Hardware Subdivision and Tessellation of Catmull-Clark Surfaces

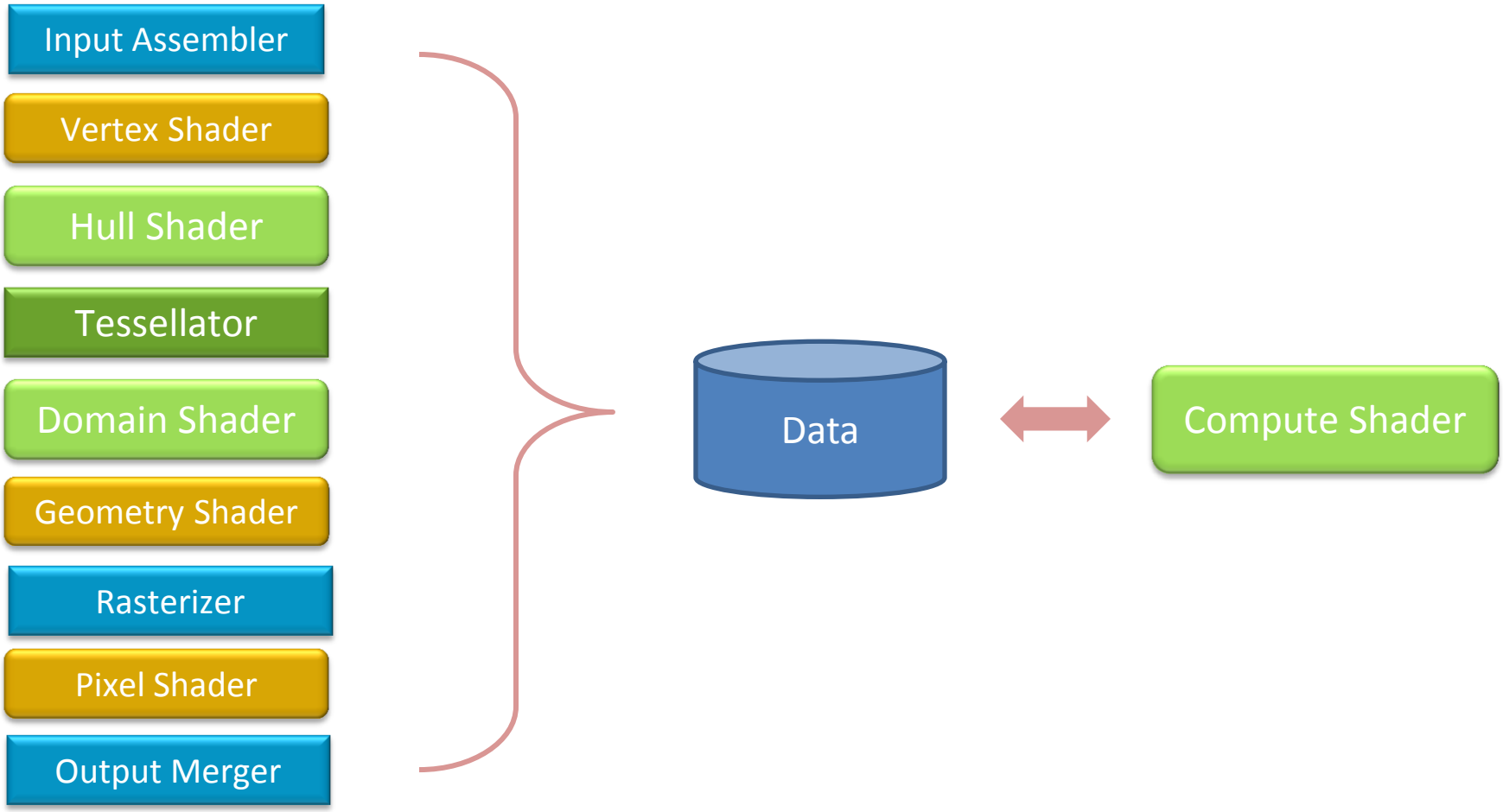
Charles Loop

Microsoft Research

Outline

- DirectX 11 Pipeline
- Subdivision Surfaces
- Gregory Patch Approximation
- GPU Subdivision

DirectX 11 Pipeline



DX Compute

- GPGPU-Computing
- Single shared context
- Cuda-like
  - HLSL Syntax:

Compute Shader

```
[numthreads(8, 1, 1)]
void Copy(uint3 blockIdx : SV_GroupID,
          uint3 DTid : SV_DispatchThreadID,
          uint3 threadIdx : SV_GroupThreadID,
          uint GI : SV_GroupIndex )
{
   Output[8*blockIdx.x + threadIdx.x] =
          Input[8*blockIdx.x + threadIdx.x];
}
```

Hardware Tessellation Pipeline

- Three new pipeline stages
  - Hull Shader (programmable)
  - Tessellator Unit (configurable)
  - Domain Shader (programmable)

Hull Shader (HS)

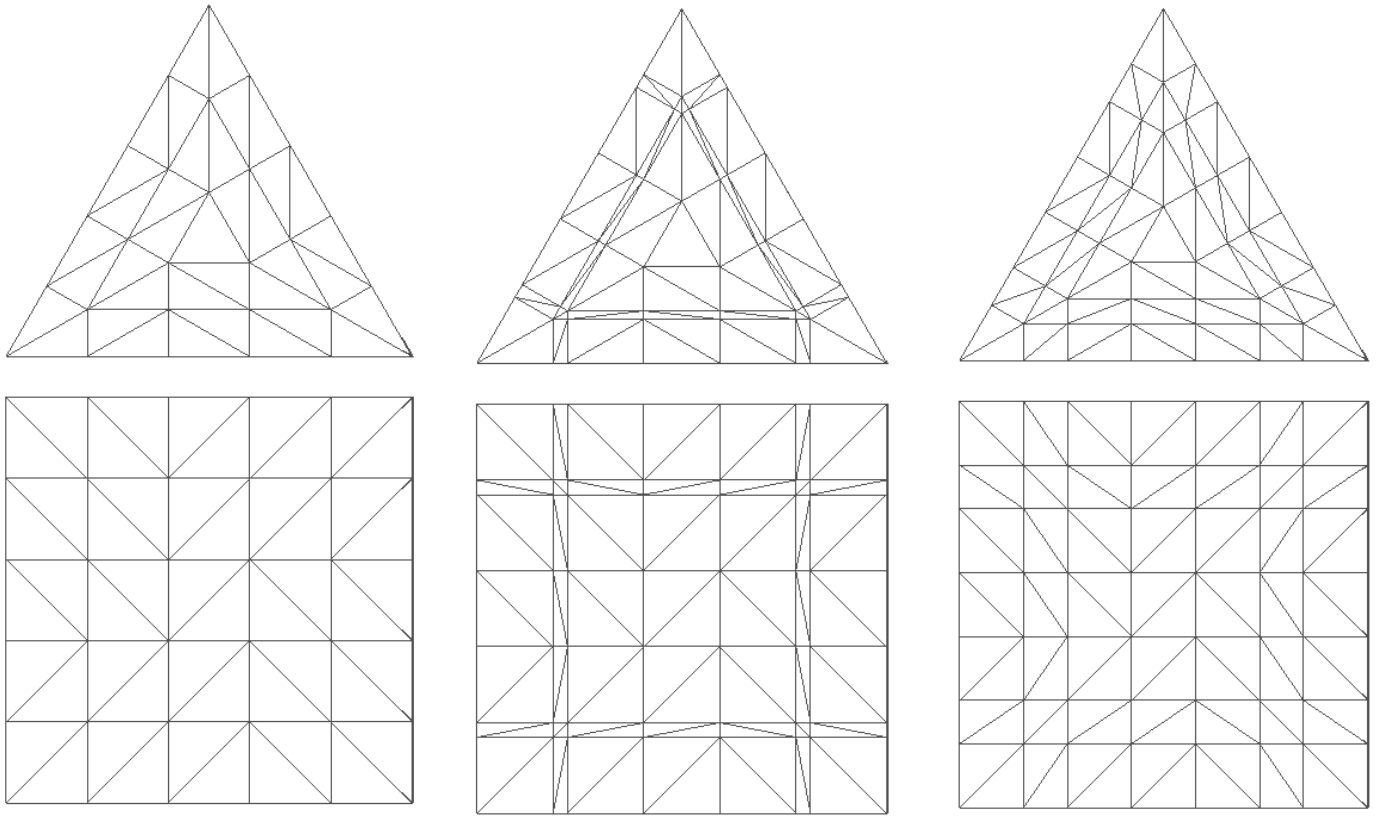
- Transform input control points
- One thread per output point.

- Hull Shader Constant Function
  - Compute edge and inside tessellation factors

Tessellator (TS)

- Fixed function stage, configurable
- Domains:
  - tri, quad, isoline
- Spacing:
  - integer, fractional, pow2

Tessellator (TS)



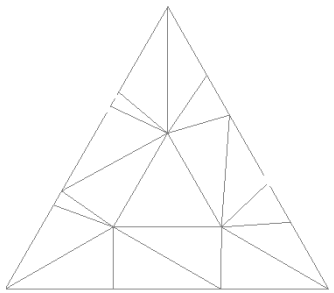
Level 5

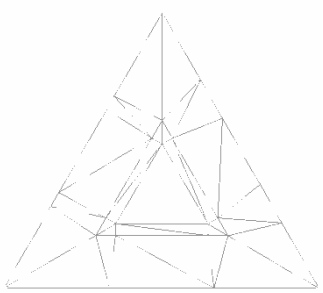
Level 5.4

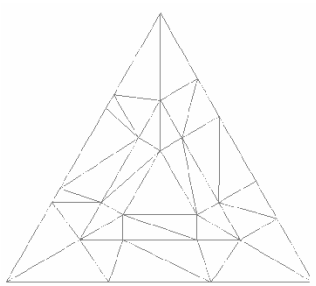
Level 6.6

Tessellator (TS)

Left = 3.5
Right = 4.4
Bottom = 3.0

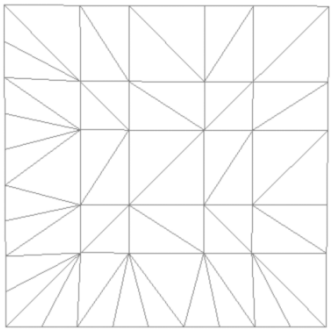


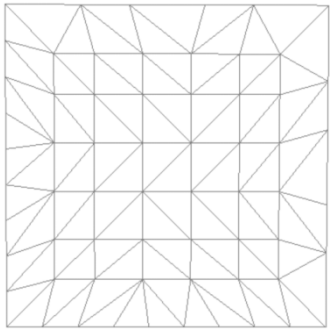


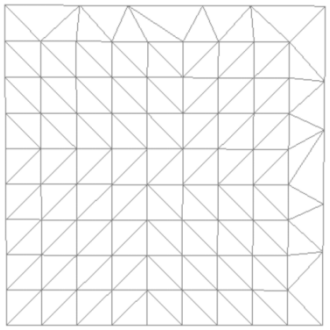


Top,Right = 4.5

Bottom,Left = 9.0







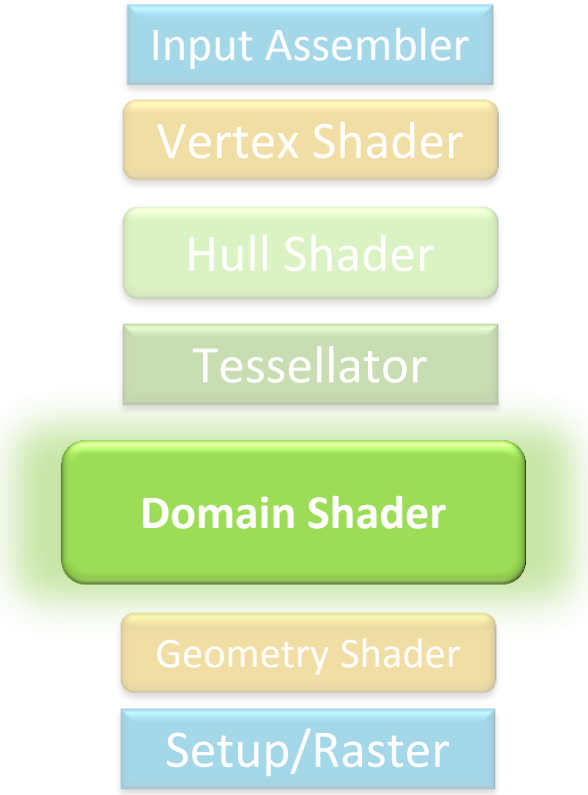
Inside Tess: minimum

Inside Tess: average

Inside Tess: maximum

Domain Shader (DS)

- Evaluate patch with input parameters $u, v, [w]$
- Interpolate attributes
- Apply displacements
- Output position, normal, etc.



Bitwise Consistent Evaluation

- Patches share boundaries
  - Coincident vertices evaluated via different code paths
  - Floating point addition non-commutative

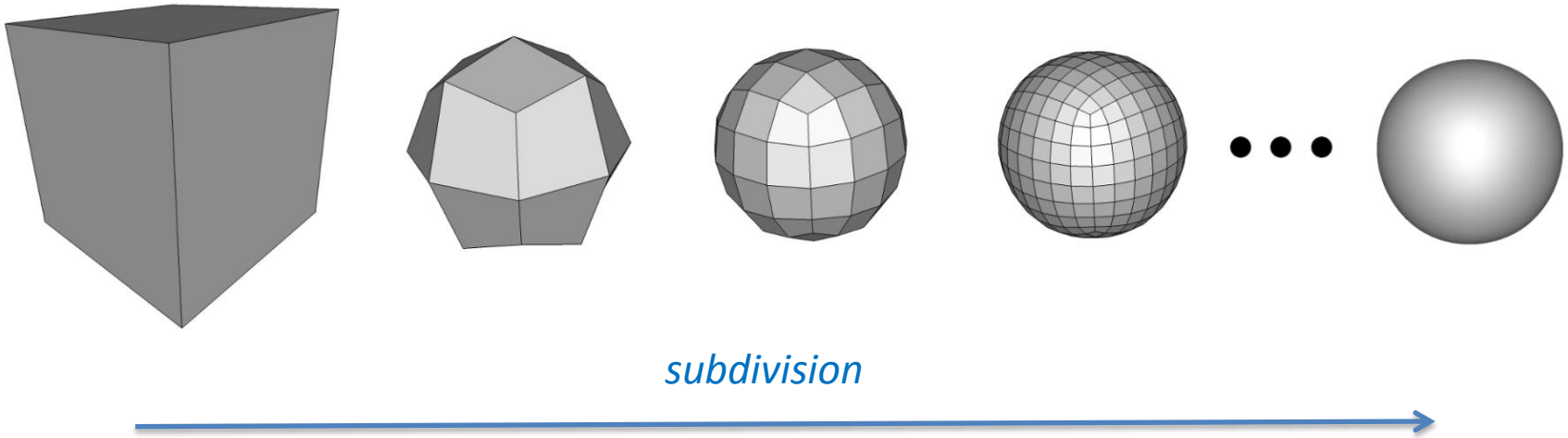
Solution: symmetric or order consistent evaluation

- References http://developer.nvidia.com

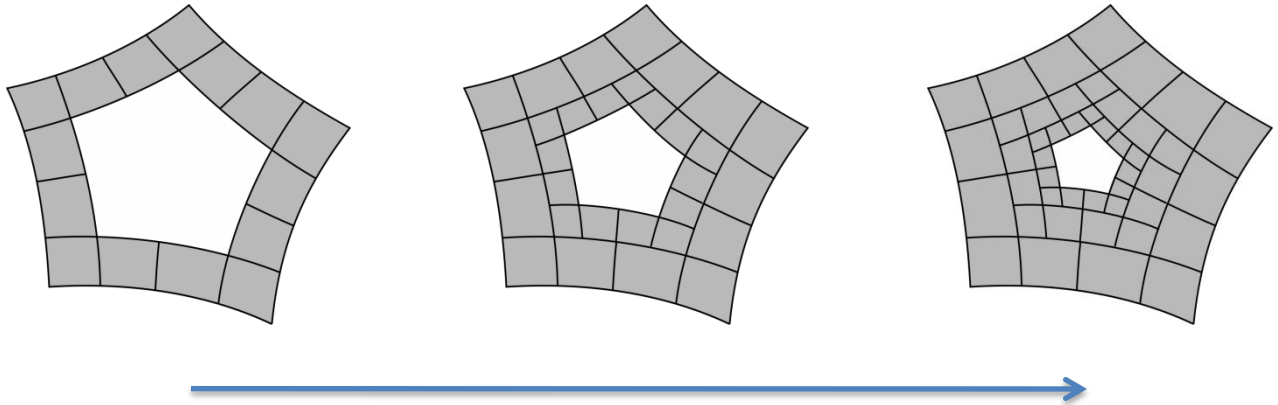
Subdivision Surfaces

Catmull-Clark Subdivision

Catmull, E. AND Clark, J. 1978,
Recursively generated B-spline surfaces on arbitrary topological meshes



Problem: Infinite number of patches

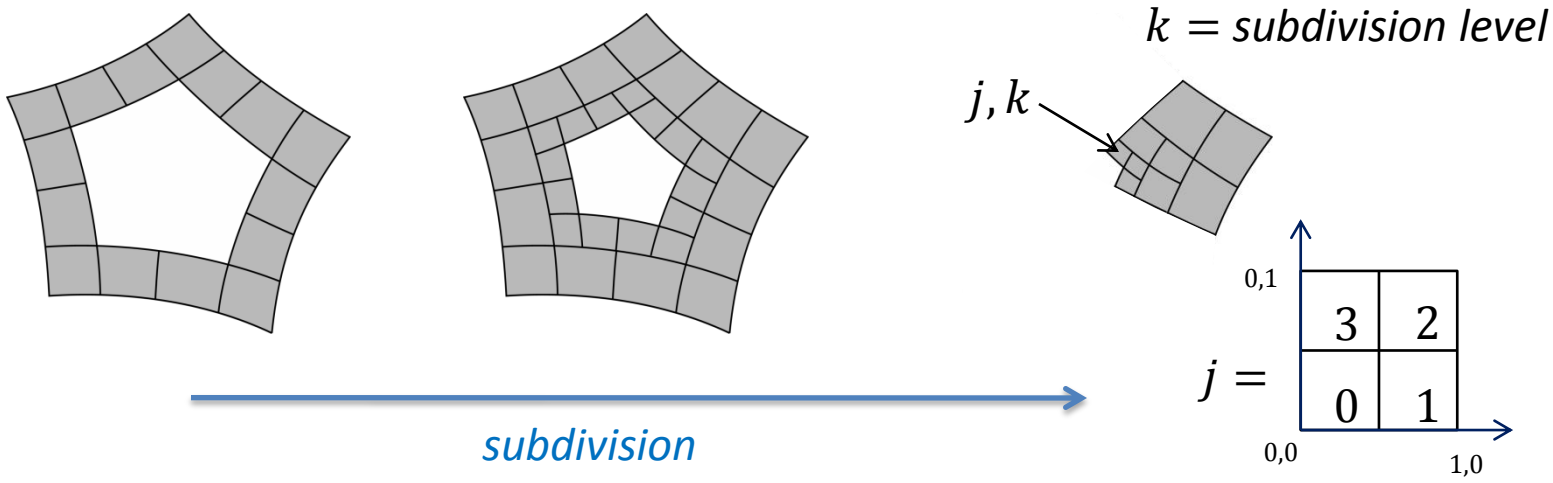


- Poor fit to hardware tessellation paradigm

- Stam, J. 1998, *Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values*

- Exact evaluation is possible, but slow

Problem: Infinite number of patches



- Poor fit to hardware tessellation paradigm

- Stam, J. 1998,
- *Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values*

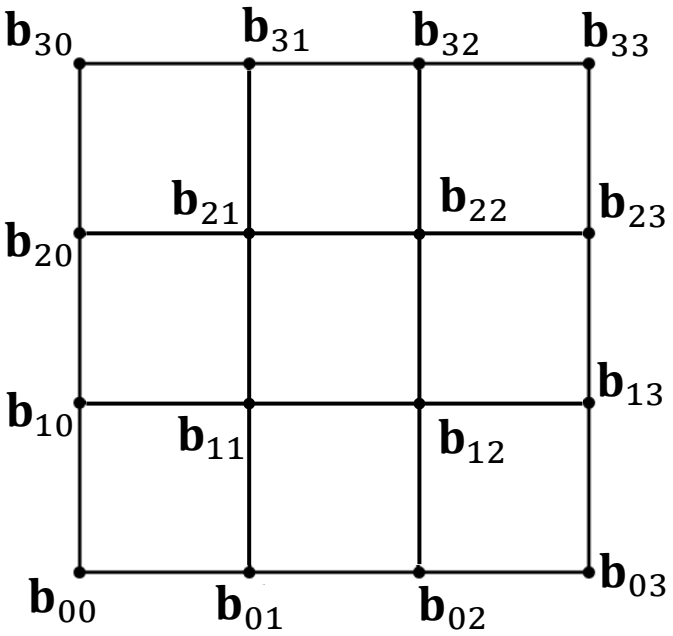
- Exact evaluation is possible, but slow

Approximate Subdivision Surfaces

Approximate Subdivision Surfaces

Loop, C. , Schaefer, S. Ni, T., AND Castaño, I. 2009,
Approximating subdivision surfaces with Gregory patches for tessellation hardware

Bicubic Bézier Patch

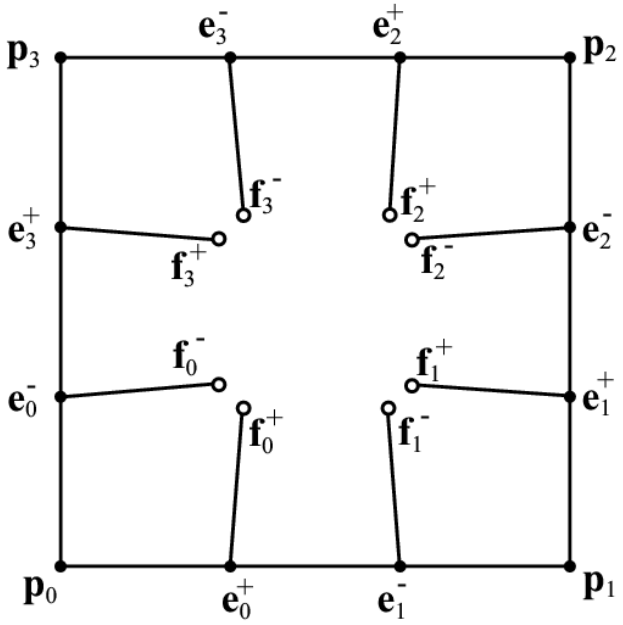


$$B(u,v) = \mathrm{b}^3(u) \cdot \mathbf{B} \cdot \mathrm{b}^3(v)$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{b}_{00} & \mathbf{b}_{01} & \mathbf{b}_{02} & \mathbf{b}_{03} \\ \mathbf{b}_{10} & \mathbf{b}_{11} & \mathbf{b}_{12} & \mathbf{b}_{13} \\ \mathbf{b}_{20} & \mathbf{b}_{21} & \mathbf{b}_{22} & \mathbf{b}_{23} \\ \mathbf{b}_{30} & \mathbf{b}_{31} & \mathbf{b}_{32} & \mathbf{b}_{33} \end{bmatrix}$$

$$\mathrm{b}^3(u) = [(1-u)^3 \quad 3(1-u)^2u \quad 3(1-u)u^2 \quad u^3\,]$$

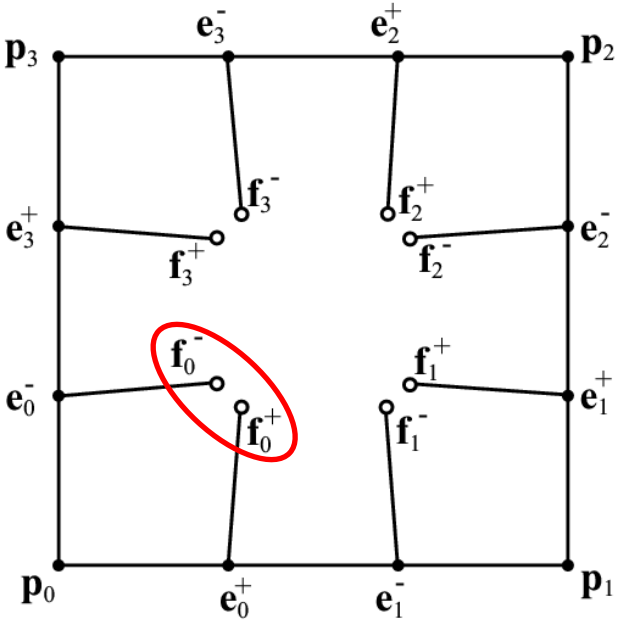
Gregory Quad Patch



$$Q(u,v) = \mathrm{b}^3(u) \cdot \mathbf{G}(u,v) \cdot \mathrm{b}^3(v)$$

$$\mathbf{G}(u,v) = \begin{bmatrix} \mathbf{p}_3 & \mathbf{e}_0^- & \mathbf{e}_3^+ & \mathbf{p}_2 \\ \mathbf{e}_0^+ & \mathbf{F}_0(u,v) & \mathbf{F}_3(u,v) & \mathbf{e}_3^- \\ \mathbf{e}_1^- & \mathbf{F}_1(u,v) & \mathbf{F}_2(u,v) & \mathbf{e}_2^+ \\ \mathbf{p}_0 & \mathbf{e}_1^+ & \mathbf{e}_2^- & \mathbf{p}_1 \end{bmatrix}$$

Gregory Quad Patch



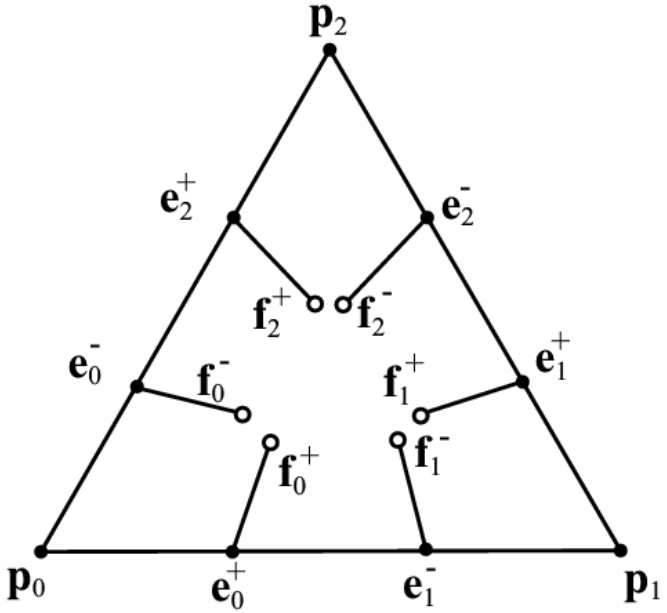
$$Q(u,v) = \mathrm{b}^3(u) \cdot \mathbf{G}(u,v) \cdot \mathrm{b}^3(v)$$

$$\mathbf{G}(u,v) = \begin{bmatrix} \mathbf{p}_3 & \mathbf{e}_0^- & \mathbf{e}_3^+ & \mathbf{p}_2 \\ \mathbf{e}_0^+ & \mathbf{F}_0(u,v) & \mathbf{F}_3(u,v) & \mathbf{e}_3^- \\ \mathbf{e}_1^- & \mathbf{F}_1(u,v) & \mathbf{F}_2(u,v) & \mathbf{e}_2^+ \\ \mathbf{p}_0 & \mathbf{e}_1^+ & \mathbf{e}_2^- & \mathbf{p}_1 \end{bmatrix}$$

$$\mathbf{F}_0(u,v) = \frac{u\,\mathbf{f}_0^+ + v\,\mathbf{f}_0^-}{u+v}$$

$\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3$ are similar

Gregory Triangle Patch

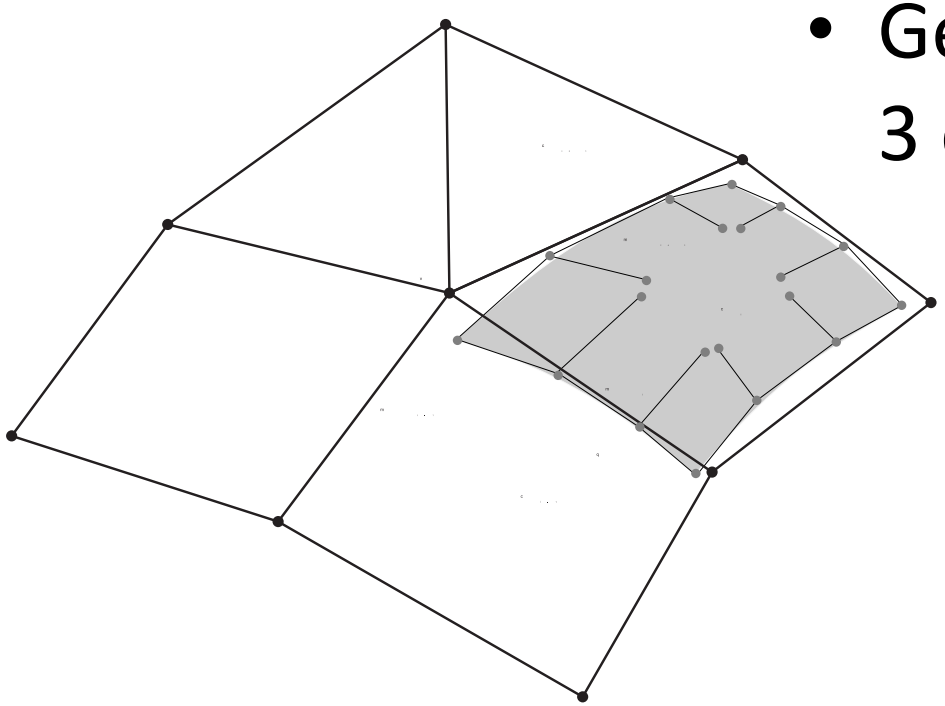


$$
\begin{aligned}
T(u,v,w) = \quad & u^3\mathbf{p}_0 + v^3\mathbf{p}_1 + w^3\mathbf{p}_2 \\
+ \quad & 3uv(u+v)(u\mathbf{e}_0^+ + v\mathbf{e}_1^-) \\
+ \quad & 3vw(v+w)(v\mathbf{e}_1^+ + w\mathbf{e}_2^-) \\
+ \quad & 3wu(w+u)(w\mathbf{e}_2^+ + u\mathbf{e}_0^-) \\
+ \quad & 12uvw(u\mathbf{F}_0 + v\mathbf{F}_1 + w\mathbf{F}_2)
\end{aligned}
$$

$$
\mathbf{F}_0(v,w) = \frac{w\,\mathbf{f}_0^- + v\,\mathbf{f}_0^+}{v+w}, \quad \mathbf{F}_1(u,w) = \frac{u\,\mathbf{f}_1^- + w\,\mathbf{f}_1^+}{w+u}, \quad \mathbf{F}_2(u,v) = \frac{v\,\mathbf{f}_2^- + u\,\mathbf{f}_2^+}{u+v}
$$

Geometric Construction

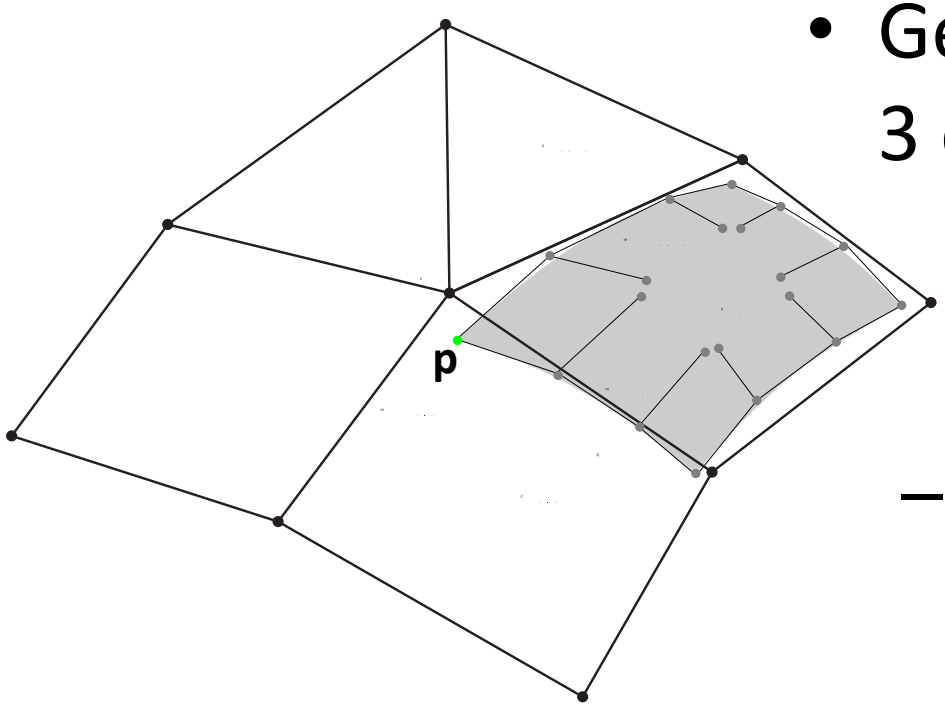
Patch Construction



- General construction for 3 or 4 sided faces

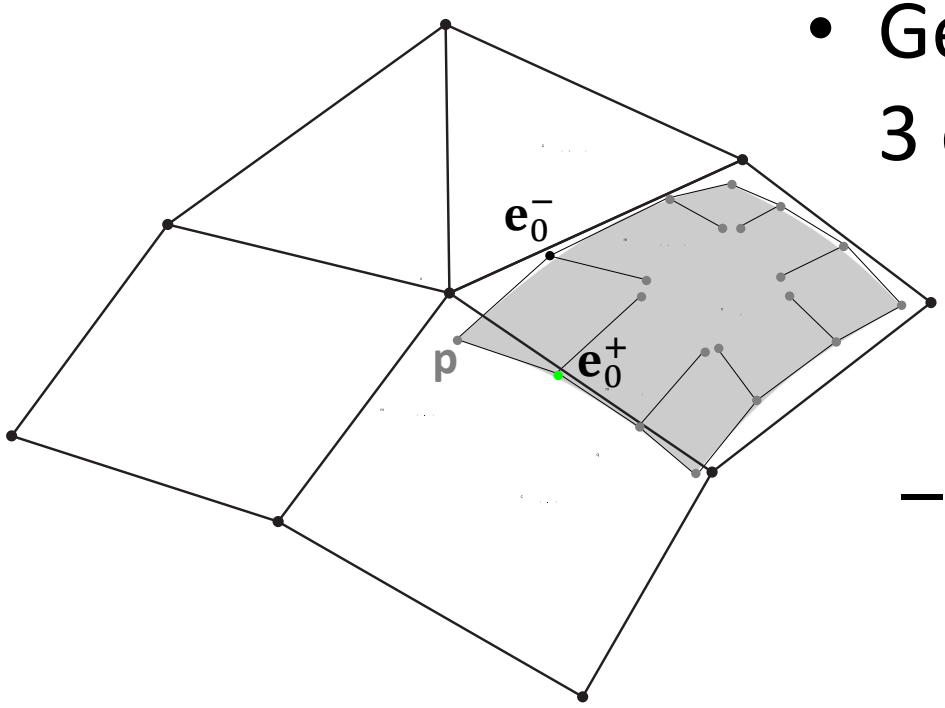
Gregory patches in 1-1 correspondence with control mesh faces

Patch Construction



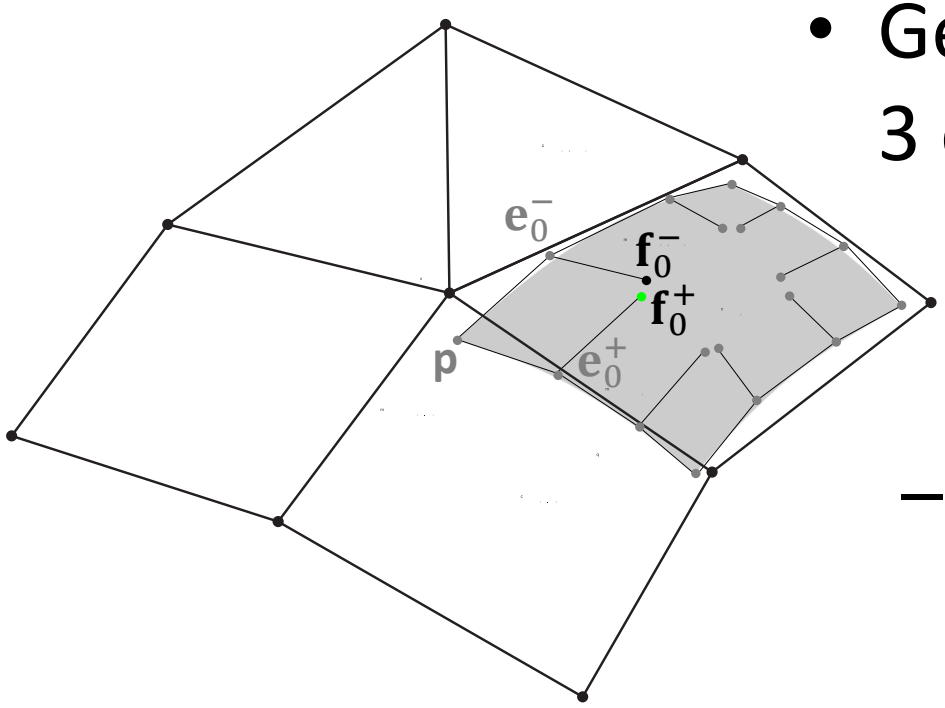
- General construction for 3 or 4 sided faces
  - Corner Points $\mathbf{p}$

Patch Construction



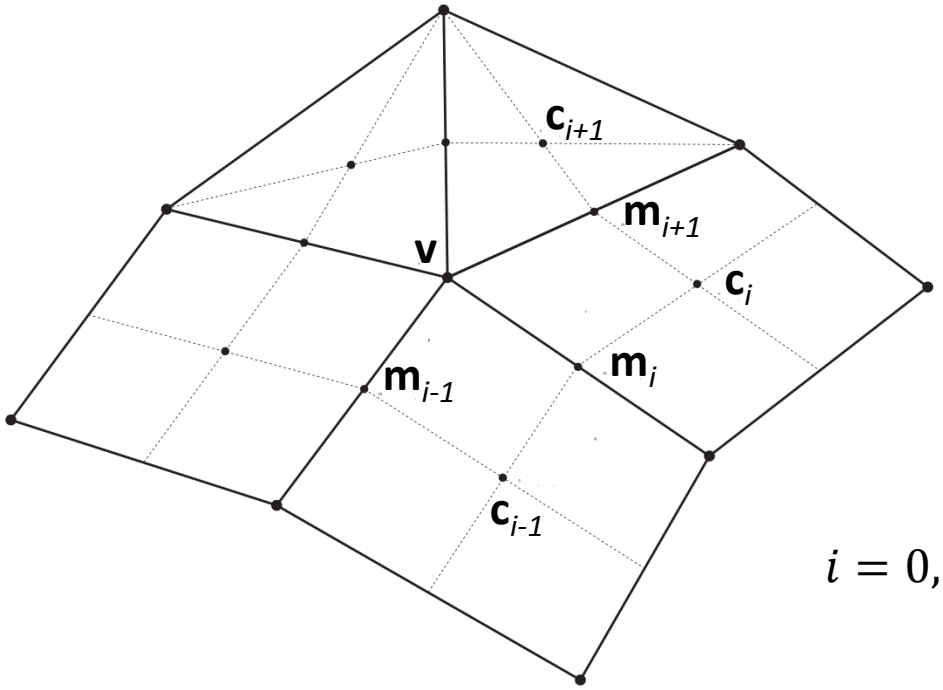
- General construction for 3 or 4 sided faces
  - Corner Points $\mathbf{p}$
  - Edge Points $\mathbf{e}_0^+, \mathbf{e}_0^-$

Patch Construction



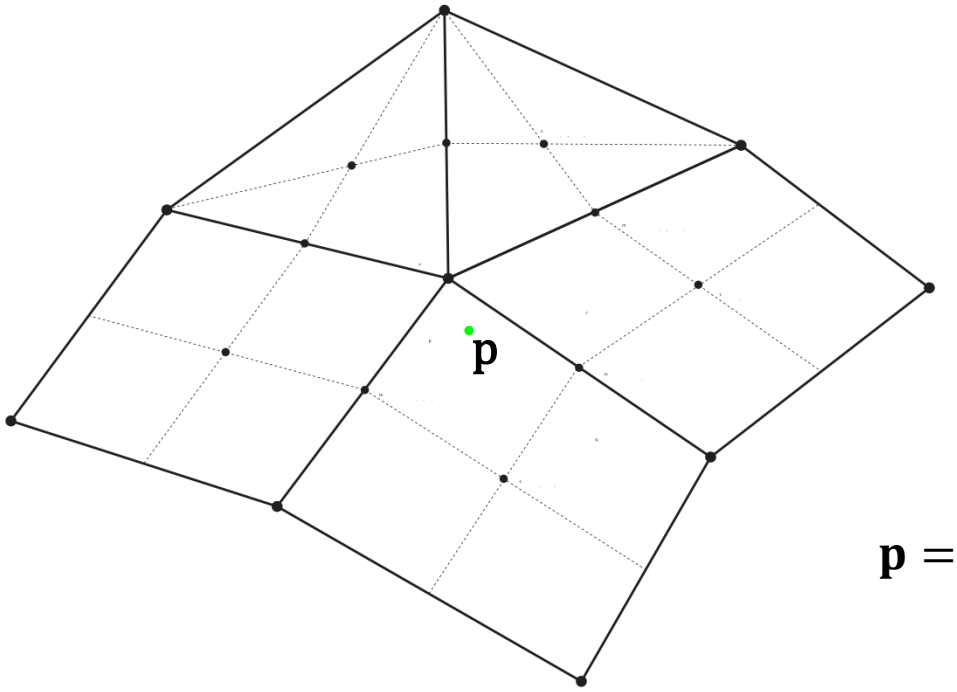
- General construction for 3 or 4 sided faces
  - Corner Points $\mathbf{p}$
  - Edge Points $\mathbf{e}_0^+, \mathbf{e}_0^-$
  - Face Points $\mathbf{f}_0^+, \mathbf{f}_0^-$

Edge Midpoints/Face Centroids



$i = 0, \ldots n-1$ where n is the valence of $\mathbf{v}$

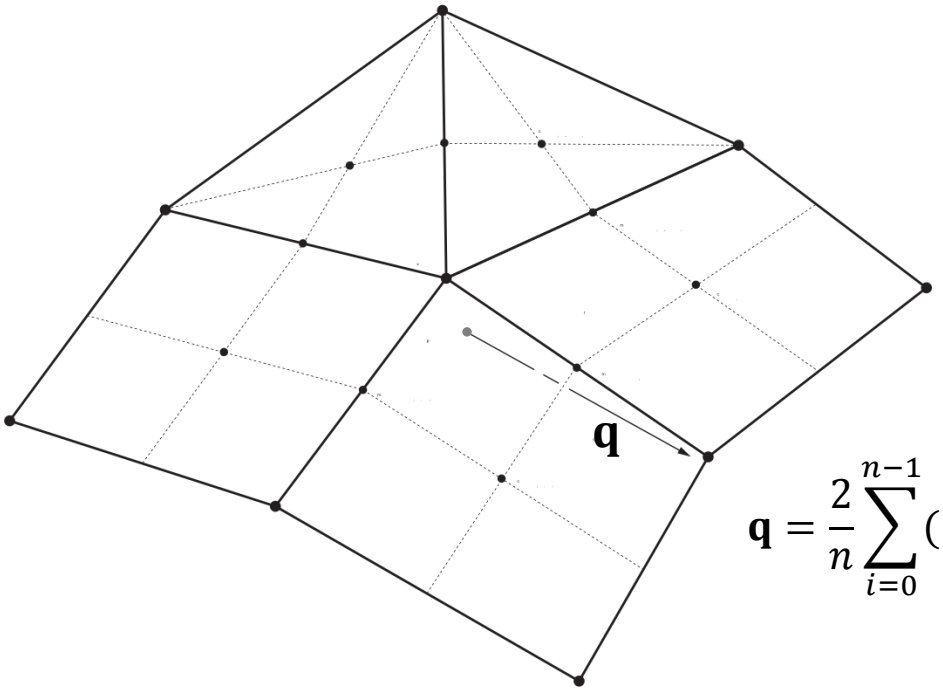
Corner Point



$$\mathbf{p} = \frac{n-3}{n+5}\mathbf{v} + \frac{4}{n(n+5)}\sum_{i=0}^{n-1}(\mathbf{m}_i + \mathbf{c}_i)$$

Interpolate limit position of Catmull-Clark Surface

Edge Points

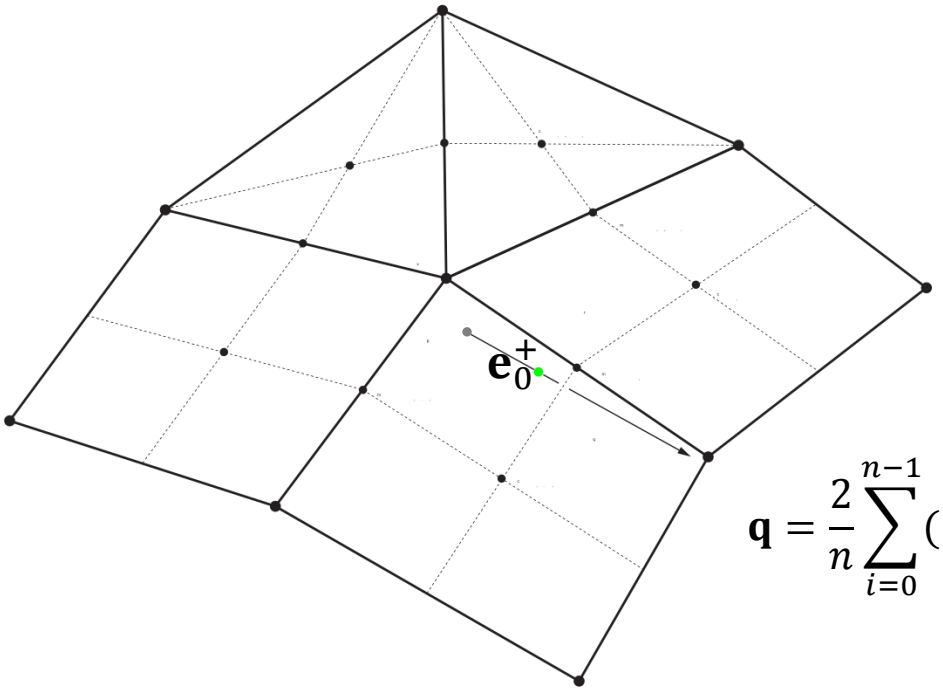


$$\mathbf{q} = \frac{2}{n}\sum_{i=0}^{n-1}\left(\left(1-\sigma\cos\left(\frac{\pi}{n}\right)\right)\cos\left(\frac{2\pi i}{n}\right)\mathbf{m}_i + 2\sigma\cos\left(\frac{2\pi i+\pi}{n}\right)\mathbf{c}_i\right)$$

$$\sigma = \left(4+\cos^2\left(\frac{\pi}{n}\right)\right)^{-1/2}$$

Interpolate limit tangent of Catmull-Clark Surface

Edge Points



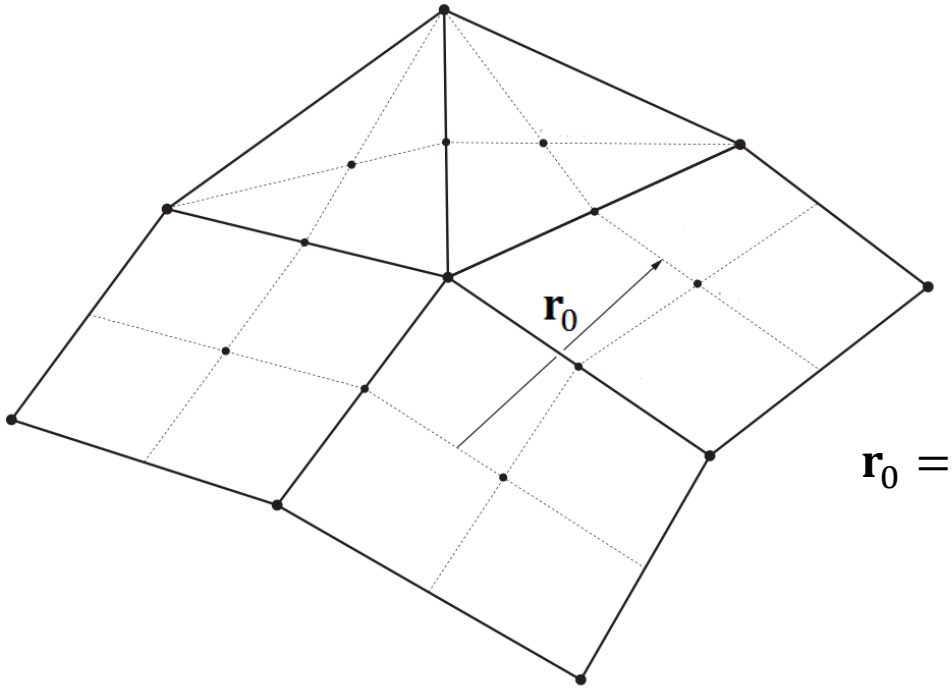
$$\mathbf{q} = \frac{2}{n}\sum_{i=0}^{n-1}\left(\left(1-\sigma\cos\left(\frac{\pi}{n}\right)\right)\cos\left(\frac{2\pi i}{n}\right)\mathbf{m}_i + 2\sigma\cos\left(\frac{2\pi i+\pi}{n}\right)\mathbf{c}_i\right)$$

$$\sigma = \left(4+\cos^2\left(\frac{\pi}{n}\right)\right)^{-1/2}$$

$$\mathbf{e}_0^+ = \mathbf{p} + \frac{2}{3}\lambda\,\mathbf{q}$$

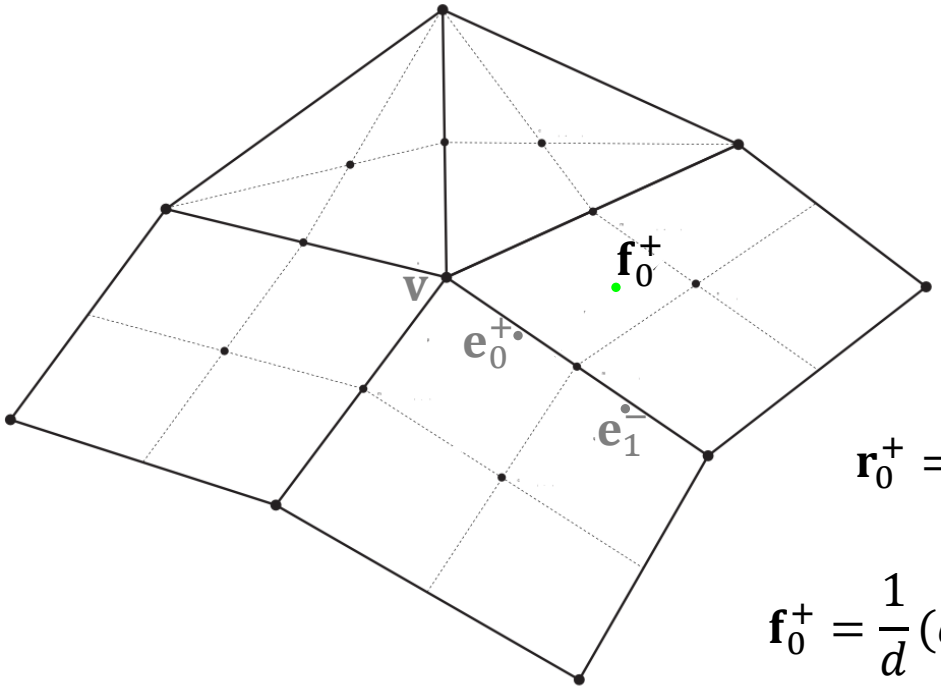
Interpolate limit tangent of Catmull-Clark Surface

Face Points



$$\mathbf{r}_0 = \frac{1}{3}(\mathbf{m}_{i+1} - \mathbf{m}_{i-1}) + \frac{2}{3}(\mathbf{c}_i - \mathbf{c}_{i-1})$$

Face Points



$$\mathbf{r}_0^+ = \frac{1}{3}(\mathbf{m}_{i+1} - \mathbf{m}_{i-1}) + \frac{2}{3}(\mathbf{c}_i - \mathbf{c}_{i-1})$$

$$\mathbf{f}_0^+ = \frac{1}{d}(c_1\mathbf{v} + (d - 2c_0 - c_1)\mathbf{e}_0^+ + 2c_0\mathbf{e}_1^- + \mathbf{r}_0)$$

$$d = \begin{cases} 3\ quad \\ 4\ triangle \end{cases}$$

Implementation

Implementation

Two Approaches

Vertex/Hull Shader Approach

- Vertex Shader:
  - Compute $\mathbf{p}, \mathbf{e}_0^+, \mathbf{e}_0^-, \mathbf{r}_i,\ i = 0, \ldots, n-1$ per vertex
- Hull Shader:
  - Compute $\mathbf{e}_j^+, \mathbf{e}_j^-, \mathbf{f}_j^+, \mathbf{f}_j^-, j = 0 \ldots 2\ (or\ 3)$
  - Pass through $\mathbf{p}_j$

Hull Shader Stencil Approach

- Cluster mesh faces by *neighborhood type*
  - permutation of a face 1-ring neighborhood
  - draw call per neighborhood type

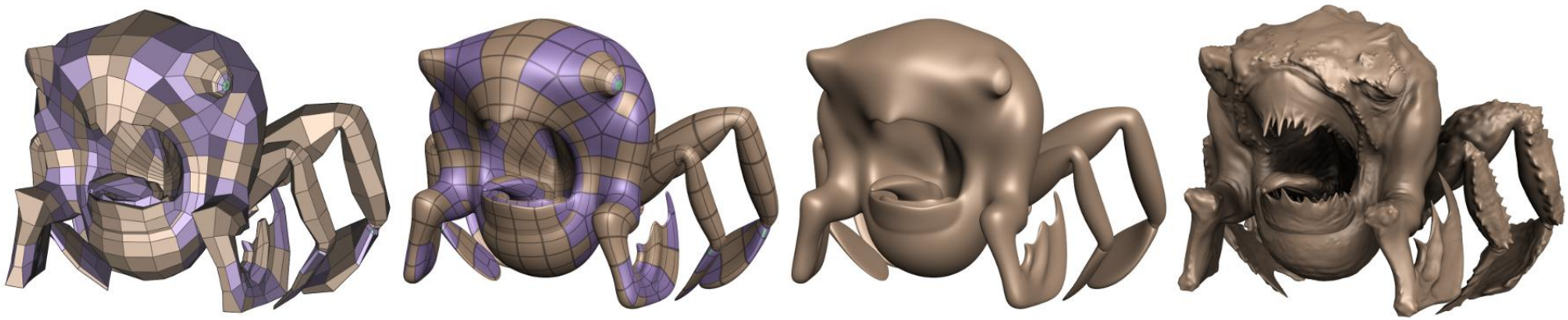
- Neighborhood determines a *stencil matrix*
  - hull shader computes matrix/vector product

Two GPU Approaches

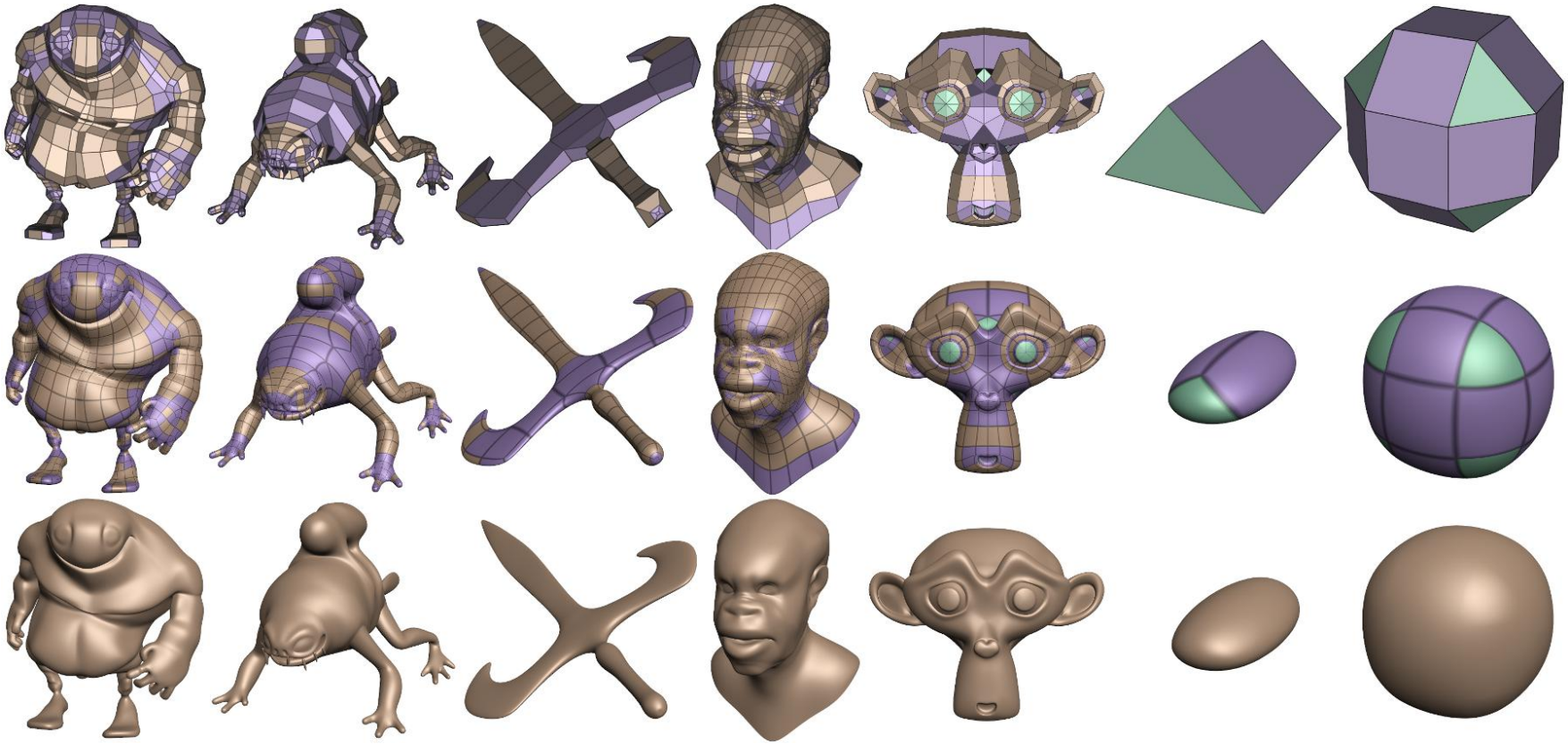
- Vertex/Hull Shaders
  - Exploit vertex-centric nature of computations
  - 3 or 4 point patch primitives
  - *Fat* vertices

- Hull Shader Stencil Approach
  - Map patch construction to hull shader exclusively
  - 6 to 32 point patch primitives
  - *Thin* vertices

Results

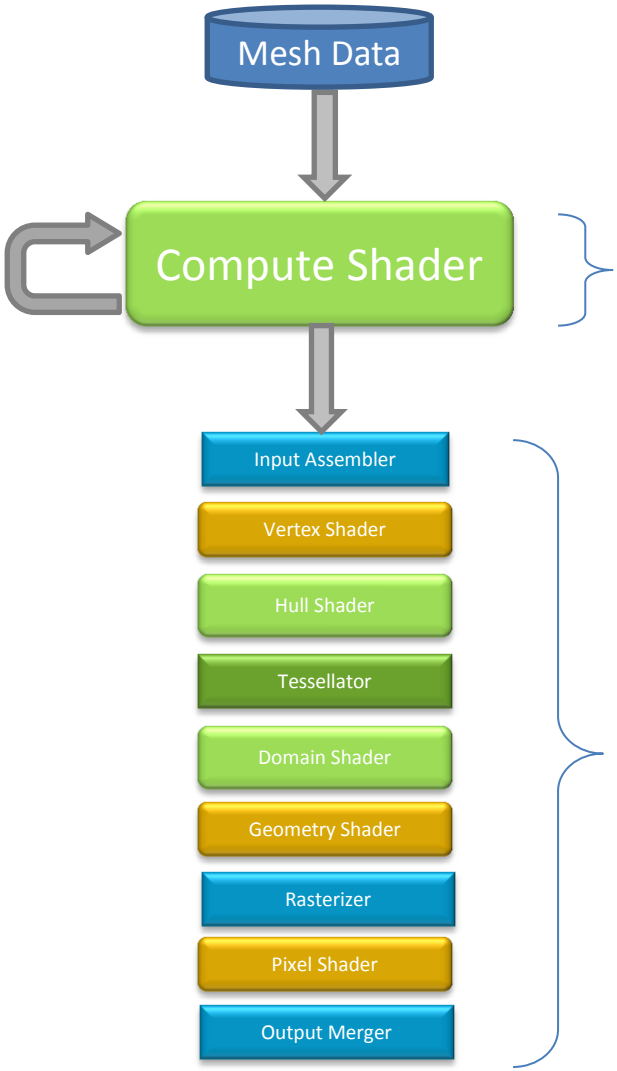


More Results



GPU Subdivision

Hybrid SubD/Tess Data Flow



Data-Parallel
Catmull-Clark
Subdivision

DirectX 11
Tessellation
Pipeline

Subdivision with Tessellation

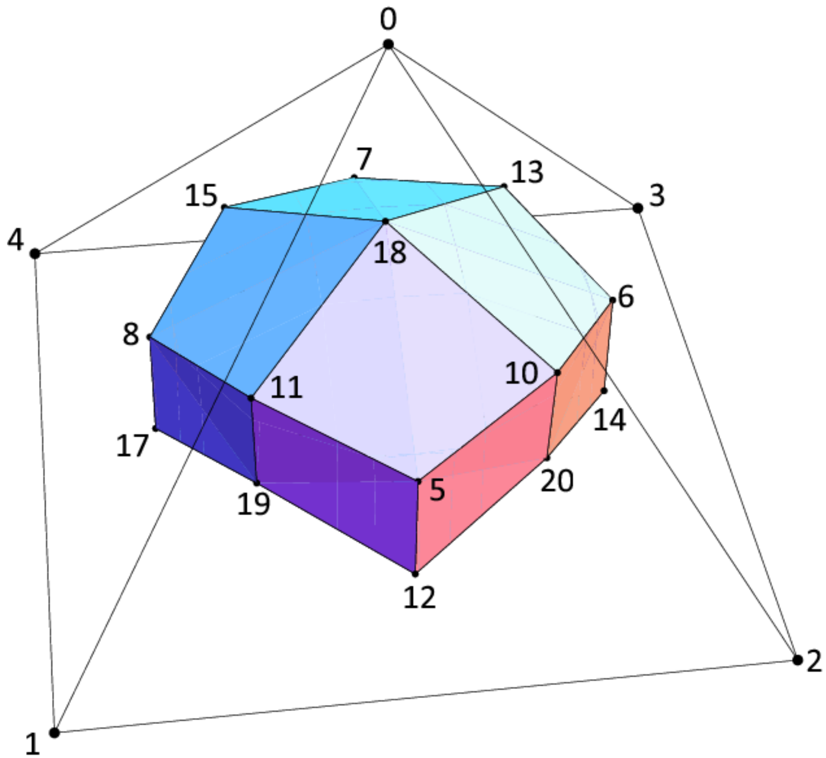
- Simplifies Mesh Connectivity
  - Quad mesh, isolated extraordinary vertices
  - Reduces hull shader matrix combinations
  - Enables more clustering, fewer draw calls
- Performance/Quality Tradeoff
  - Stam's direct evaluation procedure
  - Curvature continuous patching
  - More accurate Gregory approximation

Data-Parallel Subdivision

Catmull-Clark Refinement Rules:

1. Face points - the average of all old points defining a face.
2. Edge points - the average of the two old vertex points and two new face points incident on the edge.
3. Vertex points - the average $\frac{n-2}{n}\mathbf{V} + \frac{1}{n}\mathbf{P} + \frac{1}{n}\mathbf{Q}$ where $\mathbf{V}$ is the old vertex point, $\mathbf{P}$ is the average of the all old vertex points adjacent to the old vertex, and $\mathbf{Q}$ is the average of the new face points of all faces incident to the old vertex.

Subdivision Tables



0
1
2
3
4

old mesh

new mesh

F

a	b		c			
5	0	3	0	1	2	
6	3	3	0	2	3	
7	6	3	0	3	4	
8	9	3	0	4	1	
9	12	4	4	3	2	1

E

a	c			
10	0	5	2	6
11	1	5	0	8
12	2	5	1	9
13	0	6	3	7
14	3	6	2	9
15	0	7	4	8
16	4	7	3	9
17	1	8	4	9

V

a	b		c							
18	0	4	4	8	1	5	2	6	3	7
19	8	3	4	9	2	5	0	8		
20	14	3	3	6	0	5	1	9		
21	20	3	4	7	0	6	2	9		
22	26	3	1	8	0	7	3	9		

a *b* *c*

Face Kernel

```
[numthreads(8, 4, 1)]
void FacePoint(uint3 blockIdx : SV_GroupID,
               uint3 DTid : SV_DispatchThreadID,
               uint3 threadIdx : SV_GroupThreadID,
               uint GI : SV_GroupIndex )
{
    int FaceIdx = 8 * blockIdx.x + threadIdx.x;

    if (FaceIdx < F0){
        int h = F0_ITa[2*FaceIdx];
        int n = F0_ITa[2*FaceIdx+1];

        float q = 0.0f;
        for (int j=0; j<n; j++){
            q += VB[SRC_OFFSET + 4*F0_IT[h++] + threadIdx.y];
        }
        q /= ((float)n);

        VB[DEST_OFFSET_1 + 4*(FaceIdx) + threadIdx.y] = q;
    }
}
```

Edge Kernel

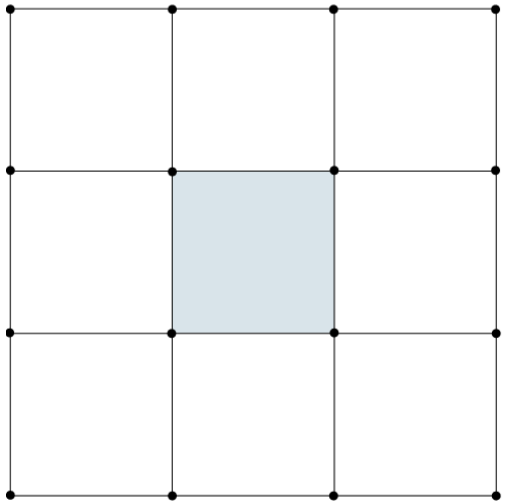
```
[numthreads(8, 4, 1)]
void EdgePoint(uint3 blockIdx : SV_GroupID,
               uint3 DTid : SV_DispatchThreadID,
               uint3 threadIdx : SV_GroupThreadID,
               uint GI : SV_GroupIndex )
{
    int EdgeIdx = 8 * blockIdx.x + threadIdx.x;

    if (EdgeIdx < E0){
        float q = 0.25f*(
            VB[SRC_OFFSET + 4*E0_IT[4*EdgeIdx+0] + threadIdx.y] +
            VB[SRC_OFFSET + 4*E0_IT[4*EdgeIdx+1] + threadIdx.y] +
            VB[SRC_OFFSET + 4*E0_IT[4*EdgeIdx+2] + threadIdx.y] +
            VB[SRC_OFFSET + 4*E0_IT[4*EdgeIdx+3] + threadIdx.y] );

            VB[DEST_OFFSET_2 + 4*EdgeIdx + threadIdx.y] = q;
    }
}
```

Vertex Kernel

```
[numthreads(8, 4, 1)]
void VertexPoint(uint3 blockIdx : SV_GroupID,
                 uint3 DTid : SV_DispatchThreadID,
                 uint3 threadIdx : SV_GroupThreadID,
                 uint GI : SV_GroupIndex)
{
    int VertexIdx = 8 * blockIdx.x + threadIdx.x;

    if (VertexIdx < V0){
        int h = V0_ITa[2*VertexIdx];
        int n = V0_ITa[2*VertexIdx+1];

        float q = 0.0f;
        for (int j=0; j<n; j++){
            q += VB[SRC_OFFSET + 4*V0_IT[h+2*j  ] + threadIdx.y]
               + VB[SRC_OFFSET + 4*V0_IT[h+2*j+1] + threadIdx.y];
        }

        float fn = (float)n;

        float wv = (fn-2)/fn;
        float wp = 1.0f/(fn*fn);

        VB[DEST_OFFSET_3 + 4*VertexIdx + threadIdx.y] =
             wv*VB[SRC_OFFSET + 4*VertexIdx + threadIdx.y] + wp*q;
    }
}
```

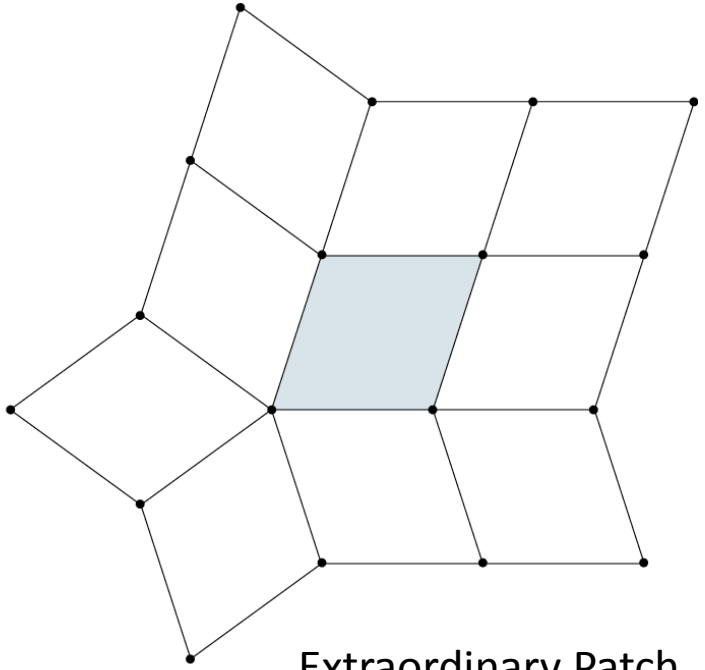
Tessellator Pipeline Stage

Simplified Patch Input



Regular Patch

16 control points



Extraordinary Patch

$2n + 8$ control points

Regular Patches

- Hull Shader
  - B-spline to Bézier $\mathbf{B} = \mathrm{Q} \cdot \mathbf{D} \cdot \mathrm{Q}^T$ 60 FLOPS
- Domain Shader
  - $pos = \mathrm{u} \cdot \mathbf{B} \cdot \mathrm{v}$, $normal = (\mathrm{du} \cdot \mathbf{B} \cdot \mathrm{v}) \times (\mathrm{u} \cdot \mathbf{B} \cdot \mathrm{dv})$
  - Reverse DeCastlejau pyramid 162 FLOPS

Domain Shader

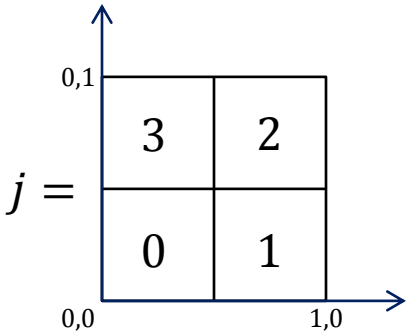
```
float u[4], du[4];
float3 uB[4], duB[4];

CubicBezier(uv.x, u, du);

for (uint i = 0; i < 4; i++) {
     uB[i] =  float3(0, 0, 0);
    duB[i] =  float3(0, 0, 0);

    for (uint j = 0; j < 4; j++) {
        float3 A = B[4*i + j];

         uB[i] +=  u[j] * A;
        duB[i] += du[j] * A;
    }
}

float3 WorldPos  = float3(0, 0, 0);
float3 Tangent   = float3(0, 0, 0);
float3 BiTangent = float3(0, 0, 0);

CubicBezier(uv.y, u, du);

for (i = 0; i < 4; i++) {
    WorldPos  +=  uB[i] *  u[i];
    Tangent   += duB[i] *  u[i];
    BiTangent +=  uB[i] * du[i];
}
```

Basis Functions

```
void CubicBezier(in float u, out float b[4], out float d[4])
{
    float t = u;
    float s = 1.0 - u;

    float a0 = s * s;
    float a1 = 2 * s * t;
    float a2 = t * t;

    b[0] =          s * a0;
    b[1] = t * a0 + s * a1;
    b[2] = t * a1 + s * a2;
    b[3] = t * a2;

    d[0] =    - a0;
    d[1] = a0 - a1;
    d[2] = a1 - a2;
    d[3] = a2;
}
```

Exact Evaluation of the Limit Surface

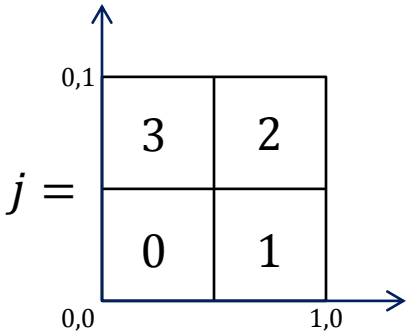
- Stam's algorithm
  - $s_{j,k}(u,v) = b(u,v) \cdot \mathrm{P}_j \cdot \mathrm{S}^k \cdot \mathbf{v}_0$



k = *subdivision level*

Exact Evaluation of the Limit Surface

- Stam's algorithm
  - $b(u,v) \cdot \mathrm{P}_j \cdot (\mathrm{V} \cdot \Lambda^k \cdot \mathrm{V}^{-1}) \cdot \mathbf{v}_0$



k = *subdivision level*

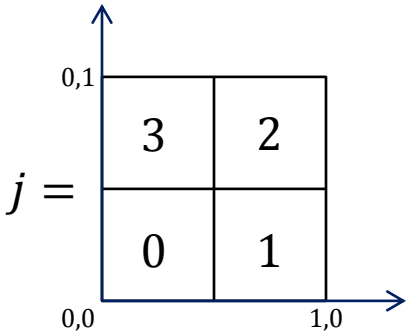
Exact Evaluation of the Limit Surface

- Stam's algorithm

$$-\underbrace{b(u,v)\cdot \mathrm{P}_j \cdot \mathrm{V}}\cdot \Lambda^k \cdot \underbrace{\mathrm{V}^{-1}\cdot \mathbf{v}_0}$$

eigen basis functions
$2n+8$ bicubic polynomials

eigen space projection
$(2n+8)\times(2n+8)$ matrix-vector multiply



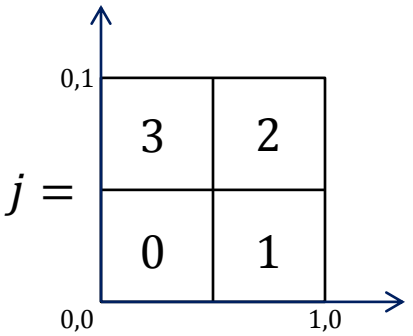
$k =$ *subdivision level*

Exact Evaluation of the Limit Surface

- Stam's algorithm
  - $\underbrace{b(u,v) \cdot \mathrm{P}_j \cdot \mathrm{V}} \cdot \Lambda^k \cdot \underbrace{\mathrm{V}^T \cdot \mathbf{v}_0}$

    eigen basis functions: $2n+8$ bicubic polynomials

    eigen space projection: $(2n+8) \times (2n+8)$ matrix-vector multiply



$k =$ *subdivision level*

- Hull Shader
  - Eigen space projection
  - $2n + 8$ threads, $6n + 24$ FLOPS
- Domain Shader
  - Determine j, k, raise Λ to k^{th}power, remap (u, v), evaluate eigen basis functions and derivatives, form sum-of-products
  - $108n + 534$ FLOPS

Curvature Continuous Patching

- Loop, C. AND Schaefer, S. 2008,
  G^2 tensor product splines over extraordinary vertices
  - Bicubic patch per regular quad
  - Biseptic patch per irregular quad (64 coefficients)
    - $2n + 8$ precomputed basis functions

- Hull Shader
  - max output: 32 vertices, pack 2 into 1
  - 32 threads, $12n + 48$ FLOPS per thread

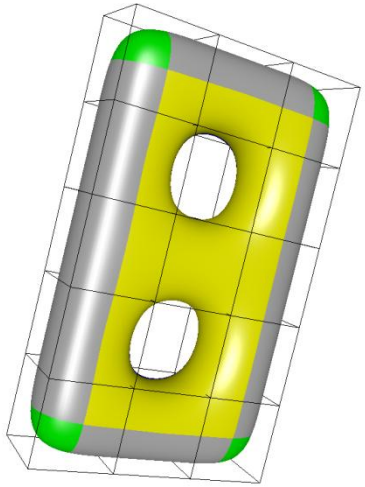
- Domain Shader
  - Independent of valence n 568 FLOPS

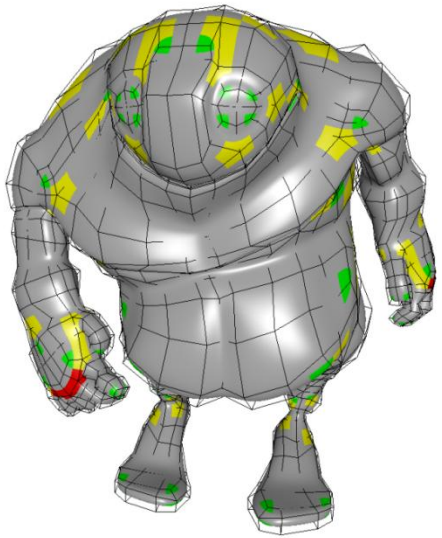
Gregory Patch Approximation

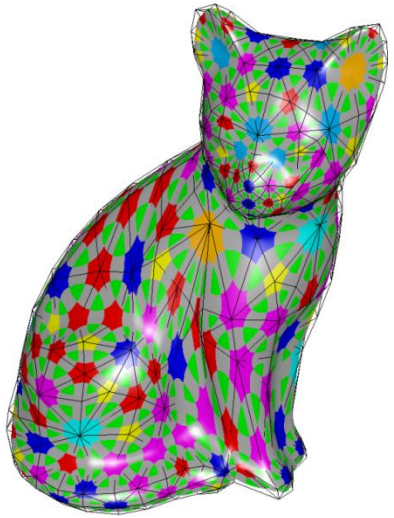
- Complexity Reduction
  - from: quads & tris, 1000's HS matrices
  - to: 9 cases $(n = 3, \dots, 12)$
- Hull Shader
  - 20 threads, $6n + 24$FLOPS per thread
- Domain Shader
  - 174 FLOPS

Results

Subdivision Stage



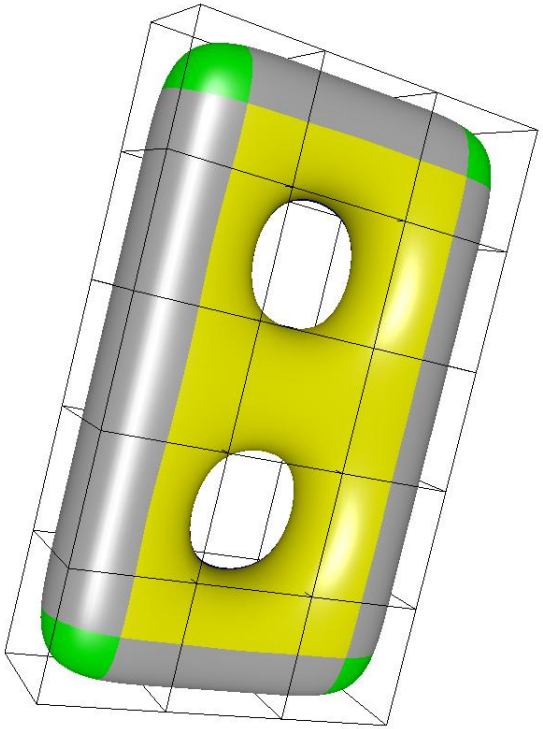


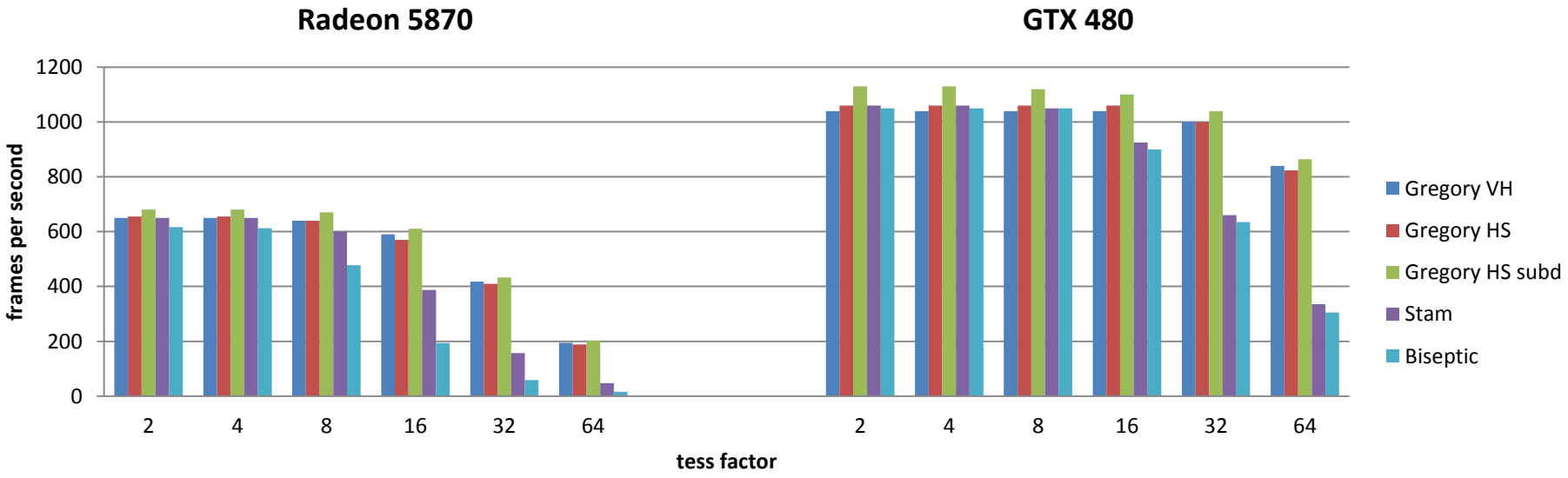


GPU	Twohole	Bigguy	Cat
Radeon 5870	0.30	0.31	0.35
GTX 480	0.33	0.36	0.48

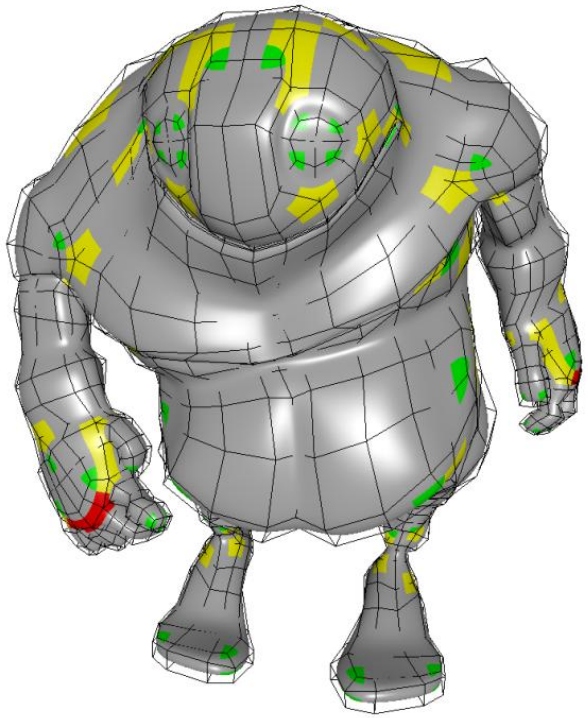
time in milliseconds

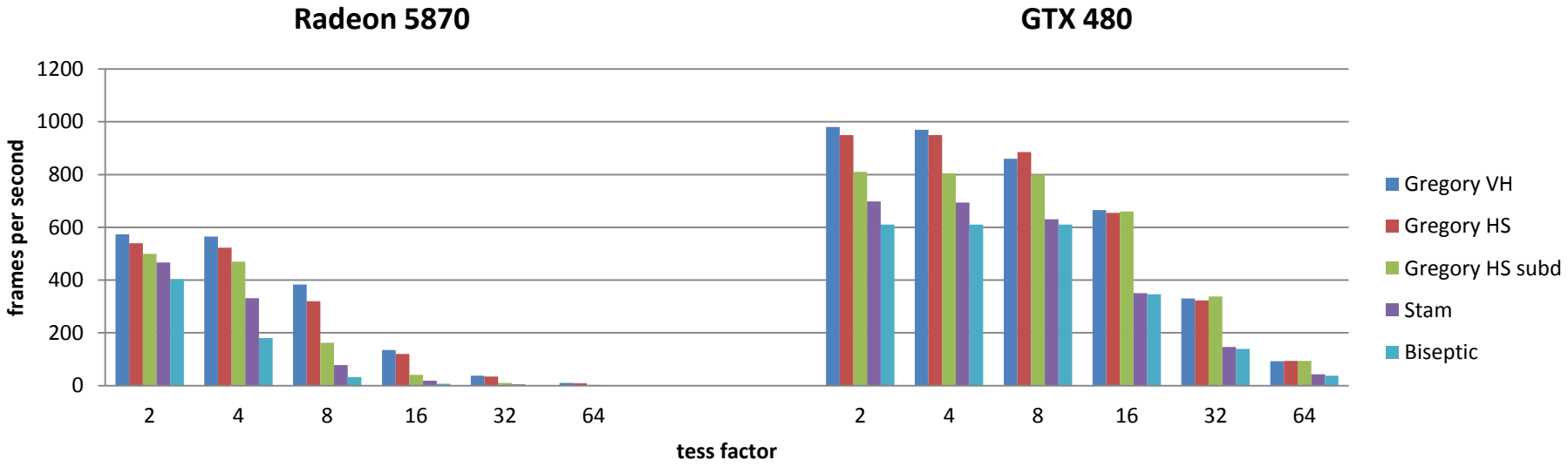
Twohole



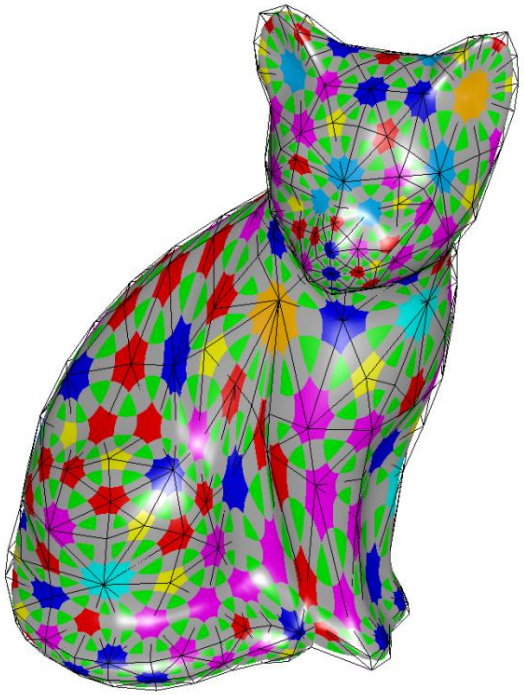

Radeon 5870
GTX 480
frames per second
1200
1000
800
600
400
200
0
2
4
8
16
32
64
tess factor
Gregory VH
Gregory HS
Gregory HS subd
Stam
Biseptic

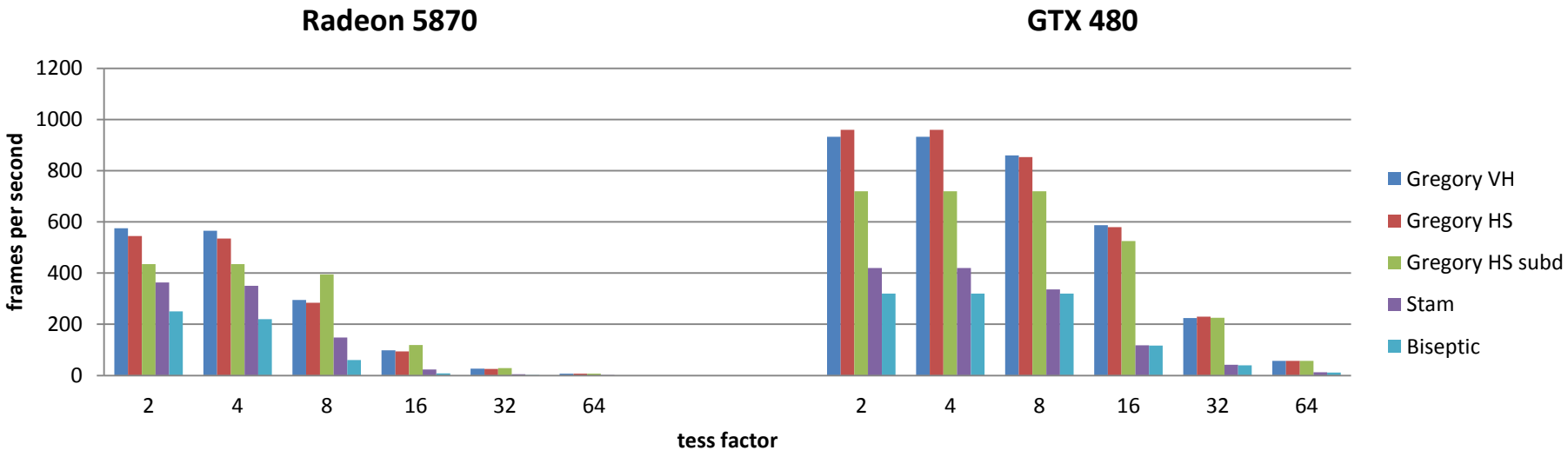
Bigguy




Radeon 5870
GTX 480
frames per second
1200
1000
800
600
400
200
0
2
4
8
16
32
64
tess factor
Gregory VH
Gregory HS
Gregory HS subd
Stam
Biseptic

Cat




Radeon 5870
GTX 480
frames per second
1200
1000
800
600
400
200
0
2
4
8
16
32
64
tess factor
Gregory VH
Gregory HS
Gregory HS subd
Stam
Biseptic

Conclusions

- Approximation is fastest
  - ideal for displacement mapping
- GPU mesh subdivision is fast
  - memory needs impractical for deep subdivision
  - tessellation big win for dense sampling
- Hybrid subd/tess is interesting
  - complexity reduction
  - speed/accuracy/quality knob

Thanks!