计算机图形学

网格及其处理

陶钧
taoj23@mail.sysu.edu.cn
中山大学 数据科学与计算机学院
国家超级计算广州中心

网格及其处理

基本概念

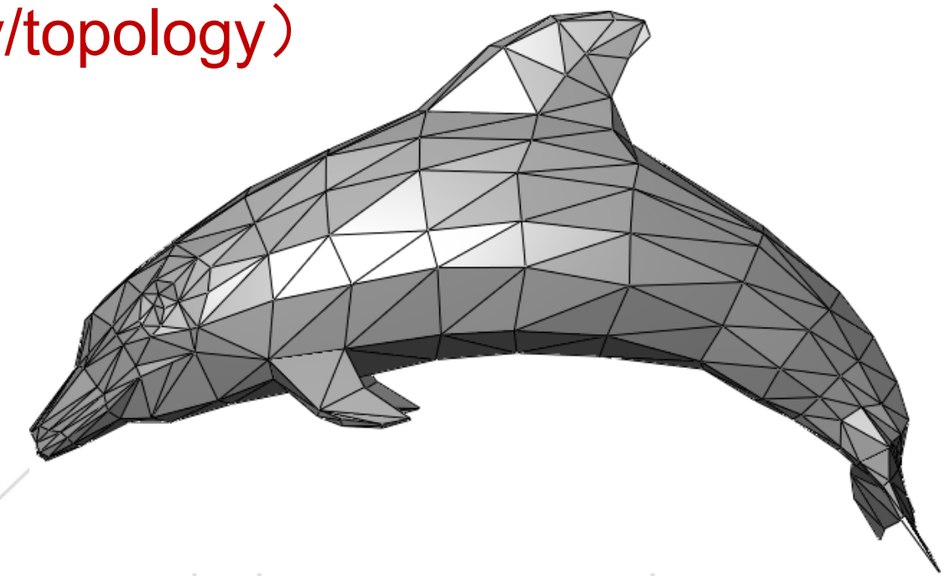
网格性质

数据结构

网格处理

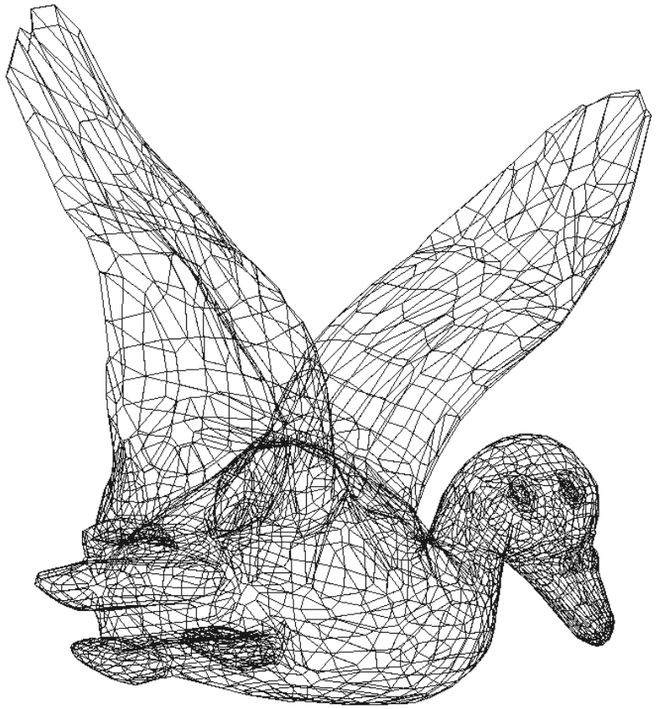
基本概念

- 网格（Mesh）
  - 也常称为多边形网格（polygonal mesh）
  - 由三种基本元素构成
    - 顶点（vertices），边（edges），面（faces）
  - 这些元素描述了网格中所包含的信息
    - 连接/拓扑信息（mesh connectivity/topology）
      - 元素间是否邻接的关系
      - 如，面所对应的边和顶点
    - 几何信息（mesh geometry）
      - 位置及其他几何特性
      - 如，顶点的位置



基本概念

- 网格（Mesh）
  - 多面体（polyhedral）野鸭模型
    - 3474个顶点，6656个面



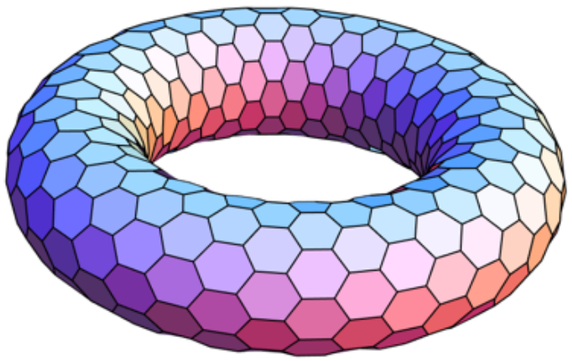
基本概念

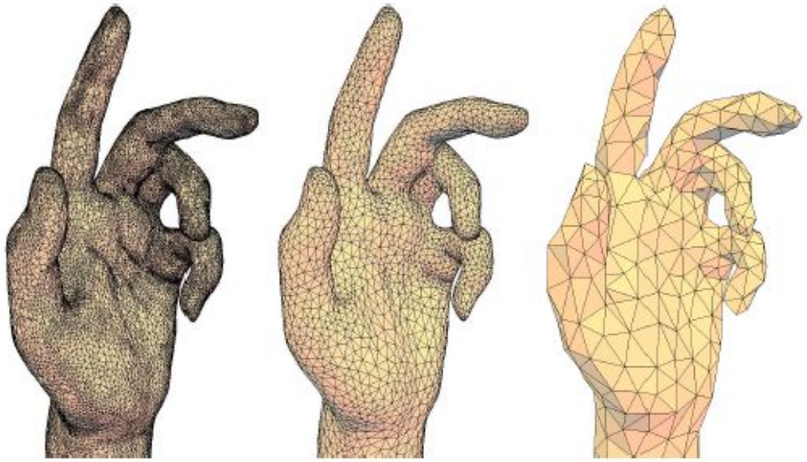
网格（Mesh）

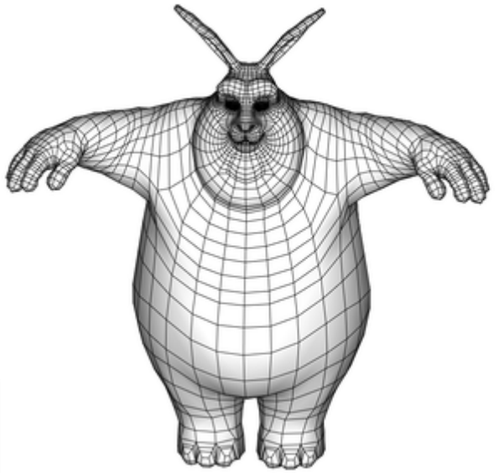
– 多面体是几何物体的线性近似

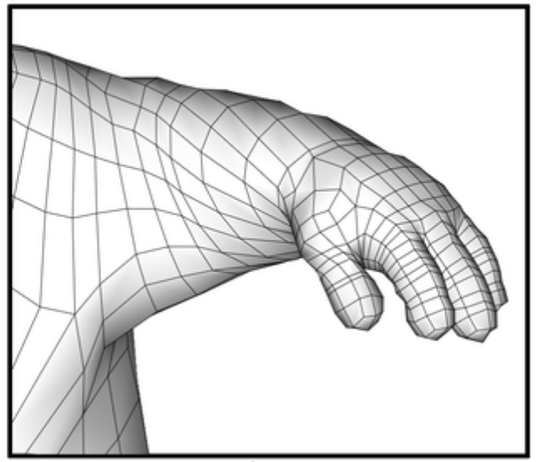
– 根据组成网格的多边形类型，网格可分为

- 三角网格（triangular meshes）
- 四边形网格（quadrilateral meshes）
- 多边形网格（polygonal meshes）







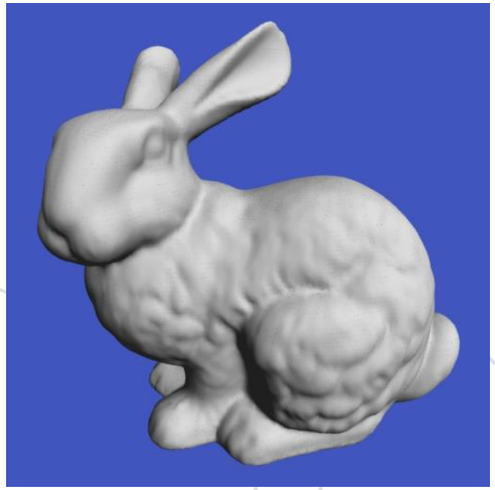


构建三维物体的网格表示

图形学中的许多常用模型都是使用真实三维物体产生的

Stanford dragon

- Faces ： 871414
- Vertices：437645
- Compressed：8.2 MB in PLY format



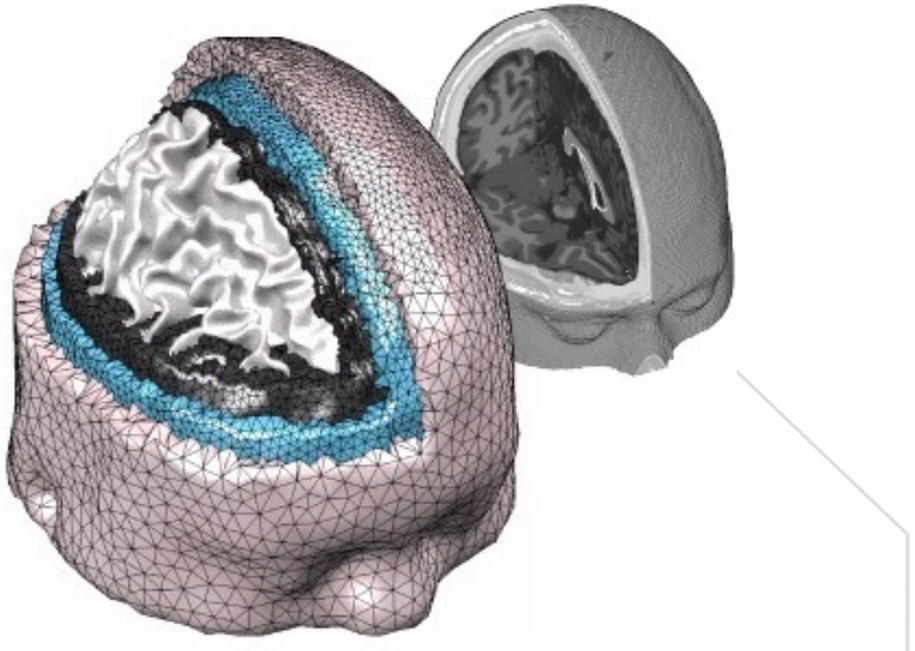
构建三维物体的网格表示

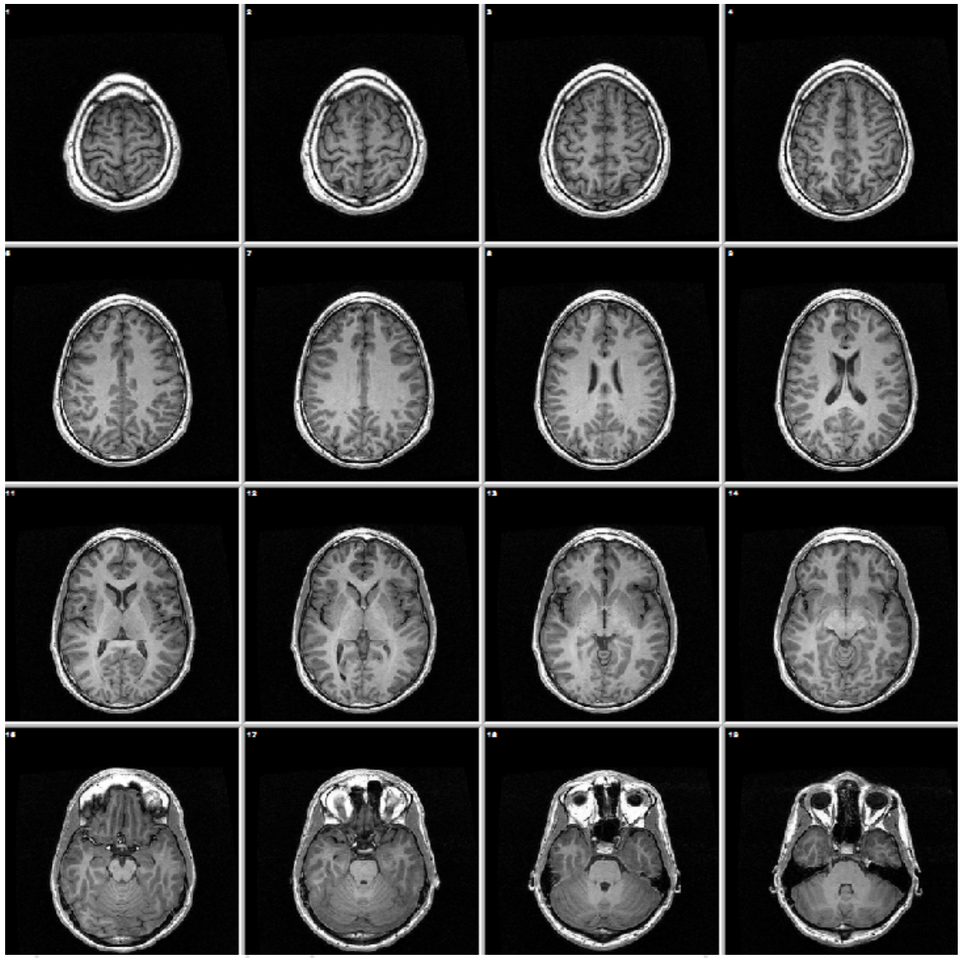
Volumetric scanning

– 从扫描切片中构建体数据（volume）

– 从体数据中构建网格

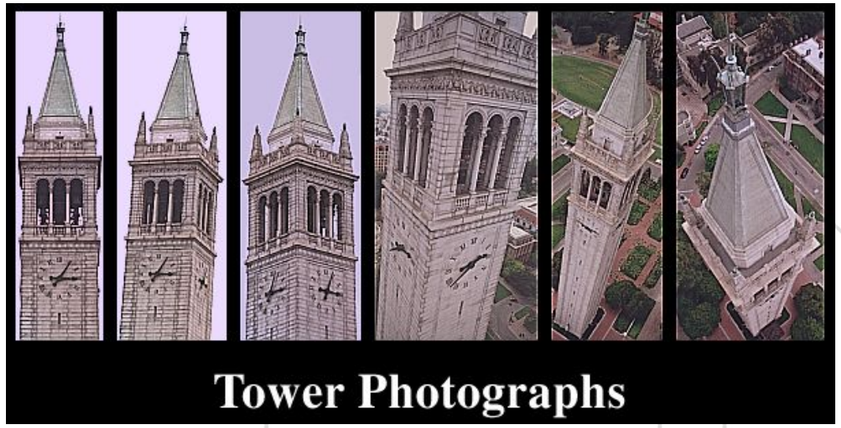
- 如，marching cubes

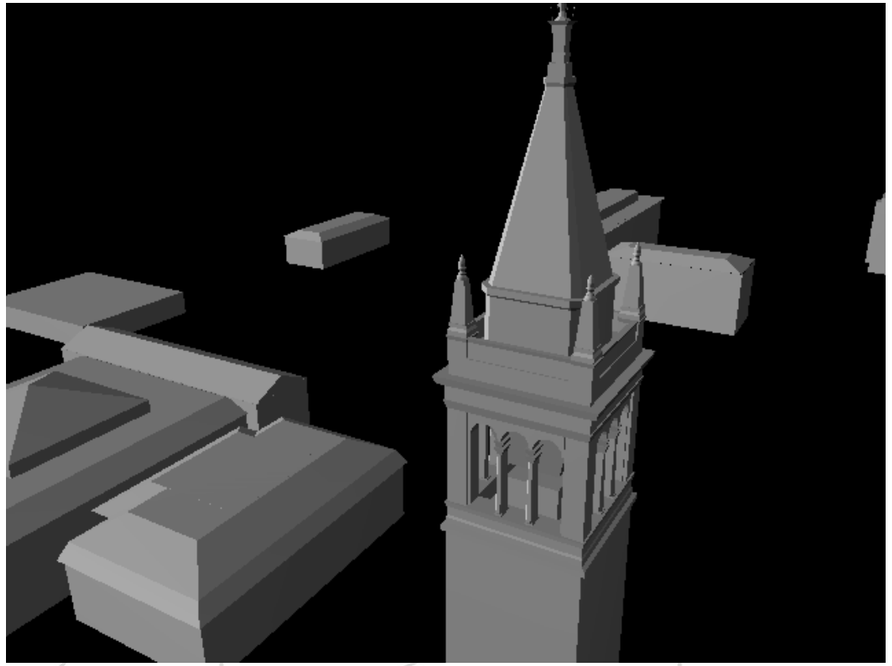




构建三维物体的网格表示

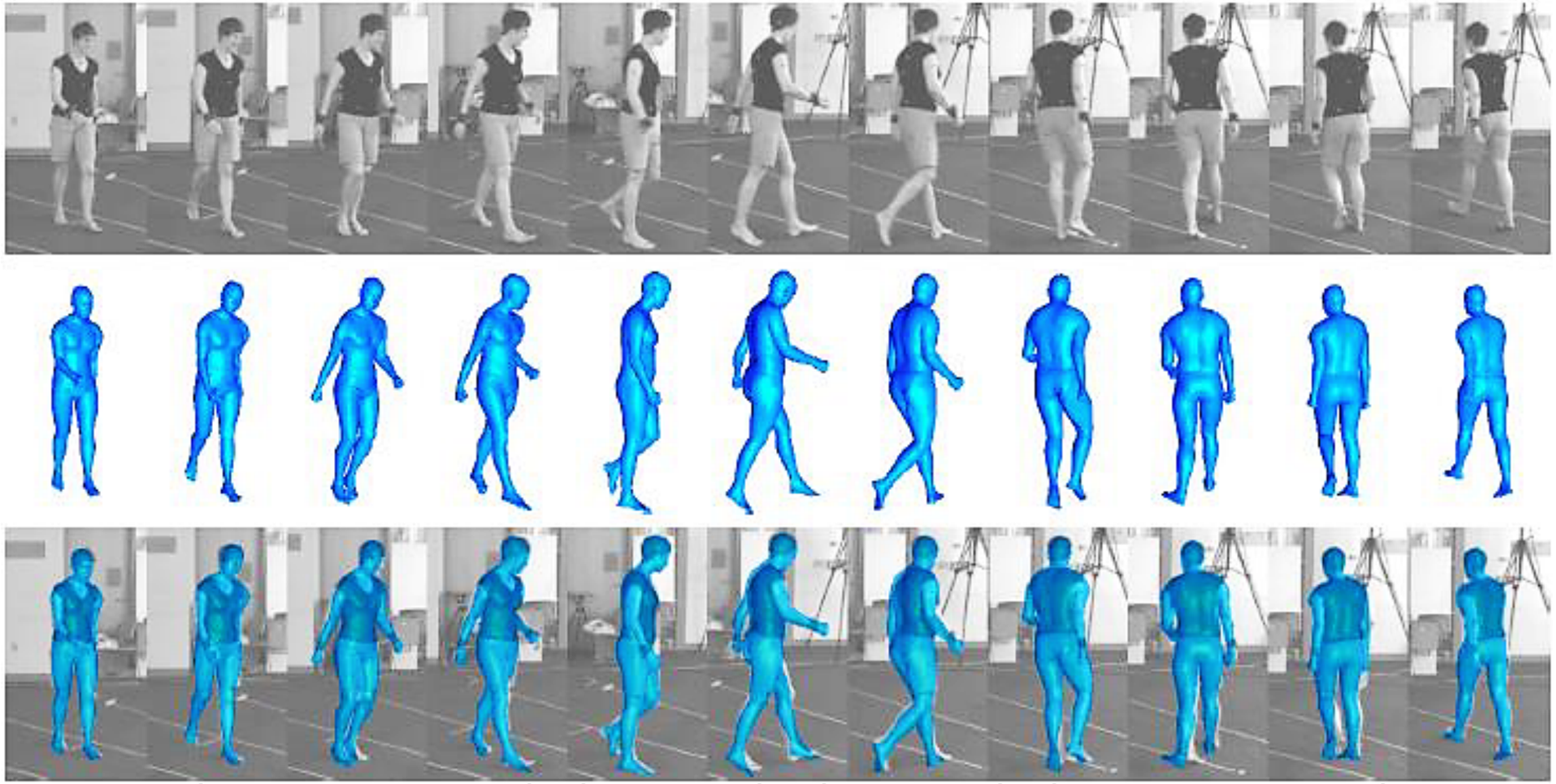
- Photogrammetry
  - 从多角度拍摄的照片中重建静态几何物体





构建三维物体的网格表示

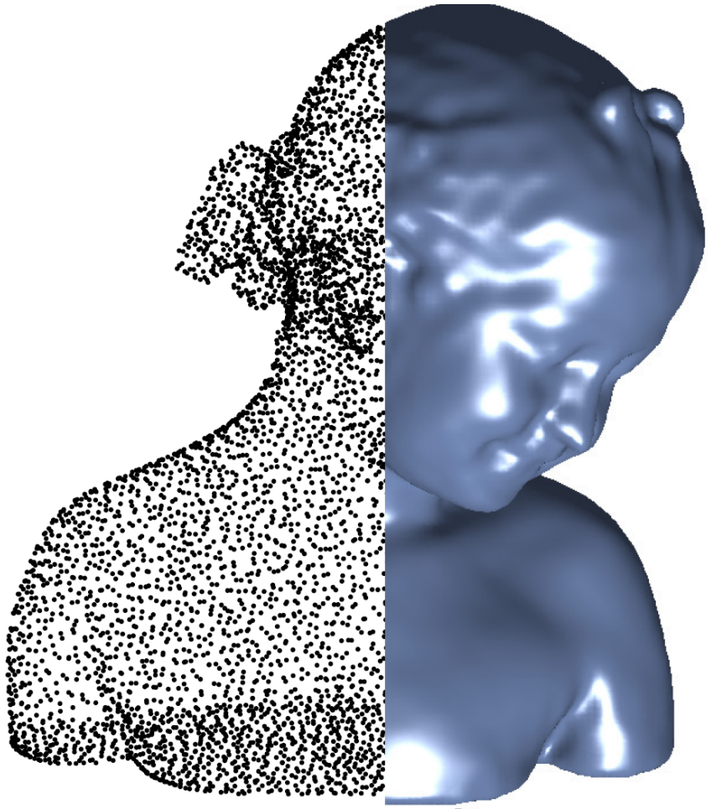
- Photogrammetry
  - 从一组动态照片中重建几何物体

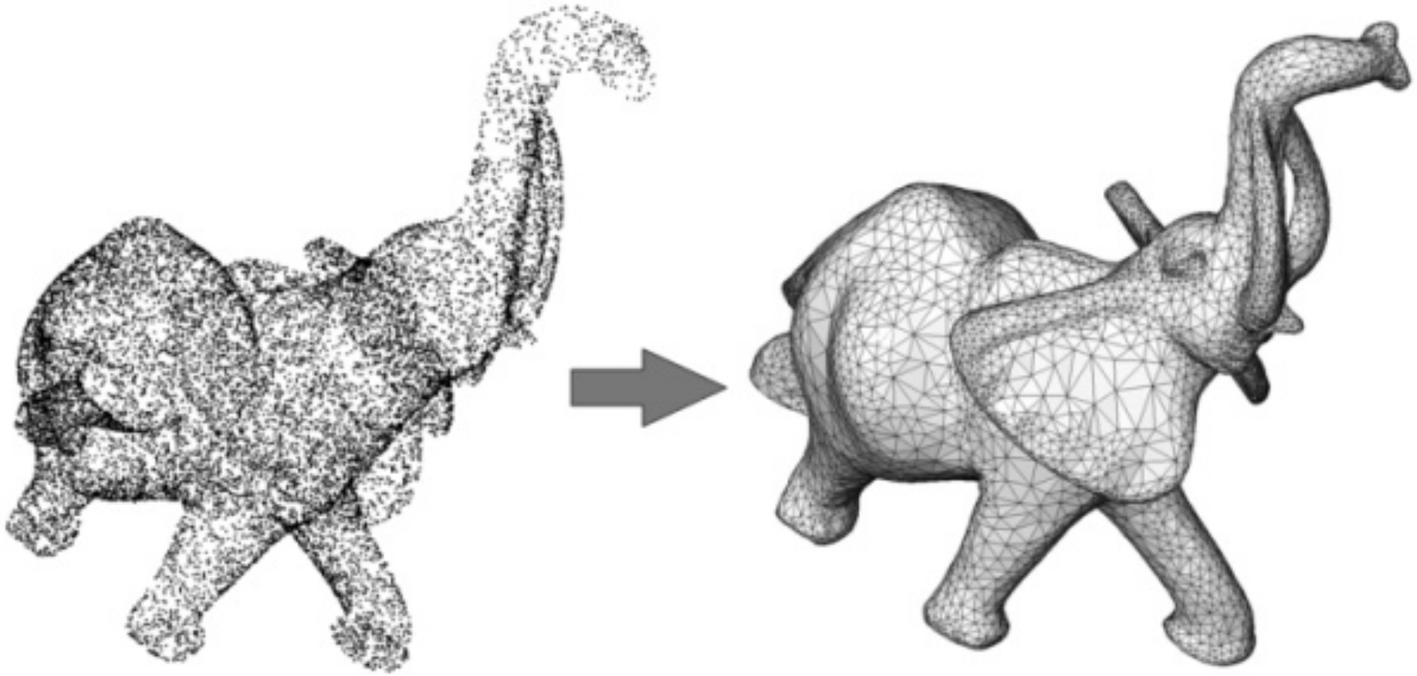


构建三维物体的网格表示

Range scanning

– 从点云数据中重建物体

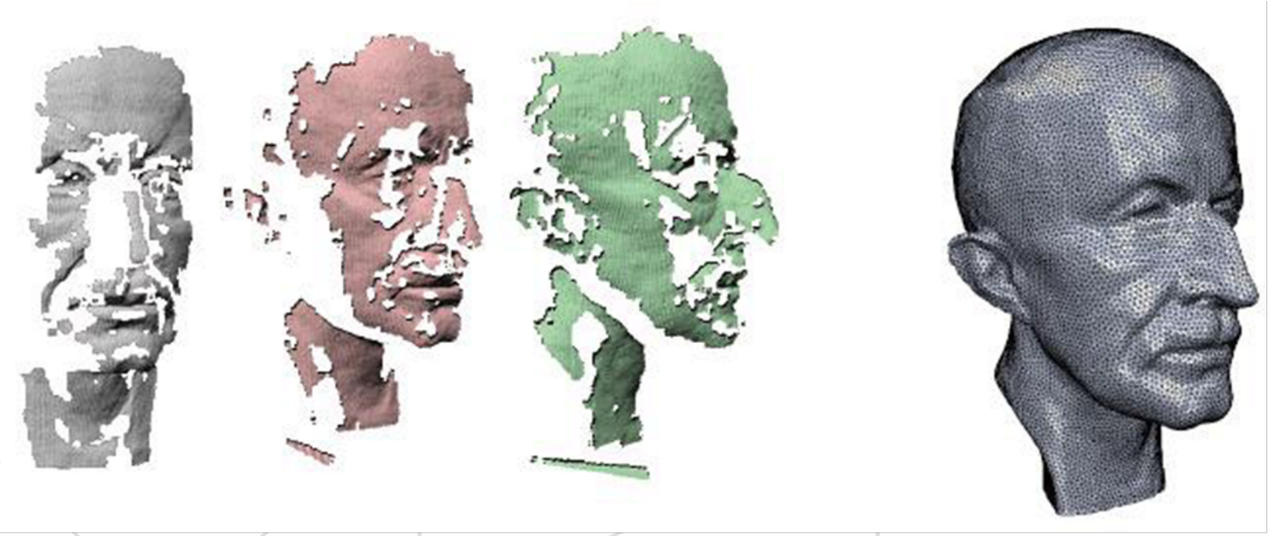




构建三维物体的网格表示

Range scanning

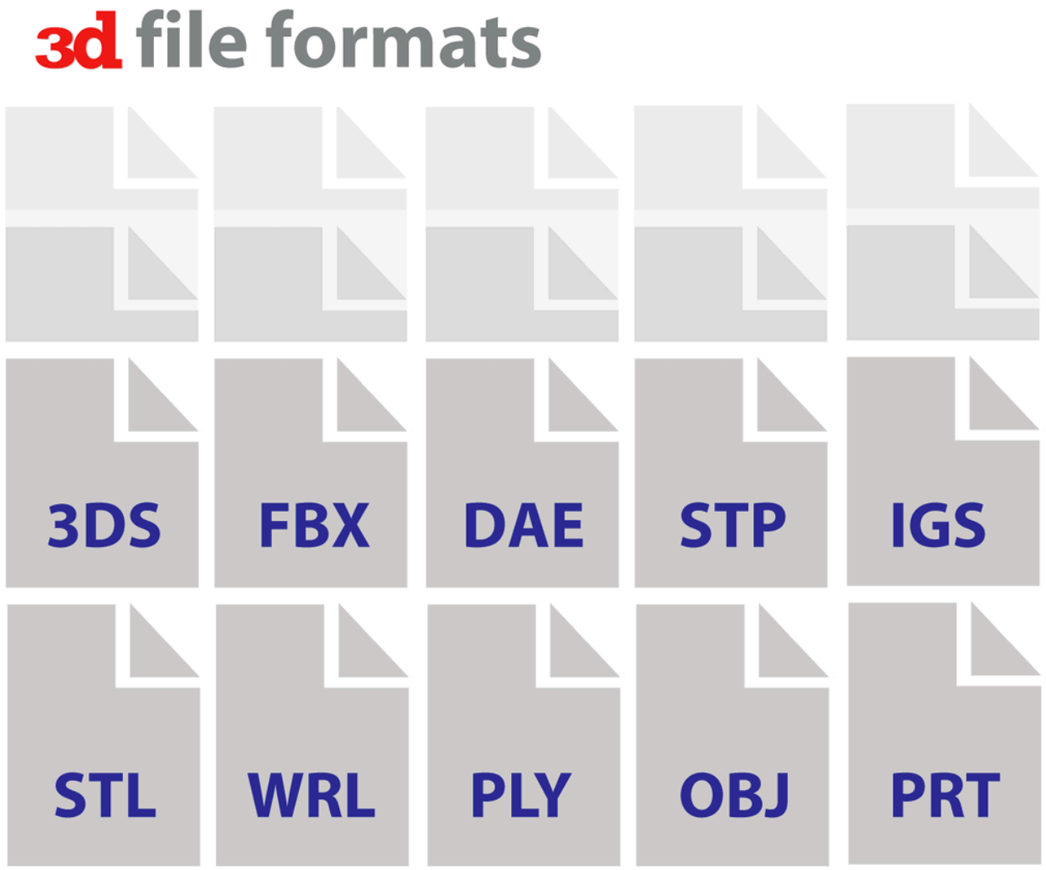
- 从点云数据中重建物体
  - 精确地校准至关重要
  - 对复杂物体产需要进行多次扫描
    - 扫描路径规划
    - 扫描配准（registration）
  - 结果常不完整并带有噪声
    - 模型修复，hole filling
    - 平滑与噪声消除



Set of raw scans　　Reconstructed model

网格表示

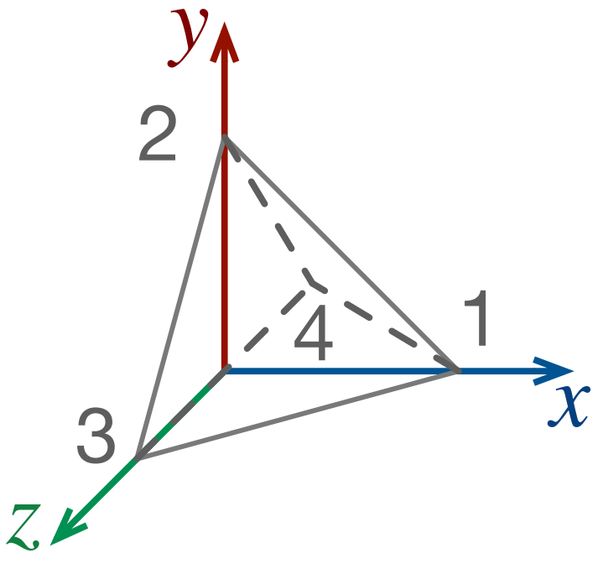
- 常见网格文件格式
  - Wavefront OBJ（*.obj）
  - 3D Max（*.max, *.3ds）
  - VRML（*.vrl）
  - Inventor（*.iv）
  - PLY（*.ply, *.ply2）
  - User-defined
- 存储形式
  - 文本
  - 二进制



网格表示

Wavefront .obj文件格式

- 顶点（vertices）
  - 以字符 v 开头
  - x, y, z坐标
- 面（faces）
  - 以字符 f 开头
  - 对应顶点在文件中的index
- 其他属性
  - 法向量，纹理坐标，材料，等




```
v  1.0  0.0  0.0
v  0.0  1.0  0.0
v  0.0  0.0  1.0
v  0.0  0.0 -1.0
f  1  2  3
f  1  3  4
f  1  4  2
f  2  4  3
```

网格表示

Wavefront .obj文件格式

```
# List of Vertices, with (x,y,z[,w]) coordinates, w is optional and defaults to 1.0.
  v 0.123 0.234 0.345 1.0
  v ...
  ...
  # Texture coordinates, in (u ,v [,w]) coordinates, these will vary between 0 and 1, w is optional and
default to 0.
  vt 0.500 1 [0]
  vt ...
  ...
  # Normals in (x,y,z) form; normals might not be unit.
.
  vn 0.707 0.000 0.707
  vn ...
  ...
  # Parameter space vertices in ( u [,v] [,w] ) form; free form geometry statement ( see below )
  vp 0.310000 3.210000 2.100000
  vp ...
  ...
  # Face Definitions (see below)
  f 1 2 3
  f 3/1 4/2 5/3
  f 6/4/1 3/5/3 7/6/5
  f ...
  ...
```

网格

基本概念

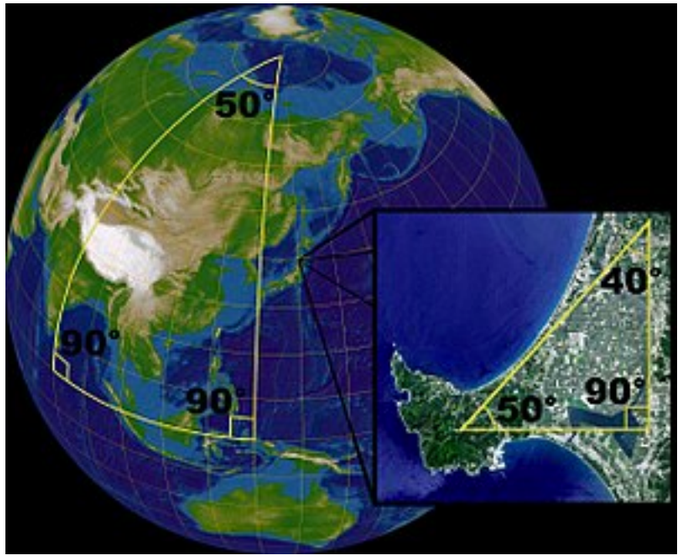
网格性质

数据结构

网格处理

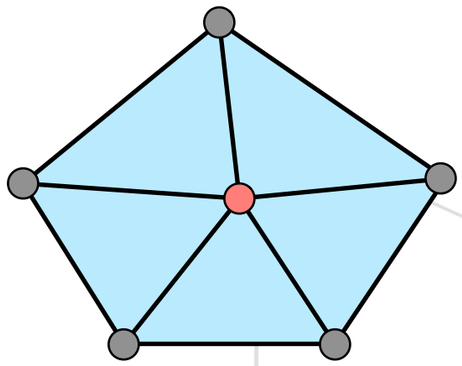
网格性质

- 流形（manifold）
  - 局部具有欧几里得空间性质的空间
    - 一般可由许多平直的片粘连而成
    - 如，欧几里得空间本身，地球的球面
  - 为研究几何形体的拓扑、微分等提供了一个自然的平台
    - 拓扑结构：“柔软”的结构
      - 所有变形（同胚）保持拓扑结构不变
    - 解析几何结构：“硬”结构
      - 整体结构固定
    - 流形介于两者之间
      - 其无穷小结构是“硬”的（可容纳微分结构）
      - 整体是“柔软”的（使其能连接独立的局部模型）

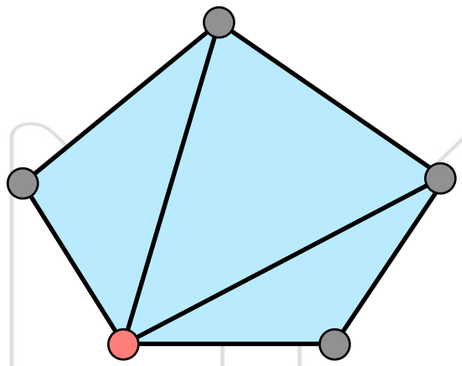


网格性质

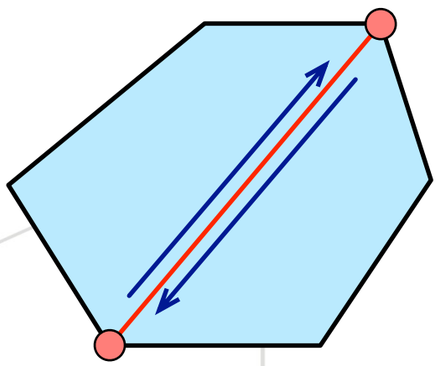
- 流形（manifold）
  - 满足以下两个条件的网格为流形
    - 每条边只属于一个或两个面
    - 每个顶点对应的面形成一个闭合（closed）或开放（open）的扇面
  - 每个面的定向（orientation)由其顶点顺序决定
    - 顺时针、逆时针
    - 两个相邻面的定向兼容（compatible）：其共有边上顶点的访问顺序相反
    - 流形为可定向（orientable）：任意两个相邻面的定向都是兼容的



closed fan



open fan



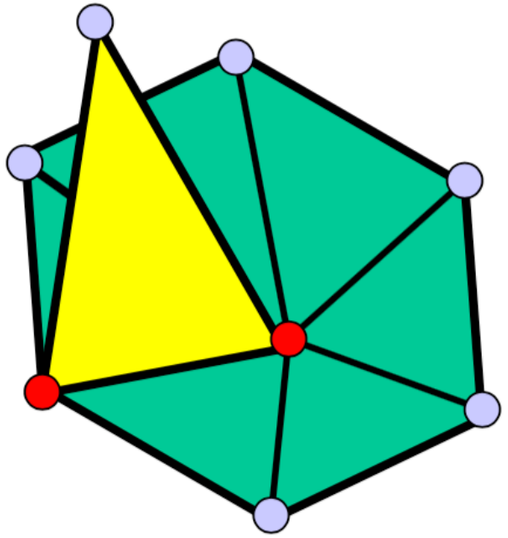
compatible

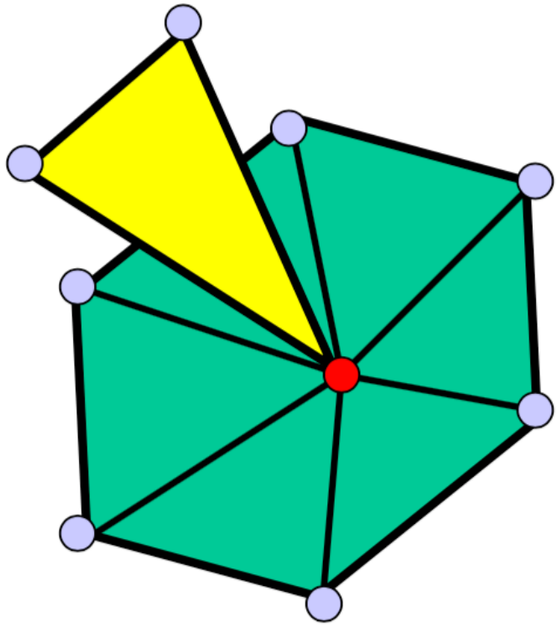
网格性质

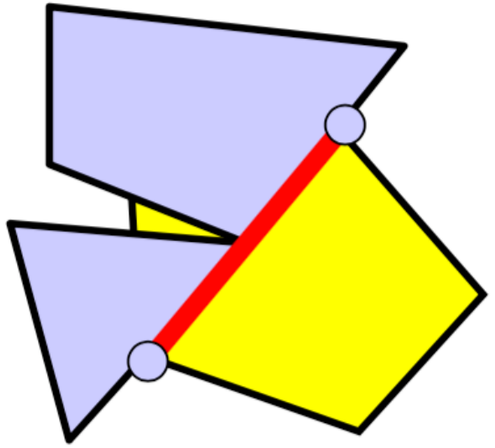
流形（manifold）

– 满足以下两个条件的网格为流形
  - 每条边只属于一个或两个面
  - 每个顶点对应的面形成一个闭合（closed）或开放（open）的扇面

– 以下网格都不是流形！





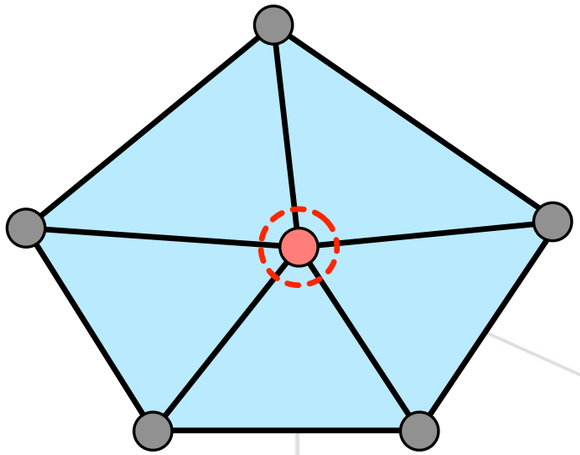


网格性质

流形（manifold）

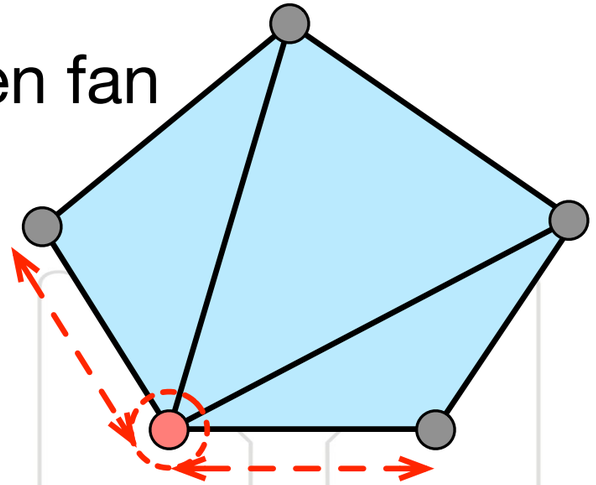
– 带边界与无边界
  - 如果流形中每一个顶点都对应一个闭合的扇面，则流形是无边界的
  - 对于带边界的流形，只属于一个面的边形成了流形的边界
  - 边界是简单多边形组成的并集

– 我们通常使用的几何形体为无边界可定向的流形

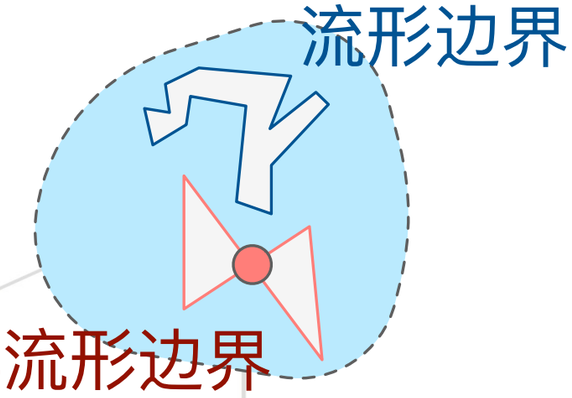


closed fan

open fan



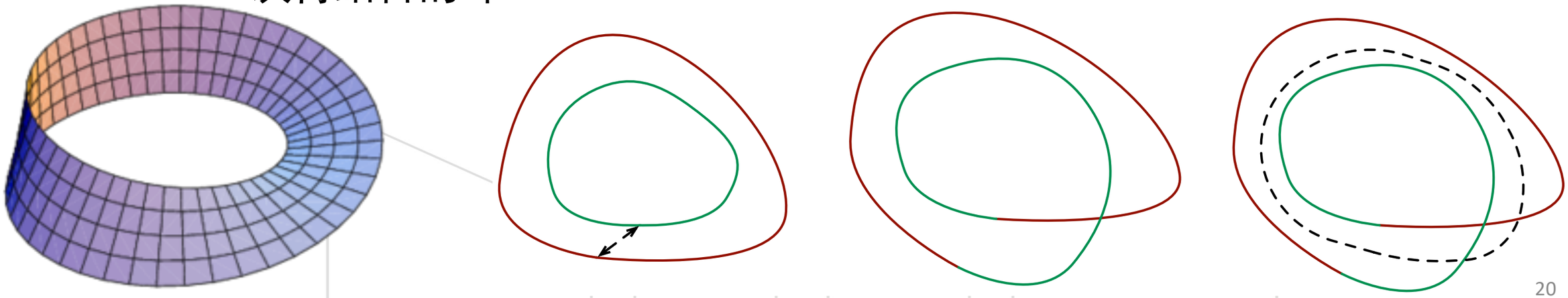
边界



网格性质

流形（manifold）

- 不可定向（non-orientable）的流形
  - 并非所有流形都是可定向的
  - 如，莫比乌斯环（Möbius band）
    - 将纸带扭转半圈后将两端粘上
    - 只有一个面和一条边界的曲面：最简单的带边界不可定向的流形
    - 沿中间剪开莫比乌斯环不会得到两条窄带子，而是一个更大的扭转了两次再结合的环

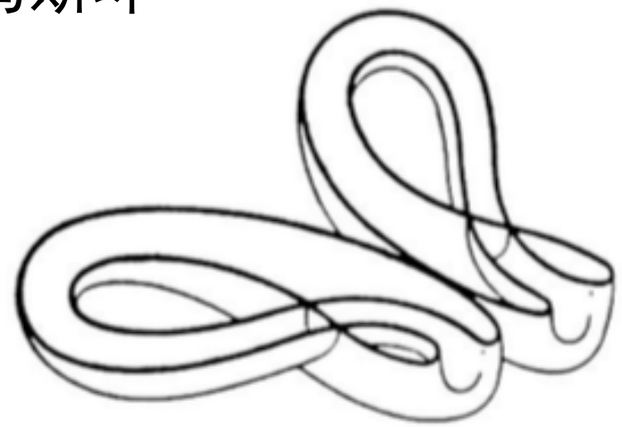


网格性质

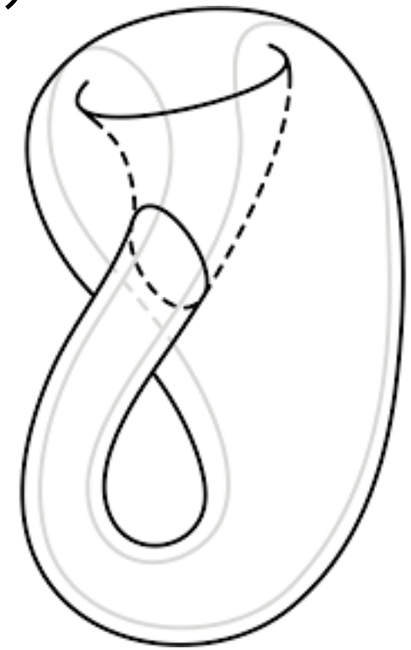
流形（manifold）

- 不可定向（non-orientable）的流形
  - 并非所有流形都是可定向的
  - 如，克莱因瓶（Klein bottle）
    - 只有一个表面：无法在三维空间中表示（可以用来装水吗？）
    - 无边界不可定向的流形
    - 切开克莱因瓶可得到两个莫比乌斯环

By Tin-Tin Yu, MTU

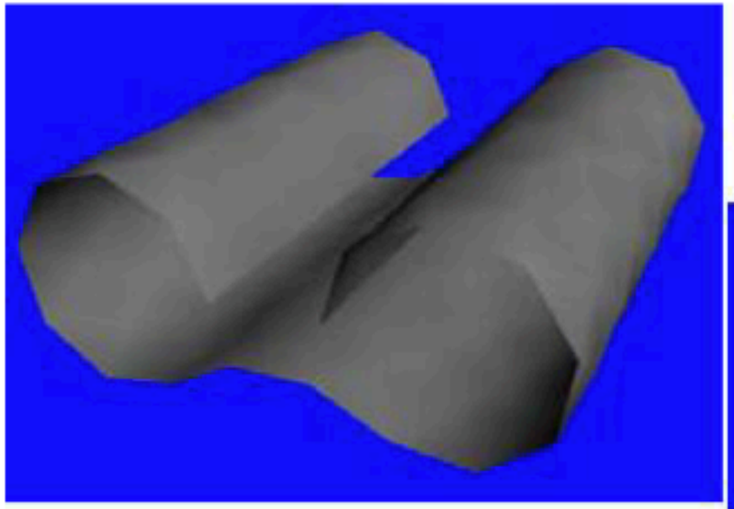


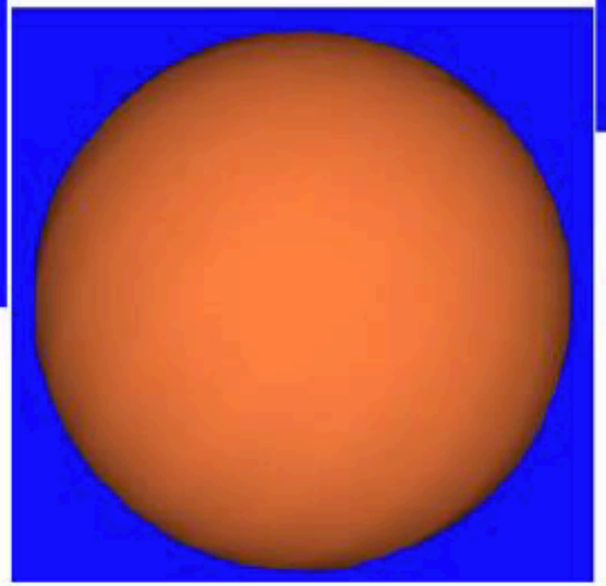
Maurice Fréchet and Ky Fan,
Invitation to Combinatorial Topology

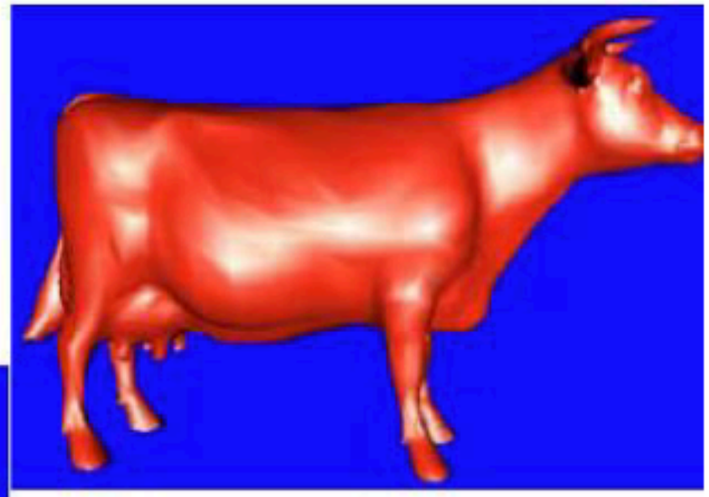


网格性质

- 可展性（developability）
  - 网格是可展的（developable）如果其能在无形变的情况下嵌入到二维欧几里得空间



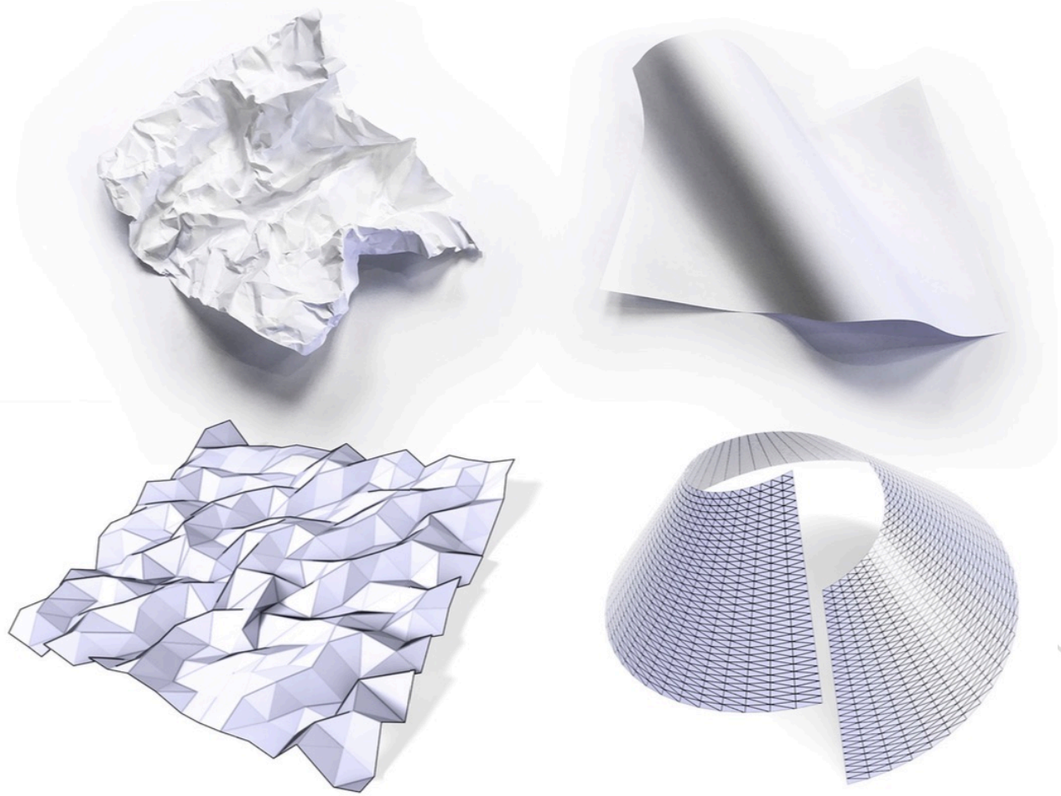


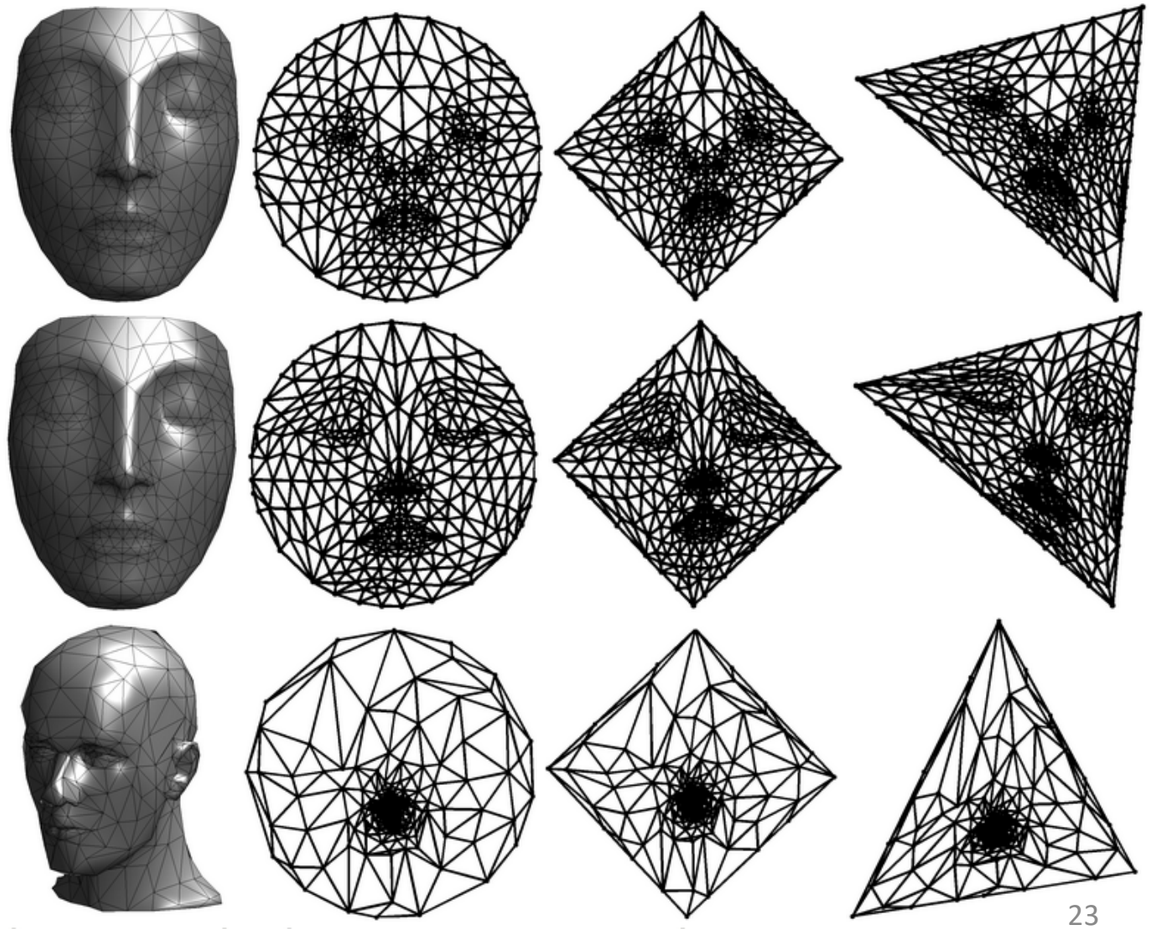


网格性质

可展性（developability）

– 网格是可展的（developable）如果其能在无形变的情况下嵌入到二维欧几里得空间

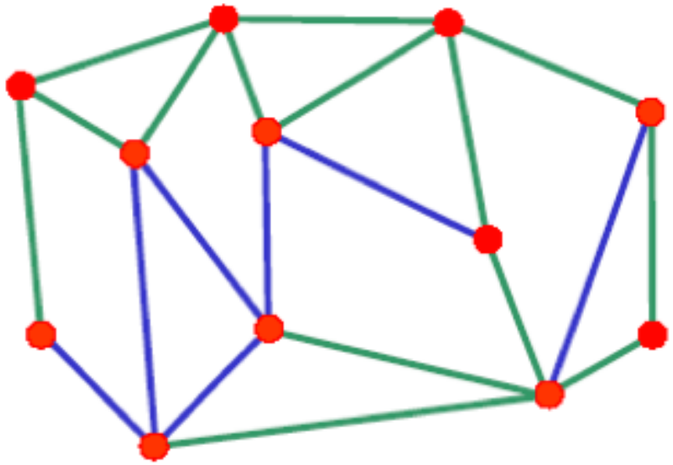




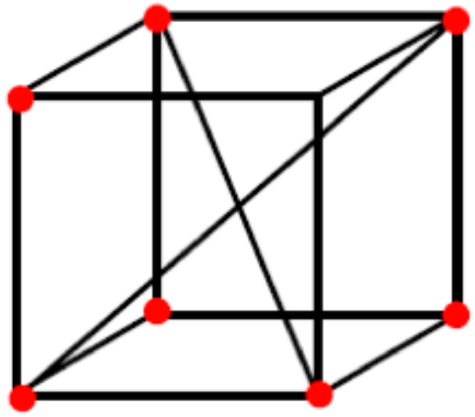
网格性质

网格与图

- 网格：顶点，边，面
  - 几何信息与拓扑信息
- 图：顶点，边，（面？）
  - 图本身没有几何信息（如顶点位置），只有拓扑信息
  - 给图中顶点分配位置，可将图嵌入到d维欧几里得空间



Embedding in R^2



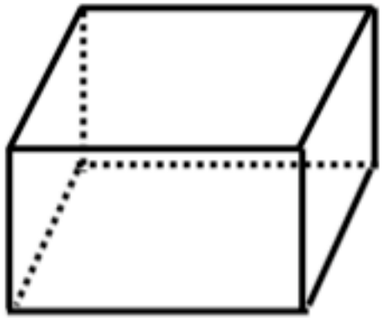
Embedding in R^3

网络性质

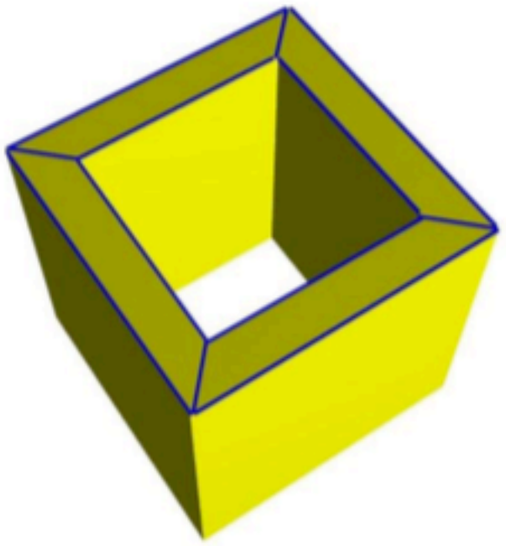
欧拉示性数（Euler–Poincaré characteristic）

$-\chi(\boldsymbol{M}) = F - E + V$

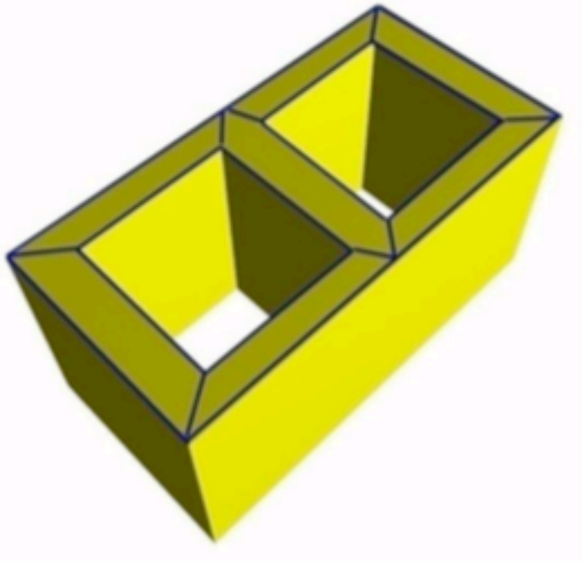
- $\boldsymbol{M}$为网格，F，E，V分别为其中面、边、顶点的数量



V=8, E=12, F=6
$\chi(\boldsymbol{M})$ =V-E+F=2



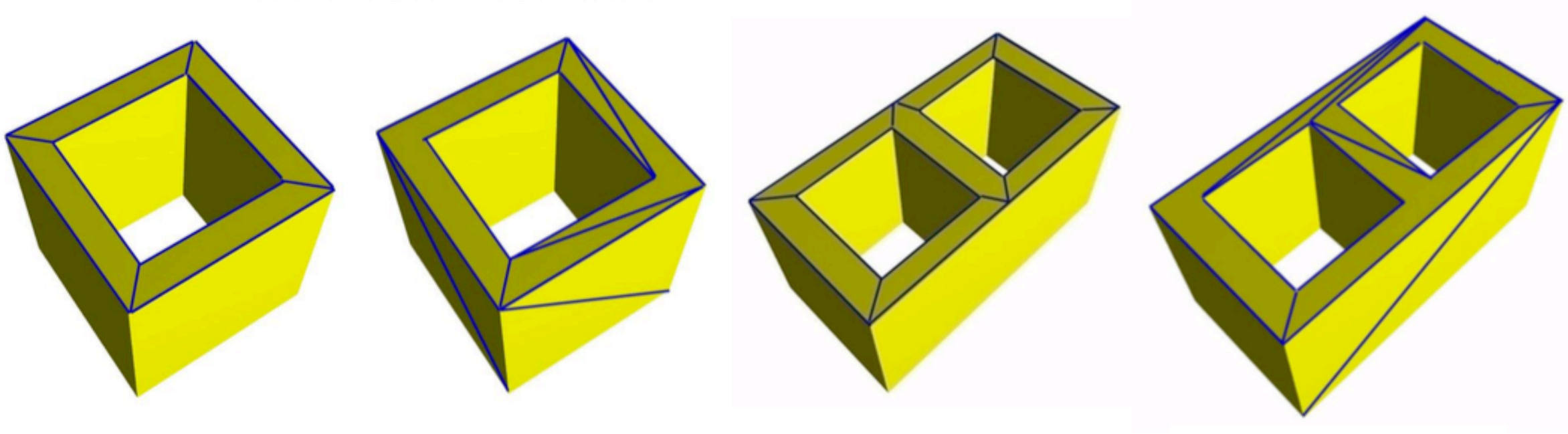
V=16, E=32, F=16
$\chi(\boldsymbol{M})$ =V-E+F=0



V=28, E=56, F=26
$\chi(\boldsymbol{M})$ =V-E+F=-2

网络性质

- 欧拉示性数（Euler–Poincaré characteristic）
  - $\chi(\boldsymbol{M}) = F - E + V$
    - 欧拉示性数只与几何形体相关，与其剖分（tessellation）无关
      - 跟形体的什么特征相关？



V=16, E=32, F=16
$\chi(\boldsymbol{M})$ =V-E+F=0

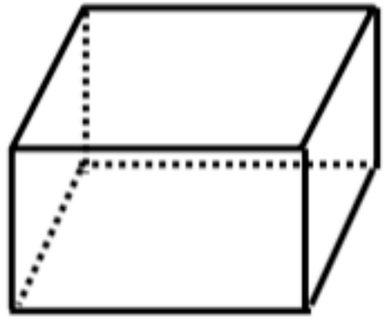
V=16, E=36, F=20
$\chi(\boldsymbol{M})$ =V-E+F=0

V=28, E=56, F=26
$\chi(\boldsymbol{M})$ =V-E+F=-2

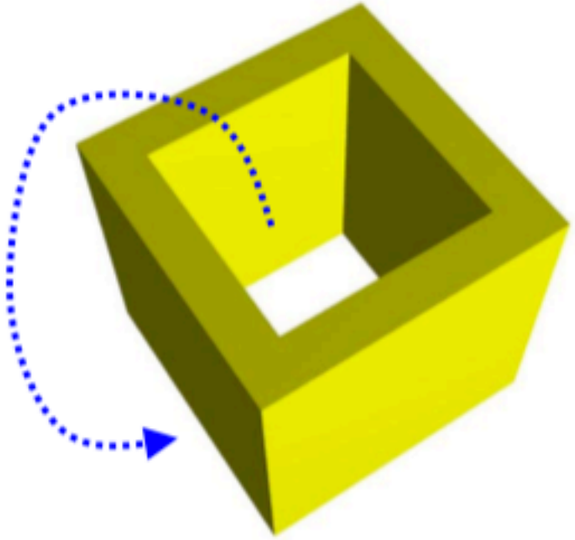
V=24, E=48, F=22
$\chi(\boldsymbol{M})$ =V-E+F=-2

网络性质

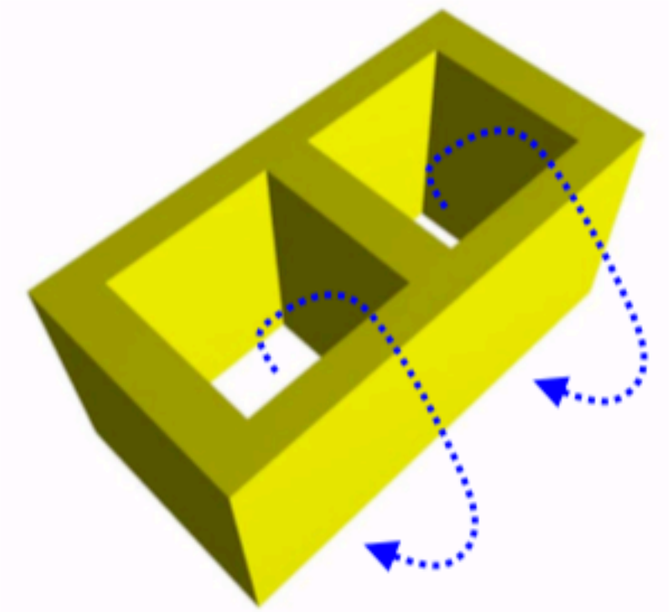
- 欧拉示性数（Euler–Poincaré characteristic）
  - $\chi(\boldsymbol{M}) = F - E + V = 2 \cdot (1 - g)$
    - 适用于可定向无边界网格
    - g 亏格（genus）：网格对应的几何形体的“把手（handle）”个数
    - 球，立方体，四面体等$g = 0$；圆环$g = 1$



$g=0 \Rightarrow \chi(\boldsymbol{M}) = 2(1-0) = 2$



$g=1 \Rightarrow \chi(\boldsymbol{M}) = 2(1-1) = 0$



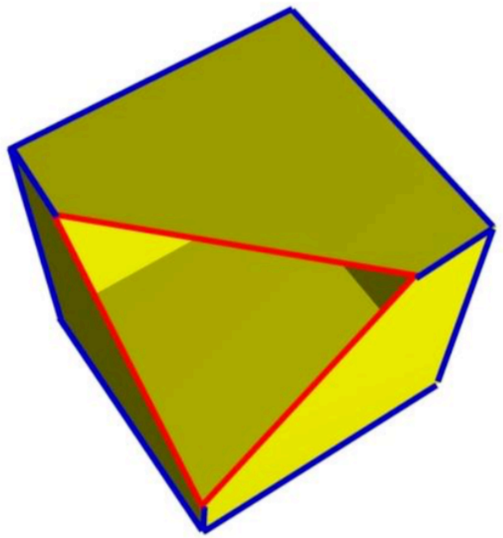
$g=2 \Rightarrow \chi(\boldsymbol{M}) = 2(1-2) = -2$

网络性质

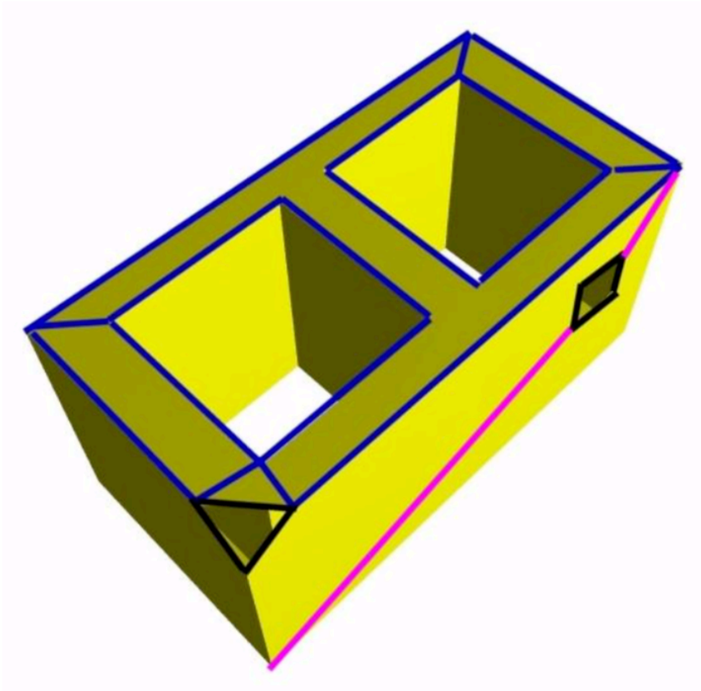
欧拉示性数（Euler–Poincaré characteristic）

$$-\chi(\boldsymbol{M}) = F - E + V = 2 \cdot (1 - g) - \partial$$

- ∂：边界数目



V = 10, E = 15, F = 6
g = 0, ∂ = 1
χ(**M**) = V-E+F = 1
χ(**M**) = 2(1-g)-∂ = 1

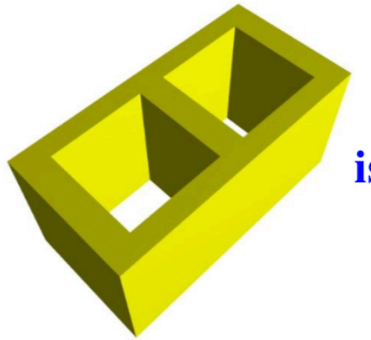


V = 30, E = 54, F = 20
g = 2, ∂ = 2
χ(**M**) = V-E+F = -4
χ(**M**) = 2(1-g)-∂ = -4

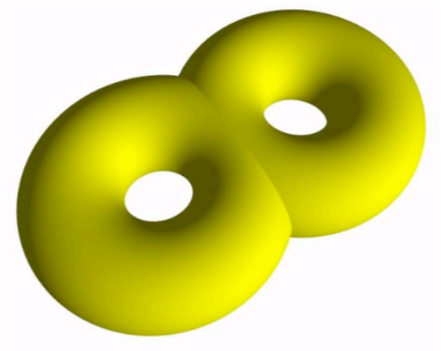
网络性质

拓扑（topology）

- 连续变化下维持不变的性质
  - 拉伸或弯曲，但不包括撕开或粘合
  - 连通性、紧致性
  - “A topologist cannot distinguish coffee mug from doughnut” –JK Kelley
- 同胚（homeomorphic）
  - 两个拓扑空间（如网格）间存在连续变化
  - 拓扑空间中的同构
  - 具有同样数目的“洞”



is homeomorphic to



网格及其处理

基本概念

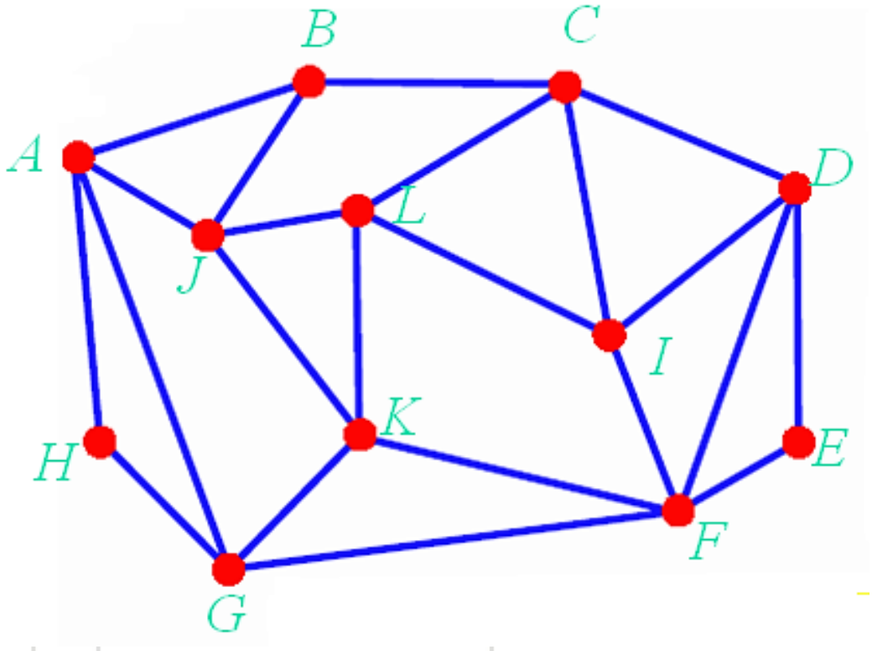
网格性质

数据结构

网格处理

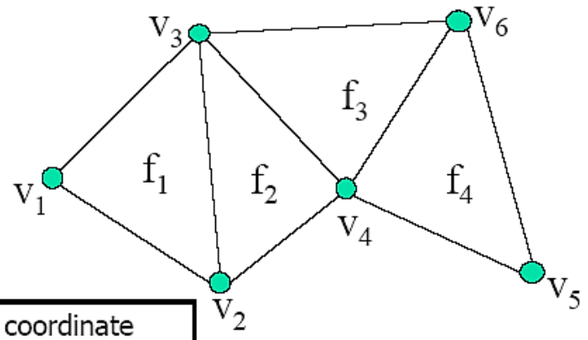
网格数据结构需要解决的问题

- 表示几何与连接信息并对其进行查询及修改
  - 几何信息主要为顶点与其位置，法向量等的对应关系
    - 相对简单
  - 查询连接信息
    - 面f的顶点？顶点v，u是否相连？
    - 与顶点v相邻的顶点？与顶点v相邻的面？
  - 修改连接信息
    - 增加或删除顶点/面
    - Mesh refinement/simplification
    - Vertex split, edge collapse
  - 简单方法（参考此前obj文件格式）
    - 顶点列表（几何）+面列表（连接）



网格数据结构的简单表示方法

- 顶点列表（几何）+面列表（连接）
- 性能分析
  - 查询面f对应的顶点：检查面列表：$O(1)$时间内完成
  - 查询顶点顶点v，u是否相连：需要对整个面列表扫描！
- 优点
  - 方便
  - 存储简单
  - 可用于表示非流形
- 缺点
  - 缺少顶点到边/面的联系
  - 无法高效完成各种查询/编辑



vertex	coordinate
v_1	(x_1,y_1,z_1)
v_2	(x_2,y_2,z_2)
v_3	(x_3,y_3,z_3)
v_4	(x_4,y_4,z_4)
v_5	(x_5,y_5,z_5)
v_6	(x_6,y_6,z_6)

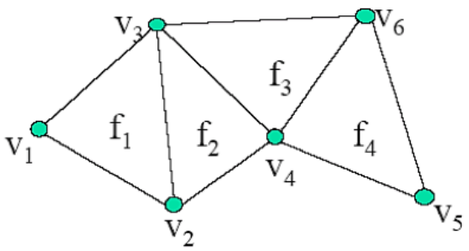
face	vertices (ccw)
f_1	(v_1, v_2, v_3)
f_2	(v_2, v_4, v_3)
f_3	(v_3, v_4, v_6)
f_4	(v_4, v_5, v_6)

网格数据结构的简单表示方法

- 顶点列表（几何）+面列表（连接）+邻接矩阵
- 性能分析
  - 查询面f对应的顶点：检查面列表：$O(1)$时间内完成
  - 查询顶点v对应的面：需要对整个面列表扫描！
- 优点
  - 方便、可用于表示非流形
  - 耗费存储空间（$O(|V|^2)$）
- 缺点
  - 缺少顶点到面的联系
  - 无法高效完成各种查询/编辑

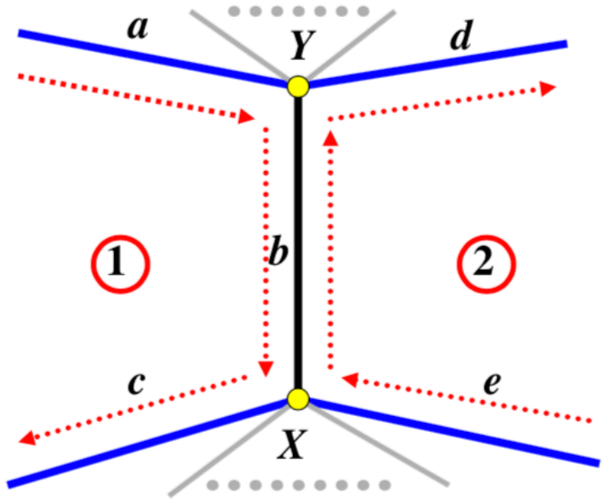
vertex	coordinate
v_1	(x_1,y_1,z_1)
v_2	(x_2,y_2,z_2)
v_3	(x_3,y_3,z_3)
v_4	(x_4,y_4,z_4)
v_5	(x_5,y_5,z_5)
v_6	(x_6,y_6,z_6)

face	vertices (ccw)
f_1	(v_1, v_2, v_3)
f_2	(v_2, v_4, v_3)
f_3	(v_3, v_4, v_6)
f_4	(v_4, v_5, v_6)

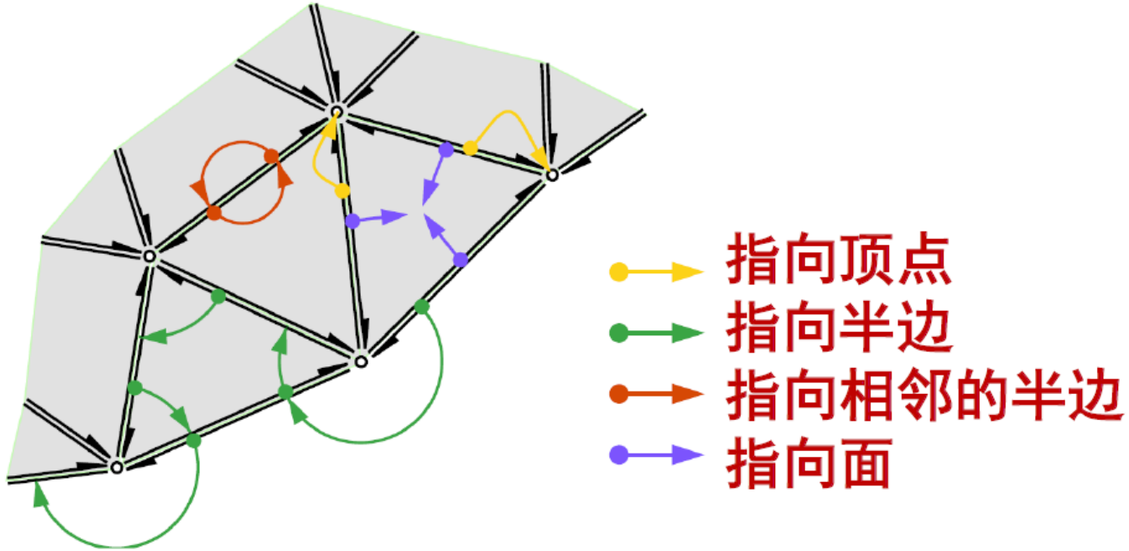


	v_1	v_2	v_3	v_4	v_5	v_6
v_1		1	1			
v_2	1		1	1		
v_3	1	1		1		1
v_4		1	1		1	1
v_5				1		1
v_6			1	1	1	

- Half-Edge数据结构
  - 基于边（edge-based）的数据结构
    - Half-edge及其早期版本winged-edge都属于此类
    - 此外还有基于点（vertex-based）及基于面（face-based）的数据结构
  - 适用于可定向的二维流形



winged-edge

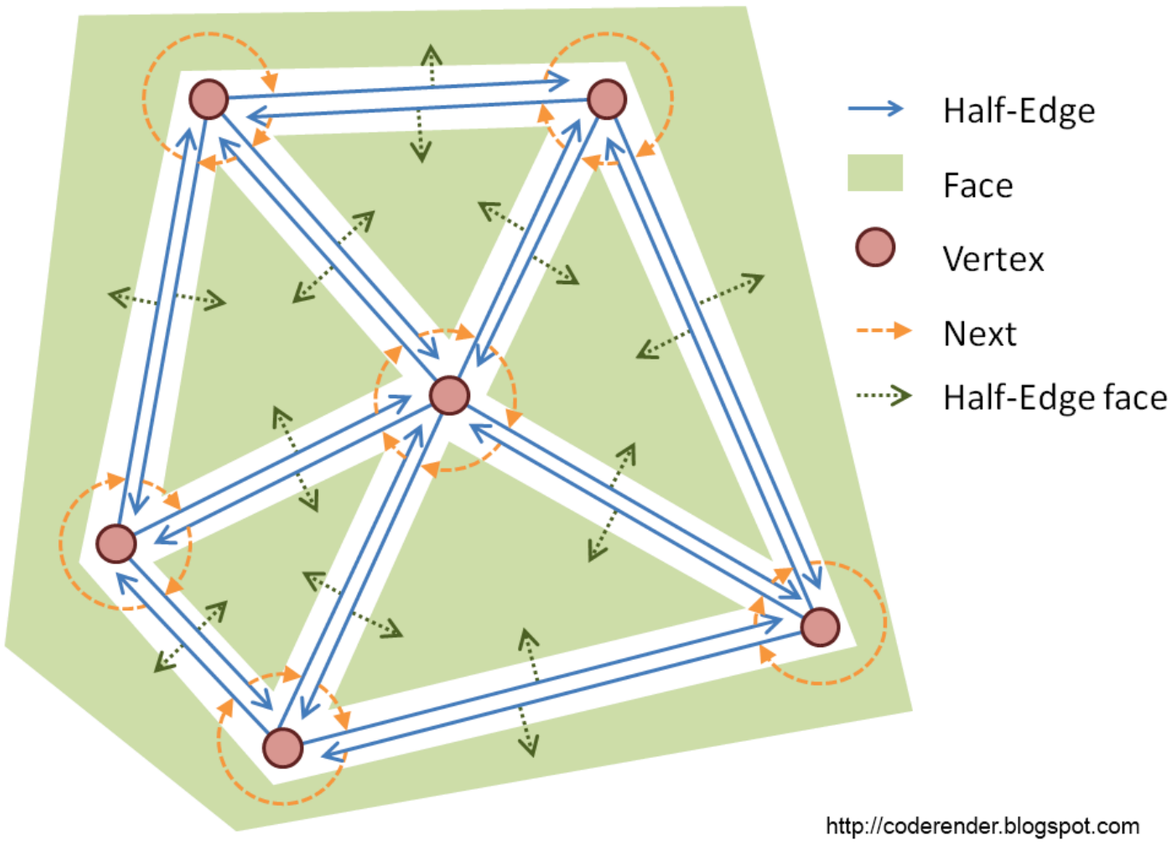


half-edge

Half-Edge数据结构

– 每条边对应两个half-edges

```
struct HE_edge {
    //half-edge的起始顶点
    HE_vert* vert;
    //同侧相邻面
    HE_face* face;
    //另一侧的half-edge
    HE_edge* pair;
    //face的下一条half-edge
    HE_edge* next;
    //face的上一条half-edge
    HE_edge* prev;
};
```



http://coderender.blogspot.com

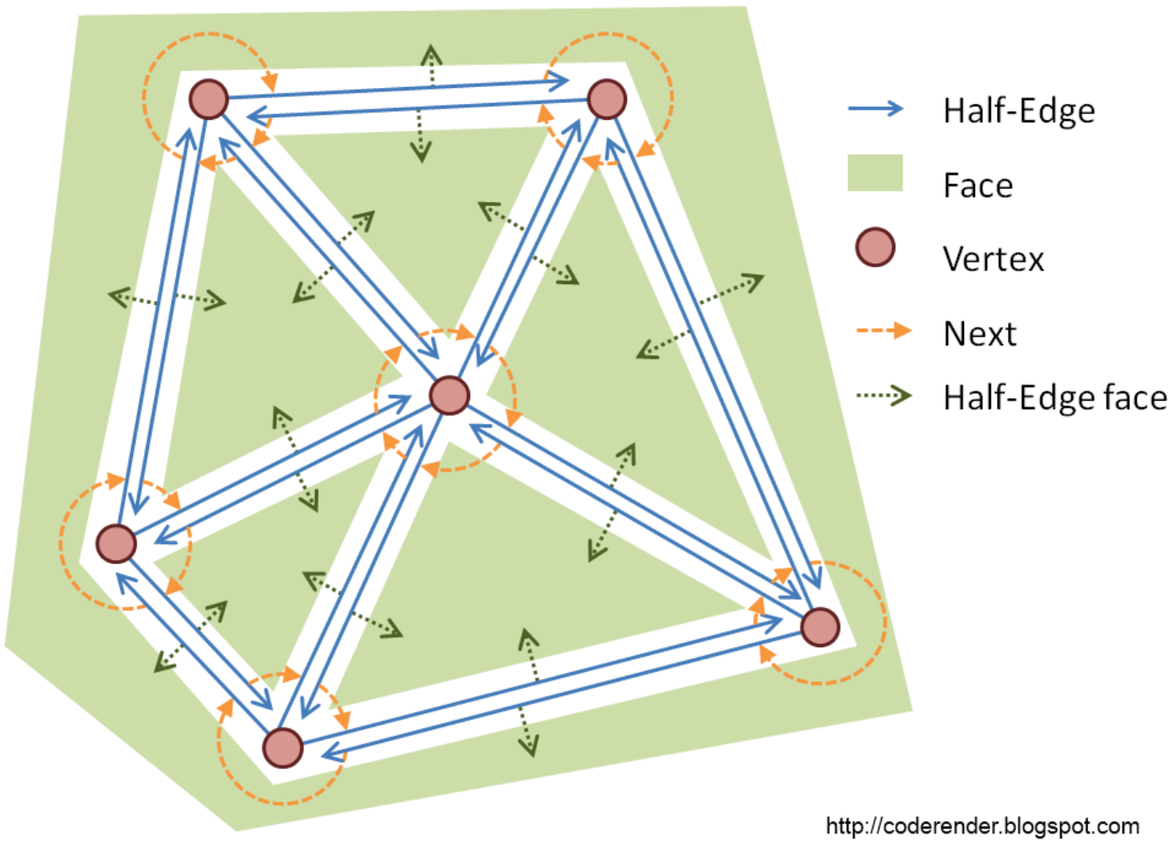
Half-Edge数据结构

– 每个面只需要其第一条half-edge的指针

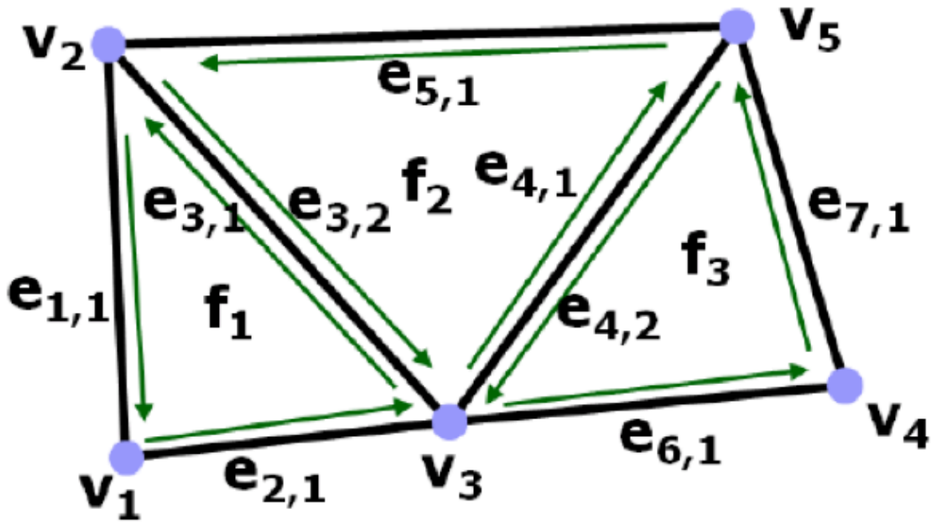
```
struct HE_face {
    //指向其第一条half-edge
    HE_edge* edge;
};
```

– 每个顶点需要其三维坐标及对应的一条half-edge

```
struct HE_vert {
    float x;
    float y;
    float z;
    //由该顶点出发的其中一条half-edge
    HE_edge* edge;
};
```



Half-Edge数据结构举例



顶点	坐标	以此为起点的半边
v_1	(x_1,y_1,z_1)	$e_{2,1}$
v_2	(x_2,y_2,z_2)	$e_{1,1}$
v_3	(x_3,y_3,z_3)	$e_{4,1}$
v_4	(x_4,y_4,z_4)	$e_{7,1}$
v_5	(x_5,y_5,z_5)	$e_{5,1}$

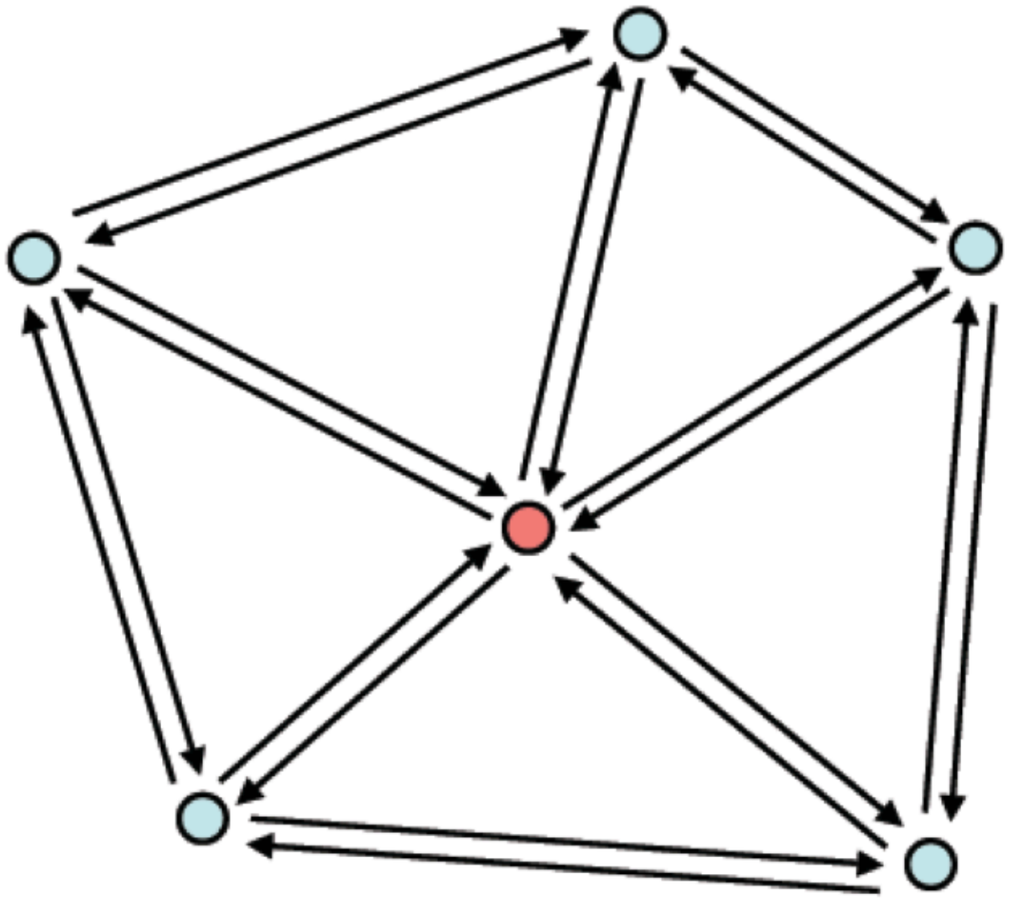
半边	起点	相邻半边	面	下条半边	前条半边
$e_{3,1}$	v_3	$e_{3,2}$	f_1	$e_{1,1}$	$e_{2,1}$
$e_{3,2}$	v_2	$e_{3,1}$	f_2	$e_{4,1}$	$e_{5,1}$
$e_{4,1}$	v_3	$e_{4,2}$	f_2	$e_{5,1}$	$e_{3,2}$
$e_{4,2}$	v_5	$e_{4,1}$	f_3	$e_{6,1}$	$e_{7,1}$

面	半边
f_1	$e_{1,1}$
f_2	$e_{3,2}$
f_3	$e_{4,2}$

Half-Edge数据结构查询举例

– 遍历顶点v的所有相邻顶点

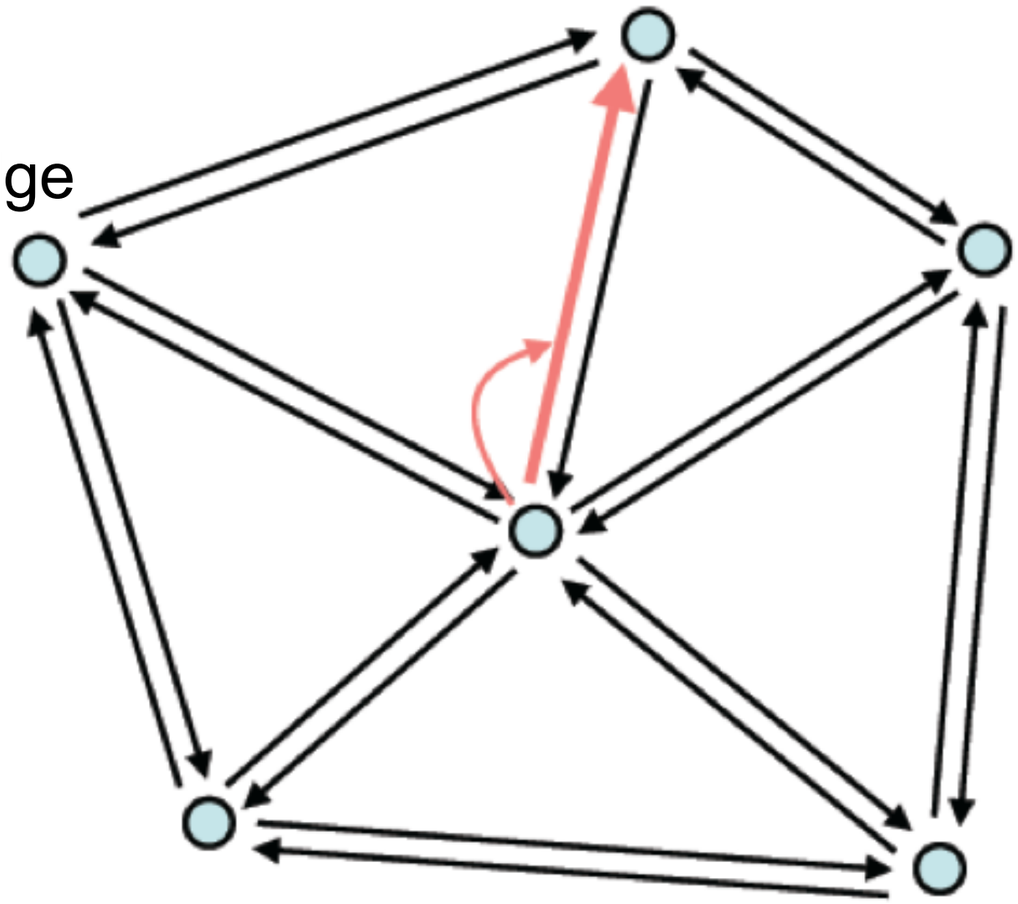
- 1. 从顶点v出发



Half-Edge数据结构查询举例

– 遍历顶点v的所有相邻顶点

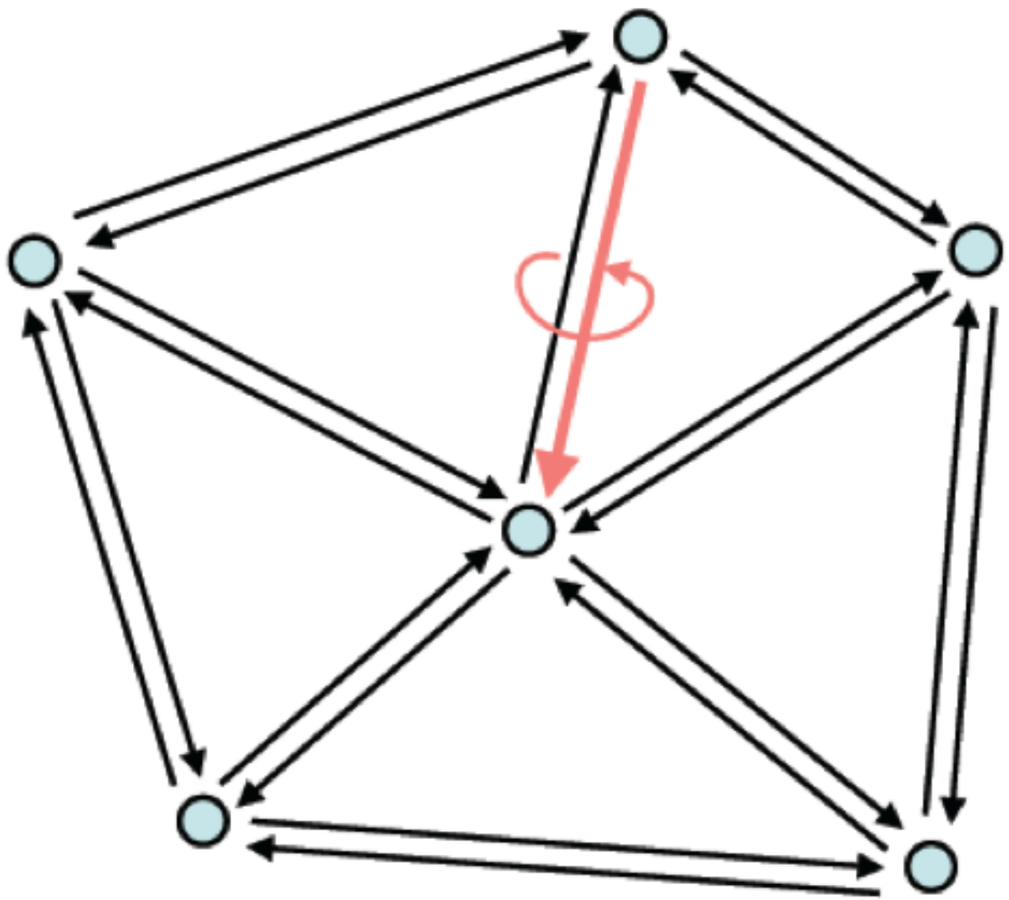
- 1. 从顶点v出发
- 2. 找到由v出发的任意一条half-edge



Half-Edge数据结构查询举例

– 遍历顶点v的所有相邻顶点

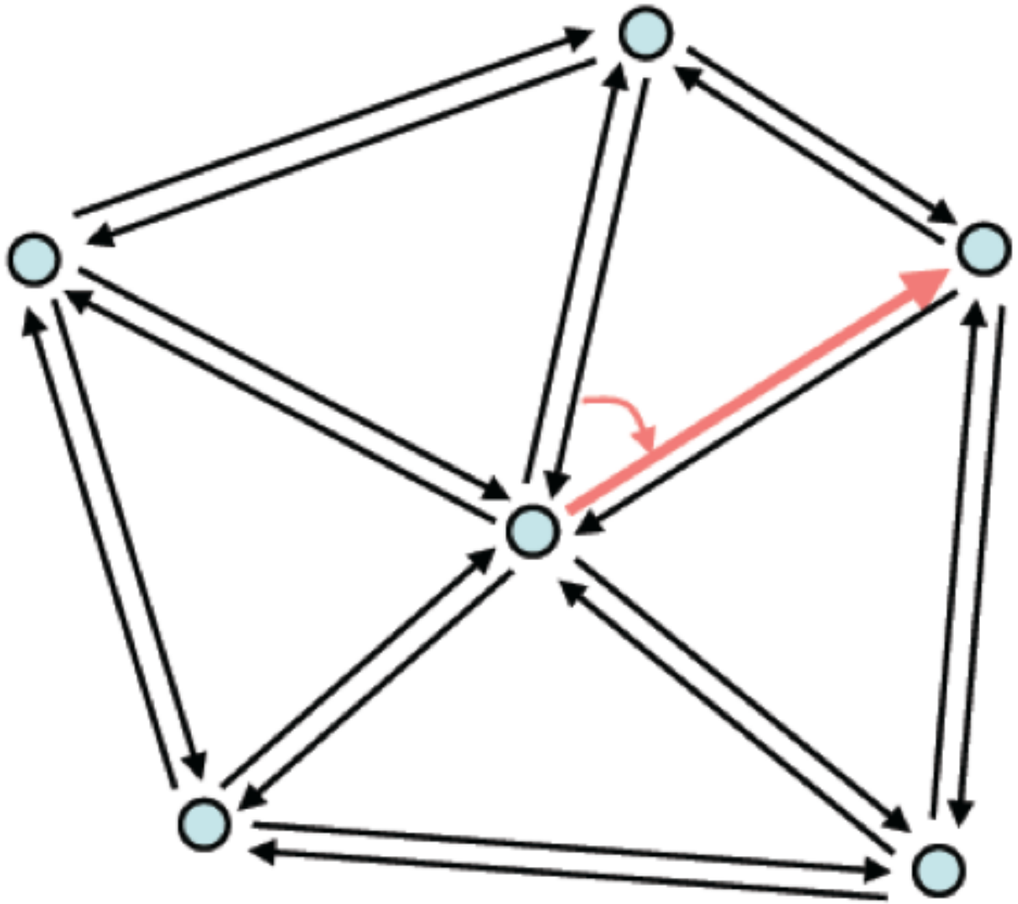
- 1. 从顶点v出发
- 2. 由v出发的任意一条half-edge
- 3. 对面的half-edge



Half-Edge数据结构查询举例

– 遍历顶点v的所有相邻顶点

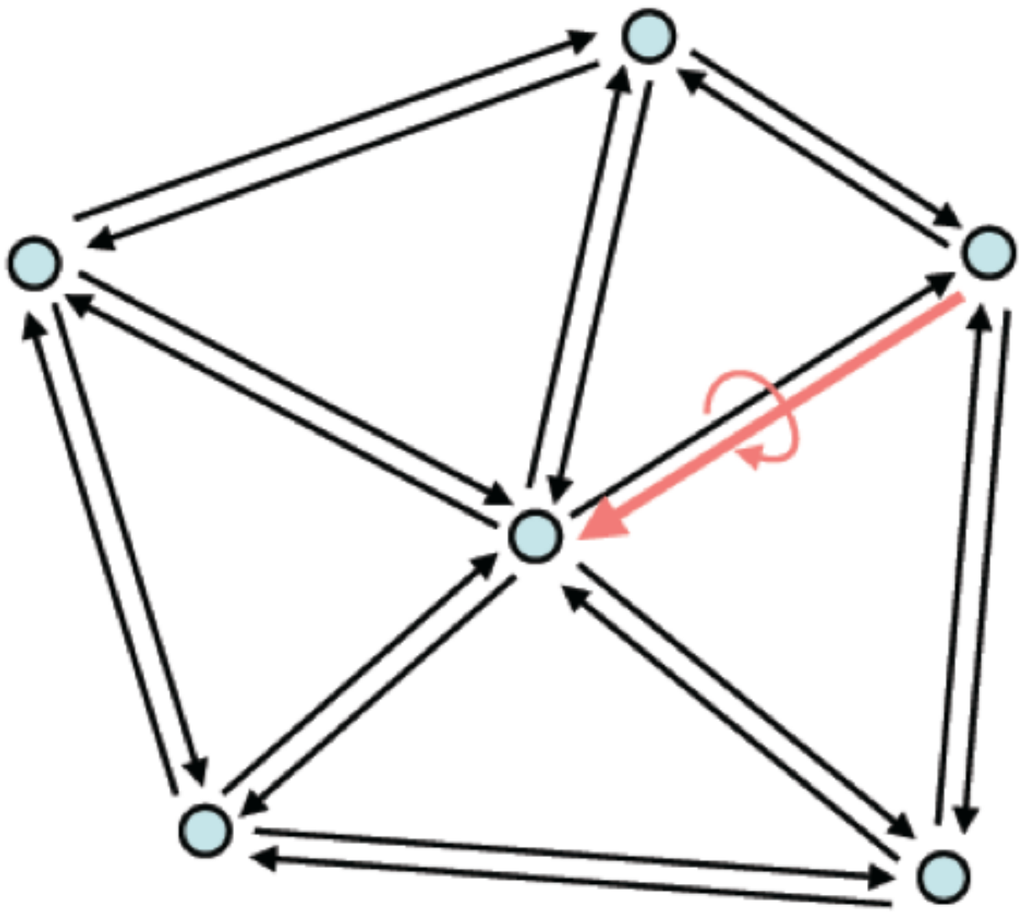
- 1. 从顶点v出发
- 2. 由v出发的任意一条half-edge
- 3. 对面的half-edge
- 4. 下一条half-edge



Half-Edge数据结构查询举例

– 遍历顶点v的所有相邻顶点

- 1. 从顶点v出发
- 2. 由v出发的任意一条half-edge
- 3. 对面的half-edge
- 4. 下一条half-edge
- 5. 对面的half-edge
- …

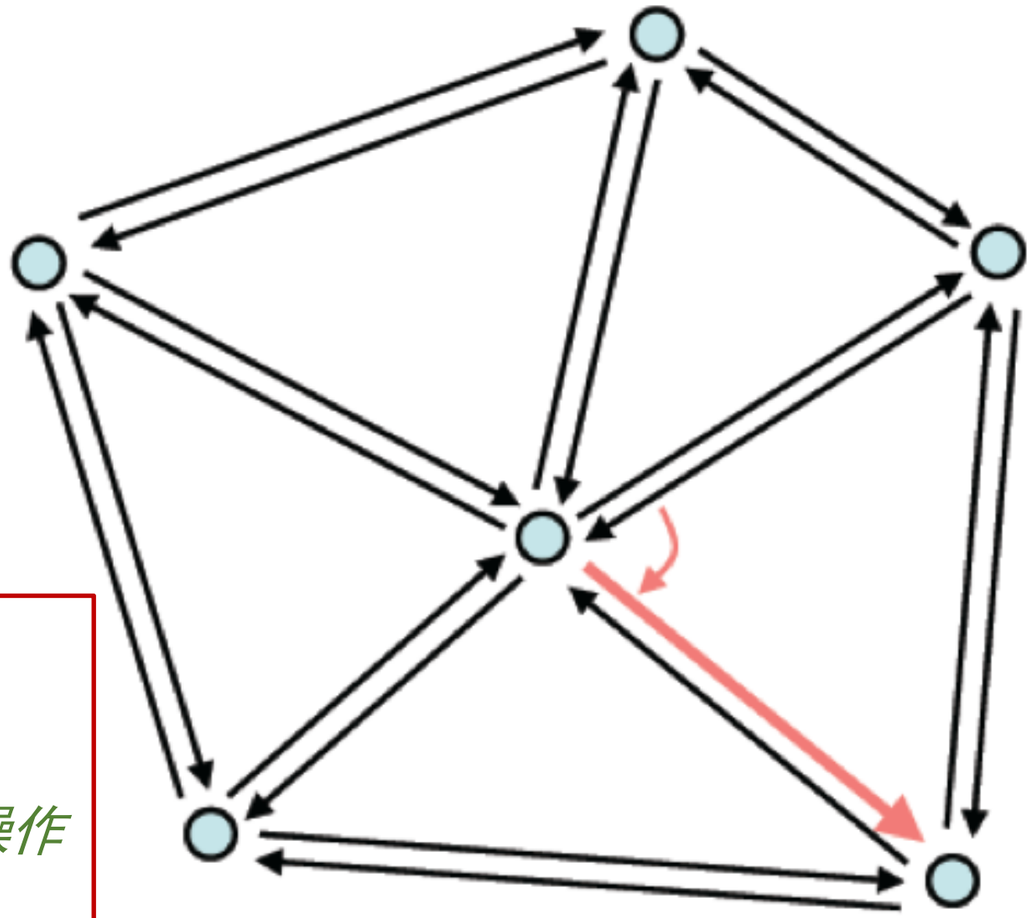


Half-Edge数据结构查询举例

– 遍历顶点v的所有相邻顶点

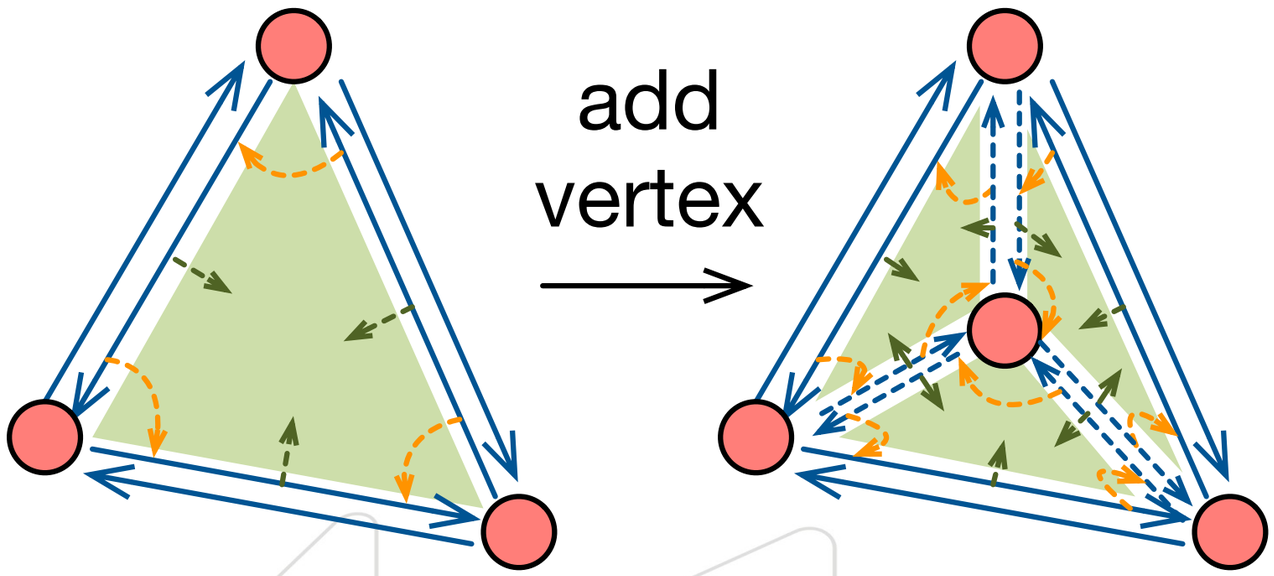
- 1. 从顶点v出发
- 2. 由v出发的任意一条half-edge
- 3. 对面的half-edge
- 4. 下一条half-edge
- 5. 对面的half-edge
- 6. 下一条half-edge
- …

```
e = v->edge;
do {
    ...
    //对 e->pair->vert进行操作
    e = e->pair->next;
} while (e!=v->edge)
```



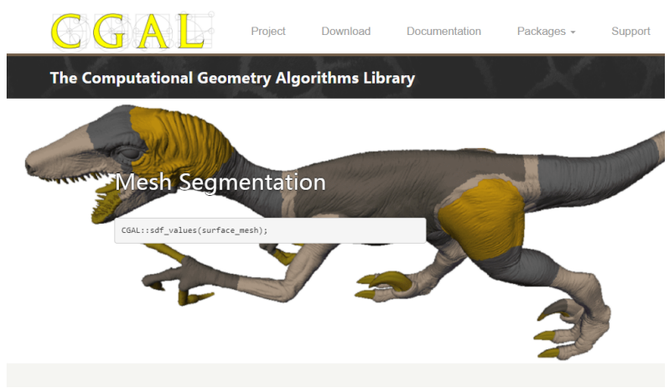
Half-Edge数据结构编辑网格基本操作

- 标记边界（mark mesh boundary）
- 创建邻接边（create edge adjacency）
- 增加顶点（add vertex）
- 增加边（add edge）
- 增加面（add face）
- 删除顶点（delete vertex）
- 删除边（delete edge）
- 删除面（delete face）



Half-Edge数据结构

- 优点：查询时间$O(1)$，操作时间$O(1)$
- 缺点：包含一定冗余信息，只能应用于二维流形
- 使用half-edge的库
  - CGAL: http://www.cgal.org/
    - the Computational Geometry Algorithms Library
    - Free for non-commercial use
  - OpenMesh: http://www.openmesh.org/
    - Mesh processing
    - Free, LGPL licence



基于网格的几何体表现形式

优点

- 简单，易于描述
- 与渲染软件/硬件所需数据形式一致
- 普遍使用
  - 大多数模拟、分析工具的输入
  - 大多数数据获取工具的输出

缺点

- 为几何体的近似表达
- 使用传统方法难以编辑
- 没有解析形式，几何属性可能难以计算
- 当几何体拓扑结构复杂细节丰富时，建模/编辑/渲染/存储开销高

网格及其处理

基本概念

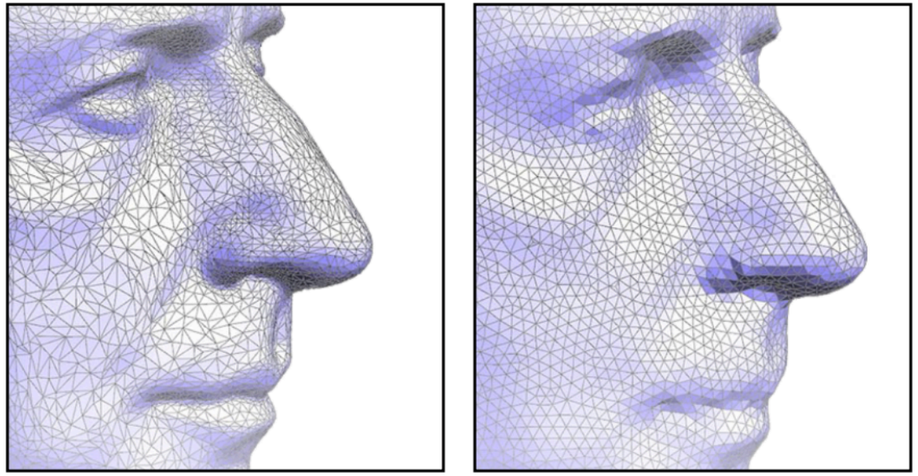
网格性质

数据结构

网格处理

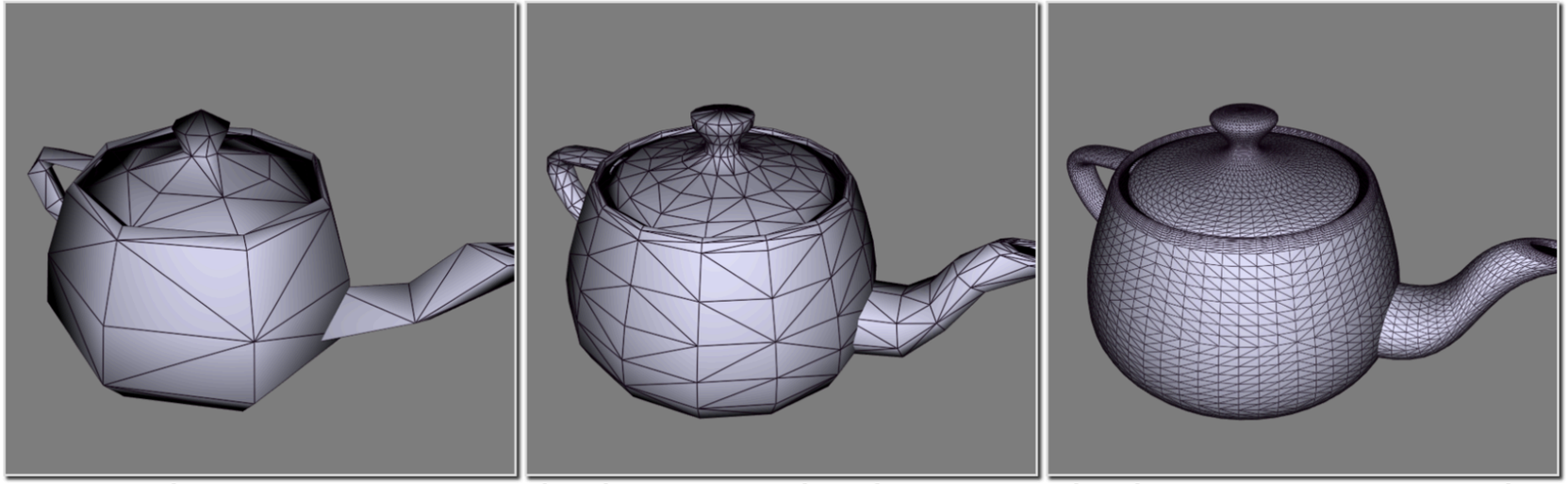
网格处理

- 光滑曲面可以有不同的网格近似
  - 网格细分（subdivision）：上采样
  - 网格简化（simplification）：下采样
  - 网格正则化



regularization

subdivision

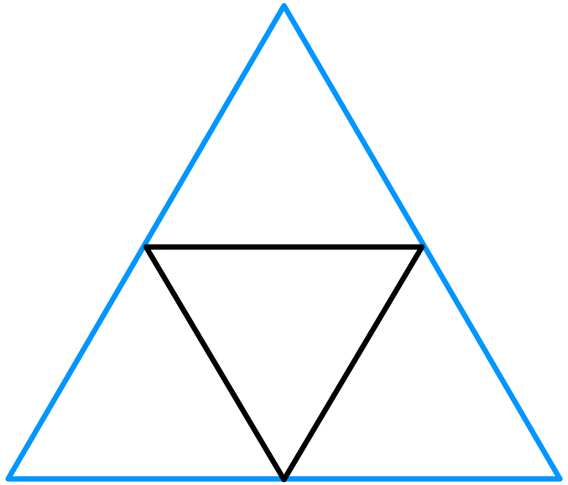


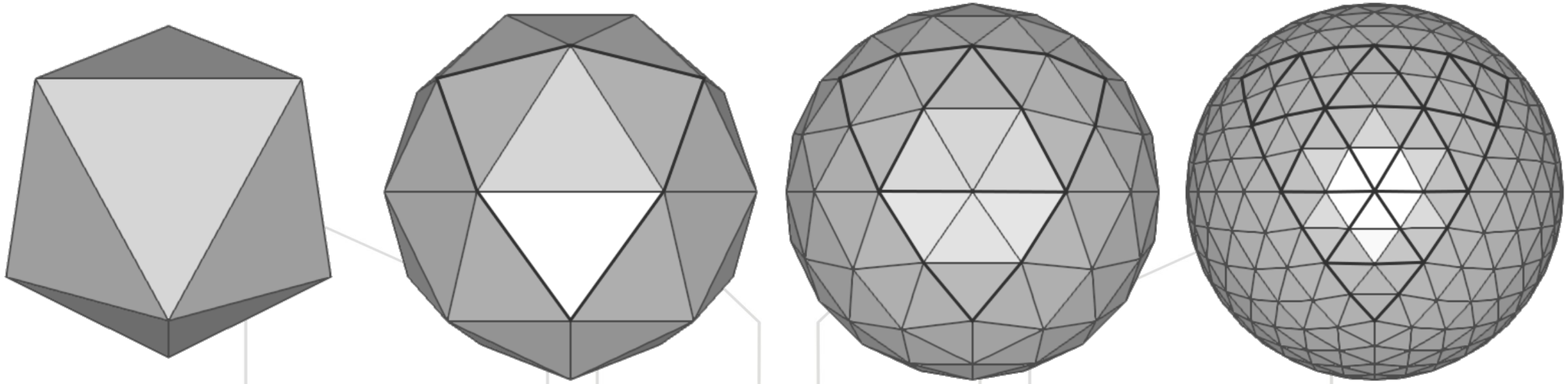
simplification

曲面细分

曲面细分通过增加顶点从较粗糙的网格中产生精细网格

- 拓扑信息改变
  - 新的顶点之间如何连接？
- 几何信息改变
  - 如何决定新的顶点的位置？



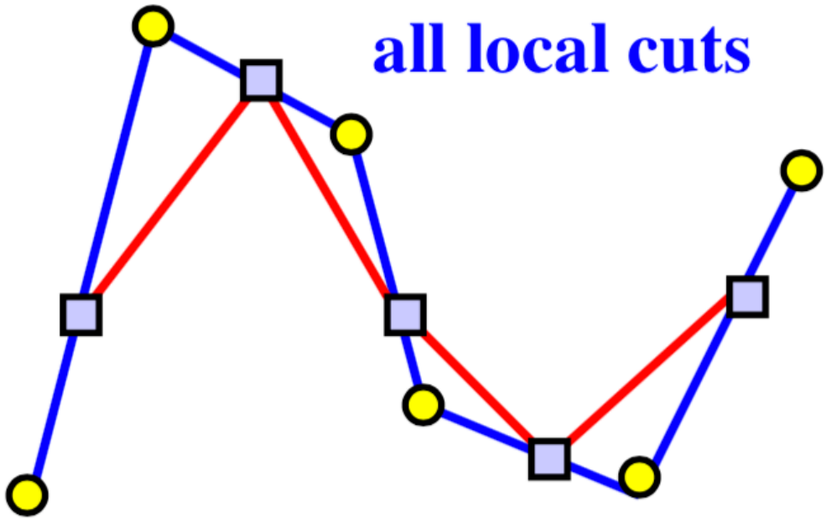


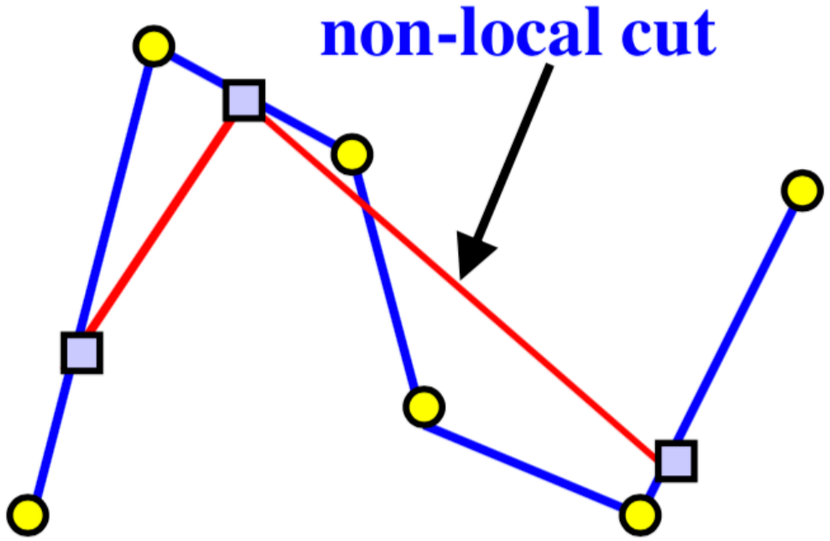
曲面细分

曲线的corner cutting

– 为了使一条折线更平滑，可以“切除”其不平滑的“角”
  - 在两条边上各取一个点，并将其连接

– Corner cutting可以为局部（local）或非局部（non-local）
  - 局部：每次去掉一个顶点，并增加两个新的顶点
  - 非局部：否则为非局部

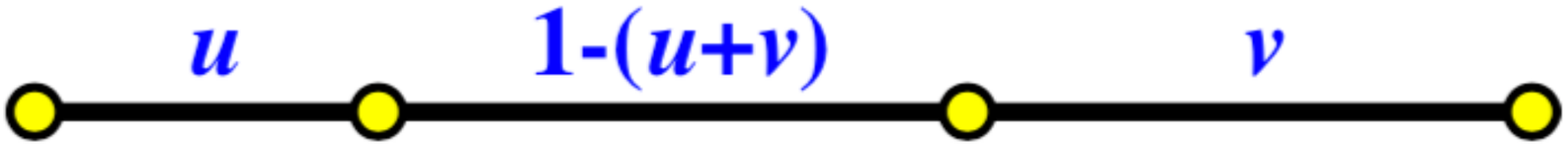




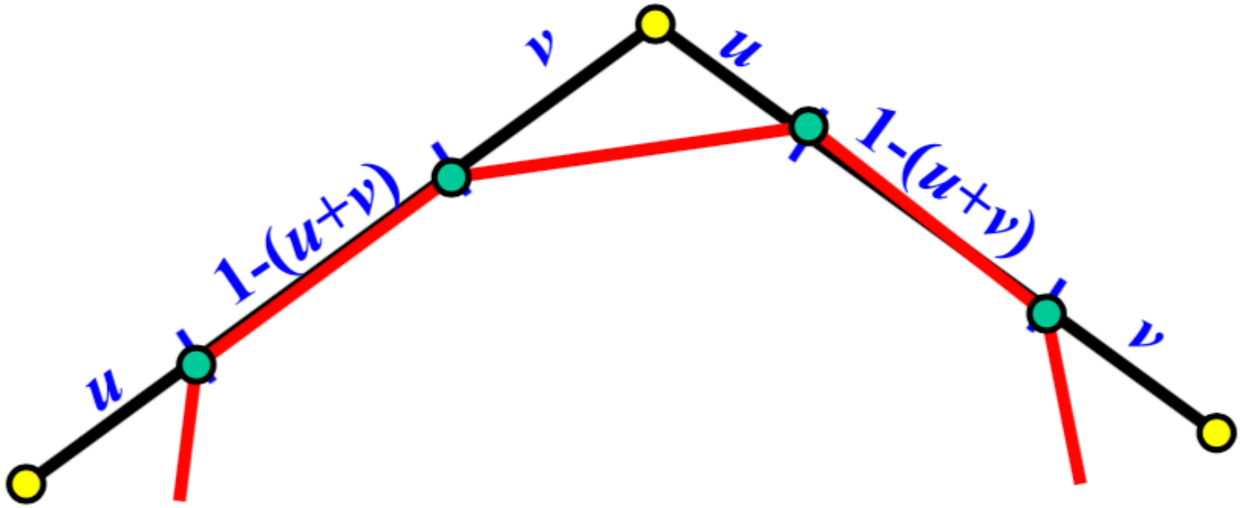
曲面细分

曲线的corner cutting

– 在每条边上，选择两个数字$u \geq 0$与$v \geq 0$，且$u + v \leq 1$，从而将按$u: 1 - (u + v): v$的比例将曲线划分为三段



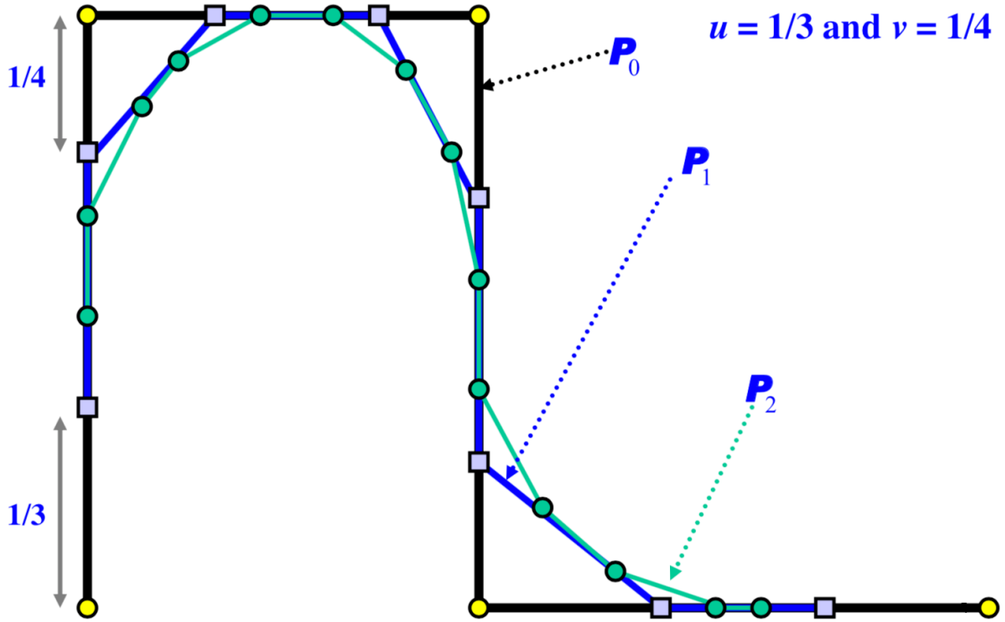
– 连接两段间的新的顶点，从而将原有折线上的“角”磨平

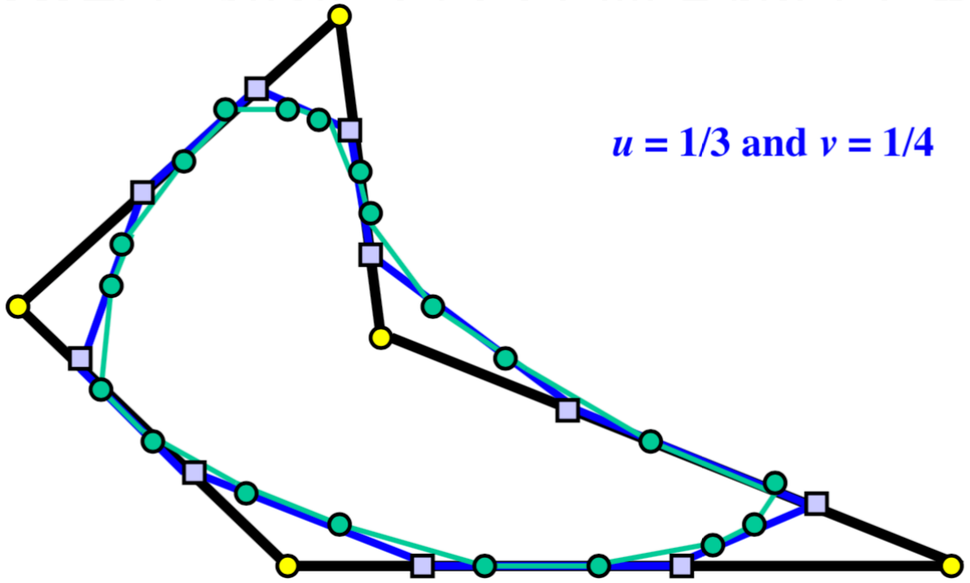


曲面细分

曲线的corner cutting

- 设原始折线为P_0，根据上述规则进行一次处理后得到折线P_1
- 对P_1进行处理后得到 P_2，以此类推
- $P_\infty = \lim_{i\to\infty} P_i$
  - 不同边上的u与 v不需要一致　每次处理时的u与 v也不需要保持不变

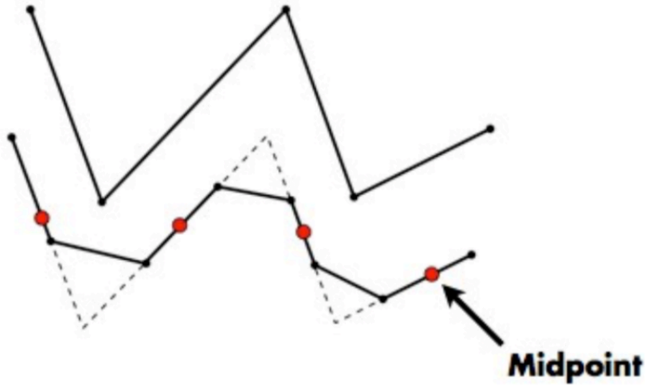


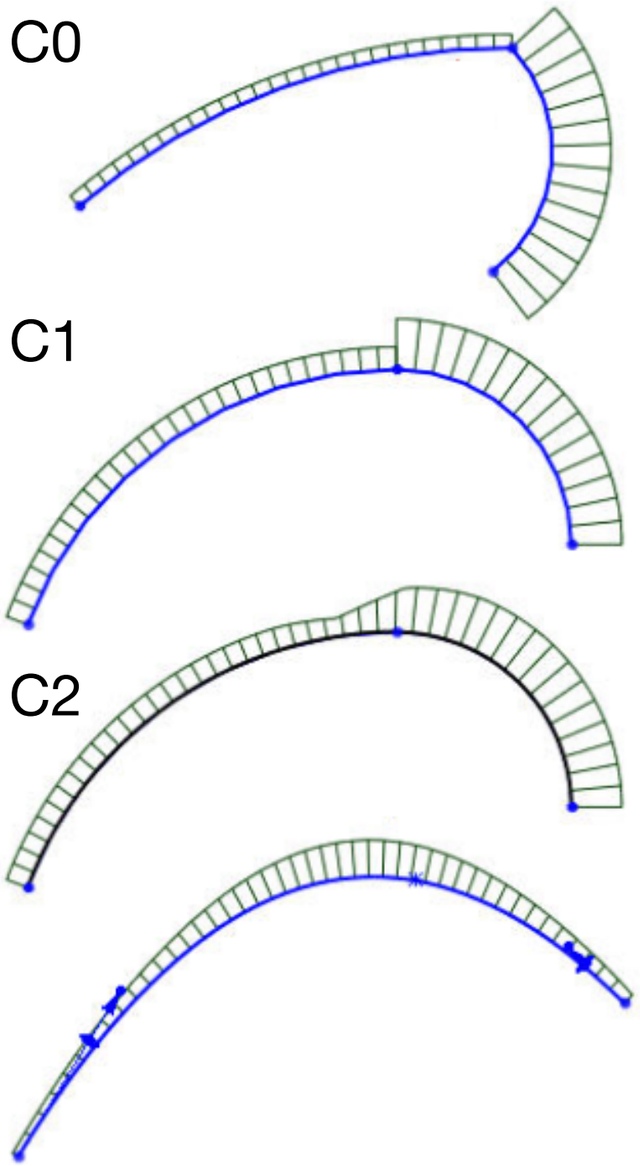


曲面细分

曲线的corner cutting

- $P_\infty = \lim_{i\to\infty} P_i$
  - 当$u \geq 0, v \geq 0, 2u + v \leq 1, u + 2v \leq 1$时，$P_\infty$为$C^1$连续的曲线
    - 由Gregory and Qu，de Boor，及Paluszny, Prautzsch, and Schäfer分别证明
  - Chaikin算法（1974年）为$u = v = 1/4$的特例
    - P_∞为 degree $= 2$的B样条曲线





曲面细分

- 曲面细分的一个著名应用为Pixar’s的短片Geri’s Game
  - 1997年获得奥斯卡最佳动画短片奖

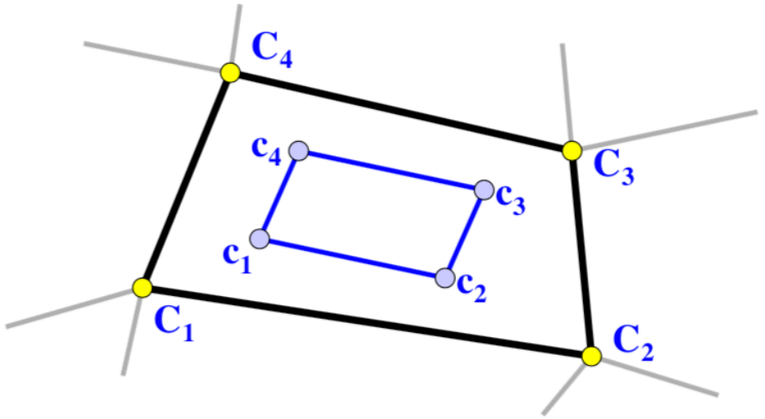
曲面细分

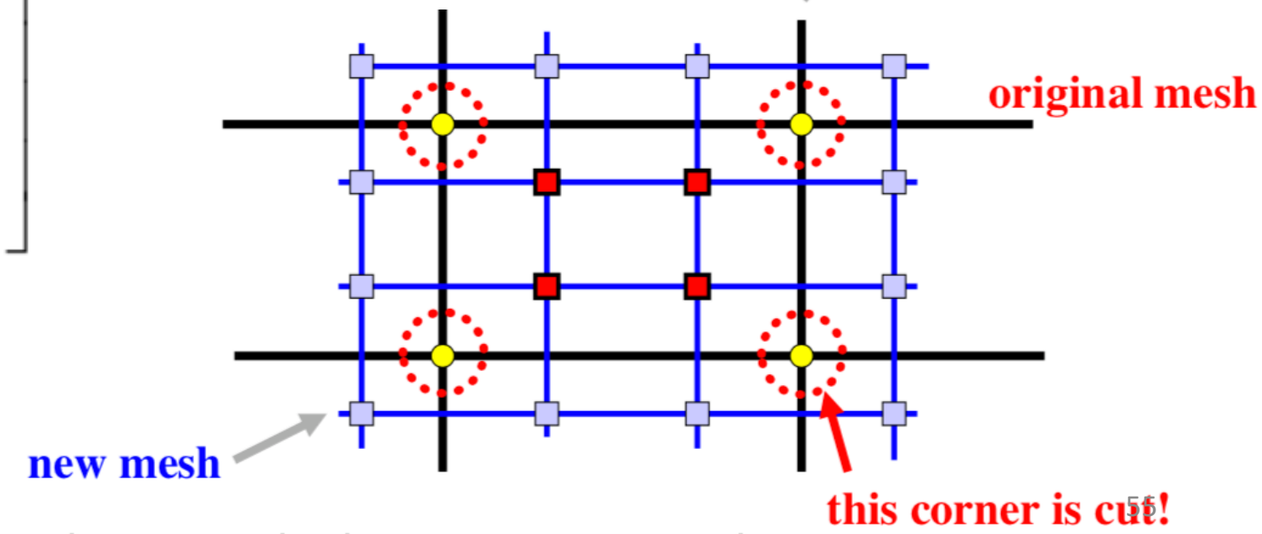
推广曲线corner cutting，可得曲面细分的模型

– 假设网格上一个四边形的顶点为C_1, C_2, C_3, C_4，新增的四个新顶点为c_1, c_2, c_3, c_4

- $c_i = \frac{3}{16}C_{i-1} + \frac{9}{16}C_i + \frac{3}{16}C_{i+1} + \frac{1}{16}C_{i+2}$
- 将这一过程写成矩阵，则

$$\mathbf{Q} = \begin{bmatrix} 9/16 & 3/16 & 1/16 & 3/16 \\ 3/16 & 9/16 & 3/16 & 1/16 \\ 1/16 & 3/16 & 9/16 & 3/16 \\ 3/16 & 1/16 & 3/16 & 9/16 \end{bmatrix} \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{c}_3 \\ \mathbf{c}_4 \end{bmatrix} = \mathbf{Q} \cdot \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \\ \mathbf{C}_3 \\ \mathbf{C}_4 \end{bmatrix}$$





Doo-Sabin subdivision

– 1978年，由Doo与Sabin提出，根据以下方式计算c_i

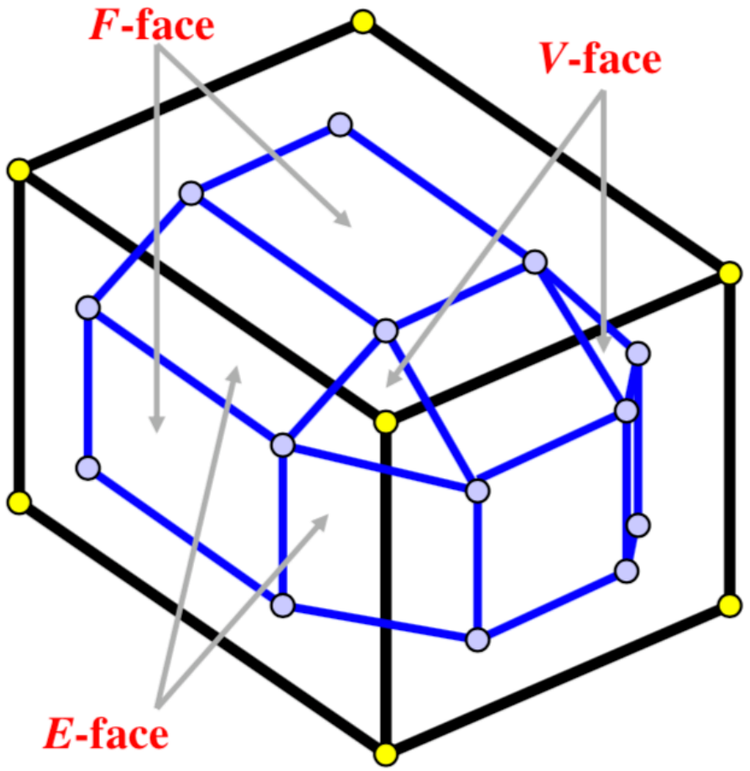
- $c_i = \sum_{j=1}^{n} \alpha_{ij} C_j$

– 其中α_{ij}的计算方式如下

- $$\alpha_{ij} = \begin{cases} \frac{n+5}{4n} & i = j \\ \frac{1}{4n}[3 + 2\cos(\frac{2\pi(i-j)}{n})] & i \neq j \end{cases}$$

– 细分后的曲面具有三种面

- F-face：连接的顶点对应原始网格中同一个面
- E-face：连接的顶点对应原始网格中同一条边
- V-face：连接的顶点对应原始网格中同一个顶点

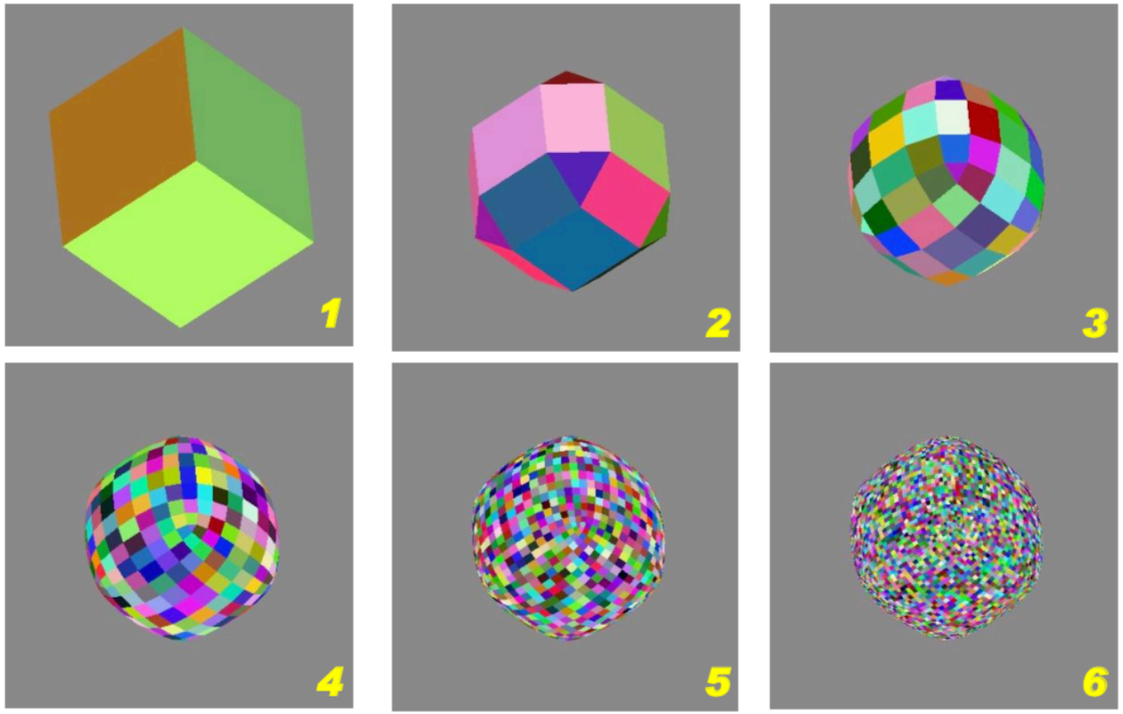


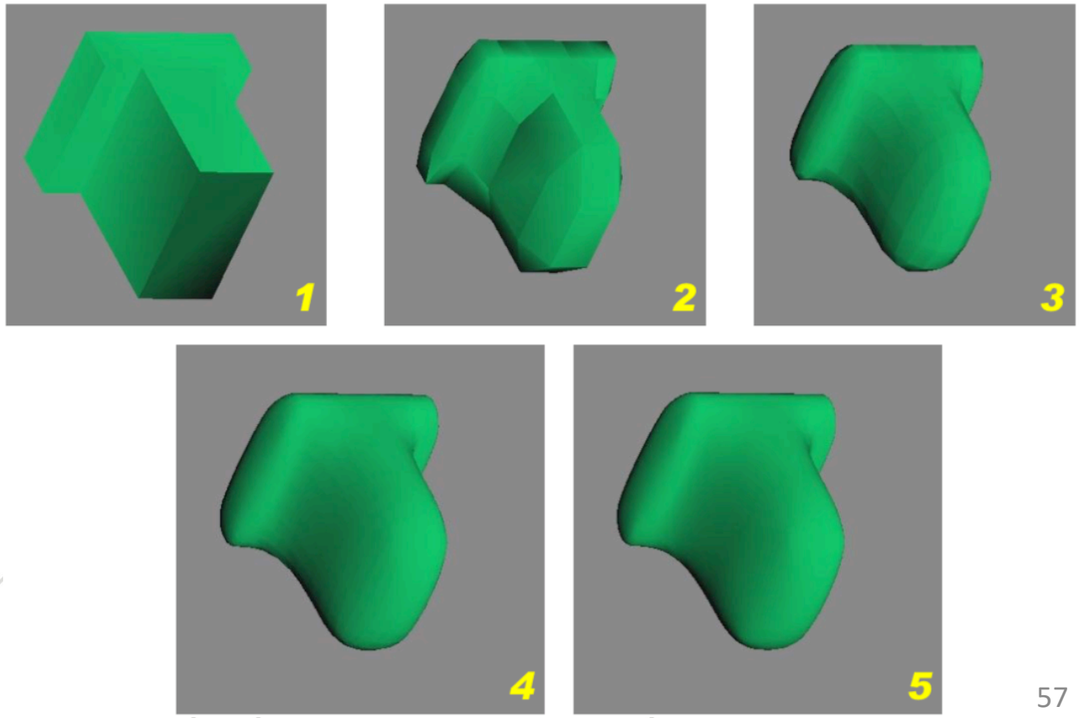
曲面细分

Doo-Sabin subdivision

– 对四边形网格进行细分后，大多数面仍然为四边形
  - 除V-faces外

– 产生曲面的极限为degree=(2,2)的B样条曲面（G^1连续）





曲面细分

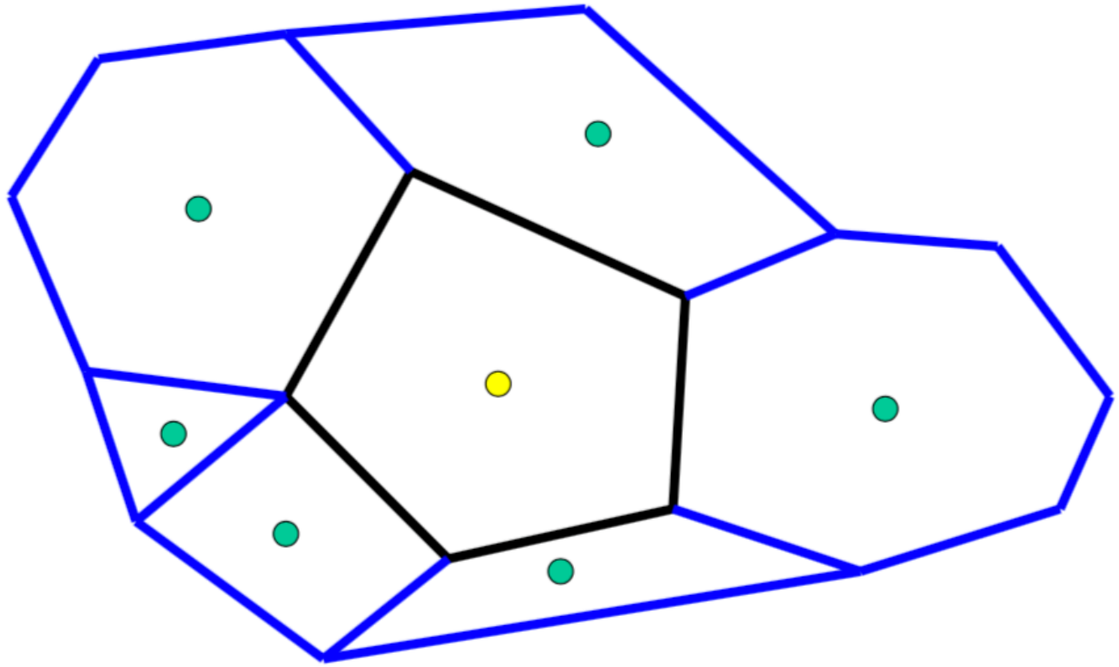
Catmull-Clark algorithm

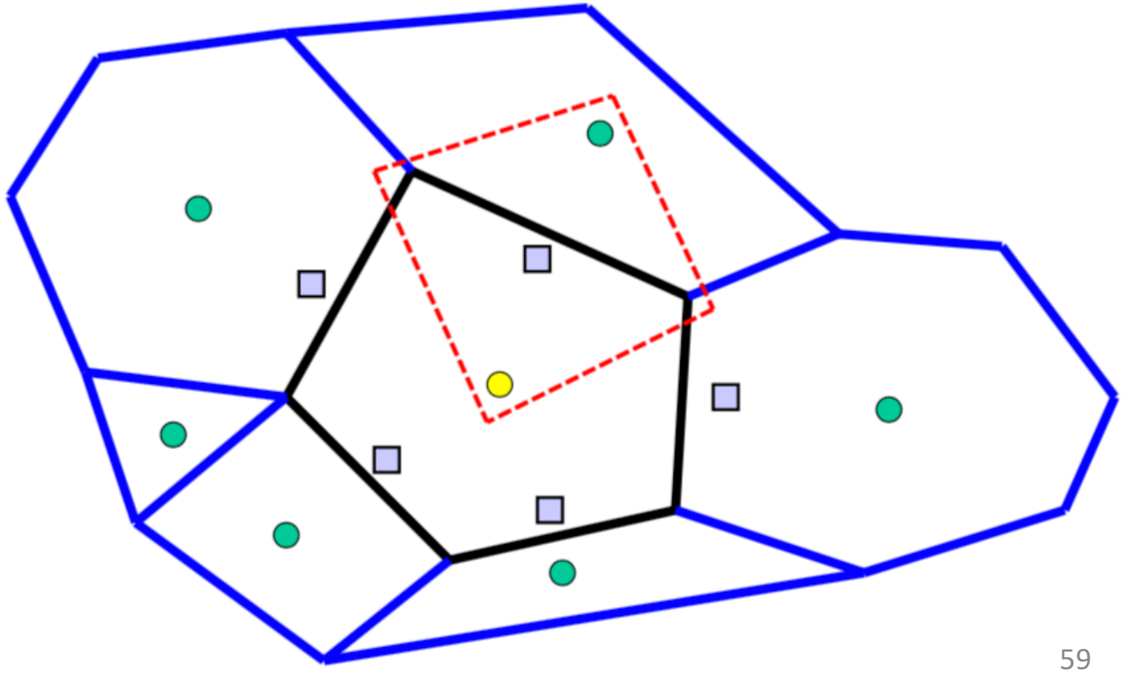
- 1978年由Catmull于Clark提出
  - 与Doo-Sabin算法于同年提出，甚至出现在同一本期刊上（Computer-Aided Design)
- Catmull-Clark算法的思路
  - 对于每个面，计算一个face point；对每条边计算一个edge point；对每个顶点计算一个vertex point
  - 连接以上新的顶点构建网格
  - 与Doo-Sabin算法有很大的区别

曲面细分

Catmull-Clark algorithm

- 每个面对应的face point为该面的质心（顶点的平均值）
- 每条边对应的edge point为该边的两个顶点及相邻面的face point的平均值



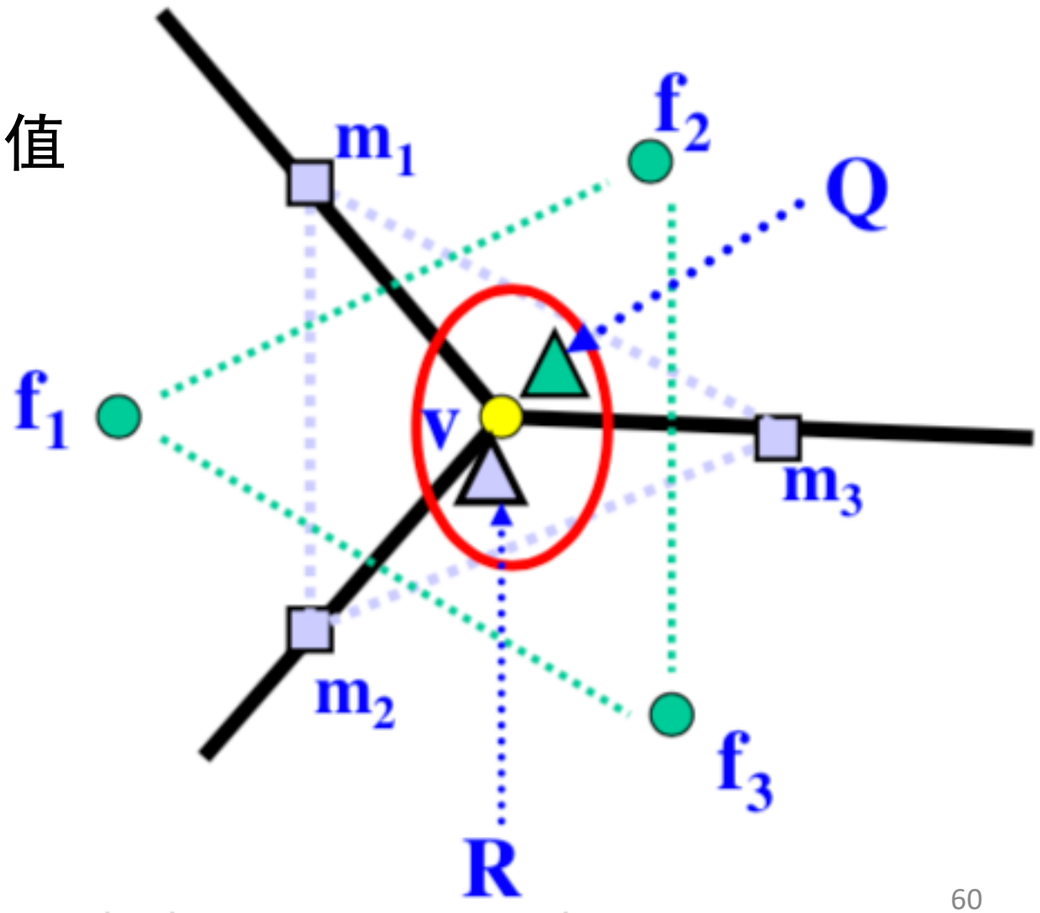


曲面细分

Catmull-Clark algorithm

– 每个顶点对应的vertex point由以下式子给出

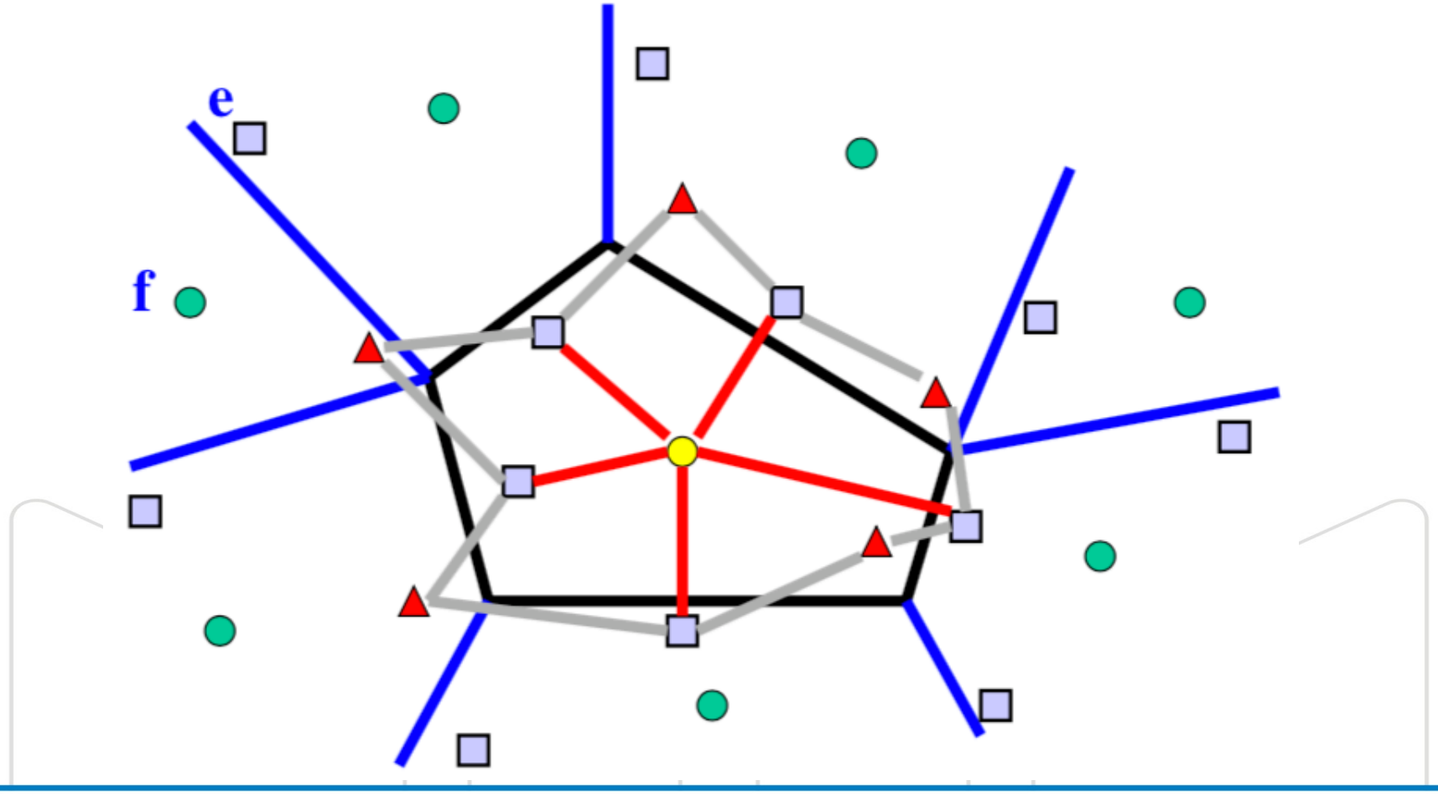
- $v = \frac{1}{n}Q + \frac{2}{n}R + \frac{n-3}{n}v$
- 其中Q为相邻面对应face points的平均值
- R为相临边中点的平均值
- v为原始顶点
- n为v的相临边数目



Catmull-Clark algorithm

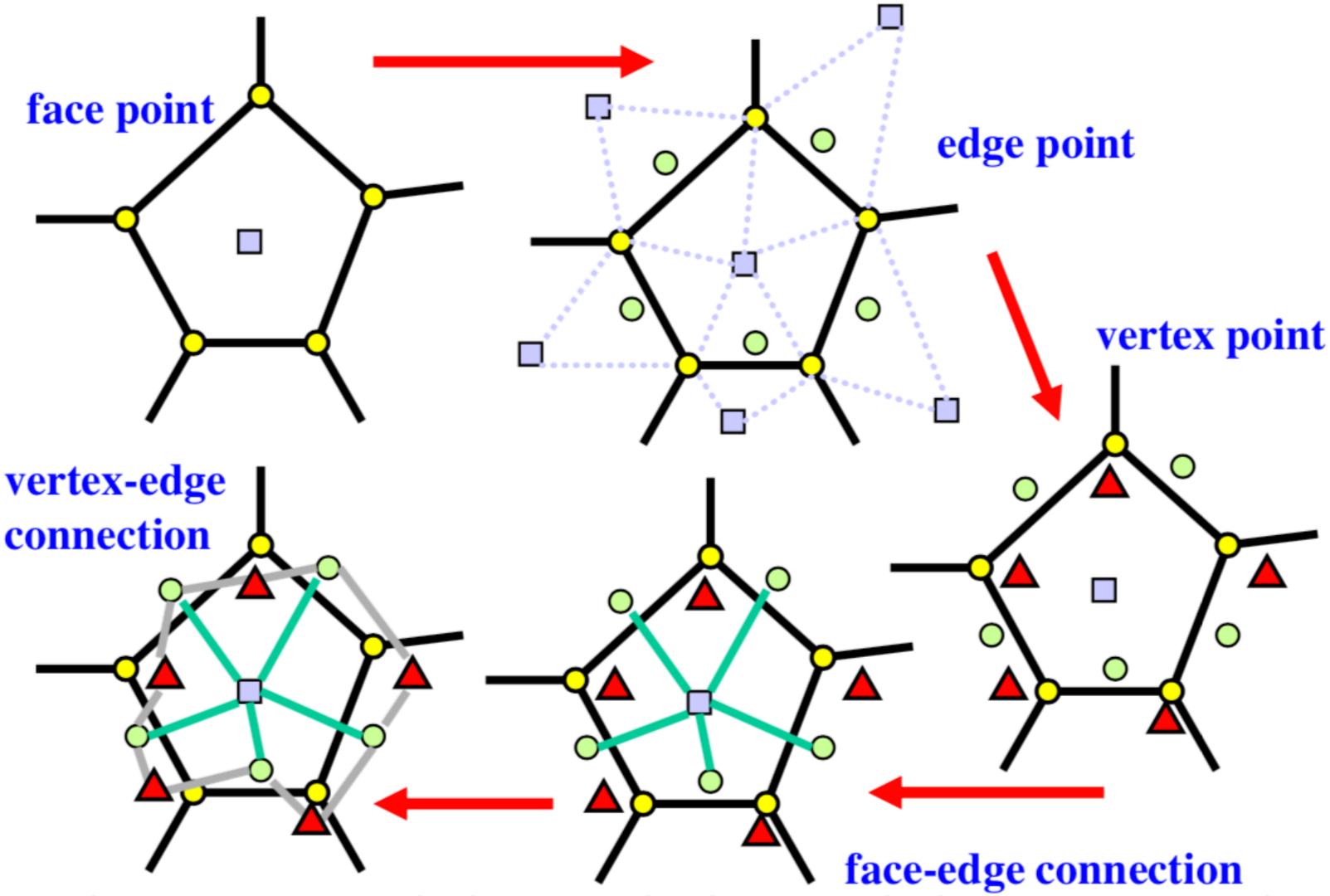
– 顶点的连接

- 对每个面，将其face point及其所有边的edge points相连，并将其edge points与对应的vertex points相连



曲面细分

- Catmull-Clark algorithm



曲面细分

Catmull-Clark algorithm

- 在第一次细分后，所有面都为四边形
- 假设每个面对应边为e_1, e_2, e_3, e_4，对应顶点为v_1, v_2, v_3, v_4，Catmull-Clark算法可表示为如下矩阵

$$\begin{bmatrix} \mathbf{v}' \\ \mathbf{e}_1' \\ \mathbf{e}_2' \\ \mathbf{e}_3' \\ \mathbf{e}_4' \\ \mathbf{v}_1' \\ \mathbf{v}_2' \\ \mathbf{v}_3' \\ \mathbf{v}_4' \end{bmatrix} = \frac{1}{16} \begin{bmatrix} 9 & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 6 & 6 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 6 & 1 & 6 & 1 & 0 & 1 & 1 & 0 & 0 \\ 6 & 0 & 1 & 6 & 1 & 0 & 1 & 1 & 0 \\ 6 & 1 & 0 & 1 & 6 & 0 & 0 & 1 & 1 \\ 4 & 4 & 4 & 0 & 0 & 4 & 0 & 0 & 0 \\ 4 & 0 & 4 & 4 & 0 & 0 & 4 & 0 & 0 \\ 4 & 0 & 0 & 4 & 4 & 0 & 0 & 4 & 0 \\ 4 & 4 & 0 & 0 & 4 & 0 & 0 & 0 & 4 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v} \\ \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \\ \mathbf{e}_4 \\ \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{v}_4 \end{bmatrix}$$

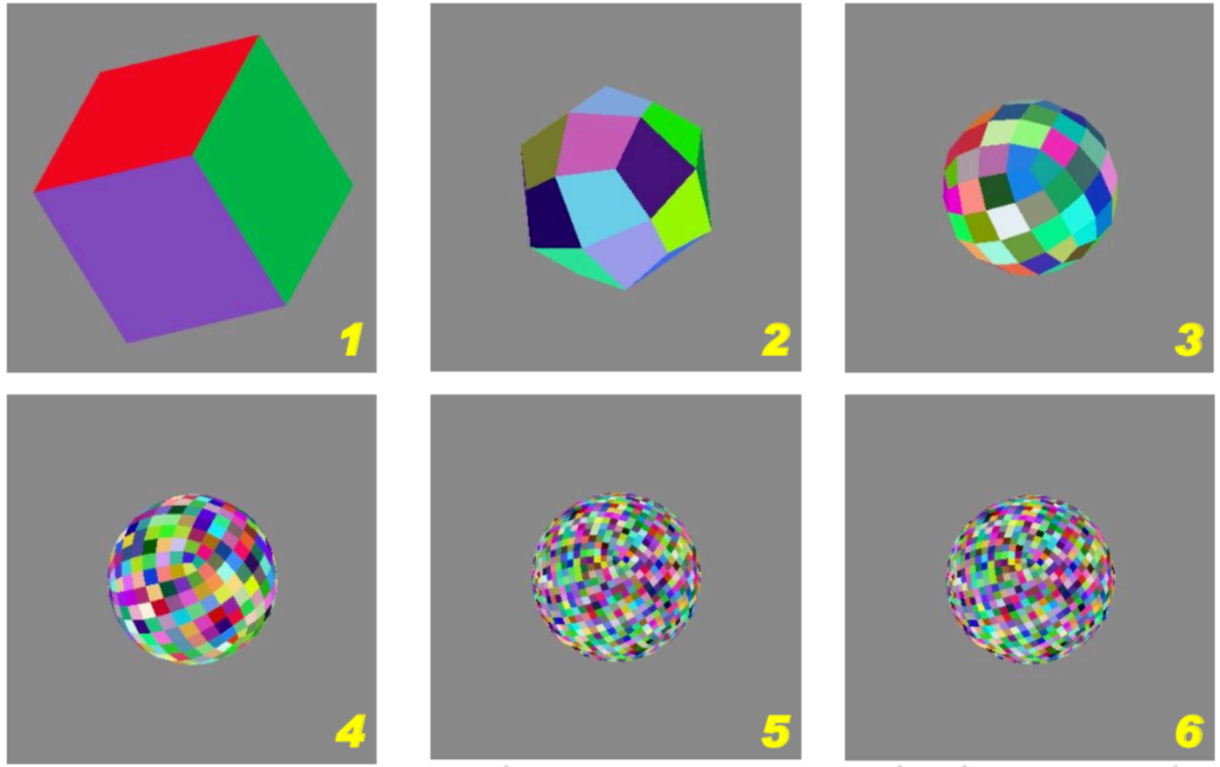
- 其顶点的极限为$v_\infty = \dfrac{n^2 v + 4\sum_{j=1}^{4} e_j + \sum_{j=1}^{4} f_j}{n(n+5)}$

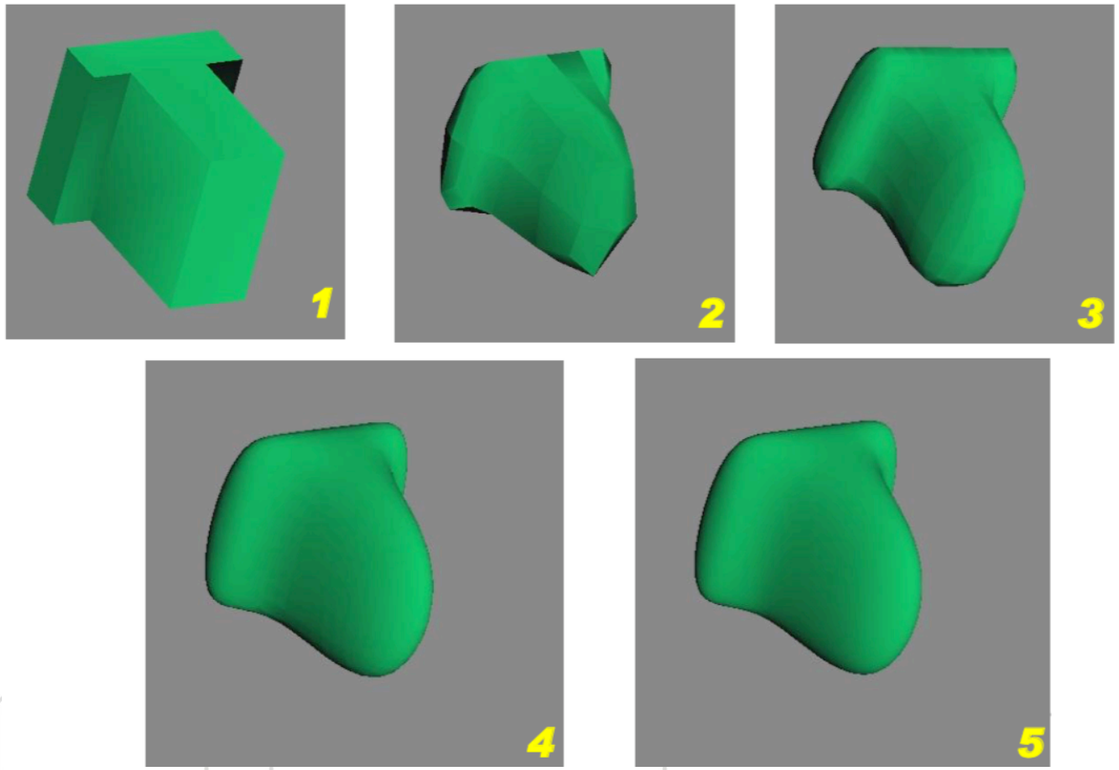
曲面细分

Catmull-Clark algorithm

– 网格的极限为degree=(3,3)的B样条曲面

- 比Doo-Sabin算法（degree=(2,2)）产生的网格更平滑

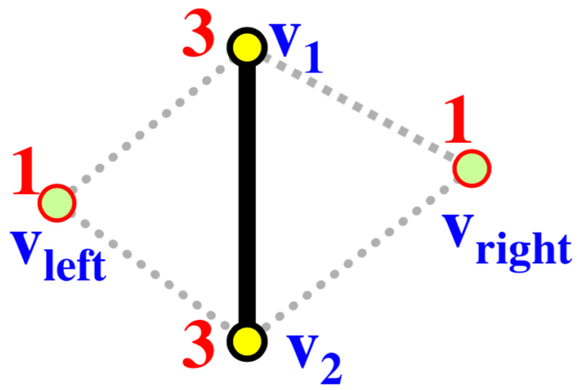




曲面细分

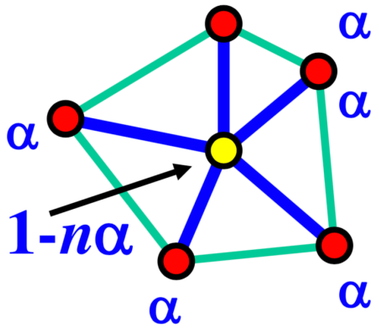
Loop's algorithm

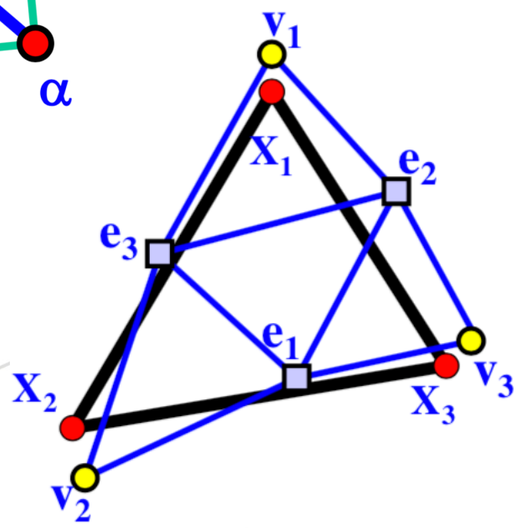
- 1987年，由Charles Loop提出
  - 此处Loop并非循环！
- Loop算法对于原网格中每条边计算一个新的edge point，对每个顶点计算一个新的vertex point
  - Loop算法只能应用于三角形网格
  - 令v_1v_2为原网格中的一条边，v_{left}, v_{right}为其相邻三角形中的另外两个顶点，则该边对应的edge point为
    - $e = \frac{3}{8}(v_1 + v_2) + \frac{1}{8}(v_{left} + v_{right})$



Loop's algorithm

- Loop算法对于原网格中每条边计算一个新的edge point，对每个顶点计算一个新的vertex point
  - 对每个顶点v，其新的vertex point v'由其相邻顶点$v_1, v_2, \cdots, v_n$的加权平均给出
    - $v' = (1 - n\alpha)v + \alpha \sum_{j=1}^{n} v_j$
    - 其中$\alpha = \begin{cases} \frac{3}{16} & n = 3 \\ \frac{1}{n}\left[\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2\right] & n > 3 \end{cases}$
  - 将每个三角形划分为四个小三角形





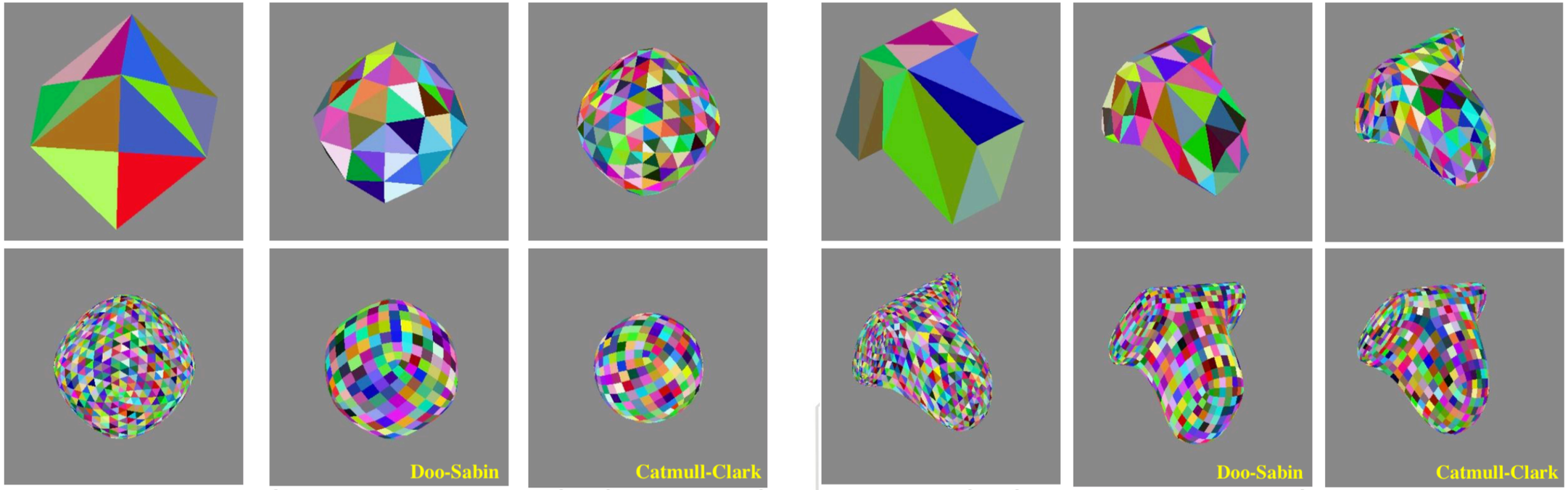
曲面细分

Loop's algorithm

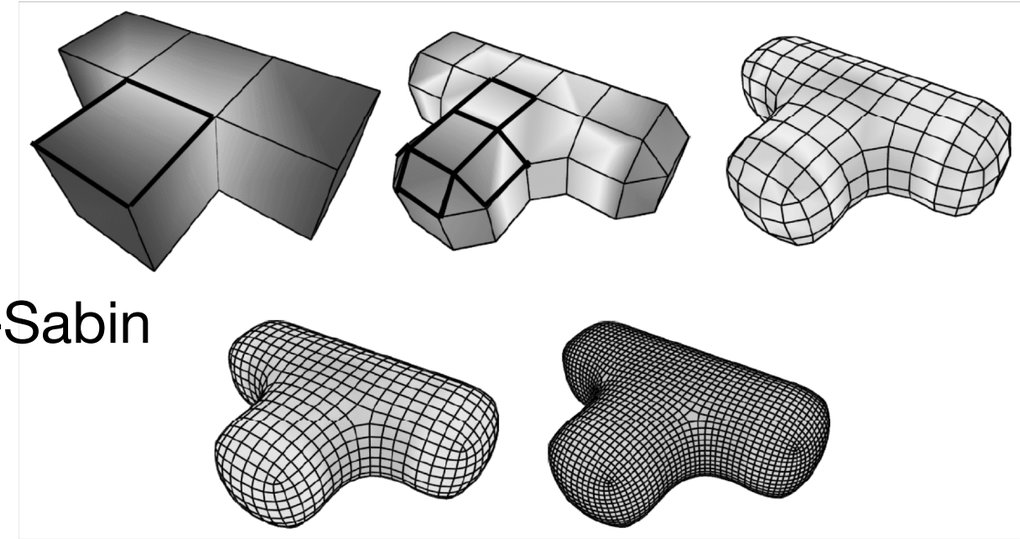
– Loop算法中，顶点的极限$v_\infty = \frac{3+8(n-1)\alpha}{3+8n\alpha} v + \frac{8\alpha}{3+8n\alpha} \sum_{j=1}^{n} v_j$

– 所有顶点valence都为6时，网格的极限为C^2连续的Bézier三角形

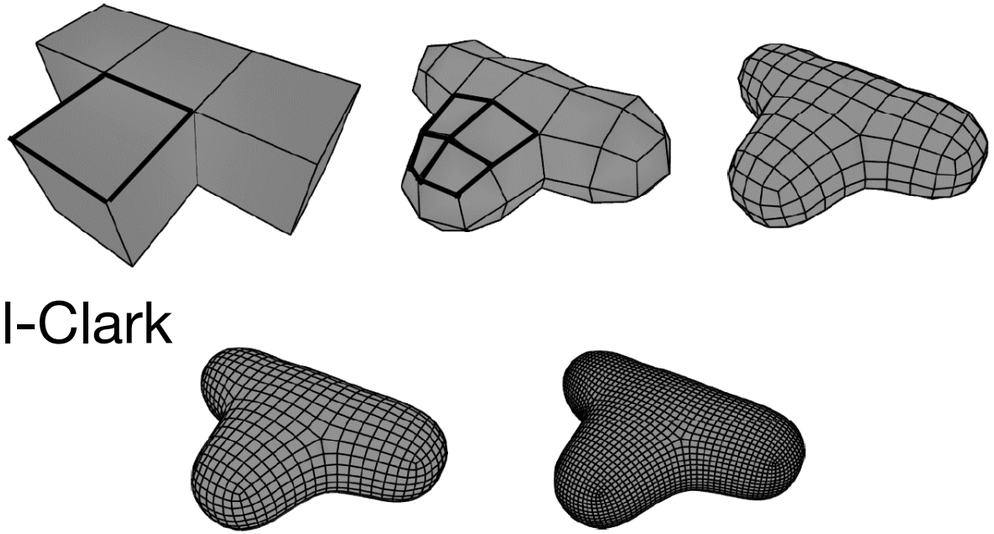
– 否则，valence不为6的顶点将收敛至extraordinary vertices (G^1)



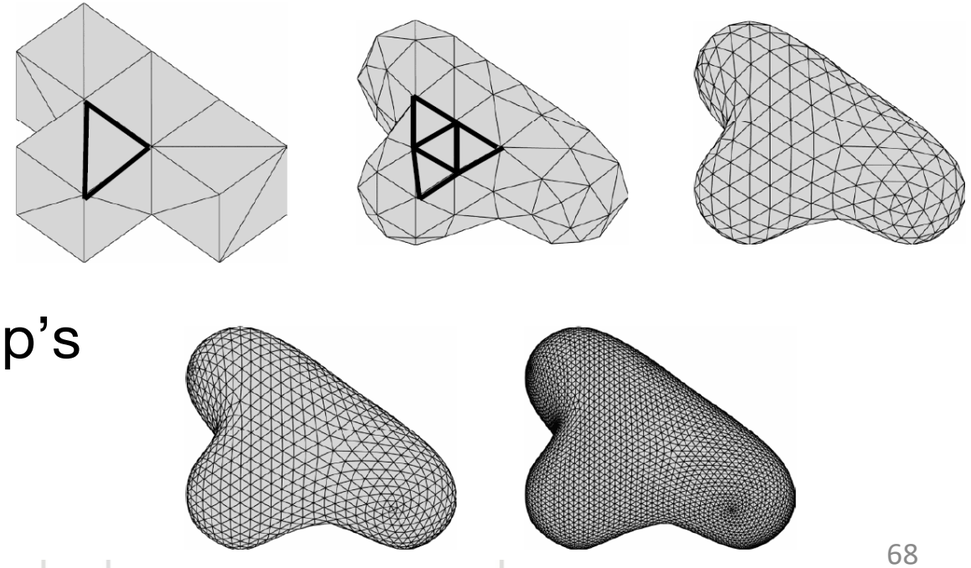
曲面细分



Doo-Sabin



Catmull-Clark



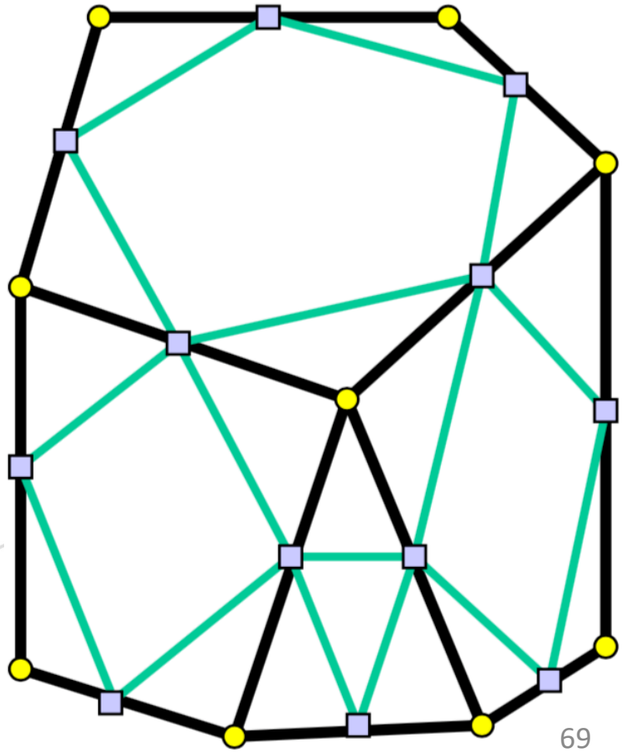
Loop’s

曲面细分

Peters-Reif算法

- 1998年，由Peters与Reif提出
- 连接每条边的中点形成新的网格
  - 新网格中有两类面：一类对应原网格中的面；另一类对应原网格中的顶点
  - 新旧顶点间的关系可由以下矩阵描述

$$\begin{bmatrix} \mathbf{v}'_1 \\ \mathbf{v}'_2 \\ \vdots \\ \mathbf{v}'_{n-1} \\ \mathbf{v}'_n \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \cdots & \cdots & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \cdots & 0 & \frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_{n-1} \\ \mathbf{v}_n \end{bmatrix}$$

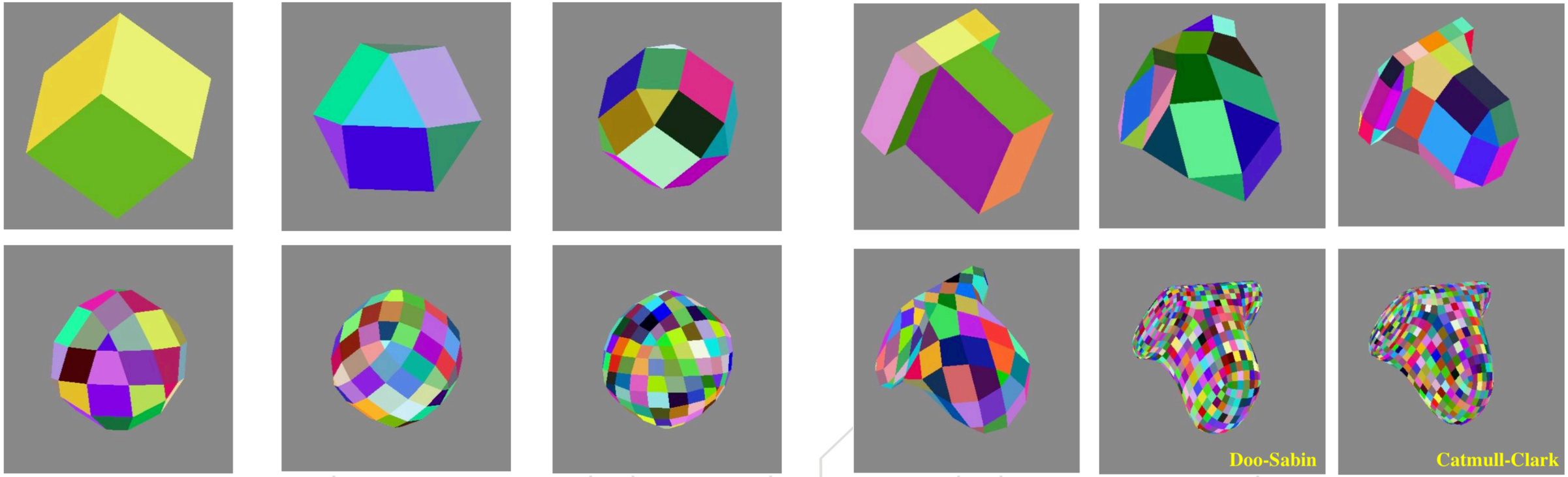


- 网格的极限为G^1连续

曲面细分

Peters-Reif算法

– 结果更接近于Doo-Sabin



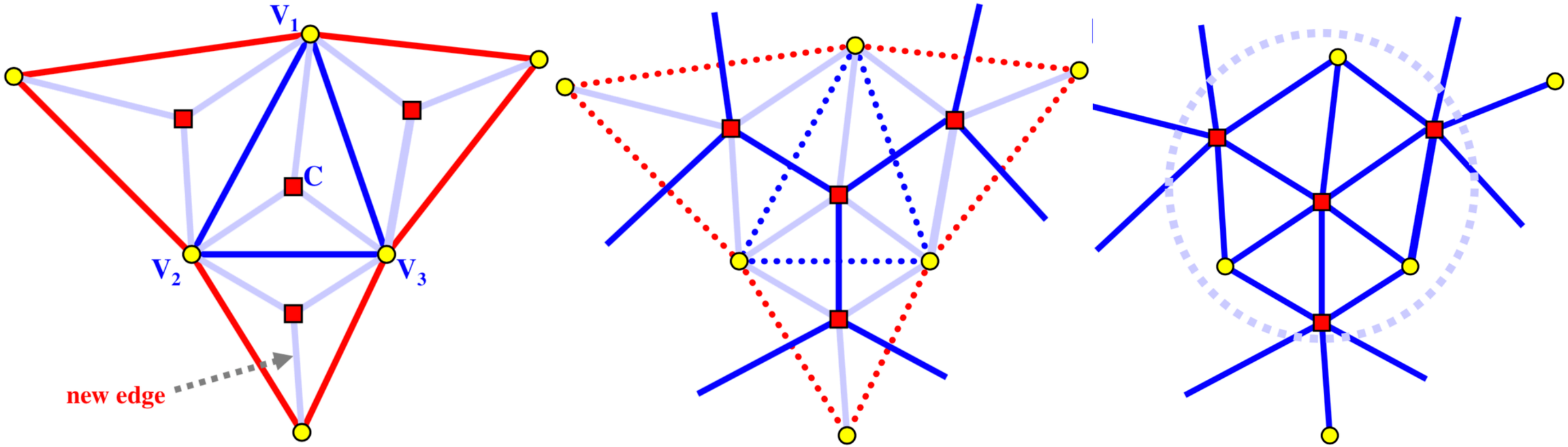
曲面细分

√3-subdivision of Kobbelt

– 2000年，由Kobbelt提出

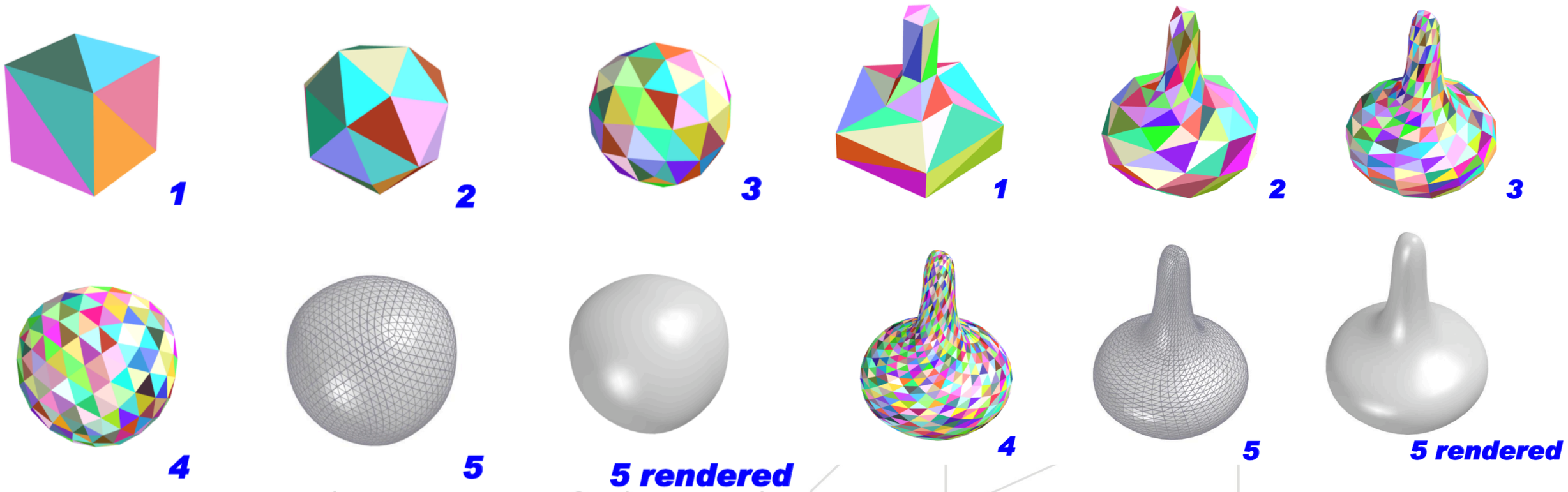
– 该算法只能应用于三角形网格，分三步进行

- 1. 1-to-3细分；2. 调整顶点位置；3. flip edges



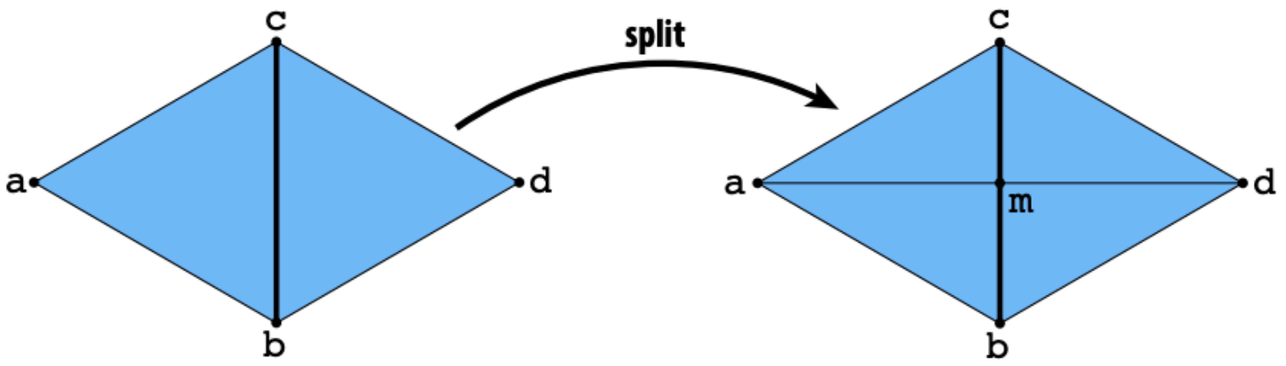
曲面细分

- $\sqrt{3}$-subdivision of Kobbelt
  - 其网格的极限为C^2连续（除extraordinary points外）

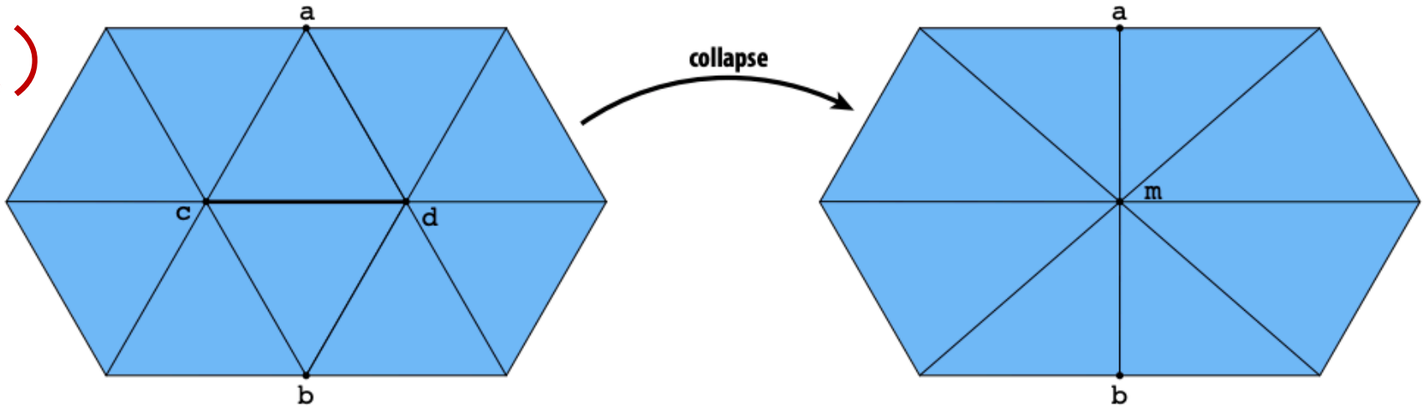


曲面处理

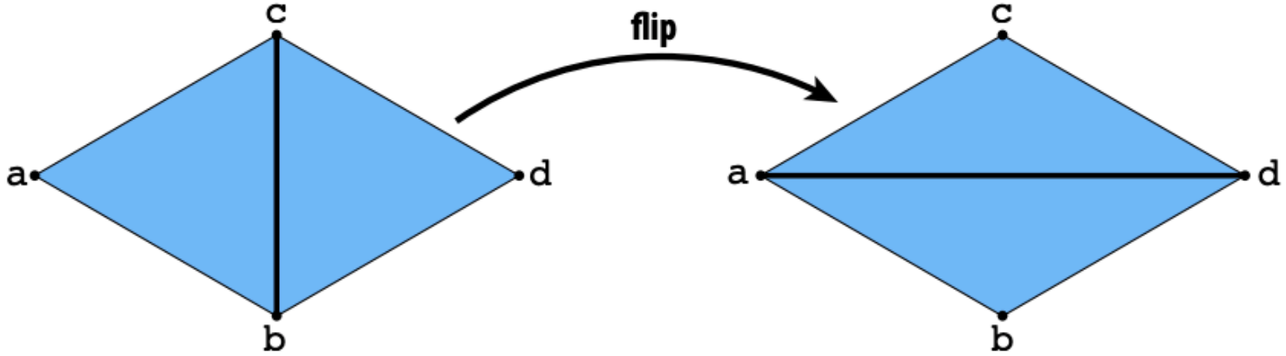
上采样（细分）



下采样（简化）



重采样



曲面简化

目标

– 减少网格中元素个数而保持其大致形状不变

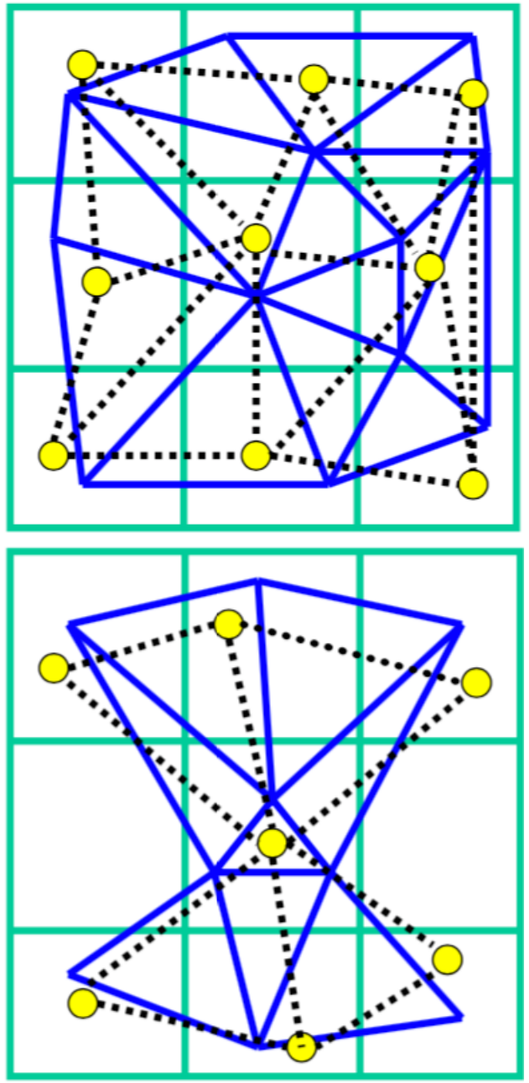
思路

– Vertex clustering, vertex removal, edge collapse

曲面简化

Vertex clustering

- 将空间划分为方块，每个方块由一个顶点表示
  - 顶点可由方块内原网格的顶点加权平均得到
    - $v = \frac{w_1 v_1 + w_2 v_2 + \cdots + w_n v_n}{w_1 + w_2 + \cdots + w_n}$
- 简单易于实现
- 保持原几何体的大致形状
  - 原网格中顶点与简化后网格中顶点距离小于方块
- 然而，即使原网格为流形，该方法也无法保证简化后的网格仍为流形
  - 网格的一部分可能简化为一个顶点

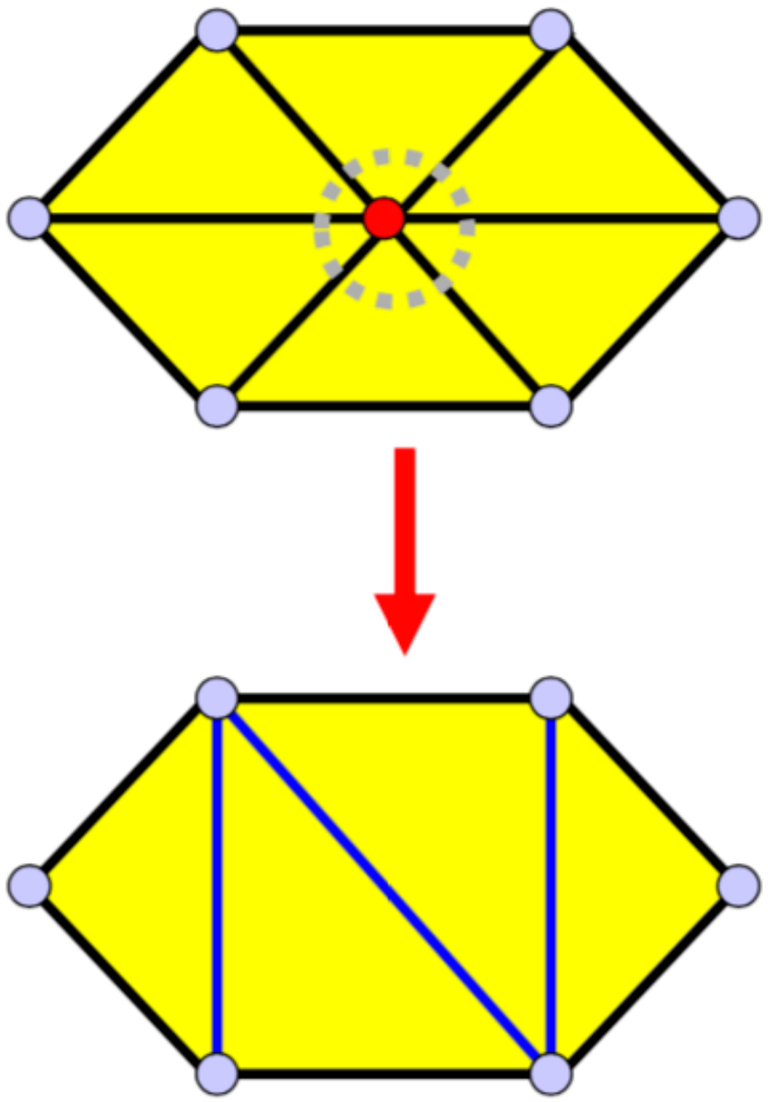


solid: original mesh
dotted: new mesh

曲面简化

Vertex removal

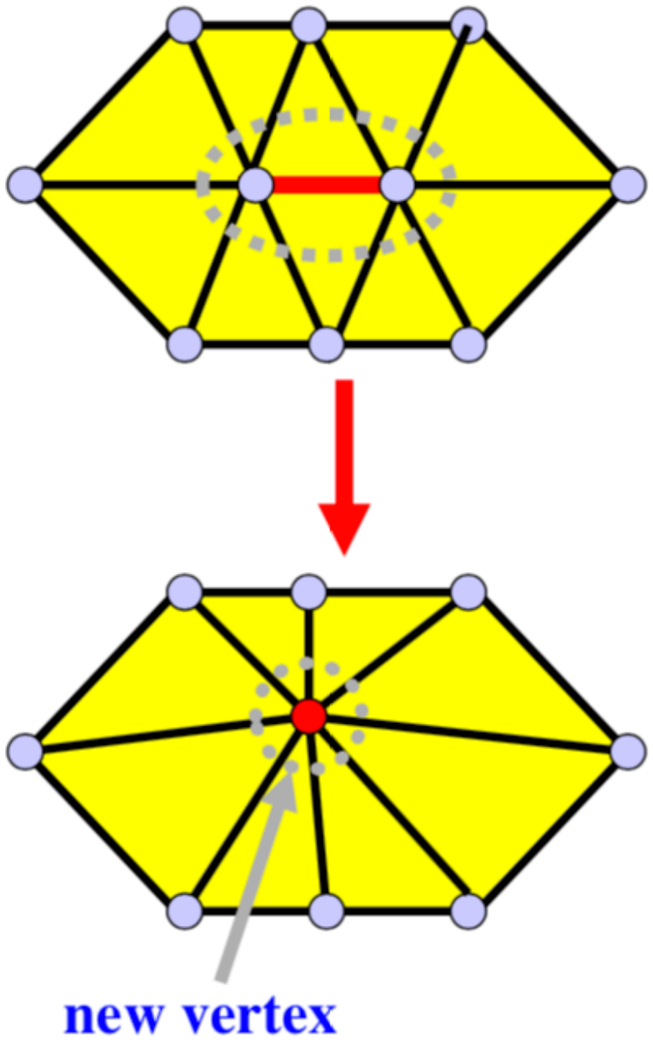
- 每次去除一个顶点
- 去除顶点将产生一个k条边的“洞”
  - k为顶点的valence
  - “补洞”需要k-2个三角形
- 因此，每次去除一个顶点，将减少1个顶点，3条边，及2个三角形



曲面简化

Edge collapse

- 每次去除一条边
- 该边对应的两个顶点合并为一个顶点
- 该边相邻的两个三角形各塌陷为一条边
- 因此，每次去除一条边后，将减少1个顶点，1条边，及2个三角形

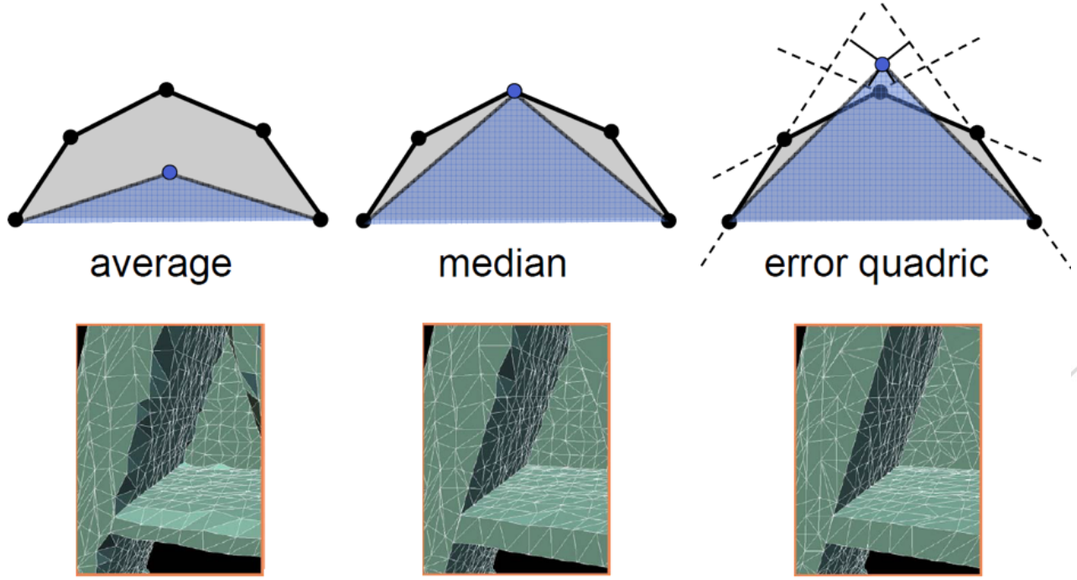


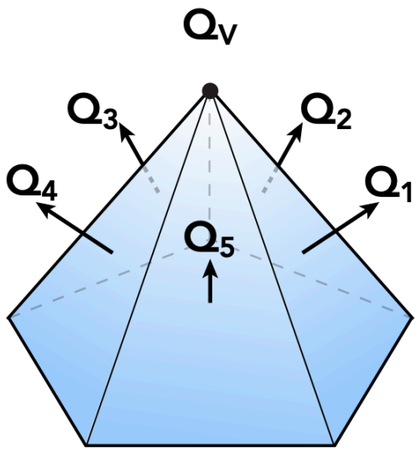
曲面简化

Quadric error metric

– 如何决定是否去除一个顶点（或一条边）？

- 计算去除后产生的误差
- 1997年，Garland与Heckbert在SIGGRAPH上发表的文章提出
- 每个顶点的误差由与其相接的面的误差之和决定
- 每条边的误差由去除后产生的顶点中最小误差决定





$$Q_V = \sum_{i=1}^{N} Q_i$$

网格及其处理小结

基本概念

- 网格基本元素：顶点，边，面
  - 拓扑信息与几何信息
- 流形：局部具有欧氏空间性质的空间

网格性质

- 定向：兼容；可定向
- 带边界与无边界；可展性
- 拓扑性质：同胚；欧拉示性数

数据结构

- half-edge：基于边的网格数据结构
- 网格元素查询；网格编辑

网格处理

- Subdivision, simplification

Questions?