计算机图形学

童 伟 华　管理科研楼1205室

E-mail: tongwh@ustc.edu.cn

中国科学技术大学　数学科学学院

http://math.ustc.edu.cn/

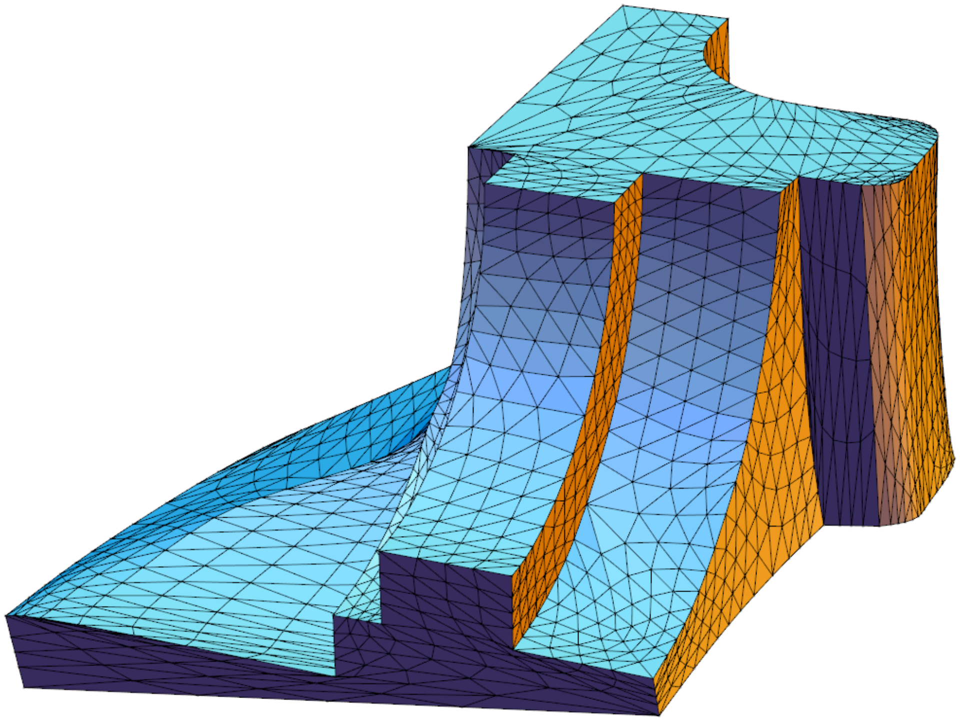
第五节 多边形网格模型

多边形网格模型

- 由多边形彼此相接构成的网格
  - 多边形称为网格的面，多边形的顶点也称为网格的顶点
  - 一般要求两张相邻面的公共边完全相同，即不能出现某一面的一个顶点在另一面的边中间
- 数学的语言：单纯复形（simplicial complex，图论或代数拓扑）
- 计算机图学事实上的工业标准，几乎所有的图形加速卡都支持网格模型的硬件加速绘制；在计算机辅助设计或实体造型中，可用于实体模型的边界表示

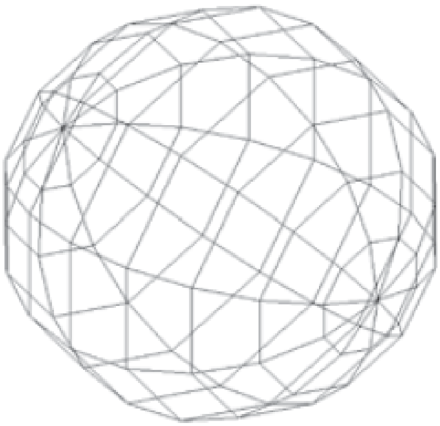
多边形网格的优势

- 容易表示
  - 数据结构简单
- 性质简单
  - 每个面只有一个法向量
  - 容易确定内外侧
- 容易绘制
  - 多边形填充
  - 纹理映射
  - 硬件加速



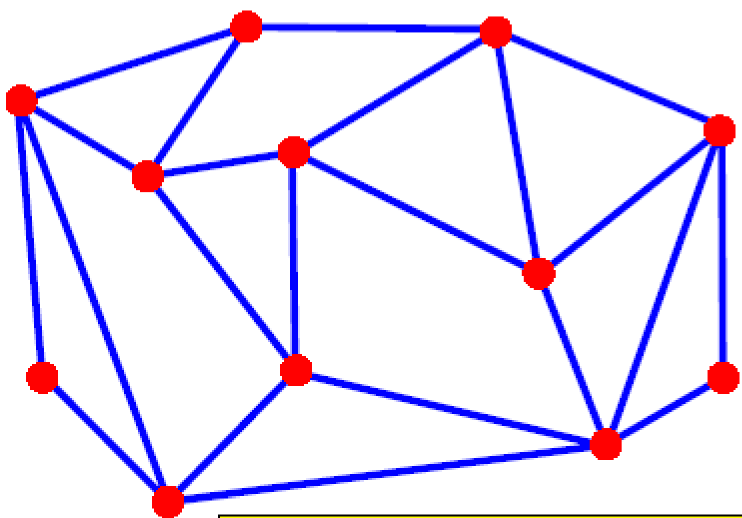
OpenGL与多边形网格

- 多边形网格是OpenGL中基本的图元类型，支持硬件加速
- 在几何建模中，还有许多其它的表示方法
  - 参数曲面（Parametric Surface）
  - 隐式曲面（Implicit Surface ）
  - 细分曲面（Subdivision Surface）
- 多边形网格是OpenGL接受其它表示的中转站
  - 利用多边形网格逼近其它表示形式的曲面，然后利用OpenGL绘制曲面
  - 例如：球面



网格的数学定义

- 利用图论的语言



G = <**V**,**E**>
V = vertices =
{A,B,C,D,E,F,G,H,I,J,K,L}
E = edges =
{(A,B),(B,C),(C,D),(D,E),(E,F),(F,G),
(G,H),(H,A),(A,J),(A,G),(B,J),(K,F),
(C,L),(C,I),(D,I),(D,F),(F,I),(G,K),
(J,L),(J,K),(K,L),(L,I)}

Vertex degree (valence) = number of edges incident on vertex
deg(J) = 4, deg(H) = 2
k-regular graph = graph whose vertices all have degree *k*

Face: cycle of vertices/edges which cannot be shortened
F = faces =
{(A,H,G),(A,J,K,G),(B,A,J),(B,C,L,J),(C,I,J),(C,D,I),
(D,E,F),(D,I,F),(L,I,F,K),(L,J,K),(K,F,G)}

连通性（Connectivity）

- 拓扑性质

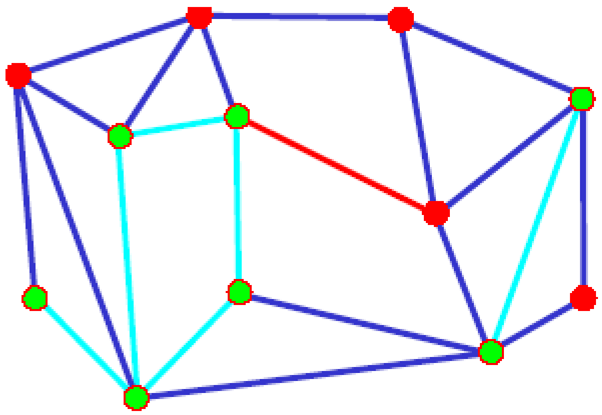
Graph is ***connected*** if there is a path of edges connecting every two vertices

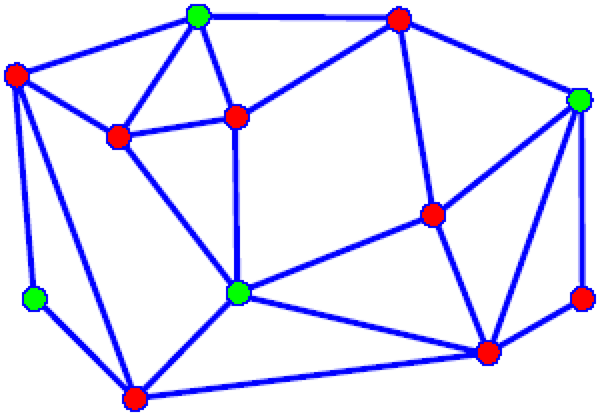
Graph is ***k-connected*** if between every two vertices there are *k* edge-disjoint paths

Graph **G’=<V’,E’>** is a ***subgraph*** of graph **G=<V,E>** if **V’** is a subset of **V** and **E’** is the subset of **E** incident on **V’**

Connected component of a graph: maximal connected subgraph

Subset **V’** of **V** is an ***independent*** set in **G** if the subgraph it induces does not contain any edges of **E**

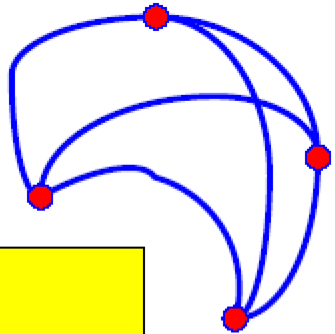




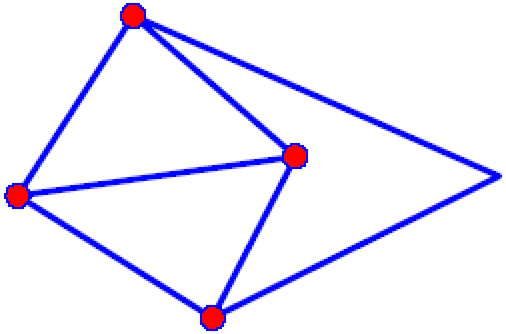
平面图（Planar Graphs）

- 数学定义

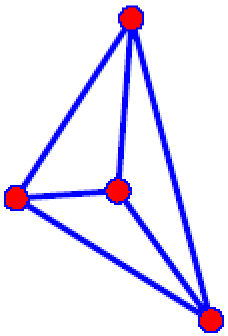
Planar Graph



Plane Graph



Straight Line Plane Graph



Planar graph: graph whose vertices and edges can be embedded in R^2 such that its edges do not intersect

Every planar graph can be drawn as a ***straight-line plane graph***

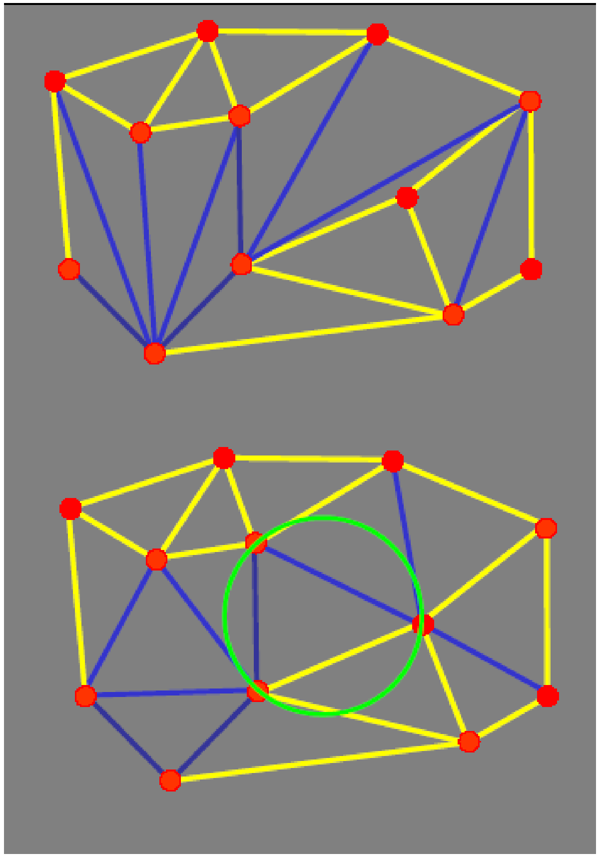
三角化（Triangulation）

- 数学定义

Triangulation: straight line plane graph all of whose faces are triangles

Delaunay triangulation of a set of points: unique set of triangles such that the circumcircle of any triangle does not contain any other point

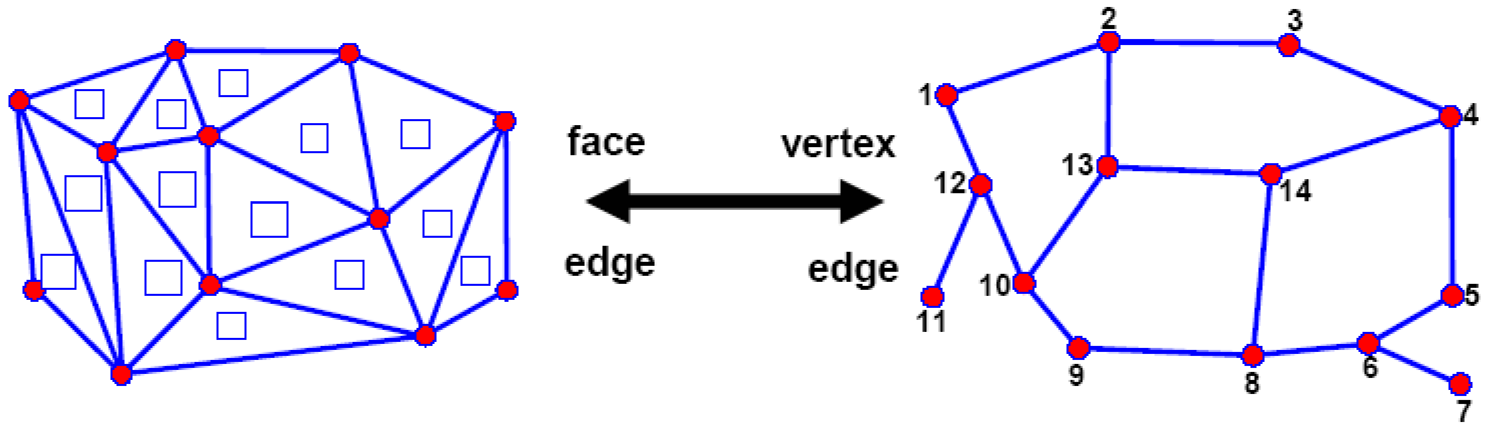
Delaunay triangulation avoids long and skinny triangles



- 思考：Delaunay三角剖分有哪些好的性质？

对偶性（Duality）

- 数学定义



- Delaunay Triangulation vs. Voronoi Graph
- 计算几何（Computational Geometry）

单纯复形

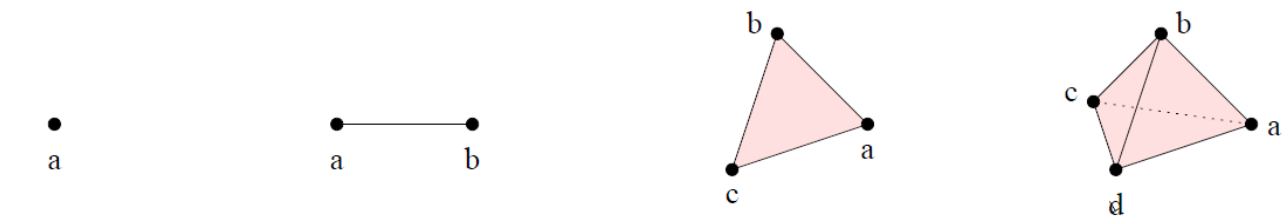
- 单纯形（Simplex）

Definition 3.1 (combinations) Let $S = \{p_0, p_1, \ldots, p_k\} \subseteq \mathbb{R}^d$. A *linear combination* is $x = \sum_{i=0}^{k} \lambda_i p_i$, for some $\lambda_i \in \mathbb{R}$. An *affine combination* is a linear combination with $\sum_{i=0}^{k} \lambda_i = 1$. A *convex combination* is a an affine combination with $\lambda_i \geq 0$, for all i. The set of all convex combinations is the *convex hull*.

Definition 3.2 (independence) A set S is *linearly (affinely) independent* if no point in S is a linear (affine) combination of the other points in S.

Definition 3.3 (k-simplex) A *k-simplex* is the convex hull of $k+1$ affinely independent points $S = \{v_0, v_1, \ldots, v_k\}$. The points in S are the *vertices* of the simplex.

Definition 3.4 (face, coface) Let σ be a k-simplex defined by $S = \{v_0, v_1, \ldots, v_k\}$. A simplex τ defined by $T \subseteq S$ is a *face* of σ and has σ as a *coface*. The relationship is denoted with $\sigma \geq \tau$ and $\tau \leq \sigma$. Note that $\sigma \leq \sigma$ and $\sigma \geq \sigma$.



vertex a | edge [a, b] | triangle [a, b, c] | tetrahedron [a, b, c, d]

- A k-simplex is a k-dimensional subspace of R^d
- 单纯形的组合结构：单纯形由一系列低维的面构成

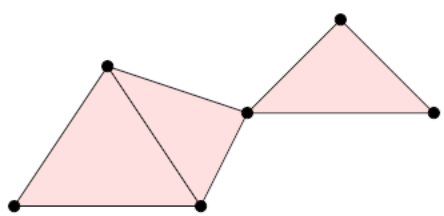
单纯复形

- 如何利用一系列的单纯形去表示形状?
- 关键：利用公共的面将单纯形拼接起来 –> 单纯复形

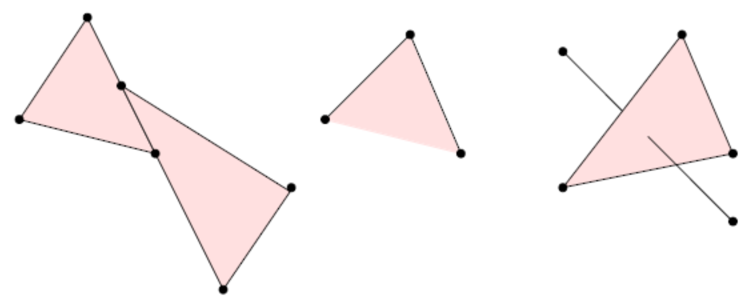
Definition 3.5 (simplicial complex) A *simplicial complex* K is a finite set of simplices such that

1. $\sigma \in K, \tau \leq \sigma \Rightarrow \tau \in K$,
2. $\sigma, \sigma' \in K \Rightarrow \sigma \cap \sigma' \leq \sigma, \sigma'$ <u>or $\sigma \cap \sigma' = \emptyset$</u>.

The *dimension* of K is $\dim K = \max\{\dim \sigma \mid \sigma \in K\}$. The *vertices* of K are the zero-simplices in K. A simplex is *principal* if it has no proper coface in K.



(a) The middle triangle shares an edge with the triangle on the left, and a vertex with the triangle on the right.

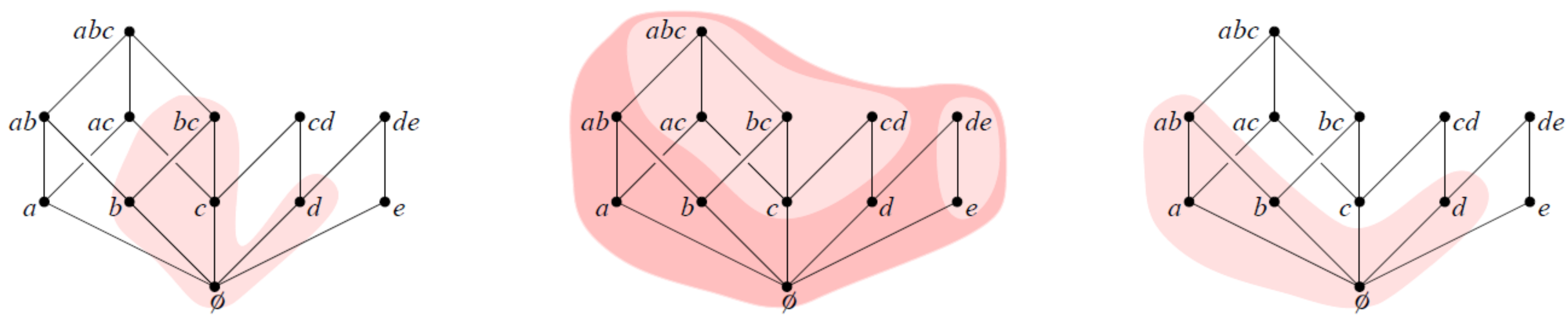


(b) In the middle, the triangle is missing an edge. The simplices on the left and right intersect, but not along shared simplices.

单纯复形

■ 子复形、链环、星形（subcomplex，link, star）

Definition 3.11 (subcomplex, link, star) A *subcomplex* is a simplicial complex $L \subseteq K$. The smallest subcomplex containing a subset $L \subseteq K$ is its closure, $\mathrm{Cl}\, L = \{\tau \in K \mid \tau \leq \sigma \in L\}$. The *star of* L contains all of the cofaces of L, $\mathrm{St}\, L = \{\sigma \in K \mid \sigma \geq \tau \in L\}$. The *link of* L is the boundary of its star, $\mathrm{Lk}\, L = \mathrm{Cl}\, \mathrm{St}\, L - \mathrm{St}\,(\mathrm{Cl}\, L - \{\emptyset\})$.



(a) $\mathrm{Cl}\,\{bc, d\}$

(b) $\mathrm{St}\,\{c, e\}$ (light) and its closure $\mathrm{Cl}\,\mathrm{St}\,\{c, e\}$ (dark)

(c) $\mathrm{Lk}\,\{c, e\}$

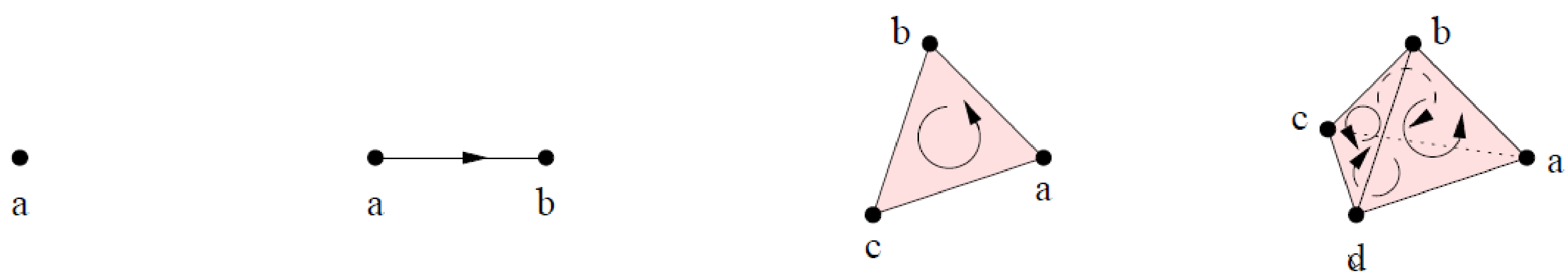
单纯复形

■ 可定向

Definition 3.14 (orientation) Let K be a simplicial complex. An *orientation* of a k-simplex $\sigma \in K$, $\sigma = \{v_0, v_1, \ldots, v_k\}, v_i \in K$ is an equivalence class of orderings of the vertices of σ, where

$$(v_0, v_1, \ldots, v_k) \sim (v_{\tau(0)}, v_{\tau(1)}, \ldots, v_{\tau(k)}) \tag{1}$$

are equivalent orderings if the parity of the permutation τ is even. We denote an *oriented simplex*, a simplex with an equivalence class of orderings, by $[\sigma]$.



vertex
a

edge
[a, b]

triangle
[a, b, c]

tetrahedron
[a, b, c, d]

Definition 3.15 (orientability) Two k-simplices sharing a $(k-1)$-face σ are *consistently oriented* if they induce different orientations on σ. A triangulable d-manifold is *orientable* if all d-simplices can be oriented consistently. Otherwise, the d-manifold is *non-orientable*

单纯复形

■ 欧拉示性数（Euler Characteristic）

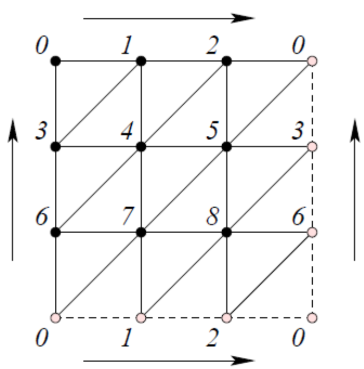
Definition 3.17 (Euler characteristic) Let K be a simplicial complex and $s_i = |\{\sigma \in K \mid \dim \sigma = i\}|$. The *Euler characteristic* $\chi(K)$ is

$$\chi(K) = \sum_{i=0}^{\dim K} (-1)^i s_i = \sum_{\sigma \in K - \{\emptyset\}} (-1)^{\dim \sigma}. \tag{2}$$

While it is defined for a simplicial complex, the Euler characteristic is an integer invariant for $|K|$, the underlying space of K. Given any triangulation of a space $\mathbb{M}$, we always will get the same integer, which we will call the Euler characteristic of that space $\chi(\mathbb{M})$.

■ 欧拉示性数：拓扑不变量

■ 可计算



(a) A triangulation for the diagram of the torus $\mathbb{T}^2$

2-Manifold	χ
Sphere $\mathbb{S}^2$	2
Torus $\mathbb{T}^2$	0
Klein bottle $\mathbb{K}^2$	0
Projective plane $\mathbb{RP}^2$	1

(b) The Euler characteristics of our basic 2-manifolds

单纯复形

■ 亏格（Genus）

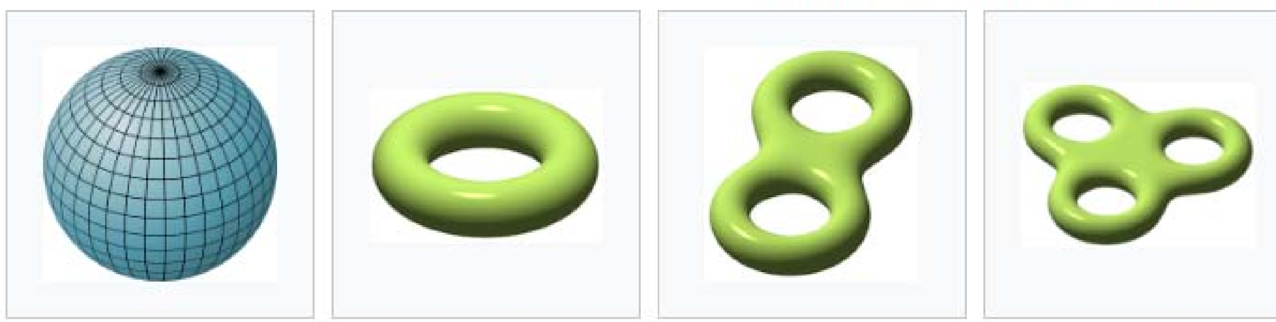
Definition 3.18 (genus) The connected sum of g tori is called a surface with *genus* g.

■ 亏格与欧拉示性数之间的关系

Theorem 3.2 *For compact surfaces* $\mathbb{M}_1, \mathbb{M}_2$, $\chi(\mathbb{M}_1 \# \mathbb{M}_2) = \chi(\mathbb{M}_1) + \chi(\mathbb{M}_2) - 2.$

Corollary 3.1 $\chi(g\mathbb{T}^2) = 2 - 2g$ *and* $\chi(g\mathbb{R}P^2) = 2 - g.$

Genus of orientable surfaces



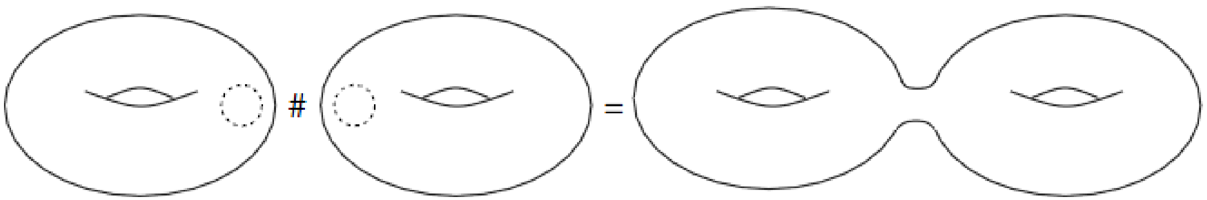
genus 0　genus 1　genus 2　genus 3

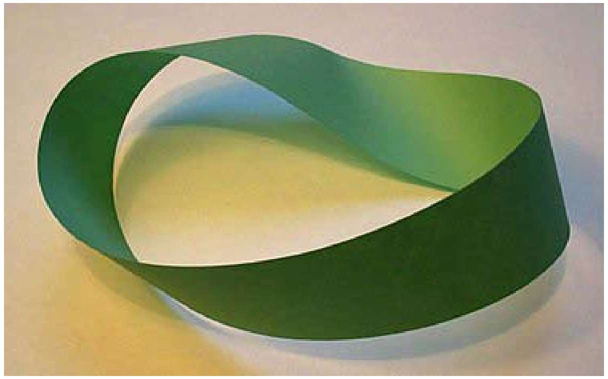
单纯复形

■ 闭曲面分类定理

Theorem 3.3 (Homeomorphism problem of 2-manifolds) *Closed compact surfaces $\mathbb{M}_1$ and $\mathbb{M}_2$ are homeomorphic, $\mathbb{M}_1 \approx \mathbb{M}_2$ iff*

1. $\chi(\mathbb{M}_1) = \chi(\mathbb{M}_2)$ *and*
2. *either both surfaces are orientable or both are non-orientable.*





Euler公式

- 对于简单多面体：

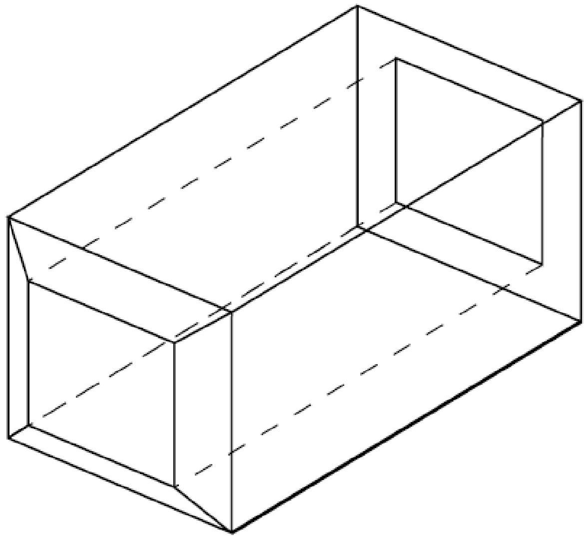
  $$V+F-E=2$$

  - 顶点数：V, 面数: F, 边数: E
  - 例如， 立方体：V=8, F=6, E=12

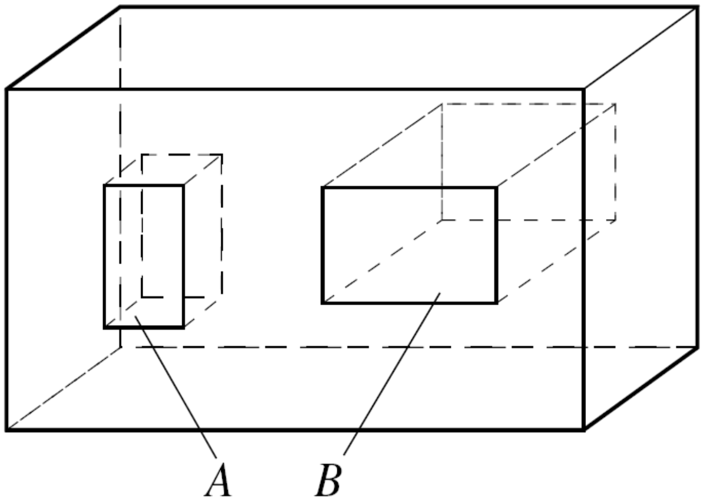
- 如果多面体不是简单的，在面上有H个洞，通过多面体的洞有G个，那么

  $$V+F-E=2+H-2G$$

Euler公式示例



V = 16, F = 16, E = 32, H = 0, G = 1



V = 24, F = 15, E = 36, H = 3, G = 1

多边形网格的类型

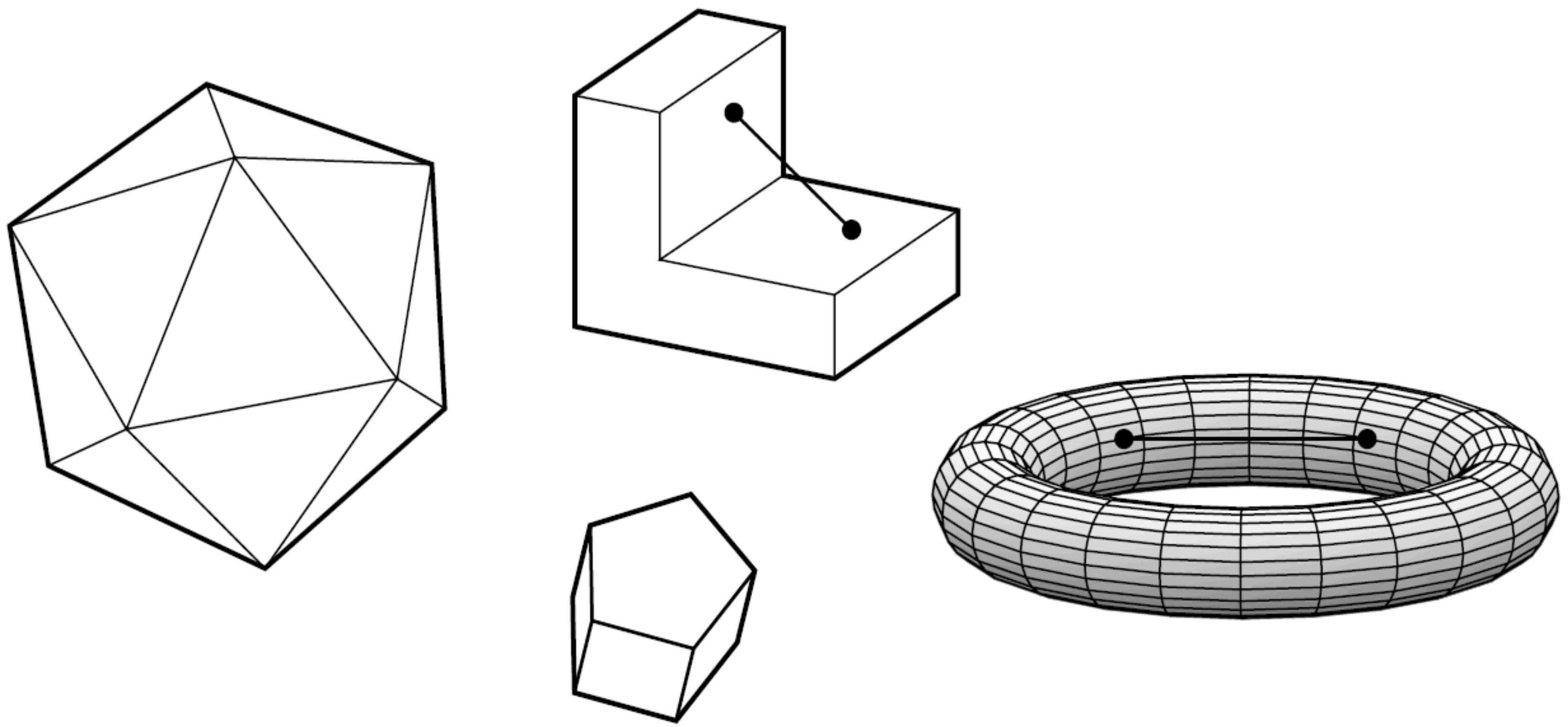
- 实体
  - 多边形网格形成一个封闭的空间区域
- 表面
  - 不形成空间封闭区域，表示一个无限薄的曲面
- 两者都称为多边形网格（polygonal mesh），有时简称为网格

网格的性质

- 给定一个由顶点、法向和面表组成的网格，那么它所表示的对象是什么呢？下列是感兴趣的性质：
  - 实体：如果网格形成一个封闭的有界区域
  - 连通性：如果任两个顶点间存在着由边构造的连续路径
  - 简单性：表示一个实体，而且没有洞，即可以没有粘贴变形到球面
  - 平面性：如果所有面都是平面多边形
    - 有些算法对平面多边形更有效
    - 因此三角网格非常实用

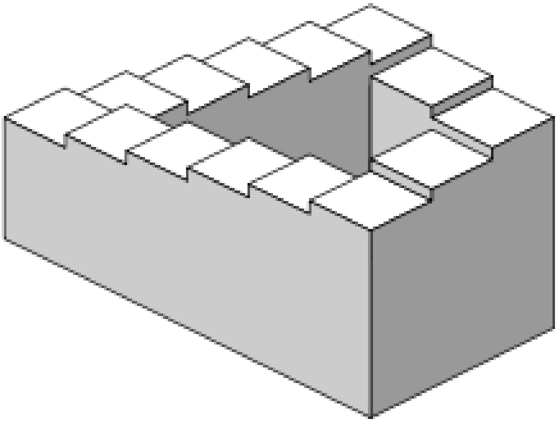
网格的性质（续）

- 凸性：网格表示凸体

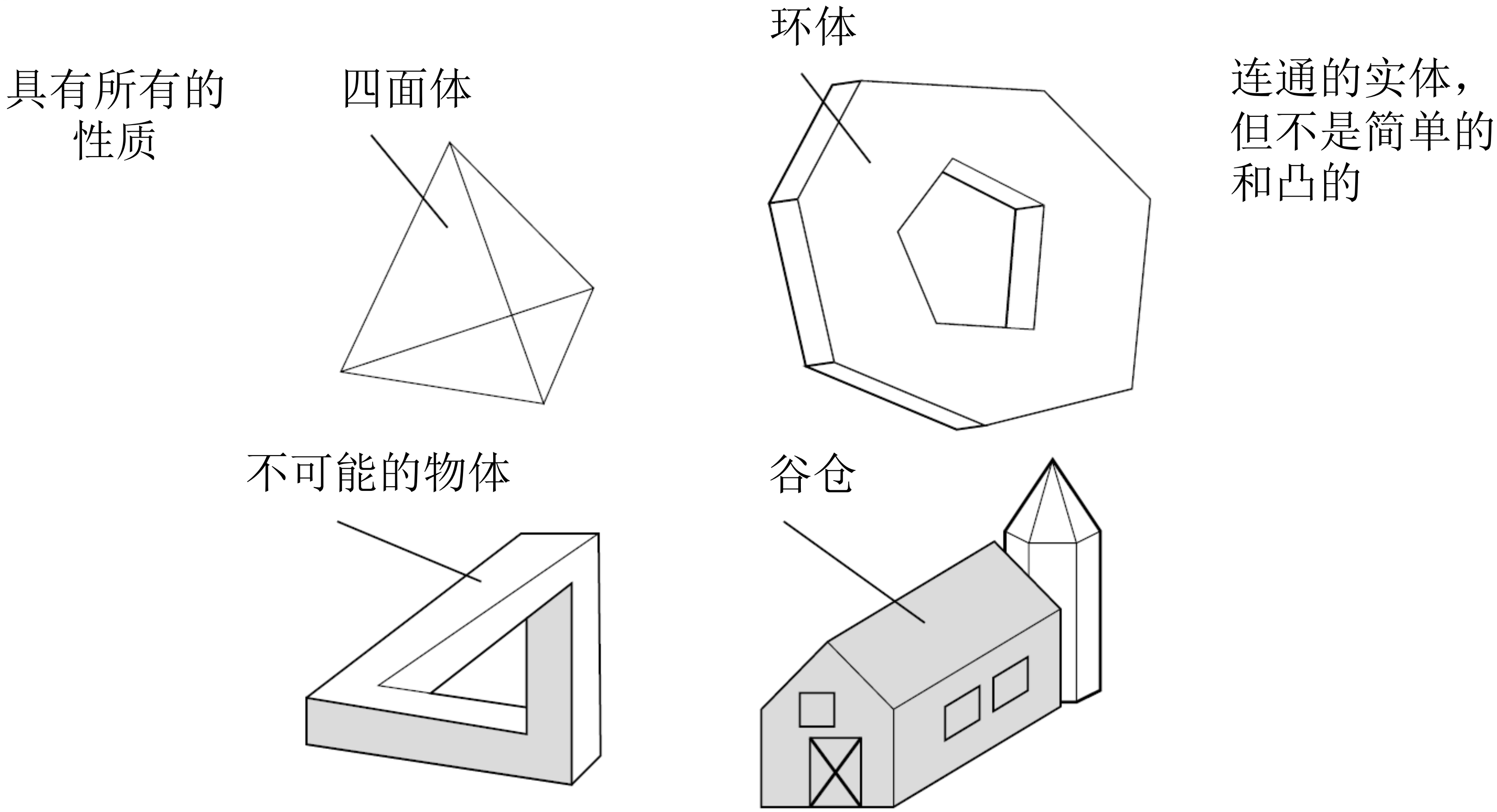


性质的检测与应用

- 有些性质比较容易检测，即存在简单算法，而有些性质则比较难以判断
  - 例如判断网格是否表示实体不是一件简单的事情
- 网格可以具有上述性质中的某几个或全部
  - 关键在于用网格做什么
  - 如果网格用来表示用某些材料构成的物理模型，那么就需要它至少是连通和实体
  - 从艺术角度考虑，网格完全可以表示非物理的实体

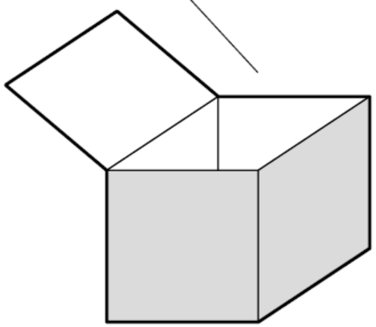


网格实例

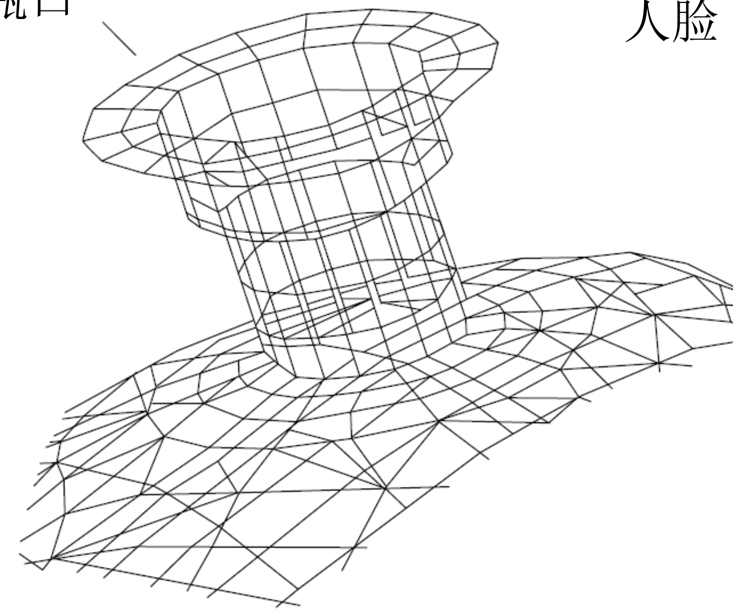


非实体的网格表示

开口盒子



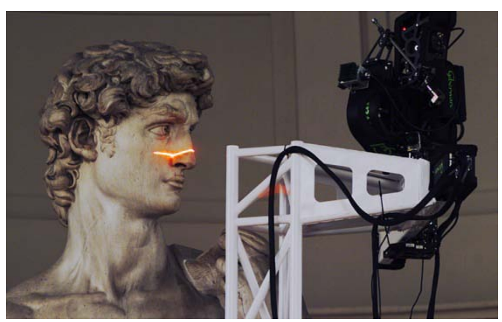
瓶口



人脸

如何得到网格

- 直接对网格进行造型是非常困难的
- 常用的方法：
  - 利用三维扫描设备，获取点云数据，进行曲面重建，获得网格模型
  - 利用几何造型软件设计模型，譬如NURBS，CSG等，然后将曲面转化为为近似的网格表示
- 任何表面都可以用多边形网格逼近到任意光滑精度，这称为多边形网格的完备性



曲面重建

网格的数据结构

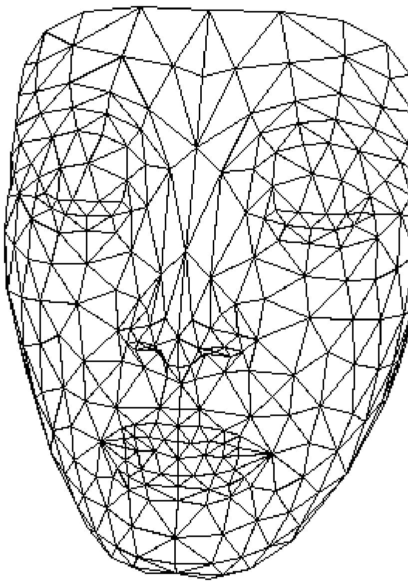
- 网格的用途
- Rendering
  - Triangle trip
- Geometry/topological queries
  - What are the vertices of face *#k*?
  - Are vertices *#i* and *#j* adjacent?
  - Which faces are adjacent face *#k*?
- Geometry/topological operations
  - Remove/add a vertex/face
  - Mesh simplification
  - Vertex split, edge collapse

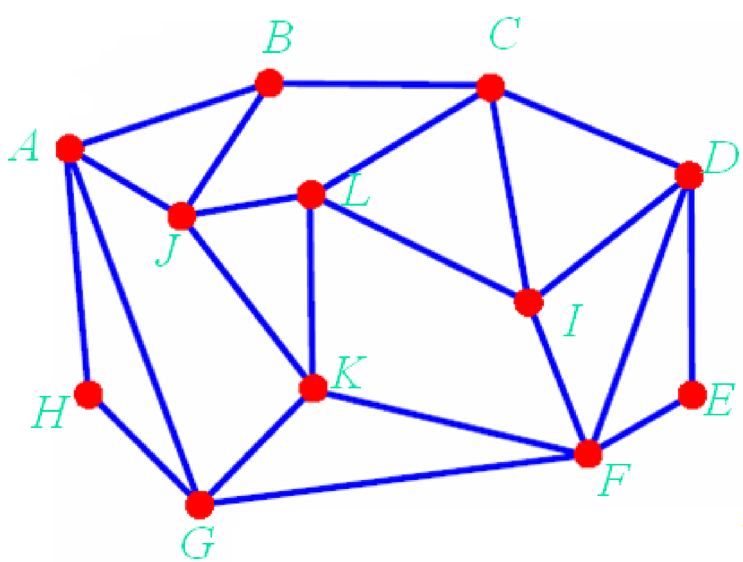
网格的存储

- Storage of generic meshes
  - Hard to implement efficiently
- Assume
  - Triangular
  - Orientable
  - Manifold
- How "good" is a data structure?
  - Space complexity
  - Time
    - Time to construct - preprocessing
    - Time to answer a query
    - Time to perform an operation (update the data structure)
  - Trade-off between time and space
  - Redundancy

网格的定义

- Position（Geometry information）
  - Vertex coordinates
- Connectivity（Topological information）
  - How do vertices connected?
- List of Edge
- Vertex-Edge
- Vertex-Face
- Combined





网格的定义

- Surface & material properties
  - Material color
  - Ambient, hightlight coefficients
  - Texture coordinates
  - BRDF, BTF
- Rendering properties
  - Lighting
  - Normals
  - Rendering modes

常用的网格文件类型

- General used mesh files
  - Wavefront OBJ (*.obj)
  - OFF (*.off)
  - PLY (*.ply, *.ply2)
  - STL (*.stl)
  - 3D Max (*.max, *.3ds)
  - VRML(*.vrl)
  - Inventor (*.iv)
- Storage
  - Text – (Recommended)
  - Binary

Wavefront OBJ File Format

- Vertices
  - Start with char ‘v’
  - (x,y,z) coordinates
- Faces
  - Start with char ‘f’
  - Indices of its vertices in the file
- Other properties
  - Normal, texture coordinates, material, etc.

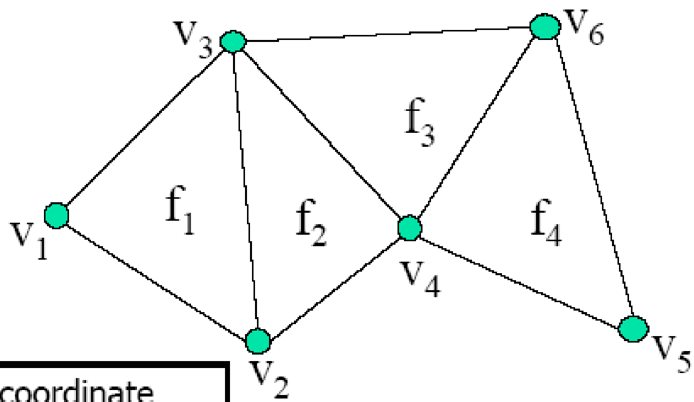
```
v 1.0 0.0 0.0
v 0.0 1.0 0.0
v 0.0 -1.0 0.0
v 0.0 0.0 1.0
f 1 2 3
f 1 4 2
f 3 2 4
f 1 3 4
```

面列表

- List of vertices
  - Position coordinates
- List of faces
  - Triplets of pointers to face vertices (c1,c2,c3)
- Queries:
  - What are the vertices of face #3?
    - Answered in O(1) - checking third triplet
  - Are vertices i and j adjacent?
    - A pass over all faces is necessary – NOT GOOD

面列表的例子

- 一个简单的例子



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)

面列表

- **Pros:**
  - Convenient and efficient (memory wise)
  - Can represent non-manifold meshes
- **Cons:**
  - Too simple - not enough information on relations between vertices & faces

邻接矩阵

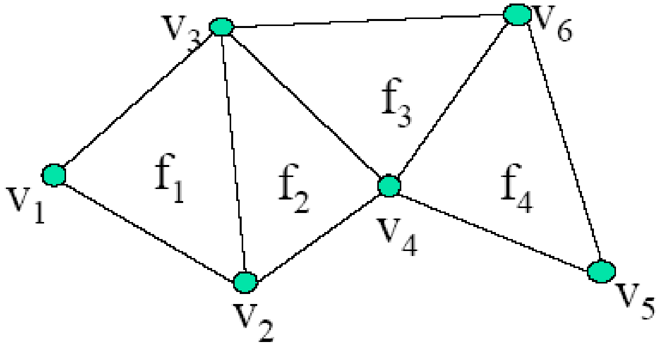
- Adjacency Matrix
- View mesh as connected graph
- Given n vertices build n*n matrix of adjacency information
  - Entry (i,j) is TRUE value if vertices i and j are adjacent
- Geometric info
  - list of vertex coordinates
- Add faces
  - list of triplets of vertex indices (v1,v2,v3)

邻接矩阵的例子

- 一个简单的例子

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	

邻接矩阵

- Adjacency Matrix – Queries
- What are the vertices of face #3?
  - O(1) – checking third triplet of faces
- Are vertices i and j adjacent?
  - O(1) - checking adjacency matrix at location (i,j).
- Which faces are adjacent to vertex j?
  - Full pass on all faces is necessary

邻接矩阵

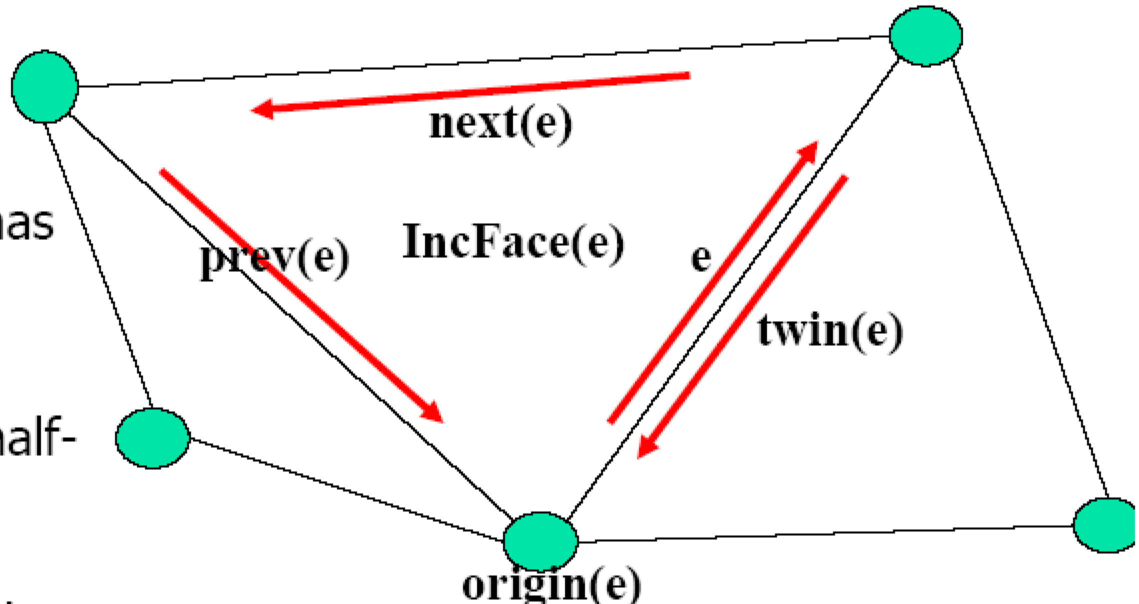
- Pros:
  - Information on vertices adjacency
  - Stores non-manifold meshes
- Cons:
  - Connects faces to their vertices, BUT NO connection between vertex and its face

双向连接边列表

- Doubly-Connected Edge List（DCEL）
- Record for each face, edge and vertex:
  - Geometric information
  - Topological information
  - Attribute information
- Half-Edge Structure

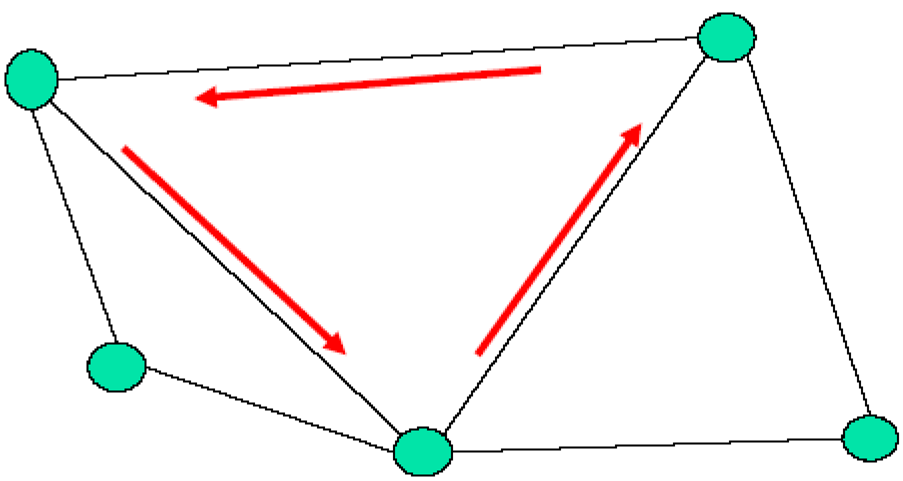
双向连接边列表

- Vertex record:
  - Coordinates
  - Pointer to one half-edge that has v as its origin
- Face record:
  - Pointer to one half-edge on its boundary
- Half-edge record:
  - Pointer to its origin, origin(e)
  - Pointer to its twin half-edge, twin(e)
  - Pointer to the face it bounds, IncidentFace(e) (face lies to left of e when traversed from origin to destination)
  - Next and previous edge on boundary of IncidentFace(e)



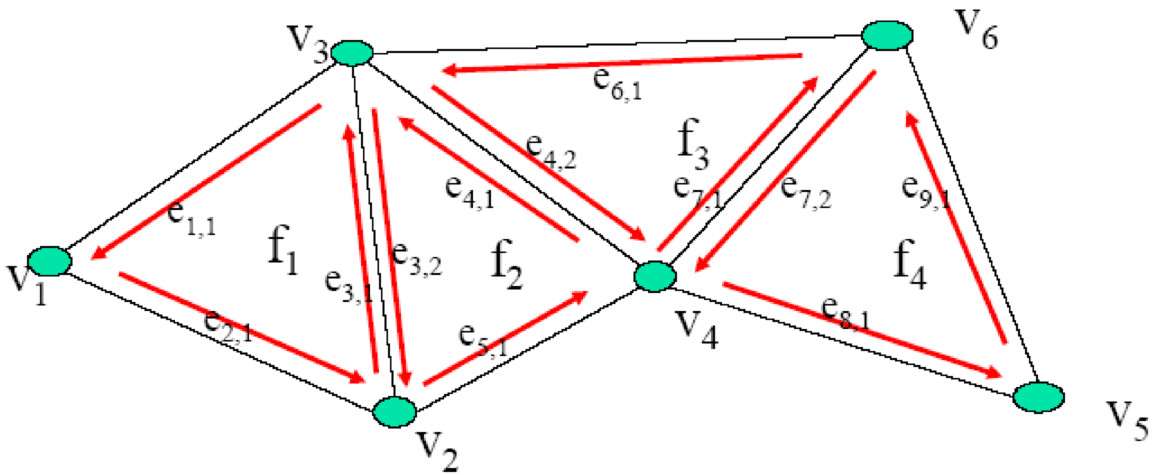
双向连接边列表

- Operations supported:
  - Walk around boundary of given face
  - Visit all edges incident to vertex v
- Queries:
  - Most queries are O(1)



双向连接边列表的例子

- 一个简单的例子

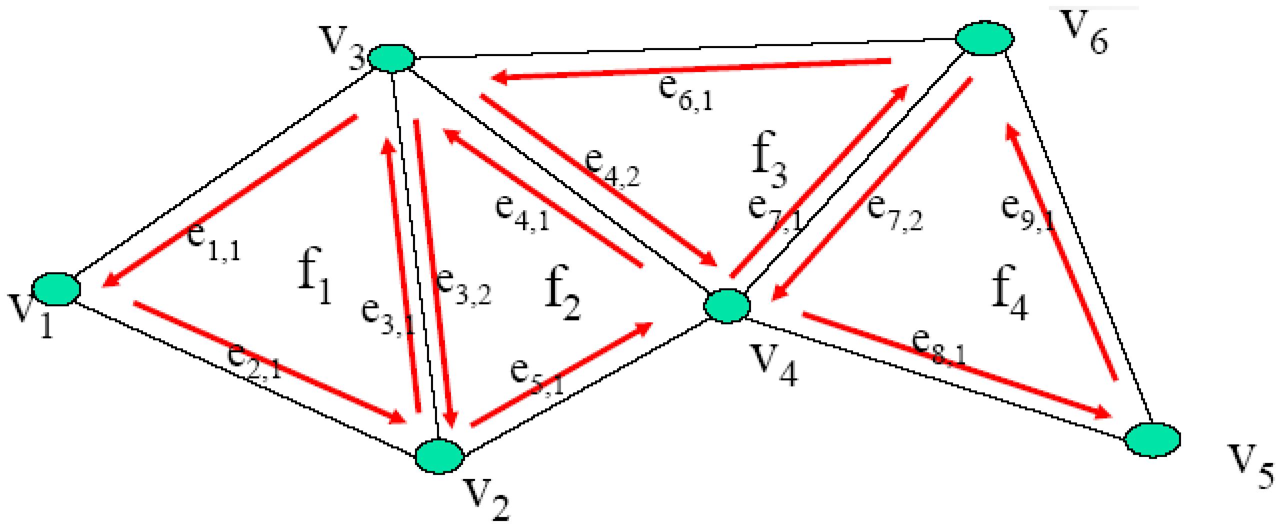


Vertex	coordinate	IncidentEdge
V_1	(x_1,y_1,z_1)	$e_{2,1}$
v_2	(x_2,y_2,z_2)	$e_{5,1}$
v_3	(x_3,y_3,z_3)	$e_{1,1}$
v_4	(x_4,y_4,z_4)	$e_{7,1}$
v_5	(x_5,y_5,z_5)	$e_{9,1}$
v_6	(x_6,y_6,z_6)	$e_{7,2}$

face	edge
f_1	$e_{1,1}$
f_2	$e_{5,1}$
f_3	$e_{4,2}$
f_4	$e_{8,1}$

双向连接边列表的例子

- 续上页



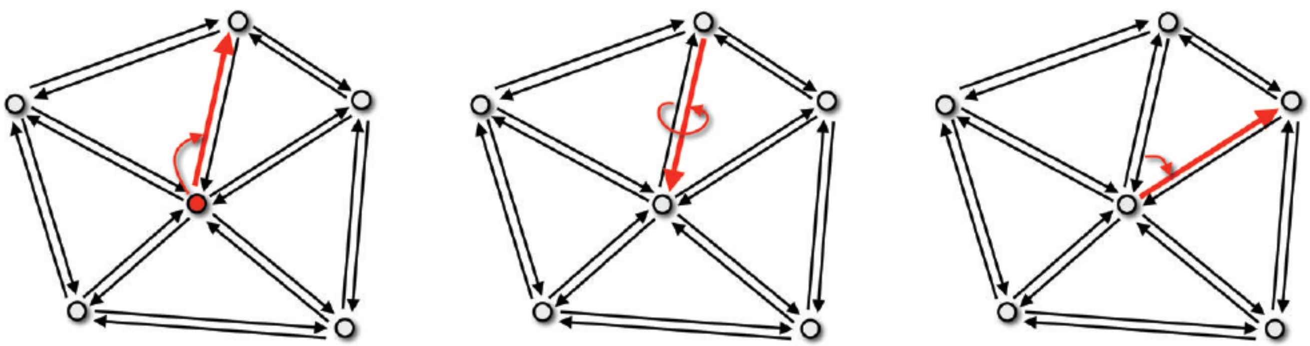
Half-edge	origin	twin	IncidentFace	next	prev
$e_{3,1}$	v_2	$e_{3,2}$	f_1	$e_{1,1}$	$e_{2,1}$
$e_{3,2}$	v_3	$e_{3,1}$	f_2	$e_{5,1}$	$e_{4,1}$
$e_{4,1}$	v_4	$e_{4,2}$	f_2	$e_{3,2}$	$e_{5,1}$
$e_{4,2}$	v_3	$e_{4,1}$	f_3	$e_{7,1}$	$e_{6,1}$

双向连接边列表

- Pros
  - All queries in O(1) time
  - All operations are O(1) (usually)
- Cons
  - Represents only manifold meshes

半边数据结构

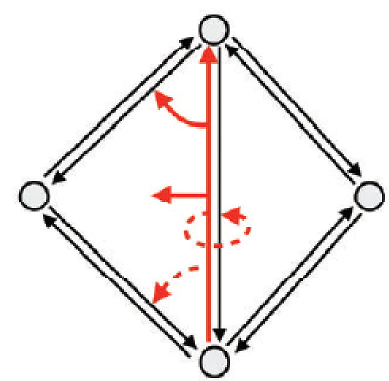
- Halfedge Based Data Structure
- 基本思想： splitting each (unoriented) edge into two oriented halfedges



Vertex	
Point	position
HalfedgeRef	halfedge

Face	
HalfedgeRef	halfedge

Halfedge	
VertexRef	vertex
FaceRef	face
HalfedgeRef	next
HalfedgeRef	prev
HalfedgeRef	opposite



常用查看网格的软件

- Commercial tools
  - Maya, 3ds Max et.al.
    - https://www.autodesk.com/products/maya/
  - 3D Exploration
    - http://www.xdsoft.com/explorer/
  - MeshLab
    - http://www.meshlab.com/
- User Written Viewers
  - Too many on the internet

网格编程库

- Use a good mesh library
  - CGAL
  - OpenMesh （MeshLab）
  - MeshMaker
  - Your own library

OpenGL

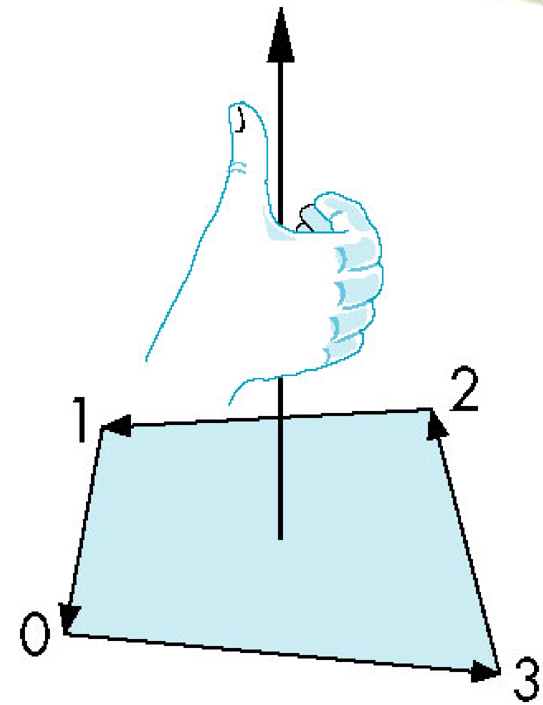
- 用每个多边形的各顶点的几何位置定义多边形
- 由此可有如下的OpenGL代码

```
glBegin(GL_POLYGON);
    glVertex3f(x1,y1,z1);
    glVertex3f(x6,y6,z6);
    glVertex3f(x8,y8,z8);
    glVertex3f(x7,y7,z7);
glEnd();
```

- 无效且无结构
  - 考虑移动一个顶点时会导致何种复杂操作

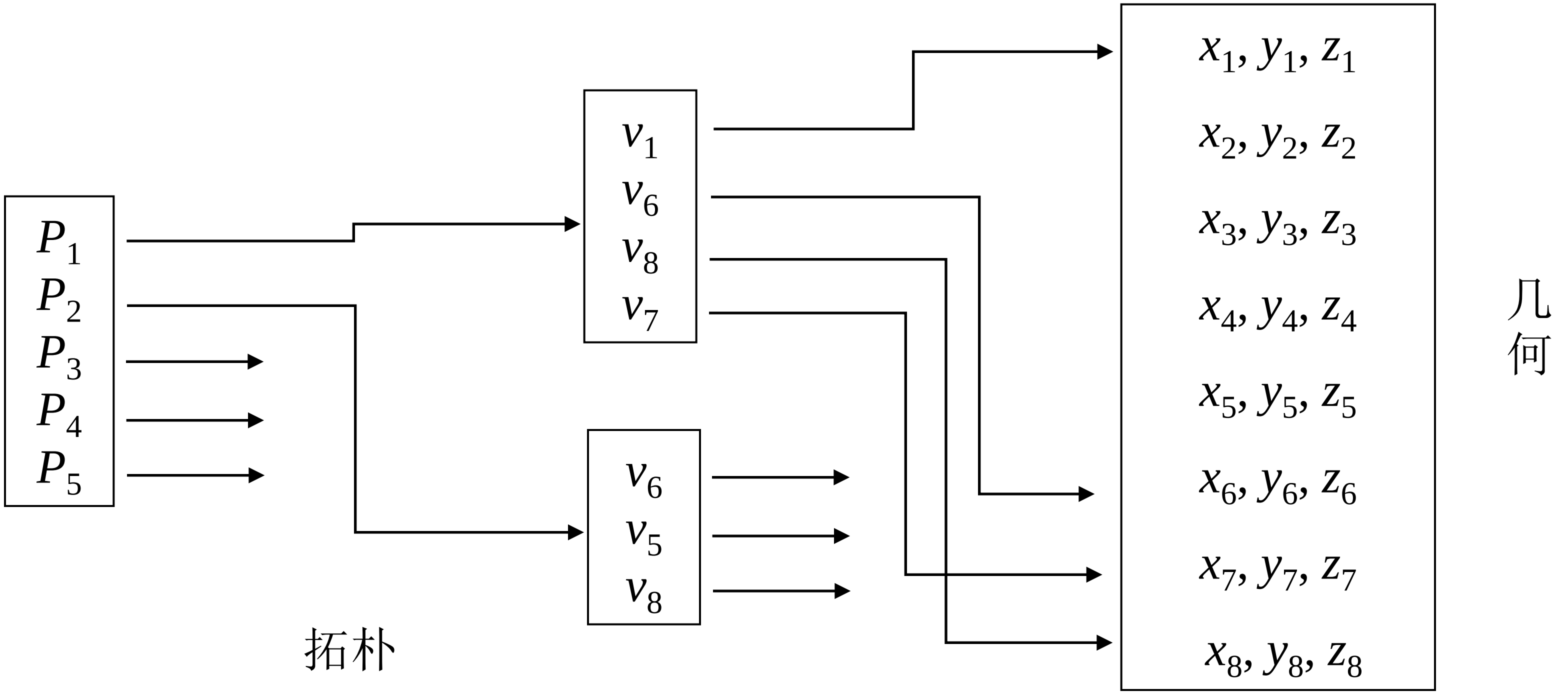
多边形的内外面

- 对于OpenGL而言，$\{v_1,v_6,v_8,v_7\}$顺序的顶点与$\{v_6,v_8,v_7,v_1\}$顺序的顶点定义等价的多边形，但是$\{v_7,v_8,v_6,v_1\}$则定义不同的多边形
- 上述两种方式定义的多边形分别称为多边形的内与外
- 利用右手法则判别
- OpenGL可以把多边形的内外面用完全不同的模式处理

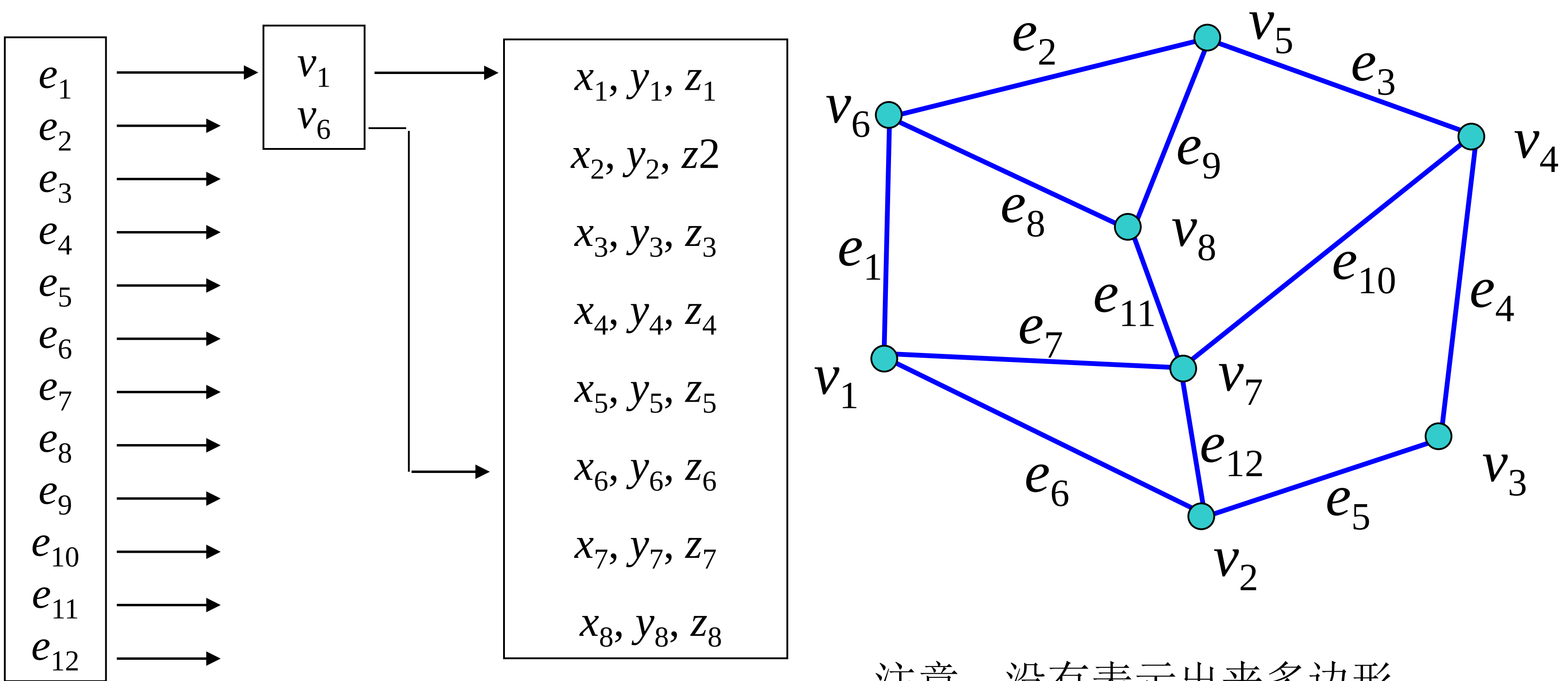


顶点表

- 把几何位置放在一个数组中
- 用各顶点构造边时，利用指向各顶点的指针
- 引入多边形表



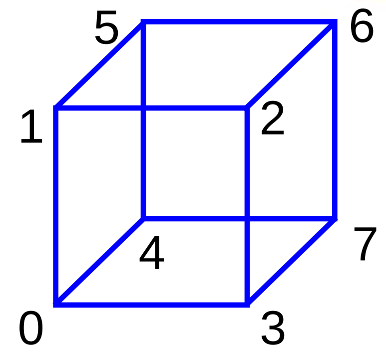
边表



注意：没有表示出来多边形

立方体

- 为立方体旋转程序建立彩色立方体



定义顶点和颜色的全局数组

```
GLdouble vertices[][3]=
  {{-1.0, -1.0, -1.0},{1.0,-1.0,-1.0},{1.0,1.0,-1.0},
  {-1.0,1.0,-1.0},{-1.0,-1.0,1.0},
  {1.0,1.0,1.0},{1.0,1.0,1.0}, {-1.0,1.0,1.0}};

GLdouble colors[][3]={{0.0,0.0,0.0},{1.0,0.0,0.0},
  {1.0,1.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},{1.0,0.0,1.0},
  {1.0,1.0,1.0},{0.0,1.0,1.0}};
```

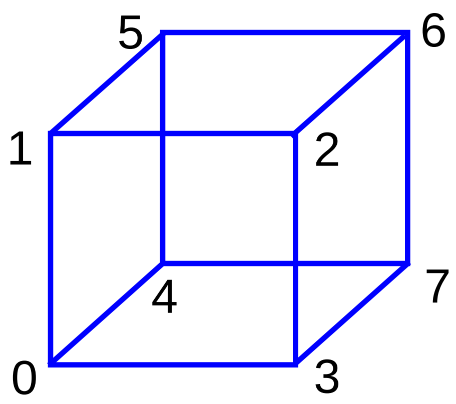
根据指标列表绘制多边形

- 根据在数组vertices中的一组指标绘制一个四边形，颜色对应于第一个指标

```
void polygon(int a, int b, int c, int d) {
  glBegin(GL_POLYGON);
      glColor3dv(colors[a]);
      glVertex3dv(vertices[a]);
      glVertex3dv(vertices[b]);
      glVertex3dv(vertices[c]);
      glVertex3dv(vertices[d]);
  glEnd();
}
```

利用表面绘制立方体

```
void colorcube() {
   polygon(0,3,2,1);
   polygon(2,3,7,6);
   polygon(0,4,7,3);
   polygon(1,2,6,5);
   polygon(4,5,6,7);
   polygon(0,1,5,4);
}
```



注意顶点的顺序保证表面的法向指向正确的方向，即立方体的外侧

效率

- 这种方法的缺陷在于为了在应用程序中建立模型，需要进行很多次函数调用才能绘制立方体
- 通过表面绘制立方体，最直接的方式需要
  - 6个glBegin和6个glEnd
  - 6个glColor
  - 24个glVertex
  - 如果应用纹理和光照的话还会更多

顶点数组

- OpenGL提供了一种功能，称为顶点数组（vertex arrays），利用这种功能可以存贮数组数据
- 支持六种类型的数组
  - 顶点
  - RGB颜色
  - 索引颜色
  - 法向
  - 纹理坐标
  - 边标志
- 我们将只需要RGB颜色与顶点数组

初始化

- 为了利用颜色与顶点数据，首先激活相应功能

```
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY);
```

- 标识数组的位置

```
glVertexPointer(3, GL_DOUBLE, 0, vertices);
```

三维数组

存贮为双精度数

数据是连接存放的

数据所在的数组

```
glColorPointer(3, GL_DOUBLE, 0, colors);
```

根据指标对应到面

- 构造表面指标的数组

```
GLubyte cubeIndices[24]={0,3,2,1,2,3,7,6,
0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4};
```

- 每四个相邻的指标描述立方体一个表面
- 利用glDrawElements取代在显示回调函数中所有的glVertex和glColor进行绘制

绘制立方体

- **方法一**

```
for(i=0; i<6; i++)
    glDrawElements(GL_POLYGON, 4, GL_UNSIGNED_BYTE,
&cubeIndices[4*i]);
```

指标数据的开始

绘制对象类型

指标的数目

指标数据的格式

- **方法二**

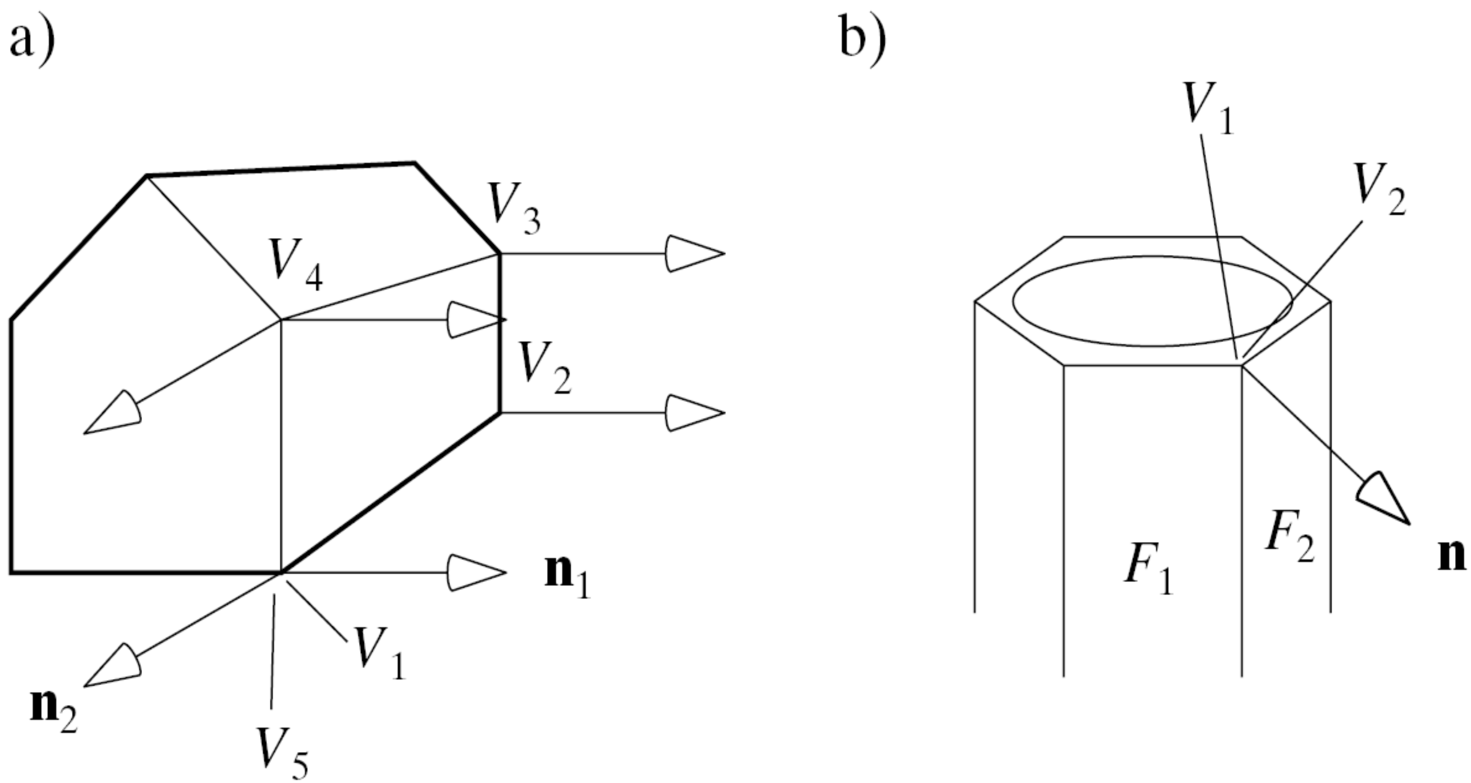
```
glDrawElements(GL_QUADS,24, GL_UNSIGNED_BYTE,
cubeIndices);
```

只需要一次函数调用就绘制出来立方体!!!

顶点法向与面法向

- 在提供多边形网格的顶点及其相连信息的同时，应当同时给出每个面的法向
- 在实际使用时，更有优势的方法是把法向与顶点关联在一起，即点法向
  - 多边形的裁剪算法
  - 明暗处理算法
- OpenGL采用的是点法向

点法向与面法向



法向计算

- 顶点可以由用户输入，但是法向计算不是很直接
  - 有时候法向可以来自于更数学的模型，例如曲面被网格逼近时，可以用原来曲面的法向作为所需要的法向
- 如果需要把某一面显示为平坦的效果，那么只要得到该面所在平面的法向就可以了
- 假设某面上连续三点为V_1, V_2, V_3, 那么$n = (V_1 - V_2) \times (V_3 - V_2)$就是所需要的法向
  - 必要时进行单位化
- 如果多边形不是完全共面，那么所采用的法向不具代表性（因此：不建议使用不共面的多边形，OpenGL不验证共面性）

Martin Newell方法

- **假设各顶点依次为(x_i, y_i, z_i), i = 0, 1, …, N – 1. n = (n_x, n_y, n_z)为所需要确定的法向，则**

$$n_x = \sum_{i=0}^{N-1} (y_i - y_{\text{next}(i)})(z_i + z_{\text{next}(i)})$$

$$n_y = \sum_{i=0}^{N-1} (z_i - z_{\text{next}(i)})(x_i + x_{\text{next}(i)})$$

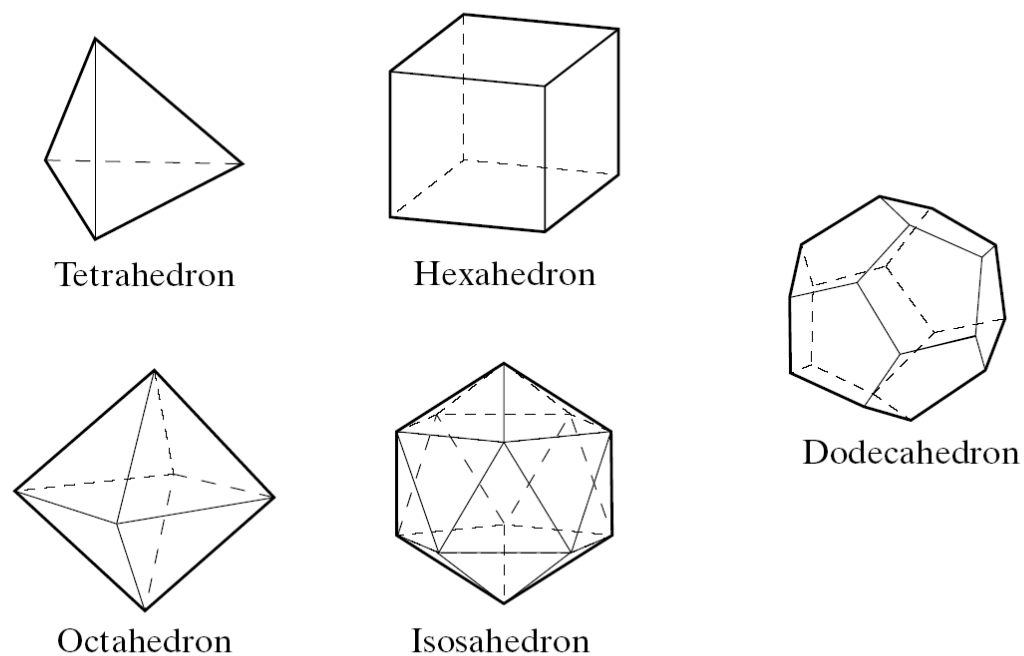
$$n_z = \sum_{i=0}^{N-1} (x_i - x_{\text{next}(i)})(y_i + y_{\text{next}(i)})$$

多面体

- 多面体是由简单表面（平面）构造的连通网格，其形成一个有限体积的封闭实体
  - 多面体的每边都有两个面共享
  - 每个顶点至少有三条边
  - 两个面之间要么无交，要么只在公共边或顶点处相交
- 四面体为多面体，环面为多面体当且仅当各面为平面

正多面体

- 如果多面体的所有面是全等的，而且每个都是正多边形，那么称之为正多面体
- 可以证明只有五种正多面体，称为Plantonic体
- 由Planto（427—347BC）给出，但在此之前就发现了十二面体玩具



Thanks for your attention!