Lecture 8:

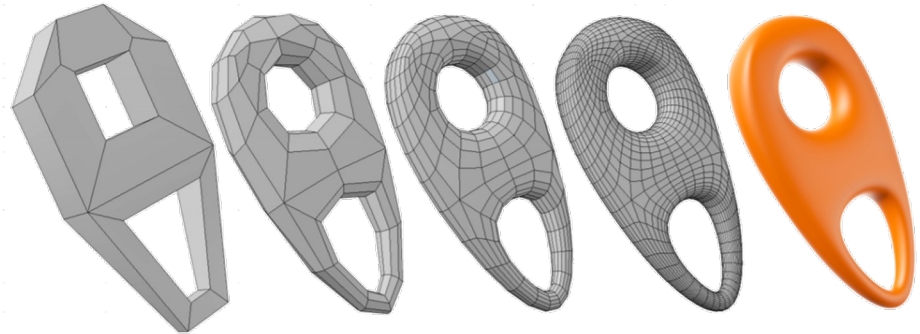
Geometry —— Mesh

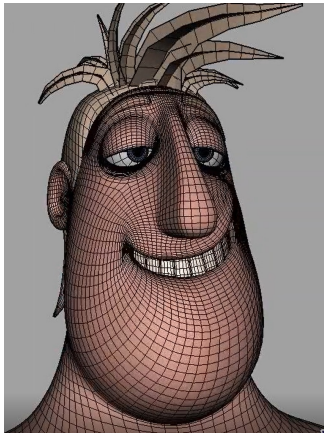
Reference

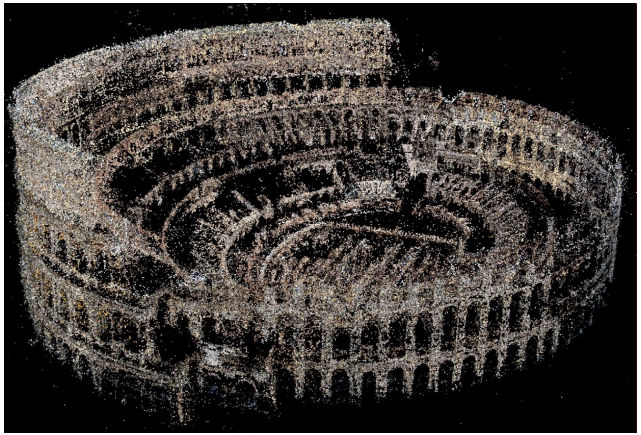
- 闫令琪，UCSB，GAMES101: 现代计算机图形学入门
  - (https://sites.cs.ucsb.edu/~lingqi/teaching/games101.html)
- 胡事民，清华大学，高等计算机图形学

Review: Many Explicit Representations in Graphics

- Triangle/Quad meshes
- Bezier surfaces
- Subdivision surfaces
- NURBS
- Point cloud
- …

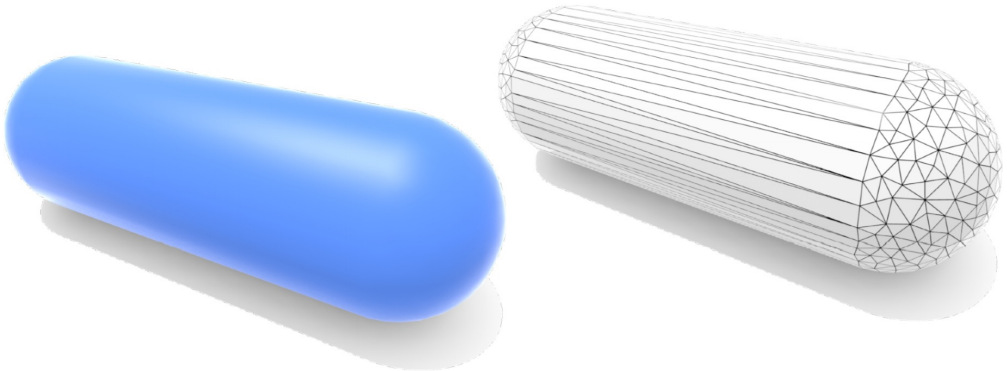






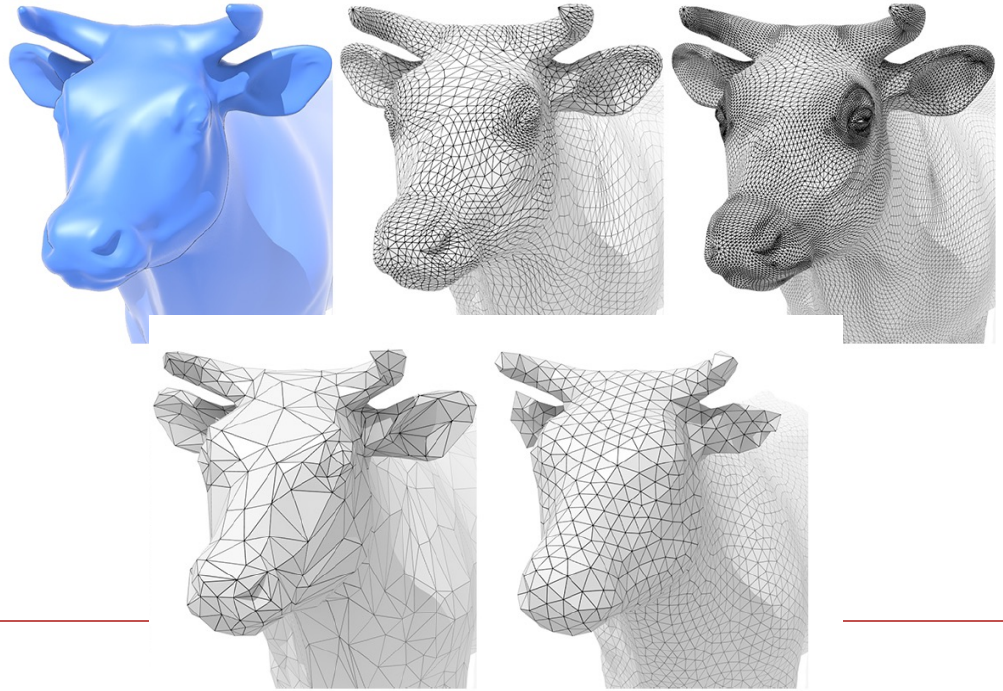
Review: Polygon Mesh (Explicit)

- Store vertices & polygons (often triangles or quads)
- Easier to do processing / simulation, adaptive sampling
- More complicated data structures
- Perhaps most common representation in graphics



Polygon Mesh (Explicit)

- Store vertices & polygons (often triangles or quads)



The Wavefront Object File Format (.obj)

- Commonly used in Graphics research
- Just a text file that specifies vertices, normals, texture coordinates **and their connectivities**

```
# This is a comment

v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 0.999999 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000

vt 0.748573 0.750412
vt 0.749279 0.501284
vt 0.999110 0.501077
vt 0.999455 0.750380
vt 0.250471 0.500702
vt 0.249682 0.749677
vt 0.001085 0.750380
vt 0.001517 0.499994
vt 0.499422 0.500239
vt 0.500149 0.750166
vt 0.748355 0.998230
vt 0.500193 0.998728
vt 0.498993 0.250415
vt 0.748953 0.250920

vn 0.000000 0.000000 -1.000000
vn -1.000000 -0.000000 -0.000000
vn -0.000000 -0.000000 1.000000
vn -0.000001 0.000000 1.000000
vn 1.000000 -0.000000 0.000000
vn 1.000000 0.000000 0.000001
vn 0.000000 1.000000 -0.000000
vn -0.000000 -1.000000 0.000000

f 5/1/1 1/2/1 4/3/1
f 5/1/1 4/3/1 8/4/1
f 3/5/2 7/6/2 8/7/2
f 3/5/2 8/7/2 4/8/2
f 2/9/3 6/10/3 3/5/3
f 6/10/4 7/6/4 3/5/4
f 1/2/5 5/1/5 2/9/5
f 5/1/6 6/10/6 2/9/6
f 5/1/7 8/11/7 6/10/7
f 8/11/7 7/12/7 6/10/7
f 1/2/8 2/9/8 3/13/8
f 1/2/8 3/13/8 4/14/8
```

Topic

- Basic Concepts
- Mesh Operations
  - Local Mesh Operations
    - Half-edge
    - Edge flip, Edge split, Edge collapse
  - Global Mesh Operations: Geometry Processing
    - Mesh subdivision, Mesh simplification

Topic

- Basic Concepts
- Mesh Operations
  - Local Mesh Operations
    - Half-edge
    - Edge flip, Edge split, Edge collapse
  - Global Mesh Operations: Geometry Processing
    - Mesh subdivision, Mesh simplification

网格相关基本概念

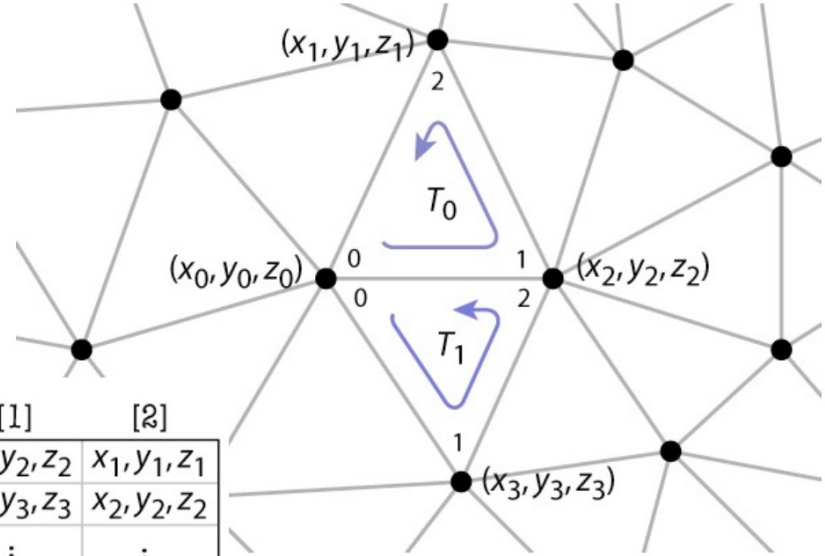
- 网格的描述
  - 一系列的面片$F = (f_1, f_2, \ldots, f_n)$
    - 每一个面片都是三角形
  - 一系列的顶点$V = (v_1, v_2, \ldots, v_n)$
  - F中的每个面片是V中顶点的序列组

e.g.

$f_1: (v_1, v_2, v_3)$, $f_2: (v_4, v_5, v_6)$,

$f_3: (v_7, v_8, v_9)$, ...

List of Triangles

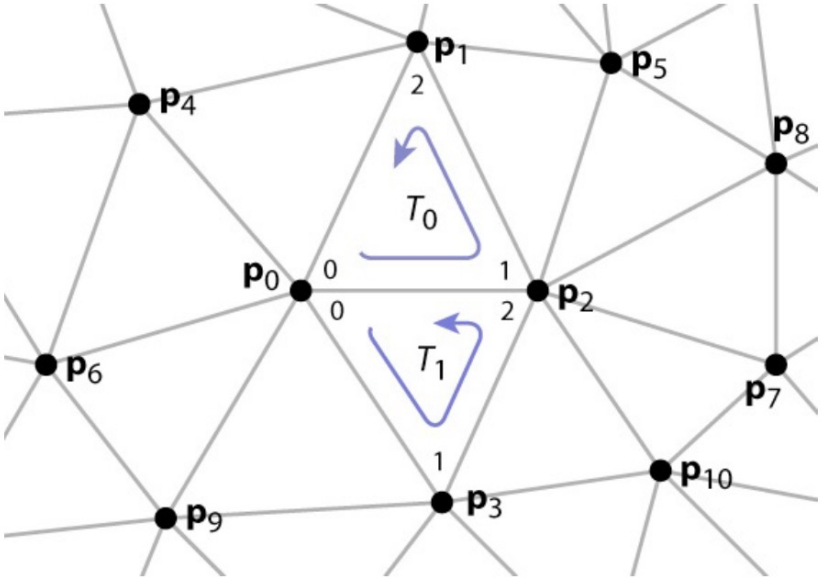


	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
	⋮	⋮	⋮

Lists of Points / Indexed Triangle

verts[0]	x_0, y_0, z_0
verts[1]	x_1, y_1, z_1
	x_2, y_2, z_2
	x_3, y_3, z_3
	⋮

tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
	⋮



网格表示的由来

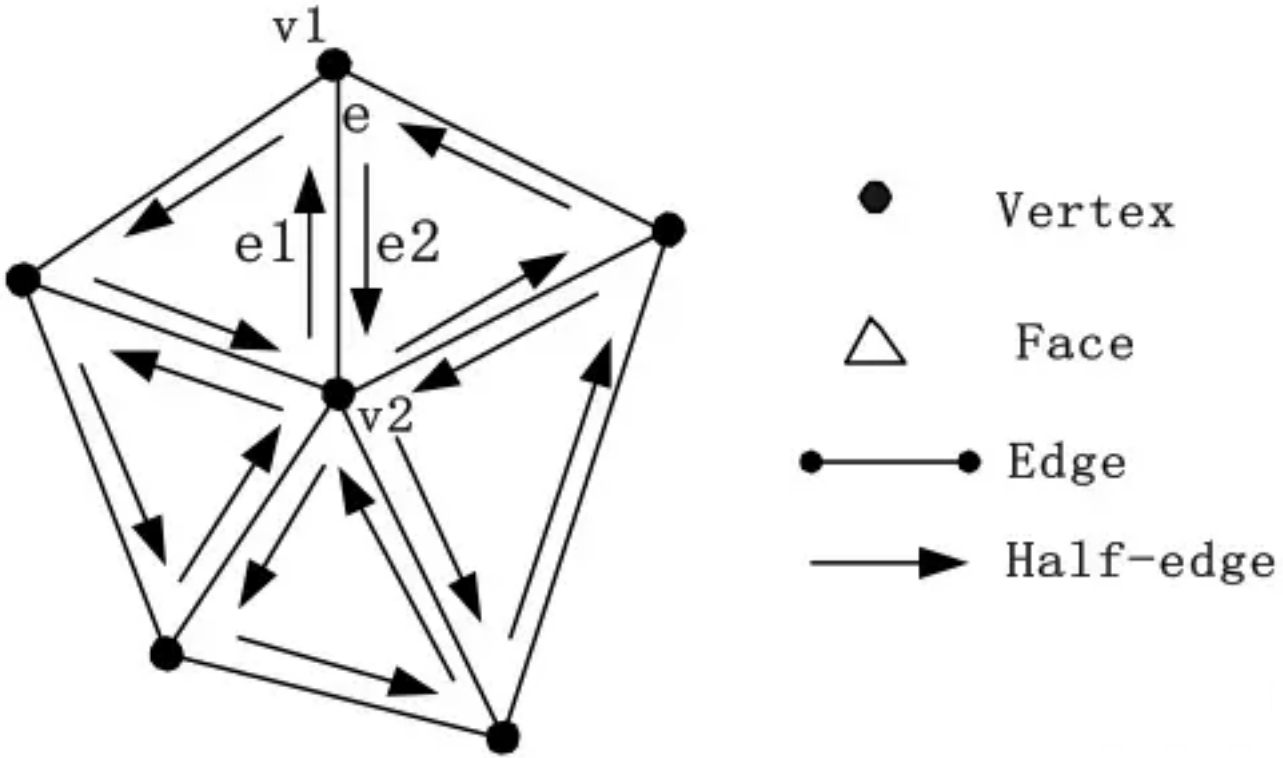
- 计算机产生的三维模型和实际获取的数据的表示模式不同
  - 图形学中需要一个统一的表示方式
  - 视觉精度和处理速度需要在可以接受的范围之内
  - 由于图形硬件的快速发展, 我们已经能够快速地进行光栅化（rasterize）和渲染（ render ）基于三角网格表示的模型

Topic

- Basic Concepts
- Mesh Operations
  - Local Mesh Operations
    - Half-edge
    - Edge flip, Edge split, Edge collapse
  - Global Mesh Operations: Geometry Processing
    - Mesh subdivision, Mesh simplification

Part 1 Local Mesh Operations

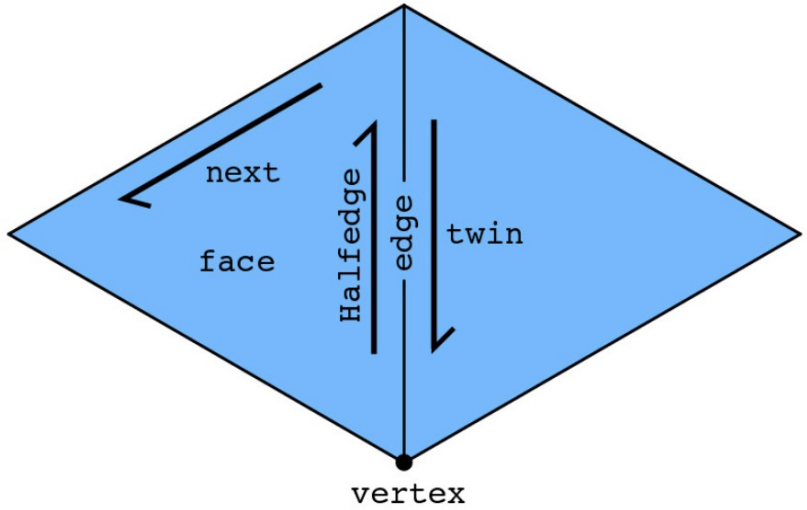
Half-edge 半边



Half-edge Data Structure

```
struct Halfedge {
   Halfedge *twin,
   Halfedge *next;
   Vertex *vertex;
   Edge *edge;
   Face *face;
}
struct Vertex {
   Point pt;
   Halfedge *halfedge;
}
struct Edge {
   Halfedge *halfedge;
}
struct Face {
   Halfedge *halfedge;
}
```

Key idea: two half-edges act as "glue" between mesh elements

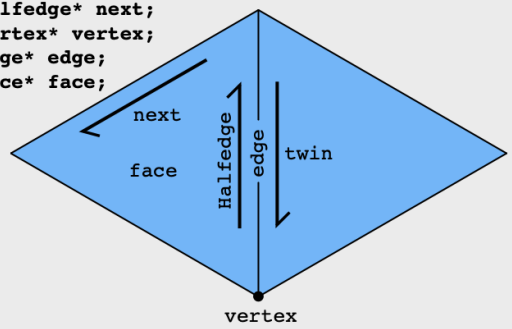


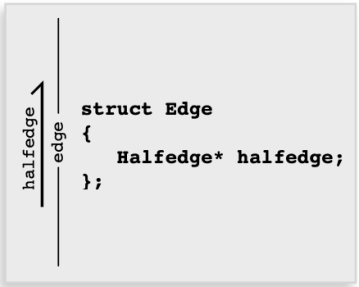
Each vertex, edge and face points to one of its half edges

Half-edge Data Structure (linked-list-like)

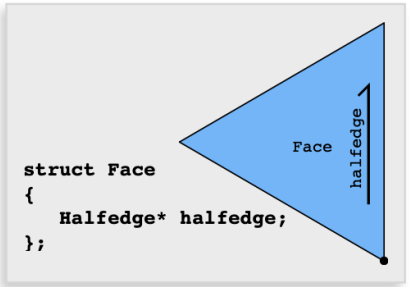
- **Store *some* information about neighbors**
- **Don't need an exhaustive list; just a few key pointers**
- **Key idea: two *halfedges* act as "glue" between mesh elements:**

```
struct Halfedge
{
   Halfedge* twin;
   Halfedge* next;
   Vertex* vertex;
   Edge* edge;
   Face* face;
};
```

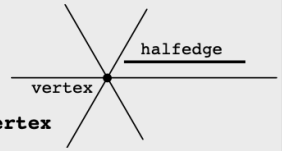





```
struct Edge
{
   Halfedge* halfedge;
};
```




```
struct Face
{
   Halfedge* halfedge;
};
```



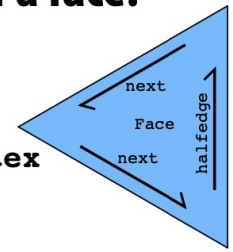

```
struct Vertex
{
   Halfedge* halfedge;
};
```

- **Each vertex, edge face points to just *one* of its halfedges.**

Half-edge makes mesh traversal easy

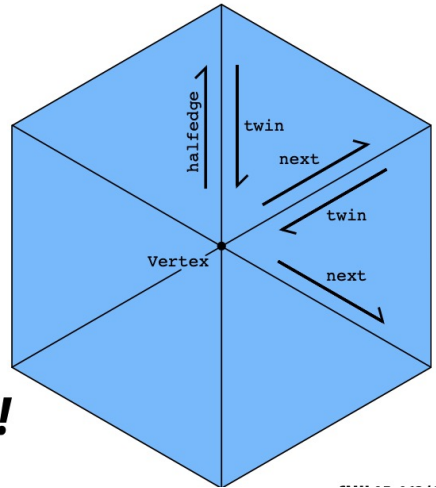
- Use "twin" and "next" pointers to move around mesh
- Use "vertex", "edge", and "face" pointers to grab element
- Example: visit all vertices of a face:

```
Halfedge* h = f->halfedge;
do {
   h = h->next;
   // do something w/ h->vertex
}
while( h != f->halfedge );
```



- Example: visit all neighbors of a vertex:

```
Halfedge* h = v->halfedge;
do {
   h = h->twin->next;
}
while( h != v->halfedge );
```

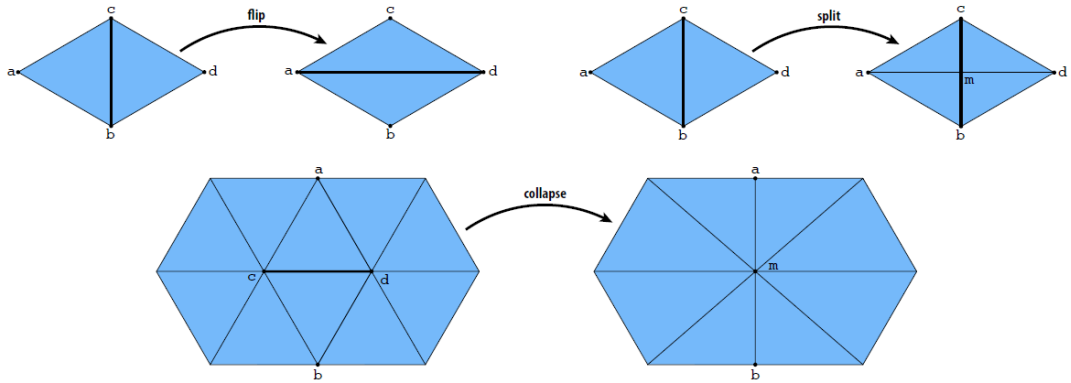


- Note: only makes sense if mesh is *manifold!*

Local Mesh Editing

Basic operations for linked list: insert, delete

Basic ops for half-edge mesh: flip, split, collapse edges

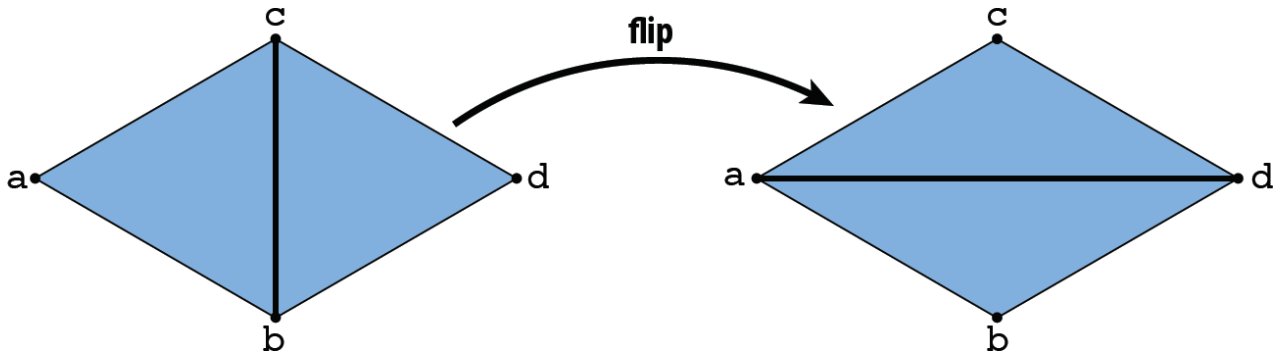


Allocate / delete elements; reassign pointers

(Care needed to preserve mesh manifold property)

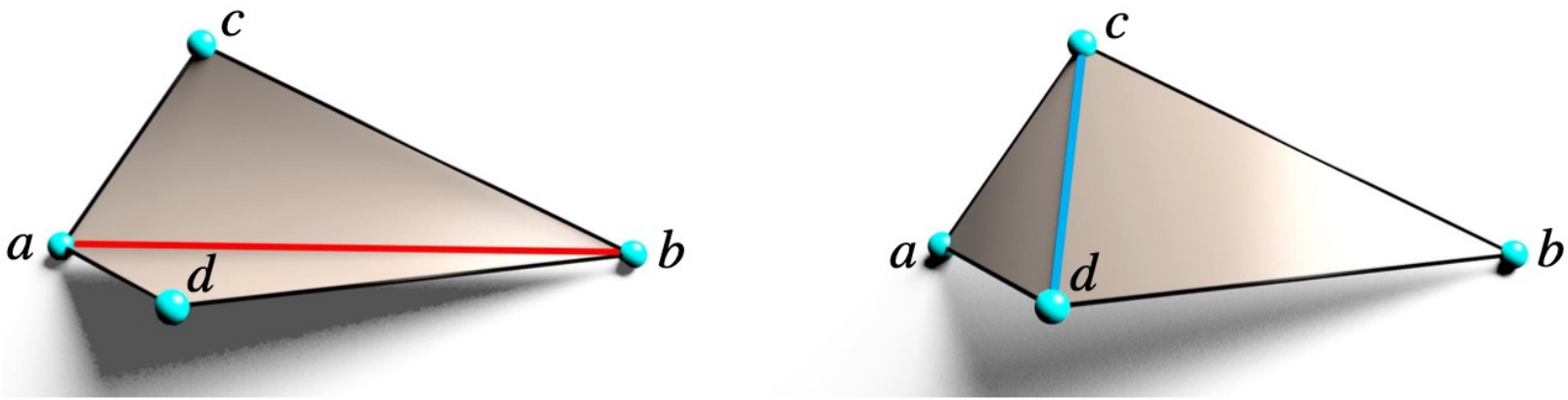
Edge Flip

- Triangles (a,b,c), (b,d,c) become (a,d,c), (a,b,d):



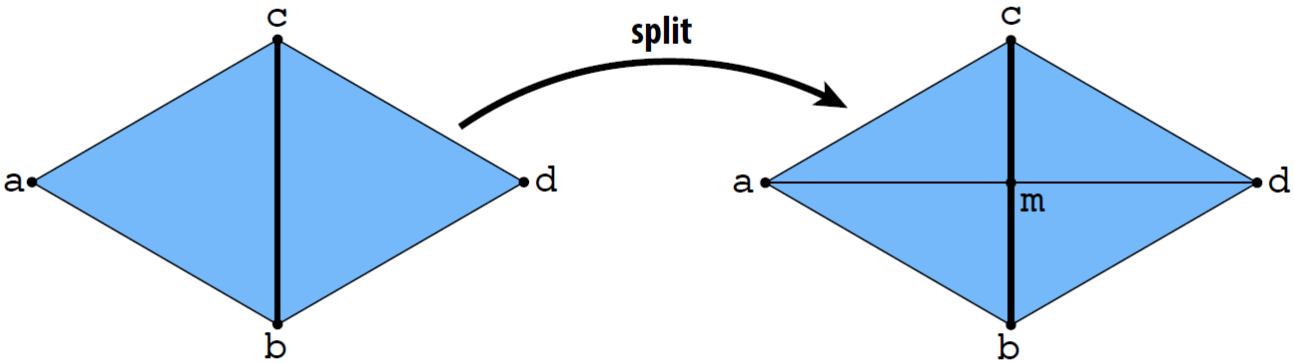
- Long list of pointer reassignments
- However, no elements created/destroyed.

Edge Flip



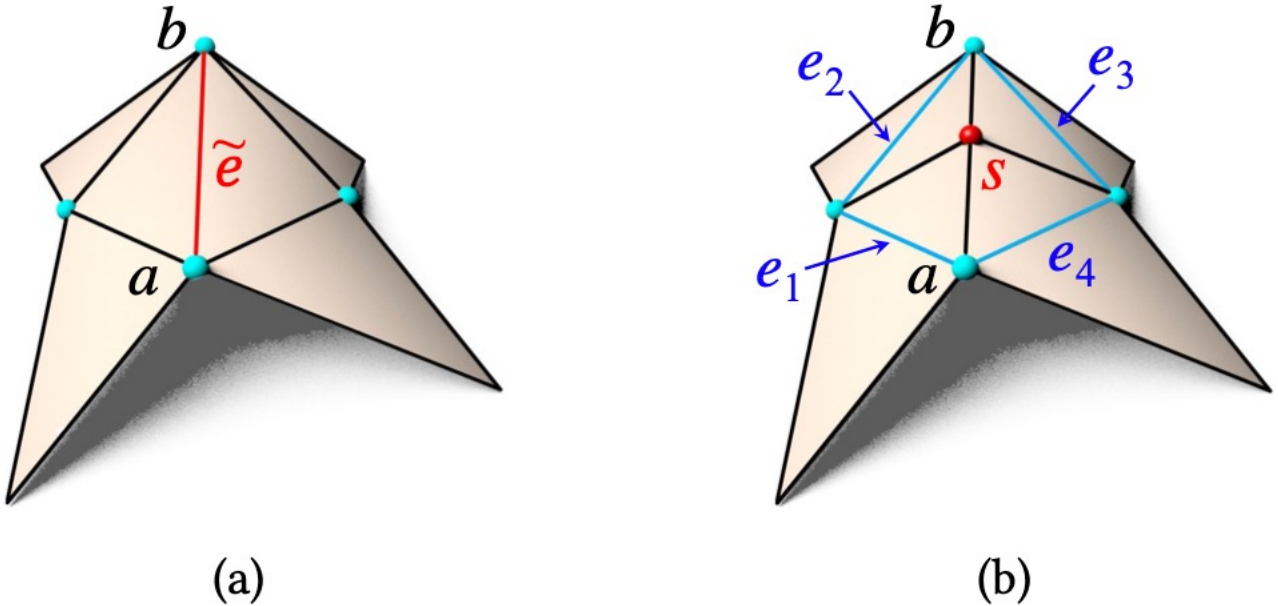
Edge Split

- Insert midpoint m of edge (c,b), connect to get four triangles:



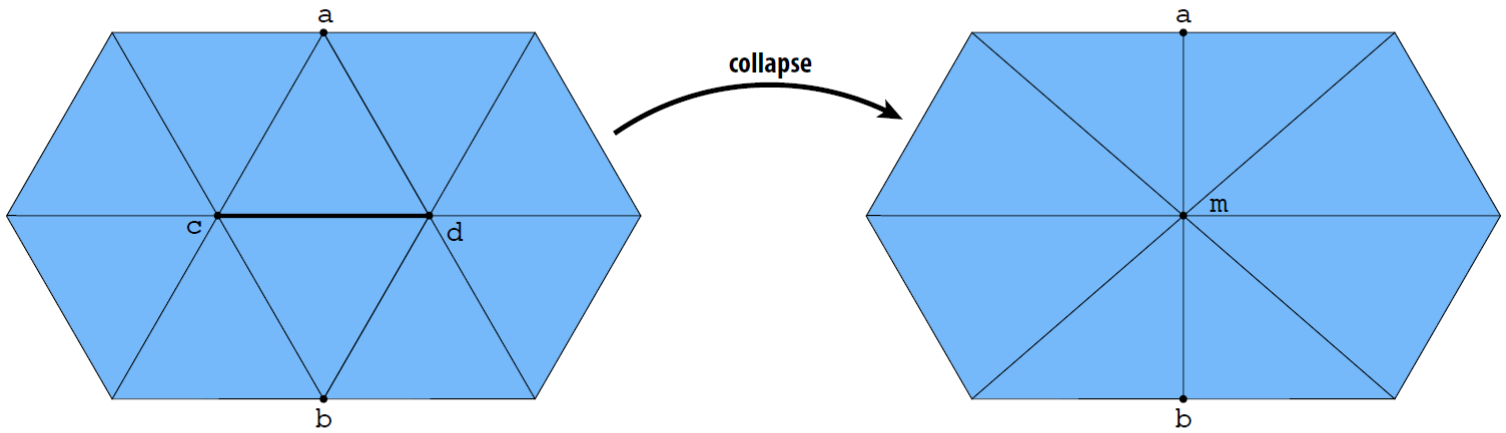
- This time have to add elements
- Again, many pointer reassignments

Edge Split



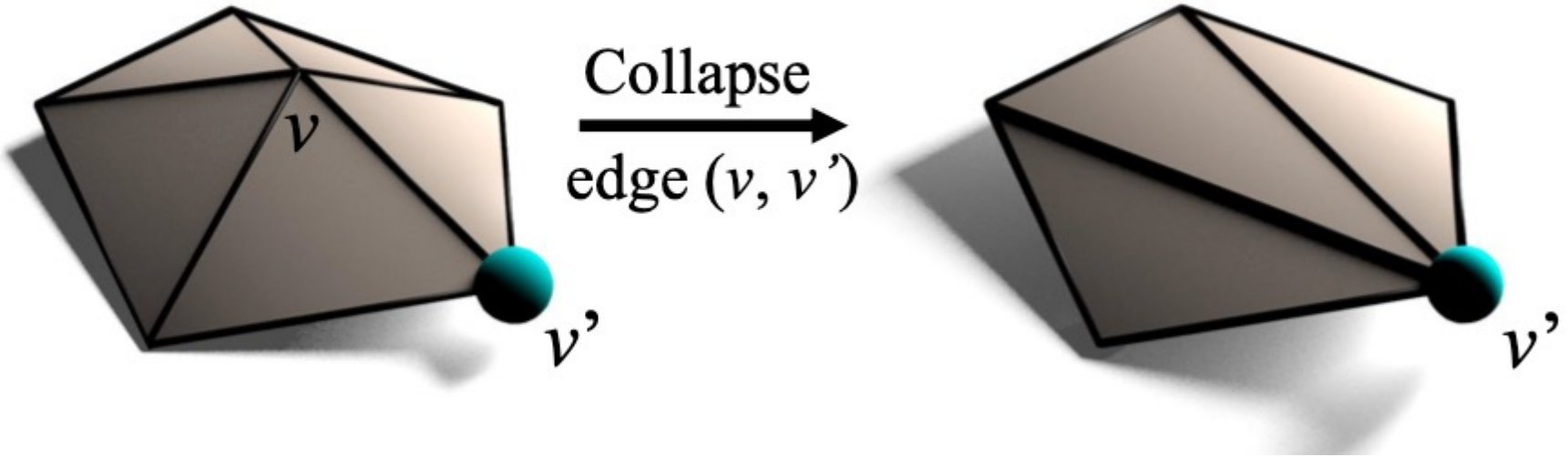
Edge Collapse

- Replace edge (c,d) with a single vertex m:



- This time have to delete elements
- Again, many pointer reassignments

Edge Collapse

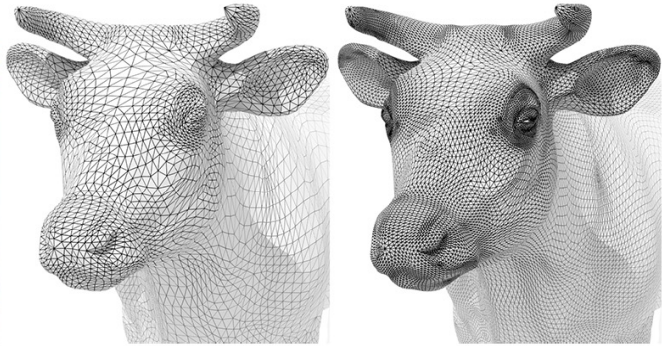


Part 2 Global Mesh Operations

Mesh Operations

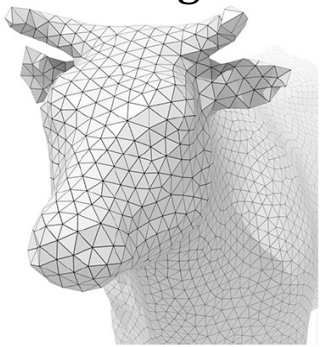
Original mesh

Mesh subdivision

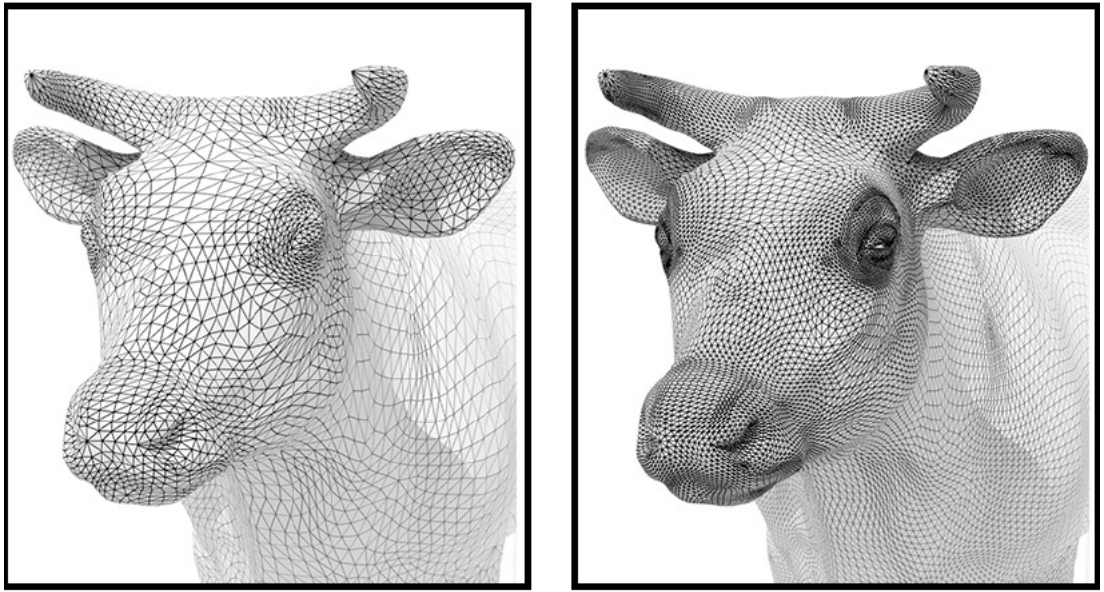


Mesh Simplification

Mesh Regularization

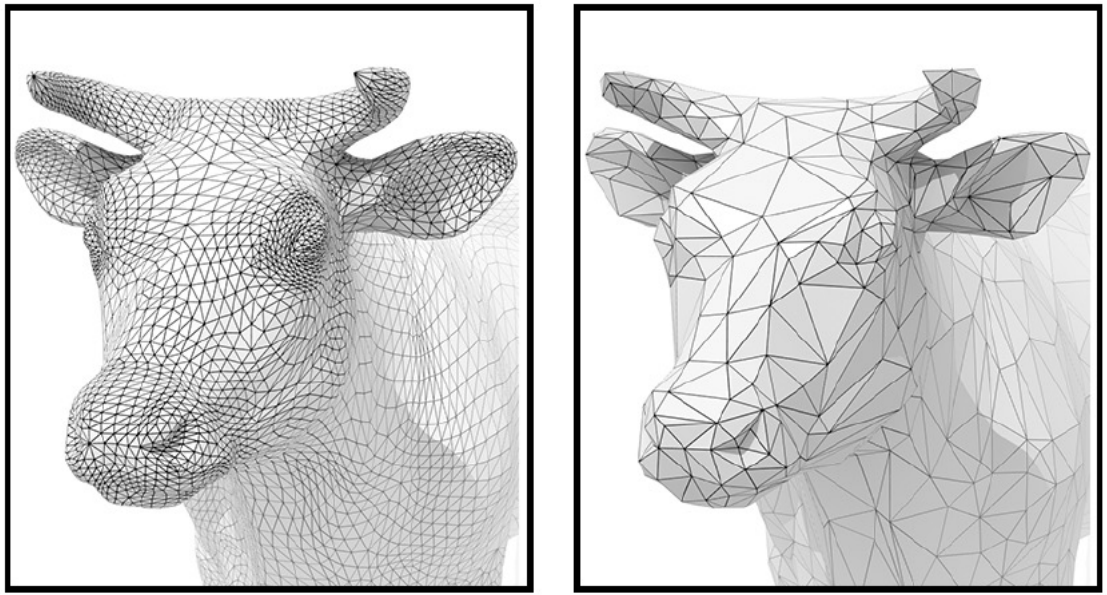


Mesh Subdivision (Upsampling)



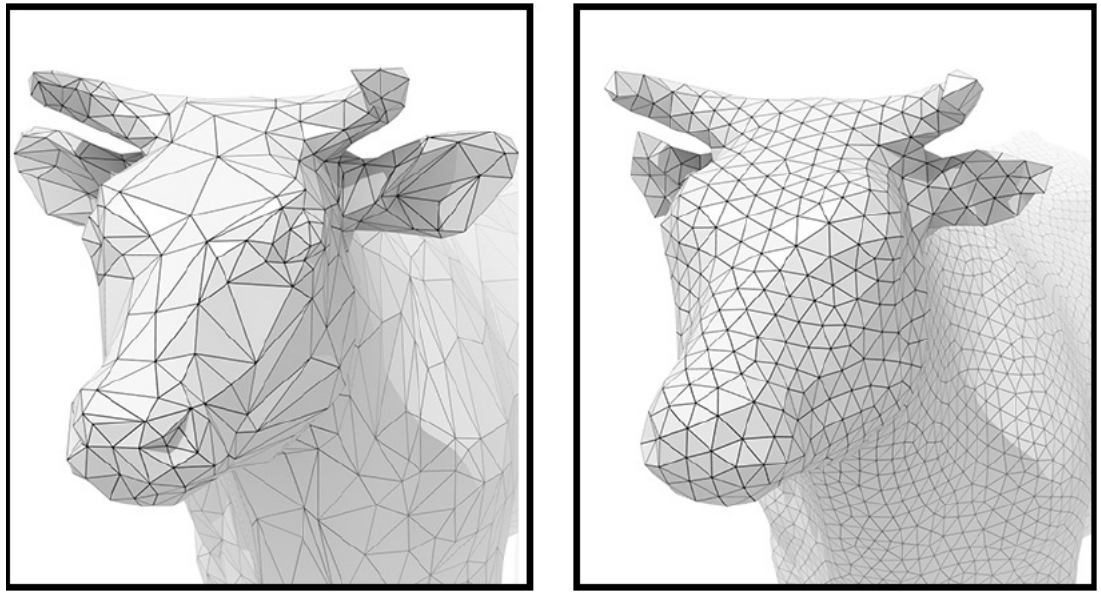
Increase resolution via interpolation

Mesh Simplification (Downsampling)



Decrease resolution; try to preserve shape/appearance

Mesh Regularization (same #triangles)

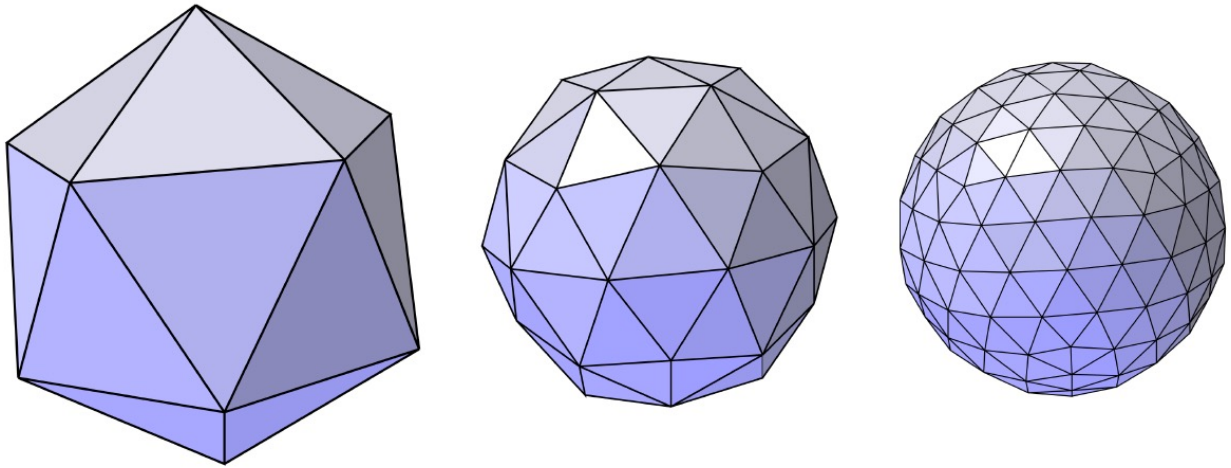


Modify sample distribution to **improve quality**

Mesh Subdivision

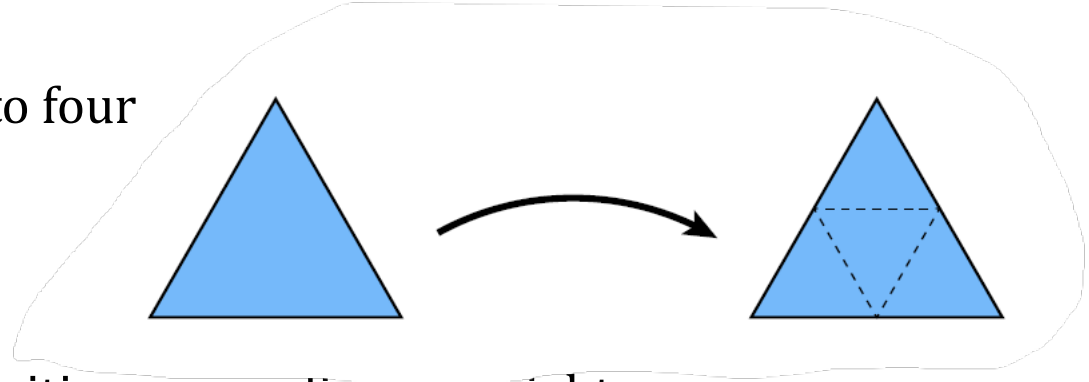
Loop Subdivision

- Common subdivision rule for triangle meshes
- First, create more triangles (vertices)
- Second, tune their positions

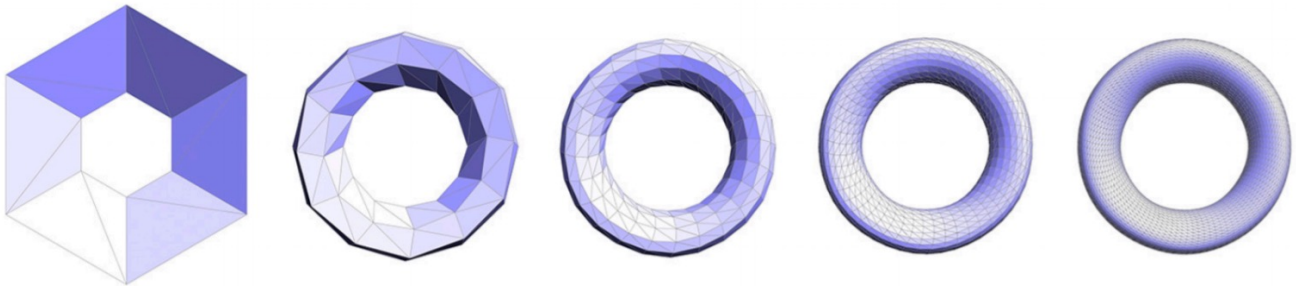


Loop Subdivision

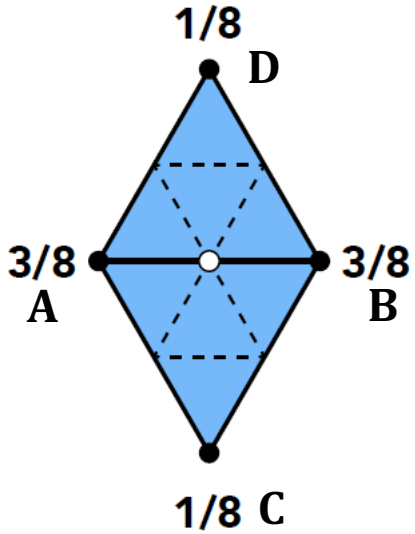
- Split each triangle into four



- Assign new vertex positions according to weights
  - New / old vertices updated differently



Loop Subdivision — Update (for new vertices)

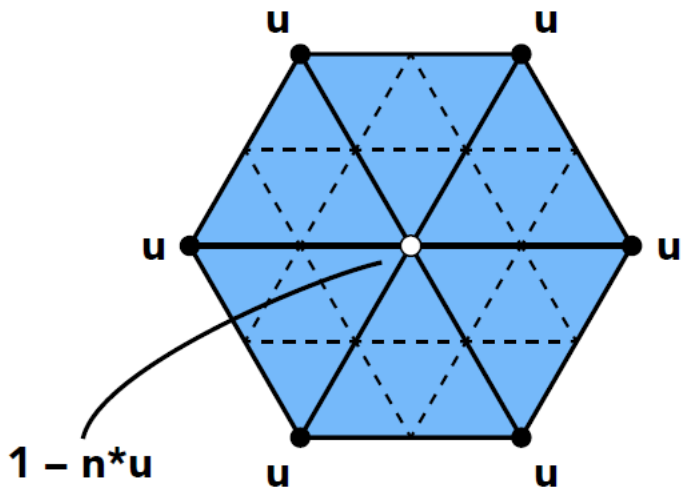


New vertices

Update to:

$3/8*(A+B)+1/8*(C+D)$

Loop Subdivision — Update (for old vertices)



Old vertices

For old vertices (e.g. degree 6 vertices here):

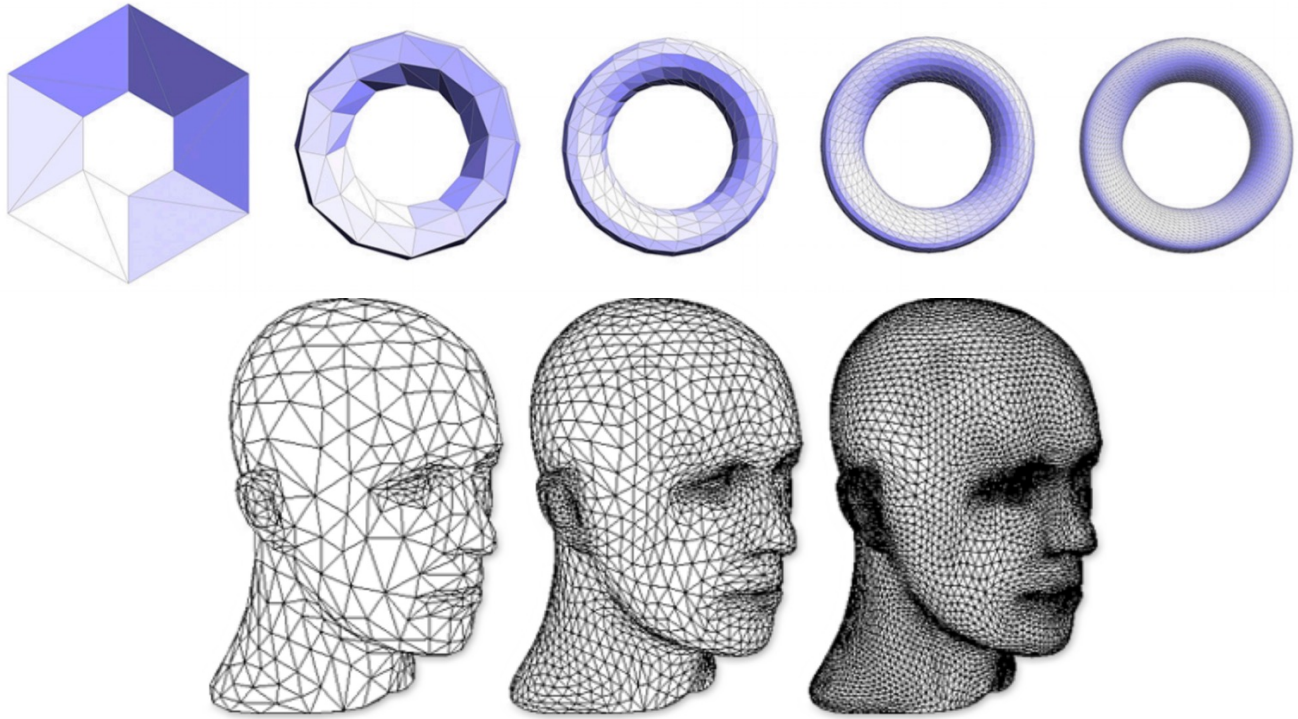
Update to:

(1 - n*u) * original_position + u * neighbor_position_sum

n: vertex degree

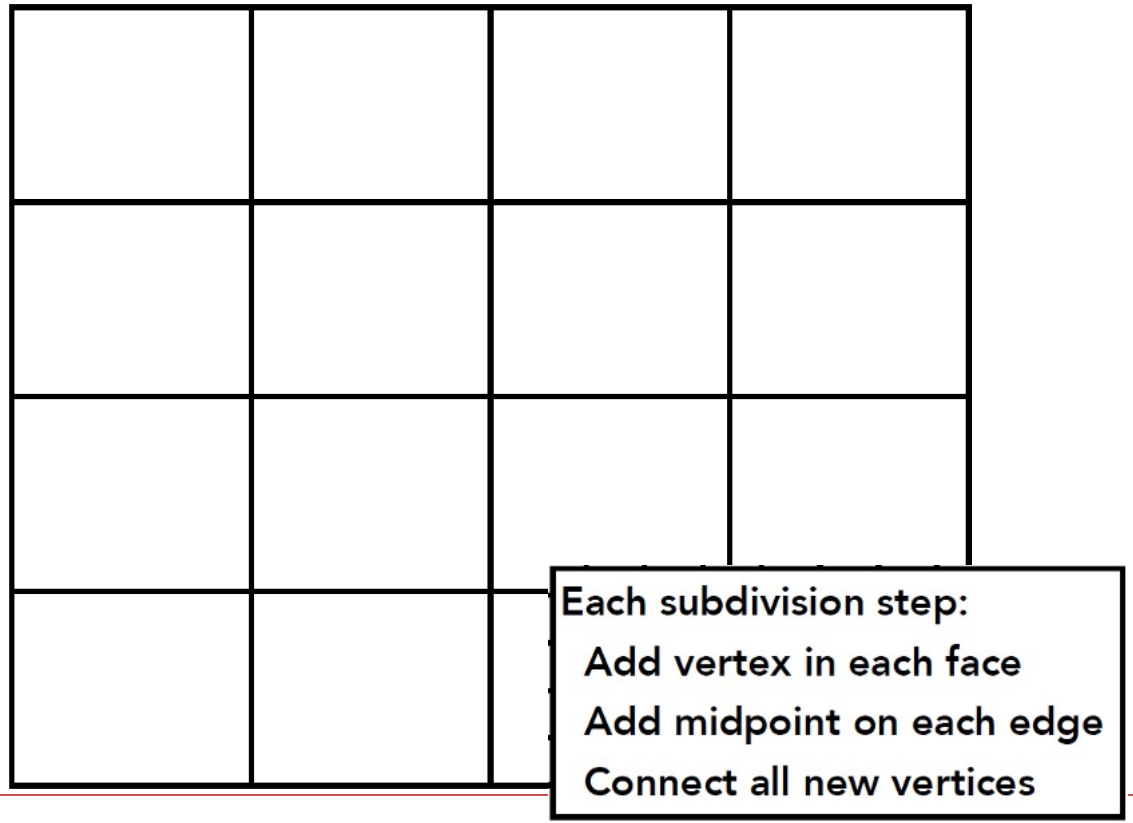
u: 3/16 if n=3, 3/(8n) otherwise

Loop Subdivision Results

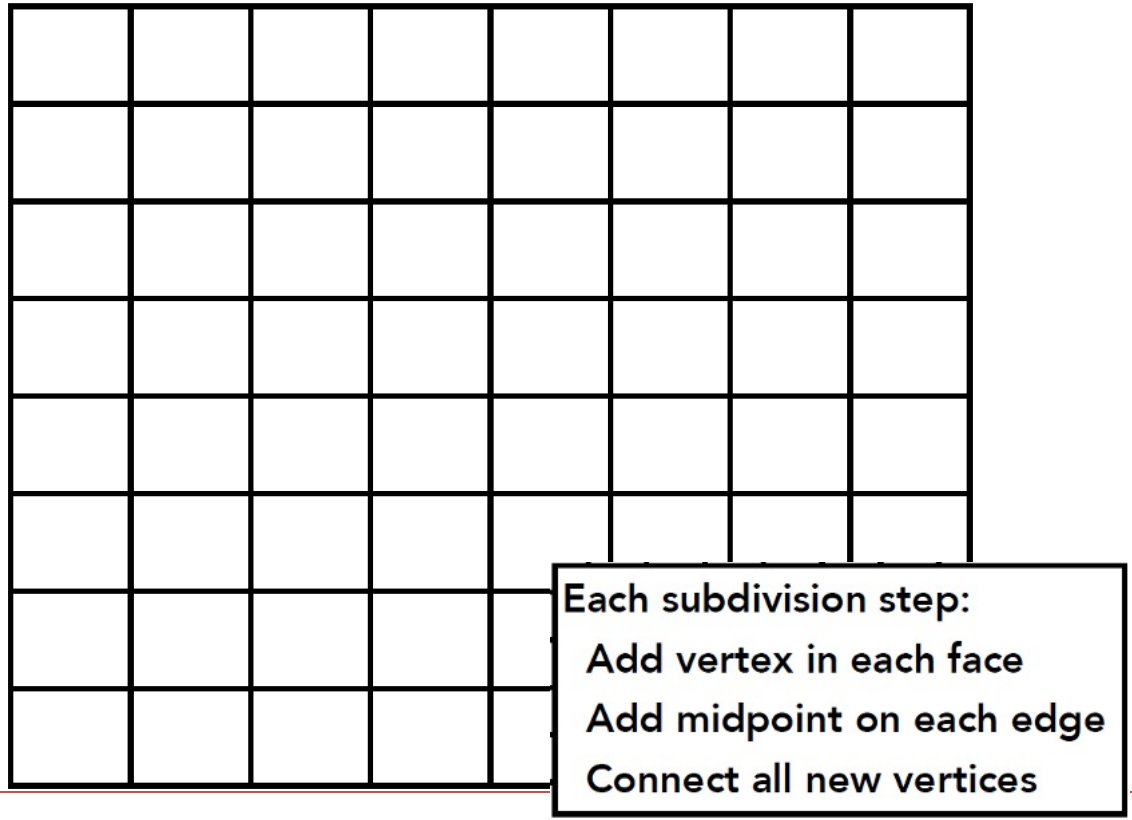


Catmull-Clark Subdivision

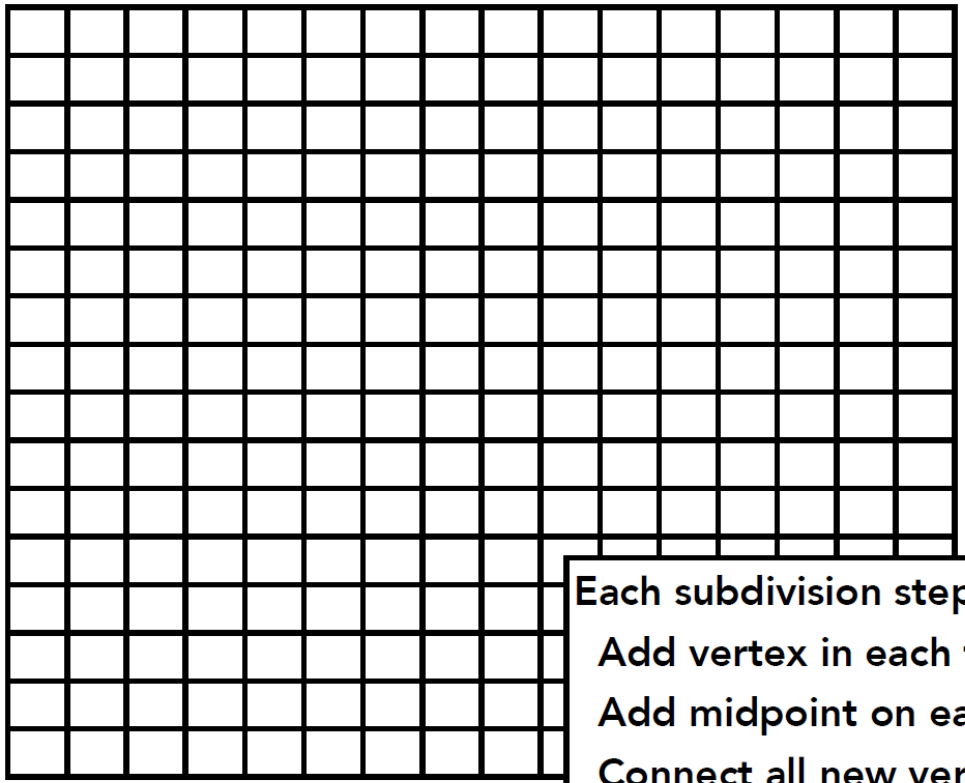
Catmull-Clark Vertex Update Rules (Regular Quad Mesh)



Catmull-Clark Vertex Update Rules (Regular Quad Mesh)



Catmull-Clark Vertex Update Rules (Regular Quad Mesh)

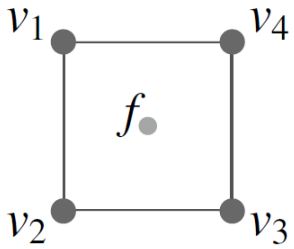


Each subdivision step:

- Add vertex in each face
- Add midpoint on each edge
- Connect all new vertices

Catmull-Clark Vertex Update Rules (Quad Mesh)

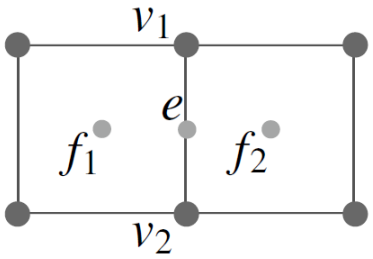
Face point

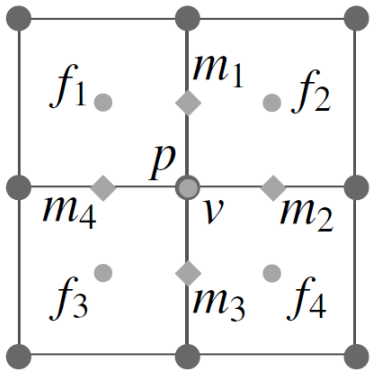


$$f = \frac{v_1 + v_2 + v_3 + v_4}{4}$$

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

Edge point





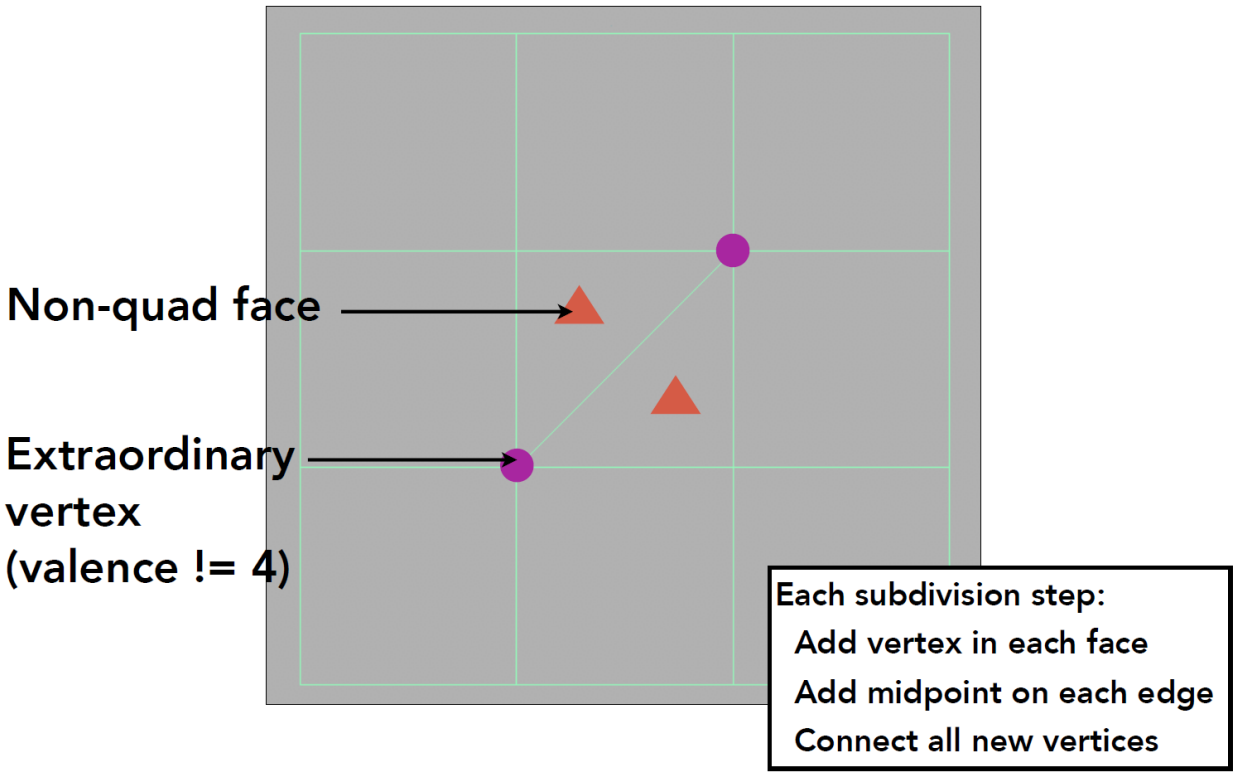
Vertex point

$$v = \frac{f_1 + f_2 + f_3 + f_4 + 2(m_1 + m_2 + m_3 + m_4) + 4p}{16}$$

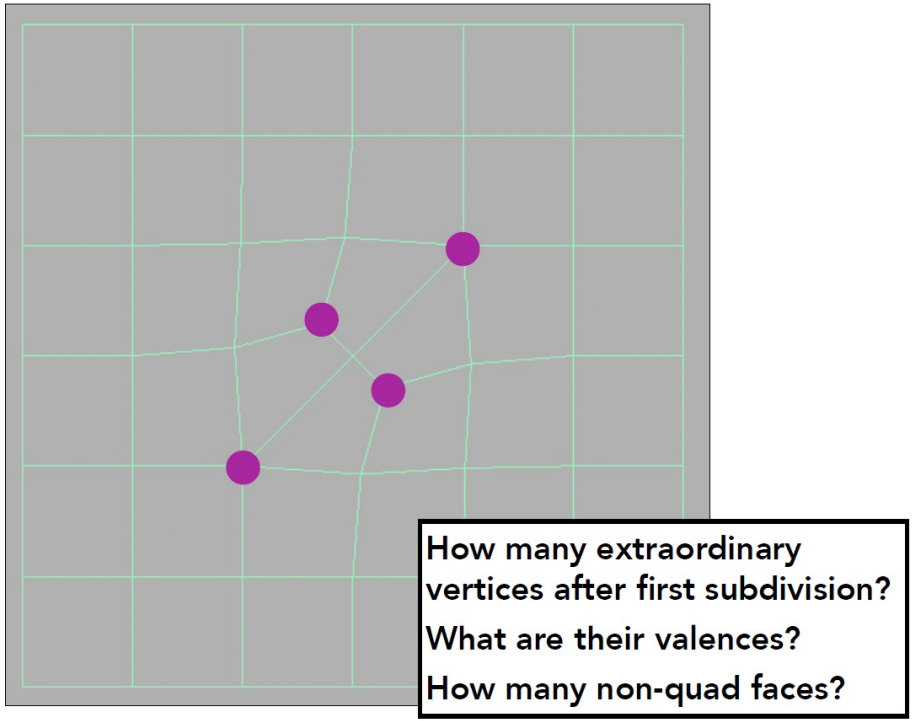
m midpoint of edge, not "edge point"

p old "vertex point"

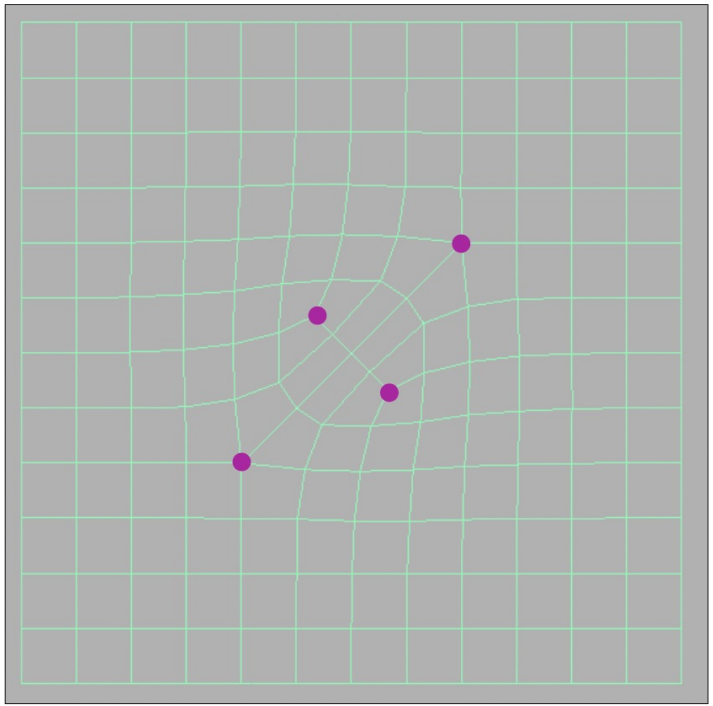
Catmull-Clark Subdivision (General Mesh)



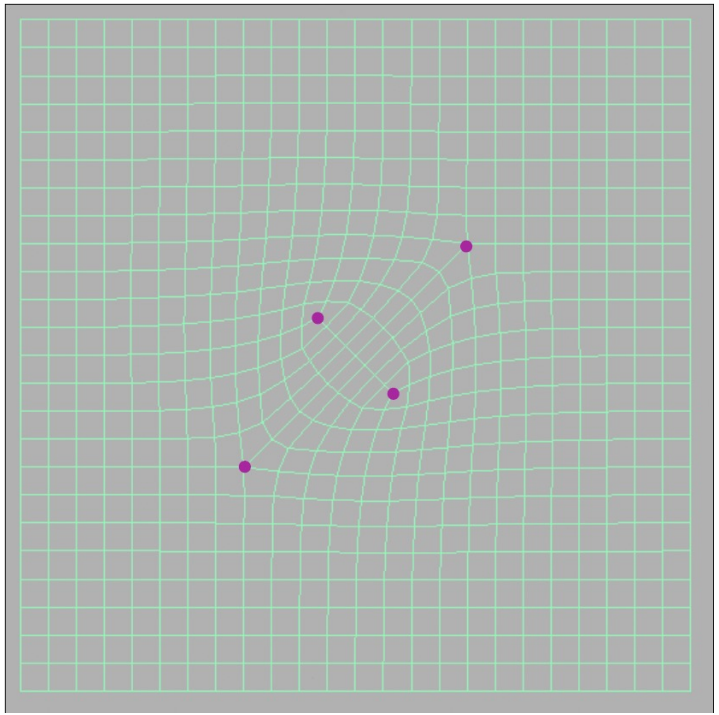
Catmull-Clark Subdivision (General Mesh)



Catmull-Clark Subdivision (General Mesh)



Catmull-Clark Subdivision (General Mesh)



Catmull-Clark Vertex Update Rules (General Mesh)

f = average of surrounding vertices

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

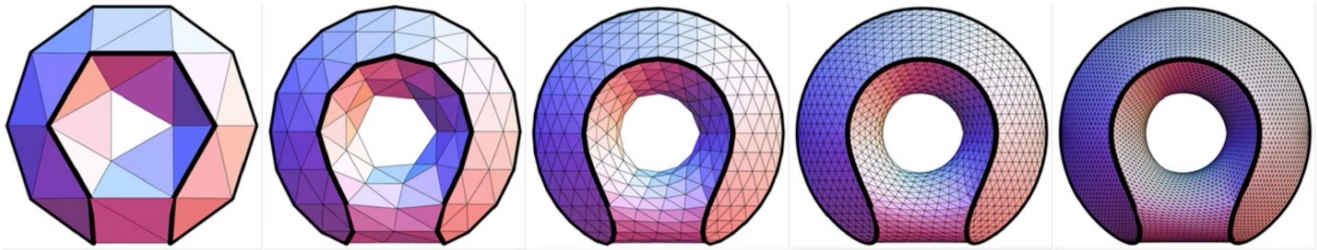
$$v = \frac{\bar{f}}{n} + \frac{2\bar{m}}{n} + \frac{p(n-3)}{n}$$

These rules reduce to earlier quad rules for ordinary vertices / faces

$\bar{m}$ = average of adjacent midpoints
$\bar{f}$ = average of adjacent face points
n = valence of vertex
p = old "vertex" point

Convergence: Overall Shape and Creases

Loop with Sharp Creases



Catmull-Clark with Sharp Creases

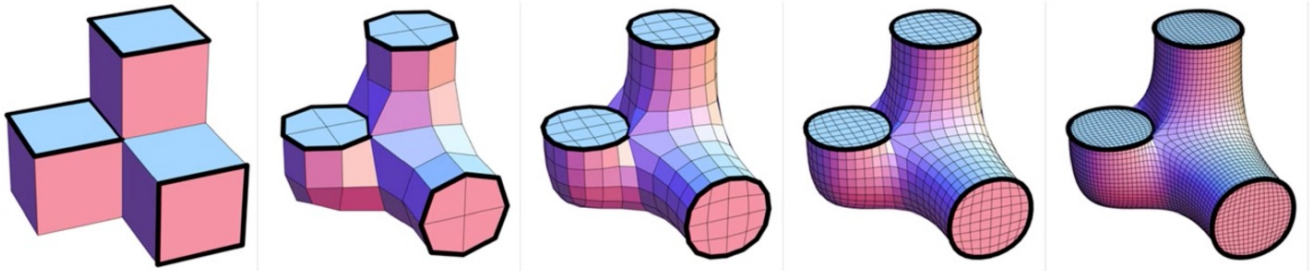


Figure from: Hakenberg et al. Volume Enclosed by Subdivision Surfaces with Sharp Creases

Subdivision in Action (Pixar's "Geri's Game")

Mesh Simplification

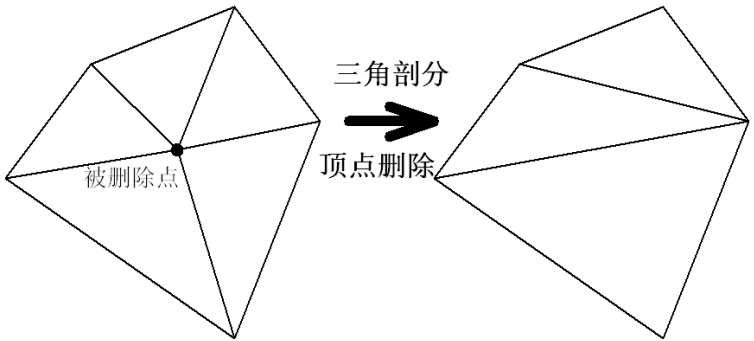
Mesh Simplification

Goal: reduce number of mesh elements while maintaining overall shape

网格化简的基本操作（1）

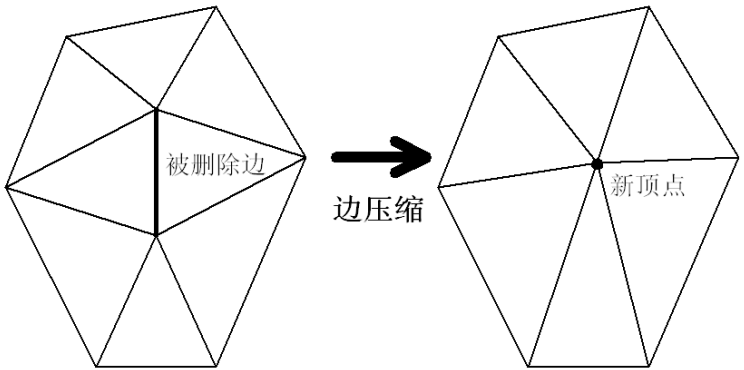
- 三种不同的基本化简操作：

1. **顶点删除操作：**删除网格中的一个顶点，然后对它的相邻三角形形成的空洞作三角剖分



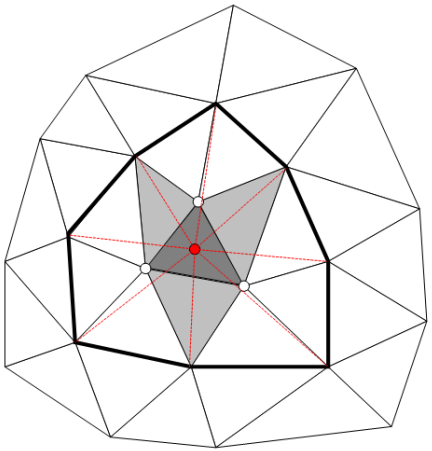
网格化简的基本操作（2）

2. **边压缩操作：**网格上的一条边压缩为一个顶点，与该边相邻的两个三角形的退化



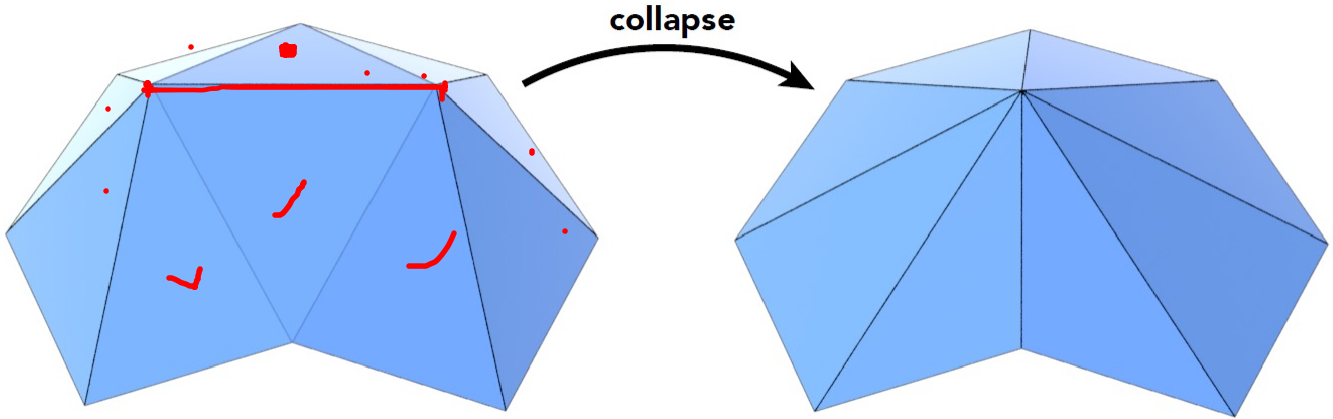
网格化简的基本操作（3）

3. **面片收缩操作：**网格上的一个面片收缩为一个顶点，该三角形本身和与其相邻的三个三角形都退化



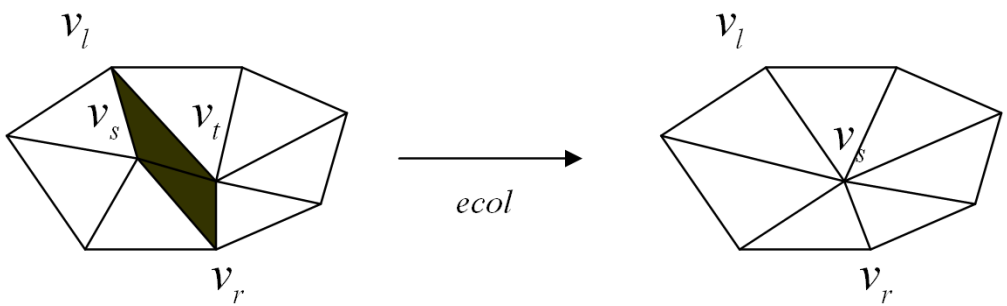
Collapsing An Edge

- Suppose we simplify a mesh using edge collapsing



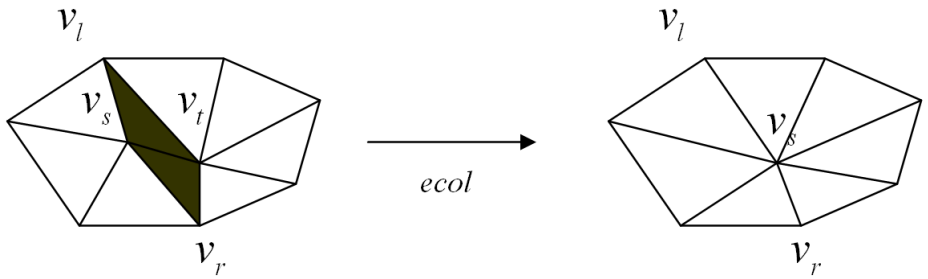
边收缩变换

- Hoppe利用边收缩变换 (edge collapse transformation) 来逐步迭代计算上述能量的优化过程
- 下图给出了一个边收缩变换的过程，它将该边的两个端点(v_s, v_t)收缩为一个顶点v_s，经过这个变形后，其相邻两个面$\{v_s, v_t, v_l\}$和$\{v_t, v_s, v_r\}$均退化为一条边



基于二次误差度量的简化技术

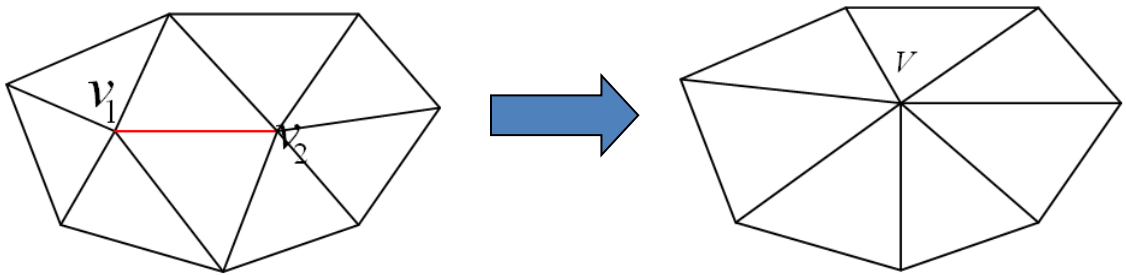
- Hoppe的边收缩(edge collapse)操作可推广为一般的顶点合并变换来描述$(v_1, v_2) \to v$，其含义是将场景中的两个顶点v_1, v_2移到一新的位置v，将连向v_1, v_2的所有边都连向v，并删除所有退化的边和面片



点对合并（1）

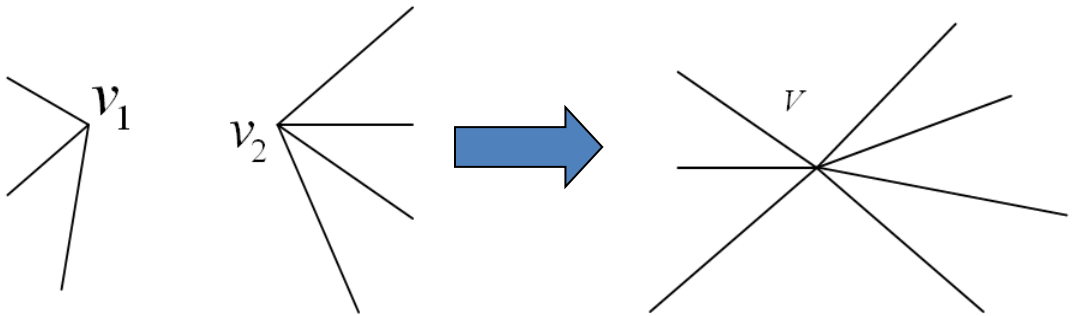
- 点对(v_1, v_2)合并的原则

（1）v_1, v_2 为某一表面上的相邻点，即v_1, v_2 为一条边



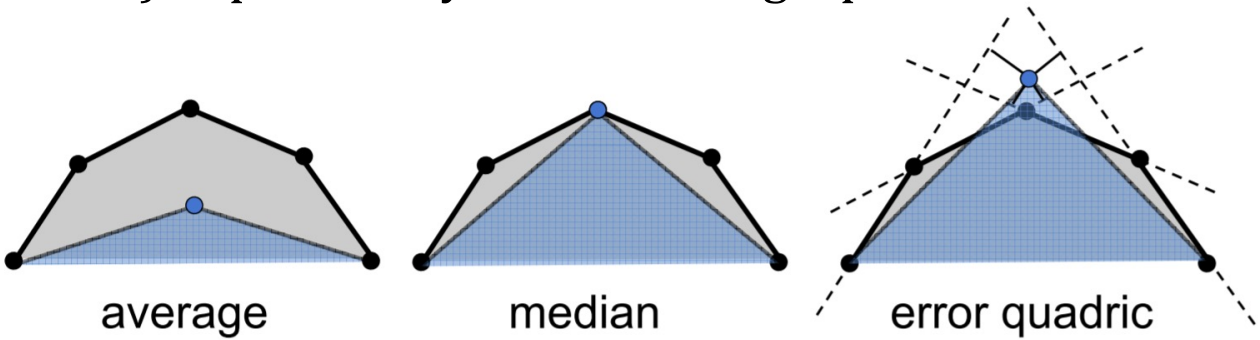
点对合并（2）

（2）$\|v_1 - v_2\| < t$, t 为用户给定的阈值参数



Quadric Error Metrics

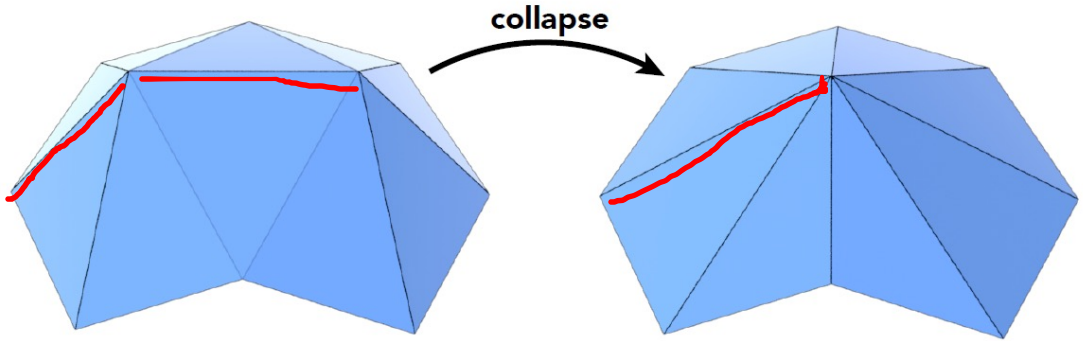
- How much geometric error is introduced by simplification?
- Not a good idea to perform local averaging of vertices
- Quadric error: new vertex should minimize its sum of square distance (L2 distance) to previously related triangle planes!



http://graphics.stanford.edu/courses/cs468-10 fall/LectureSlides/08_Simplification.pdf

Quadric Error of Edge Collapse

- How much does it cost to collapse an edge?
- Idea: compute edge midpoint, measure quadric error



- Better idea: choose point that minimizes quadric error
- More details: Garland & Heckbert 1997.

二次误差度量

- Garland和Heckbert引进了二次误差度量来刻画每一个顶点移动后引起的误差，对表面上的每一个顶点 v_a，均有许多三角面片与之相邻，记 $plane(v_a)$ 为这些三角形所在的平面方程所构成的集合，即

$$plane(v_a)=\left\{(a,b,c,d)\,\middle|\,\begin{array}{l}ax+by+cz+d=0,(a^2+b^2+c^2=1)\\ \text{is coefficients of the adjacent plane of } v_a\end{array}\right\}$$

二次误差度量

- 则我们采用如下的二次函数来度量 v_a 移动到 v 时产生的误差：

$$\Delta(v_a \to v) = \sum_{p \in plane(v_a)} (pv^T)^2 \qquad (1)$$

上式计算的是 v 到与 v_a 相邻的各个三角形面片所在平面的距离的平方

其中 v=(x, y, z, 1) 为齐次坐标，展开上式得到

$$\Delta(v_a \to v) = \sum_{p \in plane(v_a)} (pv^T)^2 = v\left(\sum_{p \in plane(v_a)} K_p\right)v^T = vQ(v_a)v^T \qquad (2)$$

$$Q(v_a) = \sum_{p \in plane(v_a)} K_p$$

二次误差度量

- （2）式中

$$K_p = p^T p = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \qquad (3)$$

- 这样，对每一顶点 v_a，在预处理时，我们均可按上述方法计算矩阵$Q(v_a)$，进而就可对其移动进行误差度量了
- 但由于每次合并时，需同时移动两点，故必须考虑同时移动多个顶点后形成的误差

多点移动的误差

- Garland和Heckbert简单地采用加法规则来刻画多点移动而形成的误差，对点对合并$(v_1, v_2) \to v$，其误差为

$$\Delta(v) = \Delta(v_1 \to v) + \Delta(v_2 \to v) = v(Q(v_1) + Q(v_2))v^T = vQv^T$$

其中 $Q = Q(v_1) + Q(v_2)$

因而，应选取$\boldsymbol{v}$使误差达到最小

多点移动的误差

- 由极值的性质知，v满足系统方程：

$$\frac{\partial \Delta(v)}{\partial x}=\frac{\partial \Delta(v)}{\partial y}=\frac{\partial \Delta(v)}{\partial z}=0$$

即

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} v=\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- 若上式有唯一解，则其解为v的最优解；否则，用伪逆技术求v
- 若伪逆技术失败，则简单地选取v为v_1，v_2或 $\frac{v_1+v_2}{2}$ 中的任何一个

Simplification via Quadric Error

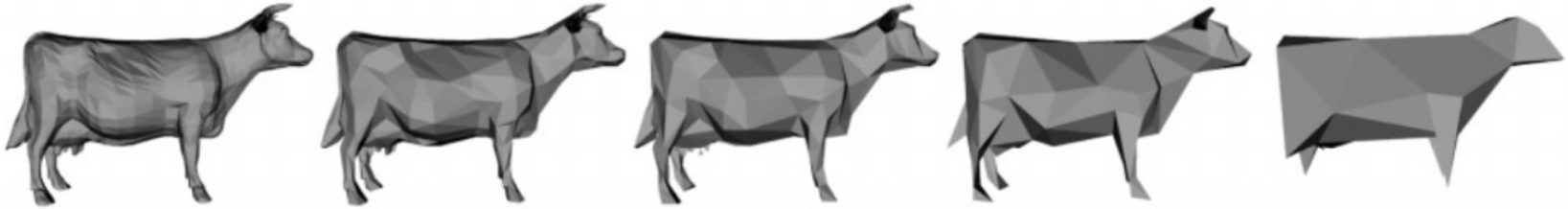
Iteratively collapse edges

Which edges? Assign score with quadric error metric*

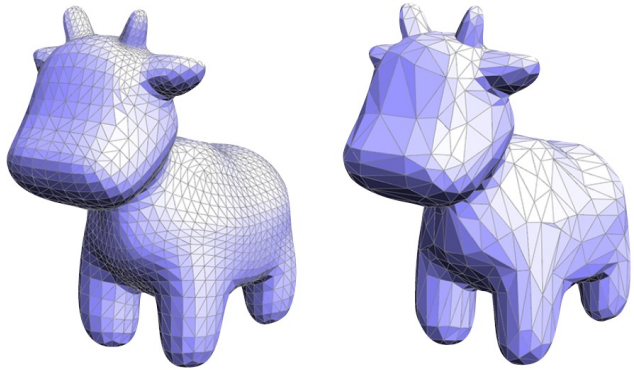
- approximate distance to surface as sum of distances to planes containing triangles
- iteratively collapse edge with smallest score
- greedy algorithm… great results!

* (Garland & Heckbert 1997)

Quadric Error Mesh Simplification



5,804 994 532 248 64



Garland and Heckbert '97

基于二次误差度量的简化技术

- 参考文献
  - Garland M., Heckbert P S., Surface simplification using quadric error matrix, Computer Graphics, 1997, 209-216.

Thank you!