Subdivision Surfaces

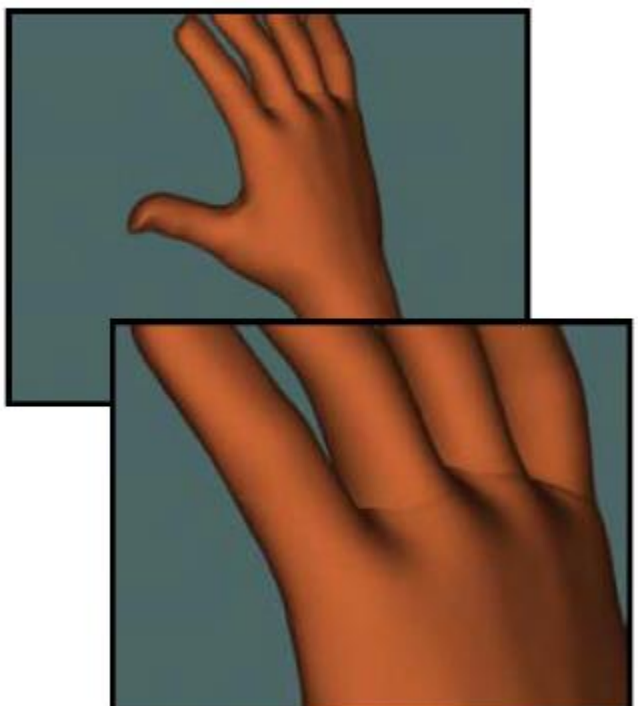
(widespread adoption in animation industry)

deRose et al., “Subdivision surfaces in character animation”

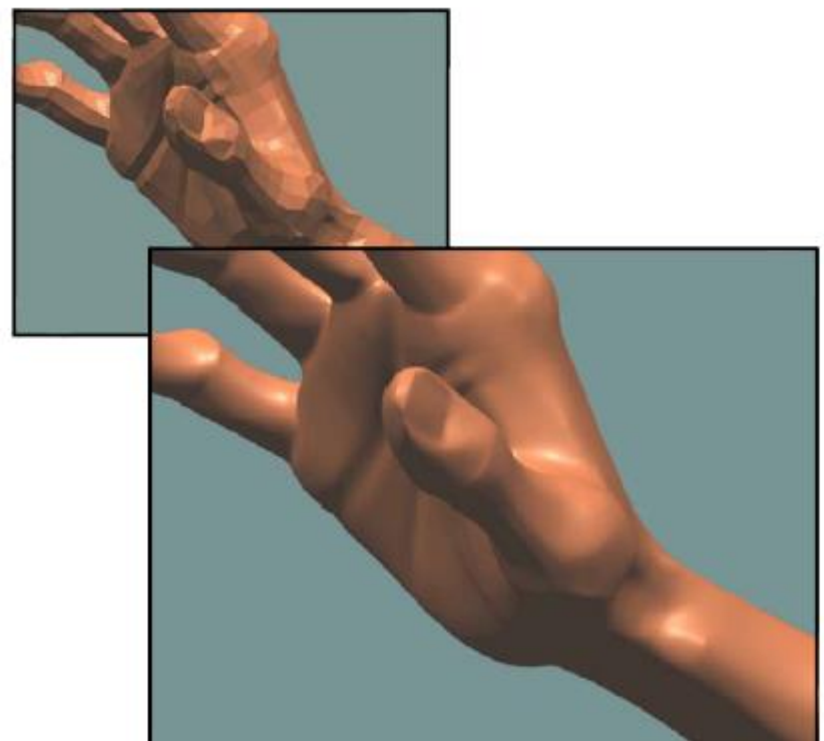
Problems with NURBS

- Difficult to maintain continuity between patches for arbitrary topology

Woody's hand (NURBS)



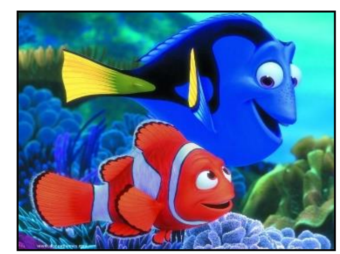
Geri's hand (subdivision)



Subdivision surfaces

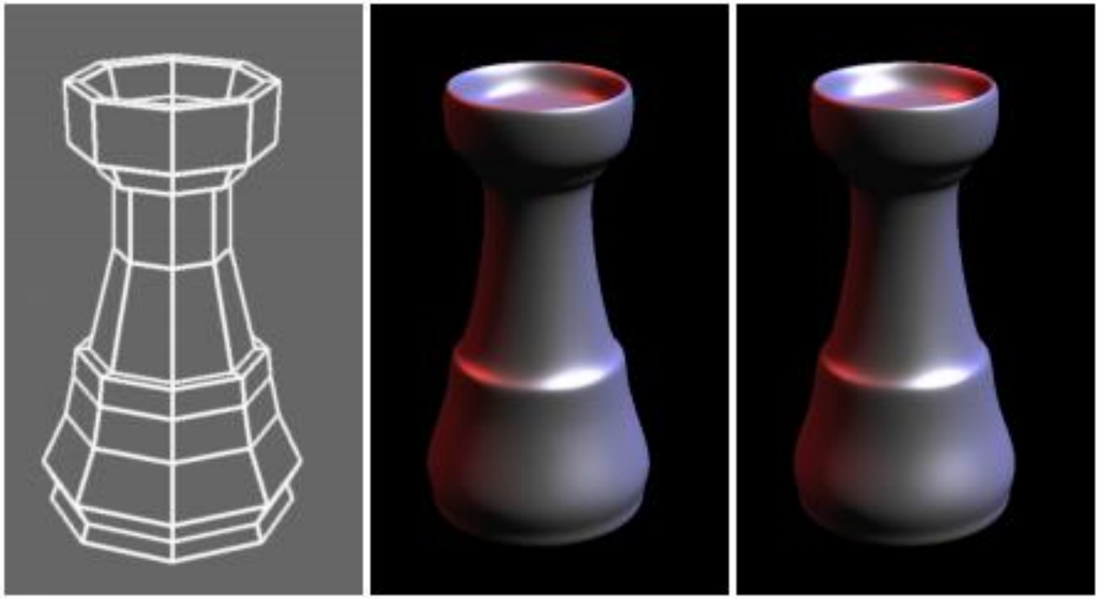
- Pixar first demonstrated subdivision surfaces in 1997 with *Geri's Game*.
  - Up until then they'd done everything in NURBS (Toy Story, A Bug's Life.)
  - From 1999 onwards everything they did was with subdivision surfaces (Toy Story 2, Monsters Inc, Finding Nemo...)

Geri's Game



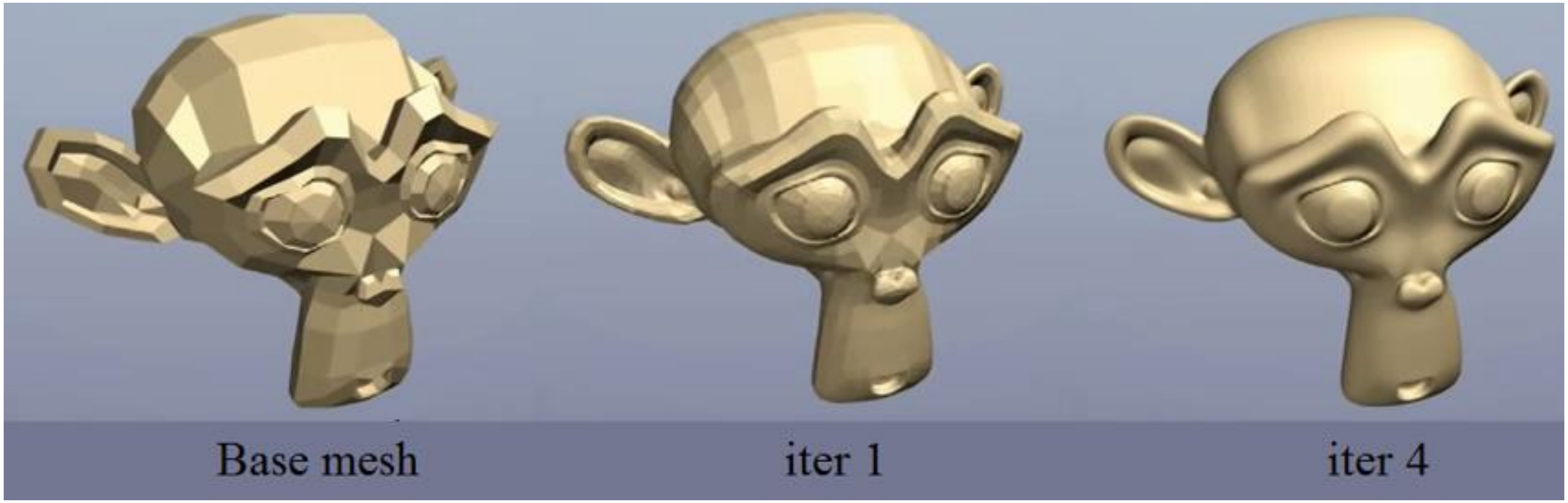
Subdivision Surfaces

- Subdivision surface: a method to represent a smooth surface via a coarser polygon mesh (base mesh).
- Each step of refinement adds new faces and vertices.
- The process converges to a smooth limit surface.



Initial mesh *Loop* *Catmull-Clark*

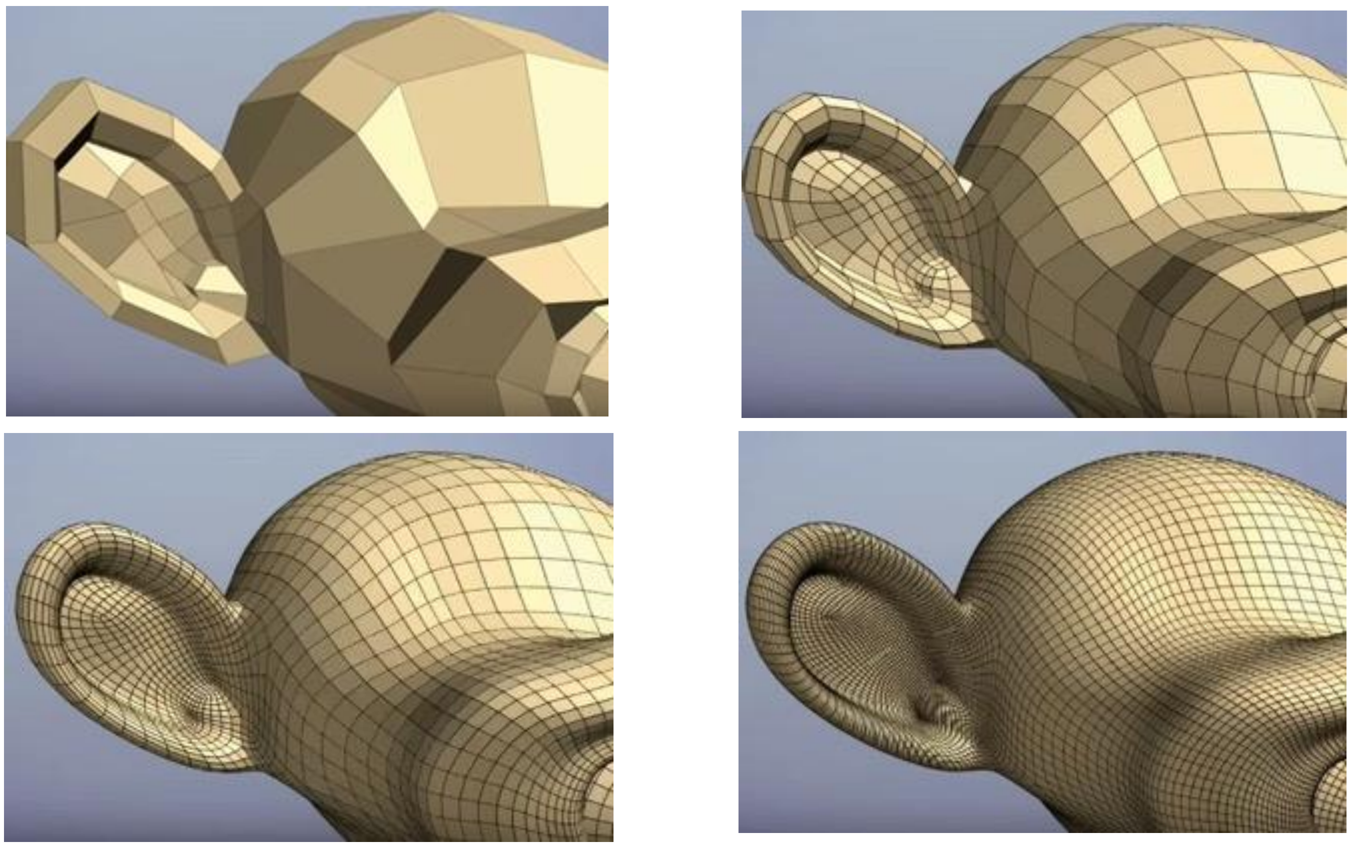
Subdivision Surfaces



Also useful for efficient distance-dependent rendering (Level Of Detail).

Subdivision Surfaces

- Not a (Gouraud) shading trick; actually changing the geometry of the model.



Subdivision Curves

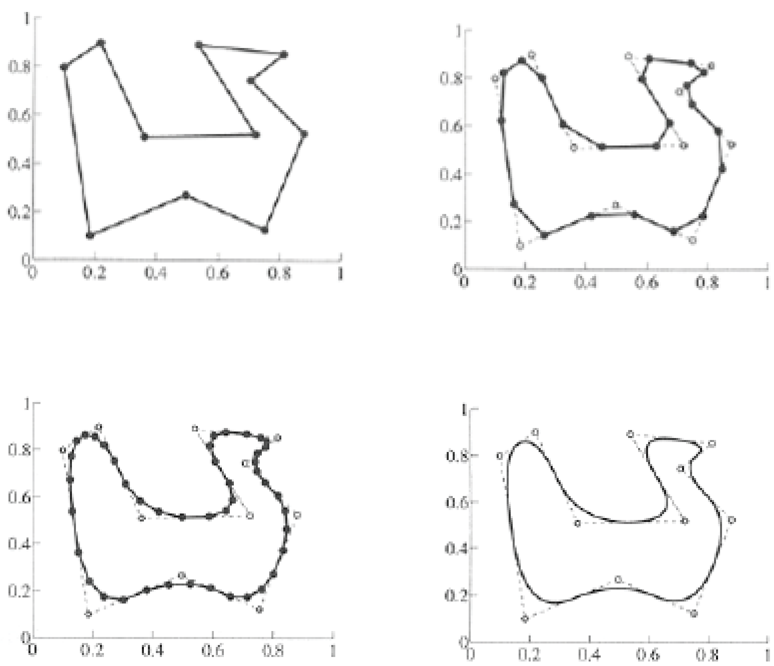
Idea:

- repeatedly refine the control polygon

$$P^1 \to P^2 \to P^3 \to \cdots$$

- curve is the limit of an infinite process

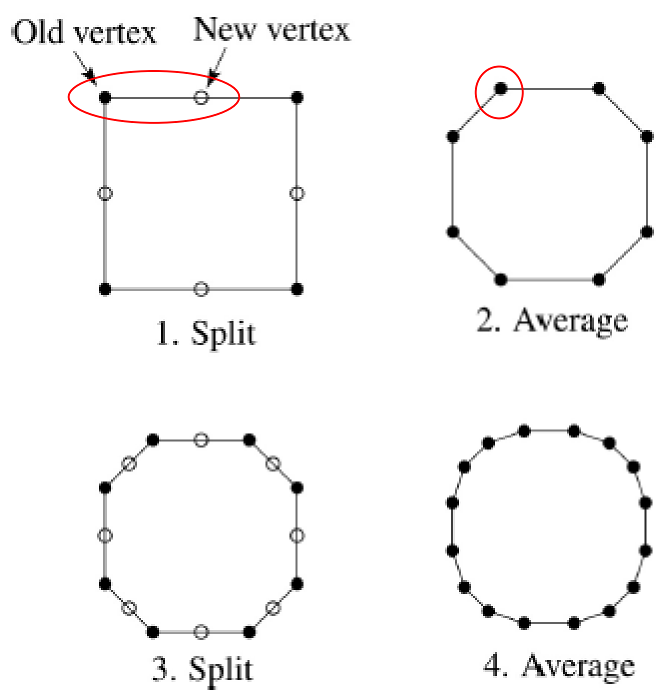
$$Q = \lim_{j \to \infty} P^j$$



Chaikin's algorithm

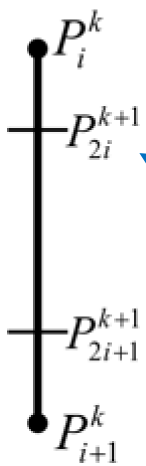
Chaikin introduced the following "corner-cutting" scheme in 1974:

- Start with a piecewise linear curve
- **Splitting step**: Insert new vertices at the midpoints
- **Averaging step**: Average each vertex with the "next" (clockwise) neighbor
- Repeat the process



Subdivision Curves

- Chaikin can be coded (combining splitting & averaging) as follows :



$$P_{2i}^{k+1} = (3/4)P_i^k + (1/4)P_{i+1}^k$$

$$P_{2i+1}^{k+1} = (1/4)P_i^k + (3/4)P_{i+1}^k$$

- k is the generation. Each generation has twice as many control points as before.
- Boundaries, if any, are treated specially.
- The limit curve is a **quadratic B-spline**!

Subdivision Curves

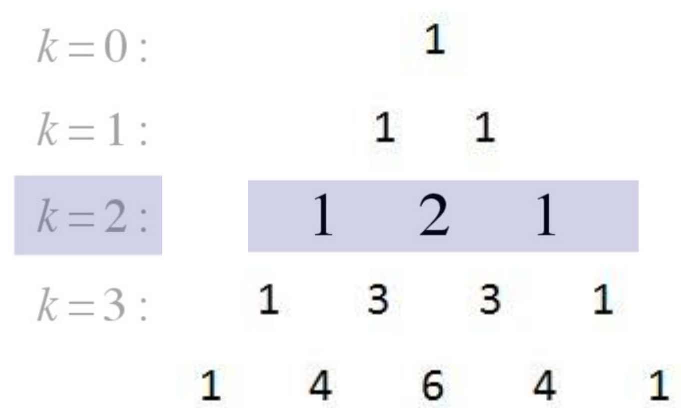
- Lane-Riesenfeld scheme
  - Insert midpoint of each edge
  - Use row *k* of Pascal's triangle (normalized to 1) as **averaging mask**
  - Limit is B-spline of degree *k*+1

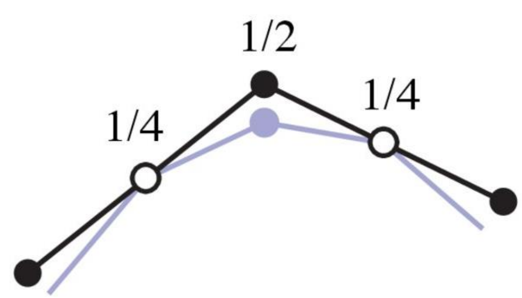
k=1: **quadratic** curve

$$r = \tfrac{1}{2}\,(1, 1)$$

k=2: **cubic** curve

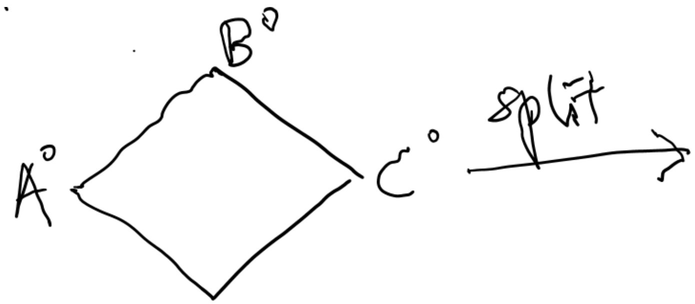
$$r = \tfrac{1}{4}\,(1, 2, 1)$$

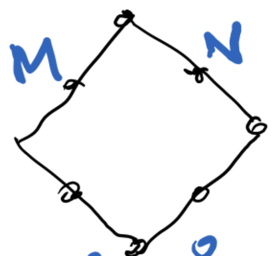




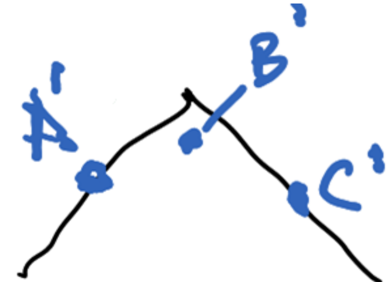
Subdivision curves

Combining split and averaging steps:





average



$M = \frac{A^0 + B^0}{2}$

$\frac{1}{4}(1, 2, 1)$ rule

$A^1 = M$

$B^1 = \frac{1M + 2B^0 + 1N}{4}$

$= \frac{1A^0 + 6B^0 + 1C^0}{8}$

1

subdivide

$\frac{1}{8}(1, 6, 1)$

When to stop dividing?

- After each split-average step, we are closer to the **limit curve**.
- How many steps until we reach the final (limit) position?
  - Infinite subdivision!
- Can we push a vertex to its **limit position** without infinite subdivision? Yes!

Recipe for subdivision curves

- After subdividing and averaging a few times to get *sufficient vertices*, push each vertex to its **limit position** by applying an **evaluation mask**.

- Each subdivision scheme has its own evaluation mask, which is mathematically determined by analyzing the subdivision and averaging rules, using eigenanalysis

- For cubic B-spline subdivision, the evaluation mask is:

$$\frac{1}{6}\begin{pmatrix}1 & 4 & 1\end{pmatrix}$$

Numberphile video

Recipe for subdivision curves

Now we can cook up a simple procedure for creating subdivision curves:

1) Subdivide (split+average) the control polygon a few times. Use the **averaging mask**.

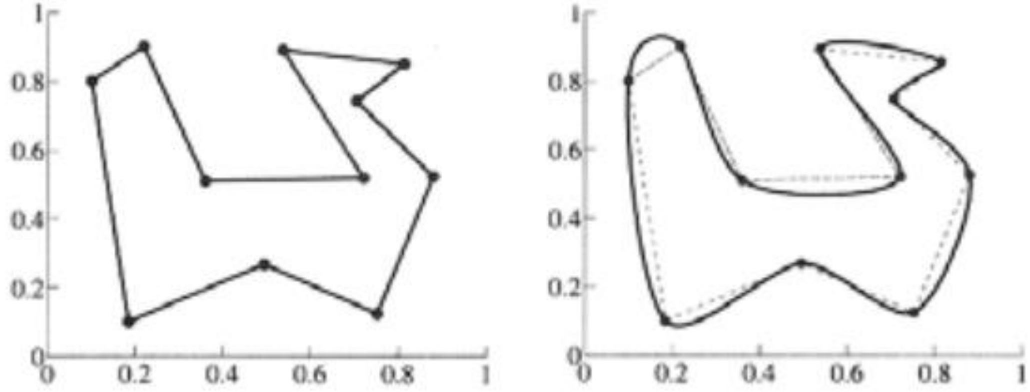
2) Push the resulting points to the limit positions. Use the **evaluation mask**.

DLG Interpolating Scheme (1987)

- Slight modification to subdivision algorithm:
  - Splitting step introduces midpoints
  - *Averaging step only changes midpoints*

- For DLG (Dyn-Levin-Gregory), use:

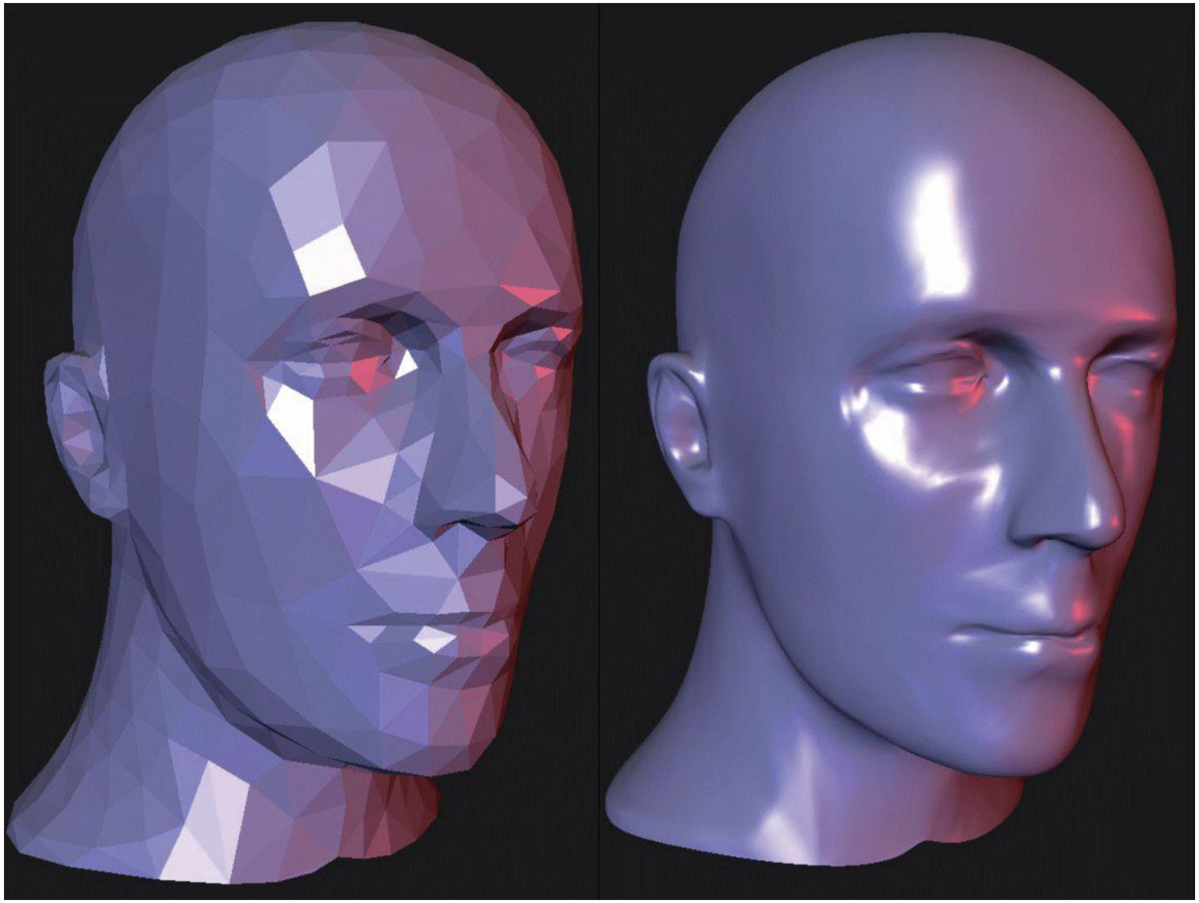
$$r_{\text{old}} = (1) \qquad r_{\text{new}} = \frac{1}{16}(-2, 5, 10, 5, -2)$$



- Since we are only changing the midpoints, the points after the averaging step do not move.

Building complex models

- We can extend the idea of subdivision from curves to surfaces…

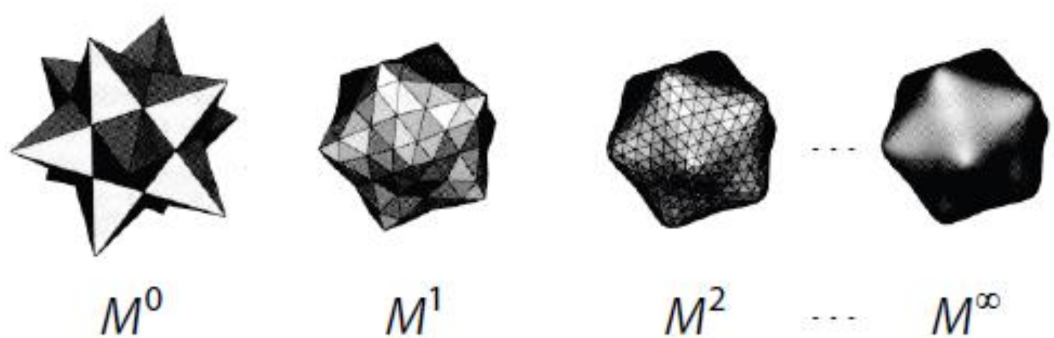


Subdivision surfaces

- Chaikin's use of subdivision for curves inspired similar techniques for subdivision surfaces.
- Iteratively refine a **control polyhedron** to produce the limit surface

$$S = \lim_{j \to \infty} M^j$$

using splitting and averaging steps.



Recipe for subdivision surfaces

As with subdivision curves, we can now describe a recipe for *creating* and *rendering* subdivision surfaces:

- Use the averaging mask to subdivide (split+average) the control polyhedron a few times.
- *Compute two **tangent vectors** using the tangent masks.*
- *Compute the **normal** from the tangent vectors.*
- Use the evaluation mask to push the resulting points to the limit positions.
- Render!

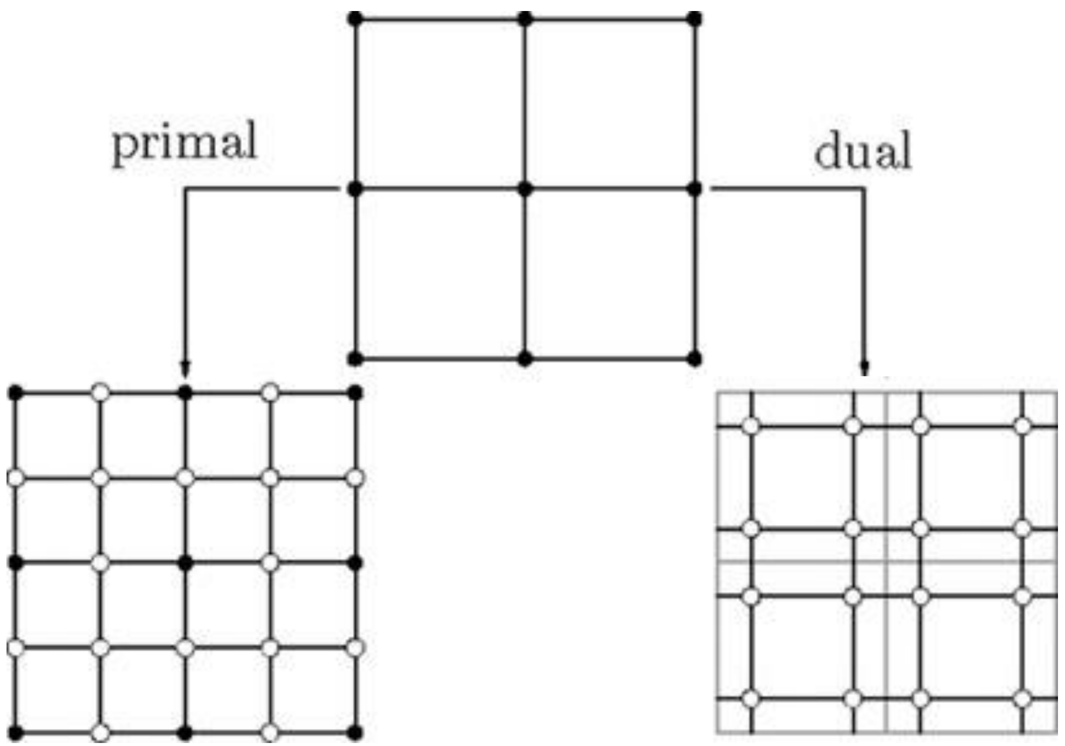
Subdivision Zoo

- There are a variety of subdivision schemes
- Most widely used are Catmull-Clark and Loop schemes

	Primal		Dual
	Triangles	**Rectangles**	
Approximating	Loop	**Catmull-Clark**	Doo-Sabin Midedge
Interpolating	Butterfly	Kobbelt	

Primal vs. Dual

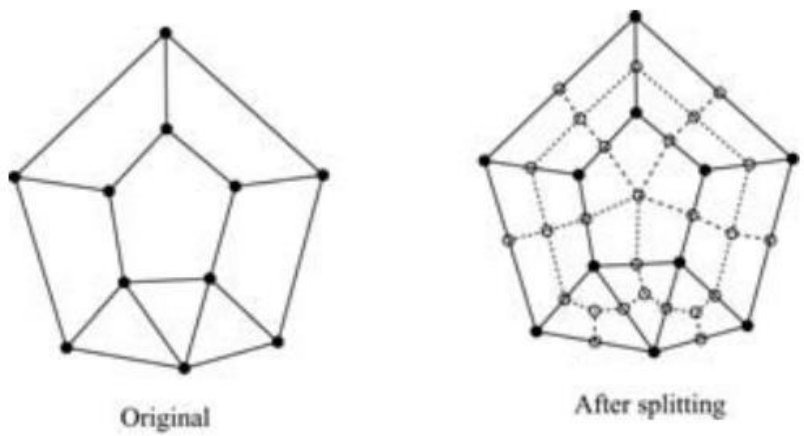
- Primal subdivision schemes split faces
- Dual subdivision schemes splits vertices



Catmull-Clark

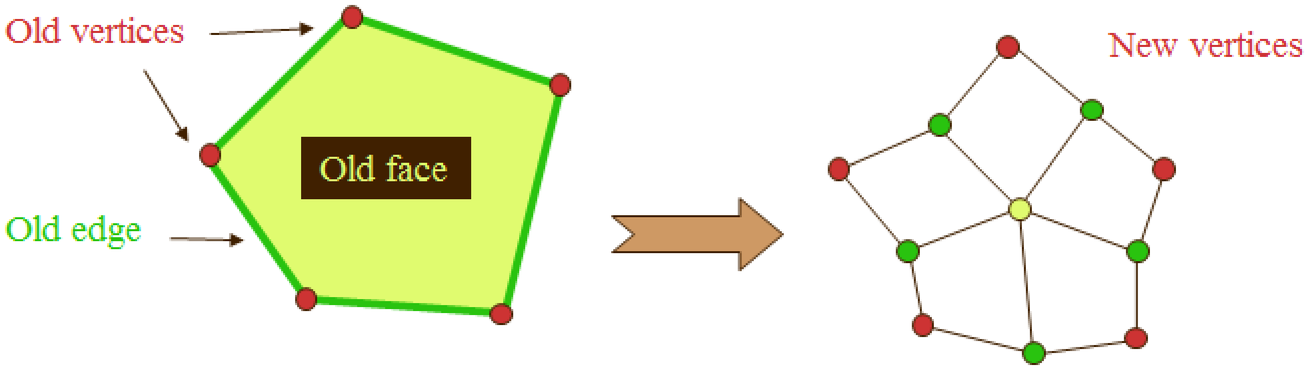
Doo-Sabin

Catmull-Clark Subdivision



- It is a primal, approximation subdivision scheme
- Applied to meshes with polygons of any # of sides
  - First iteration splits every polygon into quadrilaterals
- Limit surfaces are **bi-cubic B-splines**
- C^2 continuous limit surfaces except at extraordinary points:
  - C^1 at extraordinary points (vertices with valence $\neq 4$)

Catmull-Clark subdivision

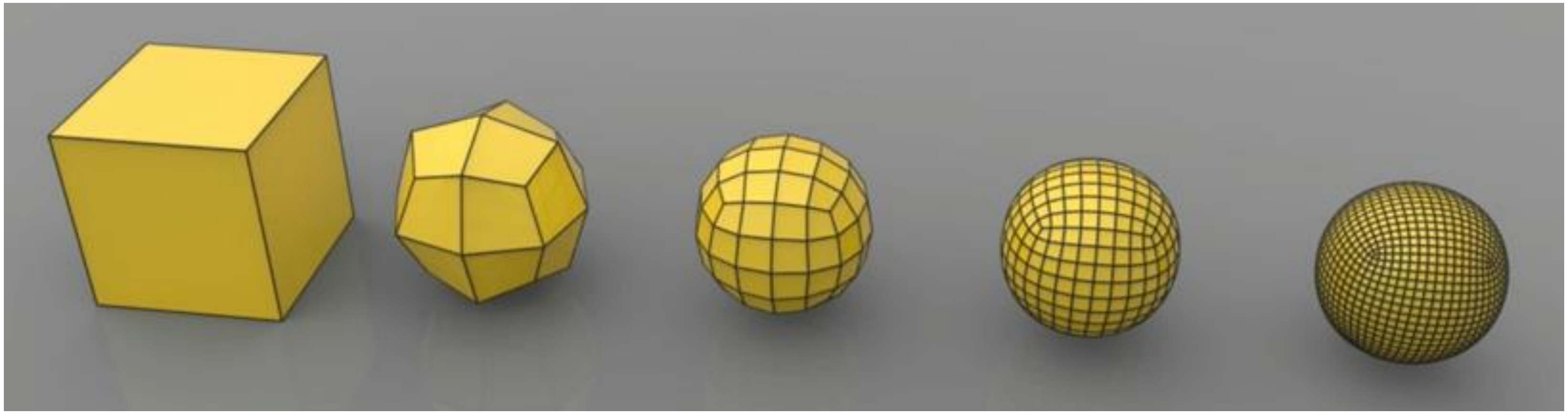


There are three kinds of new vertices:

- Yellow vertices are associated with old faces
- Green vertices are associated with old edges
- Red vertices are associated with old vertices.

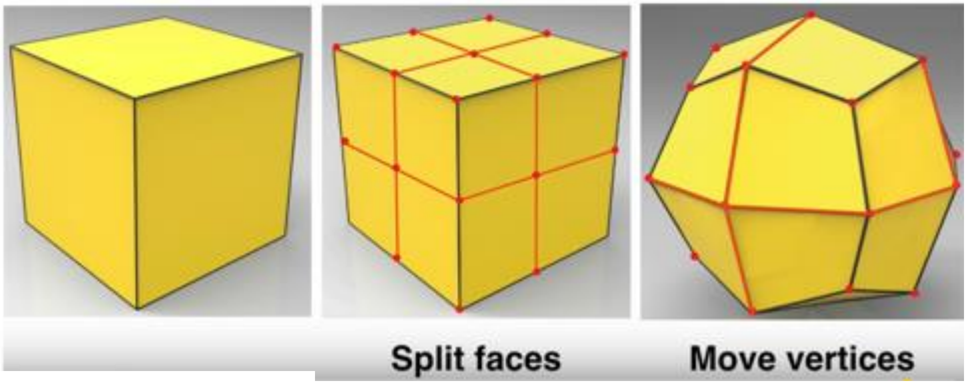
Subdivision Surfaces

Catmull-Clark subdivision to refine quad surfaces/meshes.

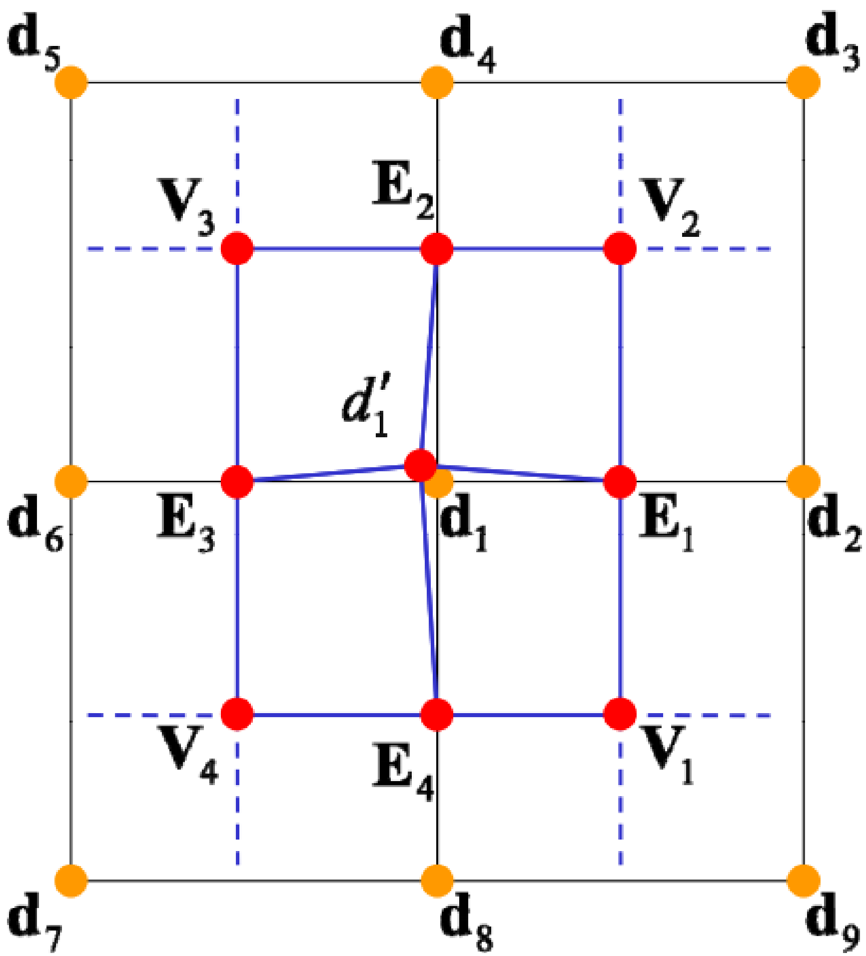


Base mesh

Catmull-Clark
limit surface



Catmull-Clark Subdivision



$$\mathbf{V}_2 = \frac{1}{n} \times \sum_{j=1}^{n} \mathbf{d}_j$$

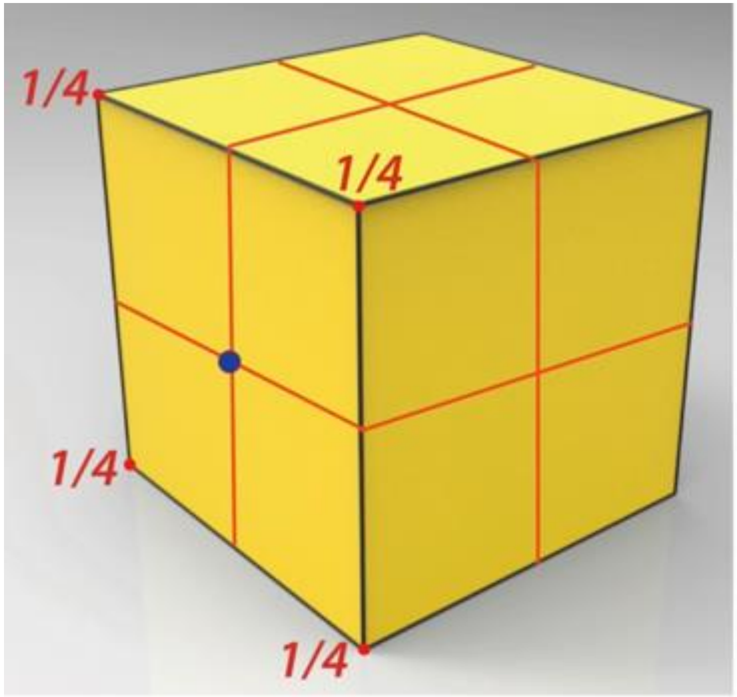
$$\mathbf{E}_i = \frac{1}{4}\left(\mathbf{d}_1 + \mathbf{d}_{2i} + \mathbf{V}_i + \mathbf{V}_{i+1}\right)$$

$$\mathbf{d}'_1 = \frac{(n-3)}{n}\mathbf{d}_1 + \frac{2}{n}\mathbf{R} + \frac{1}{n}\mathbf{S}$$

$$\mathbf{R} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{E}_i \quad \mathbf{S} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{V}_i$$

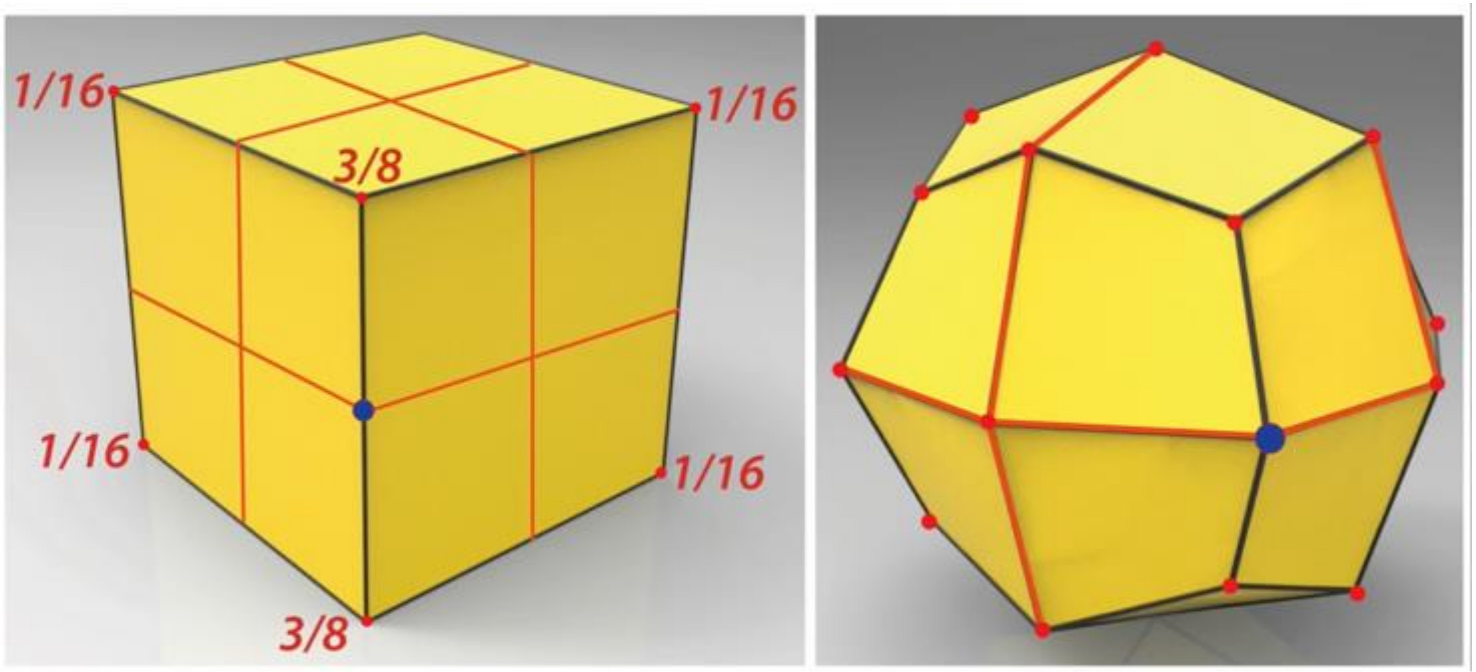
Let’s expand these equations into ready-to-code statements

Subdivision Surfaces



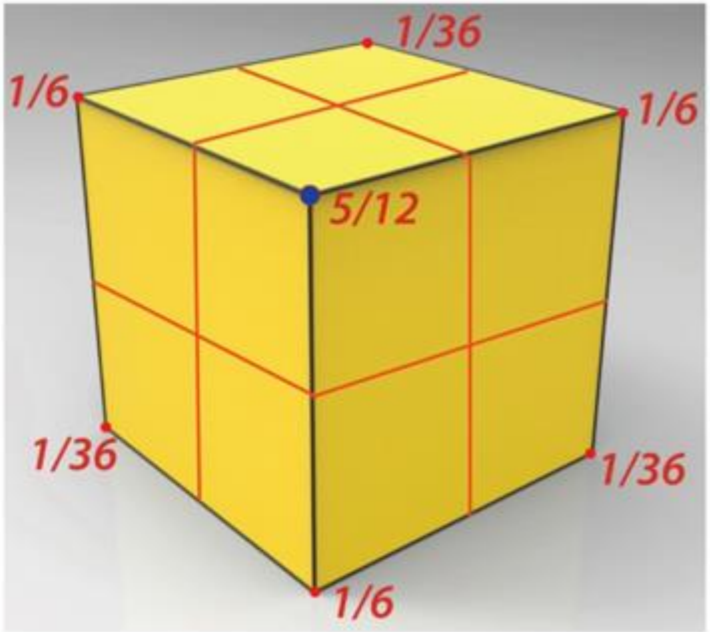
- A face point is computed using equal weights of its adjacent points.

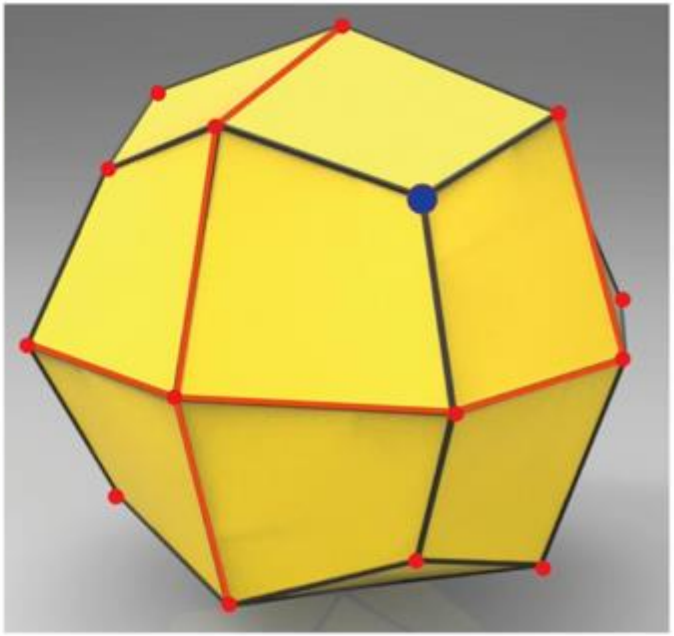
Subdivision Surfaces



- An edge point is computed using these 6 weights.

Subdivision Surfaces

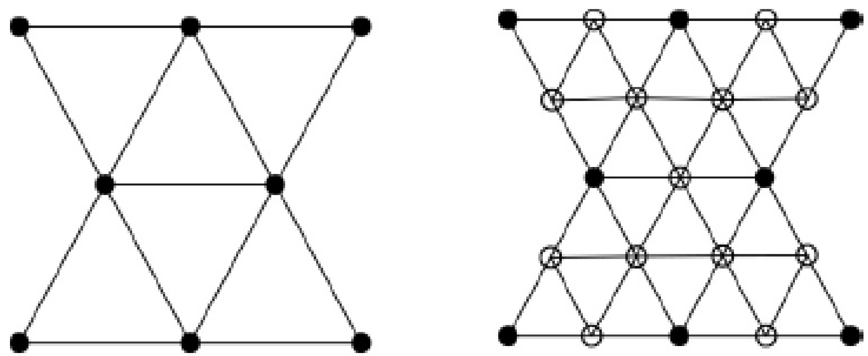




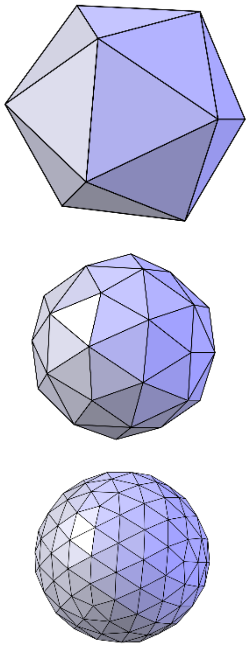
A vertex point is computed using these 7 weights.

Loop subdivision

- A common choice for triangle meshes is 4:1 subdivision – each triangular face is split into four sub-faces:



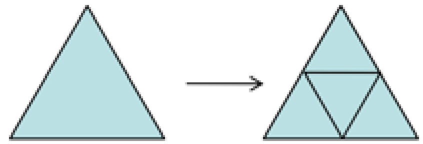
Original After splitting

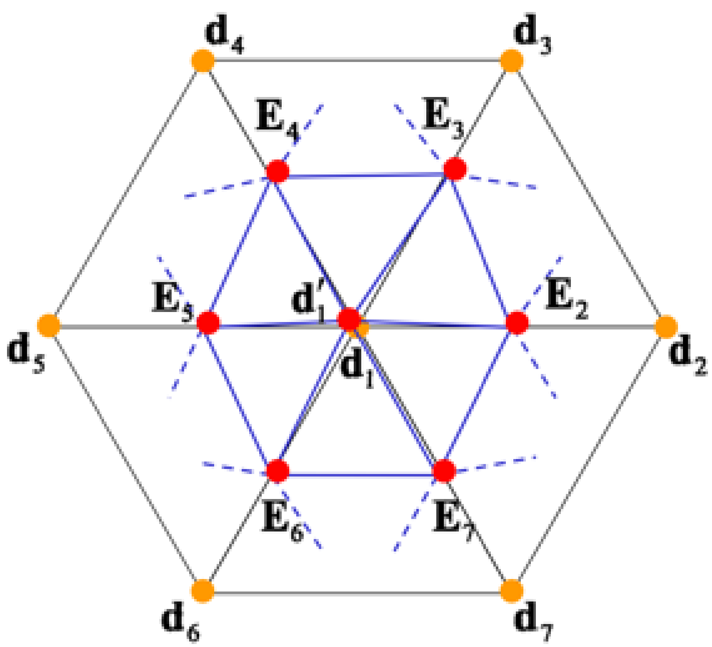


- The valence of each internal vertex is 6 for triangle meshes
- Limit surface is C^2 except at ***extraordinary*** vertices (valence not equal 6)

Loop Subdivision

Loop subdivision to refine triangular surfaces/meshes.





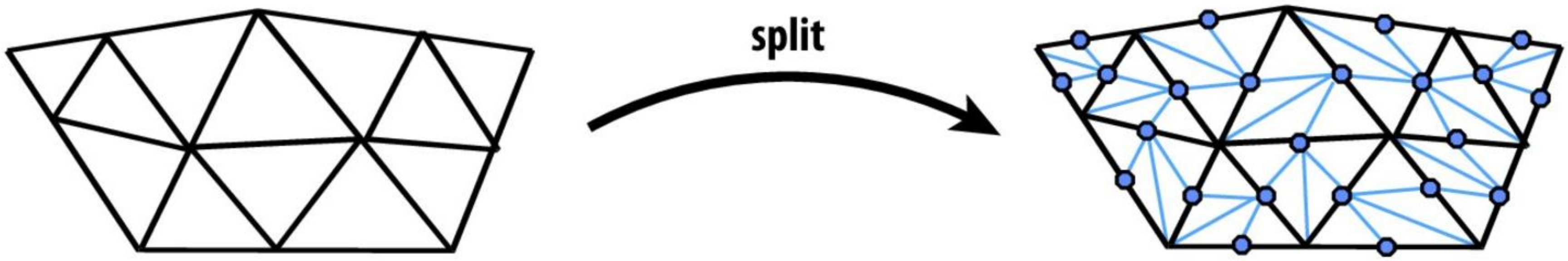
$$E_i = \frac{3}{8}(d_1 + d_i) + \frac{1}{8}(d_{i-1} + d_{i+1})$$

$$\mathbf{d}_1' = \alpha_n \mathbf{d}_1 + \frac{(1-\alpha_n)}{n}\sum_{j=2}^{n+1} \mathbf{d}_j$$

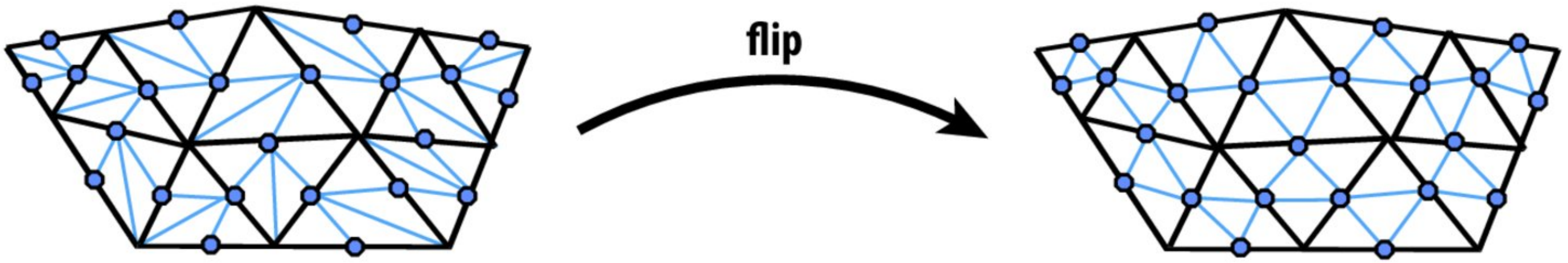
$$\alpha_n = \frac{3}{8} + \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2$$

Loop Subdivision via Edge operations

- **First, split edges of original mesh in *any* order:**



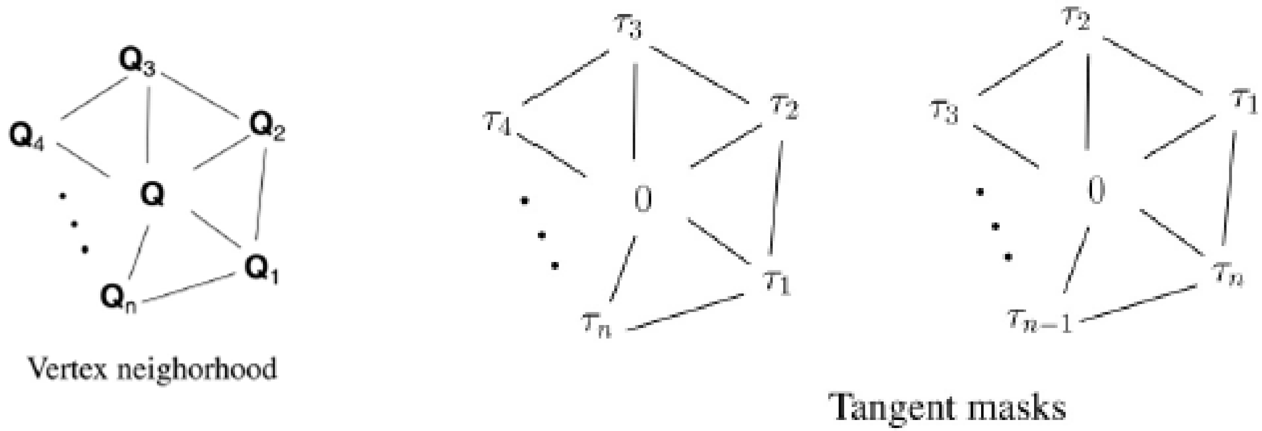
- **Next, flip new edges that touch a new & old vertex:**



(Don't forget to update vertex positions!)

Loop tangent masks

- How do we compute the normal?
- Find two tangent vectors, then take the cross product.



$$\mathbf{T}_1^{\infty} = \tau_1(n)\mathbf{Q}_1 + \tau_2(n)\mathbf{Q}_2 + \cdots + \tau_n(n)\mathbf{Q}_n$$

$$\mathbf{T}_2^{\infty} = \tau_n(n)\mathbf{Q}_1 + \tau_1(n)\mathbf{Q}_2 + \cdots + \tau_{n-1}(n)\mathbf{Q}_n$$

where

$$\tau_i(n) = \cos(2\pi i/n)$$

√3-subdivision

- √3-subdivision to refine triangular surfaces/meshes.
  - https://www.graphics.rwth-aachen.de/media/papers/sqrt31.pdf

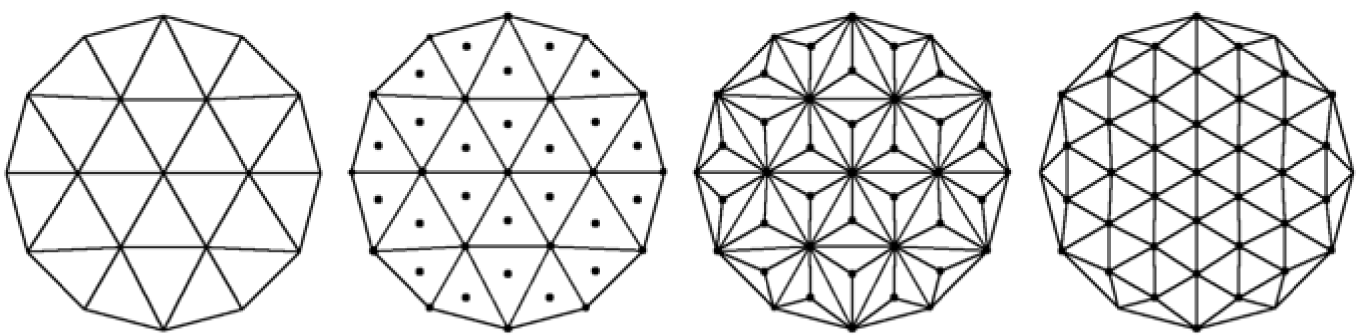


FIGURE 5. $\sqrt{3}$ Subdivision. From left to right: original mesh, added vertices at the midpoints of the faces (step 1), connecting the new points to the original mesh (step 1), flipping the original edges to obtain a new set of faces (step 3). Step 2 involves shifting the original vertices and is not shown.

- Step 2: move each original vertex v to a new position p by averaging v with the positions of its original neighboring vertices v_i for $0 \le i \le n-1$.

$$\mathbf{p} = (1 - a_n)\mathbf{v} + \frac{a_n}{n}\sum_{i=0}^{n-1}\mathbf{v}_i \qquad a_n = \frac{4 - 2\cos\left(\frac{2\pi}{n}\right)}{9}$$

√3-subdivision

√3-subdivision.

Increases the number of triangles slower than Loop's. Yields a finer gradation of hierarchy levels

Loop subdivision.

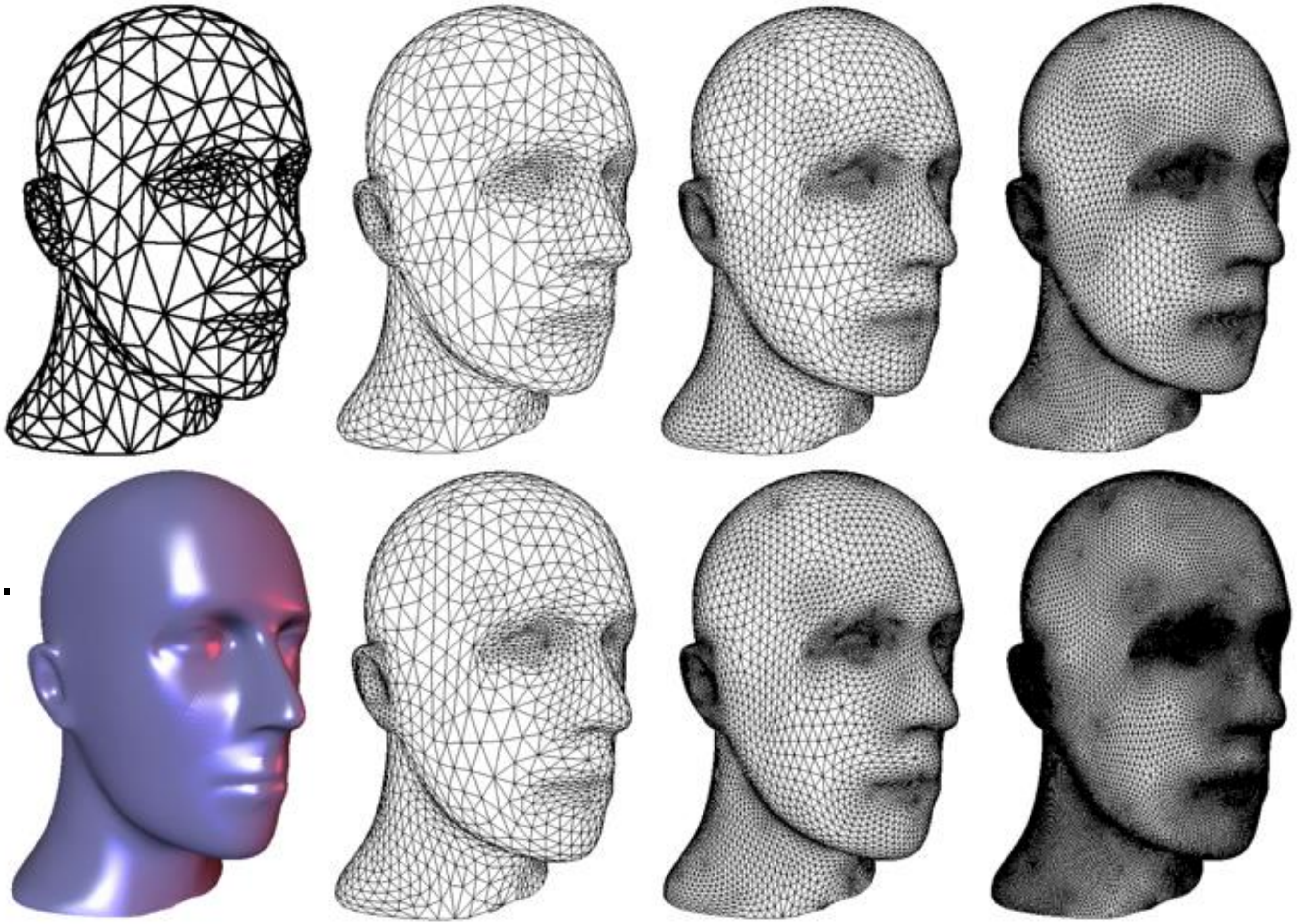
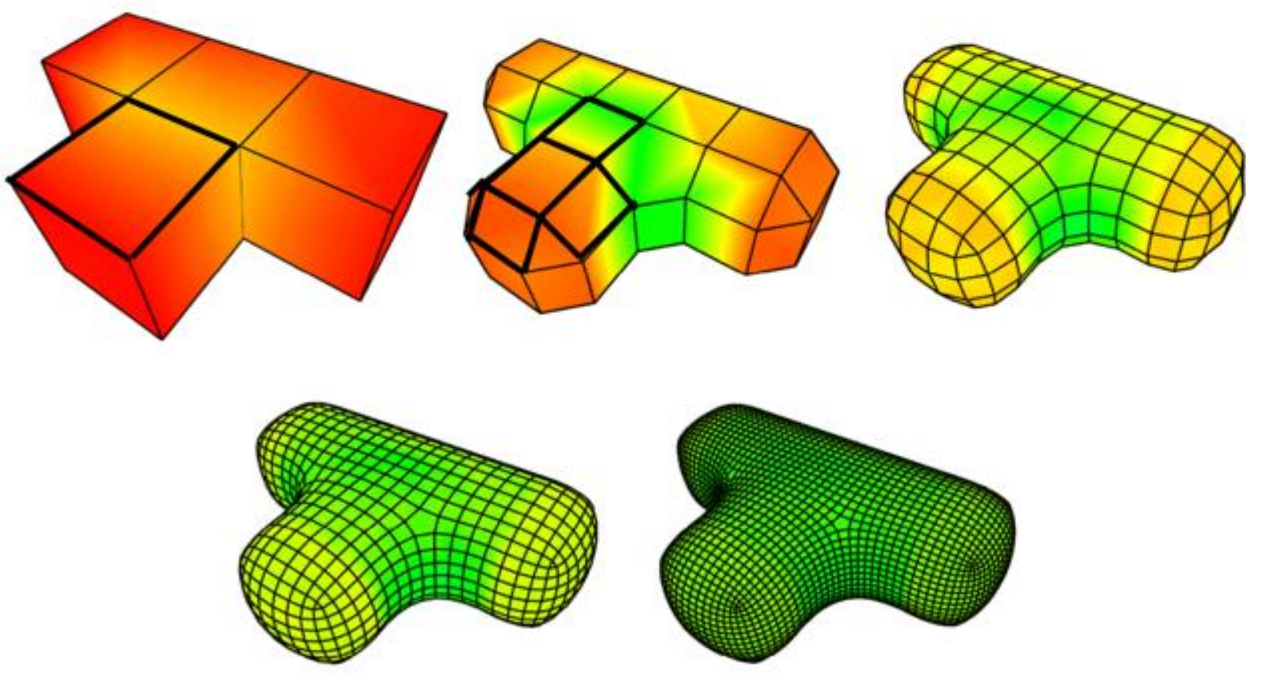


Figure 13: *Sequences of meshes generated by the $\sqrt{3}$-subdivision scheme (top row) and by the Loop subdivision scheme (bottom row). Although the quality of the limit surfaces is the same (C^2), $\sqrt{3}$-subdivision uses an alternative refinement operator that increases the number of triangles slower than Loop's. The relative complexity of the corresponding meshes from both rows is (from left to right) $\frac{3}{4} = 0.75$, $\frac{9}{16} = 0.56$, and $\frac{27}{64} = 0.42$. Hence the new subdivision scheme yields a much finer gradation of uniform hierarchy levels.*

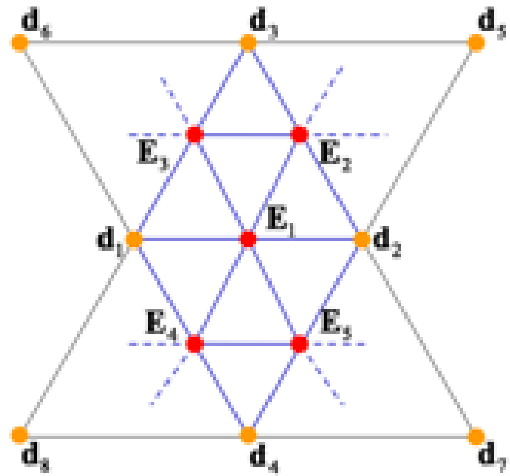
Doo-Sabin Subdivision

- A dual scheme that output a k-sided polygon for each original k-valence vertex.
- A generalization of bi-quadratic uniform B-splines



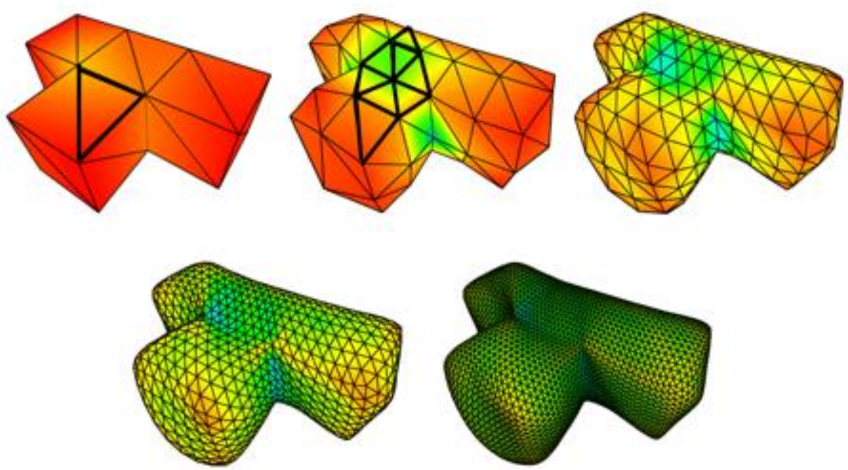
Interpolating Subdivision Surfaces

- Butterfly subdivision to refine triangular surfaces/meshes (interpolating).

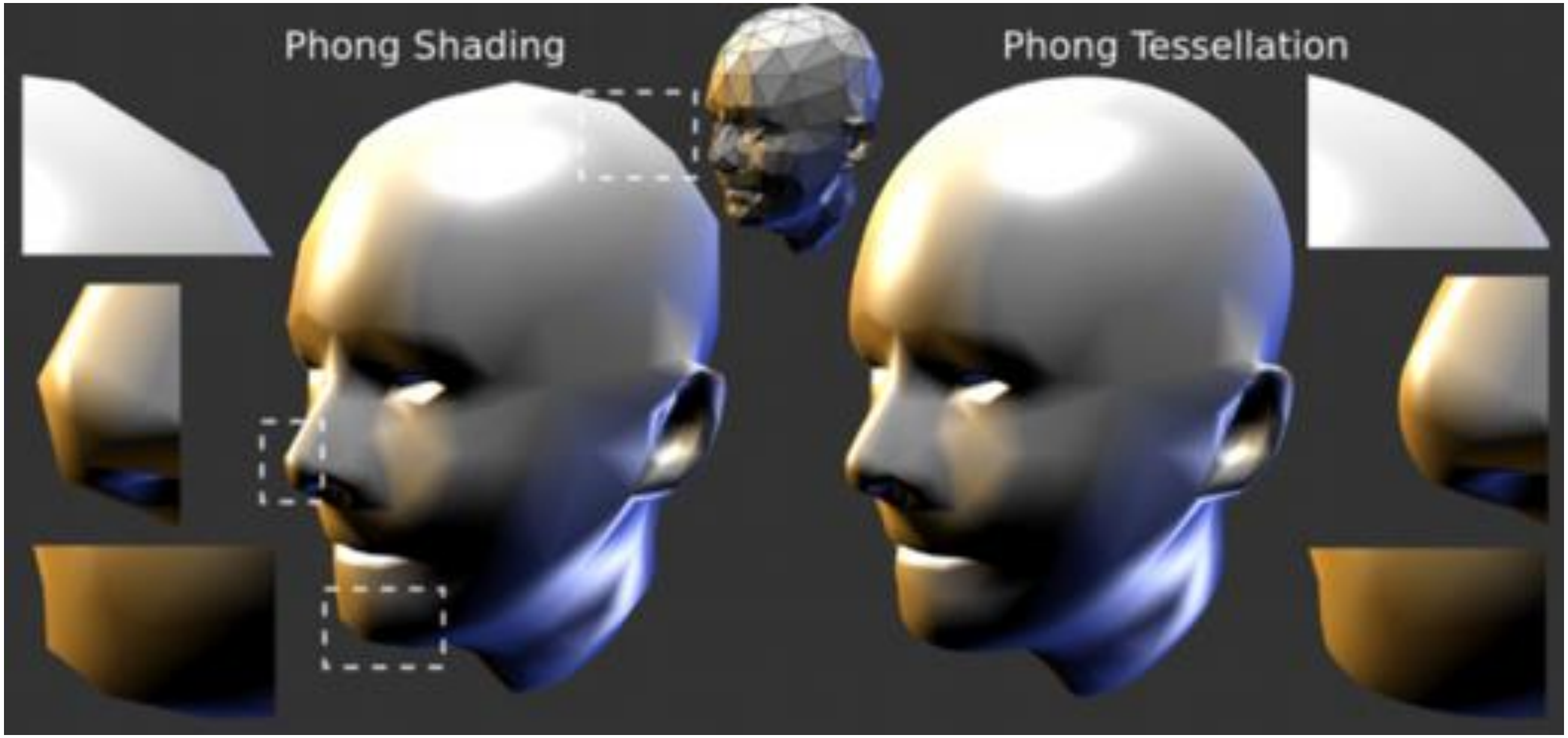


$$\mathbf{E}_1 = \frac{1}{2}(\mathbf{d}_1 + \mathbf{d}_2) + \omega(\mathbf{d}_3 + \mathbf{d}_4) - \frac{\omega}{2}(\mathbf{d}_5 + \mathbf{d}_6 + \mathbf{d}_7 + \mathbf{d}_8)$$

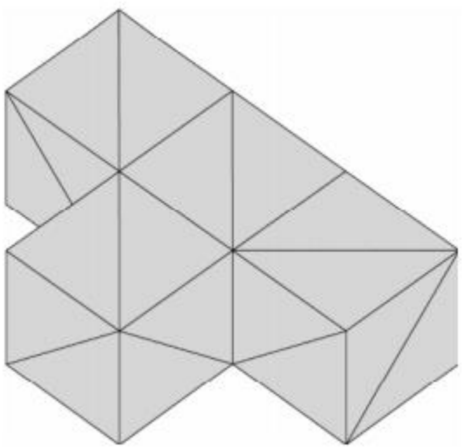
$$\mathbf{d}_i' = \mathbf{d}_i$$

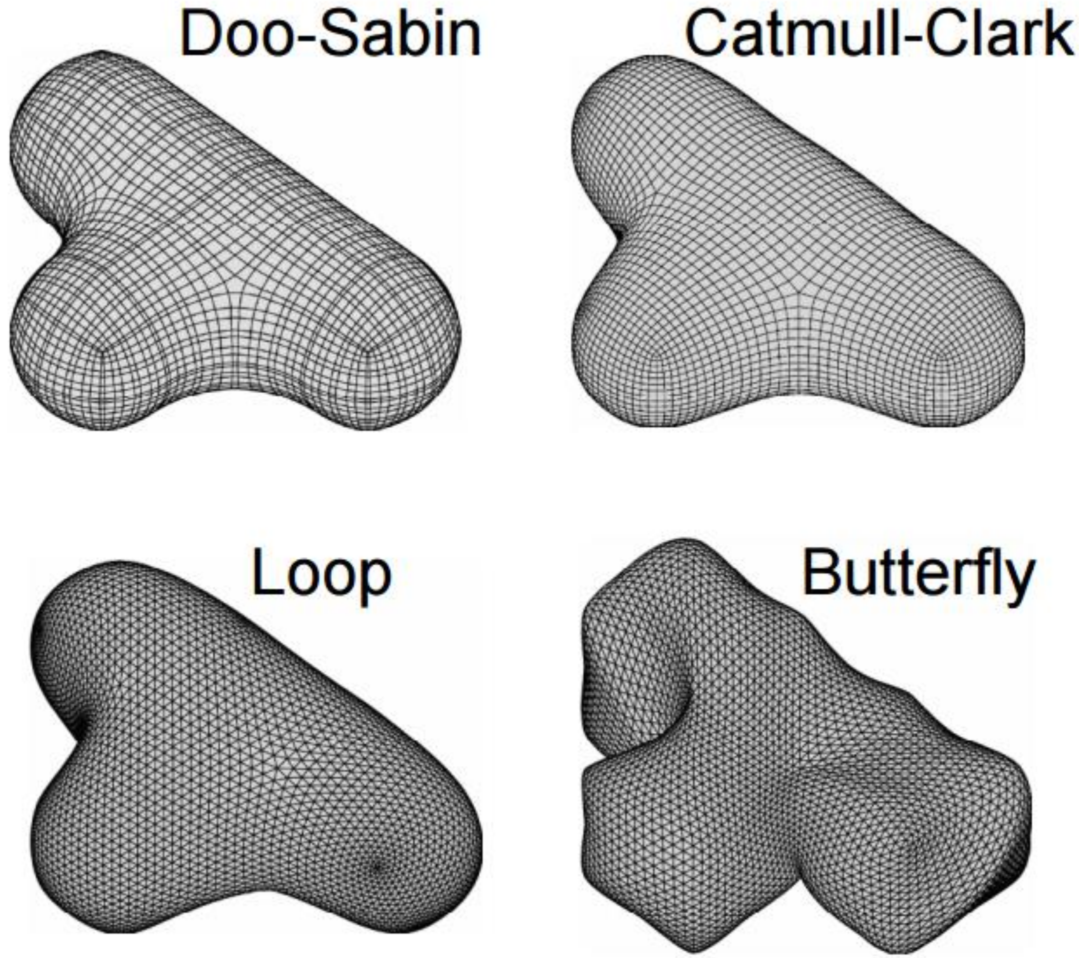


Interpolating Subdivision Surfaces



Comparison

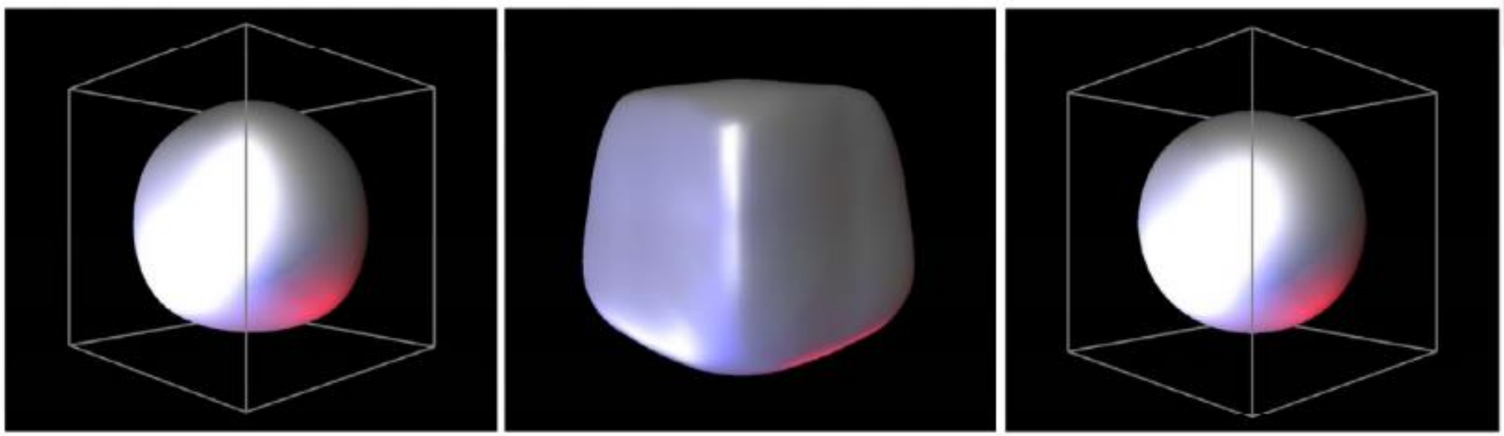




Comparison

Subdividing a cube

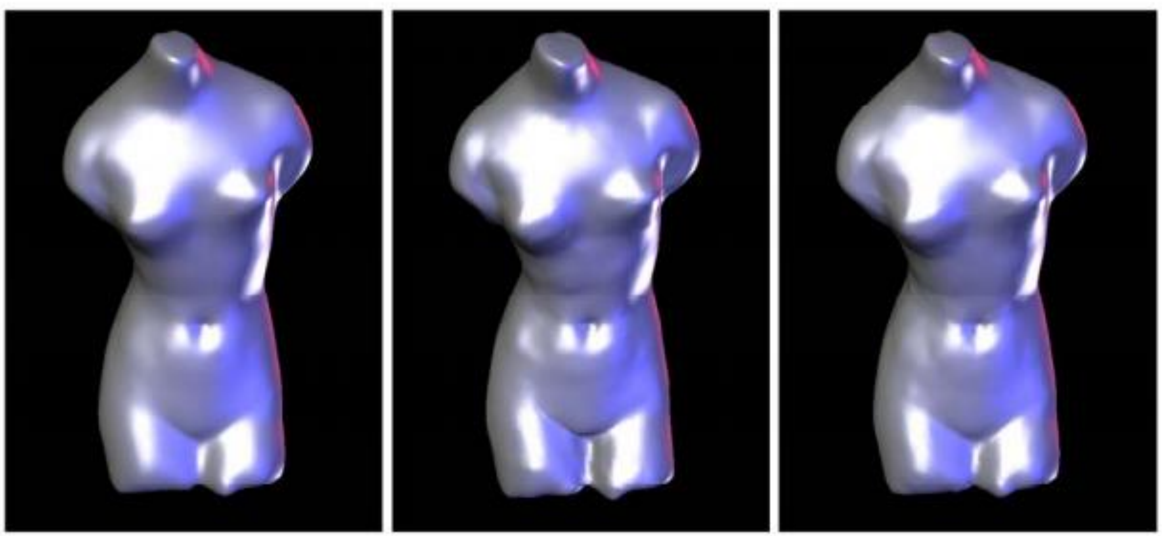
- Loop result is assymetric, because cube was triangulated first
- Both Loop and Catmull-Clark are better then Butterfly (C^2 vs. C^1)
- Interpolation vs. smoothness



Loop *Butterfly* *Catmull-Clark*

Comparison

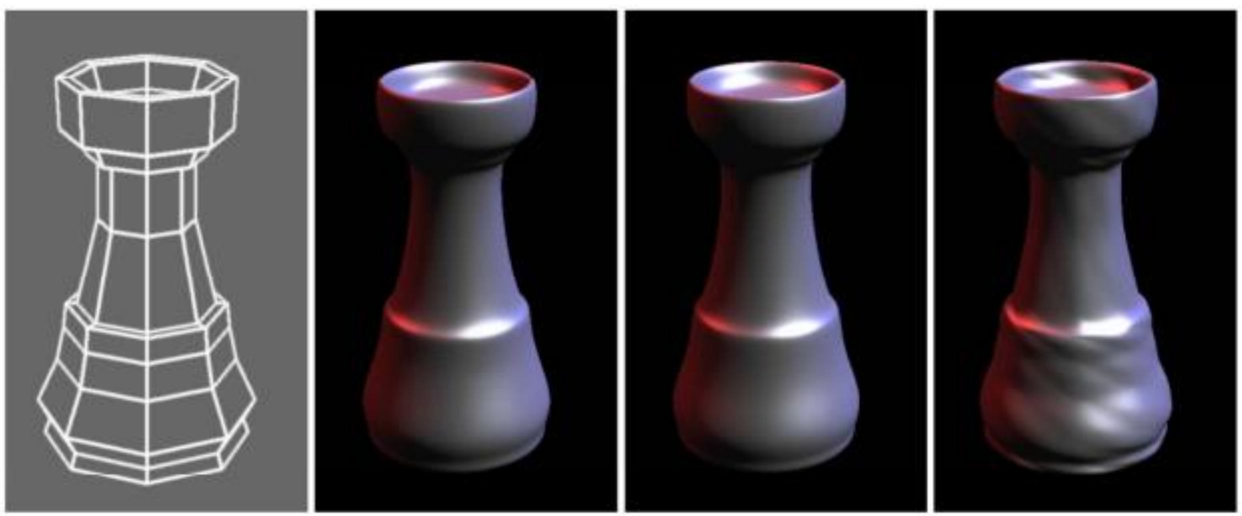
- Spot the difference?
- For smooth meshes with uniform triangle size, different schemes provide very similar results
- Beware of interpolating schemes for control polygons with sharp features



Loop *Butterfly* *Catmull-Clark*

So Who Wins?

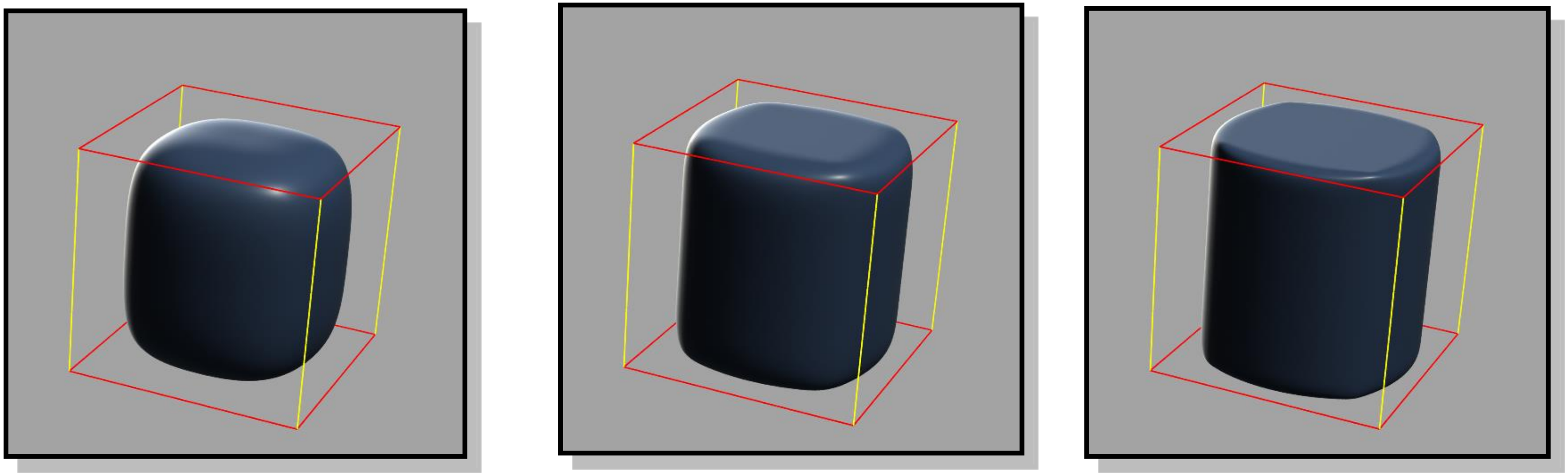
- Loop and Catmull-Clark best when interpolation is not required
- Loop best for triangular meshes
- Catmull-Clark best for quad meshes
  - Don’t triangulate and then use Catmull-Clark



Initial mesh *Loop* *Catmull-Clark* *Catmull-Clark,after triangulation*

Creases

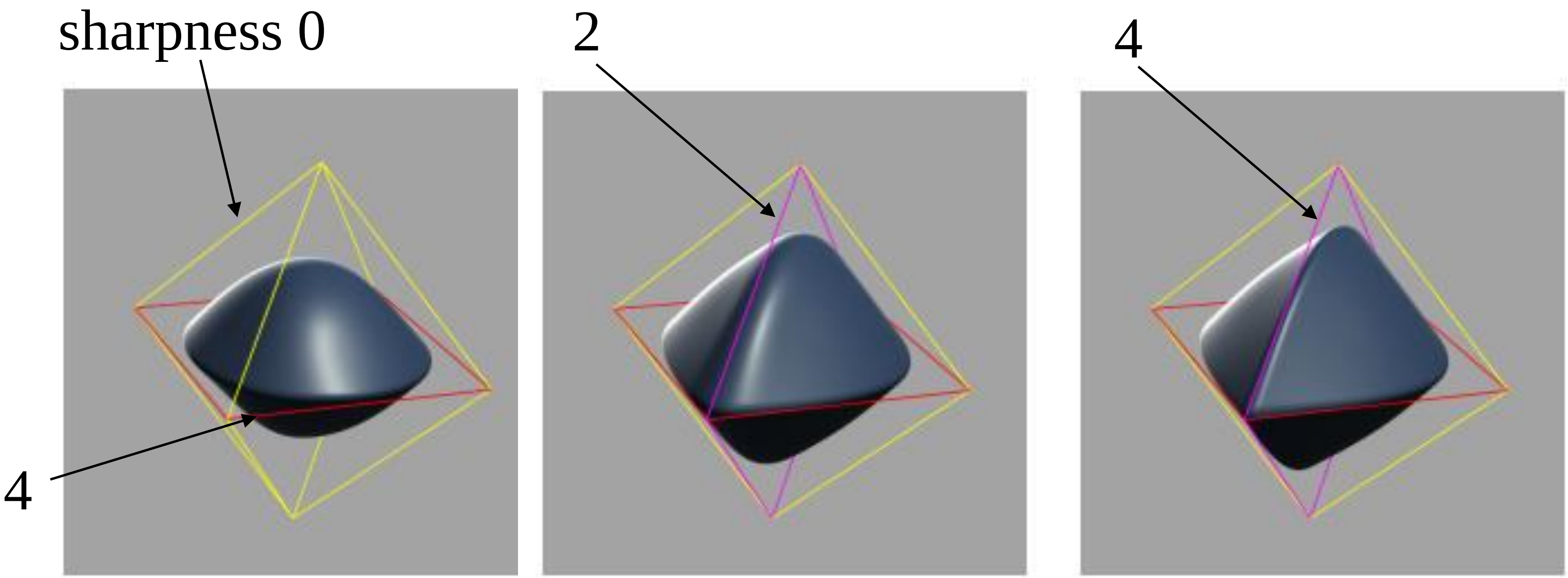
- Extensions exist for most schemes to support *creases*.
- Vertices and edges flagged with **sharpness value** for partial or hybrid subdivision.
- Crease edges of integer sharpness *s* (*s*=0 is smooth)
  - subdivide using **infinitely-sharp rules** *s* times
  - followed by **smooth rules**



Yellow are smooth edges. Red are crease edges: (left to right) increasing sharpness

Catmull-Clark subdivision surfaces in character animation by DeRose et al.

Creases



<u>Edge Sharpness</u>
yellow: 0
red: 4
magenta: 2 (middle), 4 (right)