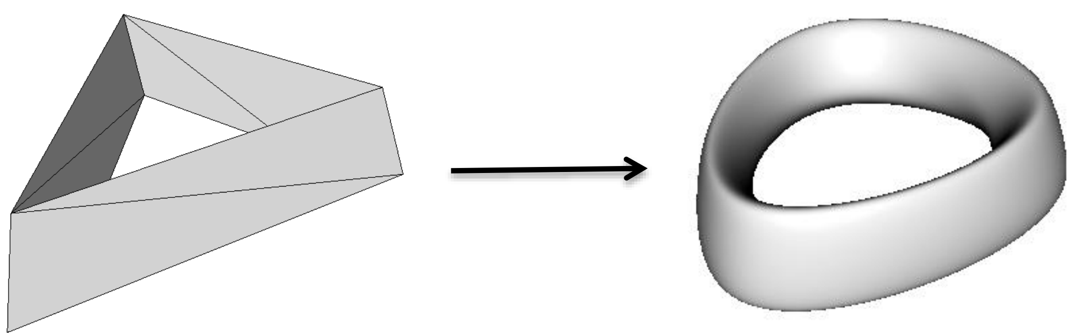
Computer Graphics & Virtual Reality

Shape Modeling : Subdivision Surfaces

Tamy Boubekeur



Applications


Copyright Pixar


Copyright Ubisoft

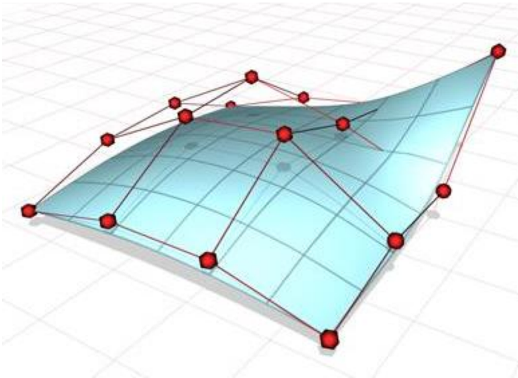
- Animation
- FX
- Video games
- CAD

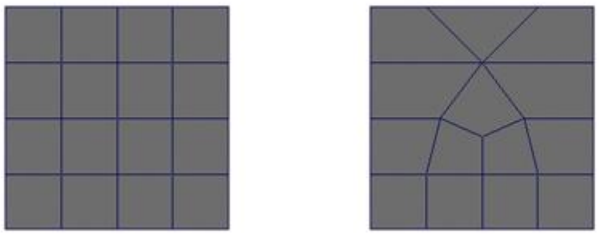

Copyright 20th Century Fox

Where do subdivision surfaces come from ?

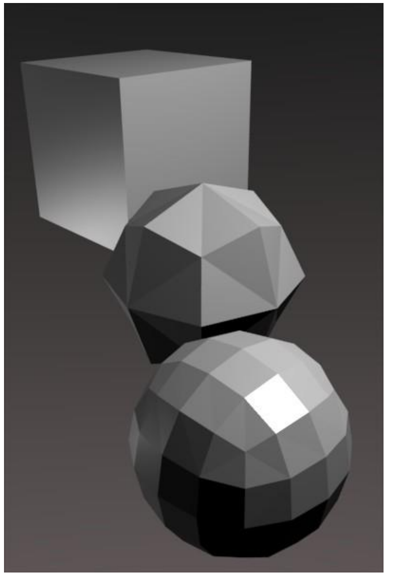
- Original issue:
  "How to define a smooth surface of arbitrary topology ?"

  NB: splines were the major representation in the 70's
  - Mesh Refinement: **local**, **recursive**, efficient et simple.
- **Idea Nb1**: avoid control net/spline and work only with
  - A mesh
  - Some refinement rules
- **Idea Nb2**: without spline, consider arbitrary topology, adapt the rules
- **Idea Nb3**: call it **subdivision surface**





Subdivision Surface



Coarse mesh (domain) + ***subdivision rules***

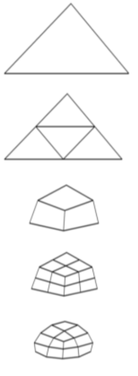
Subdivision Scheme: a set of subdivision rules

Formal Definition:

Limit geometry of a domain mesh under an infinite number of subdivision steps

Recursive Evaluation:

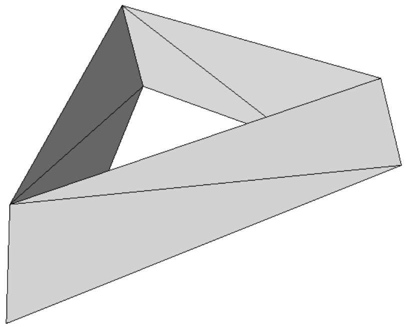
1. Tessellate
2. Smooth
3. Tessellate
4. Smooth
5. …



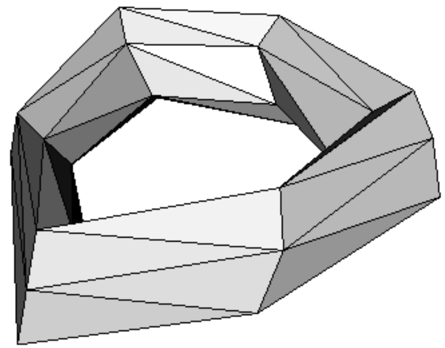
Parametric Evaluation:

– For spline-based schemes
– Use the underlying spline definition of the limit surface
– Arbitrary tessellation and direct projection on the limit surface

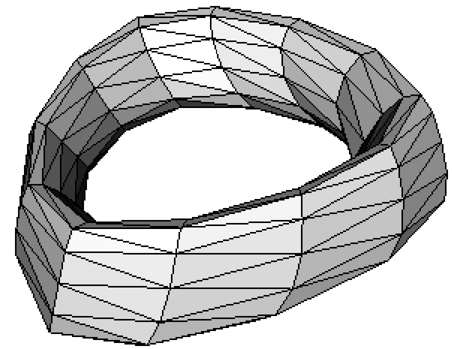
Subdivision Surfaces



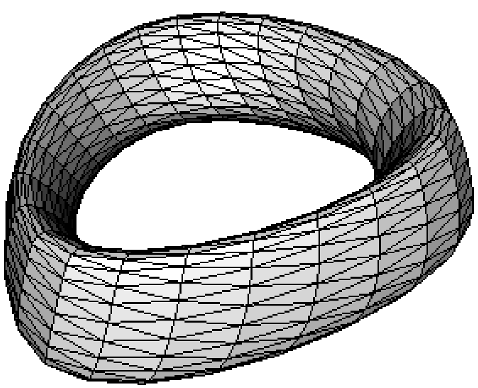
Input Mesh



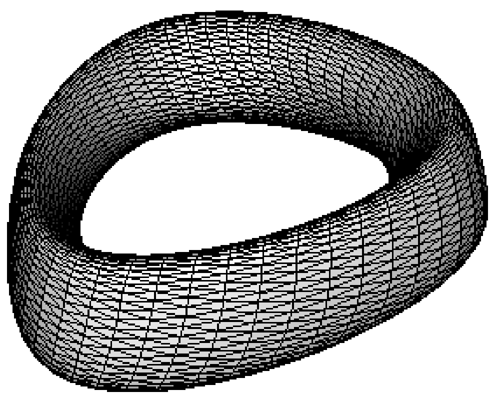
1 subd pass



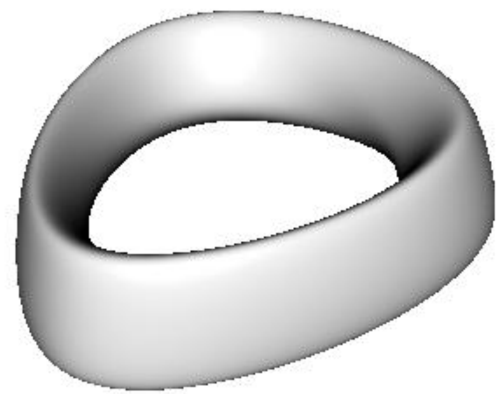
2 subd passes



3 subd passes



4 subd passes



Limit surface

Terminology

Even vertex: exists before tessellation

Odd vertex : inserted by tessellation at the current step

Regular vertex: valence 6 for triangles meshes, 4 for quad meshes

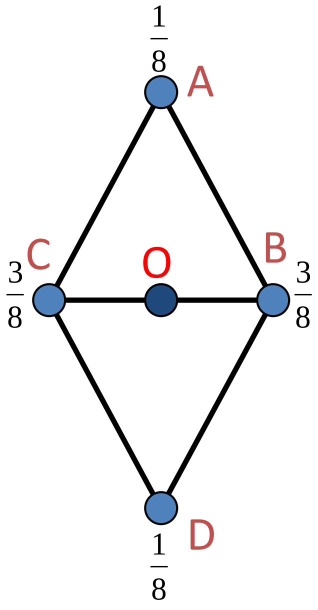
Extraordinary vertex : the other ones

Primal scheme : subdivide *polygons*

Dual scheme : subdivide *vertices*

Subdivision Mask

- A *graphical* representation of subdivision schemes
- Example :



Odd vertex mask

A,B,C,D : even vertices

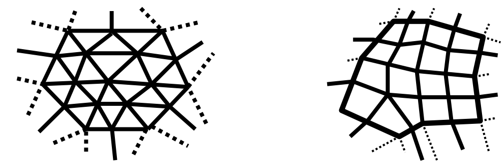
O : odd vertex

O = A/8 + 3B/8 + 3C/8 + D/8

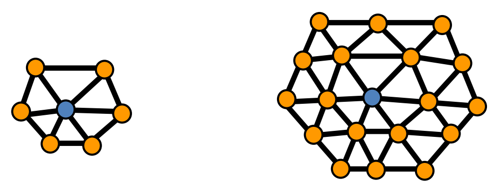
Classification criteria

- **Interpolation** or **Approximation**

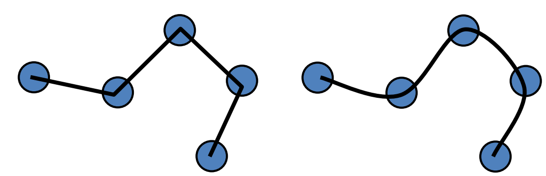
- Polygon **type**: triangle, quads, hybride, …



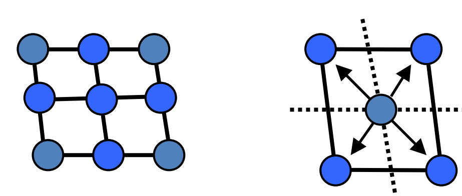
- **Locality** : neighborhood inspection required (1-ring, 2-ring, more)



- **Continuity**: C0, C1, C2



- **Primal** or Dual



Classification

Primal	**Triangle Meshes**	**Quad Meshes**
Approximation	**Loop** (C2)	**Catmull-Clark** (C2)
Interpolation	**Modified Butterfly** (C1)	Kobbelt (C1)

Dual	**Triangle Meshes**	**Quad Meshes**
Approximation	**Sqrt (3)**	Doo-Sabin (C1), Mid-Edge, Bi-Quartic (C2)

Note : continuity given for regular vertices

Loop Scheme

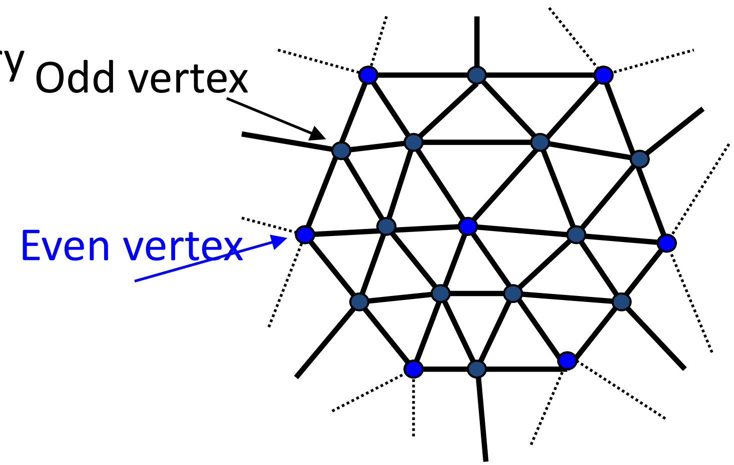
Approximating scheme for triangle mesh

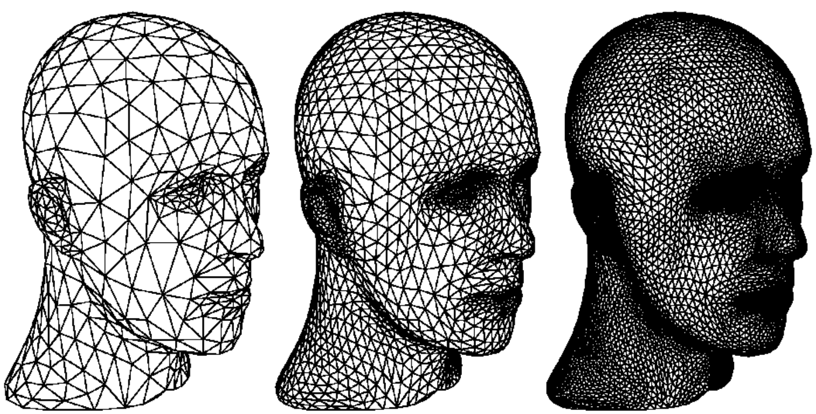
C2 continuity everywhere but on extraordinary vertices (C1 only)

Primal refinement

Tags (vertices, edges, faces) and special rules:

- Borders
- Sharp and semi-sharp crease
- Corners
- Dart points
- Normal control

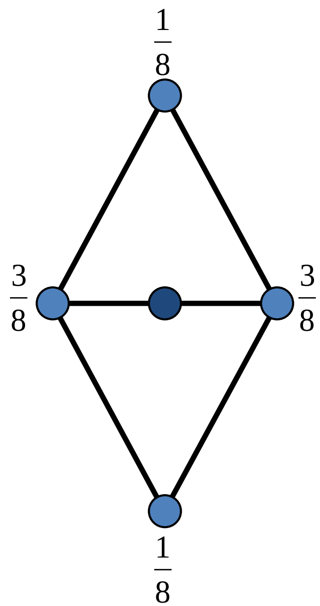




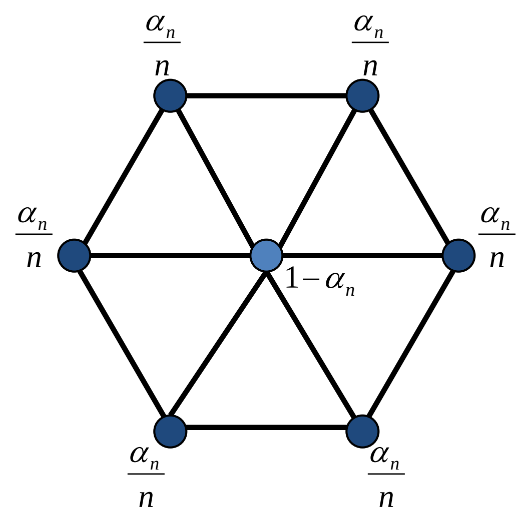
2 Loop subdivision steps

Loop Scheme

Loop masks



Odd vertices



$$\alpha_n = \frac{1}{64}\left(40 - \left(3 + 2\cos\left(\frac{2\pi}{n}\right)\right)^2\right)$$

Even vertices

Catmull-Clark Scheme

Approximating scheme for quad meshes

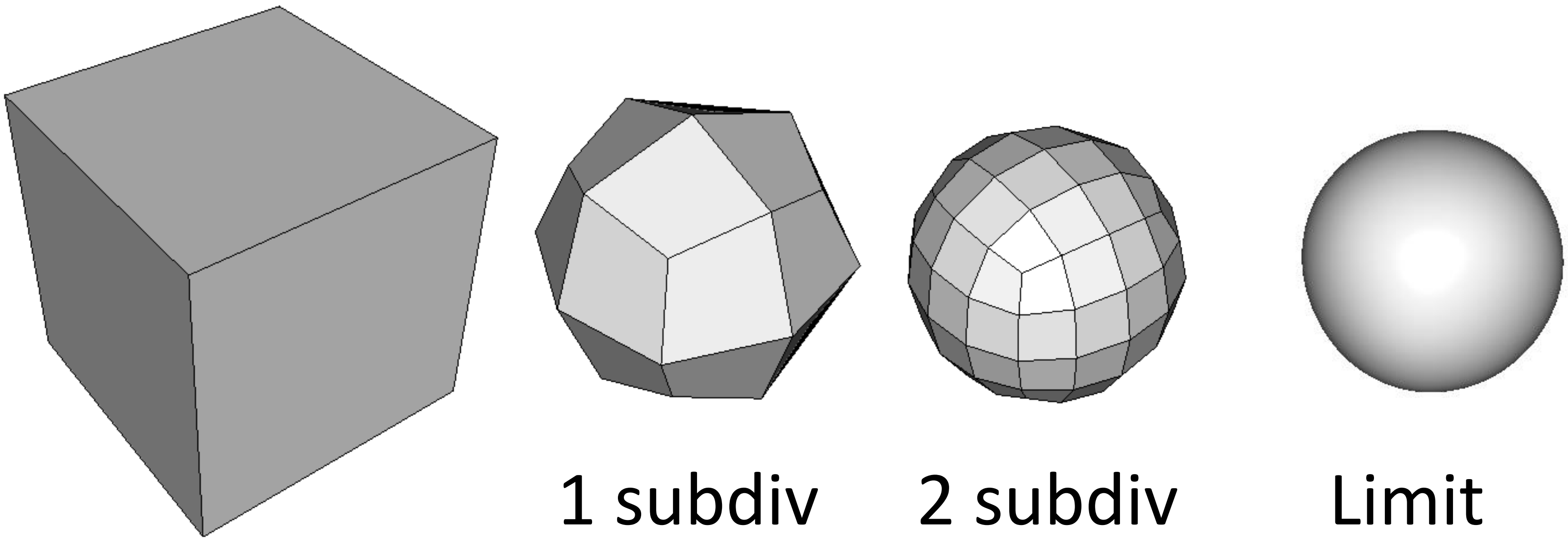
C2 everywhere but on extraordinary vertices (C1 only)

Primal scheme

Special rules: sharp creases, corners, dart points

Geri's Game (1997) : Pixar Animation Studios

Catmull-Clark Scheme



Input Mesh

1 subdiv pass

2 subdiv passes

Limit Surface

Catmull-Clark Scheme

FACE

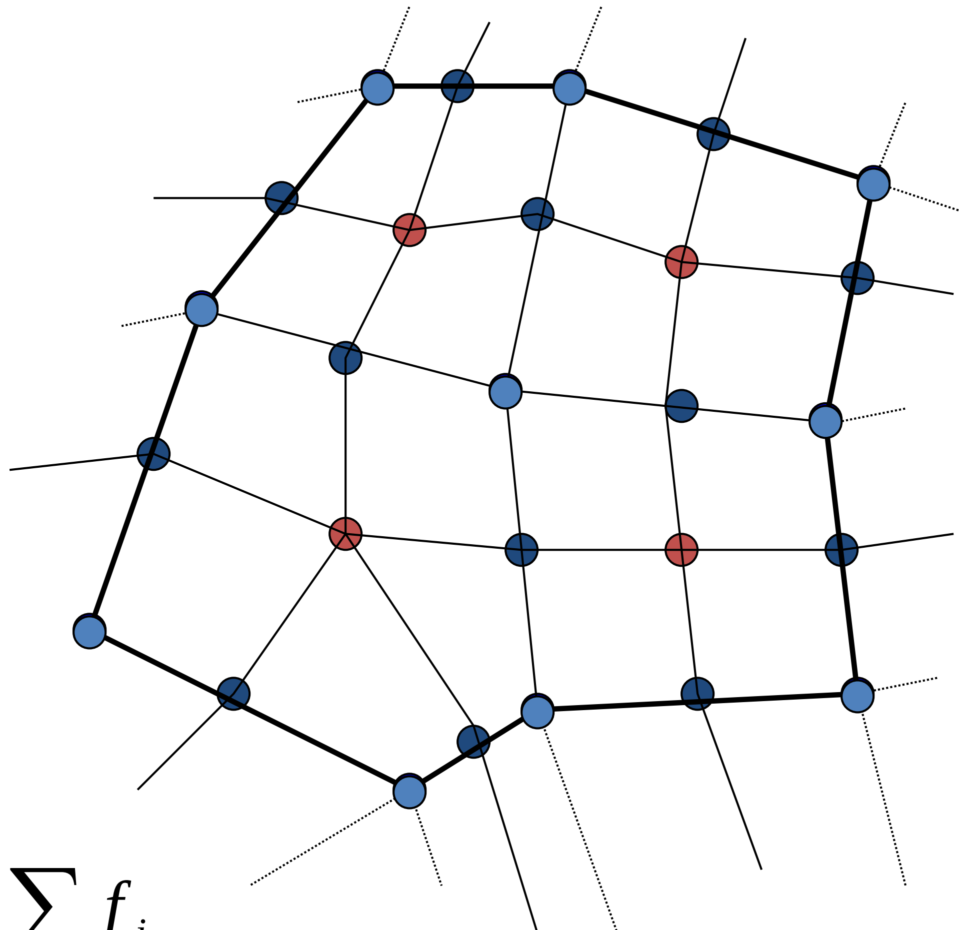
$$f = \frac{1}{n}\sum_{1}^{n} v_i$$

EDGE

$$e = \frac{v_1 + v_2 + f_1 + f_2}{4}$$

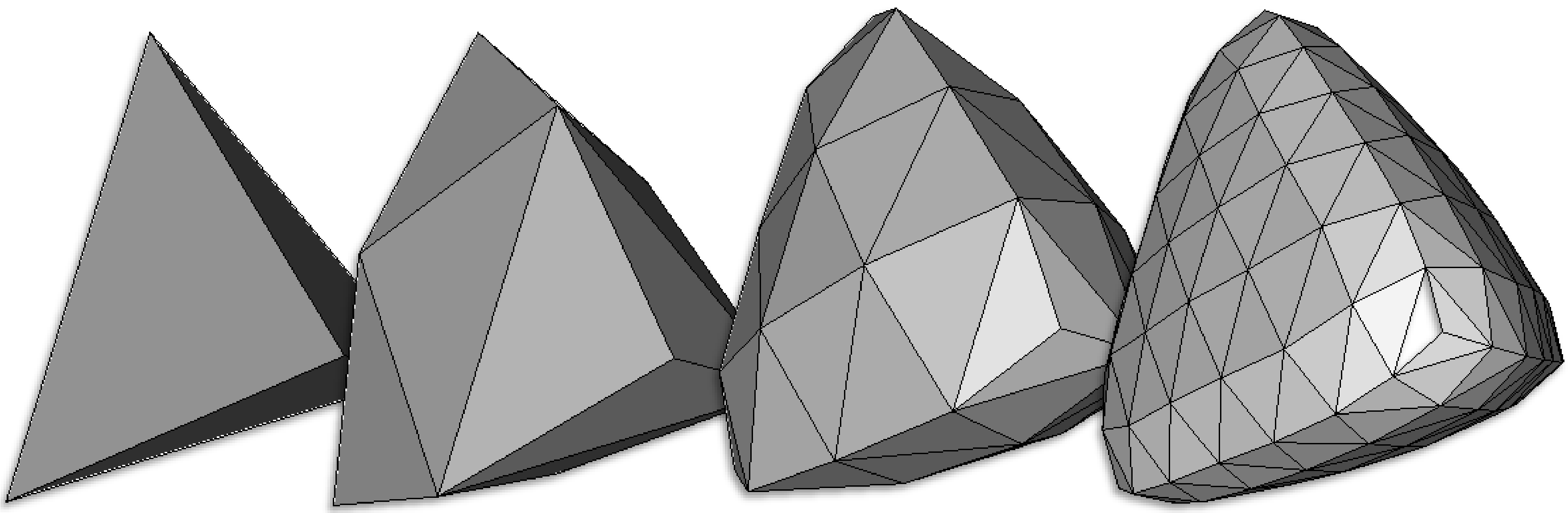
VERTEX

$$v_{i+1} = \frac{n-2}{n} v_i + \frac{1}{n^2}\sum_j e_j + \frac{1}{n^2}\sum_j f_j$$

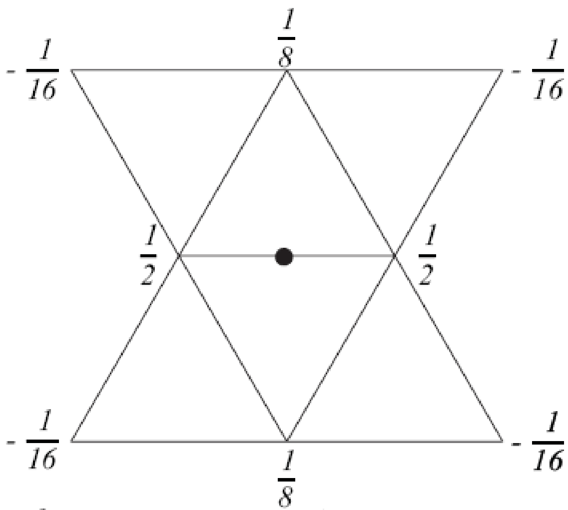


Modified Butterfly Scheme

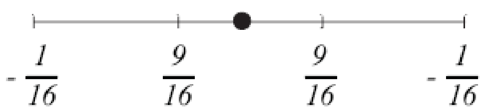
- **Interpolating** scheme for triangle meshes
- C1 everywhere



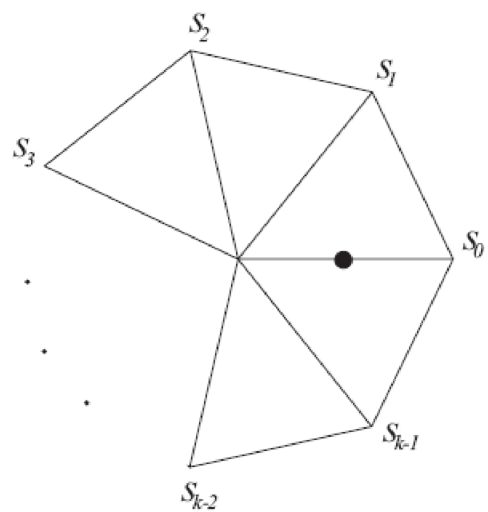
Modified Butterfly Scheme



Regular Odd Vertices



Borders and sharp creases



Extraordinaty Odd Vertices

$$k = 3, \qquad s_0 = \frac{5}{12},\ s_{1,2} = -\frac{1}{12}$$

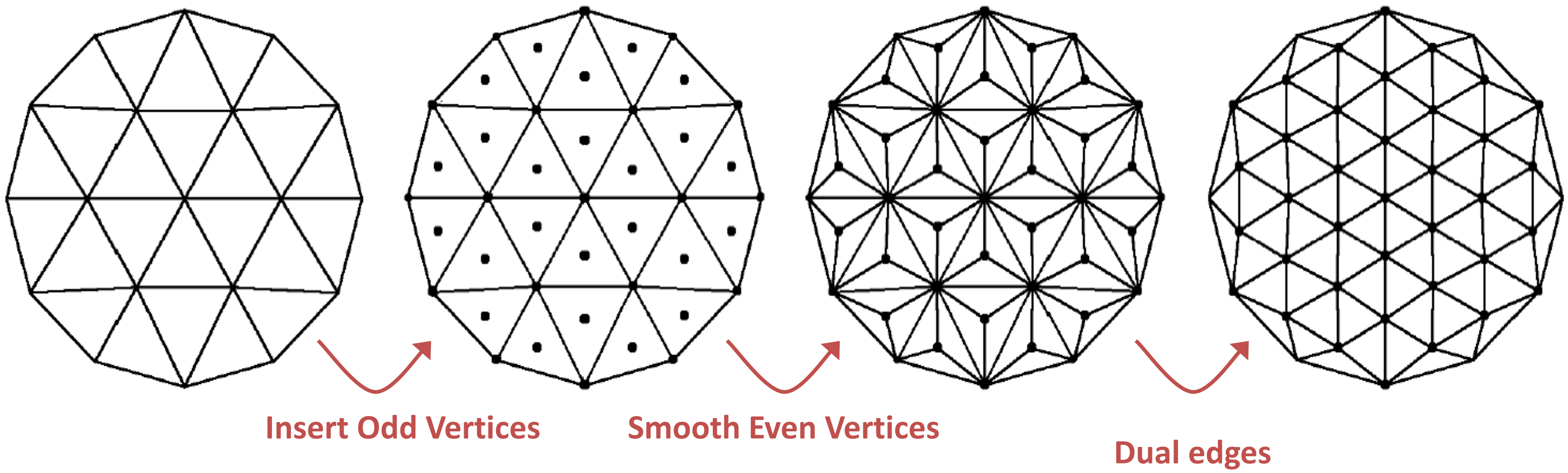
$$k = 4 \qquad s_0 = \frac{3}{8},\ s_2 = -\frac{1}{8},\ s_{1,3} = 0.$$

$$k > 5. \qquad \frac{1}{k}\left(\frac{1}{4} + \cos\frac{2i\pi}{k} + \frac{1}{2}\cos\frac{4i\pi}{k}\right)$$

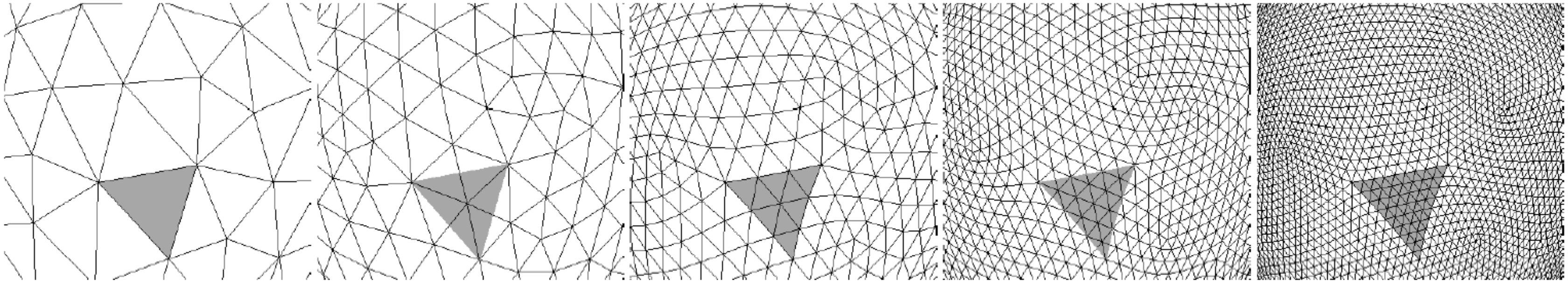
Note : interpolation requires only odd rules

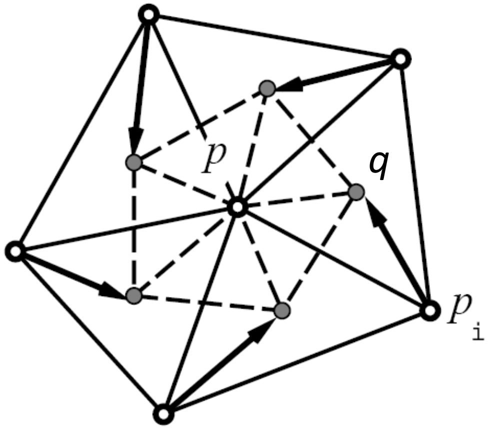
√3 Scheme

- Approximating scheme for triangle meshes
- 1-3 tessellation, local connectivity « rotation »
- More *progressive* refinement than Loop (25% less triangles)
- Sharp creases
- Straightforward adaptive subdivision



√3 Scheme





Odd vertices (insertion)

Triangle barycenter

$$\mathbf{q} := \frac{1}{3}\left(\mathbf{p}_i + \mathbf{p}_j + \mathbf{p}_k\right)$$

Even Vertices (relaxation)

Interpolation between original position and barycenter of the 1-ring neighborhood

$$S(\mathbf{p}) := (1-\alpha_n)\,\mathbf{p} + \alpha_n \frac{1}{n}\sum_{i=0}^{n-1} \mathbf{p}_i.$$

$$\alpha_n = \frac{4 - 2\cos\left(\frac{2\pi}{n}\right)}{9}$$

Subdivision Matrix

- A subdivision scheme can be expressed by a matrix M of weights *wij*
  - *wij: weight of pj when computing pi at next level*
  - M is sparse
  - M **should not be used for implementation**
  - Enable subdivision analysis
    - Eigen analysis
    - Limit surface

$$P^{i+1} = MP^{i}$$

$$\underbrace{\begin{bmatrix} w_{00} & w_{01} & \cdots & 0 \\ w_{10} & w_{11} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & w_{nj} \end{bmatrix}}_{\text{Weights}} \underbrace{\begin{bmatrix} p_0^i \\ p_1^i \\ \vdots \\ p_n^i \end{bmatrix}}_{\text{Vertices Level n}} = \underbrace{\begin{bmatrix} p_0^{i+1} \\ p_1^{i+1} \\ \vdots \\ p_n^{i+1} \end{bmatrix}}_{\text{Vertices Level n+1}}$$

Note : in fact, a scheme requires 2 matrices, one for inserting vertices (odd vertices) and one for moving existing ones (even vertices), the later being the most interesting for analysis

Adaptive Subdivision

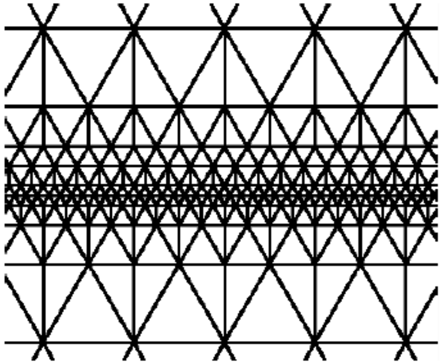
Spatially varying subdivision level on the surface

Pro.

- Subdivision mesh density control tailored using various criteria
  - Curvature
  - Distance
  - Visibility

Cons.

- Harder to implement
- Harder to analyze



Mesh combinatorics : adaptive subdivision must preserve topology (no cracks introduced).

Geometry Processing

Subdivision surfaces are useful for:

- Mesh to spline conversion
- Reconstruction from point sets
- Multiresolution
  - Analysis
  - Deformation
  - Rendering
- Parameterization
- Compression
- Simplification
- Remeshing

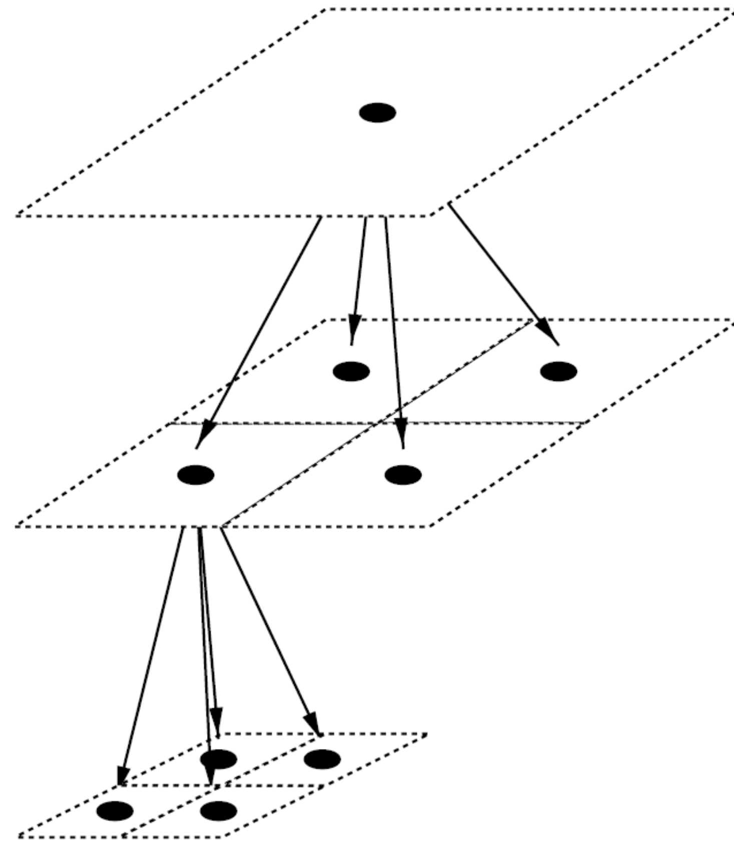
Implementation

Forest of quad-trees

- Each polygon of the base mesh a root
- Polygon subdivision = quad-tree

+ : adaptive subdivision

- : expensive and complicated



Implementation

Basis functions table (BFT)

Every point on a subdivision surface is a linear combination of a compact and local set of vertices on the base domain mesh.

1. Precompute the tables of weights for a each pair « valence/level »
2. Tessellate each polygon at desired level
3. Combine the relative base domain neighborhood using the BFT

+ : very fast, arbitrary scheme, GPU friendly

- : precomputation for each case, limited number of (rational) parameter points, adaptivity

Implementation

Hash-table

1. Build a hash-table
   - Indexed by mesh edges
   - Storing odd vertices newly inserted on edges
2. Iterate on polygons, accumulating their contribution on vertices and « edges vertices » (using the hash-table)
3. Replace each polygon by 4 new ones using the odd vertices stored in the hash-table and the even ones.

+ : straightforward to implement, works with various schemes

- : no multiresolution construction (mesh to mesh conversion)

Real Time

Dynamic coarse domain
Generate the subdivision mesh at each frame

GPGPU Rendering:

1. Subdivide 2 times on the GPU
2. Decompose in 2-rings patches
3. Unroll the rings (1D)
4. Patches them into a 2D texture
5. Render the texture on a quad, using twice higher screen resolution
   - Replace the image filtering kernel by the subdivision rules
6. Iterate until the desired level
7. Back conversion texture to mesh(1 pixel RGB = 1 vertex XYZ)

CUDA implementation

Visually plausible approximations:

- QAS (Loop)
- ACC (Catmull-Clark)

Curved PN Triangles

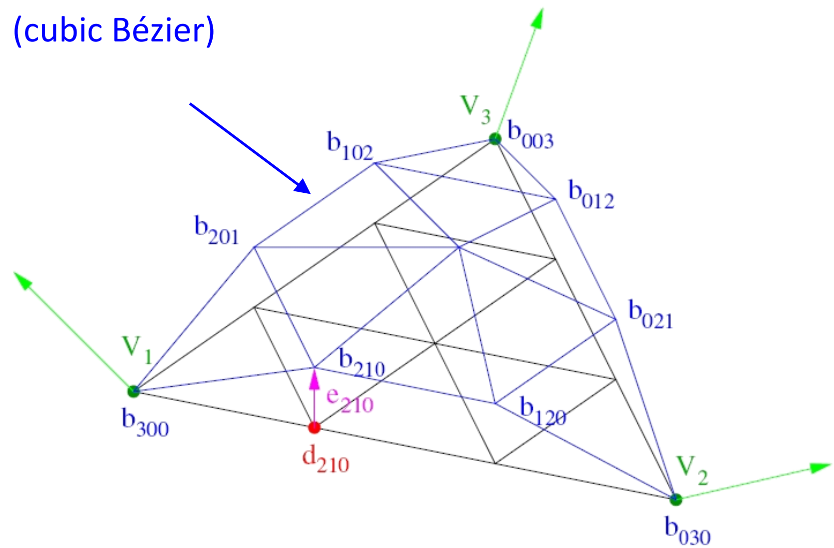
- Realtime Mesh Refinement
- **Visually smooth**
- Alternative to interpolating subdivision
- **Purely local**
- Vertex and Normal vectors used to drive the refinement
- Geometric continuity: C0 on edges

Curved PN Triangles

Principle

- generate a ***displacement field Cd*** and a ***normal field Cn*** on each triangle by creating 2 Bézier patches defined by positions and normals of each vertex of the triangle
  - **Cd**: cubic patch
  - **Cn**: quadratic patch
- Tessellate the triangle and *embed* the tessellation positions in Cd and normals in Cn
- Draw the resulting piece of mesh ***instead*** of the original triangle

Cd control network (cubic Bézier)



$$b_i = d_i + e_i$$

+ **No topology knowledge (i.e. no 1-ring structure)**

Compatible with GPU mesh format

- **Visual smoothness is not enough for some applications**

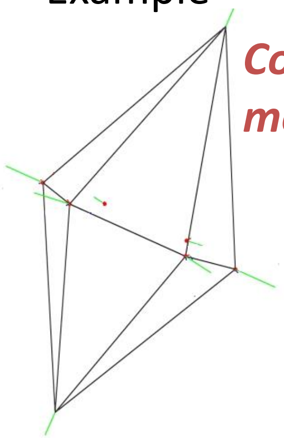
$$d_i = P_j + (P_k - P_j)/3. \quad e_i = \Pi(P_j, N_j, d_i)N_j$$

Barycentric coordinates

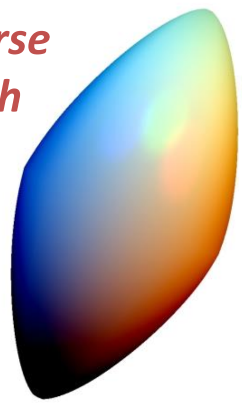
Projection on the PN plane of the nearest triangle

Curved PN Triangles

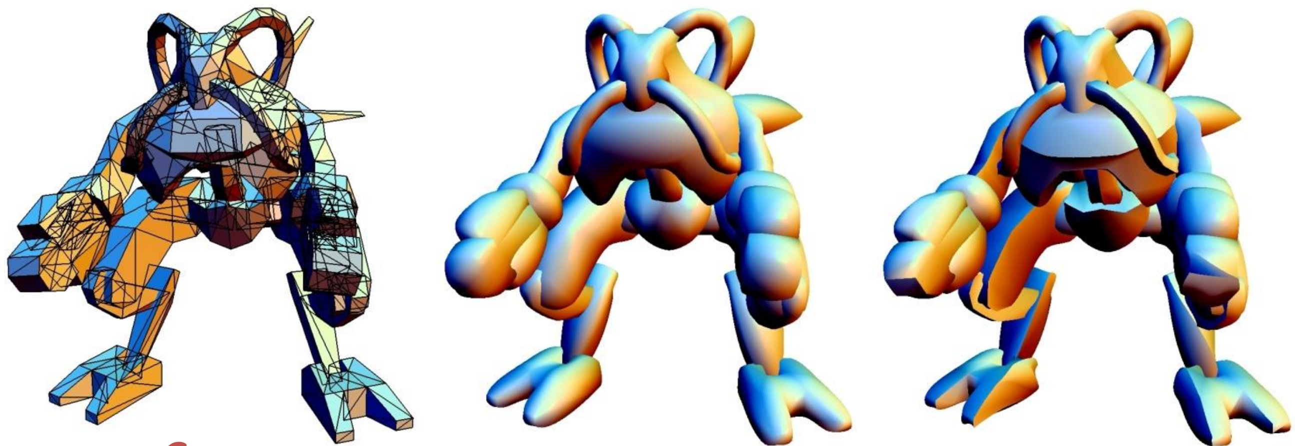
Example



Coarse mesh



Curved PN Triangles



Coarse mesh

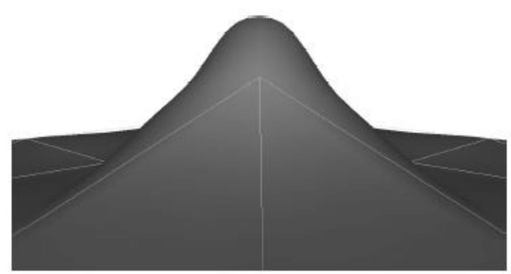
Scalar Tagged PN Triangles

Extension :

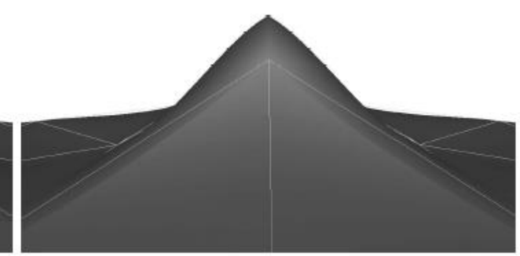
- **Scalar Tagged PN Triangle**
- **PN G1 Triangles**

Simpler operator :

- **Phong Tessellation**

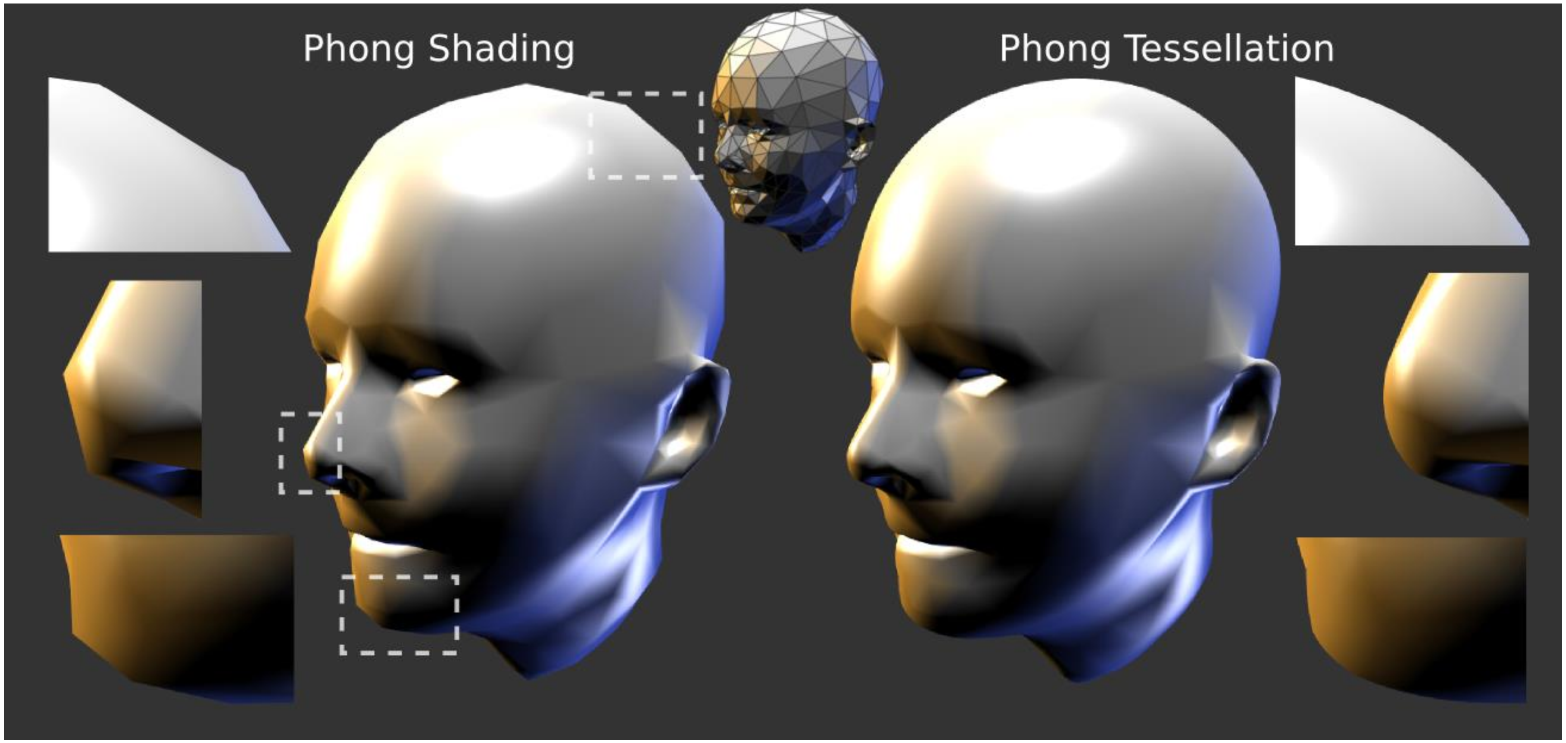


PNG1 Triangles



PN Triangles

Phong Tessellation



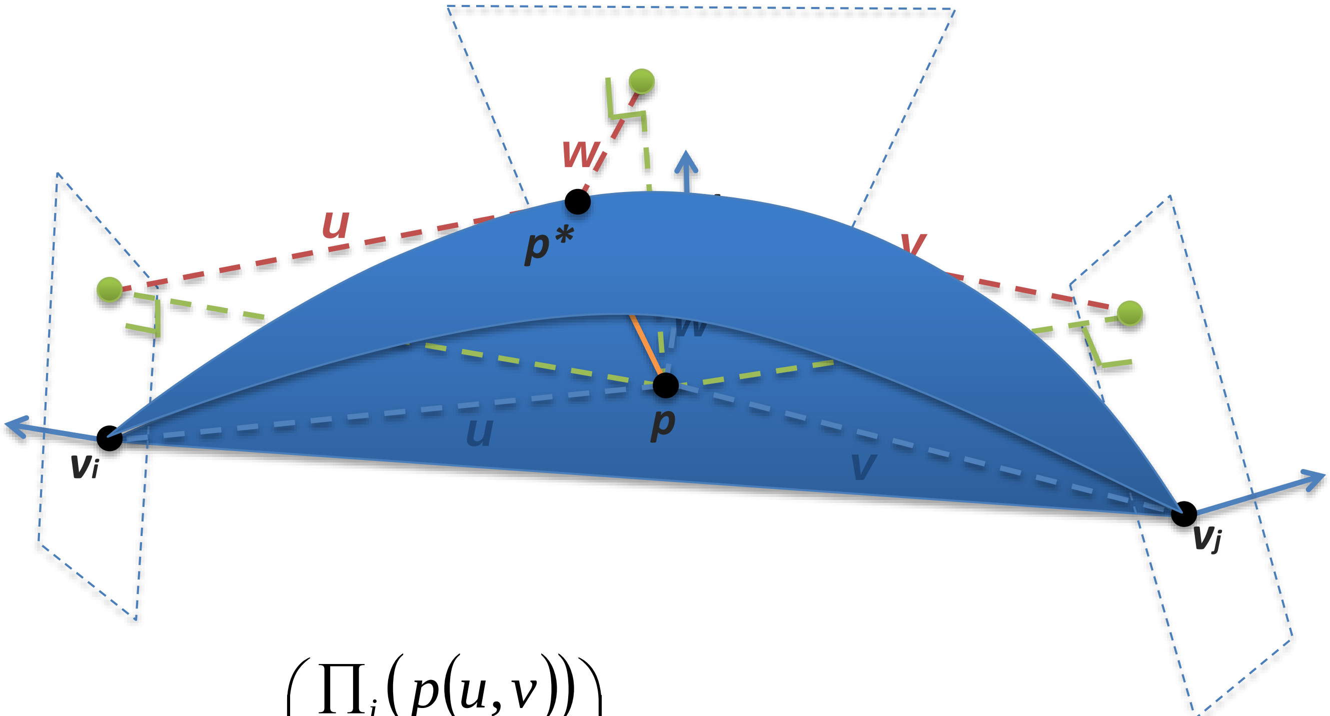
The faster visually smooth substitute to subdivision surfaces (CryEngine, Unity, Unreal Engine, etc)

Phong Tessellation

- Generate a curved displacement field

- A projection operator
  1. interpolate the 3 vertex positions
  2. project this point on the 3 tangent planes
  3. interpolate the 3 projections

Phong Tessellation

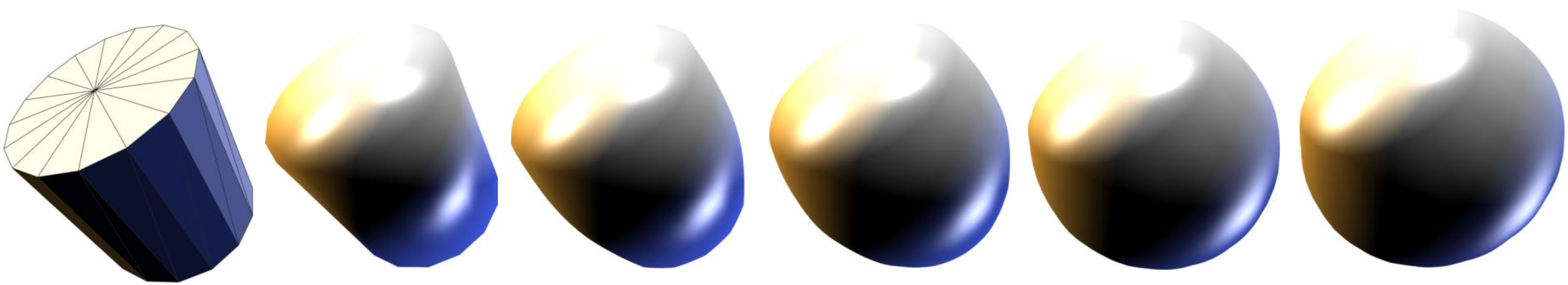


$$p^*(u,v) = (u,v,w) \bullet \begin{pmatrix} \prod_i \left(p(u,v)\right) \\ \prod_j \left(p(u,v)\right) \\ \prod_k \left(p(u,v)\right) \end{pmatrix}$$

Phong Tessellation

Modulate with a simple shape factor α

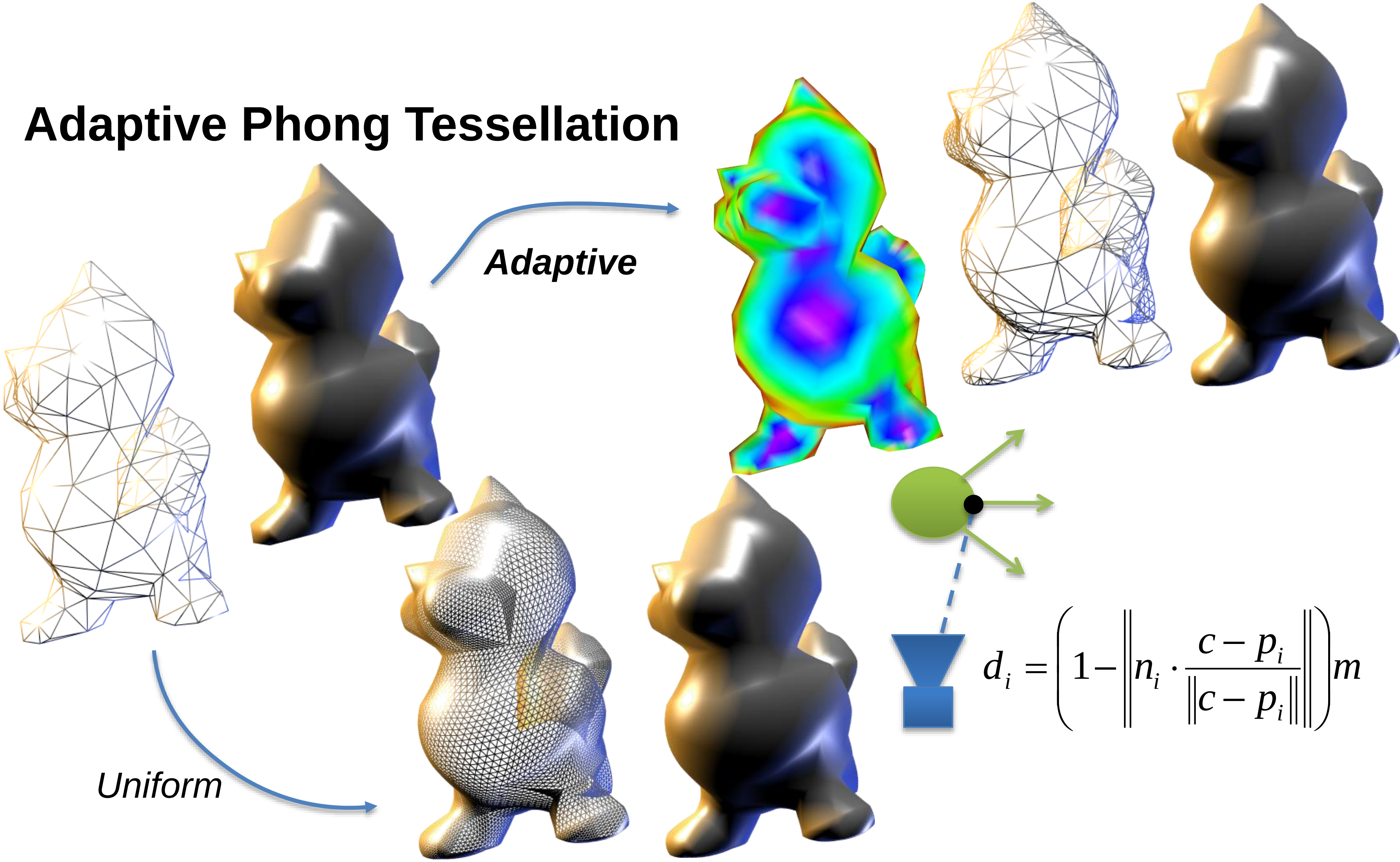
$$\mathbf{p}^{*}{}_{\alpha}(u,v) = (1-\alpha)\mathbf{p}(u,v) + \alpha(u,v,w)\begin{pmatrix}\pi_i(\mathbf{p}(u,v))\\ \pi_j(\mathbf{p}(u,v))\\ \pi_k(\mathbf{p}(u,v))\end{pmatrix}$$



Mesh α = 0 α = 1/4 α = 1/2 **α = 3/4** α = 1

Phong Tessellation

Adaptive Phong Tessellation



$$d_i = \left(1 - \left\| n_i \cdot \frac{c - p_i}{\|c - p_i\|} \right\|\right) m$$

Comparison to Subdivison

Modified Butterfly
Subdivision

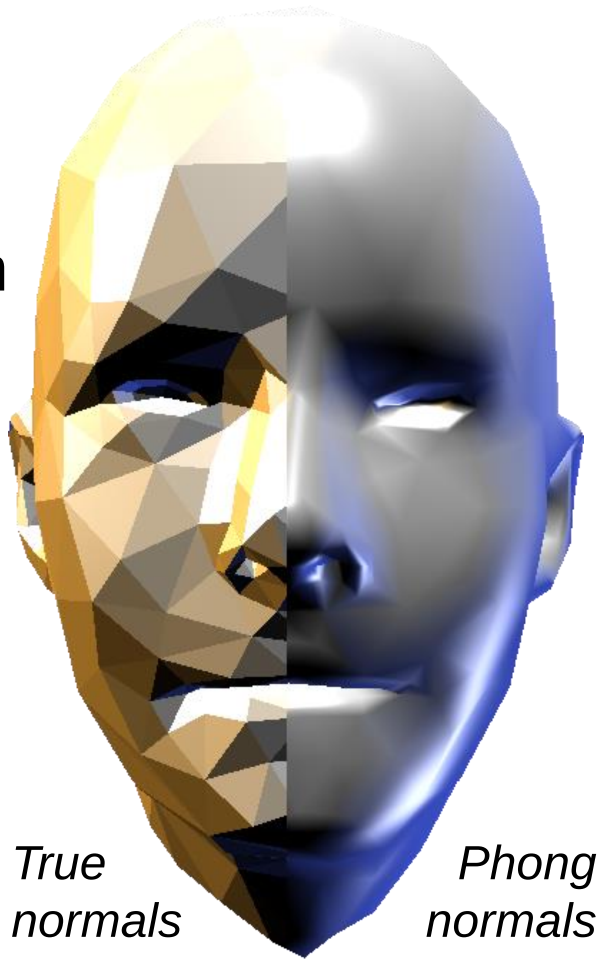
Curved PN Triangles

Phong Tessellation

Phong Tessellation

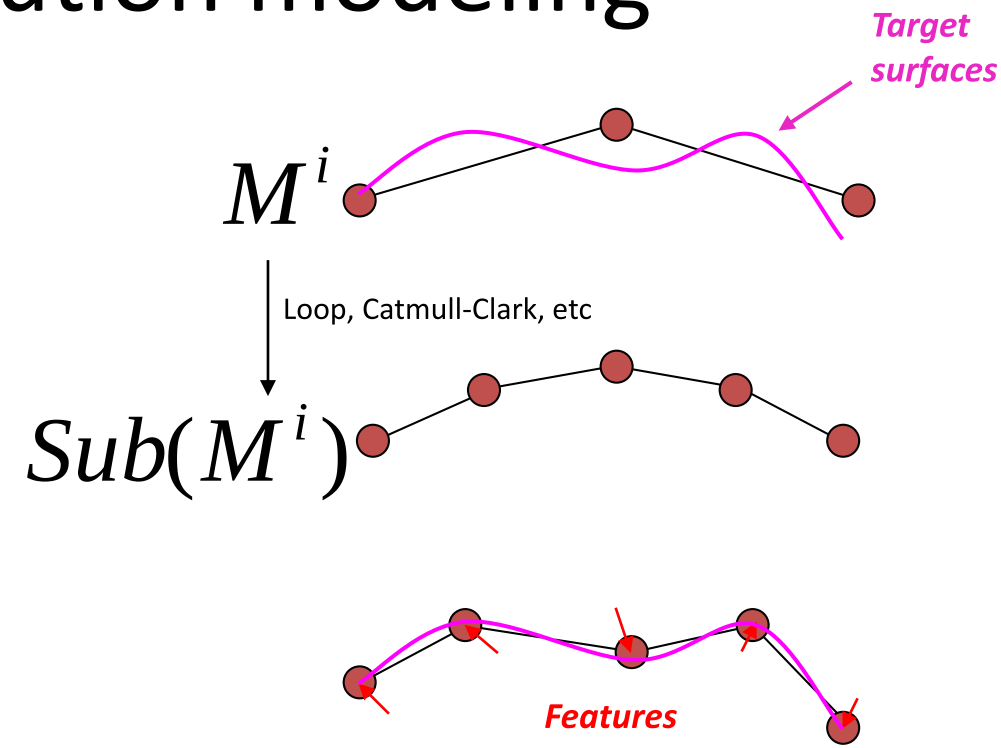
Remember "Phong Shading"

- Generate a continuous normal field on an arbitrary mesh
- Principle: interpolate vertex normals
- Purely local
  - 3 vertex normals per-triangle
- Linear [Phong 1975]
- Quadratic [Van Overveld 1997]



Multiresolution modeling

- Approximate a sampled surface (e.g., mesh, point set) with a subdivision surfaces
- *Principle: similar to wavelets*



$$M^{i+1} = Sub(M^i) + D^{i+1}$$

Subdivision mesh level i+1

Subdivision mesh level n

Displacement vector for each vertex

Subdivision Surfaces *Memo*

- **1978**:
  - First scheme, including the Catmull-Clark one (quads)
- **1987**:
  - Loop scheme: approximating scheme for triangle meshes
- **1994**:
  - Modifications on Loop (sharp creases, borders) in the context of surface reconstruction [Hoppe et al.]
- **1995**:
  - Multiresolution analysis of arbitrary meshes [Eck et al.]
  - Theoretical results on continuity [Reif et al.]
- **1996:**
  - Variational Subdivision Surfaces [Kobbelt]
- **1998**:
  - Subdivision Surface for Character Animation [DeRose et al.]
  - Exact parametric evaluation of Subdivision Surfaces discovered for Catmull-Clark and Loop schemes [Stam]
- **2000**:
  - SIGGRAPH course, 2nde version (**THE entry point**) [Zorin & Schröder]
  - Singularity control, normal control [Biermann et al.] (MPEG-4)
  - SQRT(3) scheme, *constructive* approach[Kobbelt]
  - **Displaced Subdivision Surfaces** [Lee et al.] (industrial spread started in 2004)
- **2001**:
  - Curved PN Triangle, a *fake* subdivision, more efficient and *visually* smooth [Vlachos et al.]
- **2001-2005**:
  - Factored subdivision methods (same algorithms and data-structure for many schemes)
  - Hybrid Triangle/Quad subdivision schemes

Subdivision Surface Rendering:

- **1996:** Table-based Subdivision evaluation [Pulli & Segal]
- **2003**: GPU evaluation of subdivision surfaces [Bolz & Schröder]
- **2005:** First full GPU implementation of subdivision
- **2007/8:** Efficient Realtime Substitutes