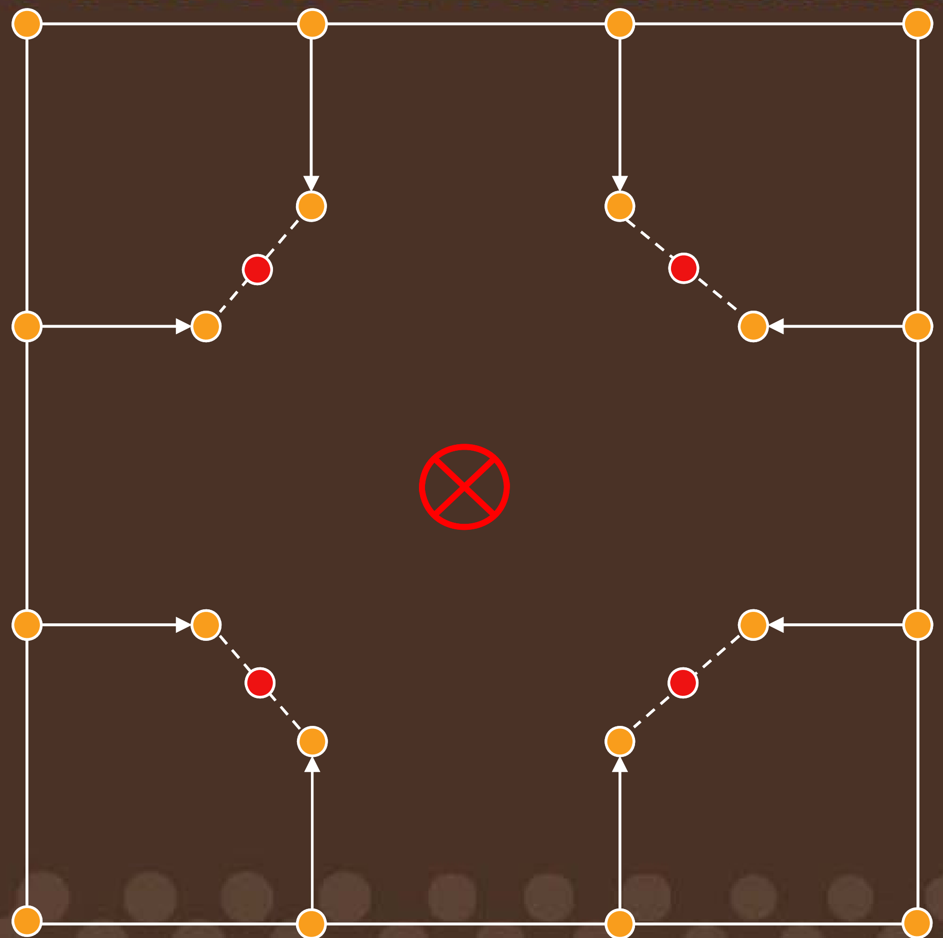
- 20 control points instead of 16
- Evaluated like Bezier where interior control point is computed as:

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

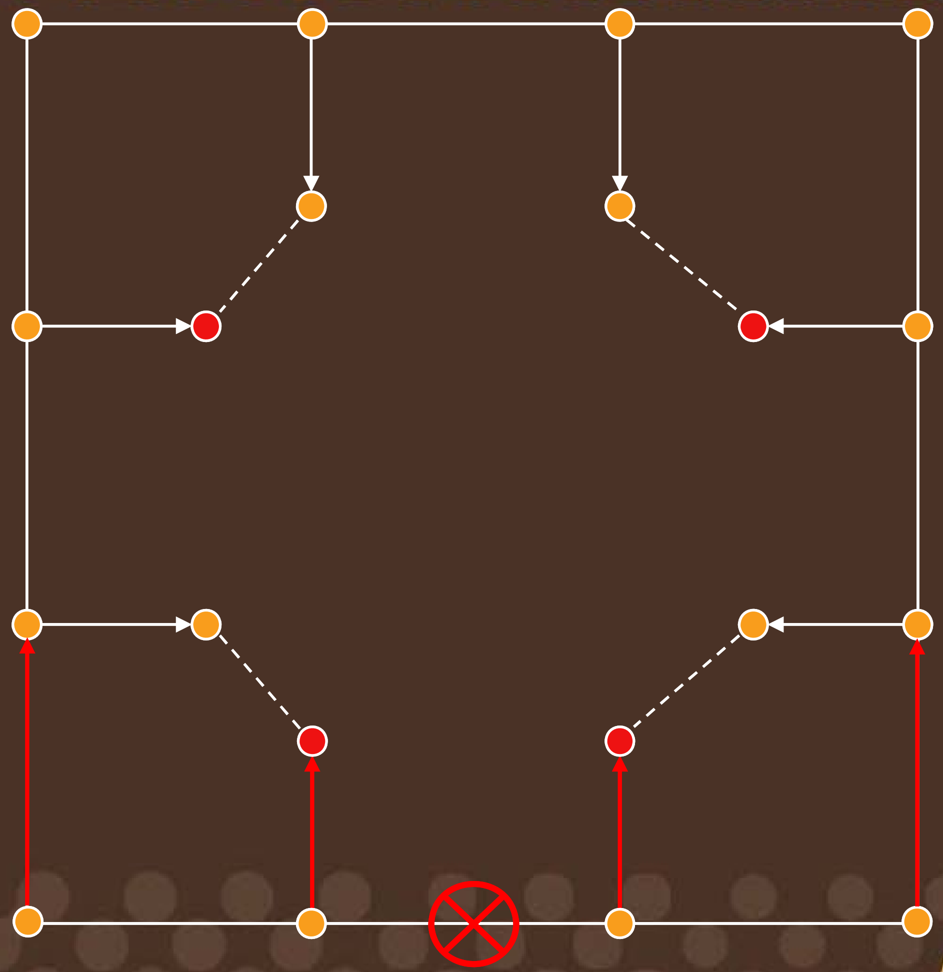


- On regular faces Gregory patch becomes a Bezier patch

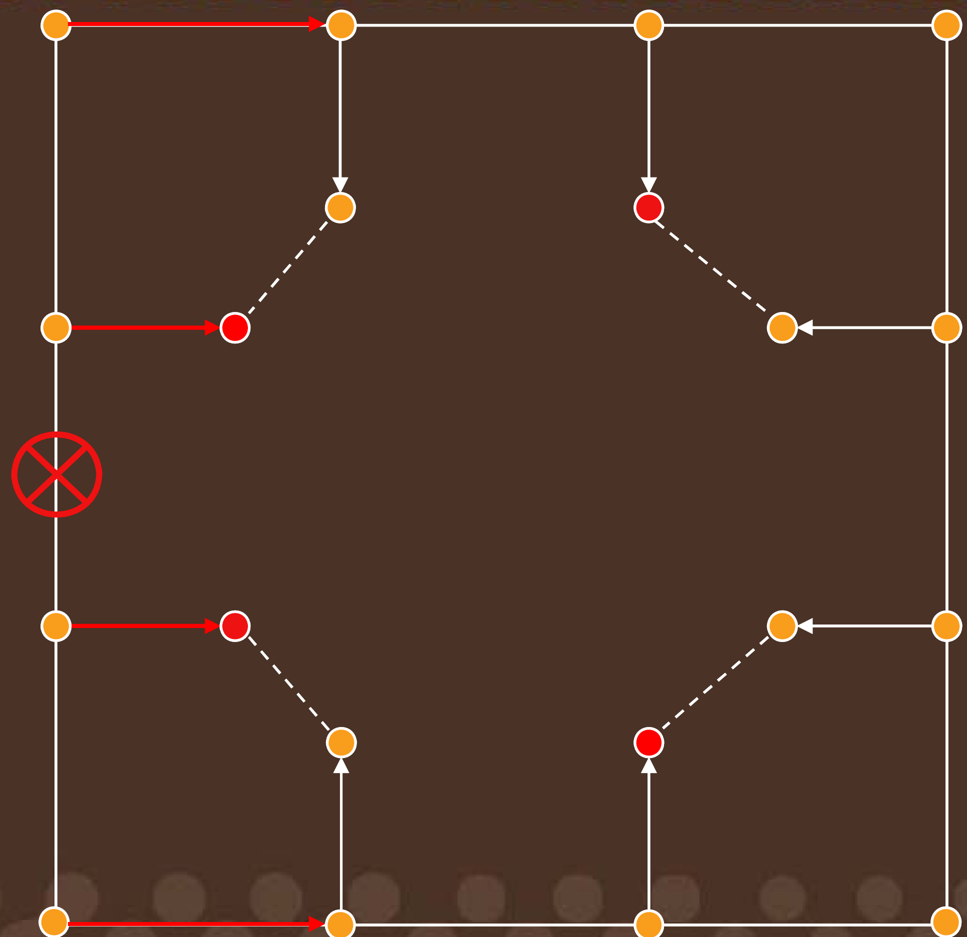
Gregory Patches



Gregory Patches



Gregory Patches



Gregory Patches

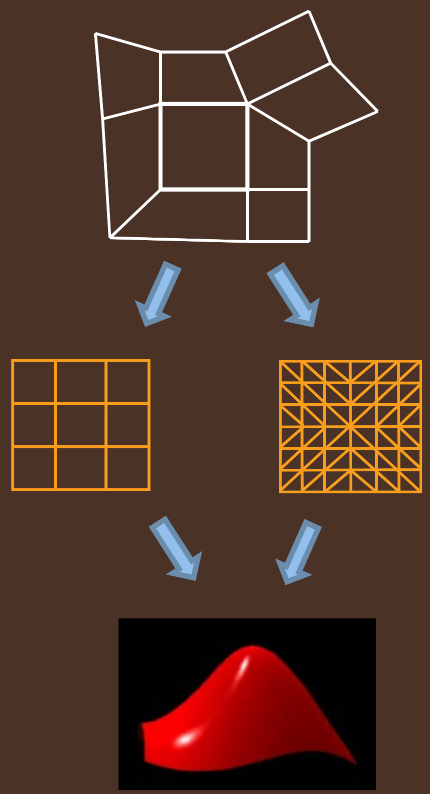
- Less control points to evaluate and transfer
  - Bezier requires 16 position & 16 tangent control points
- Surface evaluation is more efficient
  - Position and tangents can be evaluated simultaneously with de Casteljau
- Less degrees of freedom
  - Not so good approximation

Gregory Triangles

- ACC can be extended to triangular patches
- Quad-Triangle meshes supported as well

Tessellation Pipeline

- Hull Shader transforms input face to regular surface representation
- Tessellator produces a semi-regular tessellation pattern
- Domain Shader evaluates surface



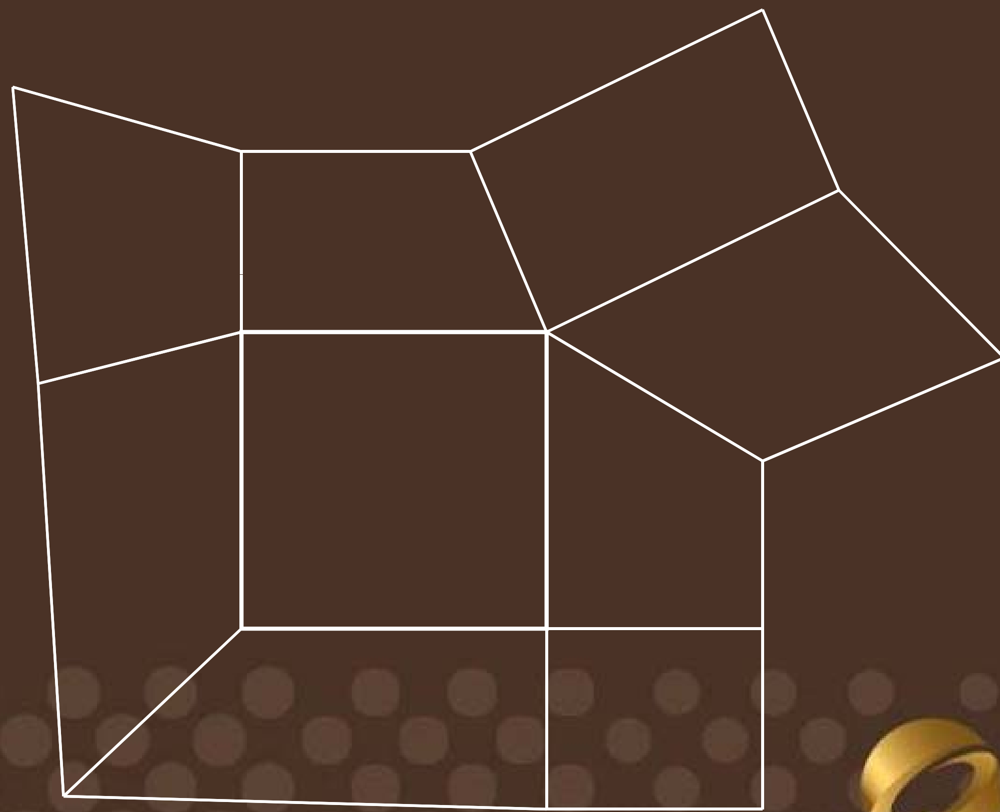
SIGGRAPH2008

Input Assembler

- New patch primitive type
  - Arbitrary vertex count (up to 32)
  - No implied topology
  - Only supported primitive when tessellation is enabled

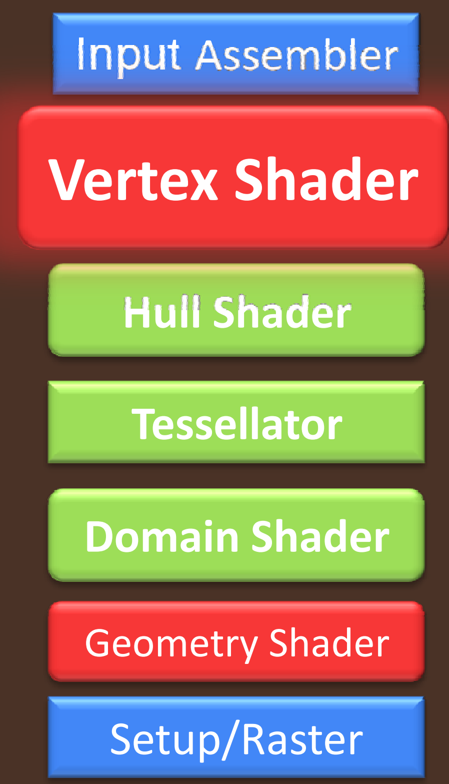
Input Assembler

- A patch is a face and its neighborhood:



Vertex Shader

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



Hull Shader (HS)

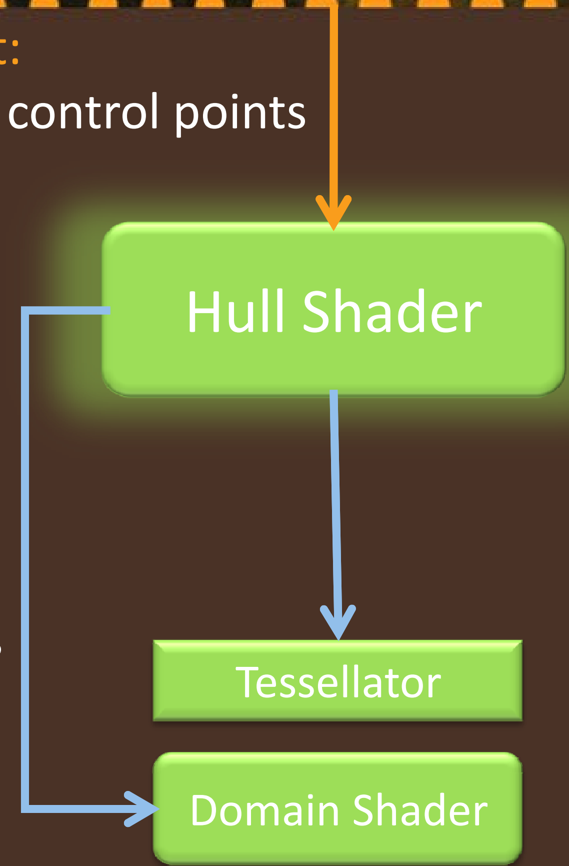
- Transforms control points to a different basis
- Computes edge tessellation levels

Hull Shader (HS)

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

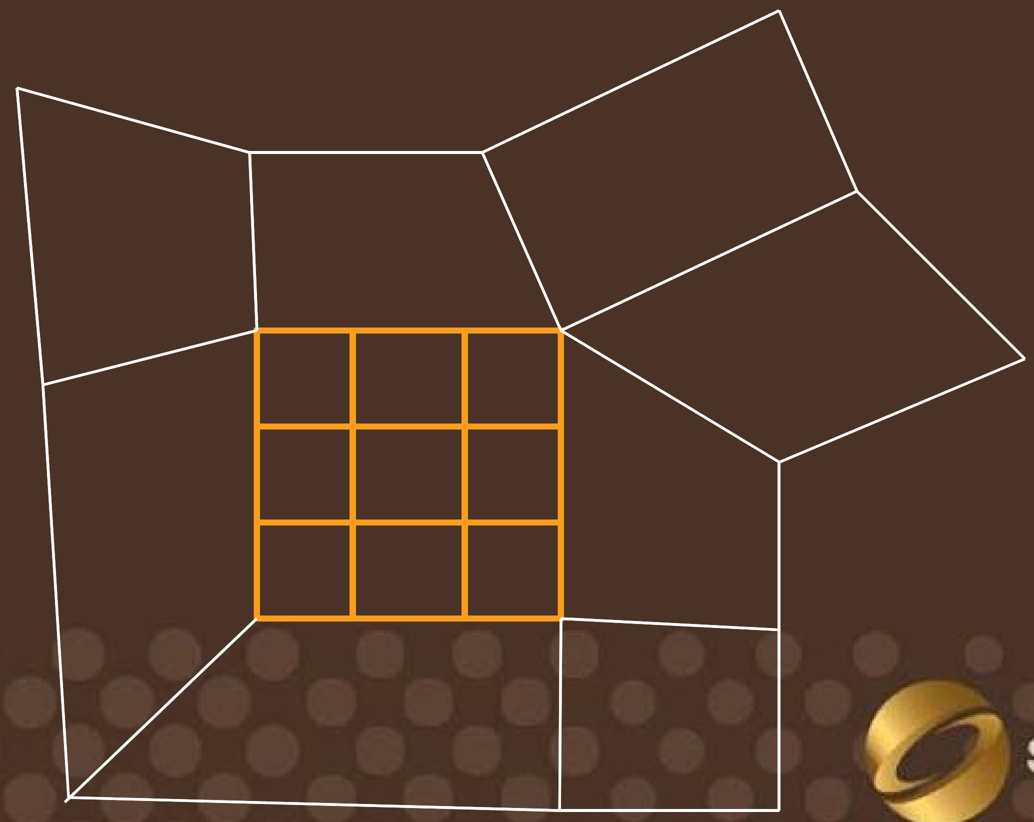
HS input:
- [1..32] control points

HS output:
- [1..32] control points
- Tessellation factors



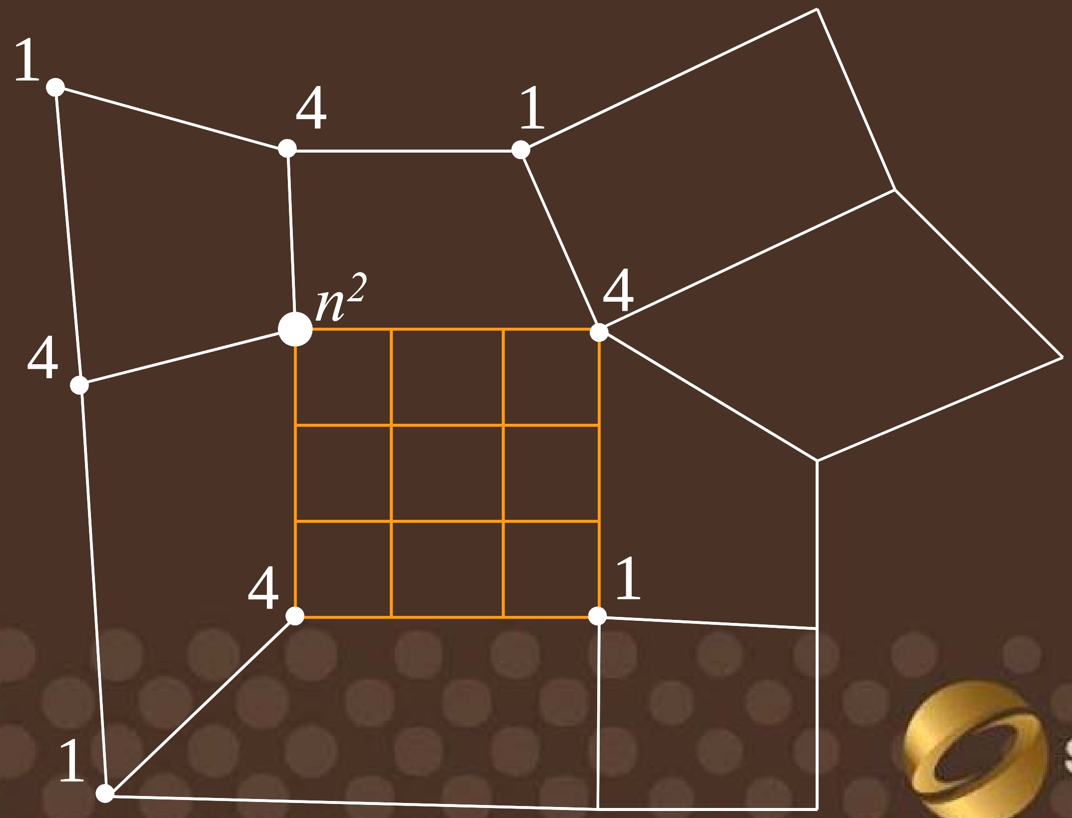
Control Point Evaluation

- HS output is a regular bicubic Bezier patch:



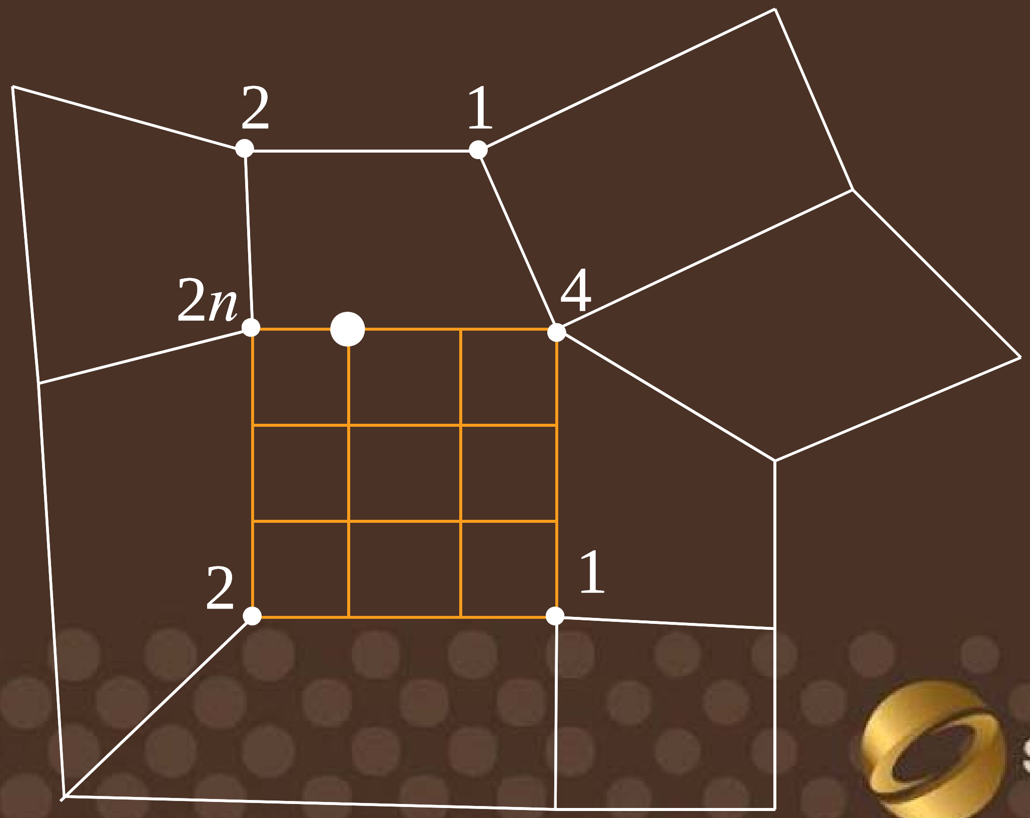
Control Point Evaluation

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



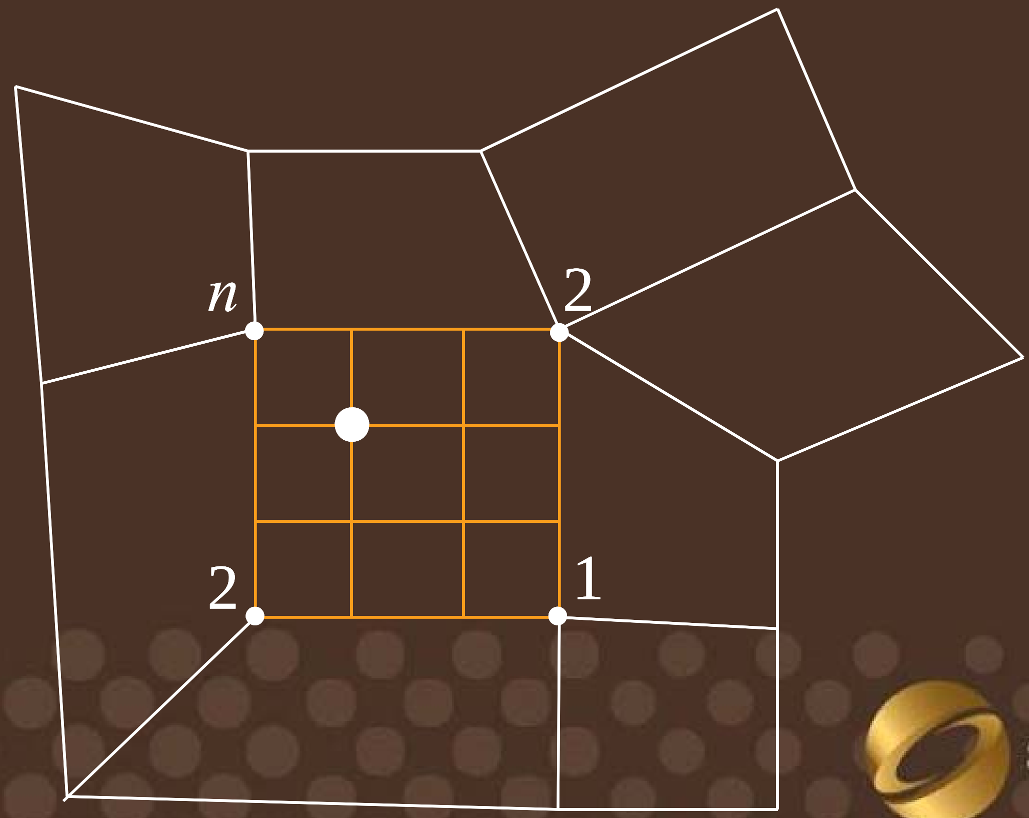
Control Point Evaluation

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



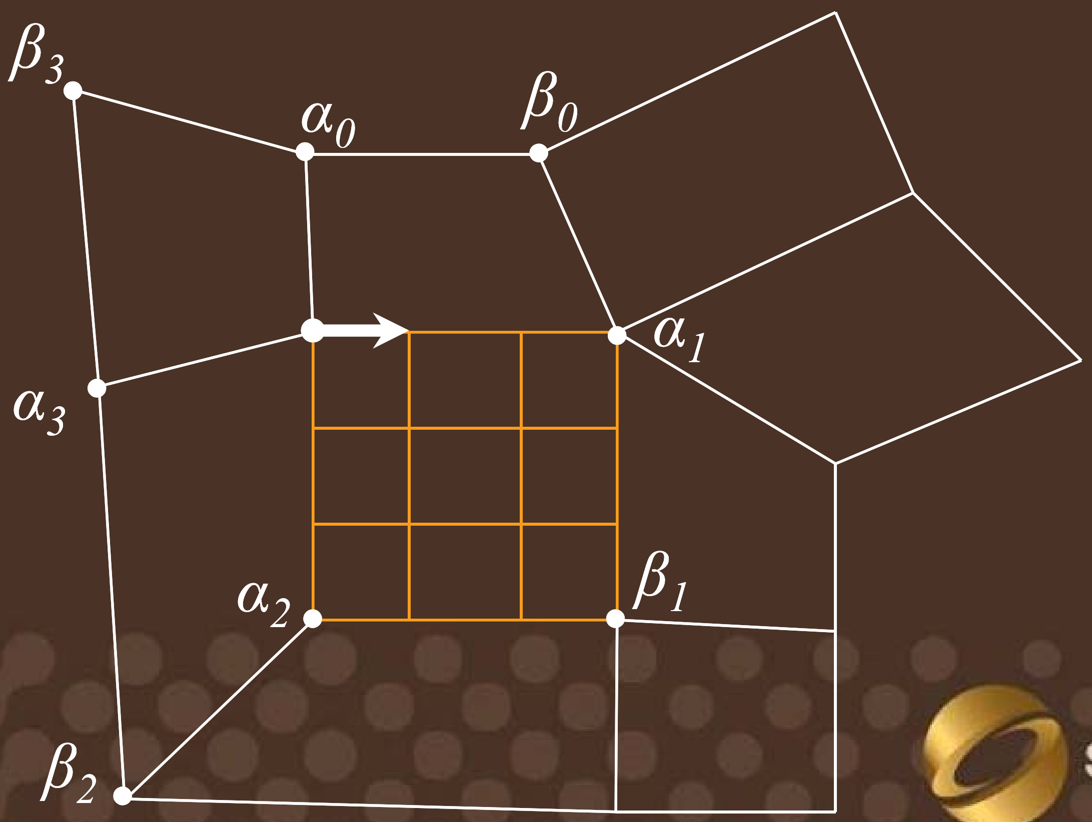
Control Point Evaluation

- 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: $P_j = \text{Sum}(W_{ij} * V_i)$
- 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];      } For each input vertex Vi
     }
     pos[threadID] = p;
}
```

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

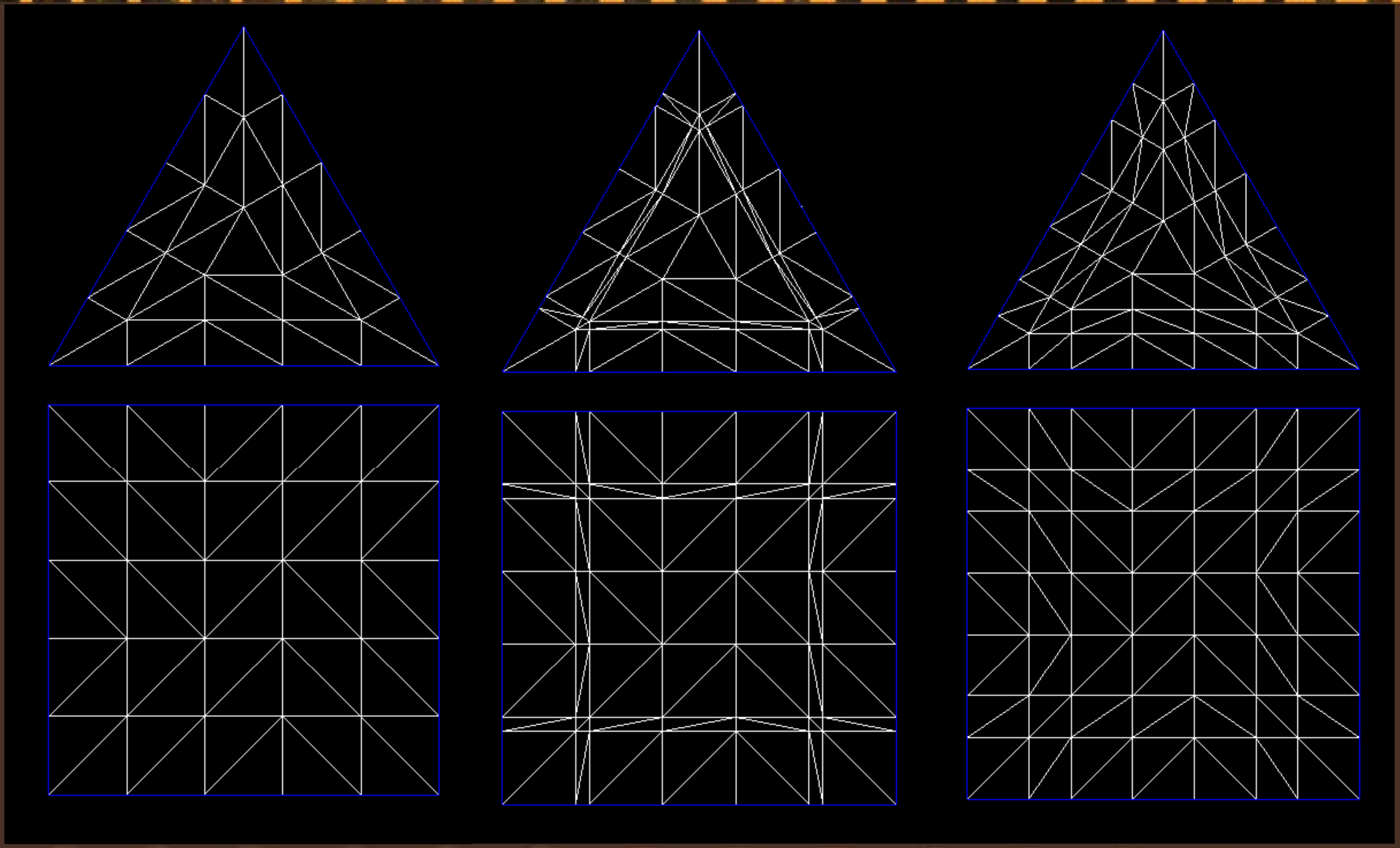
Tessellation Level Evaluation

- Hull Shader also computes tessellation levels
- Can use any metric:
  - Distance to camera
  - Screen space length of hull boundary
  - Hull curvature
  - Texture space length of patch edges
  - Precomputed edge factors based on displacement roughness

Tessellator (TS)

- Fixed function stage, but configurable
- Fully symmetric
- Domains:
  Triangle, Quad, Isolines
- Spacing:
  Discrete, Continuous, Pow2

Tessellator (TS)



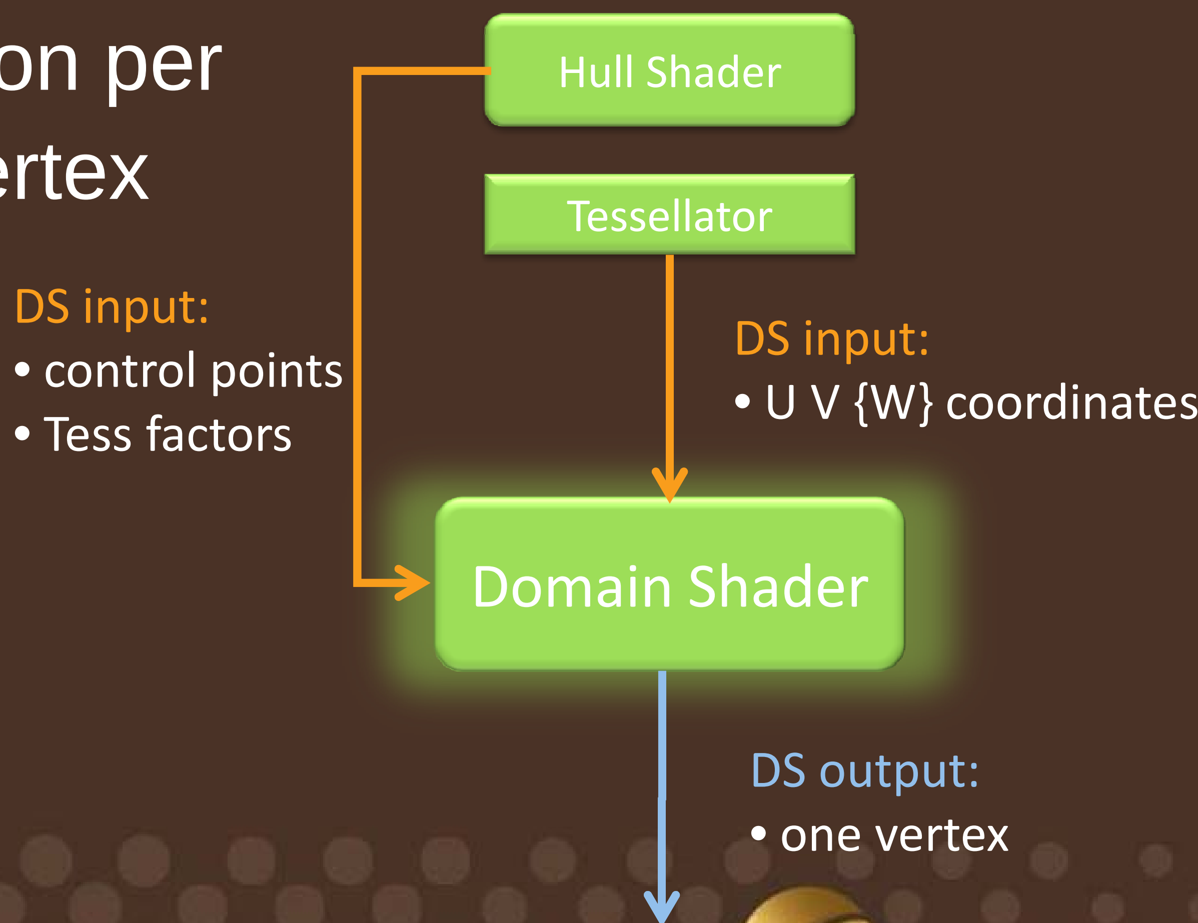
Level 5 Level 5.2 Level 5.8

Domain Shader (DS)

- Evaluate surface given parametric UV coordinates
- Interpolate attributes
- Apply displacements

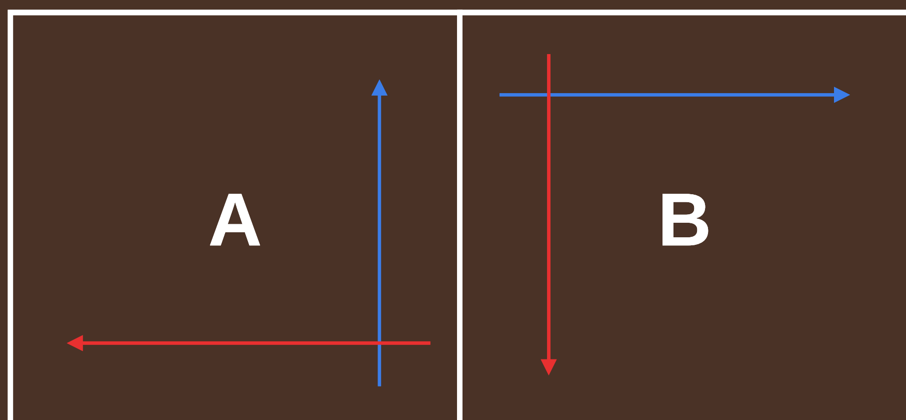
Domain Shader (DS)

- One invocation per generated vertex



Watertight Surface Evaluation

- Special care has to be taken to obtain watertight results (prevent cracks)
- All computations need to be the same in the U and V direction and be symmetric along the patch edges

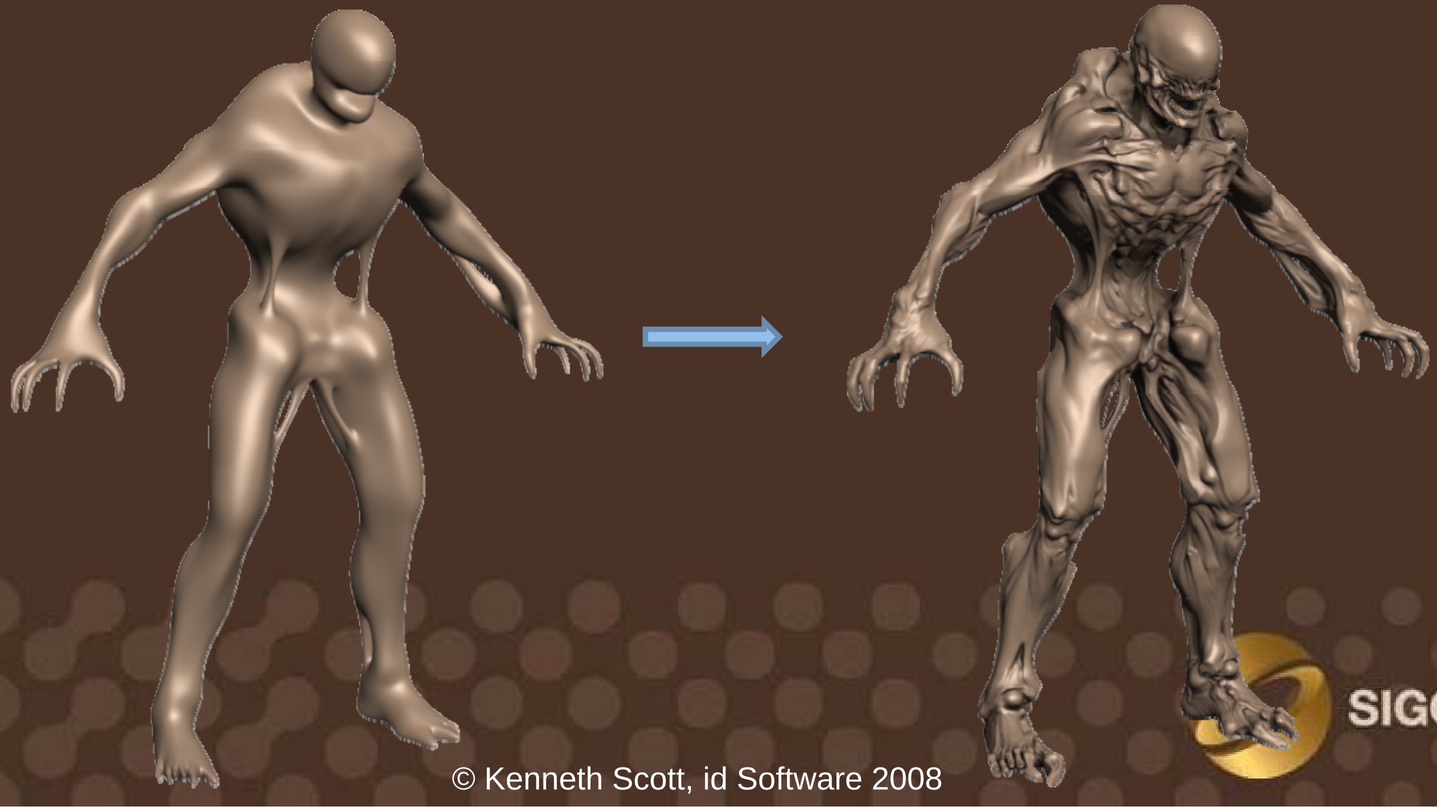


Floating Point Consistency

- FP addition is non commutative:
  – A + B + C + D != D + C + B + A
  – (A + B) + (C + D) == (D + C) + (B + A)
- FMA is not equivalent to MUL+ADD
  – A*a + B*b → FMA(A*a, B, b) != FMA(B*b, A, a)
- Beware of compiler optimizations
  – Use precise keyword

Displacement Mapping

- Sample displacement value from a texture
- Displace vertex along its normal

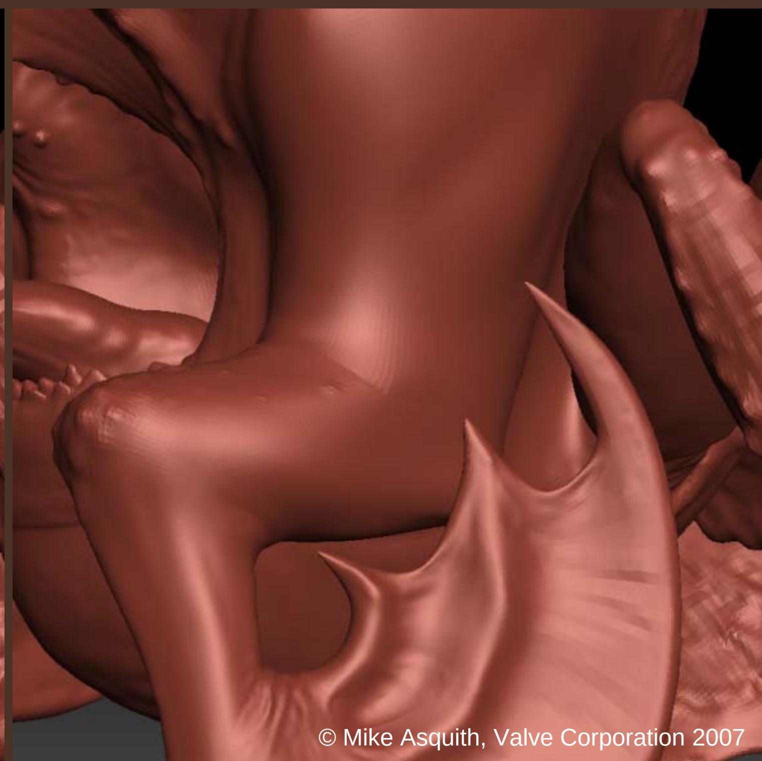


SIGGRAPH2008

Vector Displacements

- Native representation of most sculpting tools

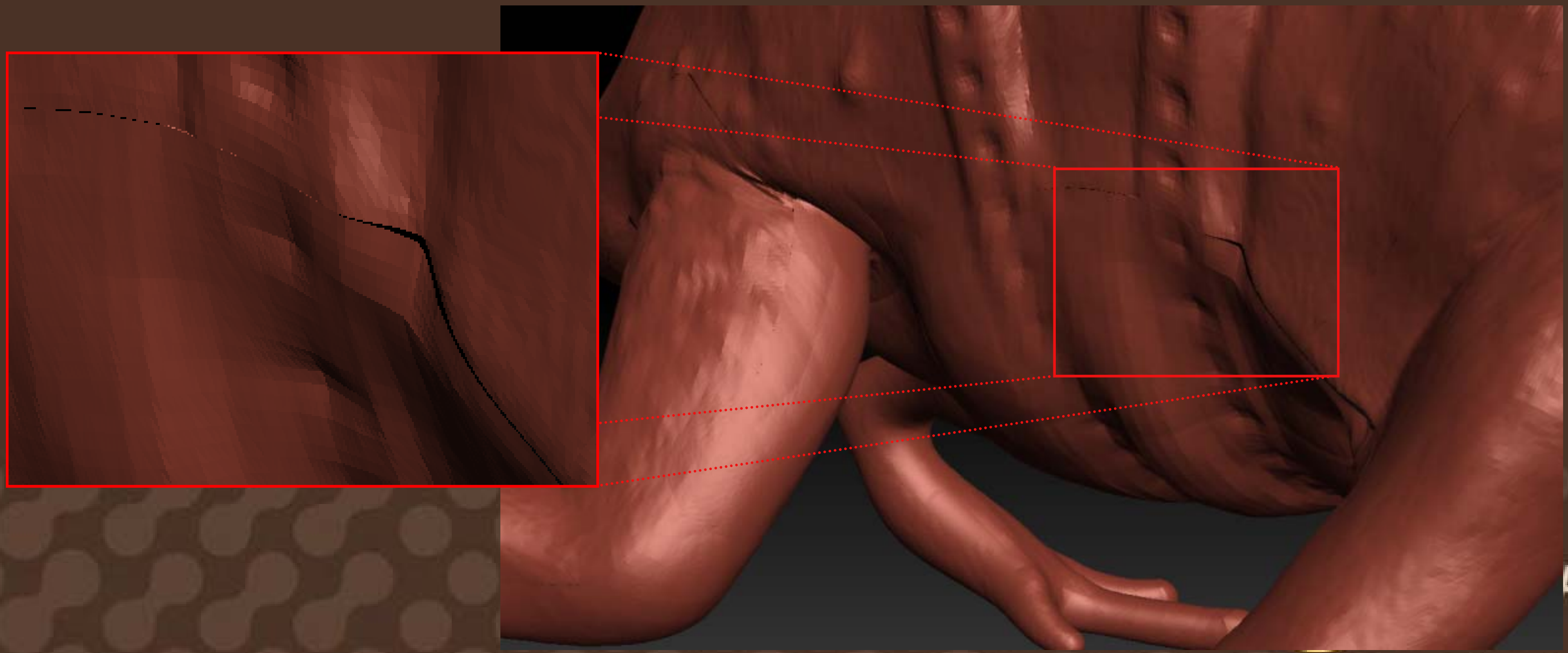
1D Displacements



3D Displacements

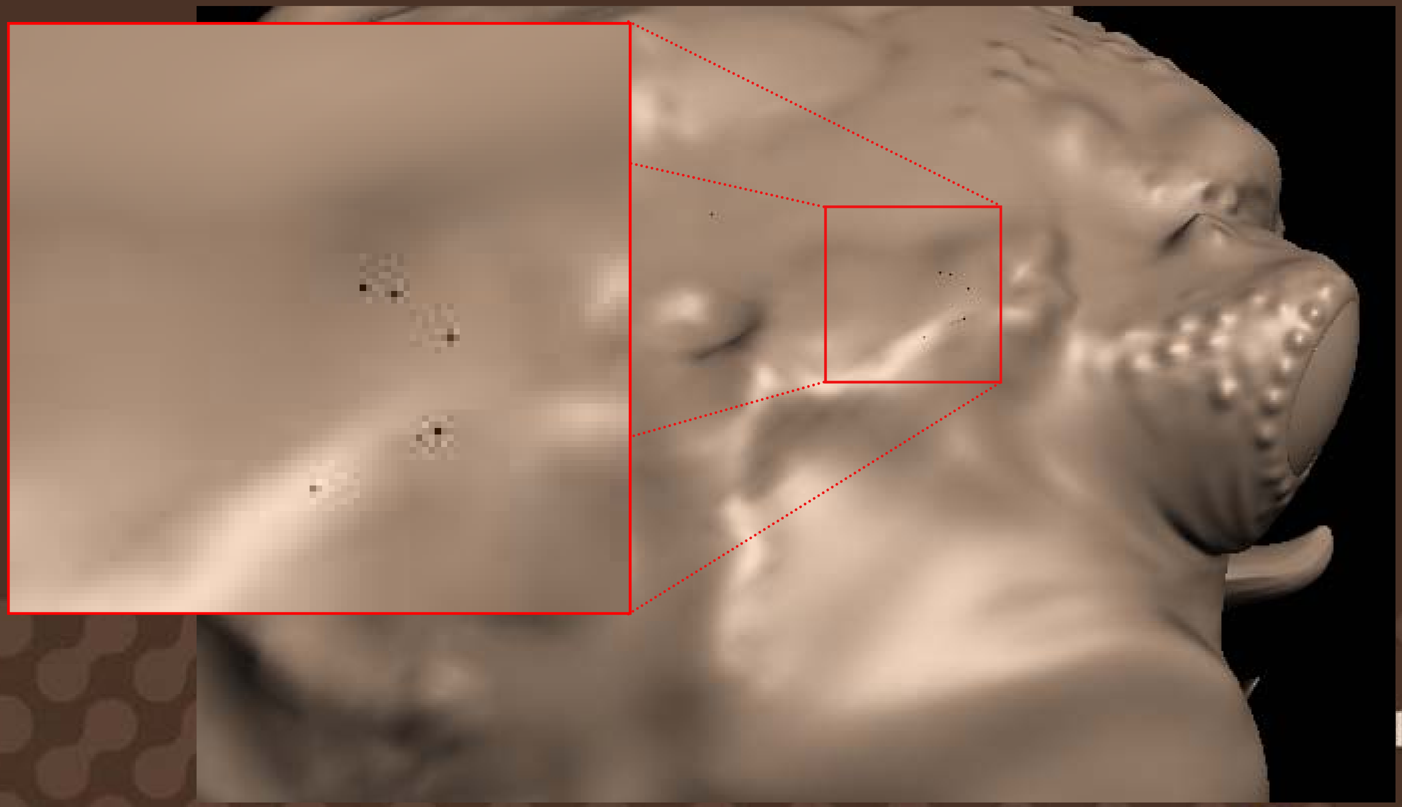
Watertight Texture Sampling

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



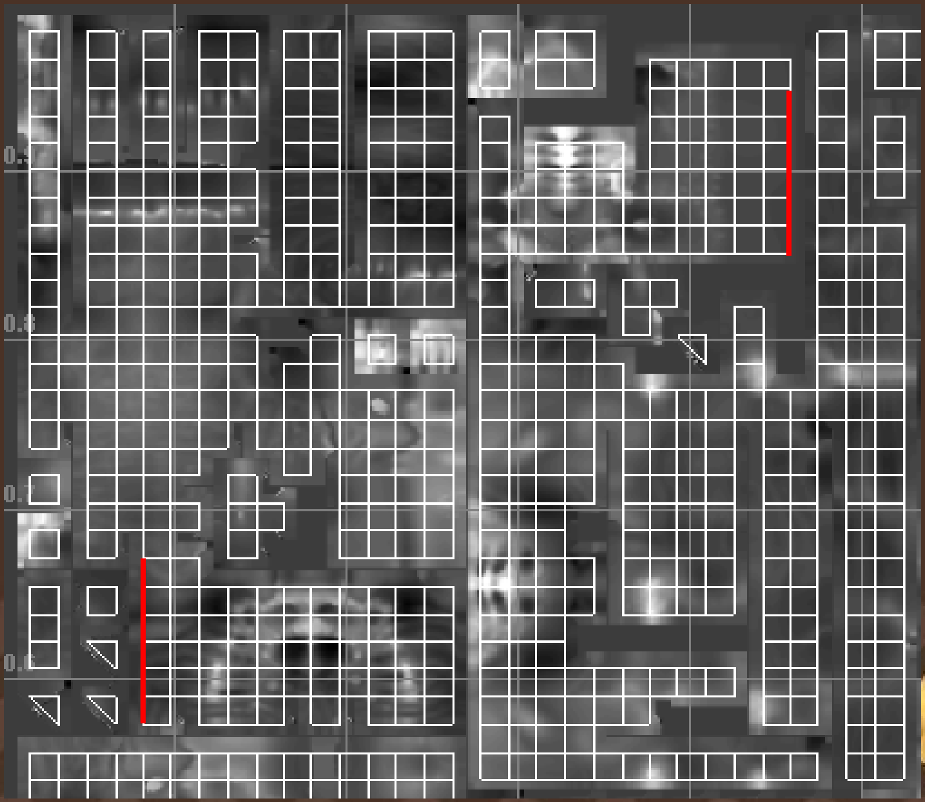
Watertight Texture Sampling

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



Watertight Texture Sampling

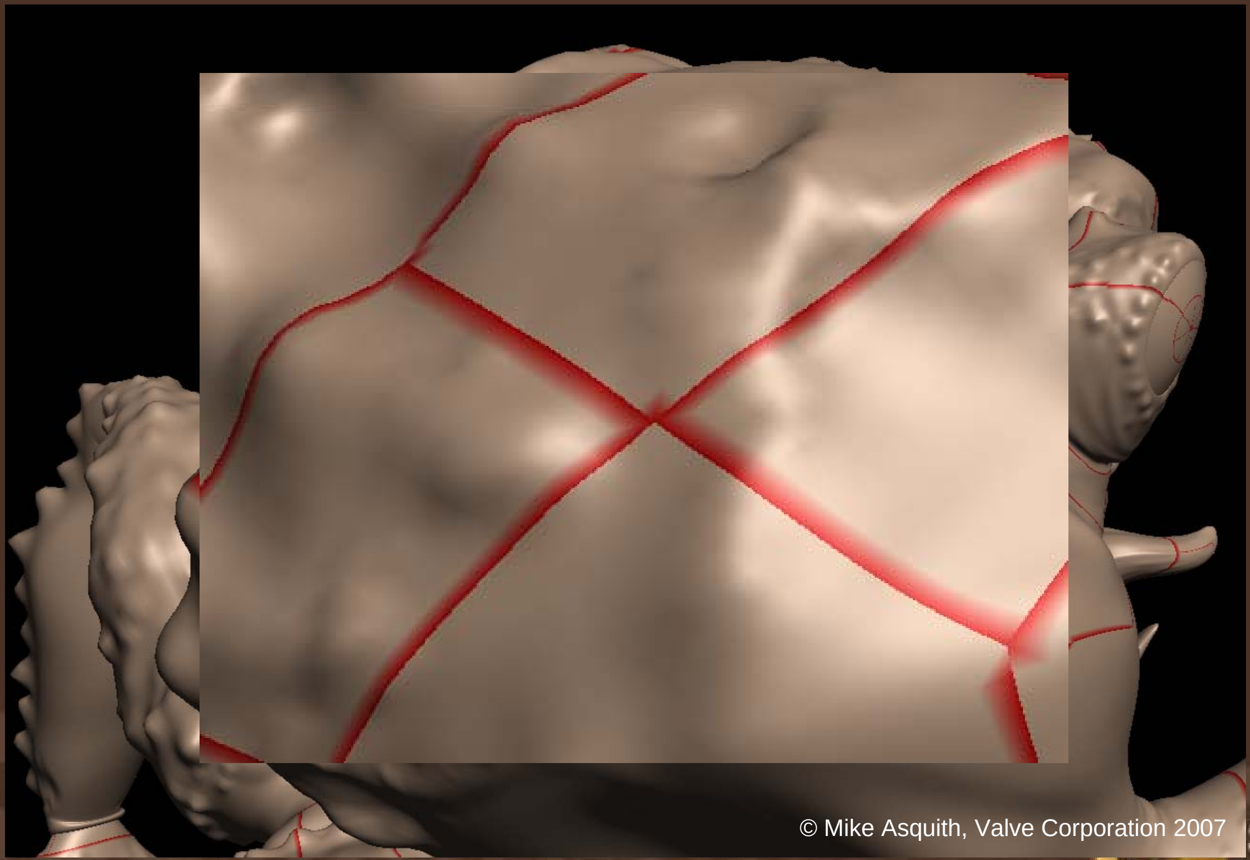
- Texture coordinate interpolation yields different result depending on location of the seam edges:



SIGGRAPH2008

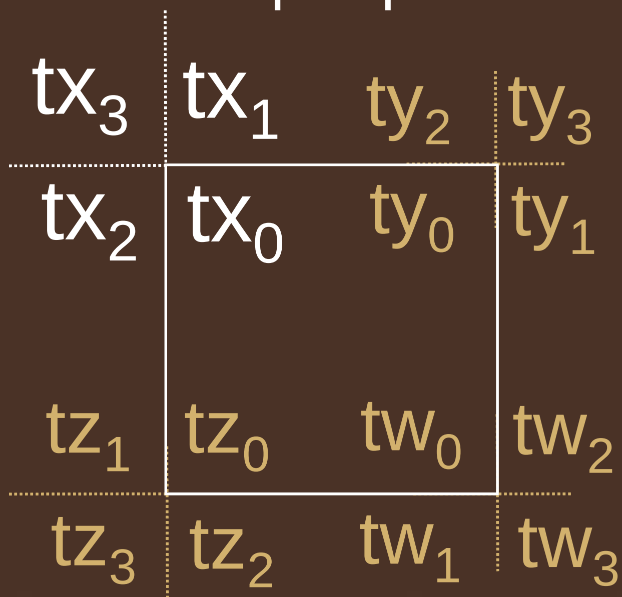
Watertight Texture Sampling

- Solution: define edge and corner ownership



Watertight Texture Sampling

- Store 4 texture coordinates per vertex
  - 16 per patch



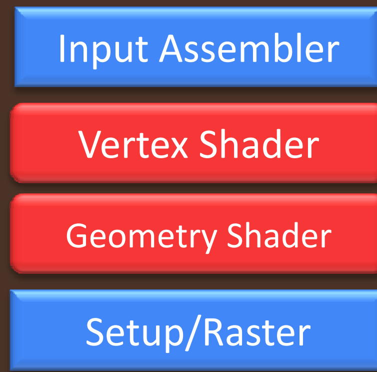

```
// float2 tx[4], ty[4], tz[4], tw[4];

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];
```

Implementation on Current HW

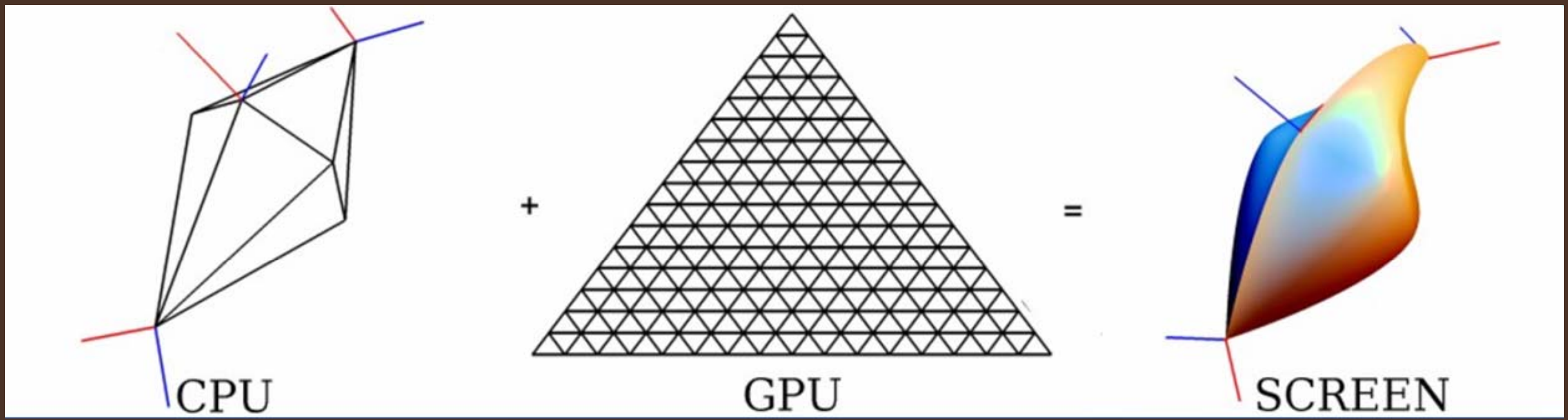
- Geometry Shader not designed for tessellation
  - Limited output size
  - Serial geometry generation



- Use geometry instancing instead!

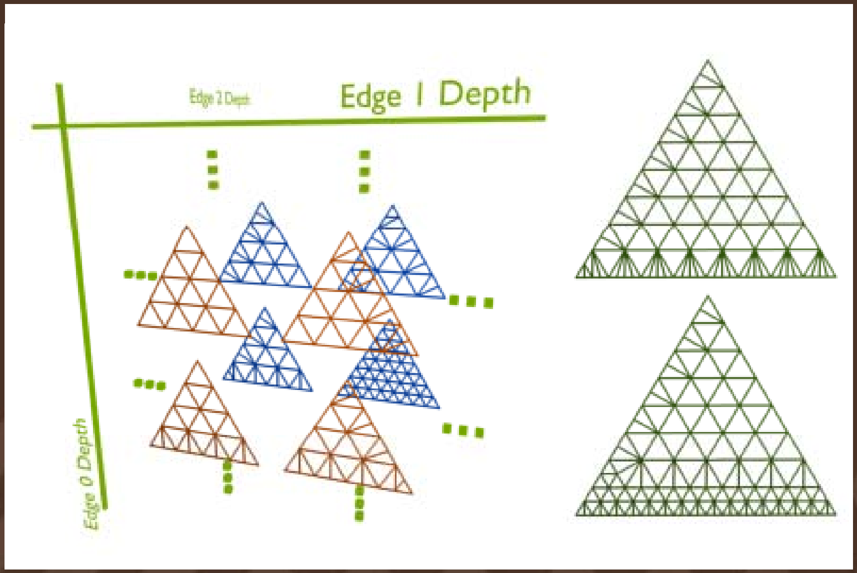
Instanced Tessellation Literature

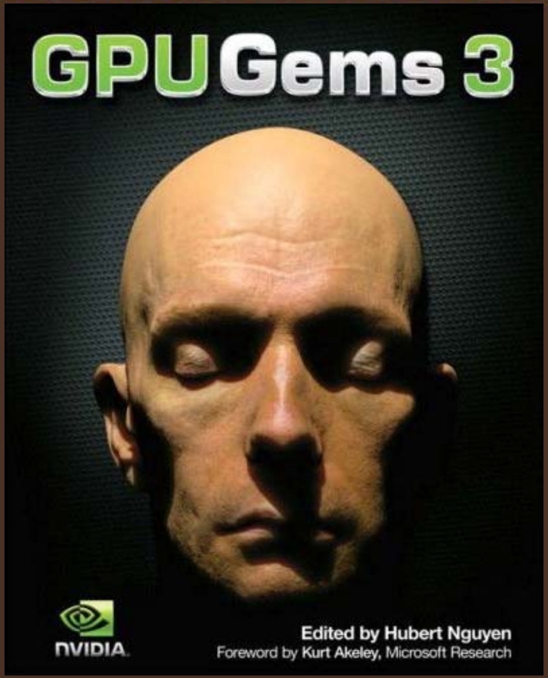
- Introduced by Boubekeur and Schlick
  - Generic Mesh Refinement On GPU
    http://iparla.labri.fr/publications/2005/BS05/GenericMeshRefinementOnGPU.pdf



Instanced Tessellation Literature

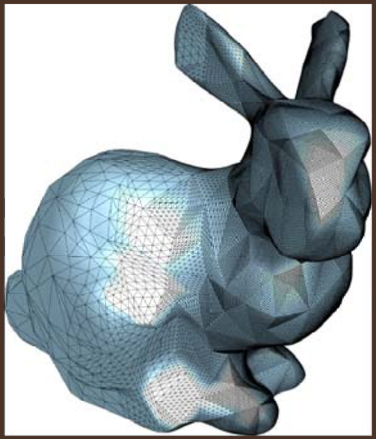
- Generic Adaptive Mesh Refinement, GPU Gems 3
- A Flexible Kernel for Adaptive Mesh Refinement on GPU
  http://iparla.labri.fr/publications/2008/BS08/ARK.pdf

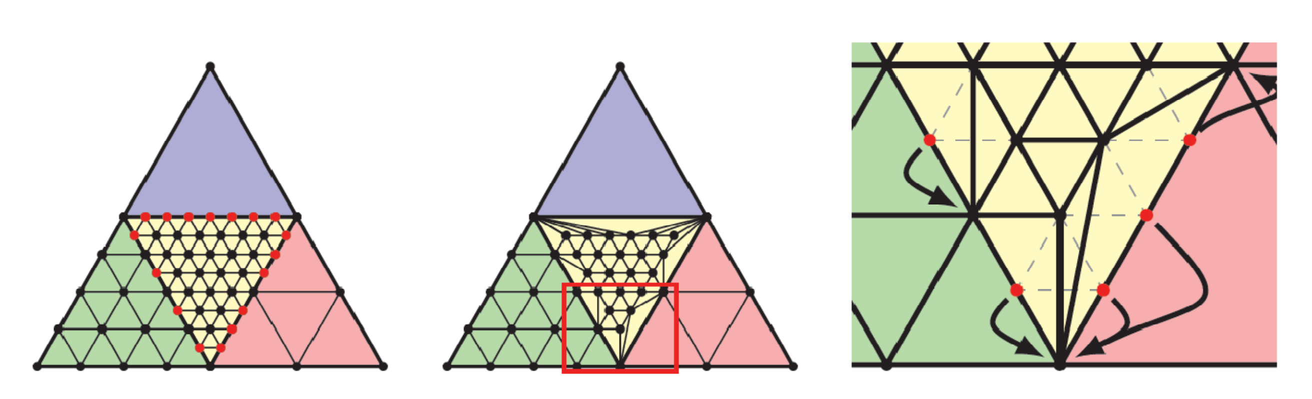




Instanced Tessellation Literature

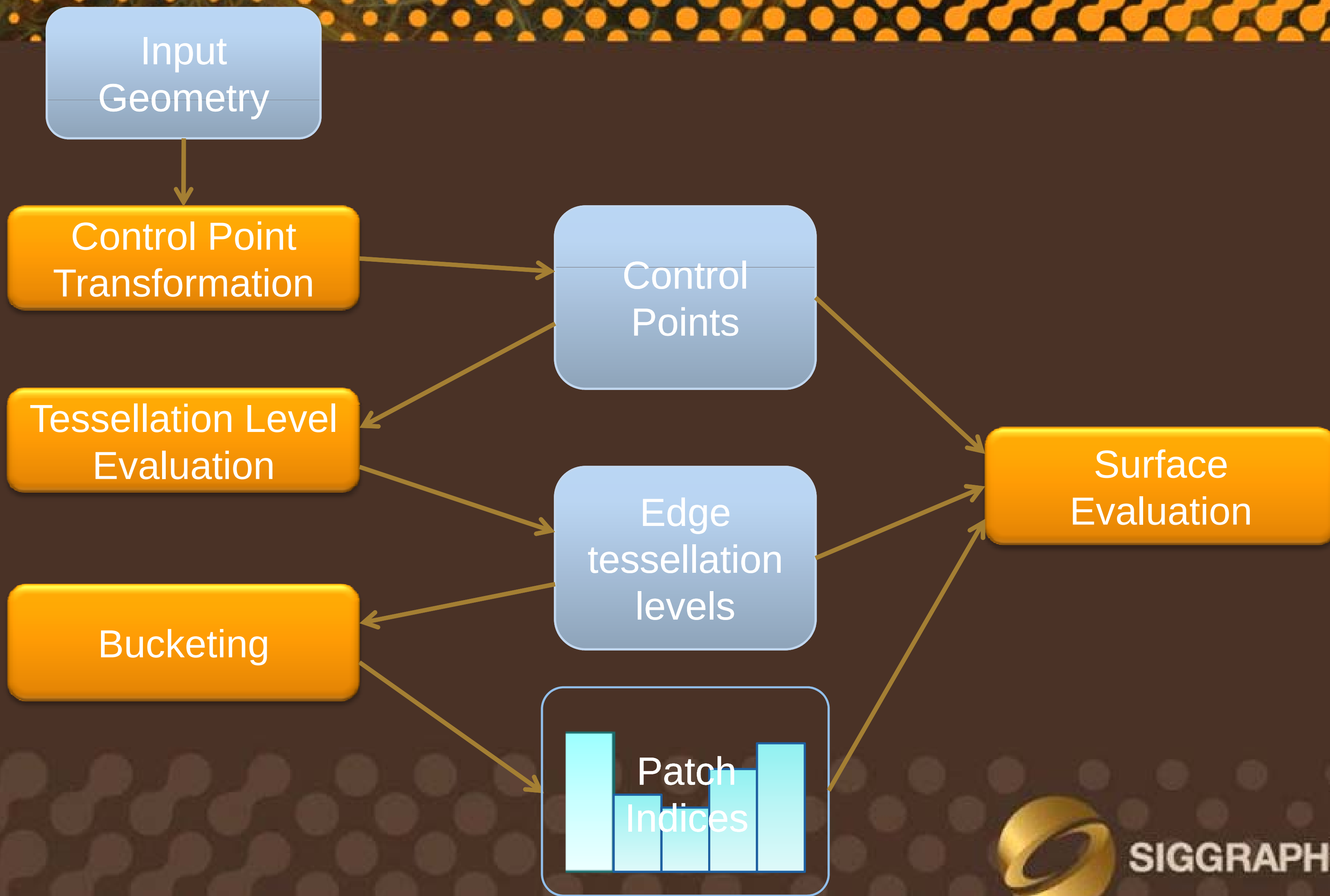
- Semi-uniform Adaptive Patch Tessellation
  C. Dyken, M. Reimers, J. Seland
  - 1D array of pre-computed tessellations
  - Edge vertices collapsed to nearest vertex common in both tessellations





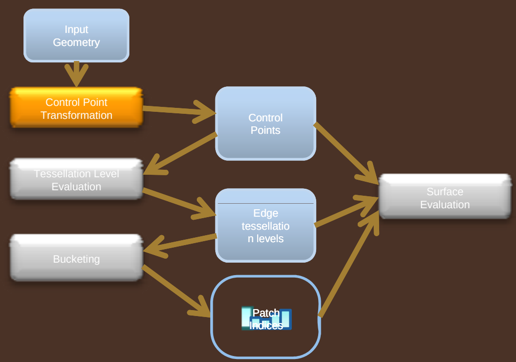
Dyken, Reimers & Seland, Univ. Oslo & SINTEF ICT

Instanced Tessellation Pipeline



Control Point Evaluation

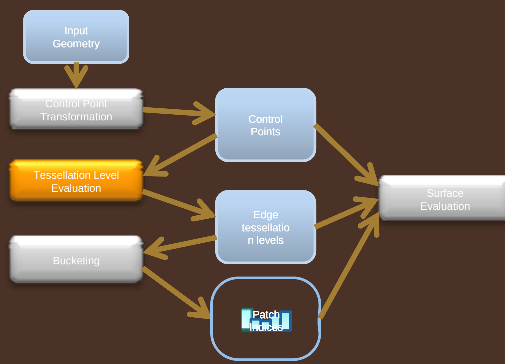
- Evaluated in the vertex shader
  - Render P groups of N vertices using instancing
    - P is number of patches
    - N is number of control points
  - Each vertex shader computes a separate control point
    - Use InstanceID to index patch
    - Use VertexID to index control point
  - Stream-out result



SIGGRAPH2008

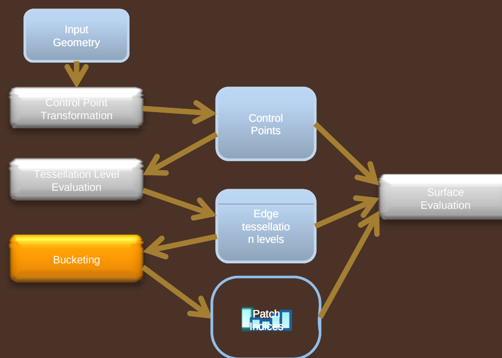
Tessellation Level Evaluation

- Evaluated in the vertex shader
  - Load control points produced by previous pass
  - Use one thread per patch
  - Evaluate desired LOD metric
  - Stream-out result



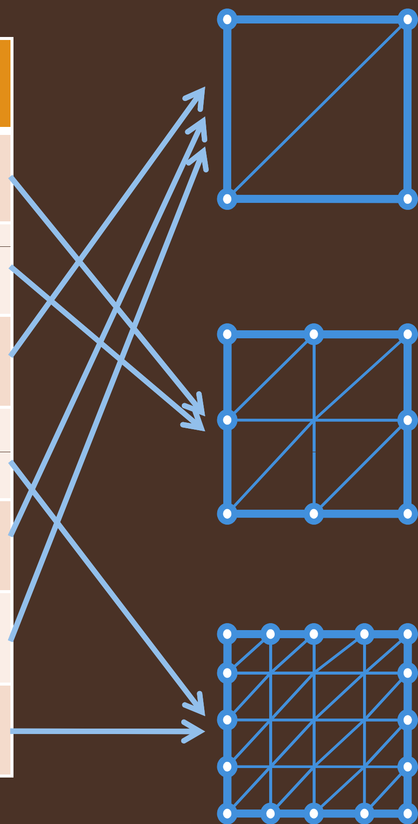
Bucketing

- Build list of patches for each tessellation level
  - Load edge tessellation levels produced by previous pass
  - Use MAX(edge level) to select the desired bucket
  - Append patch ID to bucket



Bucketing

Patch	Factor
0	2
1	2
2	1
3	4
4	1
5	1
6	4



Patch index buffer

2	4	5

Patch index buffer

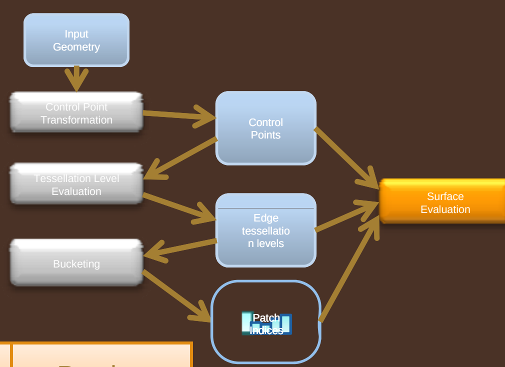
0	1

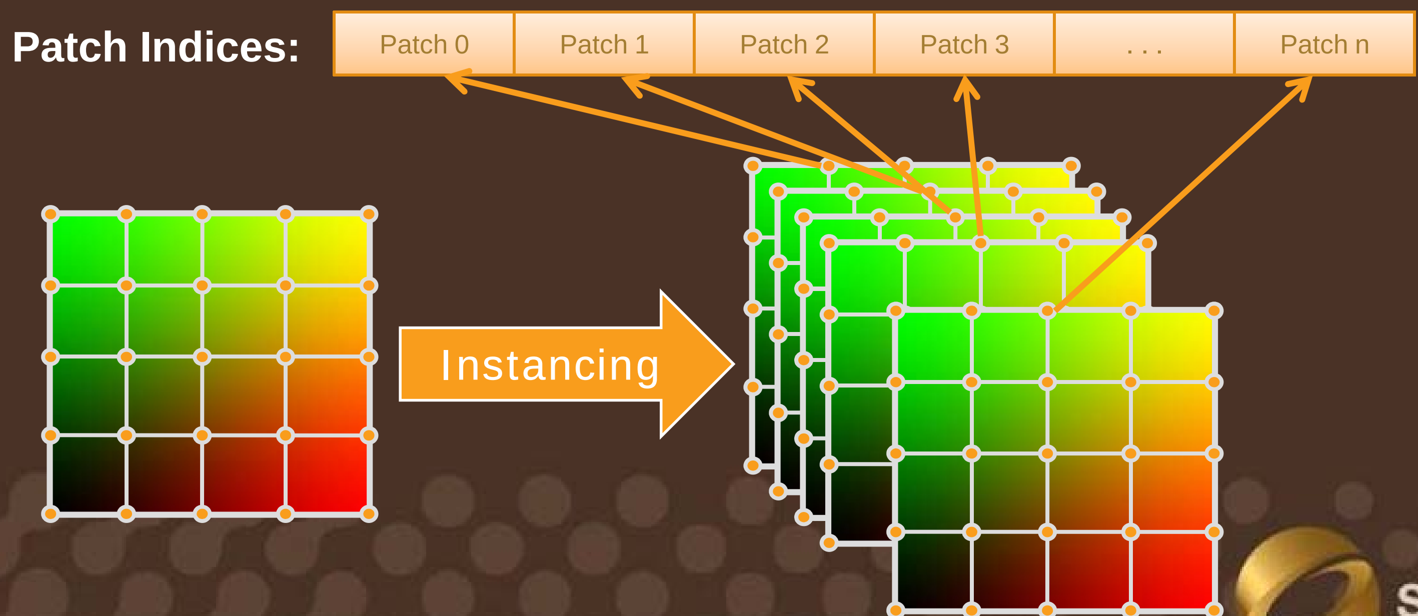
Patch index buffer

3	6

Surface Evaluation

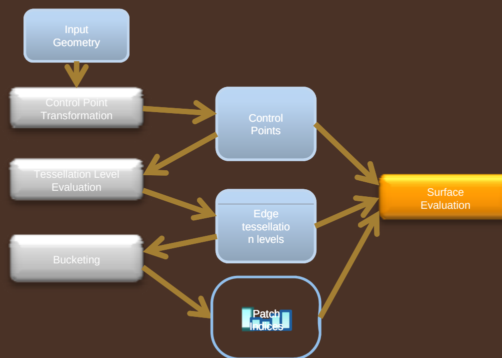
- One pass for each bucket
- Render pre-tessellated patch with instance count equal to patch count





Surface Evaluation

- Load patch index from bucket’s patch list
- Load edge tessellation levels to stitch boundaries
- Load control points to evaluate surface

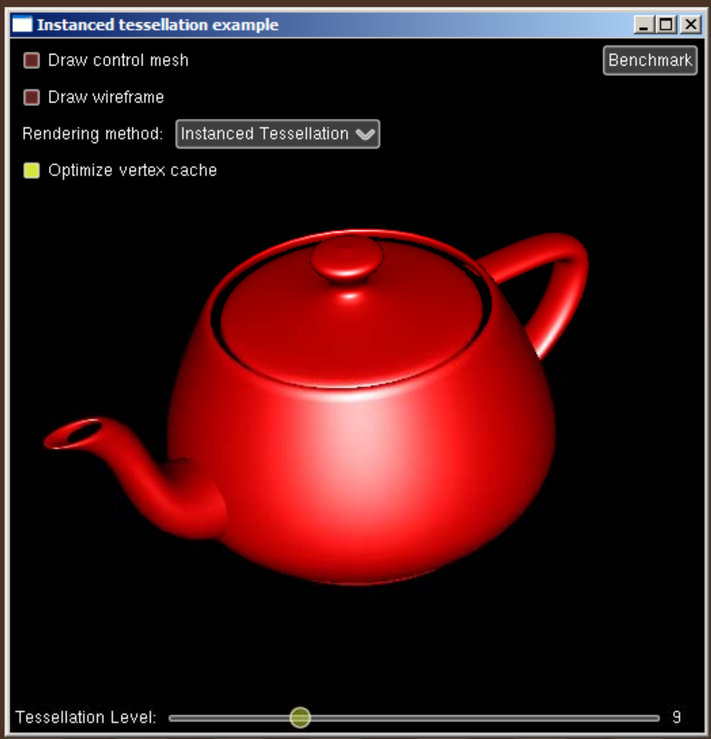


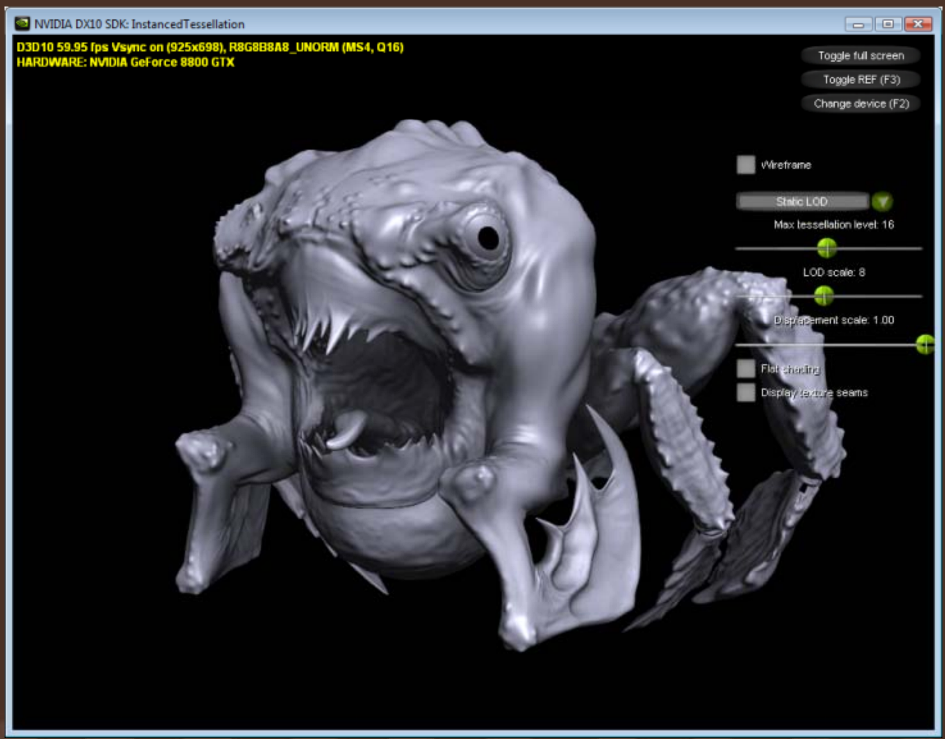
Instanced Tessellation Pipeline

- No continuous LODs
  - Tessellation patterns limited to POW2
  - Degenerate triangles generated for stitching
- Multiple passes required to compute data
- One draw call for each bucket

Instanced Tessellation

- Demos

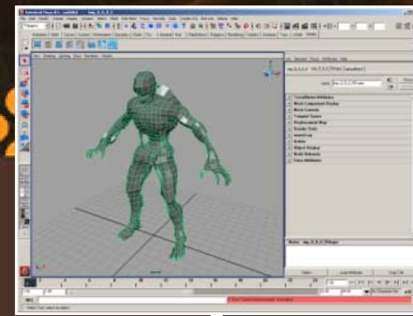




- Download at http://developer.nvidia.com

Production Pipeline

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

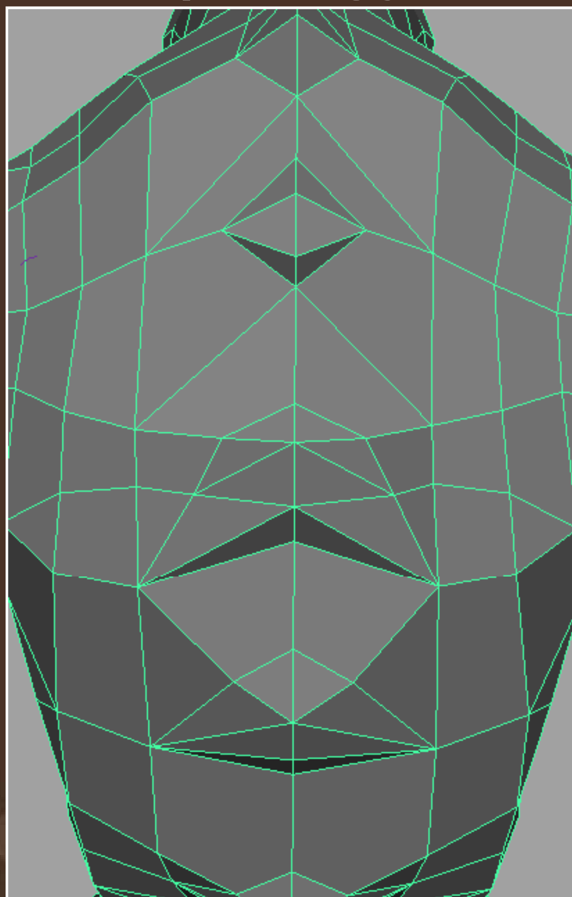


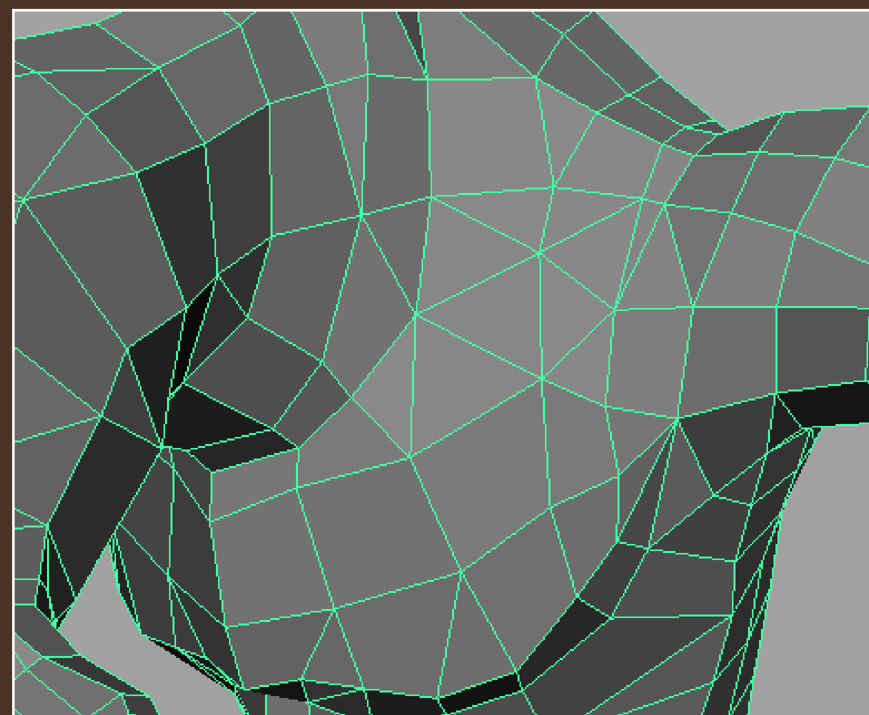
Modeling

- 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

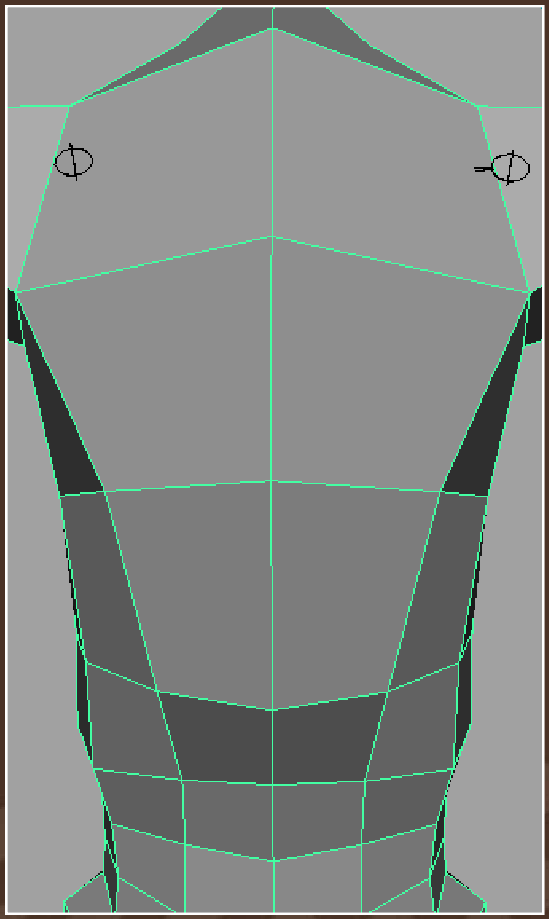
- 105 topology combinations

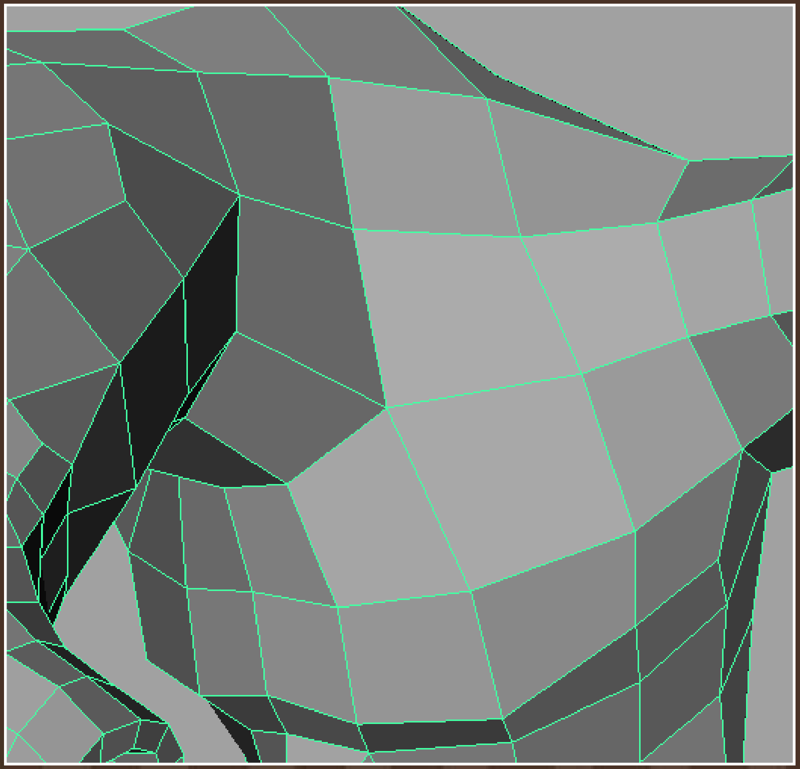




Topology Optimization

- 23 topology combinations

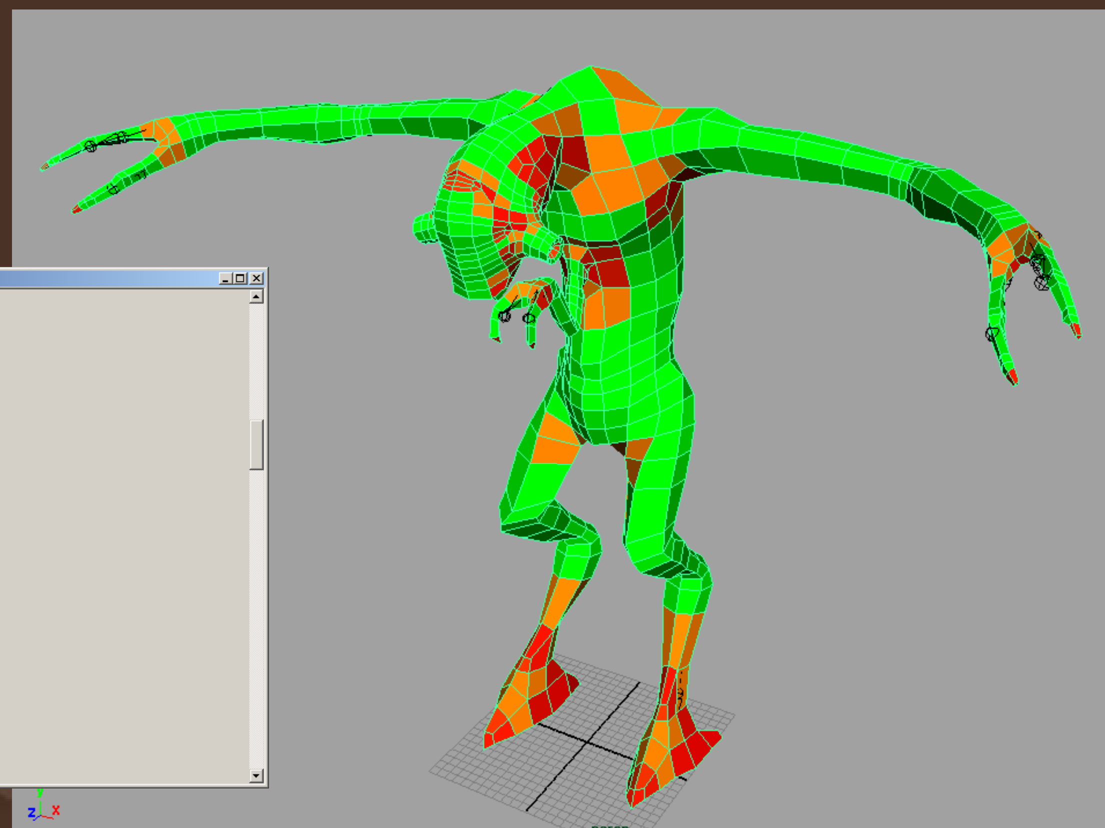


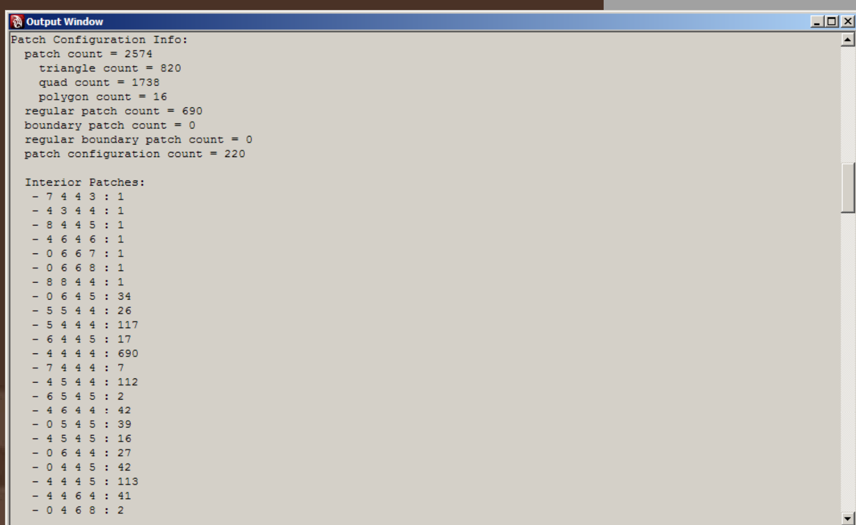


SIGGRAPH2008

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
```

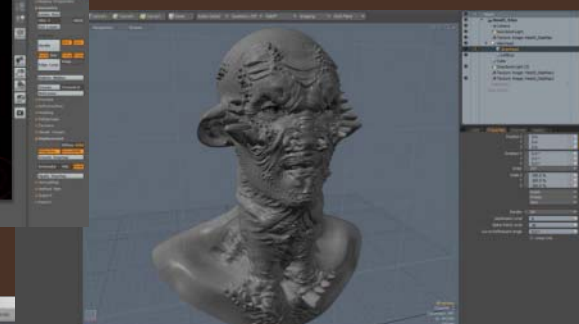
NVIDIA Mesh Processing Tool

- 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

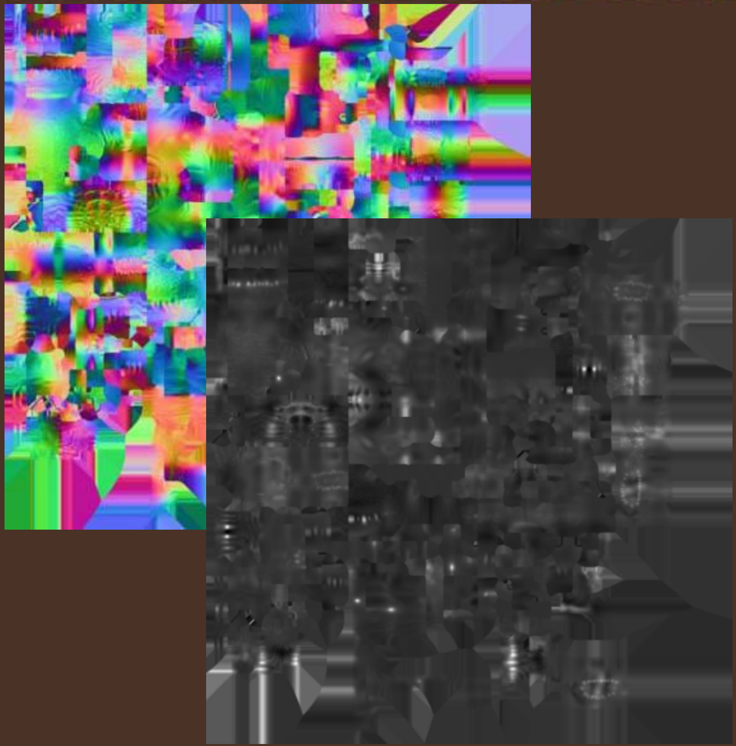
- Many tools available:
  - Autodesk® *Mudbox*™
  - Pixologic ZBrush®
  - modo™, Silo, Blender, etc.






Baker Tools

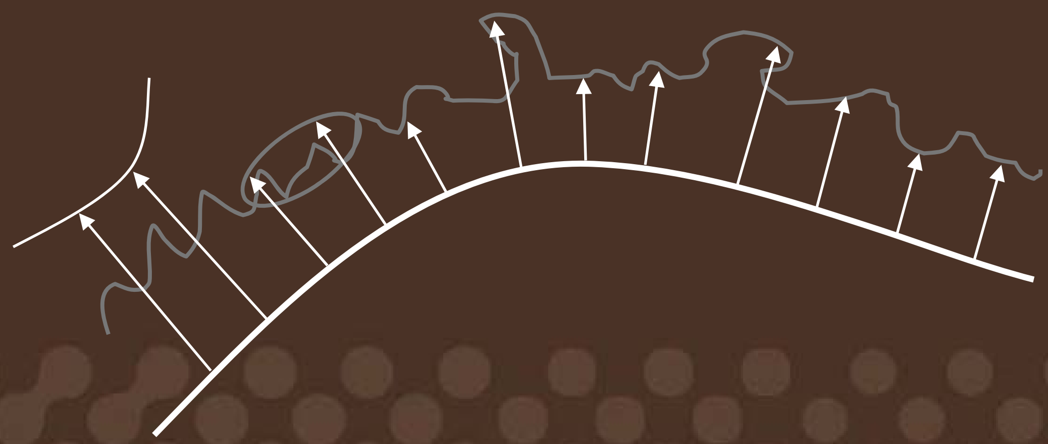
- Many options:
  - xNormal™
  - Mudbox™, ZMapper
  - Melody™, etc.
  - PolyBump™, etc.



- Two approaches
  - Ray casting
  - Dual parameterization

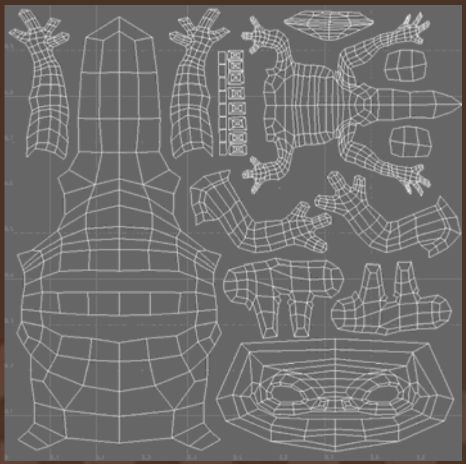
Capturing Attributes

- 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



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

- Ignacio Castaño icastano@nvidia.com