Geometry Representations

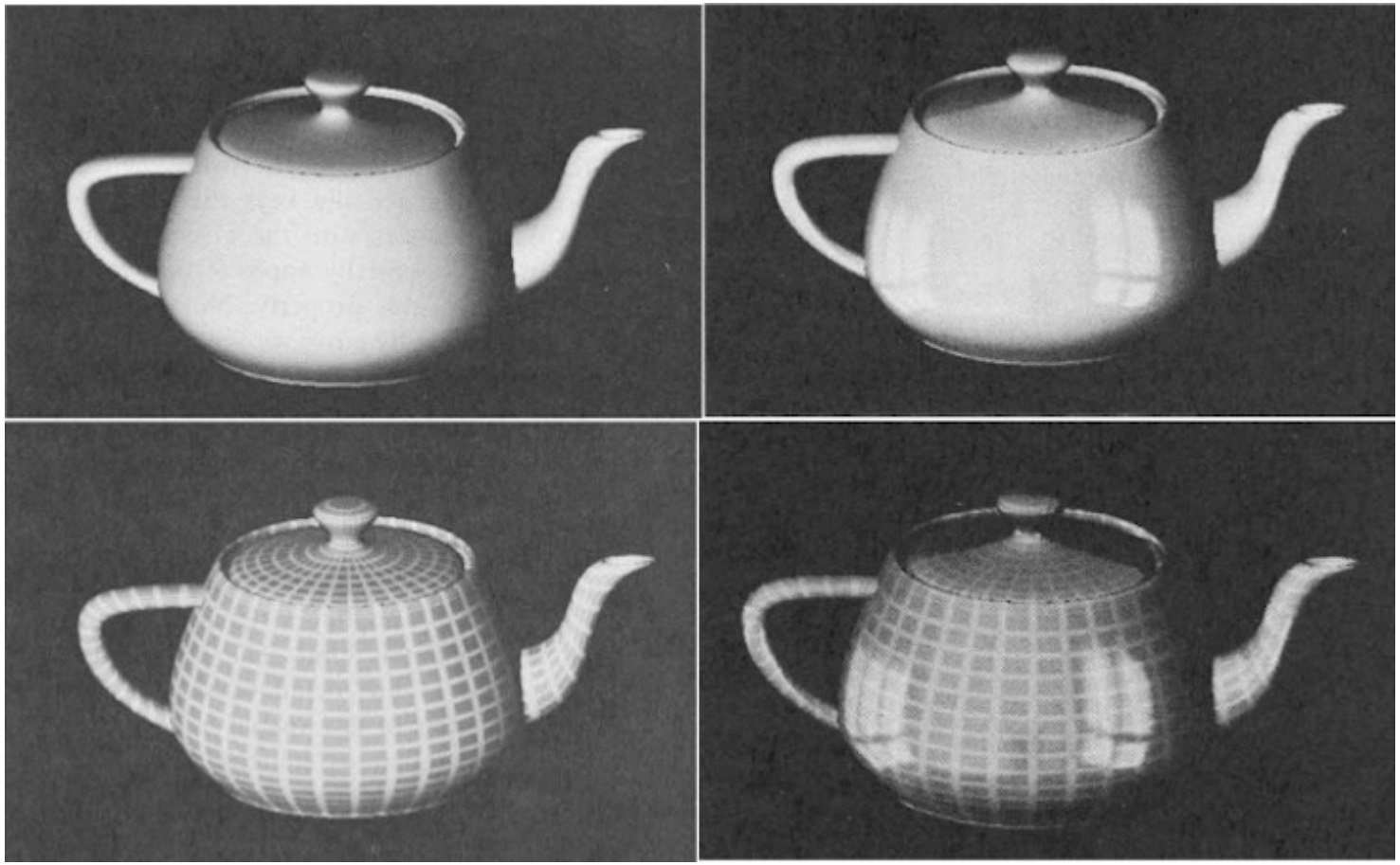
What is geometry?

ge·om·e·try “measurement of earth”

1. The study of shapes, sizes, patterns and positions.
2. The study of spaces where some quantity (lengths, angles, etc.) can be measured.

Examples of geometry – the Utah Teapot

The actual teapot, now displayed at
the Computer History Museum
in Mountain View, California



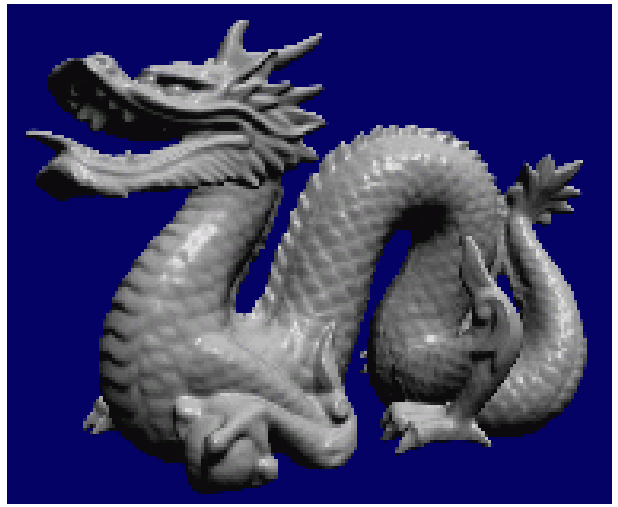
Martin Newell's early teapot renderings
(Martin created teapot model in 1975 using Bezier curves)

Examples of geometry – The Stanford Bunny

Mesh created by reconstruction from laser scans

Photograph of the scanned statue (purchased by Greg Turk at a store on University Ave in 1994)

The Stanford 3D Scanning Repository



http://graphics.stanford.edu/data/3Dscanrep/

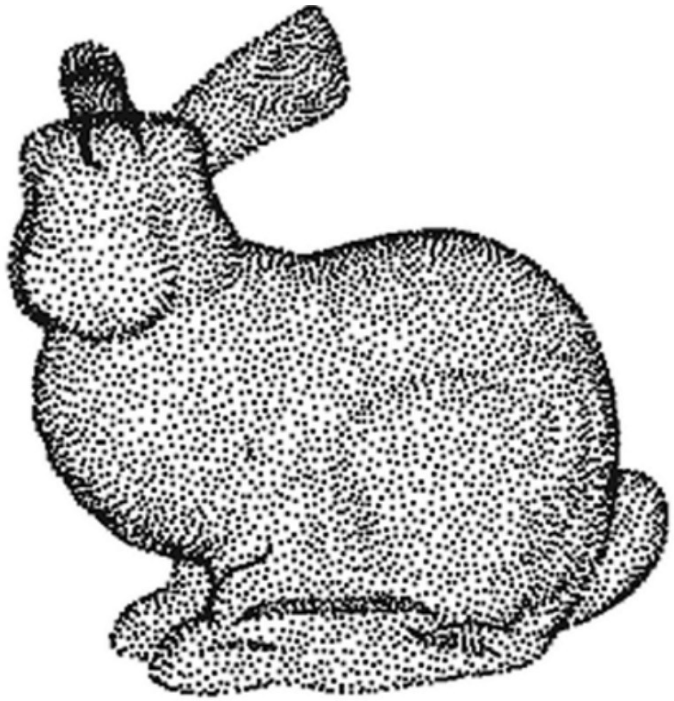
How to encode geometry on a computer?

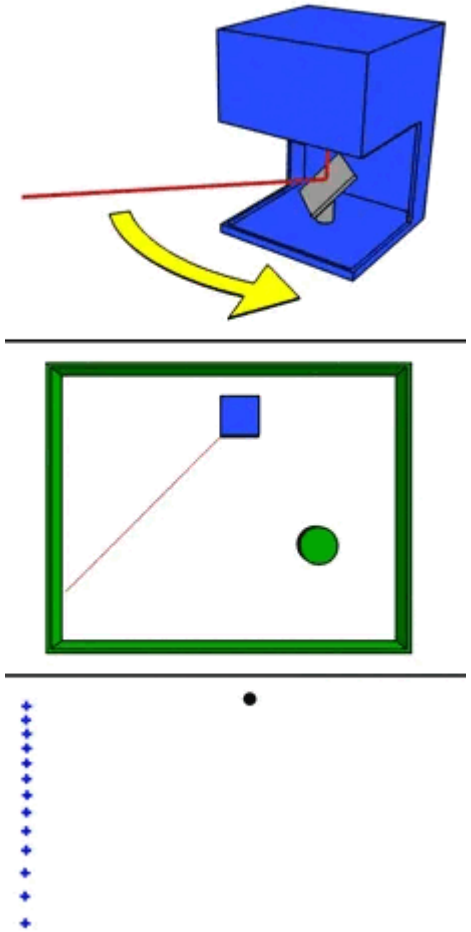
- Explicit
  - Point cloud
  - Polygon mesh
  - Subdivision surface
  - …
- Implicit
  - Level set
  - Algebraic surface
  - …

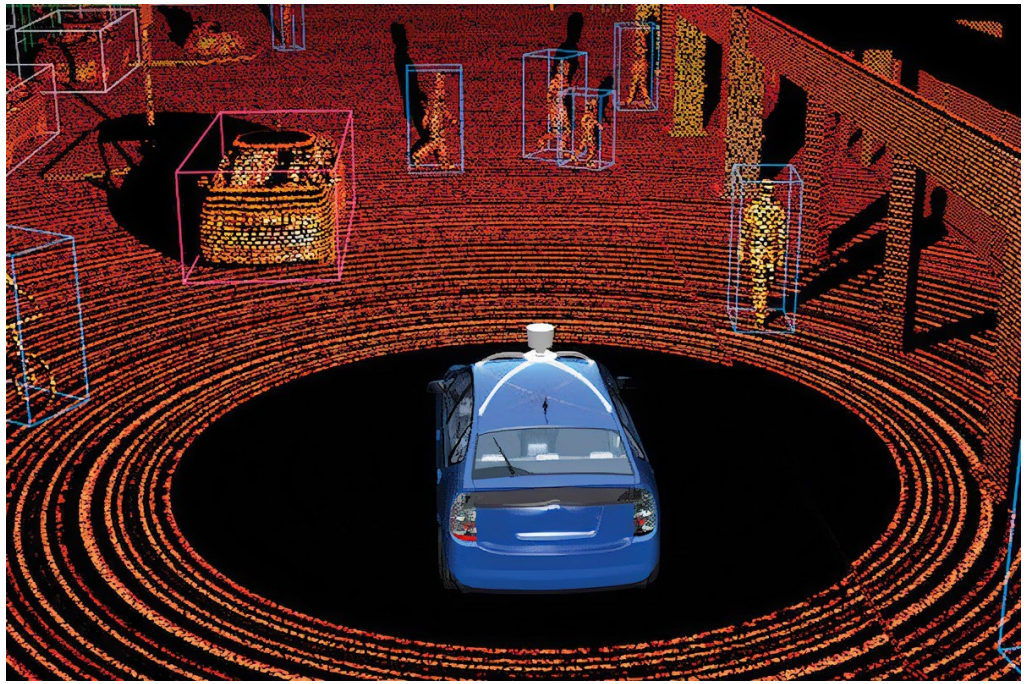
- Each choice best suited to a different task/type of geometry

Point cloud

- Scanned by LiDAR or other scanners
- Points on object surface

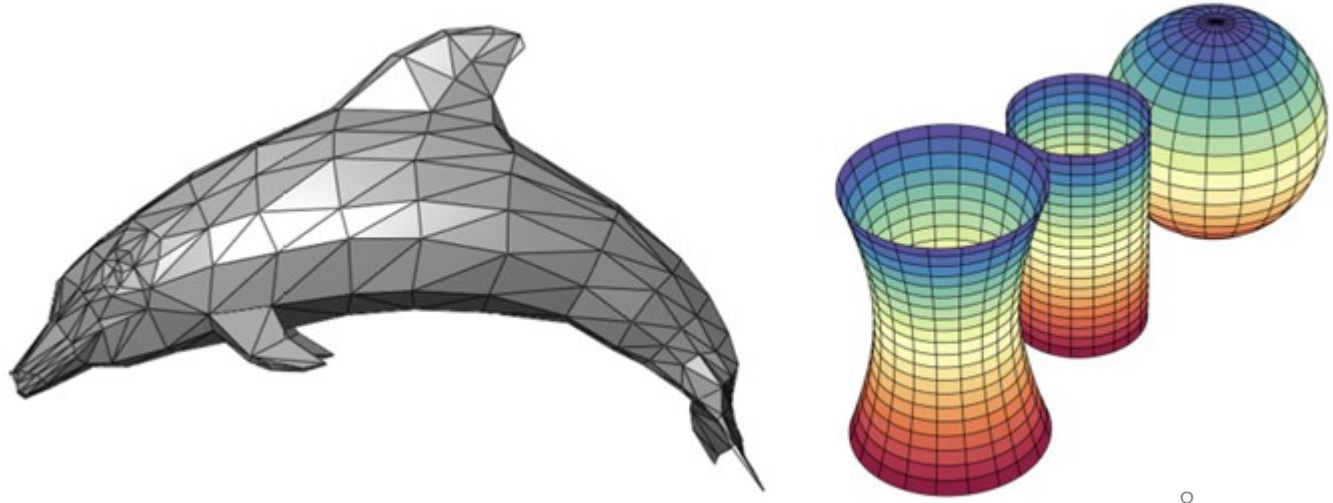






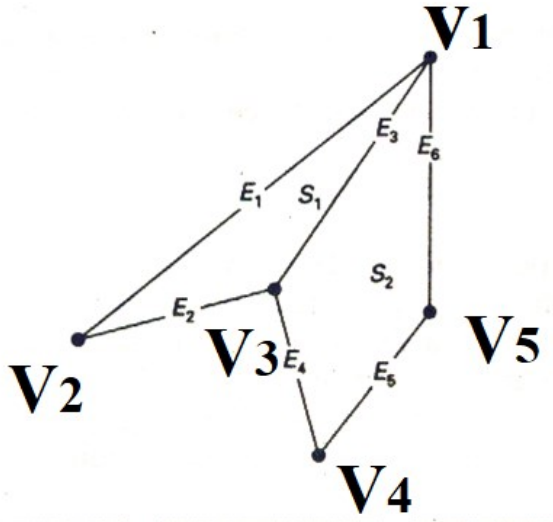
Polygon mesh

- Store vertices and polygons (triangles and quads most often)
- Perhaps most common representation in graphics
- Easy (?) to do processing, simulation and adaptive sampling
- More complicated data structures



Polygon mesh

- Vertex table, edge table and polygon-surface table
- BTW, Euler's theorem on *polyhedrons*: υ − *e* + *f* = 2

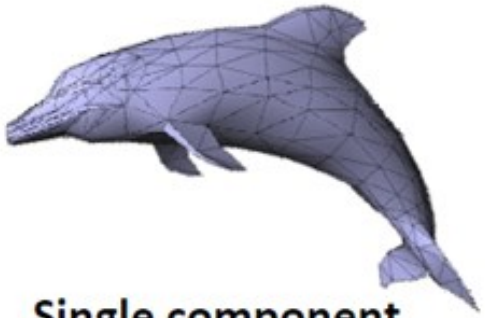


VERTEX TABLE	
V_1:	x_1, y_1, z_1
V_2:	x_2, y_2, z_2
V_3:	x_3, y_3, z_3
V_4:	x_4, y_4, z_4
V_5:	x_5, y_5, z_5

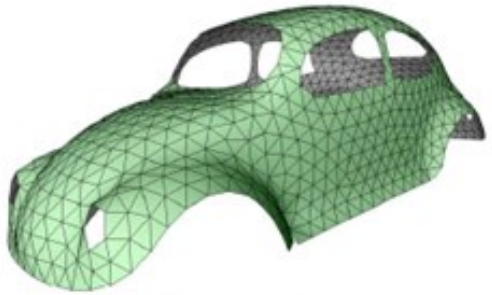
EDGE TABLE	
E_1:	V_1, V_2
E_2:	V_2, V_3
E_3:	V_3, V_1
E_4:	V_3, V_4
E_5:	V_4, V_5
E_6:	V_5, V_1

POLYGON-SURFACE TABLE	
S_1:	E_1, E_2, E_3
S_2:	E_3, E_4, E_5, E_6

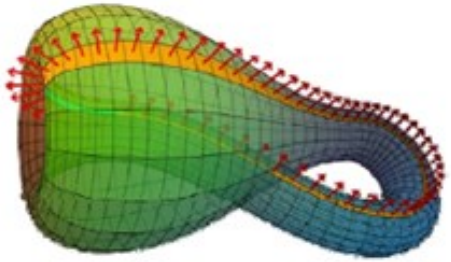
Examples of polygon mesh



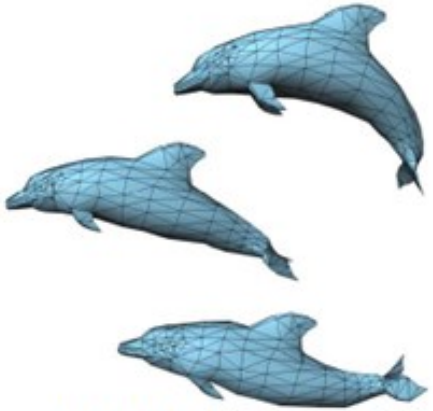
Single component, closed, triangular, orientable manifold



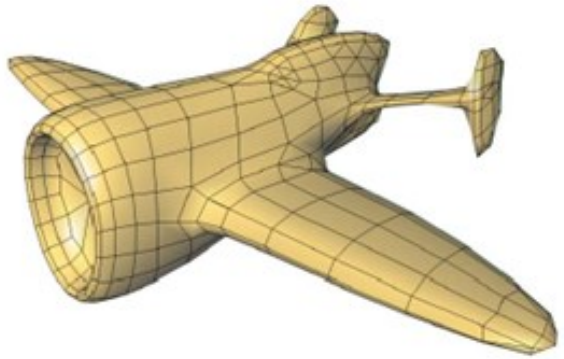
With boundaries



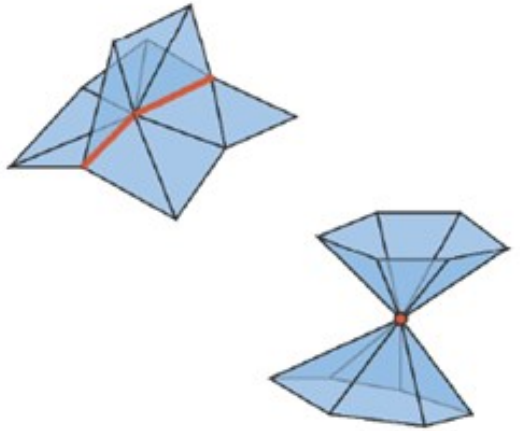
Not orientable



Multiple components



Not only triangles



Non manifold

Triangle mesh

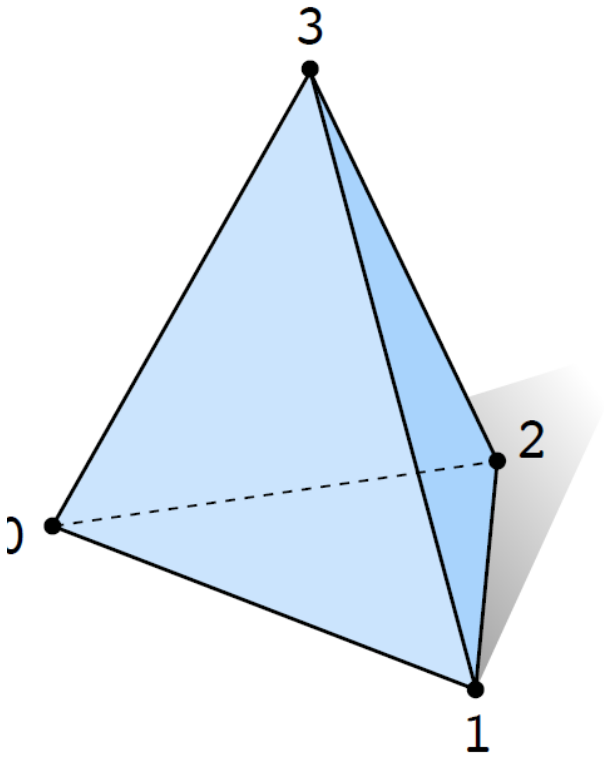
- Store vertices as triples of coordinates (x, y, z)
- Store triangles as triples of indices (i, j, k)
- E.g., for a tetrahedron

VERTICES

	x	y	z
0:	−1	−1	−1
1:	1	−1	1
2:	1	1	−1
3:	−1	1	1

TRIANGLES

	i	j	k
	0	2	1
	0	3	2
	3	0	1
	3	1	2



Tetrahedron (Wavefront OBJ)

```obj
# Tetrahedron with specified face order

# List of vertices (v x y z)
v -1 -1 -1  # Vertex 1
v  1 -1  1  # Vertex 2
v  1  1 -1  # Vertex 3
v -1  1  1  # Vertex 4

# List of faces (f vertex1 vertex2 vertex3)
f 1 3 2  # Face 1
f 1 4 3  # Face 2
f 4 1 2  # Face 3
f 4 2 3  # Face 4
```

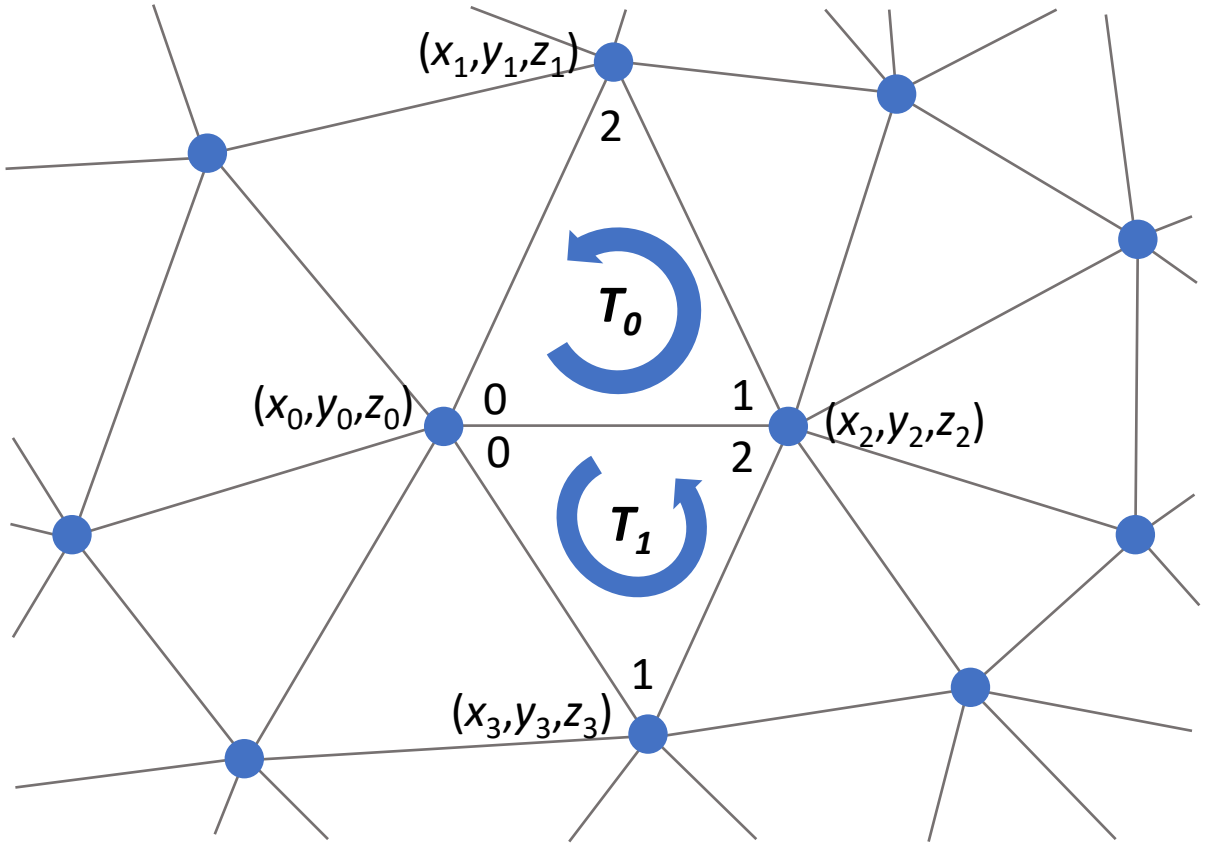
Polygon mesh

- Triangle Meshes
- Quad Meshes
- Subdivision Surfaces
- Mesh Parameterization

Triangle mesh representation, Separate triangles

- List of triangles (triangle soup)

	[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
	⋮	⋮	⋮

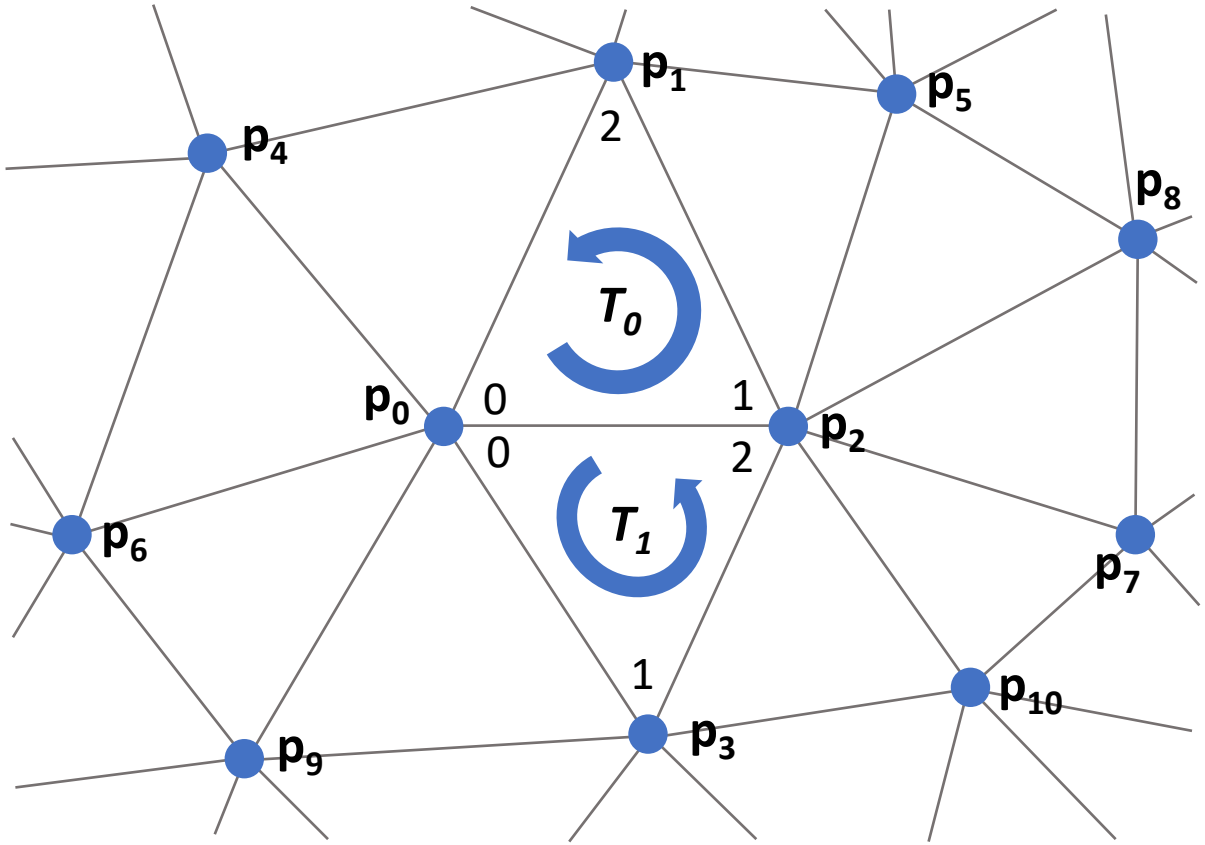


Triangle mesh representation, Indexed triangle set

- List of vertices / 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
	⋮

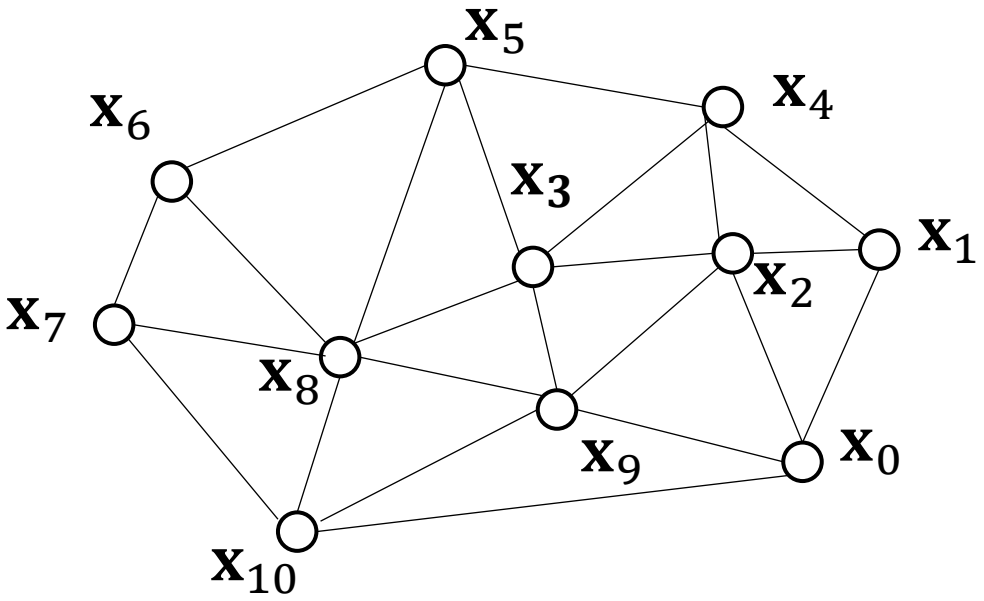


Comparison

- List of triangles
  - GOOD: simple
  - BAD: contains redundant vertex information
- List of vertices + List of indexed triangles
  - GOOD: sharing vertex position information, reduces memory usage
  - GOOD: ensure integrity of the mesh (changing a vertex's position in 3D space causes that vertex in all the polygons to move)
  - BAD: more complex
- E.g., How to find triangle neighbors?

Triangle mesh representation, Indexed triangle set

- E.g., How to find triangle neighbors?



represented by

Vertex Positions $\mathbf{x}$

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_0 \\ \vdots \\ \mathbf{x}_i \\ \vdots \\ \mathbf{x}_{10} \end{bmatrix} \in \mathbf{R}^{33}$$

Triangle list: {(0,1,2),(2,1,4),(2,4,3), ⋯ } (index triples)

Triangle Edge list: {{0,1,0} {1,2,0} {0,2,0}; {1,2,1}, {1,4,1}, {2,4,1}, {1,4,2} ⋯}

Sort and remove

Edge list: {{0,1,0} {0,2,0}, {0,2,4},{0,9,4}, {0,9,5},{0,10,5} ⋯
{1,2,0} {1,2,1}, {1,4,1}, {2,3,2}, {2,3,3}, {2,4,1}, {2,4,2} ⋯}

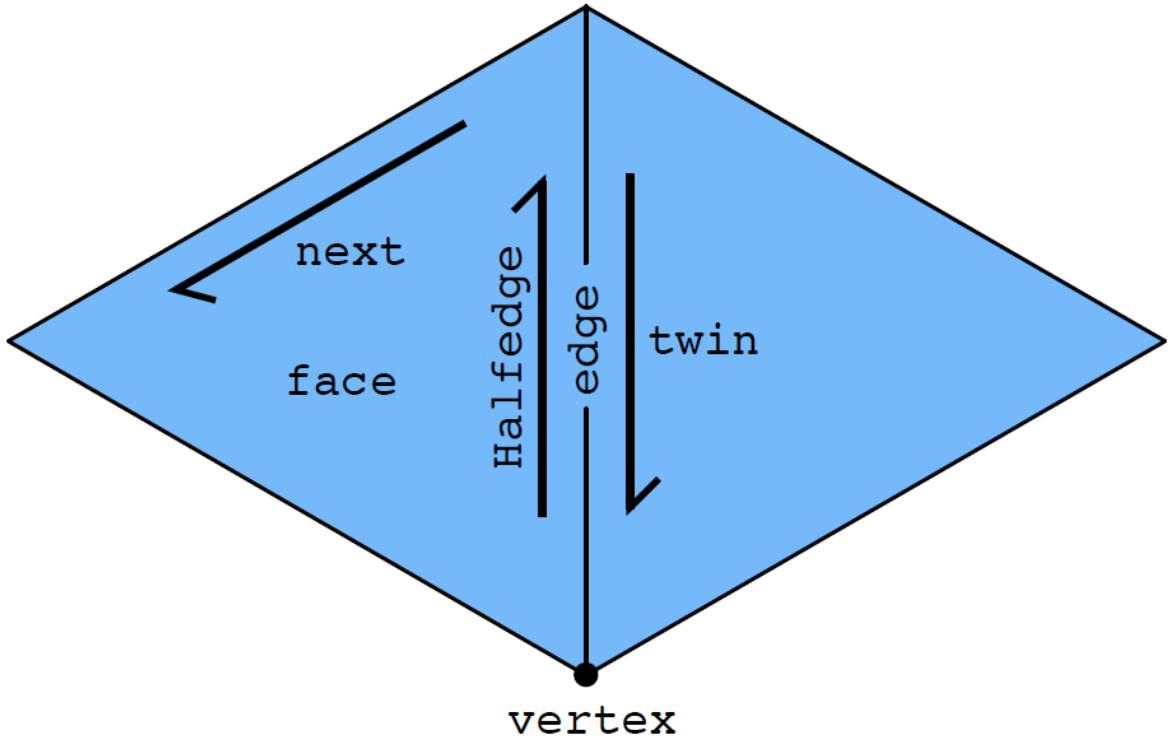
Edge list: {{0,1}, {0,2}, {0,9}, {0,10}, {1,2}, {1,3}, {1,4},⋯}

Neighboring triangle list: {{0, 1}, {1, 2}}

How to find vertex neighbors?

Half-edge data structure

- Key idea: two half-edges act as “glue” between mesh elements
- Each vertex, edge and face points to one of its half edges

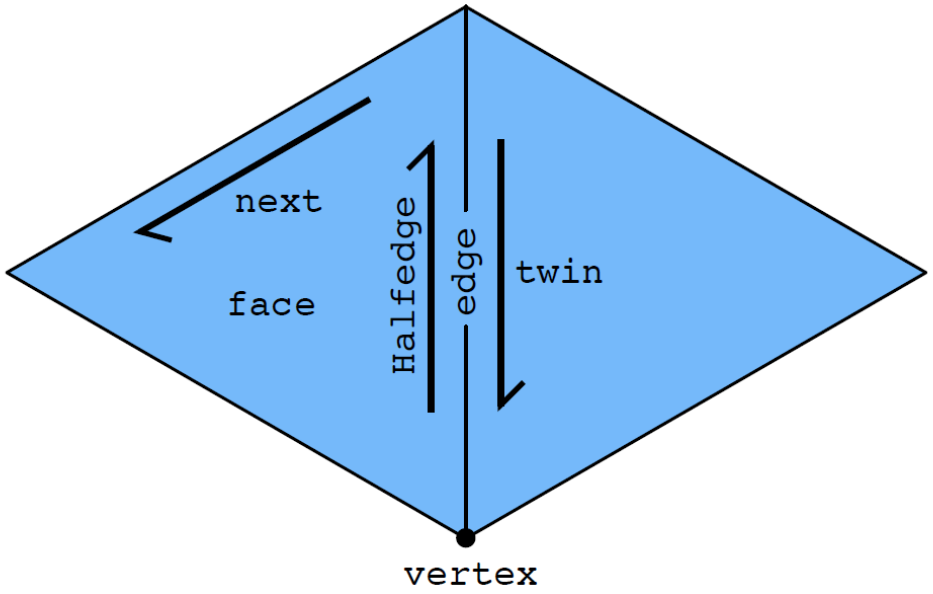


https://jerryyin.info/geometry-processing-algorithms/half-edge/

Half-edge data structure

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

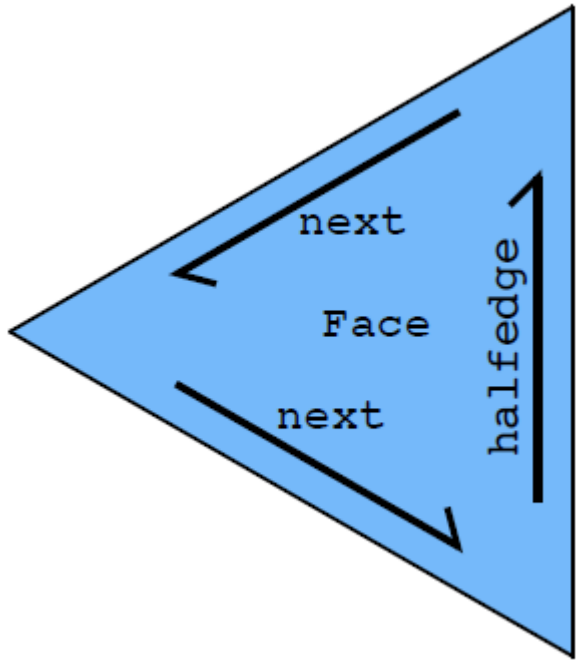
```
struct Vertex
        Vec3 pos;
        Halfedge *halfedge;
struct Edge
        Halfedge *halfedge;
Struct Face
        Halfedge *halfedge;
```



Half-edge data structure

- Example 1: process all vertices of a face

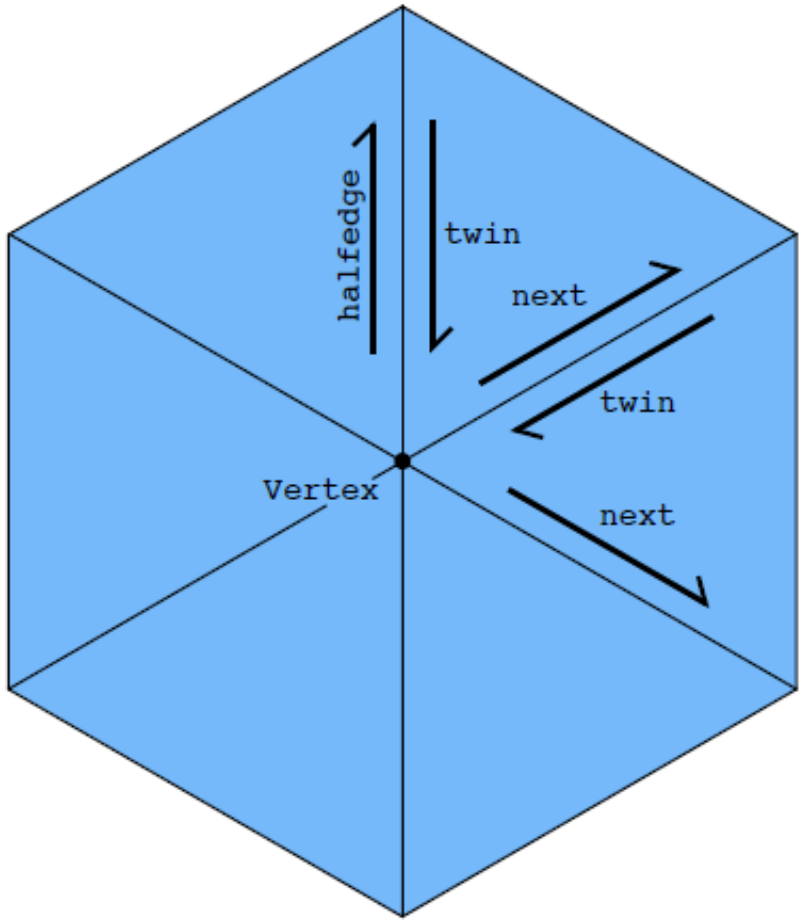
```
Halfedge* h = f-> halfedge;

do

        do_work (h->vertex);

        h = h->next;

while (h ! = f-> halfedge);
```



Half-edge data structure

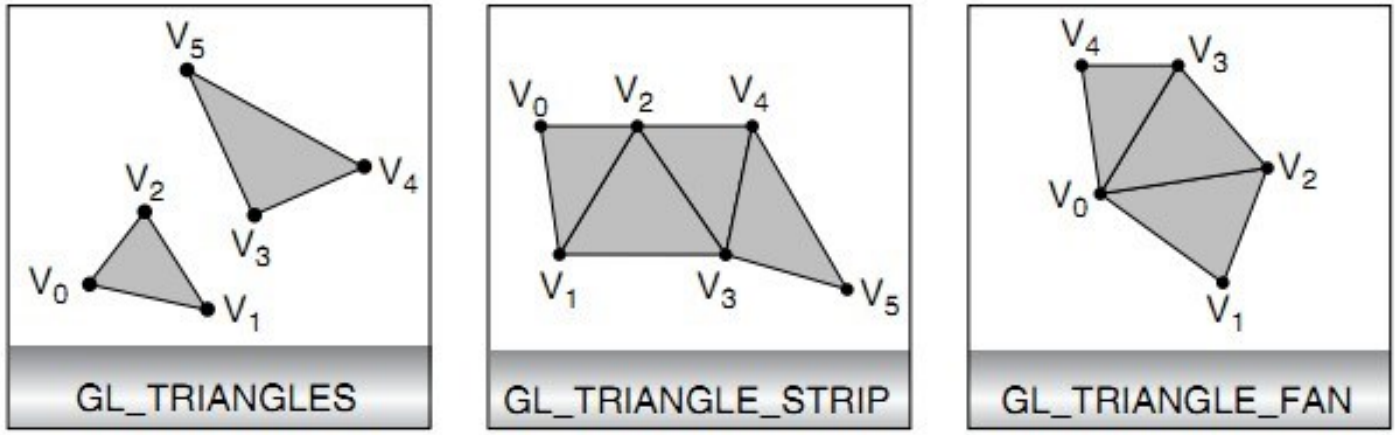
- Example 2: process all edges around a vertex

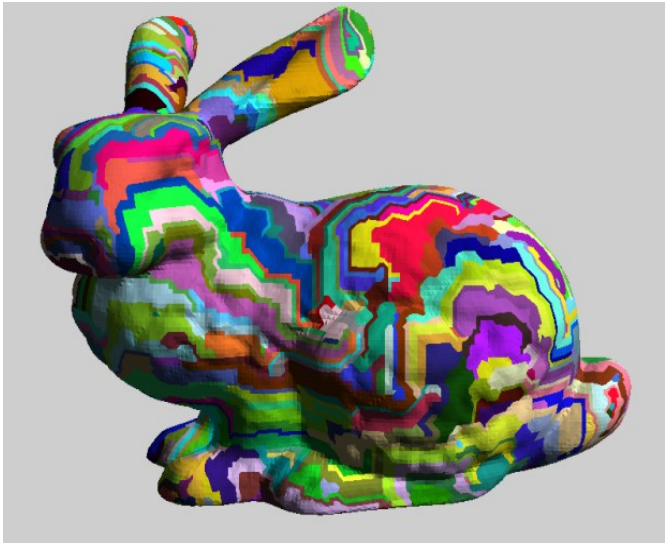
```
Halfedge* h = v-> halfedge;

do

        do_work (h->edge);

        h = h->twin->next;

while (h ! = v-> halfedge);
```



Other data structures and properties for triangles

- How to be more efficient?
- E.g., triangle strips



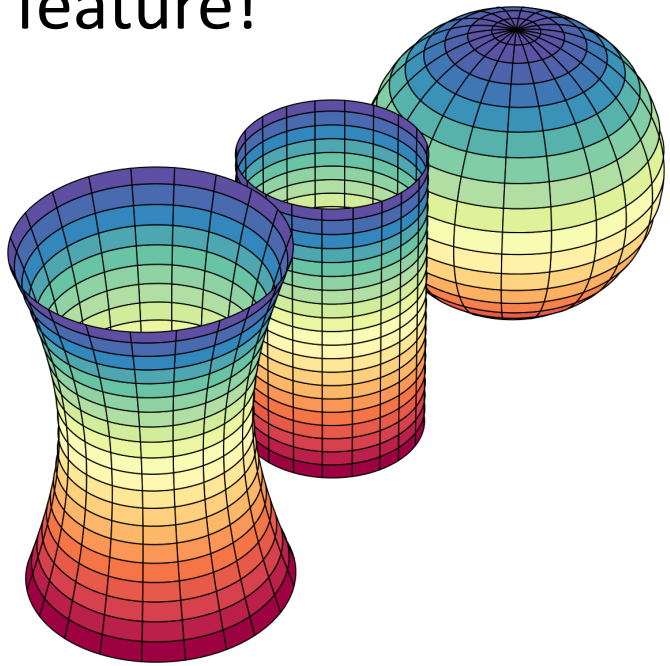


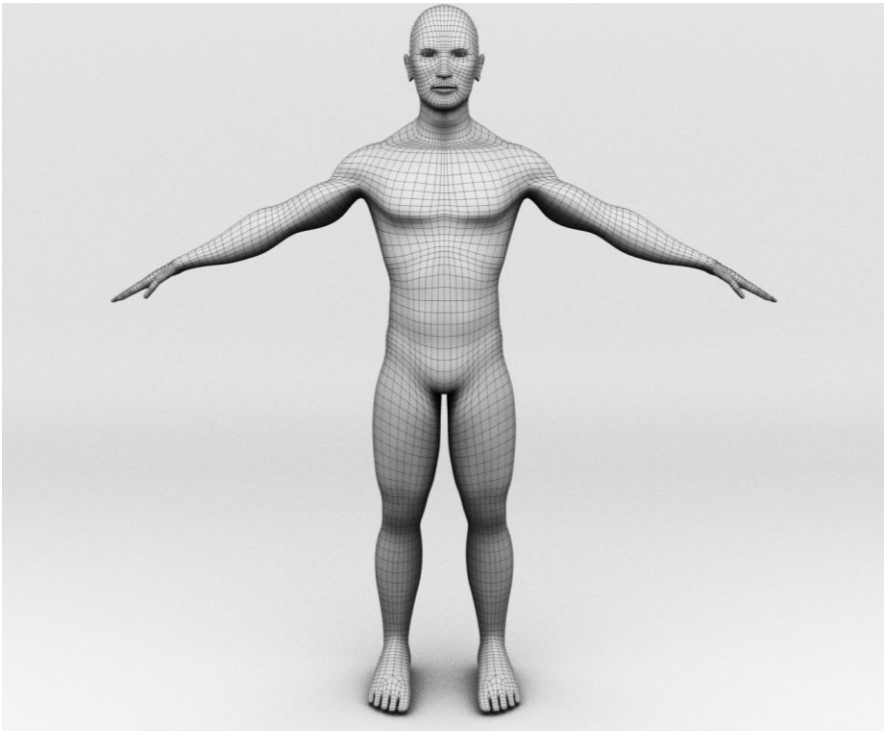
- Not as popular as indexed triangle sets

Quad mesh

- Nice regularity, easy to store
- Easy to parameterize, good feature!

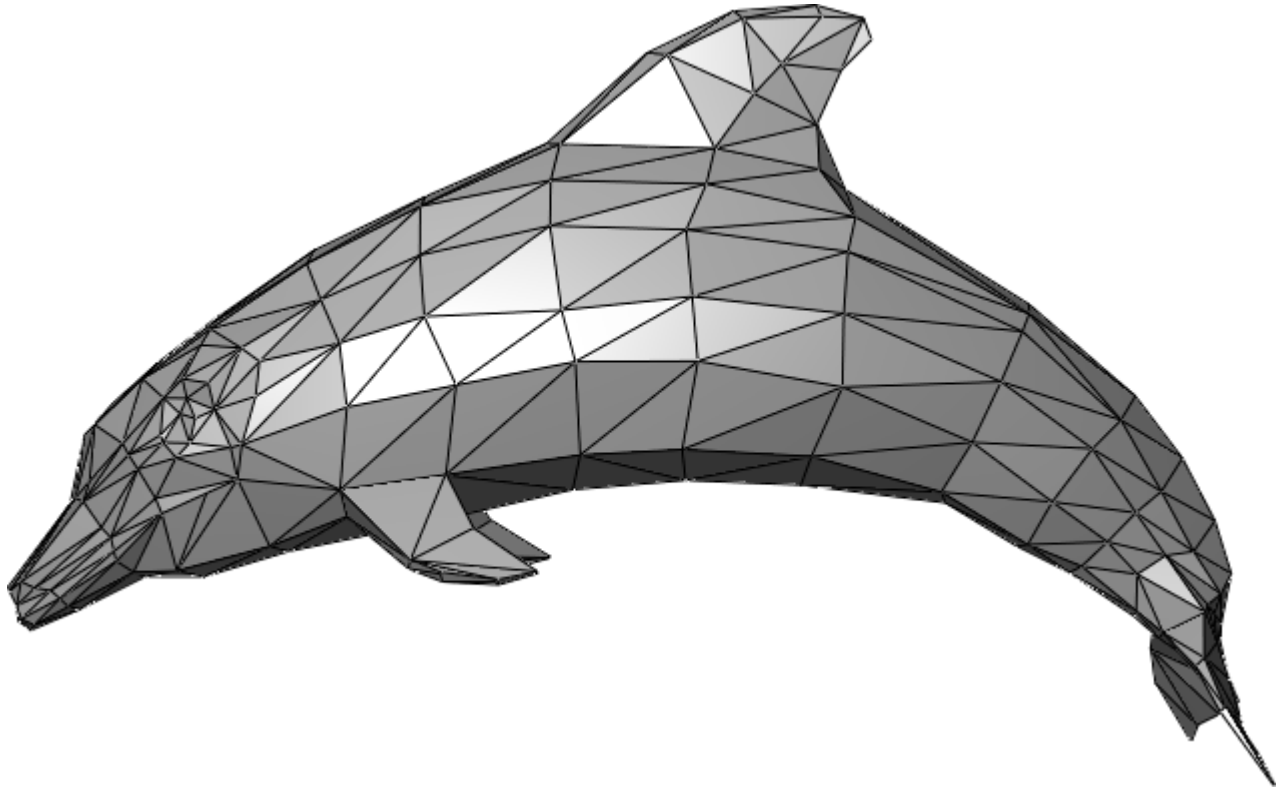
  (e.g., texture mapping)

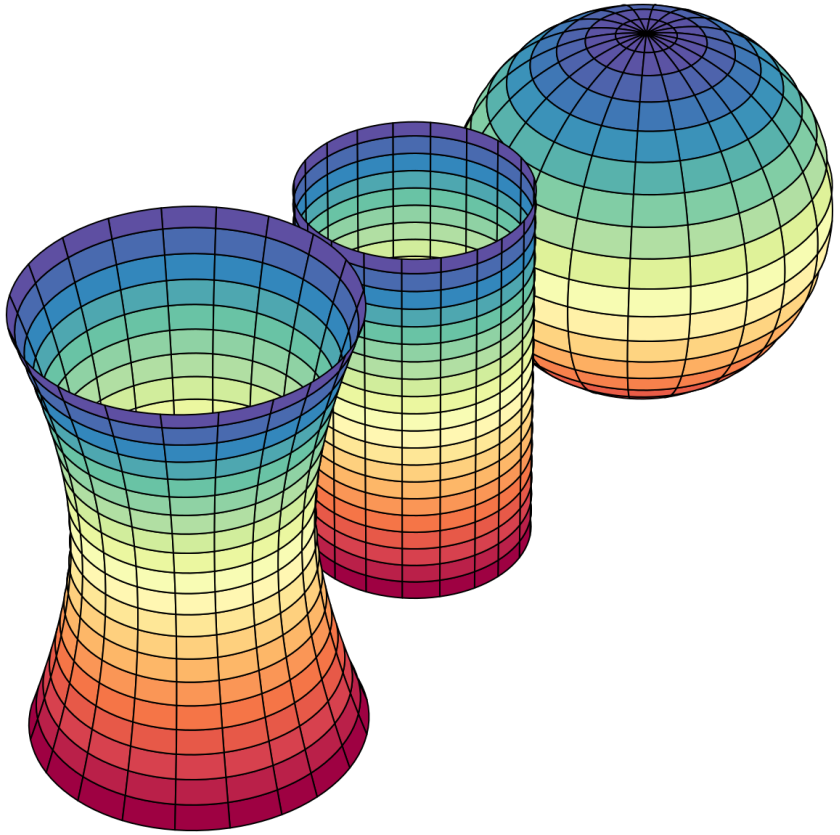




Pros and Cons of polygon surface

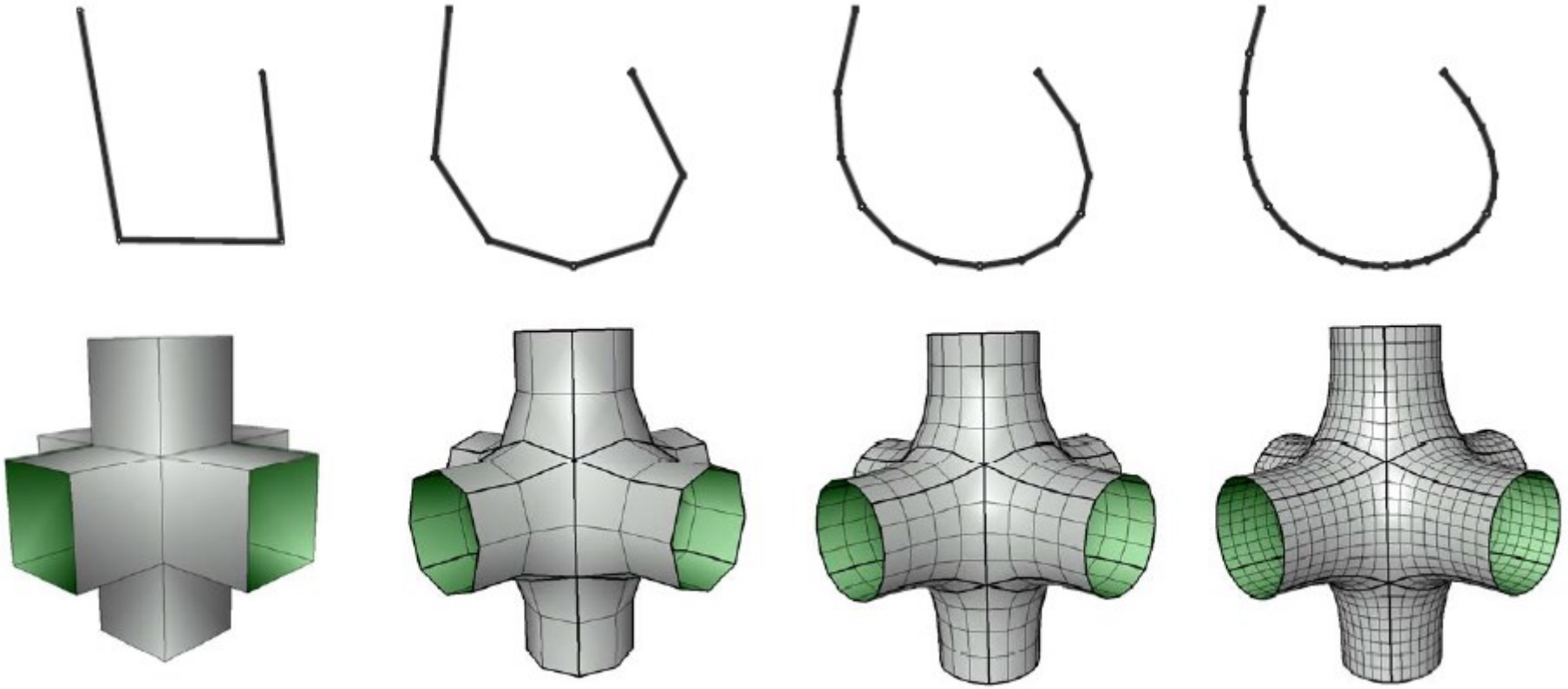
- GOOD: Simple, fast in rendering and processing
- BAD: Need much more surfaces to present smooth curved surfaces



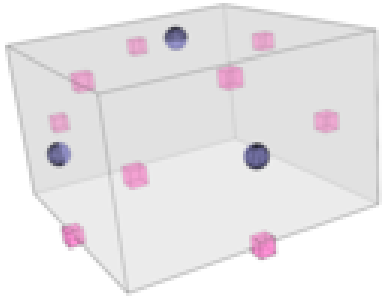


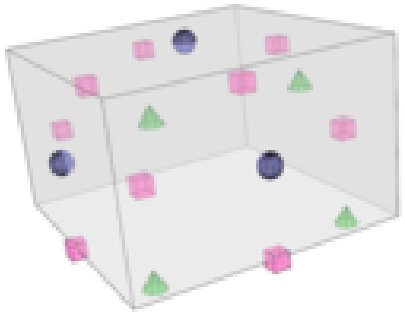
Subdivision surface

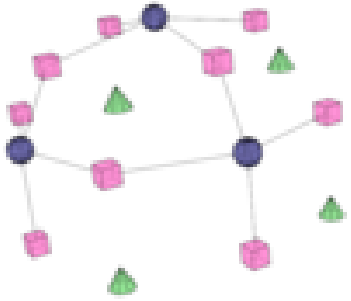
- Subdivision defines a smooth curve or surface as the limit of a sequence of successive refinements

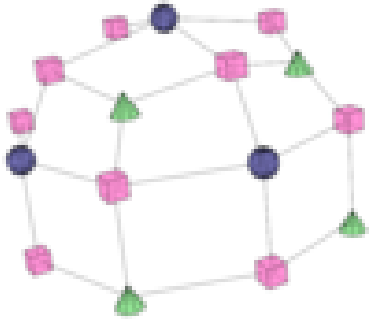


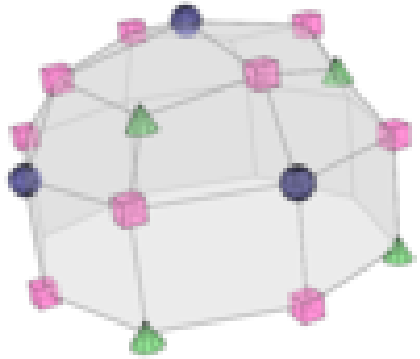
Catmull-Clark vertex update rules



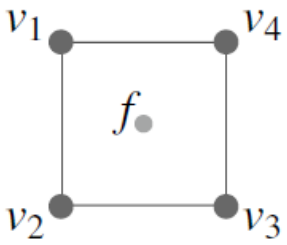








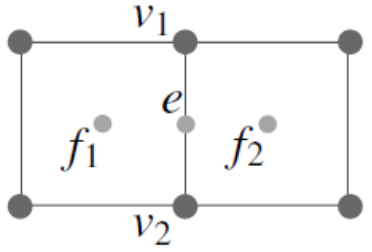
Face point

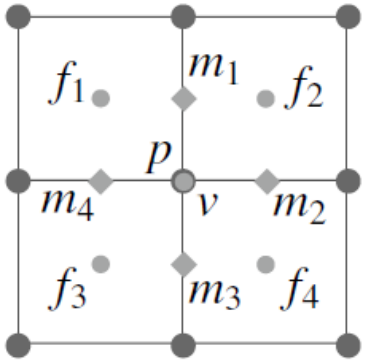


$$f = \frac{v_1 + v_2 + v_3 + v_4}{4}$$

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

Edge point





Vertex point

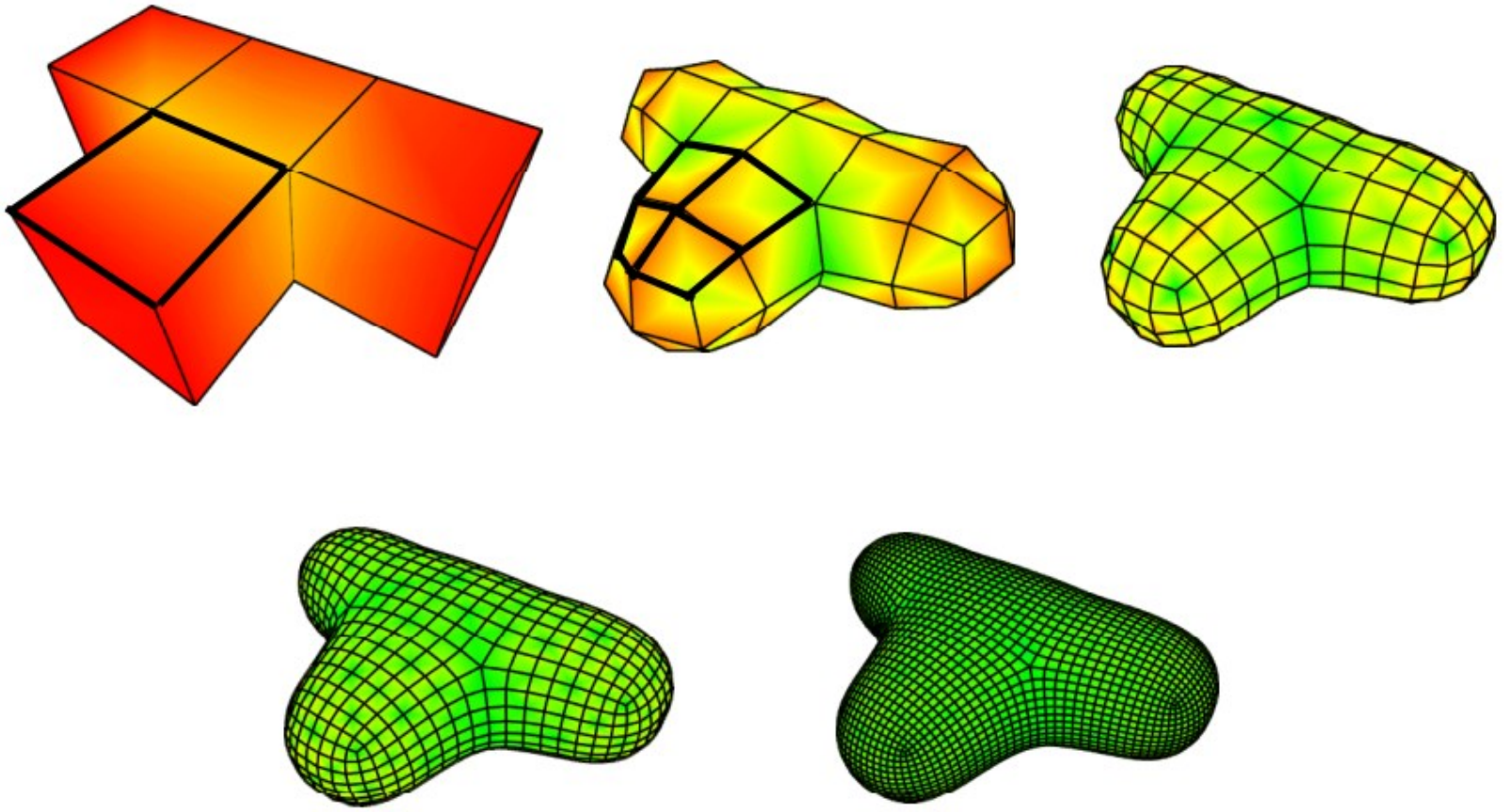
$$v = \frac{1}{n}\left[\frac{f_1 + f_2 + f_3 + f_4}{4} + 2\frac{m_1 + m_2 + m_3 + m_4}{4} + (n-3)p\right]$$

m midpoint of edge, not "edge point"

p old "vertex point"

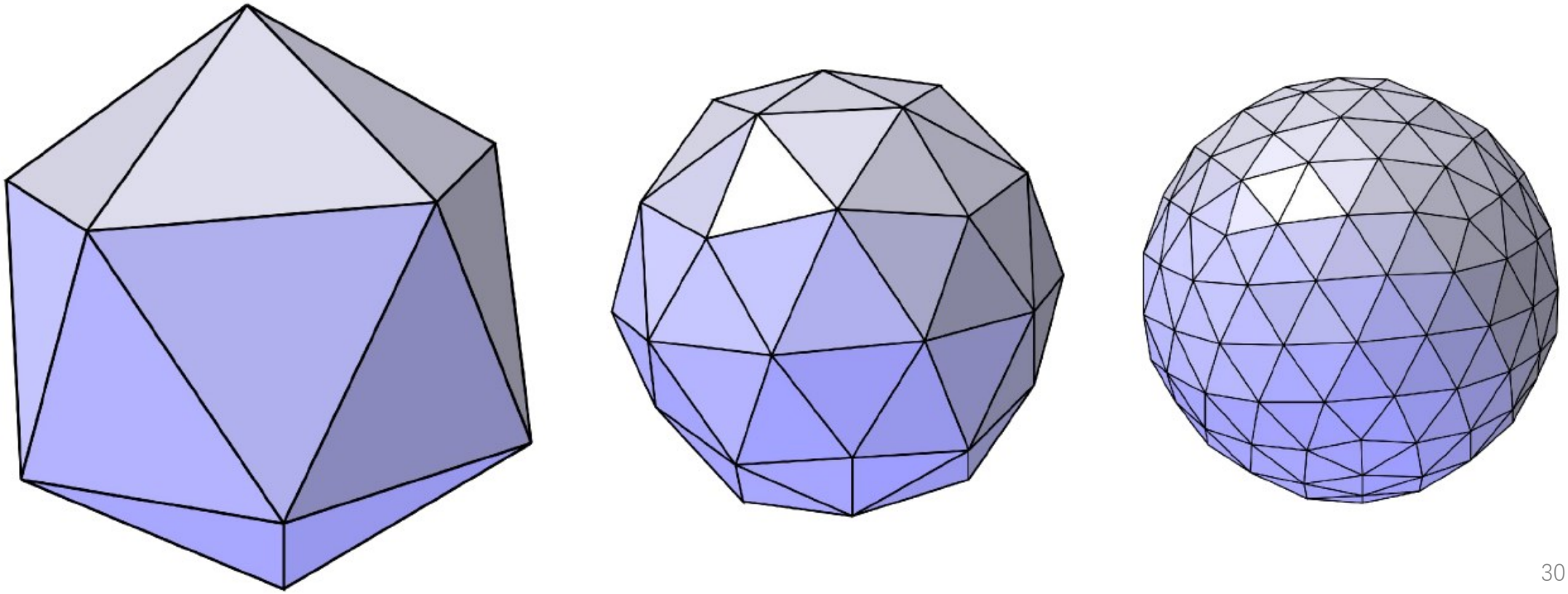
1. Add new face point, new edge point
2. Update Vertex point (n: num of neighbor faces/edges)
3. Connect points to form new Edges
4. Store new Faces (1 old face -> 4 new)

Catmull-Clark Subdivision



Loop subdivision

- Common subdivision rule for triangle meshes

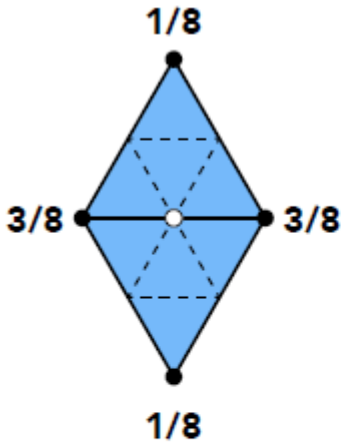


Loop subdivision algorithm

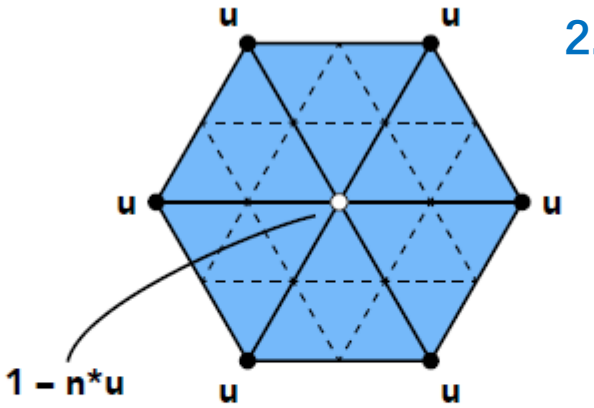
- Split each triangle into four by adding new vertex on each edge
- Update old vertex positions using weighted sum of prior vertex positions

1. Add New Vertex:

- Two neighbor faces: $\frac{3}{8}(v_0 + v_2) + \frac{1}{8}(v_1 + v_3)$
- One neighbor faces: $\frac{v_0+v_1}{2}$



New vertices
(weighted sum of vertices on split edge, and vertices "across from" edge)



Old vertices
(weighted sum of edge adjacent vertices)

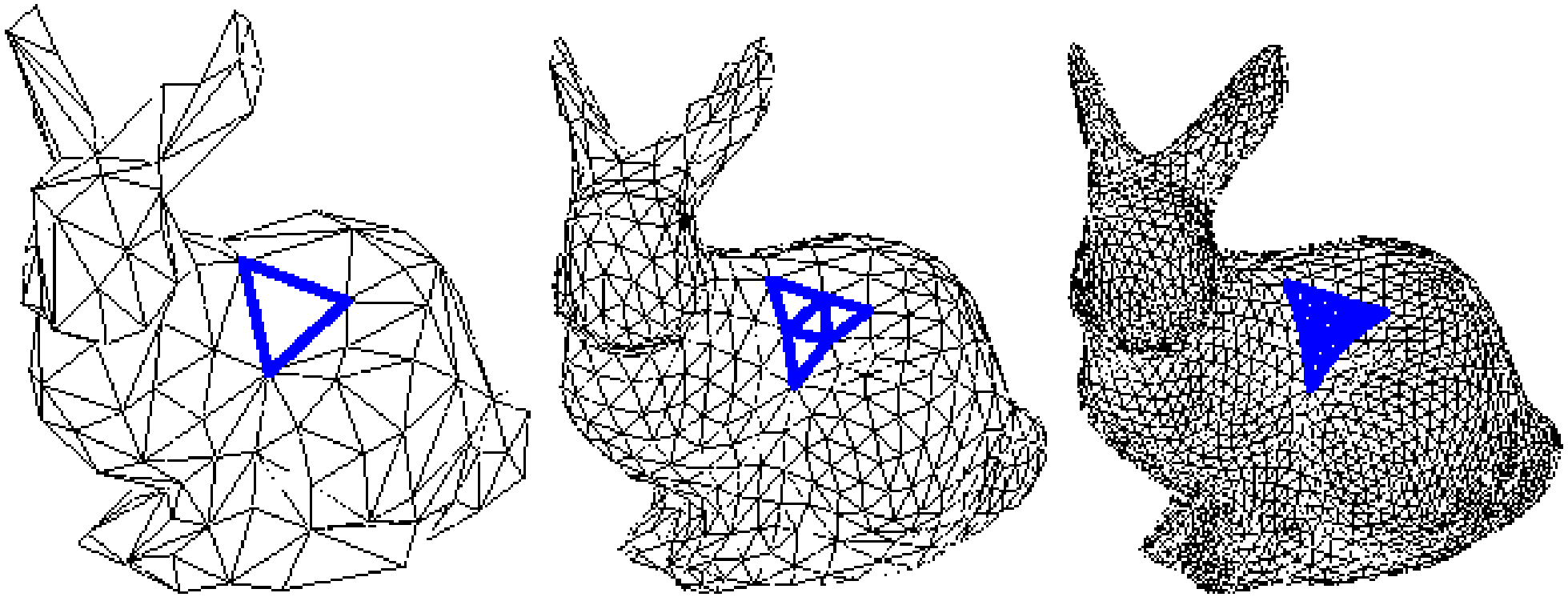
2. Update Old Vertex:

$$\mathbf{v}' = (1 - n * u)\mathbf{v} + \sum_{i=1}^{n} u\mathbf{v}_i$$

n = vertex degree

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

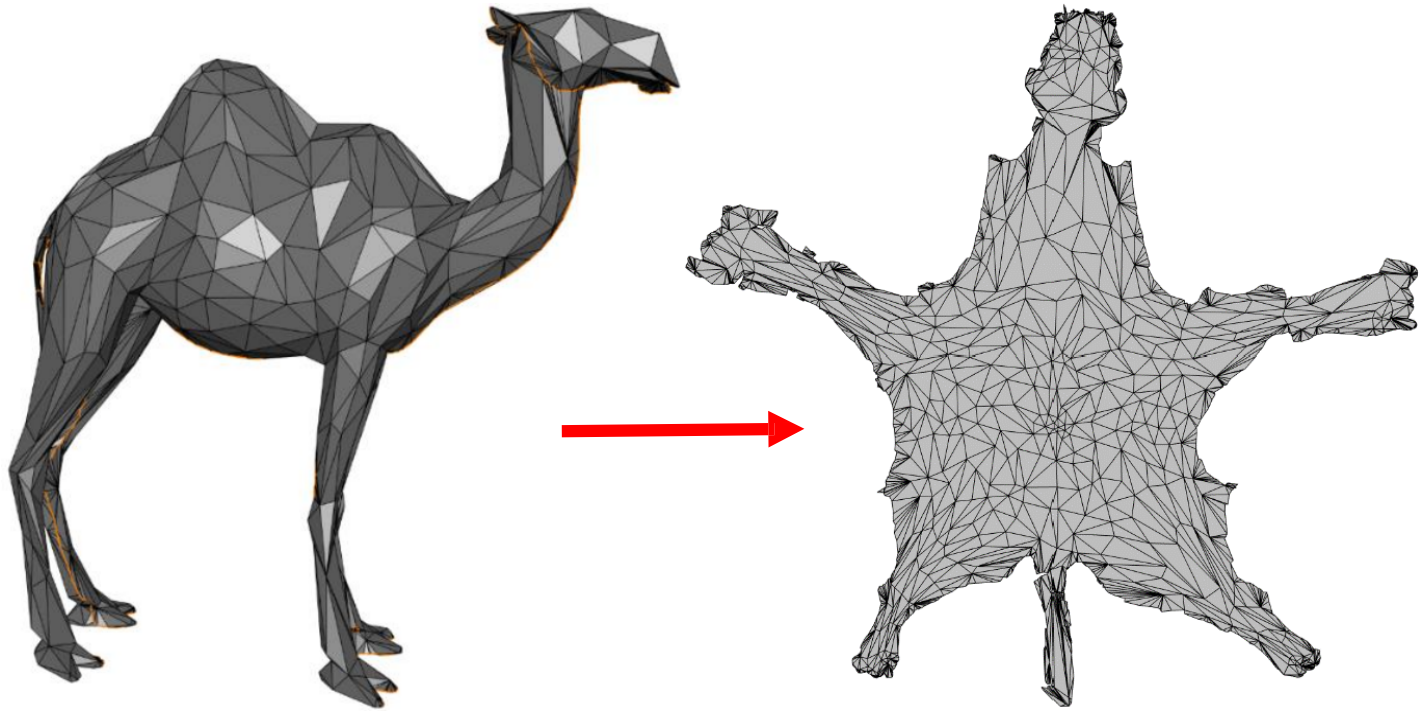
Loop subdivision algorithm



Mesh Parameterization

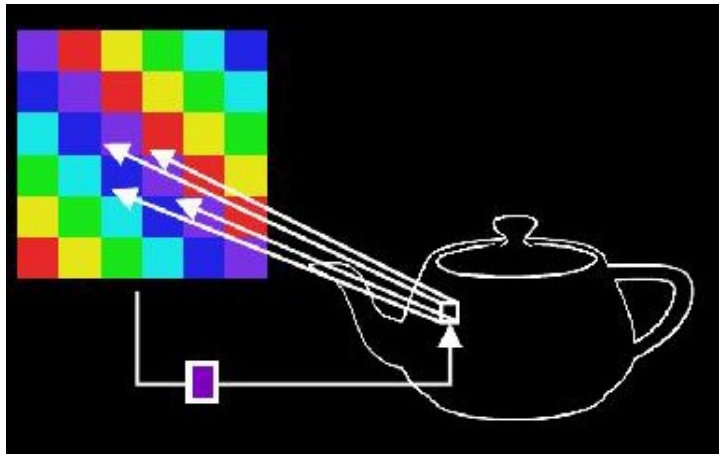
- A function that puts input surface in **one-to-one** correspondence with a 2D domain.

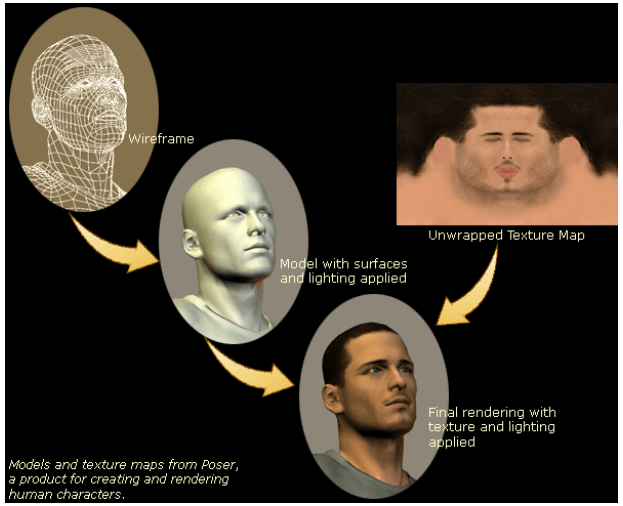
- Parameterization of a Triangulated Surface
  - all (u_i, v_i) coordinates associated with each vertex $\boldsymbol{v}_i = \quad x_i, y_i, z_i$



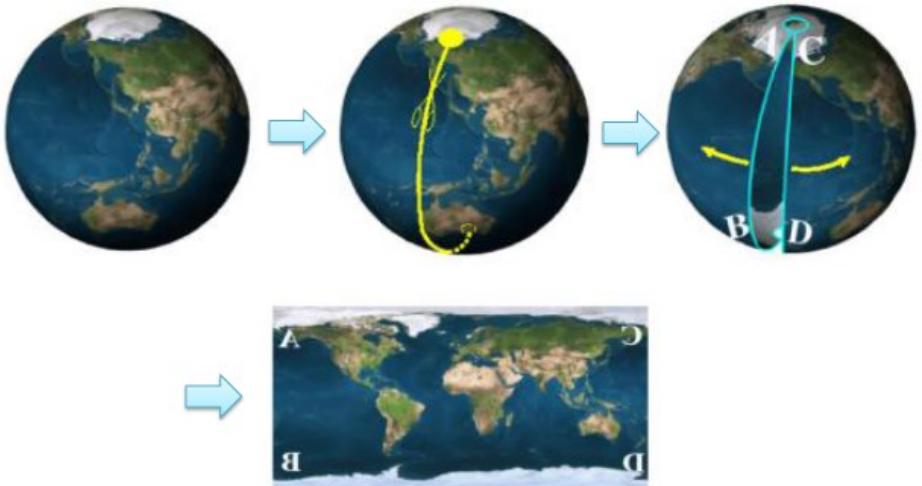
Applications of Mesh Parameterization

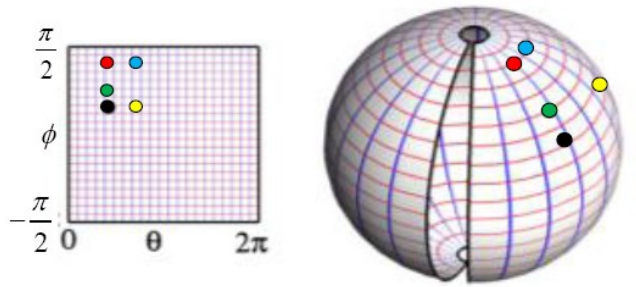
- Texture mapping





- Word Map

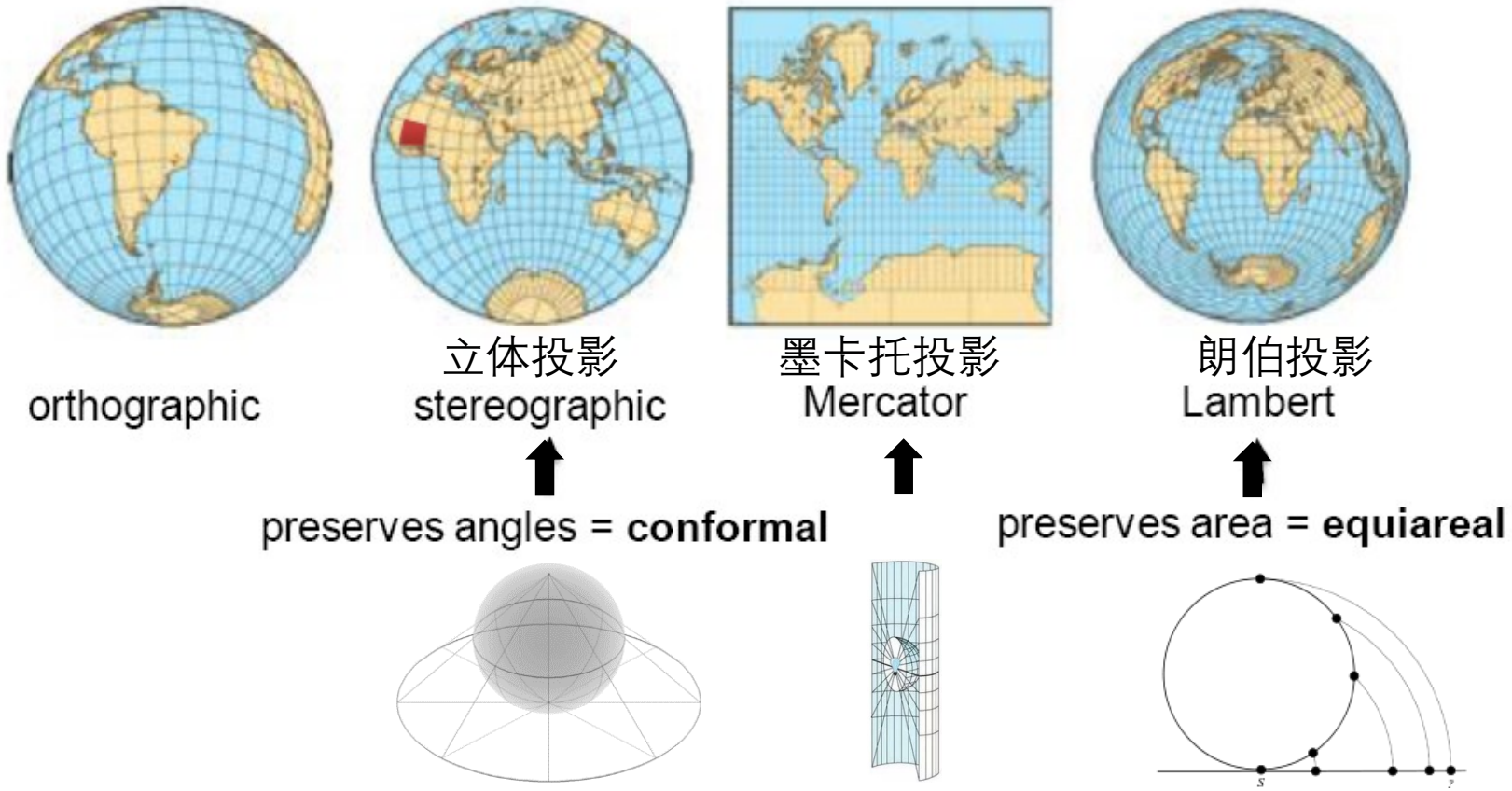




$$x(\theta,\phi) = R\cos\theta\cos\phi$$
$$y(\theta,\phi) = R\sin\theta\cos\phi$$
$$z(\theta,\phi) = R\sin\phi$$

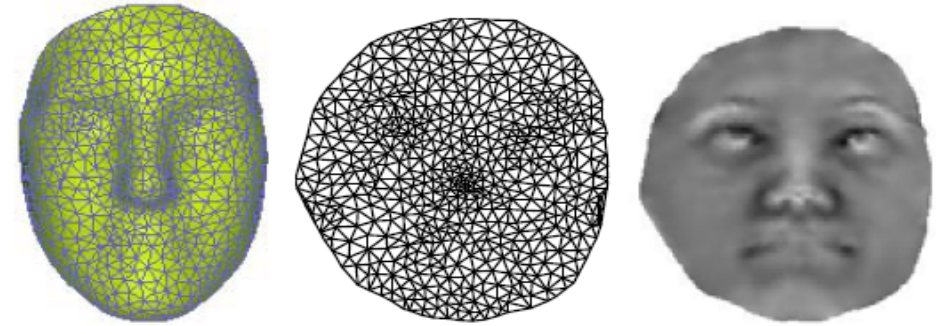
Mesh Parameterization

- World Maps

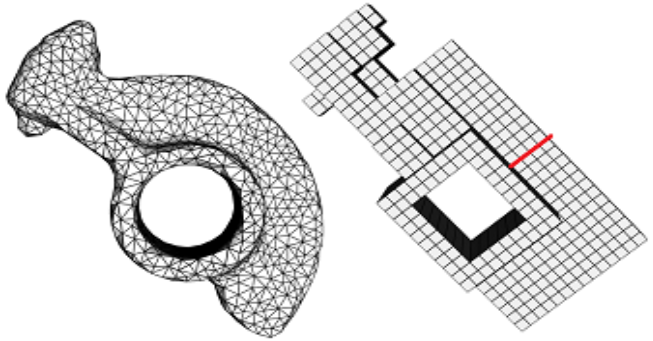


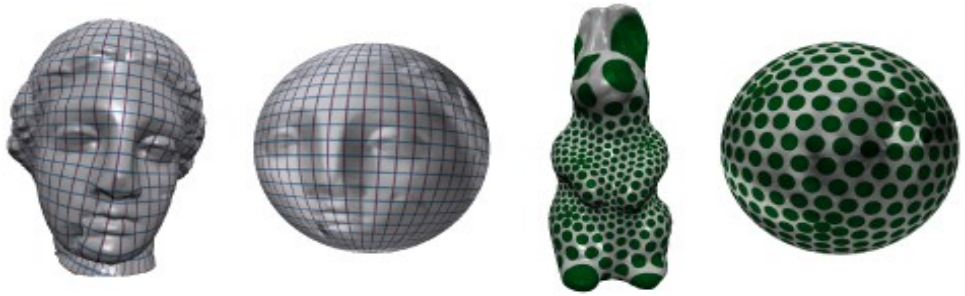
Mesh Parameterization

- Different parameterization domains:
  - Planar Parameterization （平面参数化）
  - Spherical Parameterization （球面参数化）
  - simplicial parameterization （基域参数化）
  - ……..



Planar



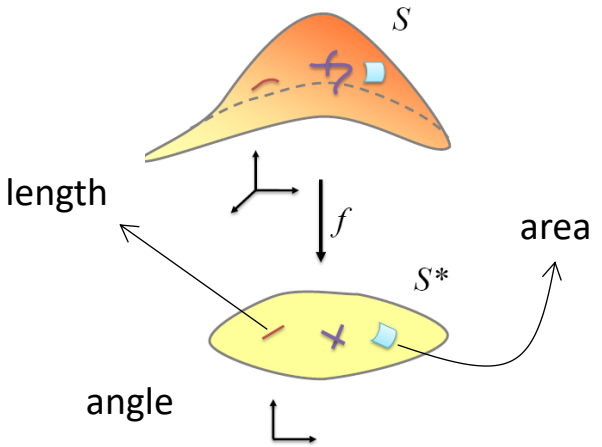


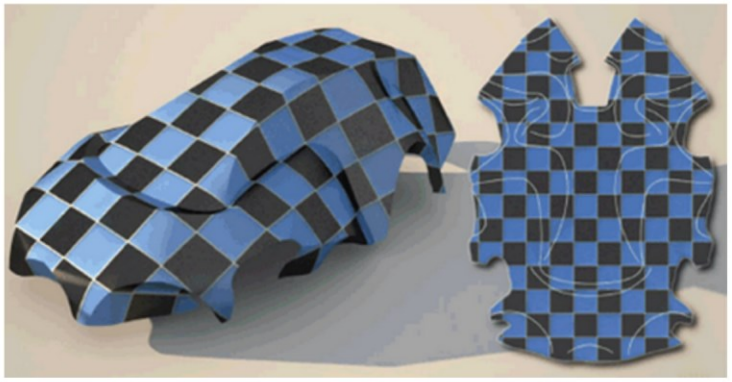
Spherical

simplicial

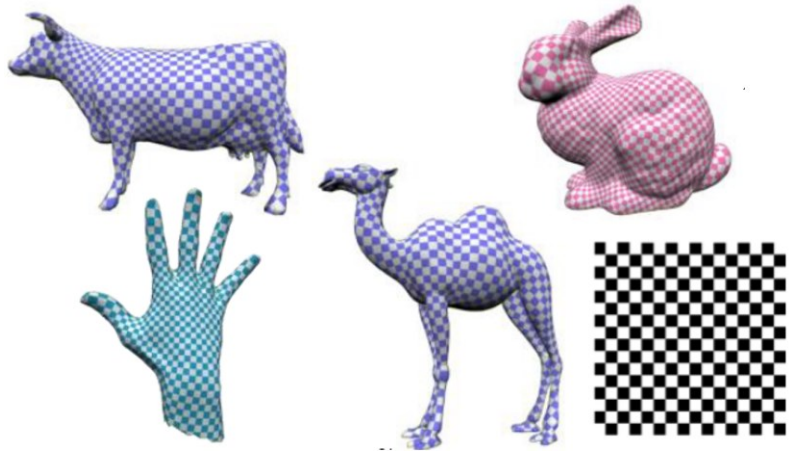
Mesh Parameterization

- Constraining on different geometric properties:
  - **Isometric** length-preserving
  - **Equiareal** area-preserving
  - **Conformal** angle-preserving
  - Isometric = equiareal + conformal

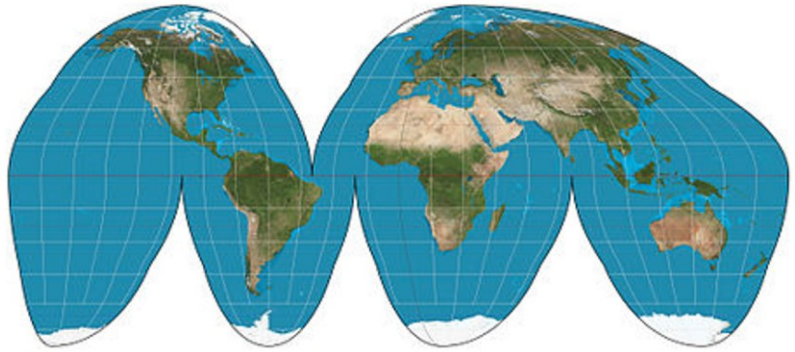




isometric



equiareal

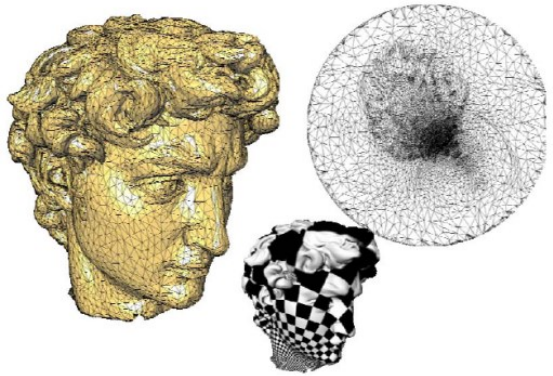


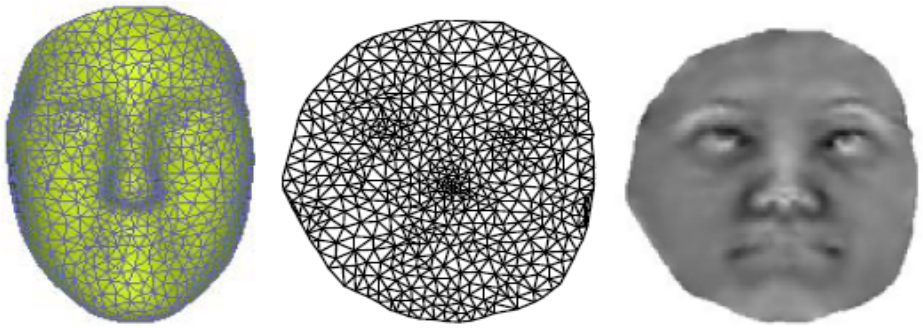
conformal

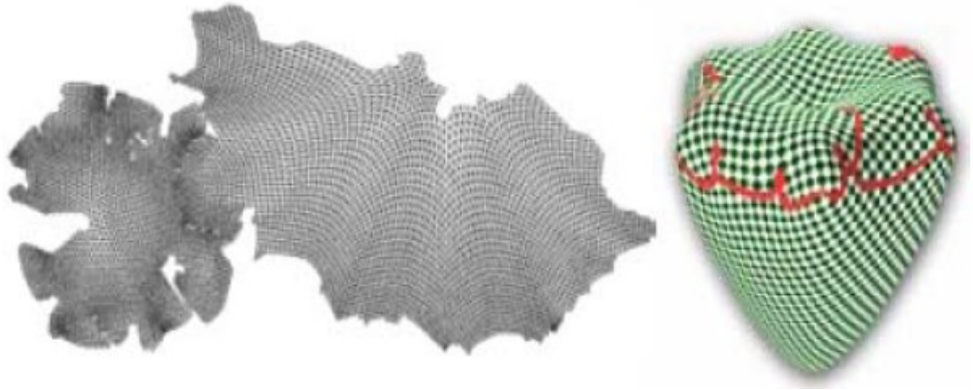
Mesh Parameterization

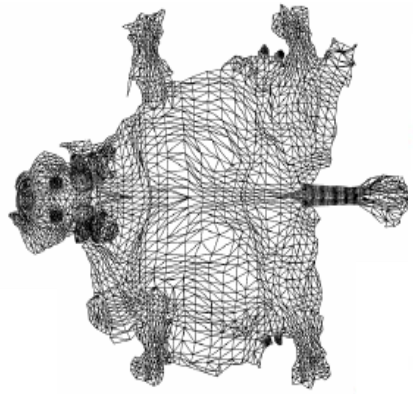
Let’s look at planer parameterization (平面参数化)

- Find the mapping between plane and mesh surface
- open mesh
  - Fixed boundary
  - Free boundary
- close mesh
  - Set boundary





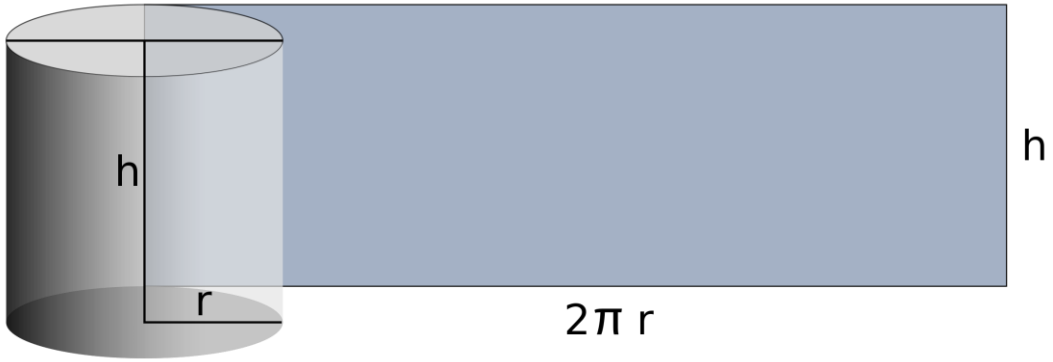




Mesh Parameterization

Planer parameterization (平面参数化)

- The most constrained case: shape preserving
  - Preserve Length, area, angle
  - Just for **Developable surface**
  - Distortion for general surface



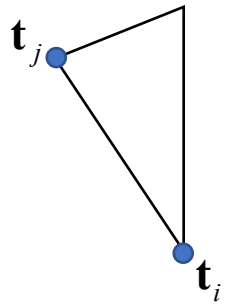
Developable surface

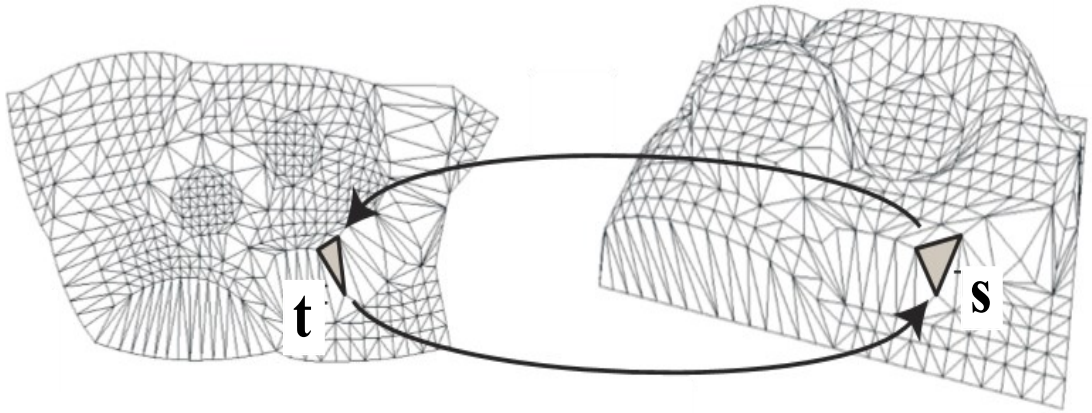
General surface

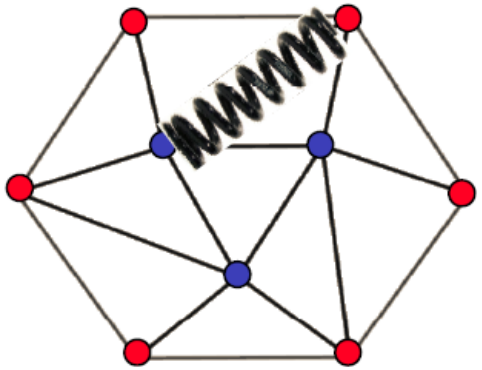
Mesh Parameterization

Let's introduce a method for planer parameterization

- Spring system







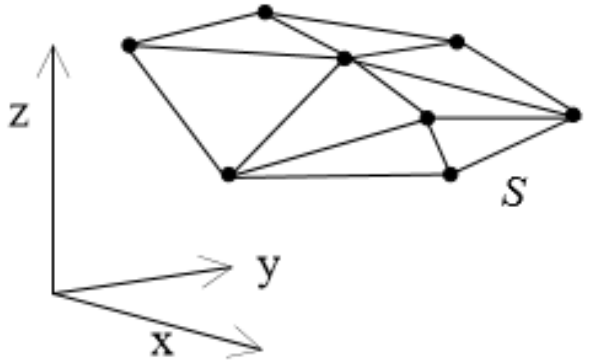
$$\mathbf{t}_i = (u_i, v_i) \quad \Leftarrow \quad \mathbf{s}_i = (x_i, y_i, z_i)$$

$$E = \frac{1}{2}\sum_{i=1}^{n}\sum_{j\in N_i}\frac{1}{2}D_{ij}\,\|\,\mathbf{t}_i - \mathbf{t}_j\,\|^2$$

Spring coefficient

Mesh Parameterization

Planer parameterization

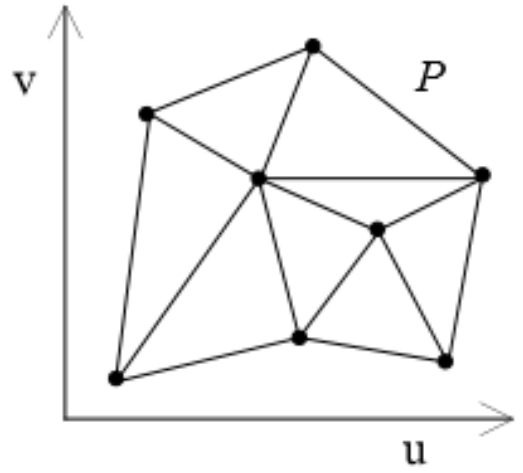


- Spring system

$$\frac{\partial E}{\partial \mathbf{t}_i} = \sum_{j \in N_i} D_{ij}(\mathbf{t}_i - \mathbf{t}_j) = 0$$

$$\Rightarrow \quad \sum_{j \in N_i} D_{ij}\mathbf{t}_i = \sum_{j \in N_i} D_{ij}\mathbf{t}_j$$

$$\Rightarrow \quad \mathbf{t}_i = \sum_{j \in N_i} \lambda_{ij}\mathbf{t}_j, \qquad \lambda_{ij} = D_{ij} / \sum_{k \in N_i} D_{ik}$$



- coefficient λ_{ij}

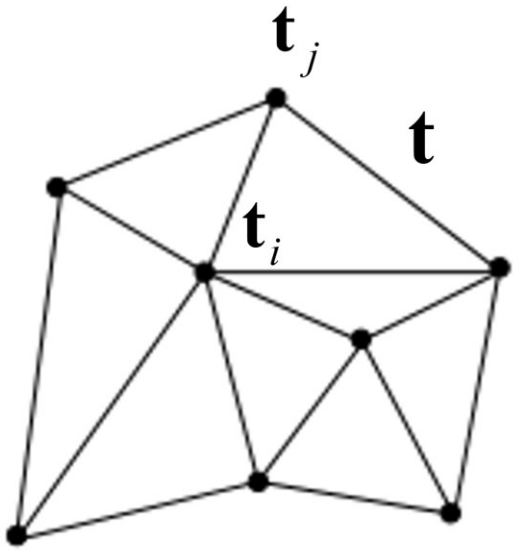
Mesh Parameterization

Planer parameterization

- Spring system
- coefficient λ_{ij}
- Linear equations

$$\mathbf{t}_i - \sum_{j \in N_i} \lambda_{ij} \mathbf{t}_j = 0$$

$$\Rightarrow \quad \mathbf{A}\mathbf{t} = 0$$



- Add constrain to avoid trivial solution

Mesh Parameterization

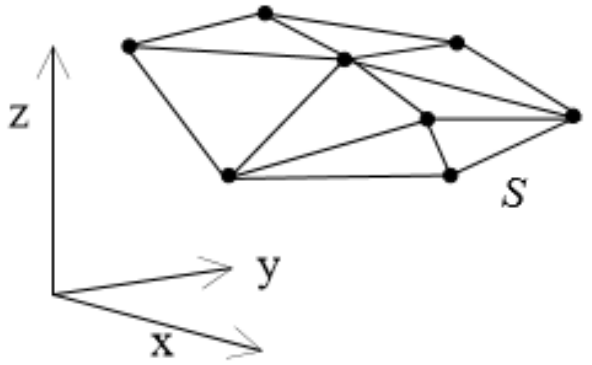
Planer parameterization

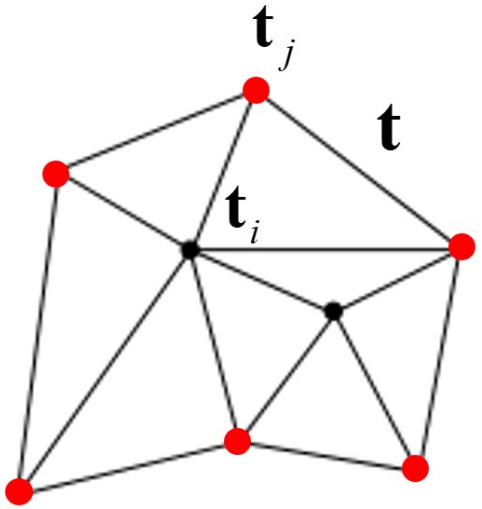
- Spring system
- coefficient λ_{ij}
- Linear equations
  - Fixed boundary Barycentric coordinates

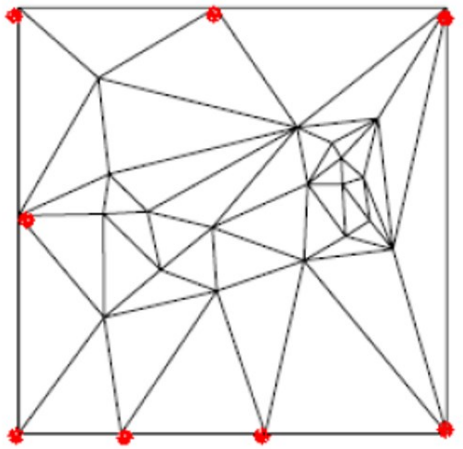
$$\mathbf{t}_i - \sum_{j \in N_i} \lambda_{ij} \mathbf{t}_j = 0$$

Red vertices: Known t

$$\Rightarrow \quad \bar{\mathbf{A}}\mathbf{t} = \mathbf{b}$$



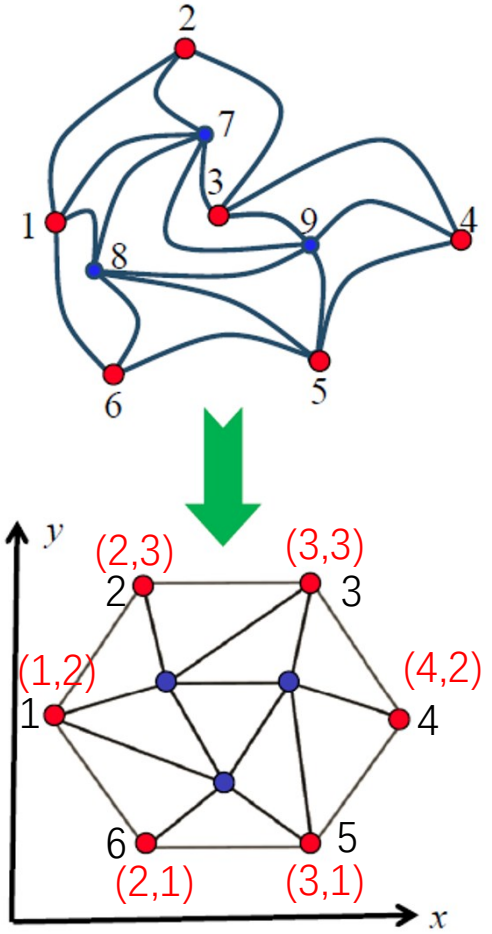




Mesh Parameterization

- Planer parameterization
- Spring system, coefficient λ_{ij}
- Linear equations
  - Fixed boundary Barycentric coordinates

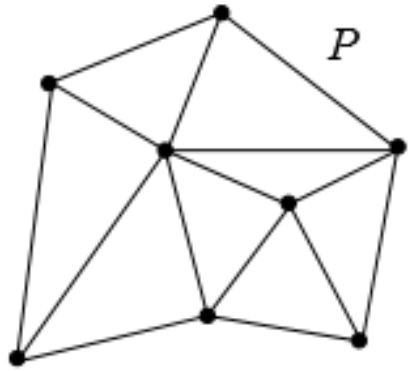
$$\overline{\mathbf{A}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & -5 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & -5 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5 \end{pmatrix} \quad \mathbf{b}_x = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 3 \\ 2 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ ? \\ ? \\ ? \end{matrix} \quad \mathbf{b}_y = \begin{pmatrix} 2 \\ 3 \\ 3 \\ 2 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ ? \\ ? \\ ? \end{matrix}$$



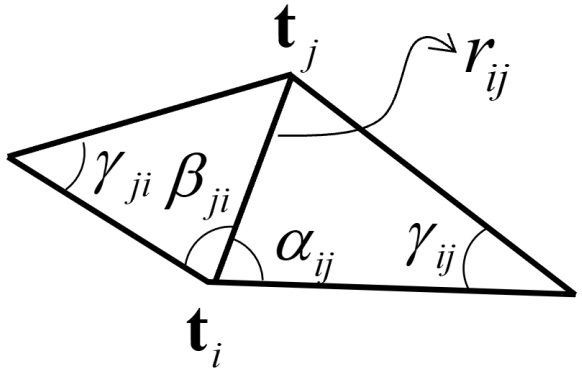
Mesh Parameterization

Planer parameterization

- Spring system $\lambda_{ij} = D_{ij} / \sum_{k \in N_i} D_{ik}$
- coefficient λ_{ij}



- Average $\lambda_{ij} = 1/n_i$
- Harmonic coordinate $\lambda_{ij} = \left(\cot\gamma_{ij} + \cot\gamma_{ji}\right)/2$



- Mean coordinate $\lambda_{ij} = \left(\tan\frac{\alpha_{ij}}{2} + \tan\frac{\beta_{ji}}{2}\right)/r_{ij}$

Quad mesh, geometry image

- How to convert a general polygonal mesh to quad mesh is still a research topic
- And further to a geometry image