Computer Graphics

Mesh Processing

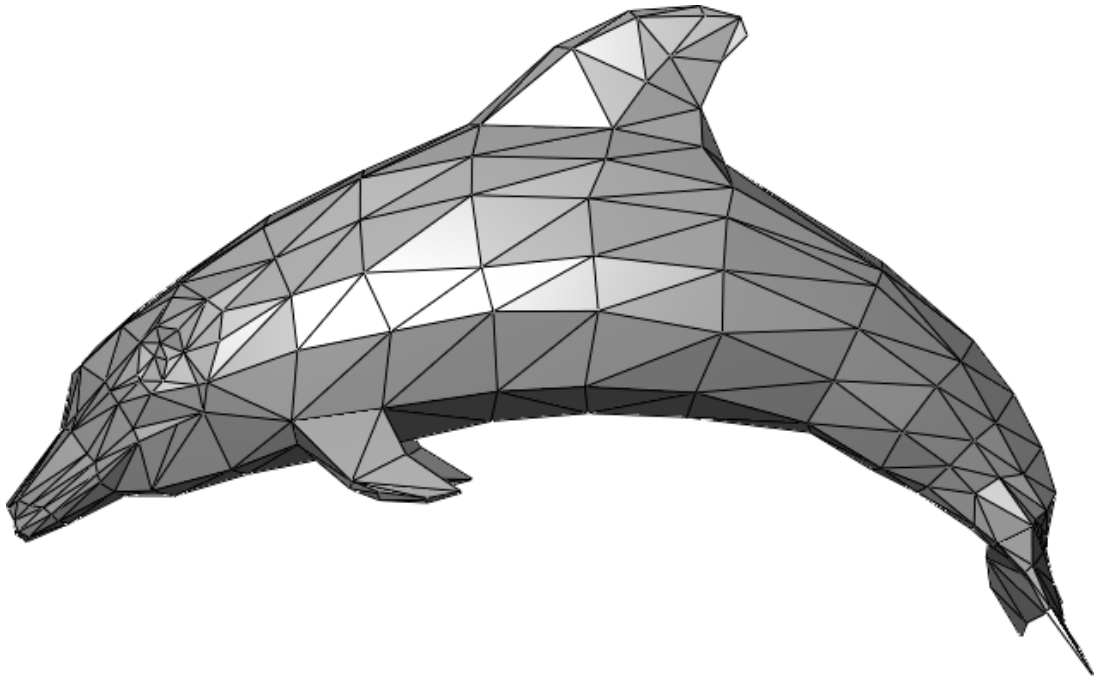
Teacher: A.prof. Chengying Gao(高成英)

E-mail: mcsgcy@mail.sysu.edu.cn

School of Data and Computer Science

What is Polygon Mesh?

- A polygon mesh is a collection of vertices, edges, and faces that defines the shape of a polyhedral object in 3D computer graphics and solid modeling.

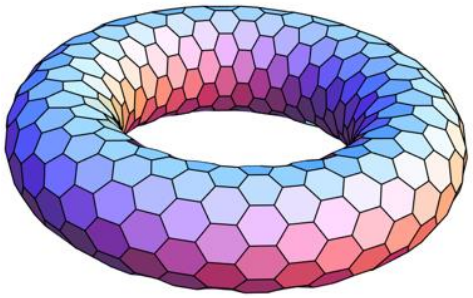


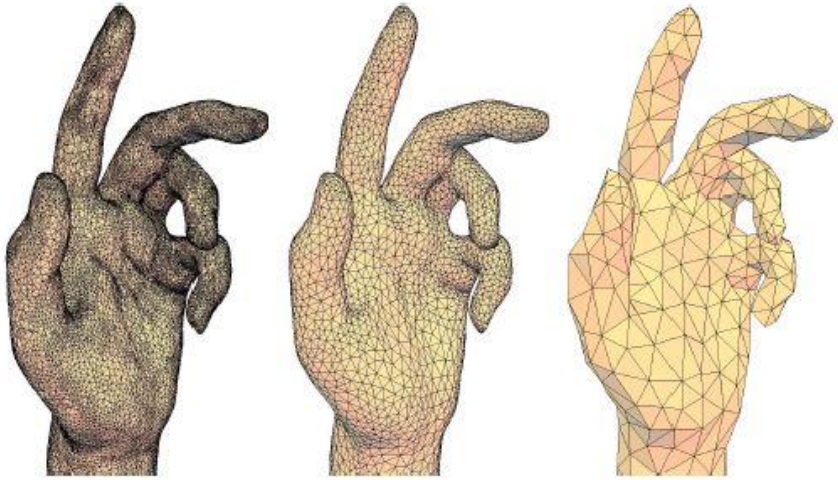
Example – Polyhedral widgeon

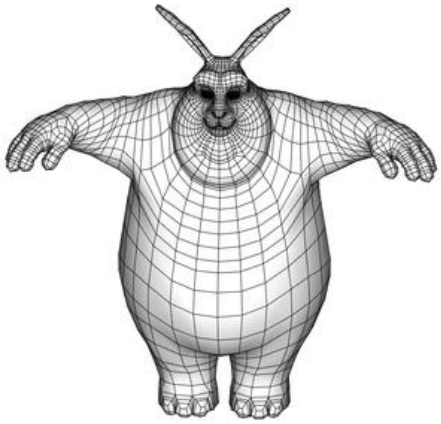
6656 faces (面)， 3474 vertices (顶点)

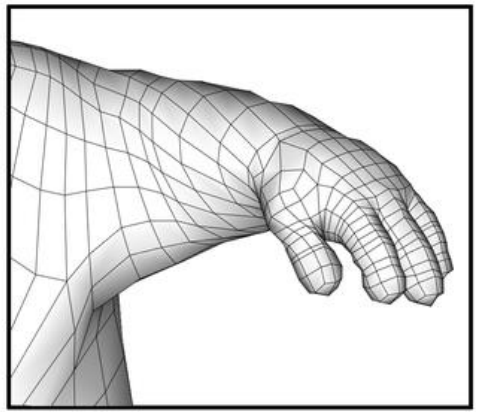
Categories of Polyhedron (多面体)

- Polyhedron are essentially linear approximation
  - Triangular meshes (三角网格)
  - Quadrilateral meshes (四边形网格)
  - Polygonal meshes (多边形网格)









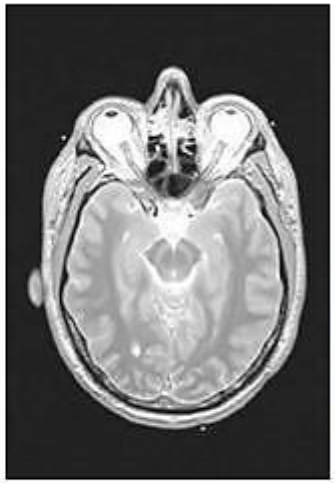
Outline

- Meshes (polyhedra)
  - Data Acquisition
  - Data structure for proximity retrieving
  - Mesh processing

Volumetric Scanning

- Build voxel structure by scanning slices

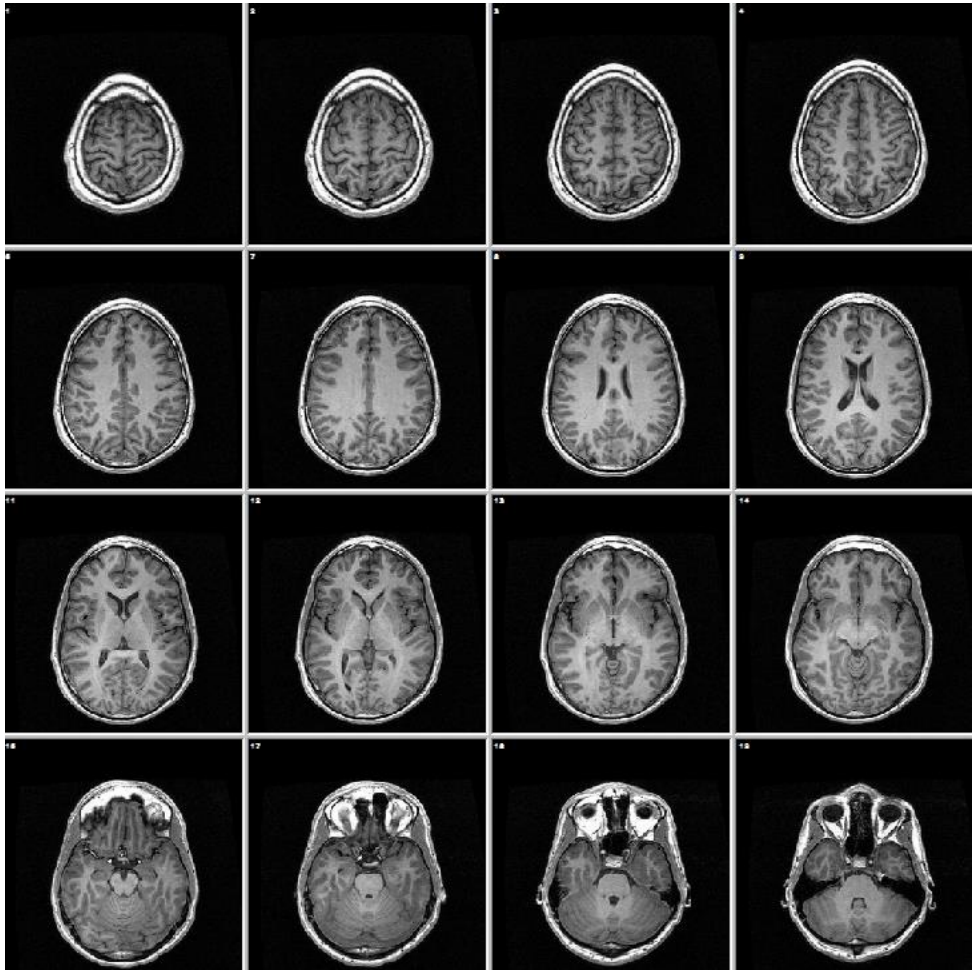
CT

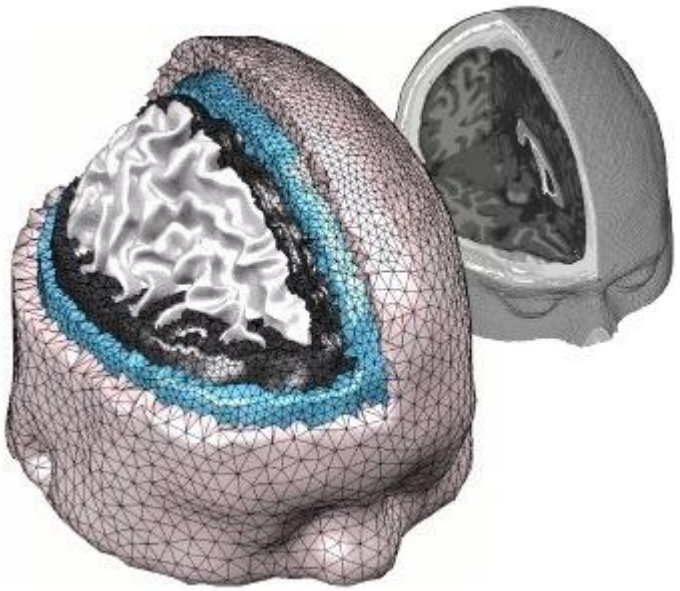


MRI

Volumetric Scanning

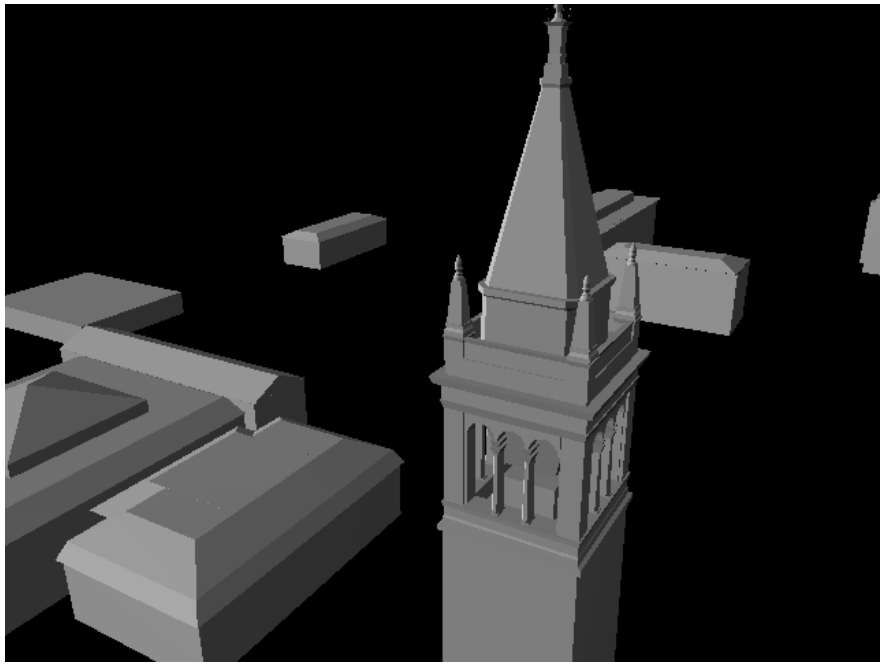
- Build voxel structure by scanning slices

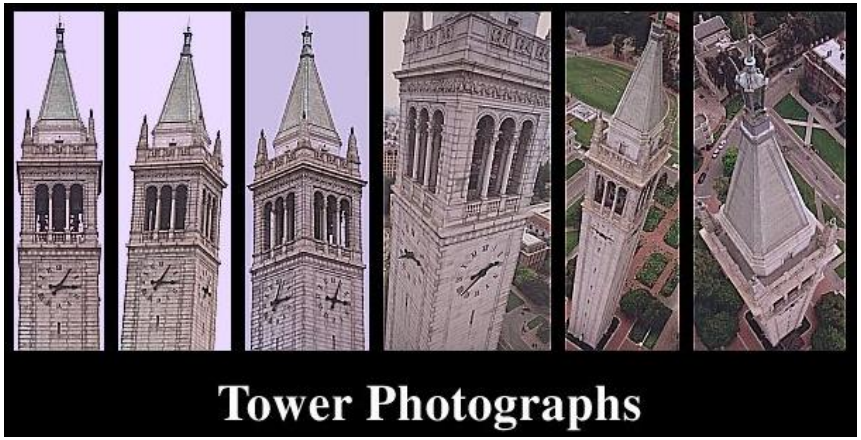




Photogrammetry

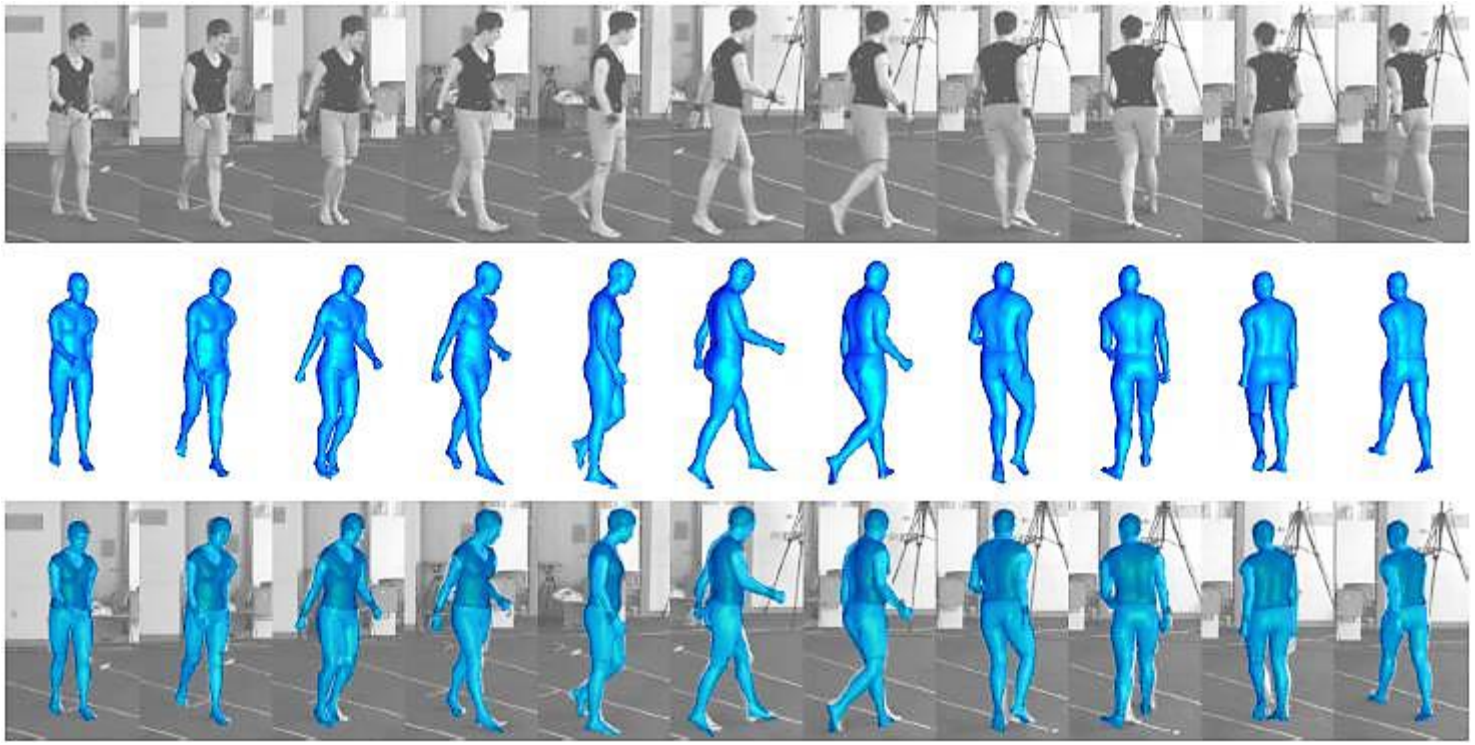
- Reconstruction from photographs





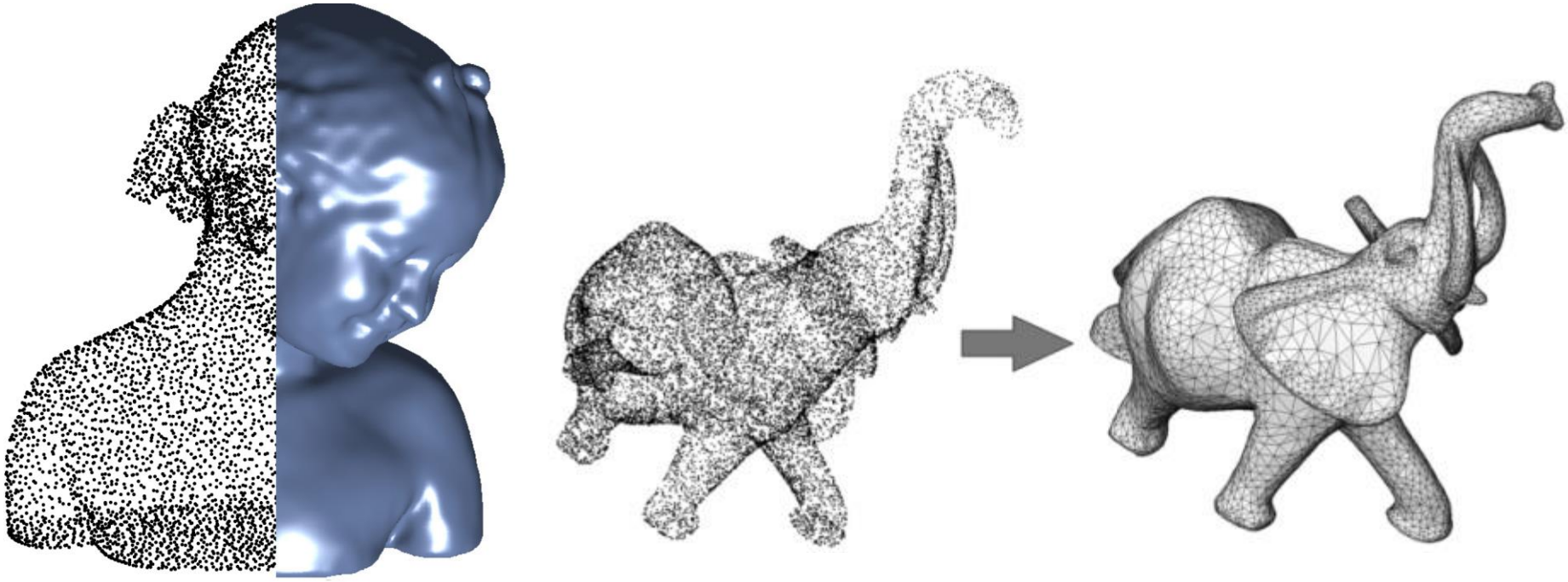
Photogrammetry

- Reconstruction from a series of photos (video)



Range Scanning

- Reconstruction from point cloud

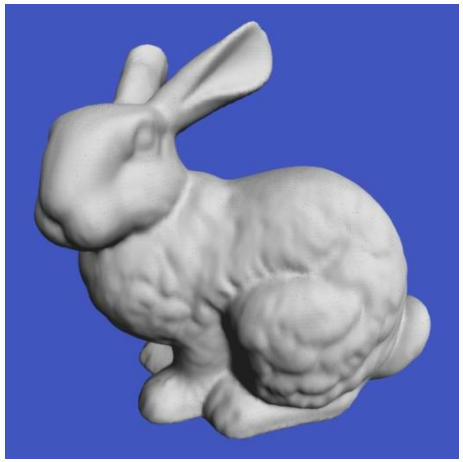


Getting Meshes from Real Objects

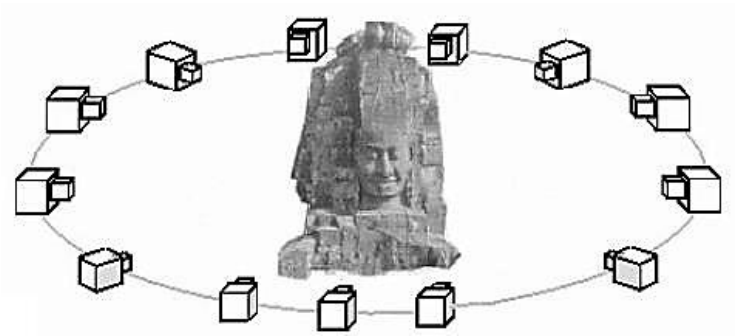
- Many models used in Graphics are obtained from real objects

Stanford dragon

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



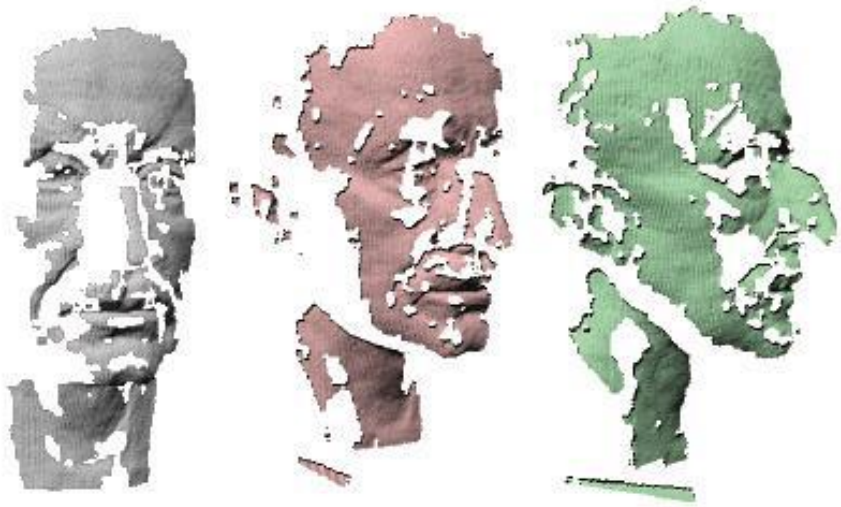
Getting Meshes from Real Objects



Range Scanning

- Accurate calibration is crucial
- Multiple scans required for complex objects
  - scan path planning
  - scan registration
- Scans are incomplete and noisy
  - model repair, hole filling
  - smoothing for noise removal

Range Scanning: Reconstruction



Set of raw scans

Reconstructed model

General Used Mesh Files

- General used mesh files
  - Wavefront OBJ (*.obj)
  - 3D Max (*.max, *.3ds)
  - VRML(*.vrl)
  - Inventor (*.iv)
  - PLY (*.ply, *.ply2)
  - User-defined(*.m, *.liu)
- 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
```

Wavefront .obj file

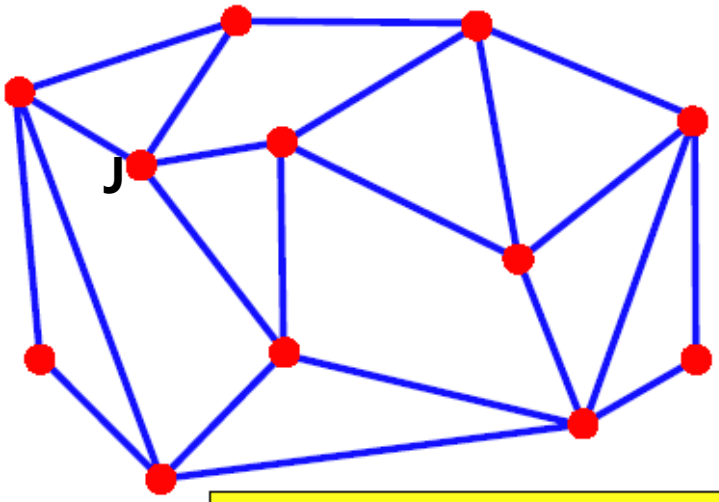
```
# List of Vertices, with (x,y,z[,w]) coordinates, w is optional and defaults to 1.0.
  v 0.123 0.234 0.345 1.0
  v ...
  ...
  # Texture coordinates, in (u ,v [,w]) coordinates, these will vary between 0 and 1, w is optional and
default to 0.
  vt 0.500 1 [0]
  vt ...
  ...
  # Normals in (x,y,z) 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 ...
  ...
```

Outline

- Meshes (polyhedra)
  - Data Acquisition
  - Data structure for proximity retrieving
    - Meshes: Definitions & Terminologies
    - Mesh Data Structure
  - Mess processing

Meshes: Definitions & Terminologies

Standard Graph Definition：



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)}

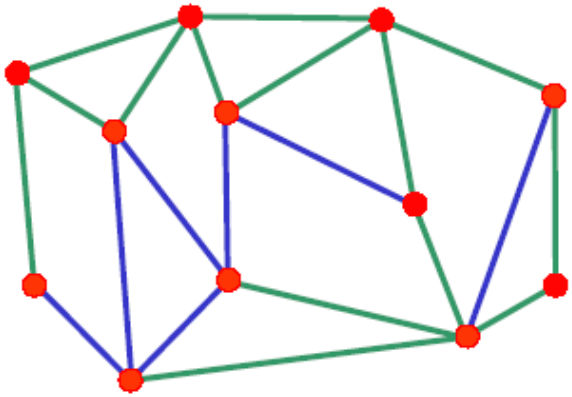
Vertex degree (valence) = number of edges incident on vertex
deg(J) = 4, deg(H) = 2
k-regular graph = graph whose vertices all have degree *k*

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)}

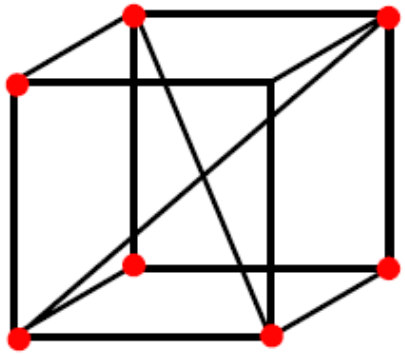
Meshes: Definitions & Terminologies

- Graph Embedding

Graph is ***embedded*** in R^d if each vertex is assigned a position in R^d



Embedding in R^2



Embedding in R^3

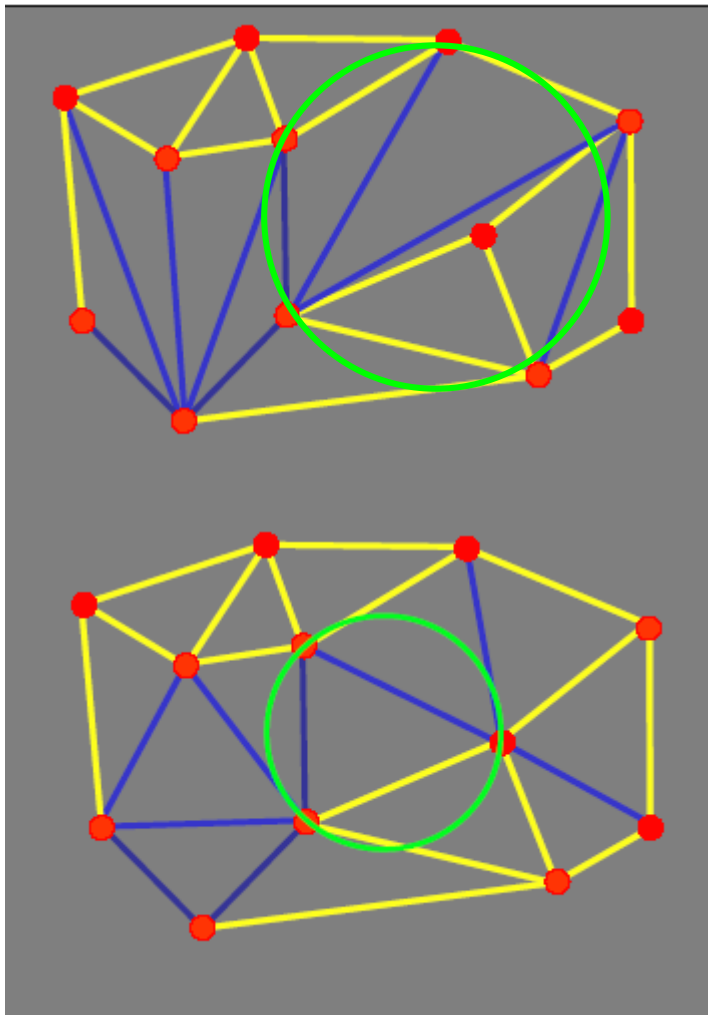
Meshes: Definitions & Terminologies

- Triangulation

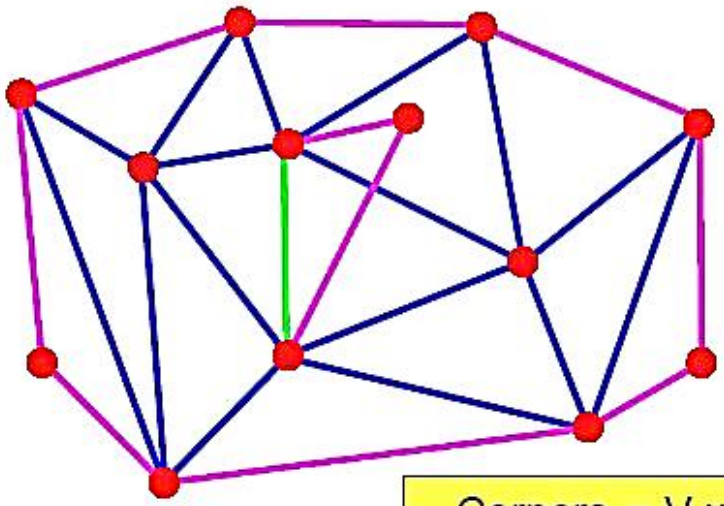
Triangulation: straight line plane graph all of whose faces are triangles

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



Meshes

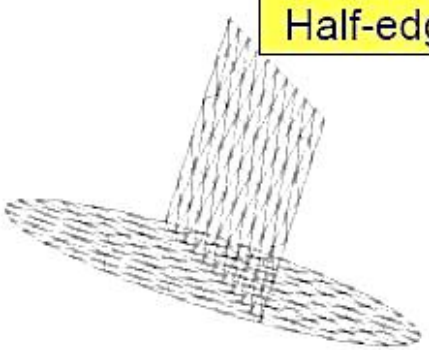


Mesh: straight-line graph embedded in R^3

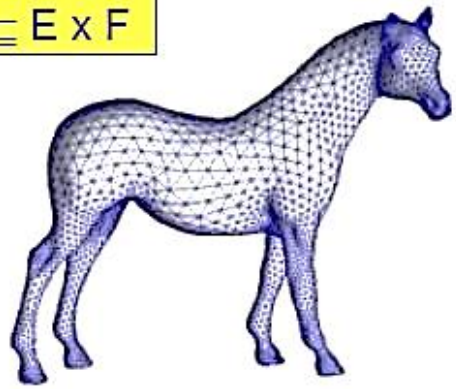
Boundary edge: adjacent to exactly *one* face
Regular edge: adjacent to exactly *two* faces
Singular edge: adjacent to more than two faces

Closed mesh: mesh with no boundary edges
Manifold mesh: mesh with no singular edges

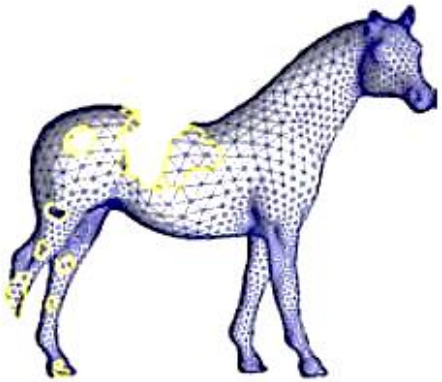
Corners $\subseteq$ V x F
Half-edges $\subseteq$ E x F



Non-Manifold



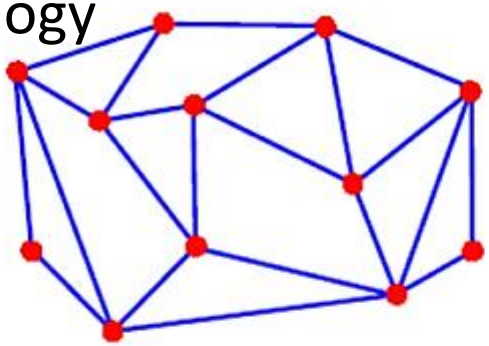
Closed Manifold



Open Manifold

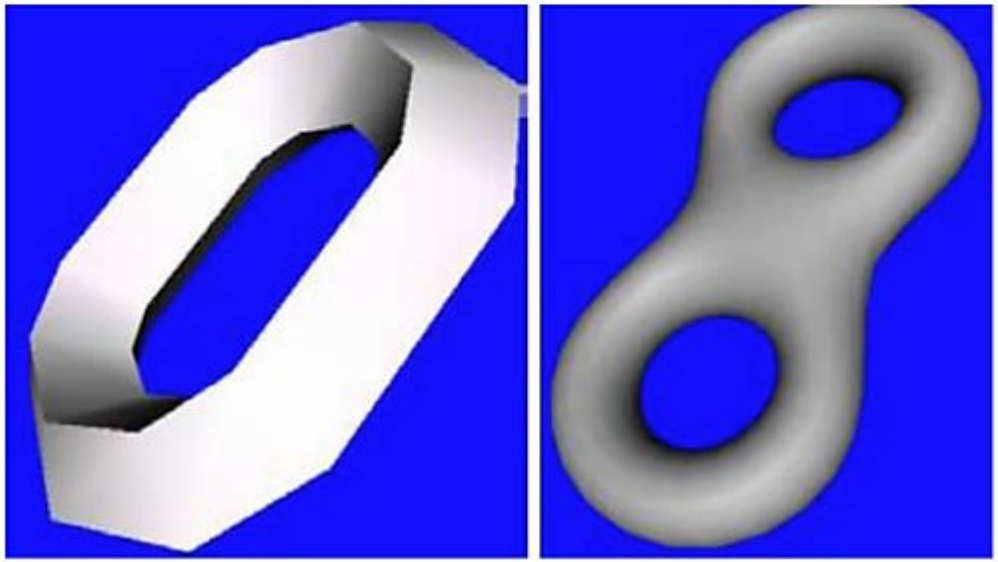
Meshes: Definitions & Terminologies

- Topology



亏格 *Genus* of graph: *half* of *maximal* number of closed paths that do *not* disconnect the graph (number of "holes")

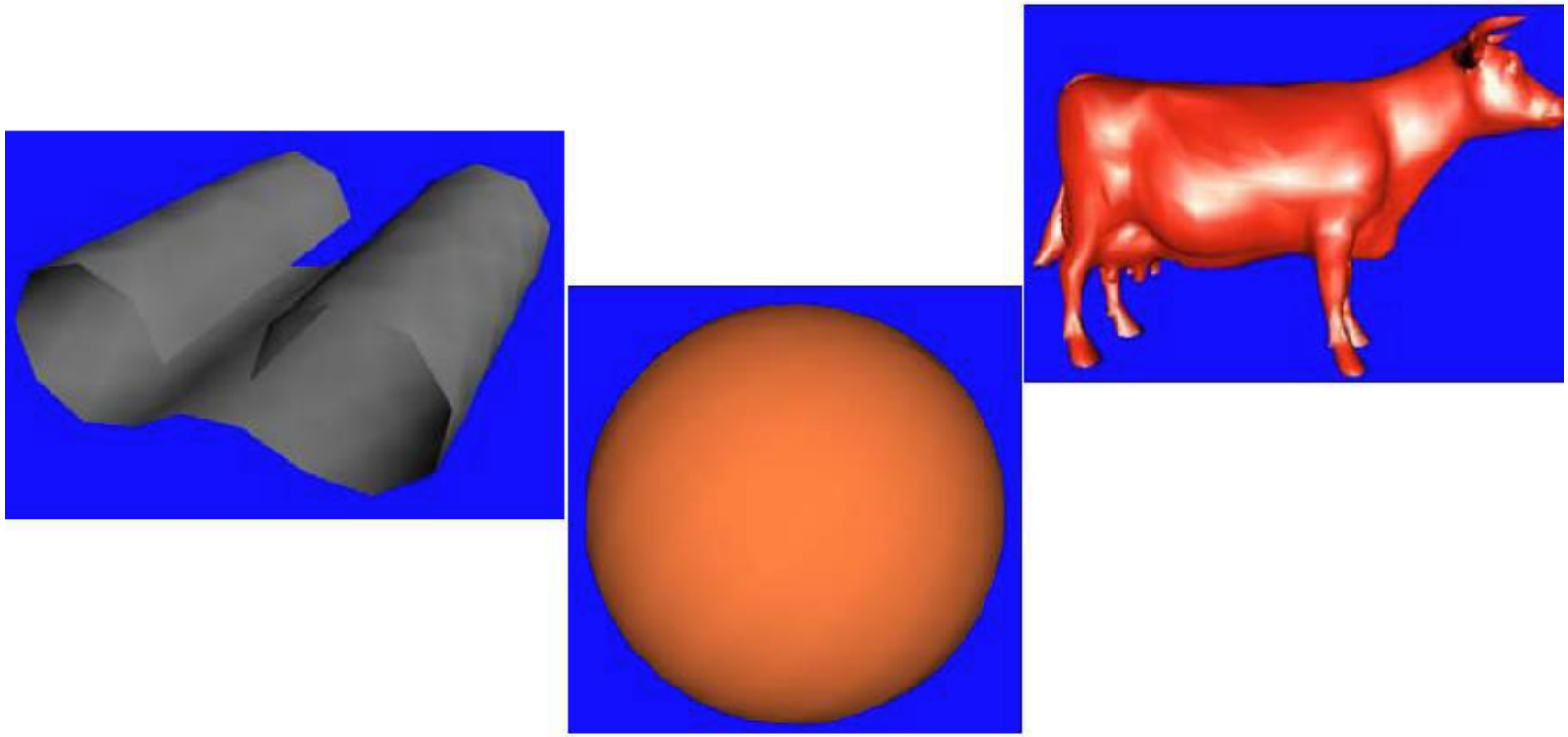
Genus(sphere)= 0
Genus(torus) = 1



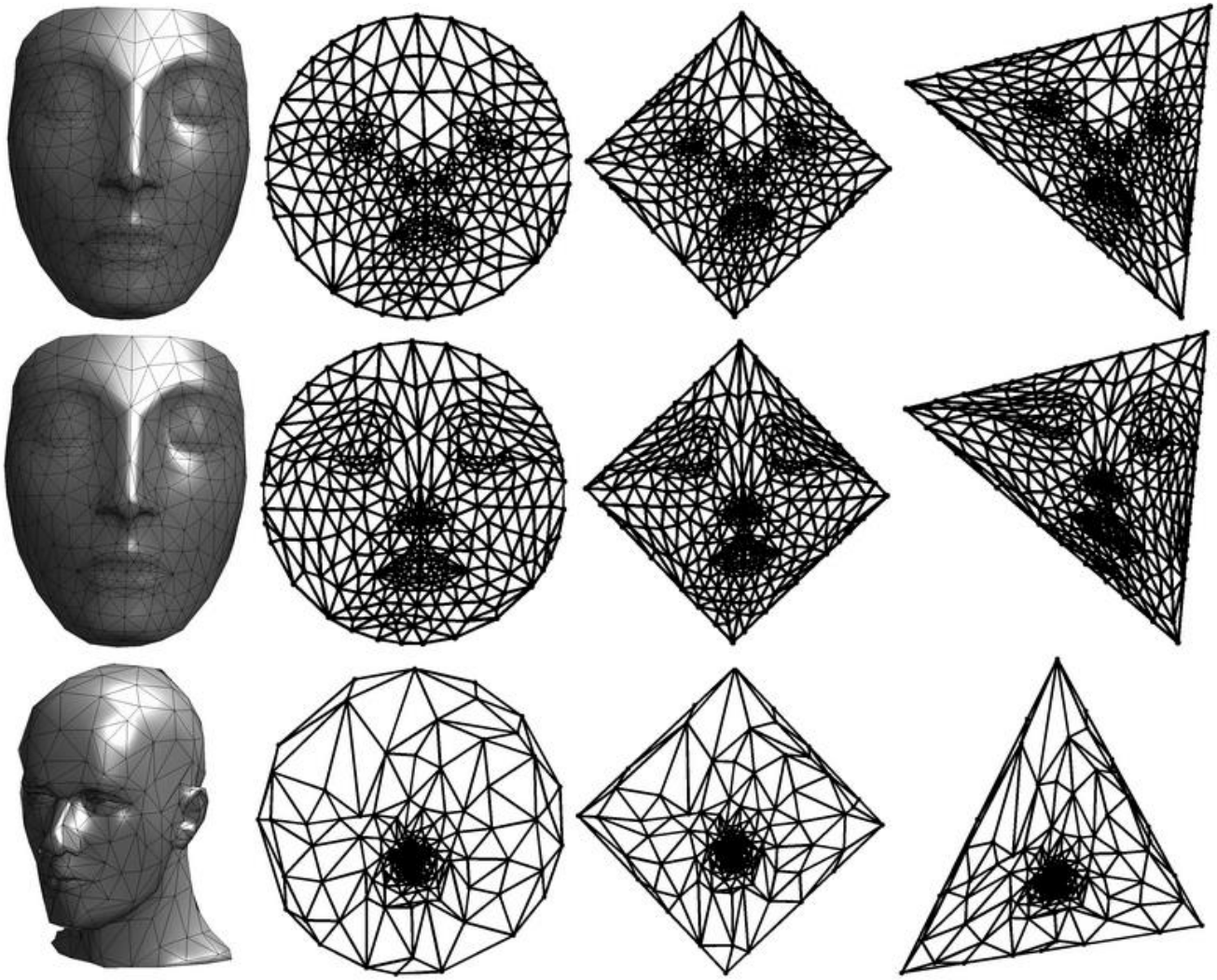
Meshes: Definitions & Terminologies

- Developability (可展性)

Mesh is ***developable*** if it may be embedded in R^2 without distortion



Meshes: Definitions & Terminologies

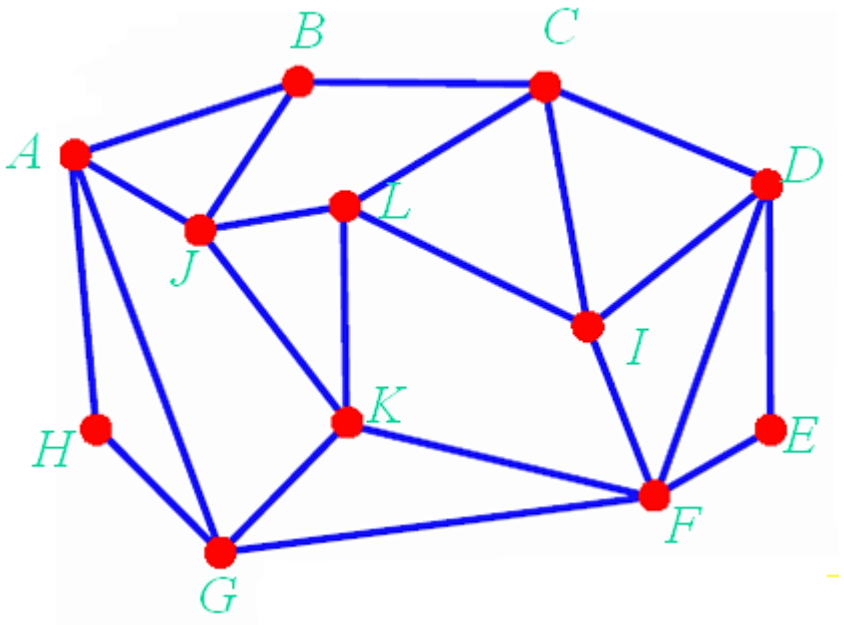


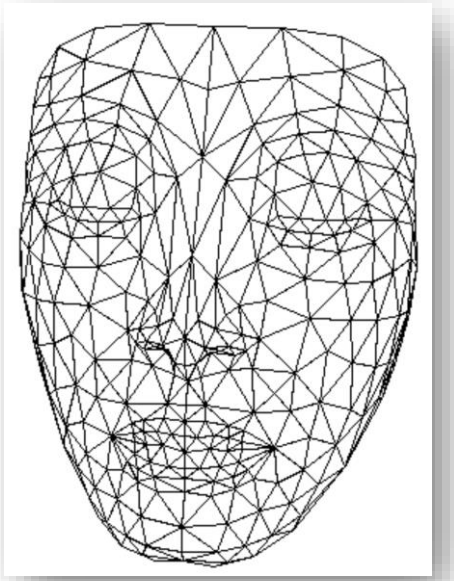
Mesh Data Structure

- How to store geometry and connectivity?
- Geometry queries
  - What are the vertices of face #k?
  - Are vertices #i and #j adjacent?
  - Which faces are adjacent face #k?
- Geometry operations
  - Remove/add a vertex/face
  - Mesh simplification
  - Vertex split, edge collapse

Define a mesh

- Geometry
  - Vertex coordinates
- Connectivity
  - How do vertices connected?



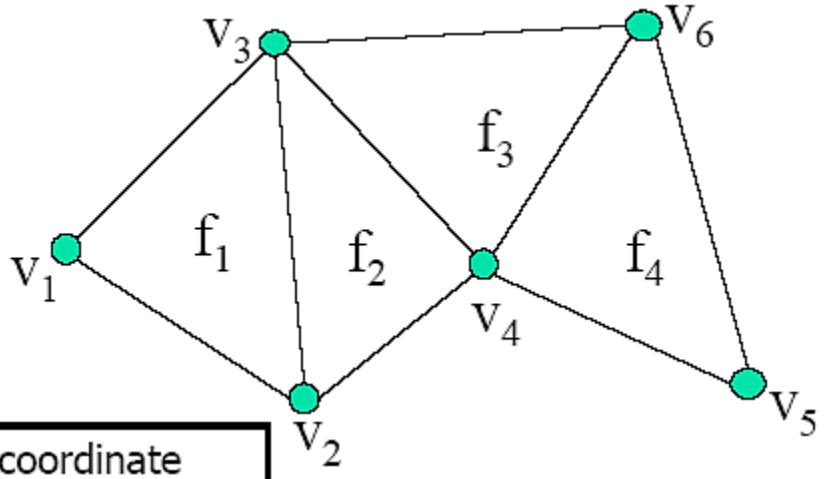


- List of Edge
- Vertex-Edge
- Vertex-Face
- Combined

List of Faces

- 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

List of Faces – Example



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)

List of Faces – Analysis

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

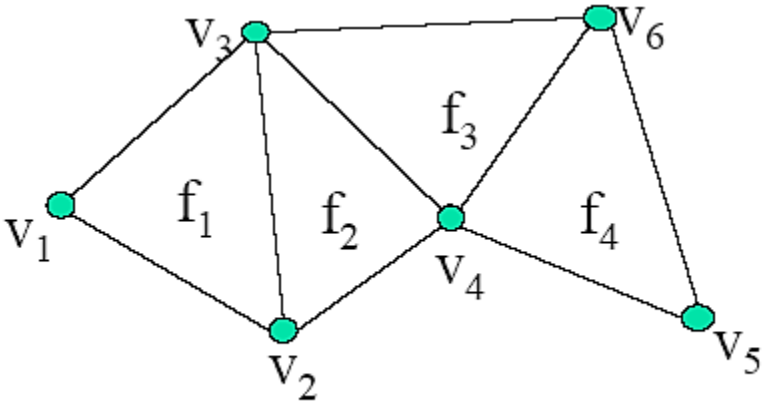
Adjacency Matrix – Definition

- 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)

Adjacency Matrix – Example

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	

Adjacency Matrix – Queries

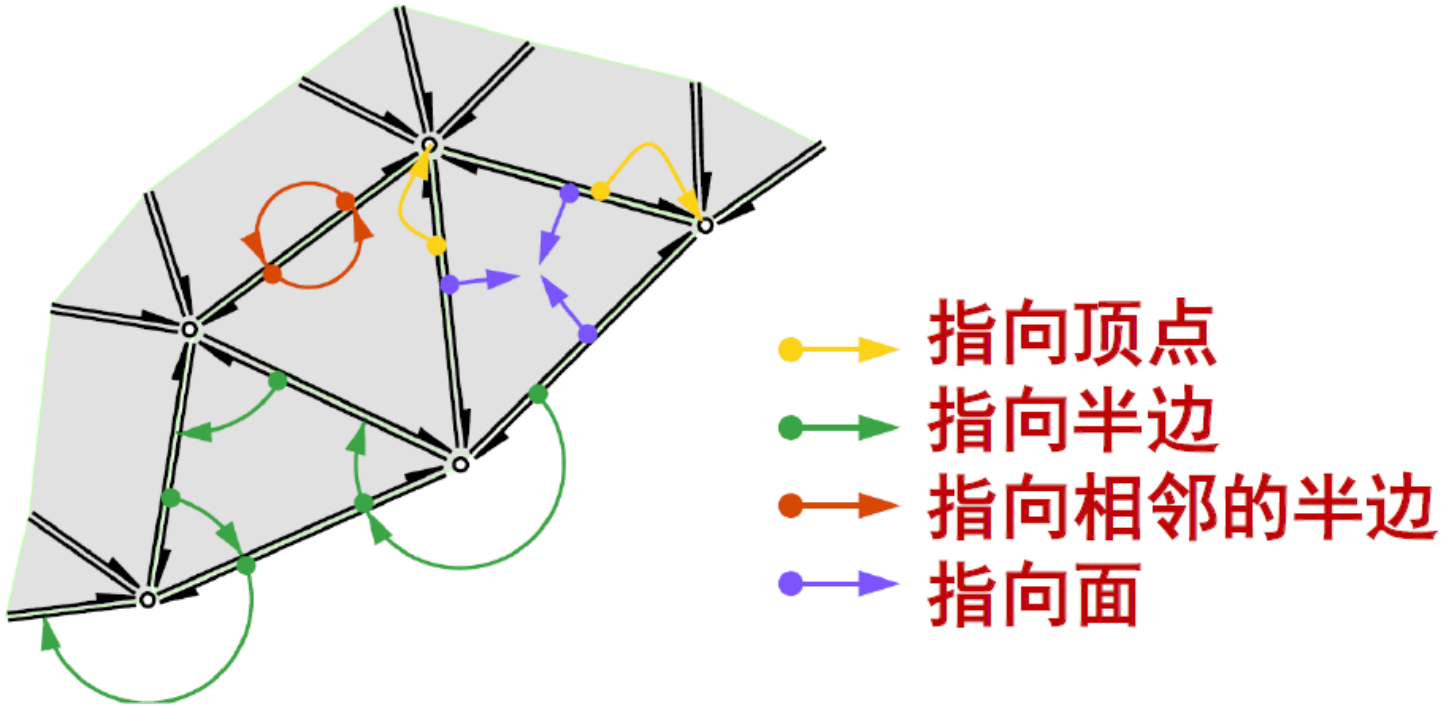
- What are the vertices of face #3?
  - O(1) – checking third triplet of faces
- Are vertices i and j adjacent?
  - O(1) - checking adjacency matrix at location (i,j).
- Which faces are adjacent to vertex j?
  - Full pass on all faces is necessary

Adjacency Matrix – Analysis

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

Half-Edge Structure

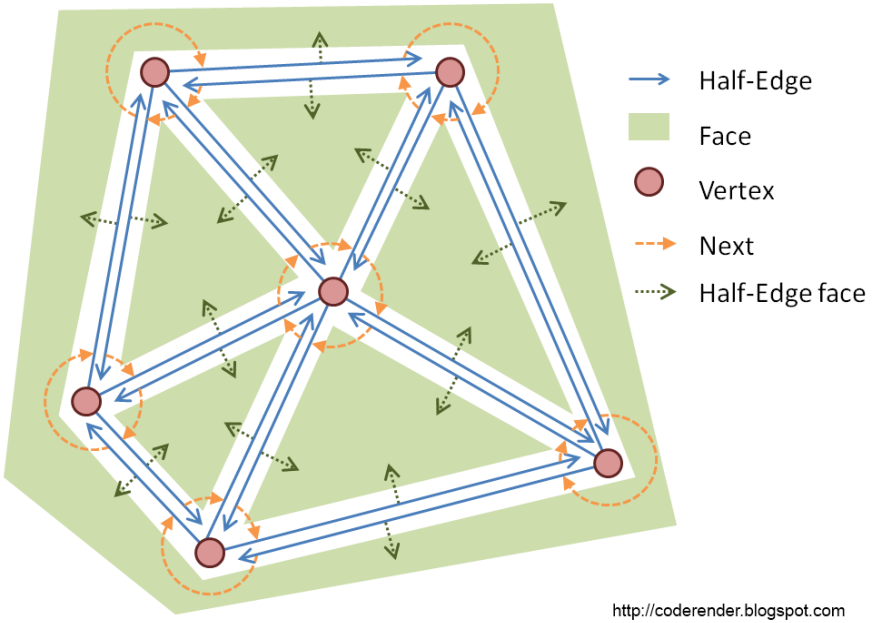
- Orientable 2D manifolds and its sub set: special polygonal meshes (适用于有向的二维流形)



Half-Edge Structure

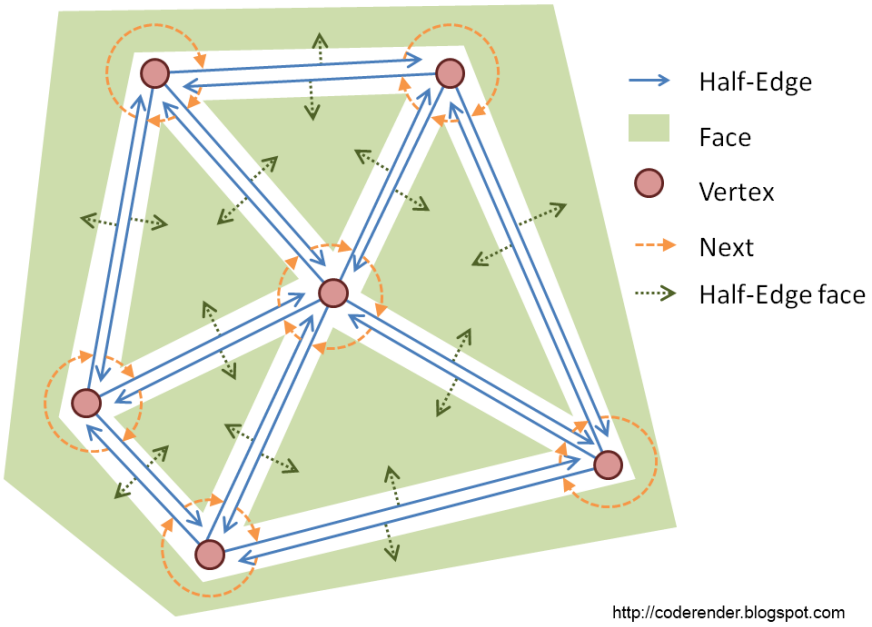
- Half-edge (each edge corresponds to two half-edges)
  - Pointer to the first vertices
  - To adjacent face
  - To next half-edge (逆时针方向)
  - To the other half-edge of the same edge
  - To previous half-edge (opt.)

```
struct HE_edge {
    HE_vert* vert; // vertex at the start of the half-edge
    HE_face* face; // face the half-edge borders
    HE_edge* pair; // oppositely oriented adjacent half-edge
    HE_edge* next; // next half-edge around the face
    HE_edge* prev; // prev half-edge around the face
};
```



Half-Edge Structure

- Face：we only need a pointer to one of its half-edge



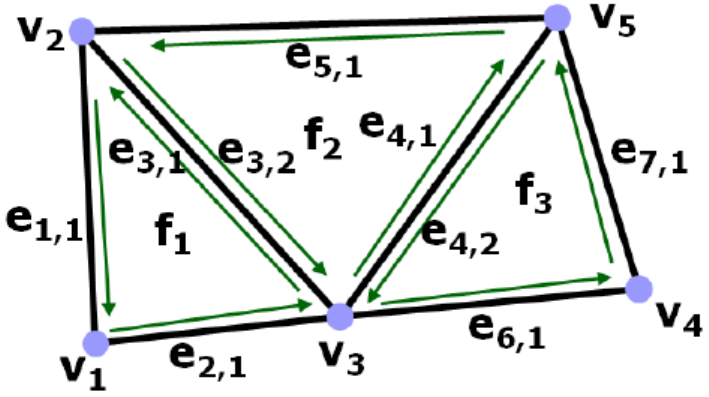

```
struct HE_face {
    HE_edge* edge; // one of the half-edges bordering the face
};
```

Half-Edge Structure

- Vertices
  - 3D coordinates
  - Pointer to the half-edge starting from it

```
struct HE_vert {
    float x;
    float y;
    float z;
    HE_edge* edge; // one of the half-edges
                   //emantating from the vertex
};
```

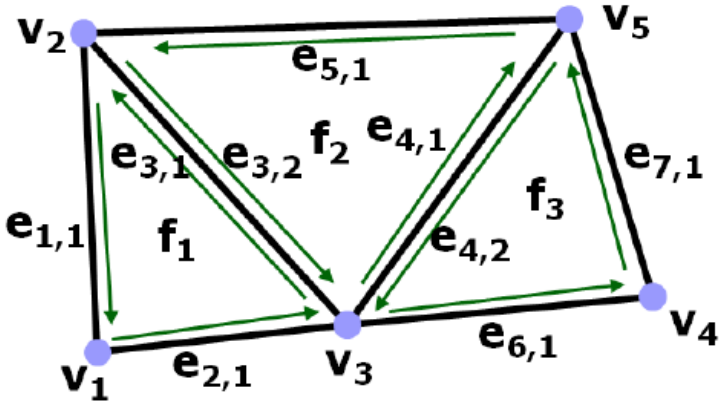
Example: half-edge structure



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)	$e_{5,1}$

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

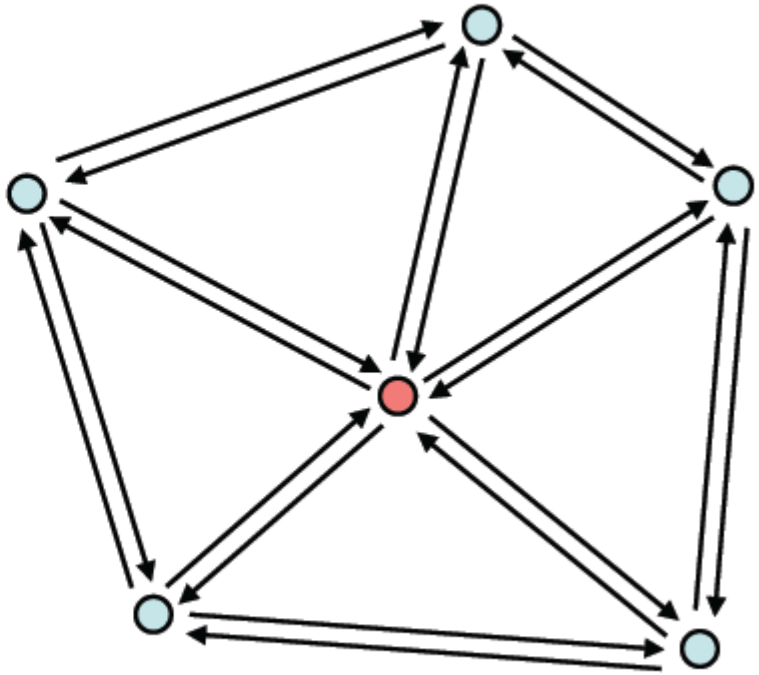
Example (continued)



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	$e_{6,1}$	$e_{7,1}$

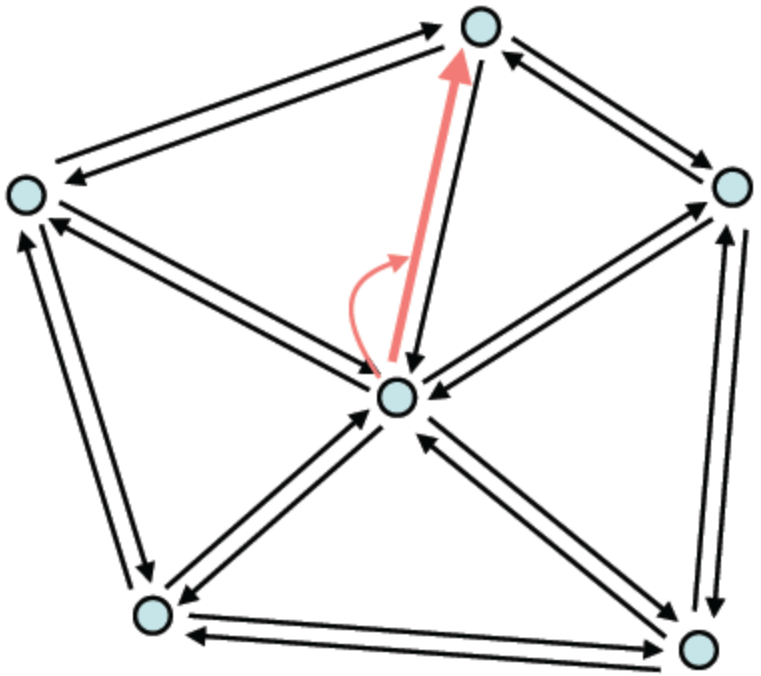
One-Ring Traversal

1. Start at vertex



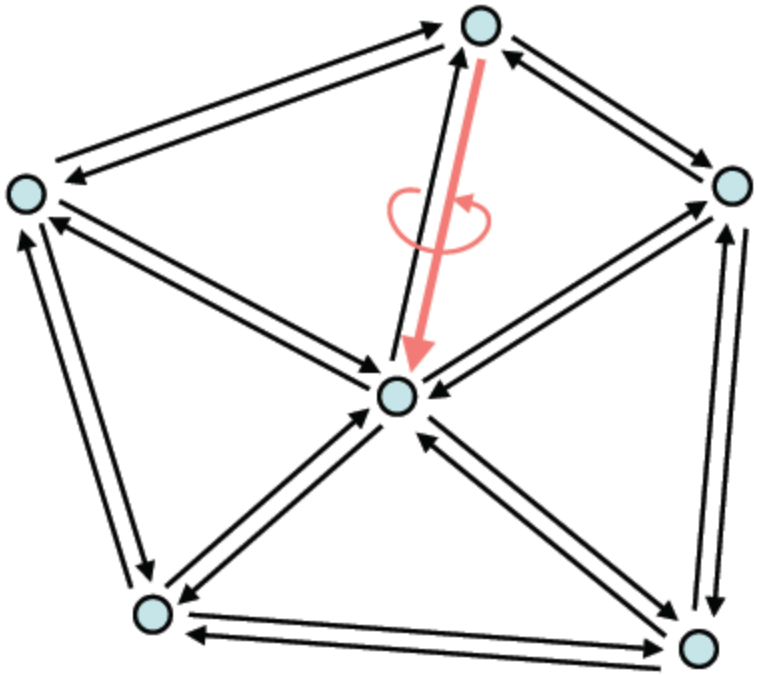
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge



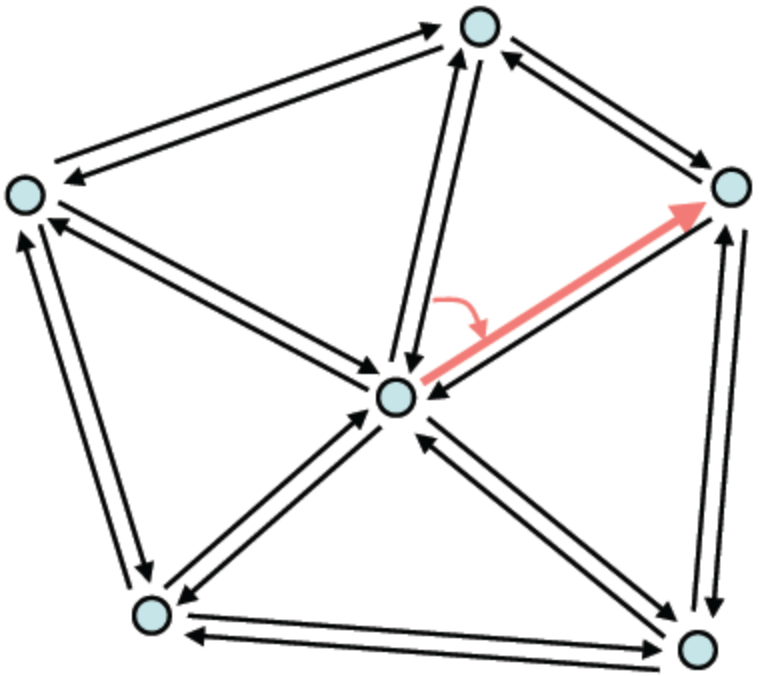
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge



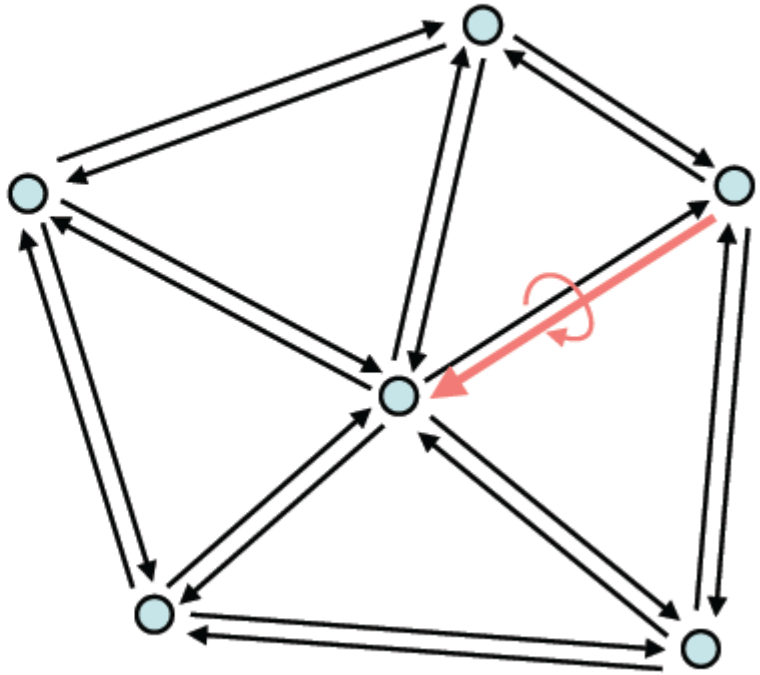
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge



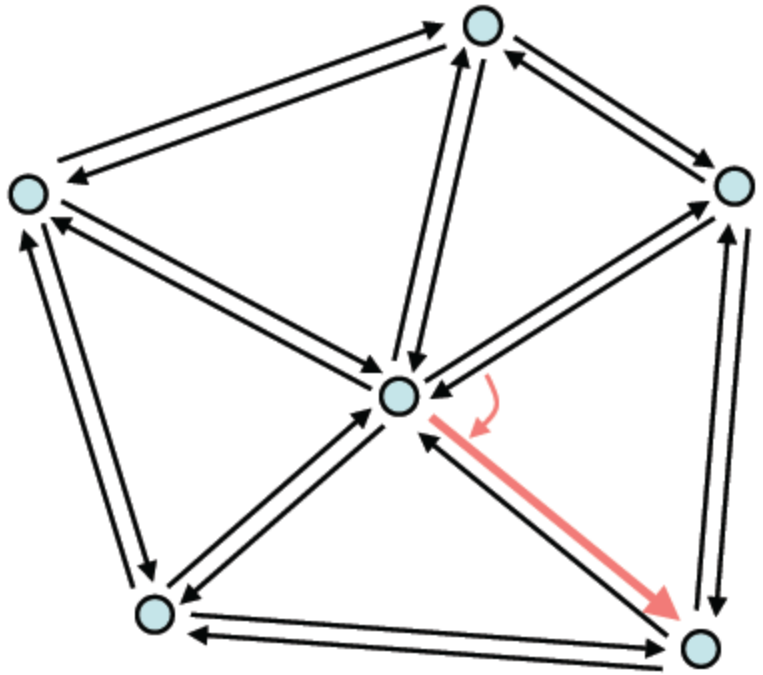
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
5. Opposite



One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
5. Opposite
6. Next
7. ...



Traversal operations

Vertices adjacent to a vertex v, mesh without boundary

```
he = v->halfedge;
do {
  he = he->sym->next;
  ... // perform operations with
      // he->vertex
} while (he != v->halfedge)
```

No “if” statements.

Basic operations

- Mark mesh boundary (标记边界点)
- Create edge adjacency (创建邻接边)
- Add vertex (增加顶点)
- Add edge (增加边)
- Add polygonal face (增加面)
- Delete polygonal face (删除面)
- Delete edge (删除边)
- Delete vertex (删除顶点)

Discussion

- Advantage and disadvantage(优缺点)：
  - Adv.：Query time O(1)，operation time O(1)
  - Dis.： redundancy & only applicable to 2D manifolds
- For more information refer to
  - CGAL：
    - the Computational Geometry Algorithms Library， http://www.cgal.org/
    - Free for non-commercial use
  - OpenMesh： http://www.openmesh.org/
    - Mesh processing
    - Free, LGPL licence
  - Meshlab: http://meshlab.sourceforge.net/

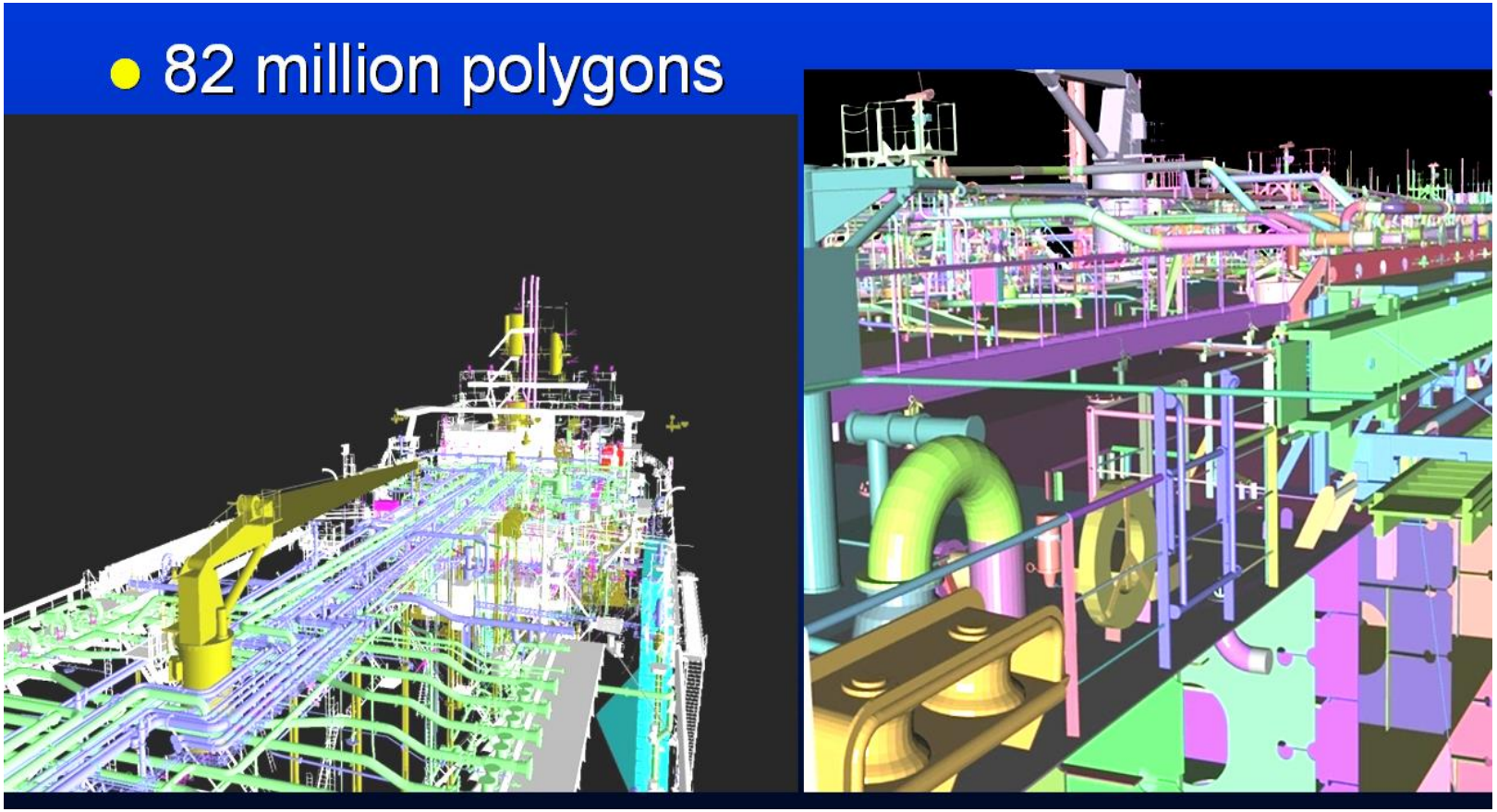
Advantage and disadvantage in polygon representation

- Advantage
  - Simplicity - ease of description
  - Based data for rendering software/hardware
  - Input to most simulation/analysis tools
  - Output of most acquisition tools
    - laser scanner, CT, MRI, etc...

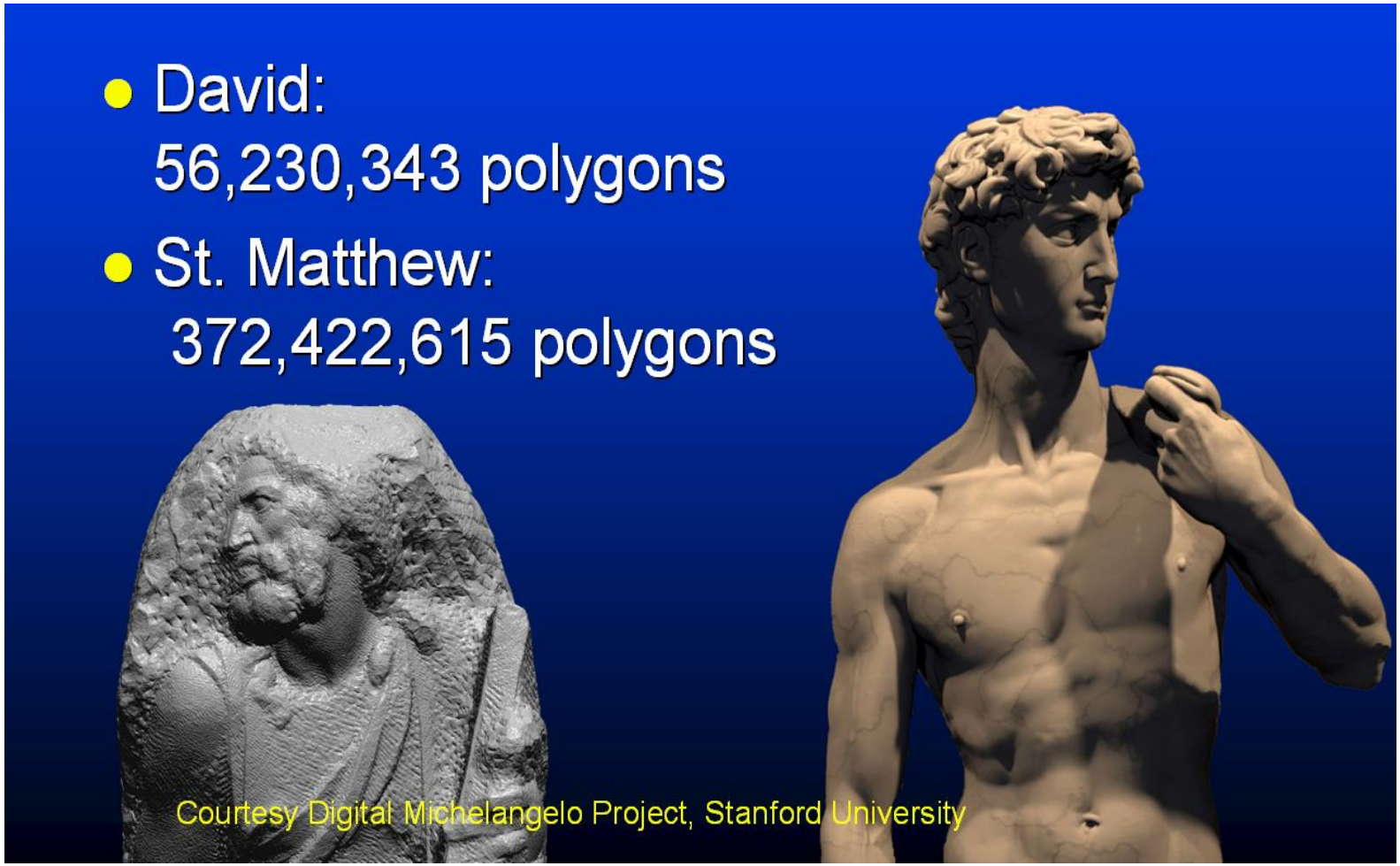
Advantage and disadvantage in polygon representation

- Disadvantage
  - Approximation, it is hard to satisfy real time interaction
  - It is hard to edit mesh with traditional method.
  - Without analytical form, geometric attribute is hard to compute
  - When expressed object with complex topology and rich details, modeling/editing/rendering/storing will have more burden.

Example: Modelling_Meshes



Example: Modelling_Meshes



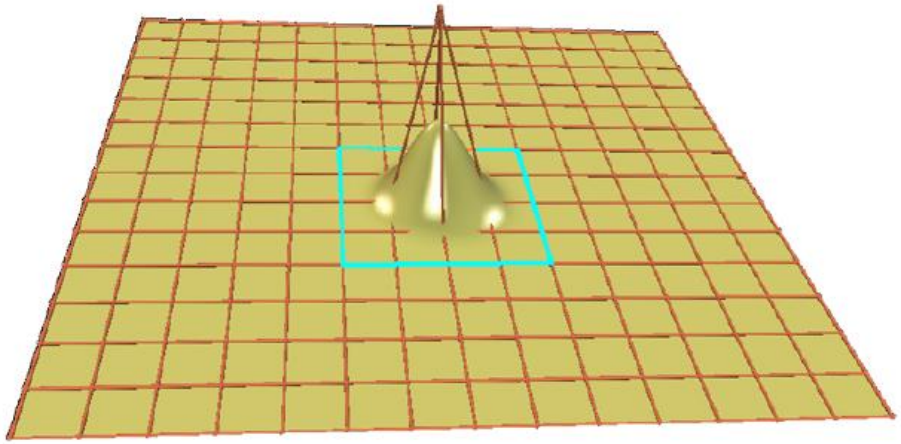
Outline

- Meshes (polyhedra)
  - Data Acquisition
  - Data structure for proximity retrieving
  - Mess processing
    - Subdivision
    - Mesh simplification

Spline Surfaces

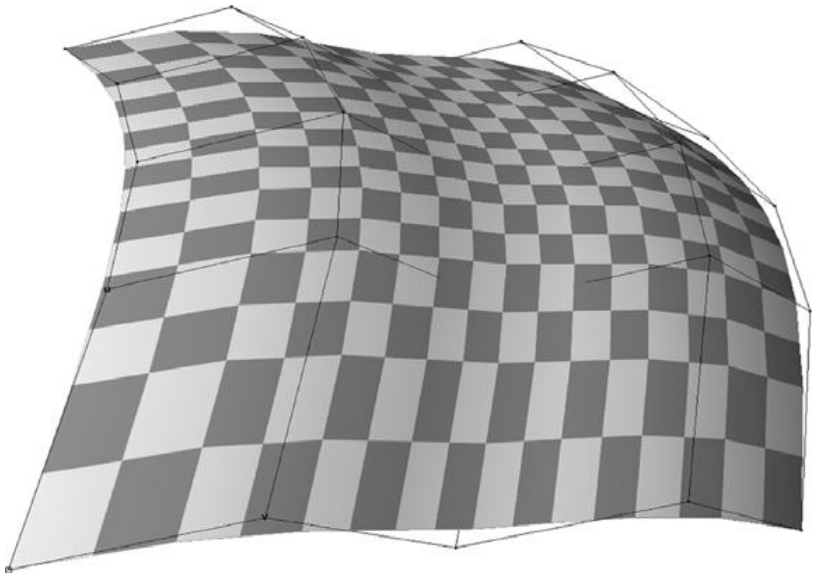
- Tensor product surfaces (“curves of curves”)
  - Rectangular grid of control points

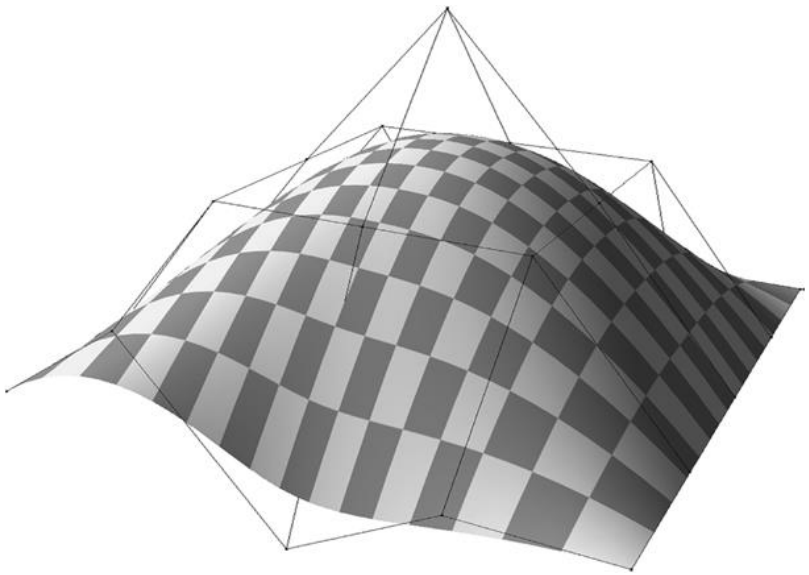
$$\mathbf{p}(u,v) = \sum_{i=0}^{k} \sum_{j=0}^{l} \mathbf{p}_{ij} \, N_i^n(u) N_j^n(v)$$



Spline Surfaces

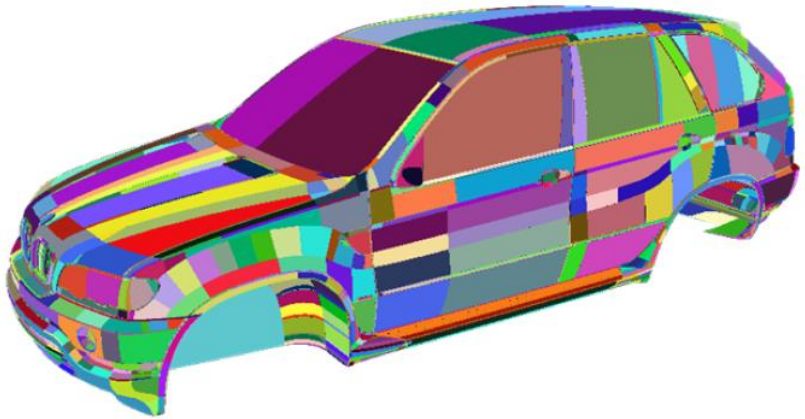
- Tensor product surfaces (“curves of curves”)
  - Rectangular grid of control points
  - Rectangular surface patch





Spline Surfaces

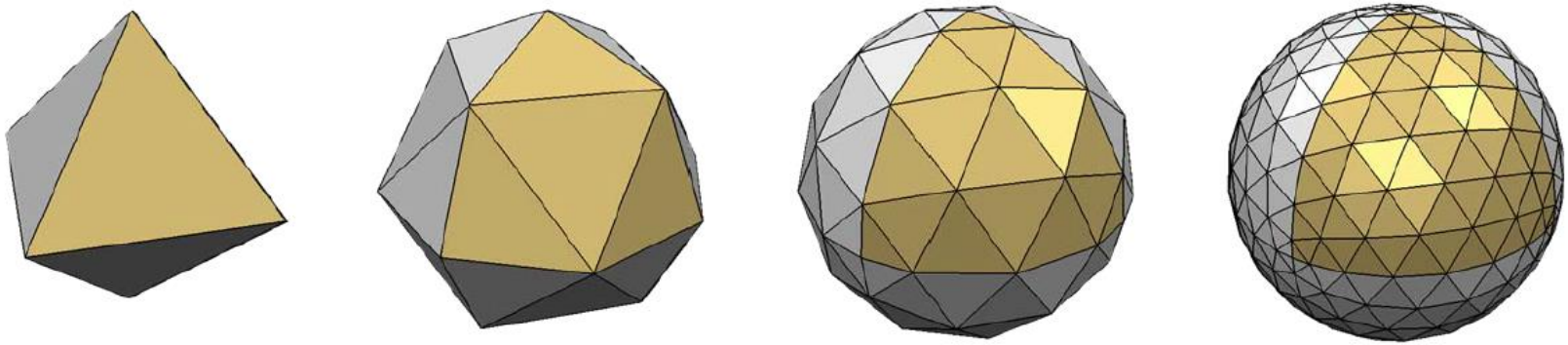
- Tensor product surfaces (“curves of curves”)
  - Rectangular grid of control points
  - Rectangular surface patch



- Problems:
  - Many patches for complex models
  - Smoothness across patch boundaries
  - Trimming for non-rectangular patches

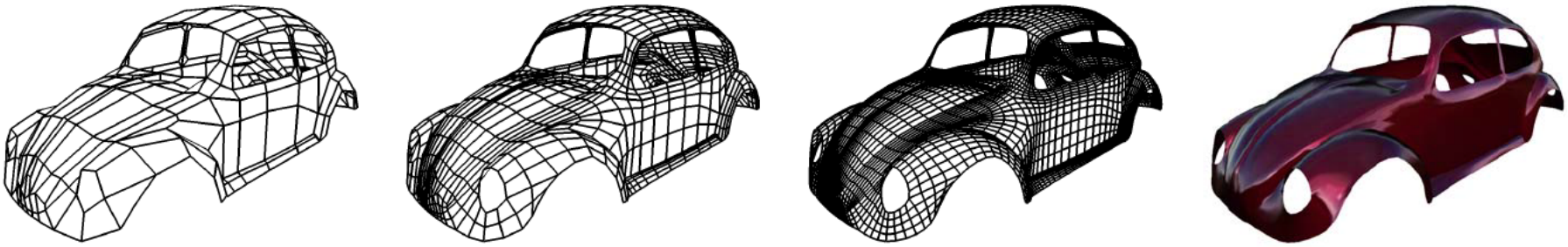
Subdivision

- Generalization of spline curves/surfaces
  - Arbitrary control meshes
  - Successive refinement(subdivision)
  - Converges to Smooth limit surface
  - Connection between splines and meshes



Subdivision

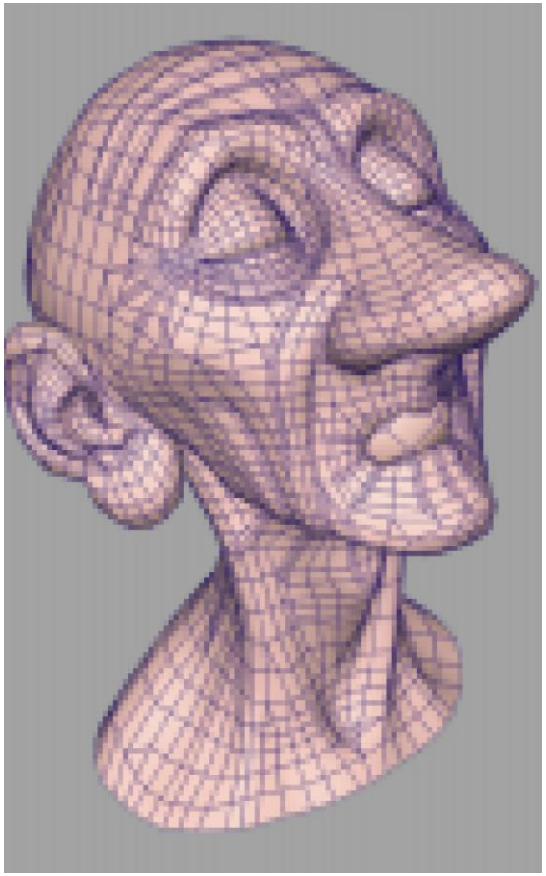
- Generalization of spline curves/surfaces
  - Arbitrary control meshes
  - Successive refinement(subdivision)
  - Converges to Smooth limit surface
  - Connection between splines and meshes



Subdivision

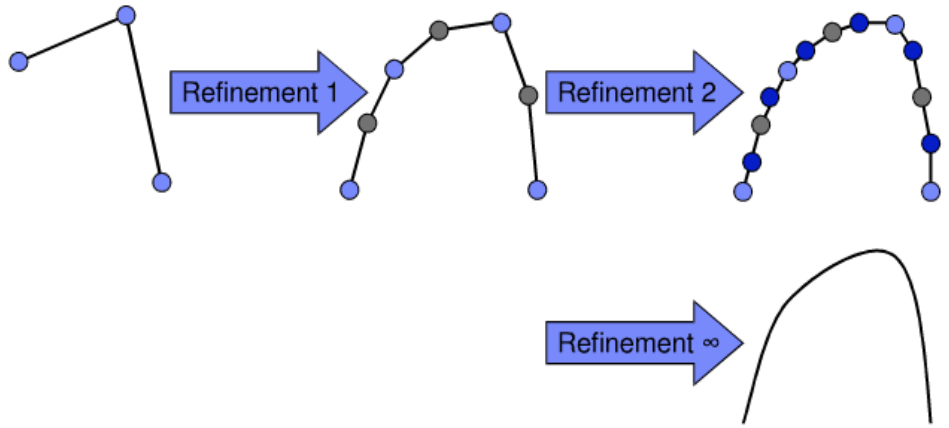
- Subdivision
  - the publication of the papers by Catmull and Clark, Doo and Sabin in 1978 marked the beginning of subdivision for surface modeling. Now we can regularly see subdivision used in movie production.
  - a loose description of subdivision
    - Given a original mesh, generate a smoother one by a sequence of successive refinement

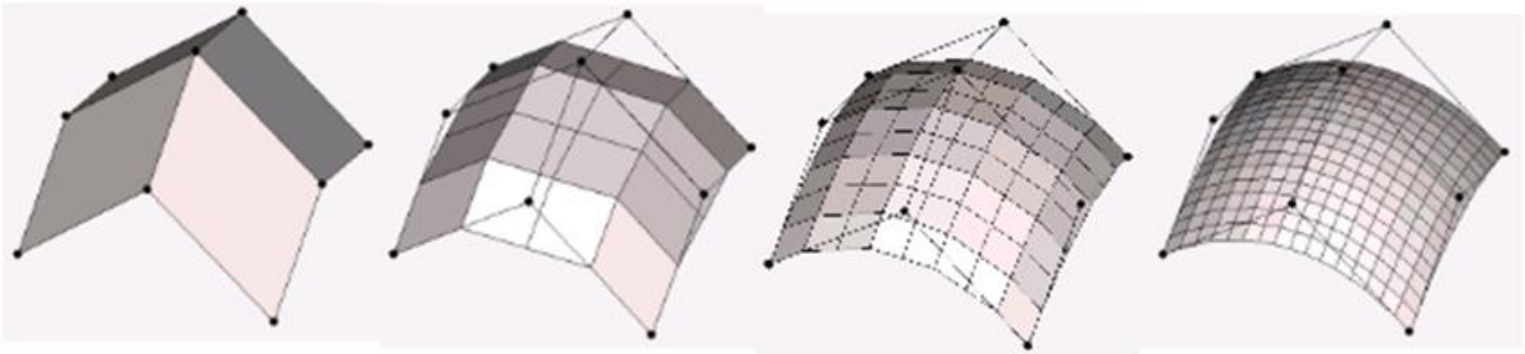
Subdivision



What is Subdivision?

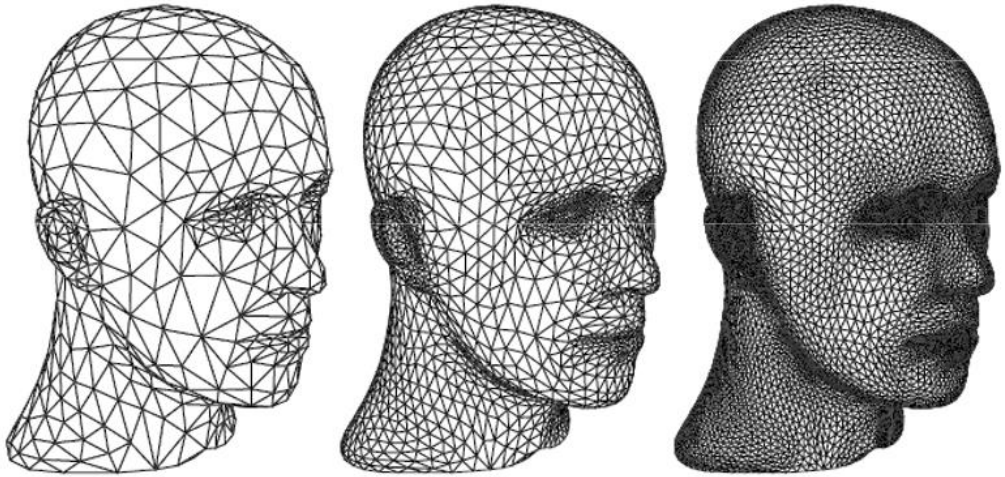
- Subdivision is a process in which a poly-line/mesh is recursively refined in order to achieve a smooth curve/surface.





Subdivision

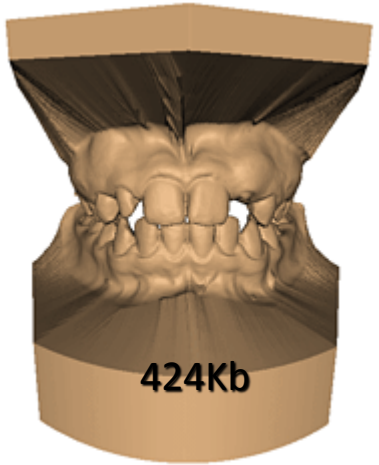
- Another Example of Subdivision

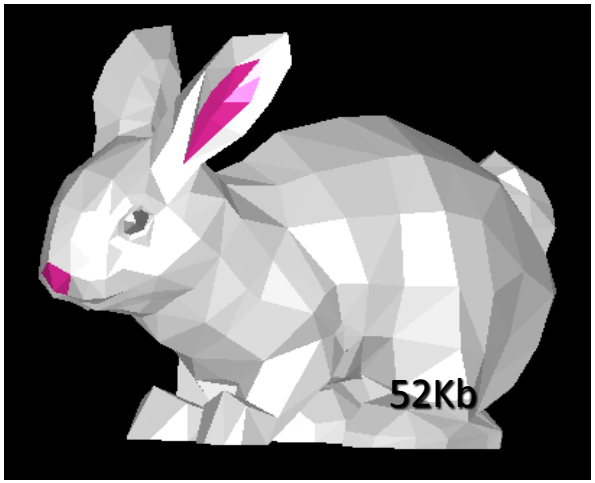


Example of subdivision, showing 3 successive levels of refinement. On the left an initial triangular mesh. Each triangle is split into 4 according to a particular subdivision rule (middle). On the right the mesh is subdivided in this fashion once again.

Why Subdivision?

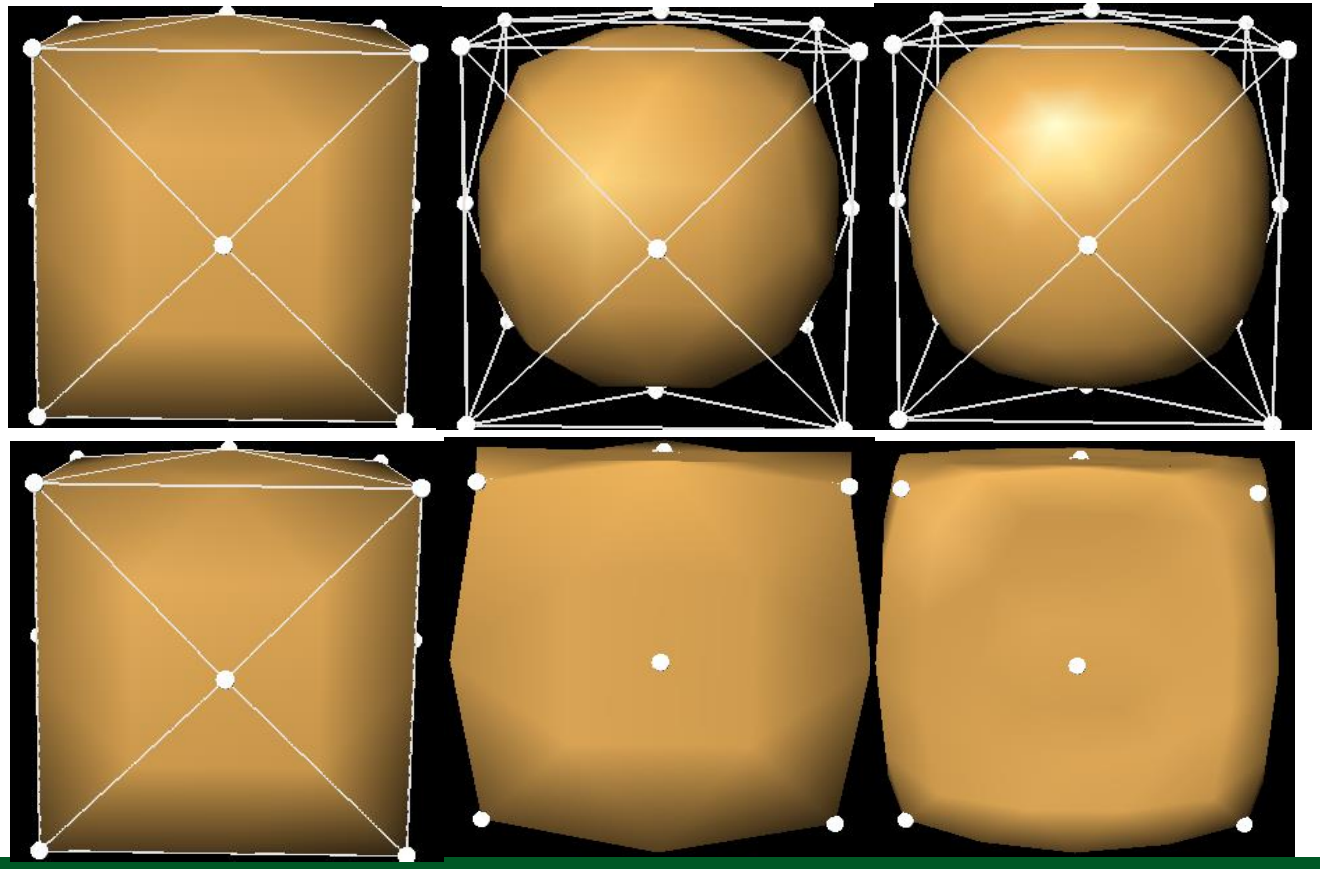
- LOD（Level of Detail）
- Compression
- Smoothing





Type of Subdivision

- Two main groups of schemes:
  - Approximating - original vertices are moved
  - Interpolating – original vertices are unaffected

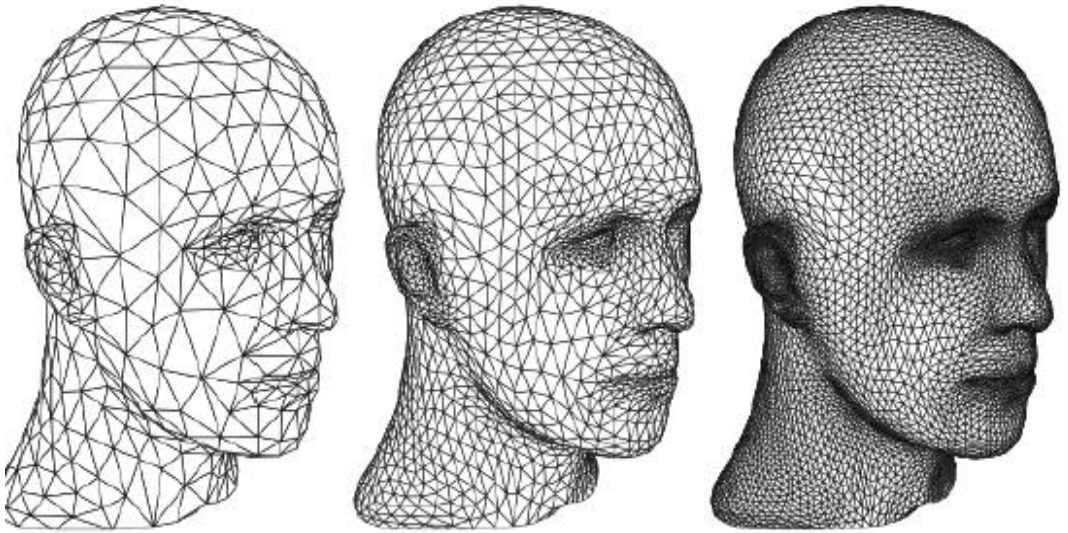


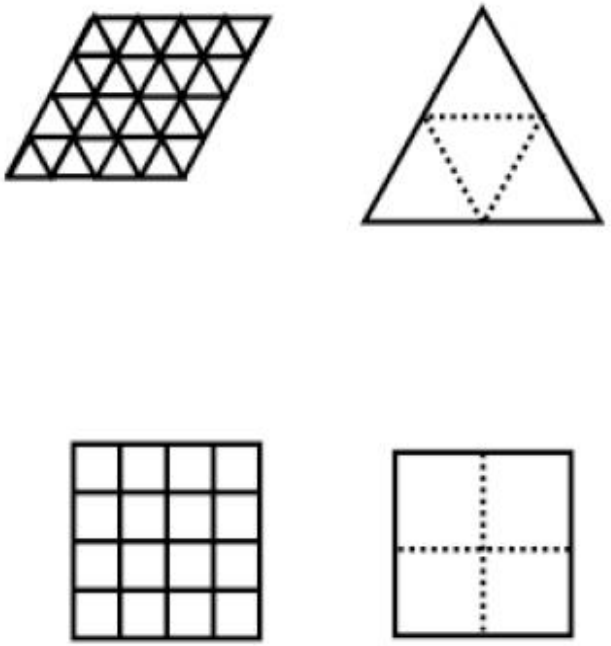
Basic Conceptions of Subdivision surfaces

- Subdivision schemes are usually defined on different types of polygonal meshes
  - Triangular meshes (三角网格)
  - Quadrilateral meshes (四边形网格)
  - Tri/quad meshes (混合网格)

- Valence of vertices (顶点的价)
  - The number of edges adjacent to the vertex

Triangles and quads





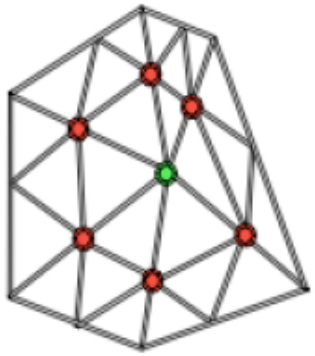
Basic Conceptions of Subdivision surfaces

- Regular vertex and irregular vertex
  - Vertex of valence 4 in a quad mesh is regular; and irregular otherwise
  - Vertex of valence 6 in a tri. mesh is regular; and irregular otherwise
- At each iteration
  - Refine mesh
  - Increase number of vertices (approximately) * 4
- Mesh vertices converge to a limit surface
  - After infinite number of subdivision steps

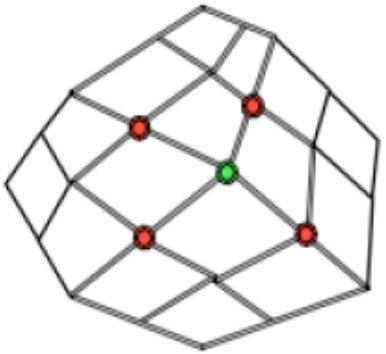
Extraordinary Vertices

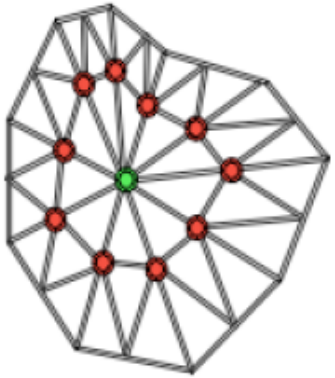
Triangle meshes

Quad meshes

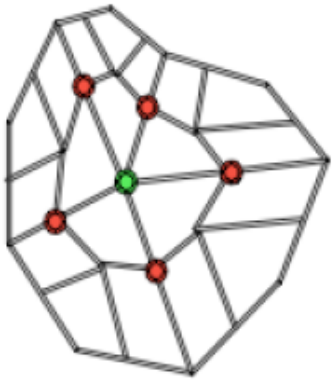


regular





extraordinary

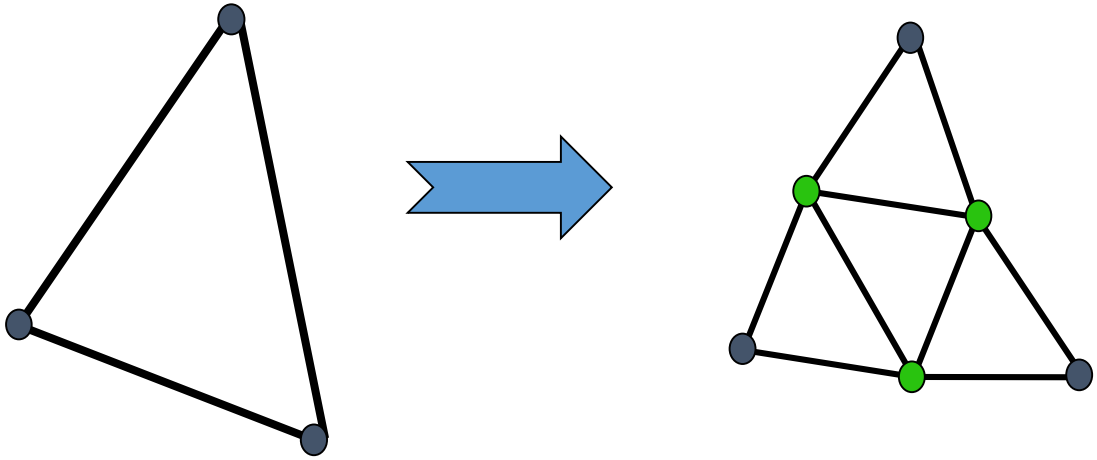


Basic Conceptions of Subdivision surfaces

- Every subdivision method has:
  - A rule to refine the mesh connectivity
  - Rules to calculate location of new vertices and old if they are effected
- A scheme always consists of 2 main parts:
  - A rule to generate the **topology** of the new mesh.
  - Rules to determine the **geometry** of the vertices in the new mesh.

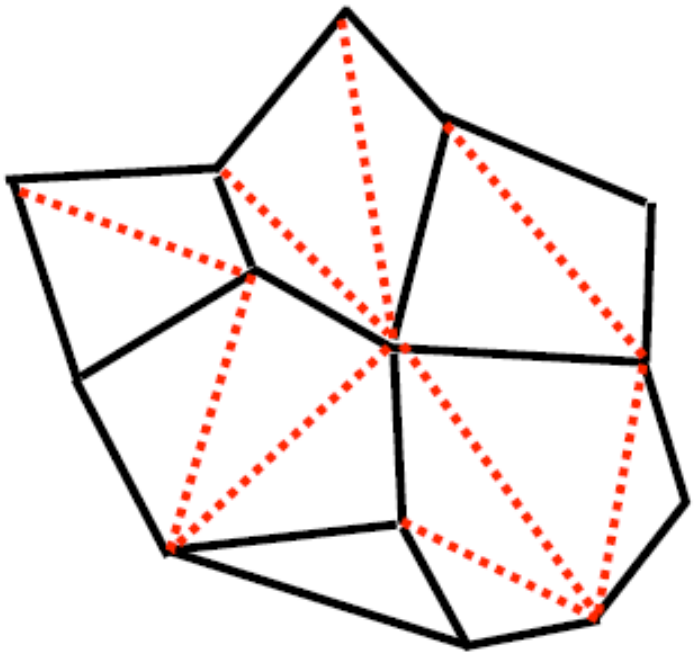
Triangular Subdivision

- Works only for triangle meshes



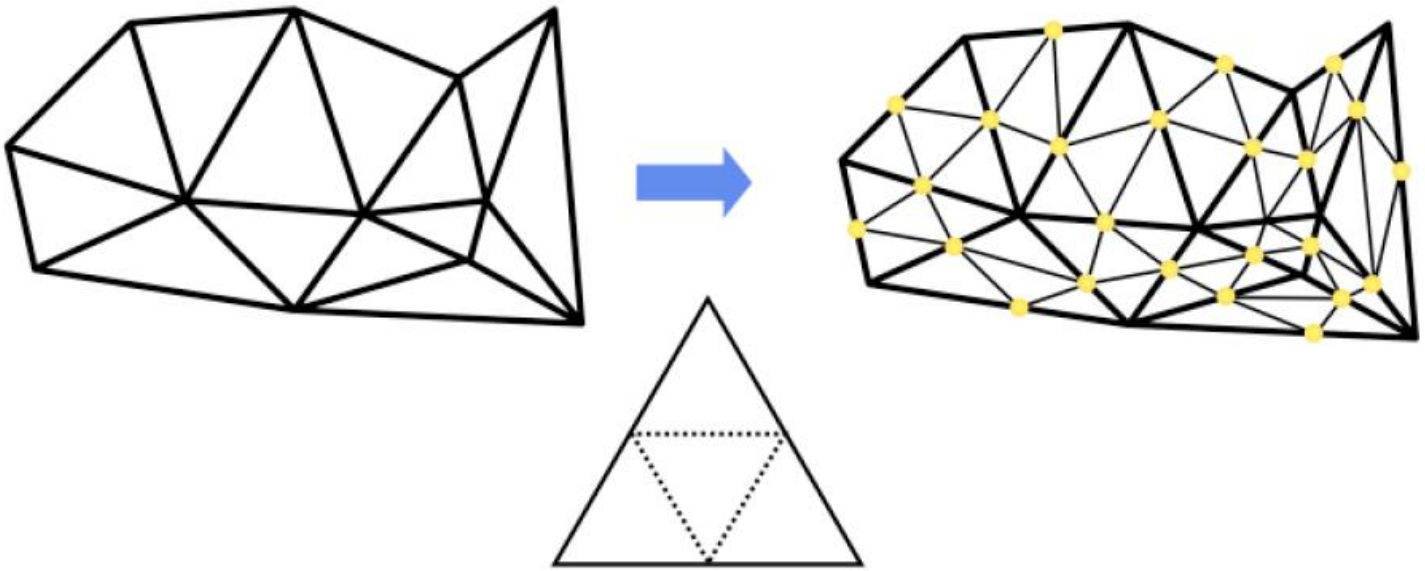
- Every triangle replaced by 4 new triangles
- Two kinds of new vertices:
  - Green vertices associated with old edges
  - **Black** vertices associated with old vertices

Reduction to a triangular mesh



Example: Loop Scheme

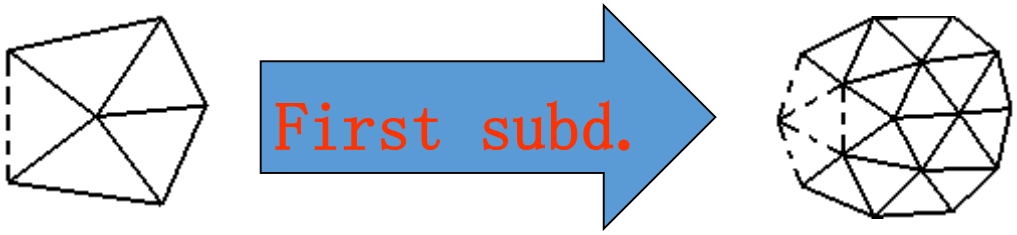
Refinement rule



regular vertex insertion
"quadrisection"

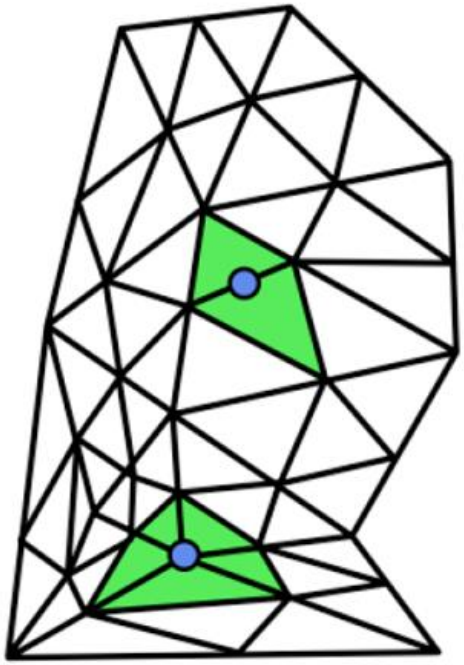
Loop subdivision surfaces

- Approximating
- Topologic rules: insert a new vertex to each edge



Two geometric rules:

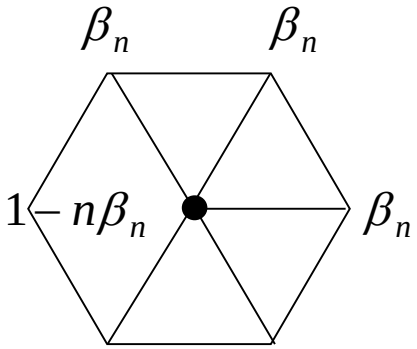
- **even (update old points)**
- **odd (insert new)**



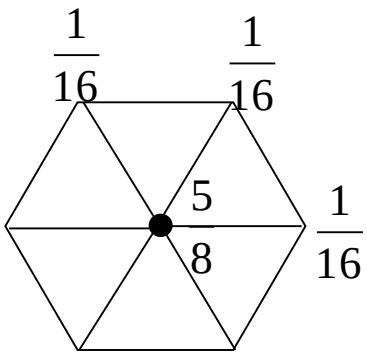
Loop Scheme

- **Geometric rules**

1. inner vertex

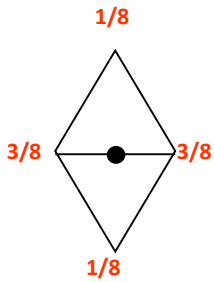


V-顶点, valence n



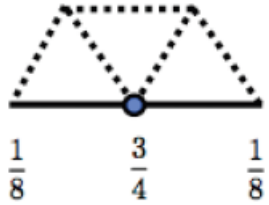
n = 6

$$\beta_n = \frac{1}{n}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2\right)$$

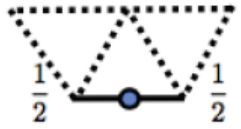


E-边点

2. Boundary vertex

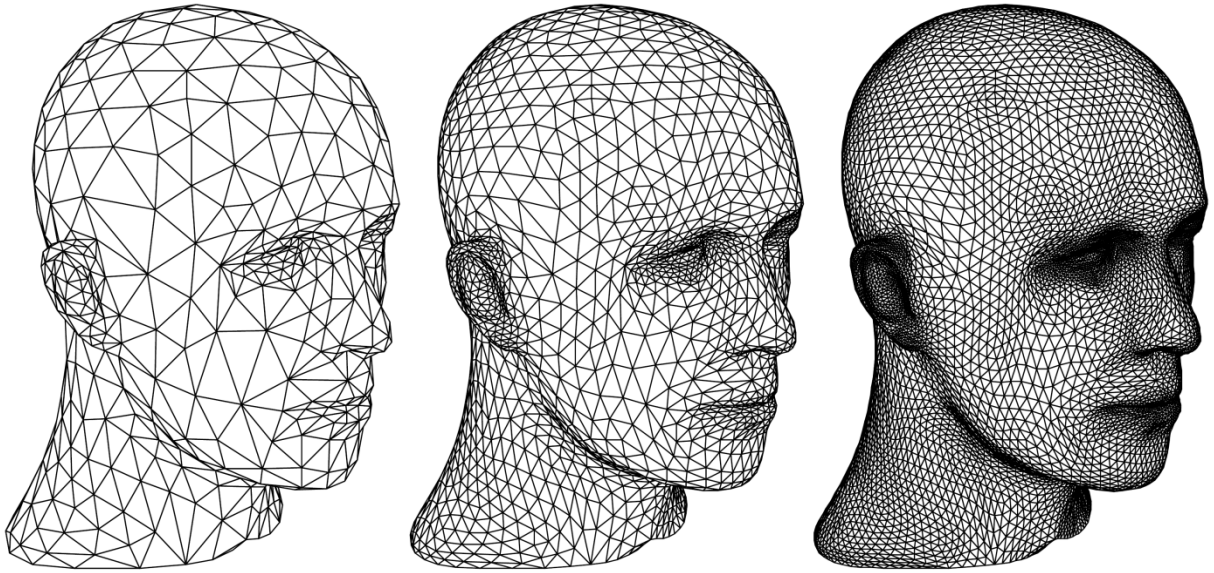


V-顶点

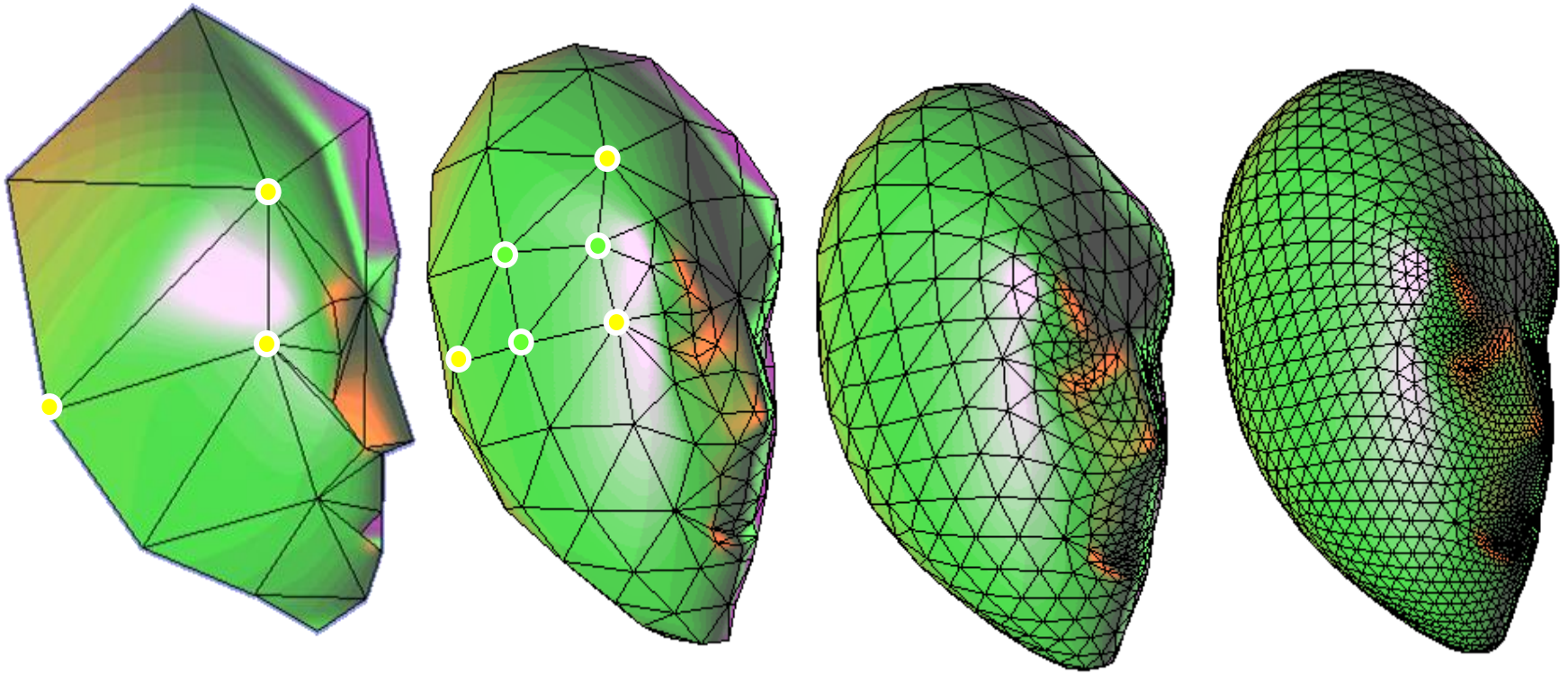


E-边点

Loop subdivision: example



Loop subdivision: example



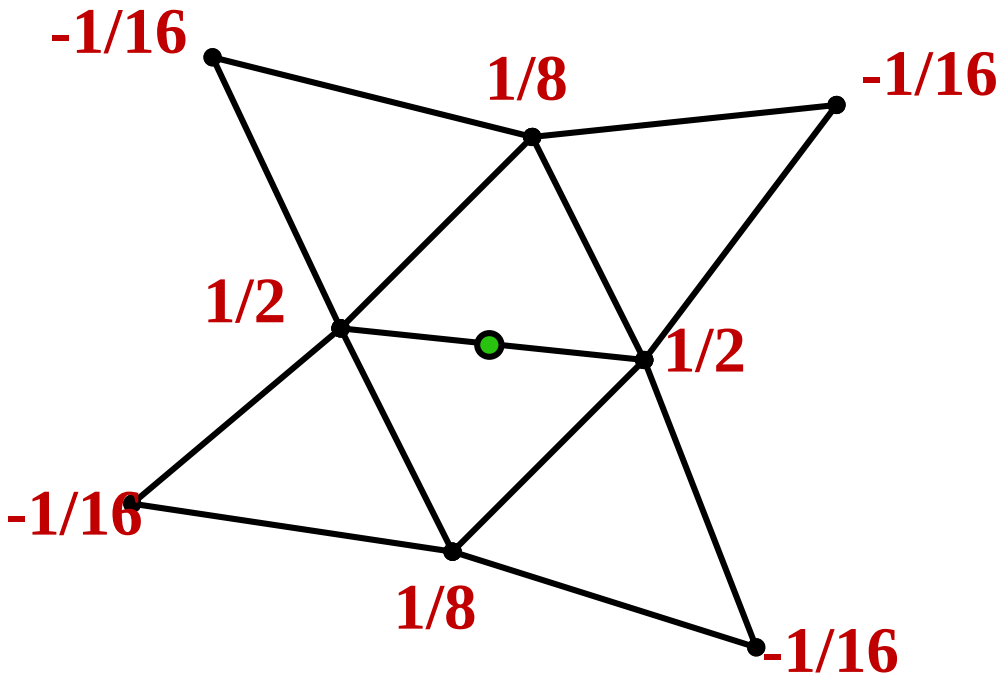
Initial mesh | 1st iteration | 2nd iteration | Limit surface

The Limit Surface

- Limit surfaces of Loop's subdivision is C_2 almost everywhere
- Finite set of singular locations where the surface is C_1.

Butterfly Subdivision

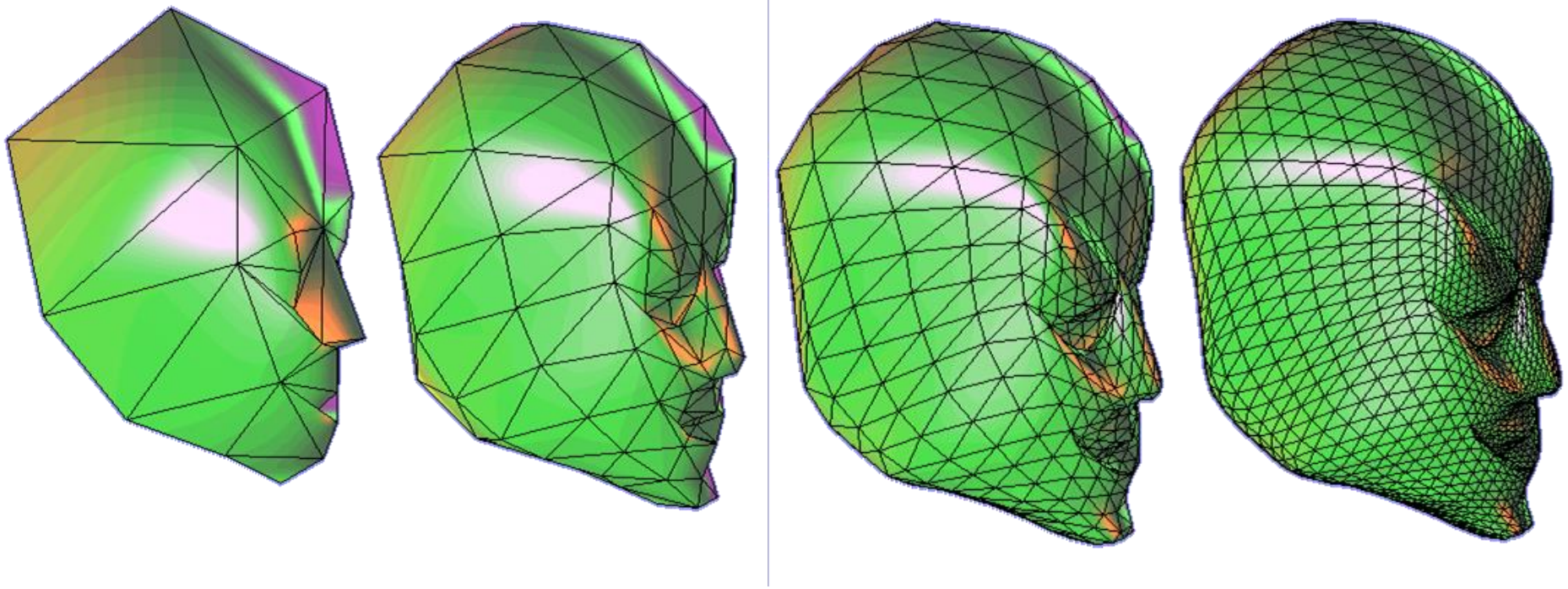
- Interpolating scheme
- New black vertices inherit location of old vertices
- New green vertices computed by following stencil



The Limit Surface

- Limit surfaces of Butterfly's subdivision is C_1 almost everywhere
- Finite set of singular locations where the surface is C_0.

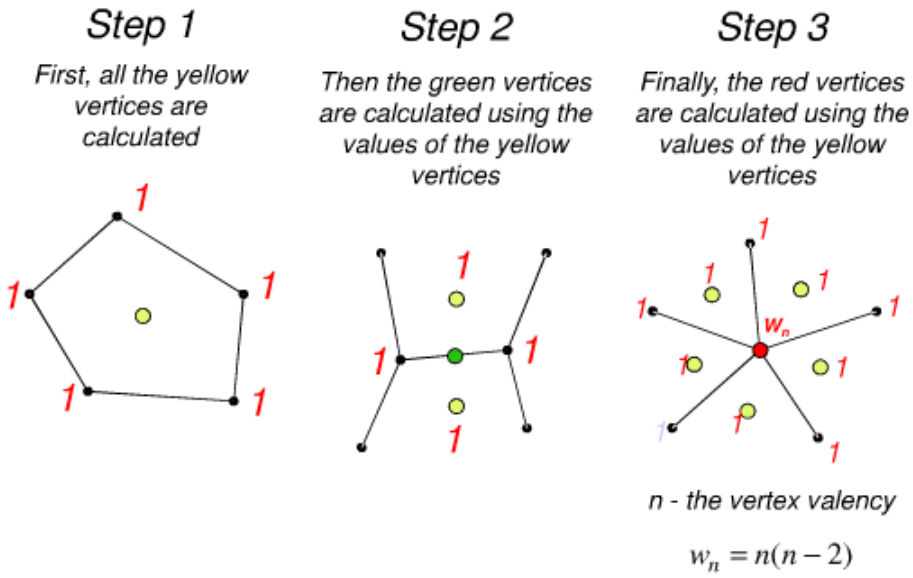
Example: Butterfly



Initial mesh 1st iteration 2nd iteration Limit surface

Catmull-Clark subdivision surfaces

- Catmull–Clark surfaces are defined on a mesh of an arbitrary polyhedron.



Catmull-Clark subdivision surfaces

- FACE

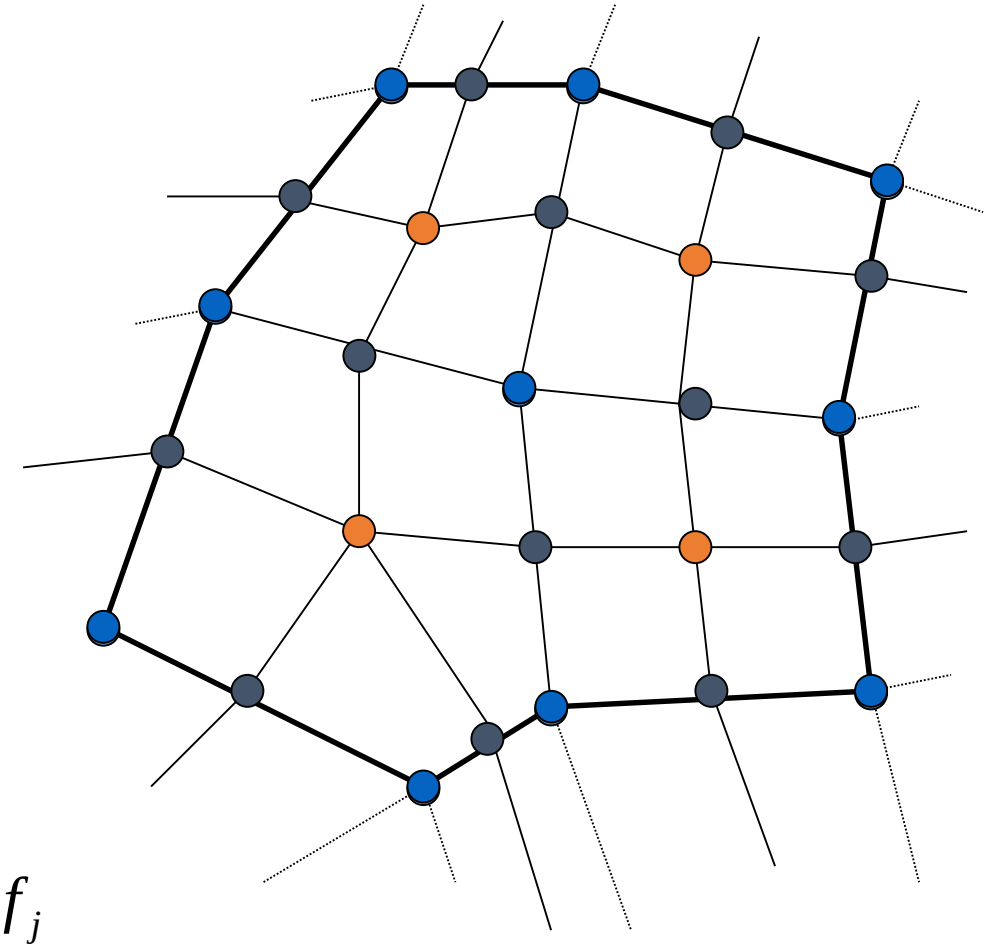
$$f = \frac{1}{n}\sum_{1}^{n} v_i$$

- EDGE

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

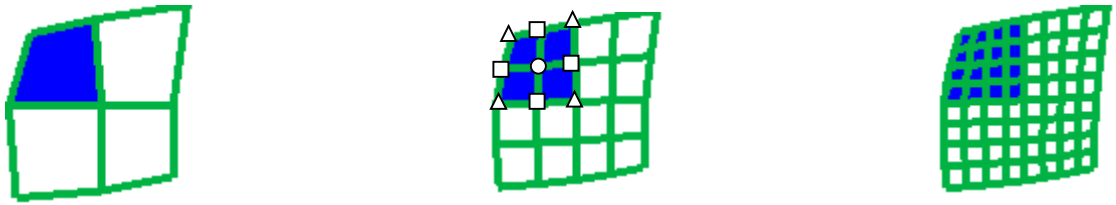
- VERTEX

$$v_{i+1} = \frac{n-2}{n} v_i + \frac{1}{n^2}\sum_j e_j + \frac{1}{n^2}\sum_j f_j$$



Catmull-Clark subdivision surfaces

- Topologic rules: insert a vert. to each edge and each face, the old vert. is reserved(but it isn’t the same.).

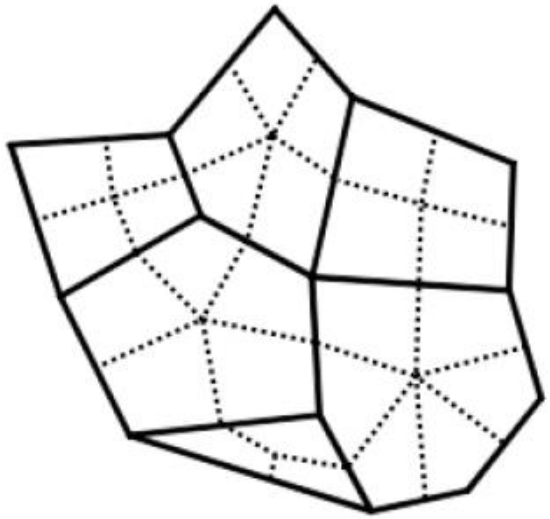


- The new mesh will consist only of quadrilaterals, which won't in general be planar . The new mesh will generally look smoother than the old mesh.

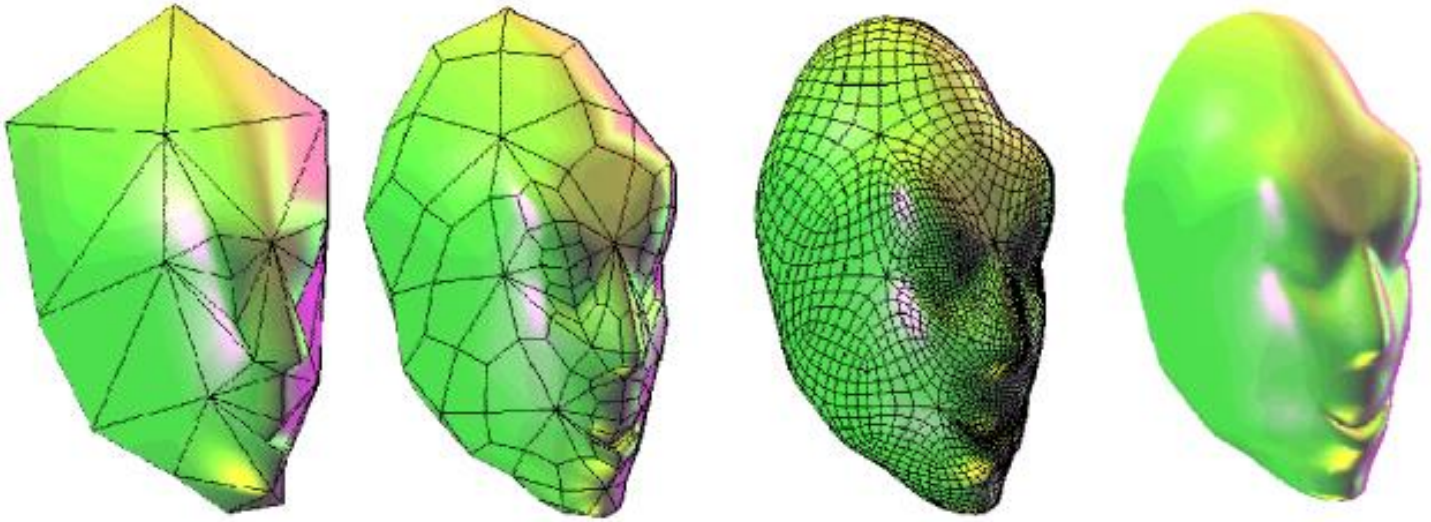
Catmull-Clark Scheme

Reduction to a quadrilateral mesh

- **do one step of subdivision with special rules: only quads remain**



Example: Catmull-Clark



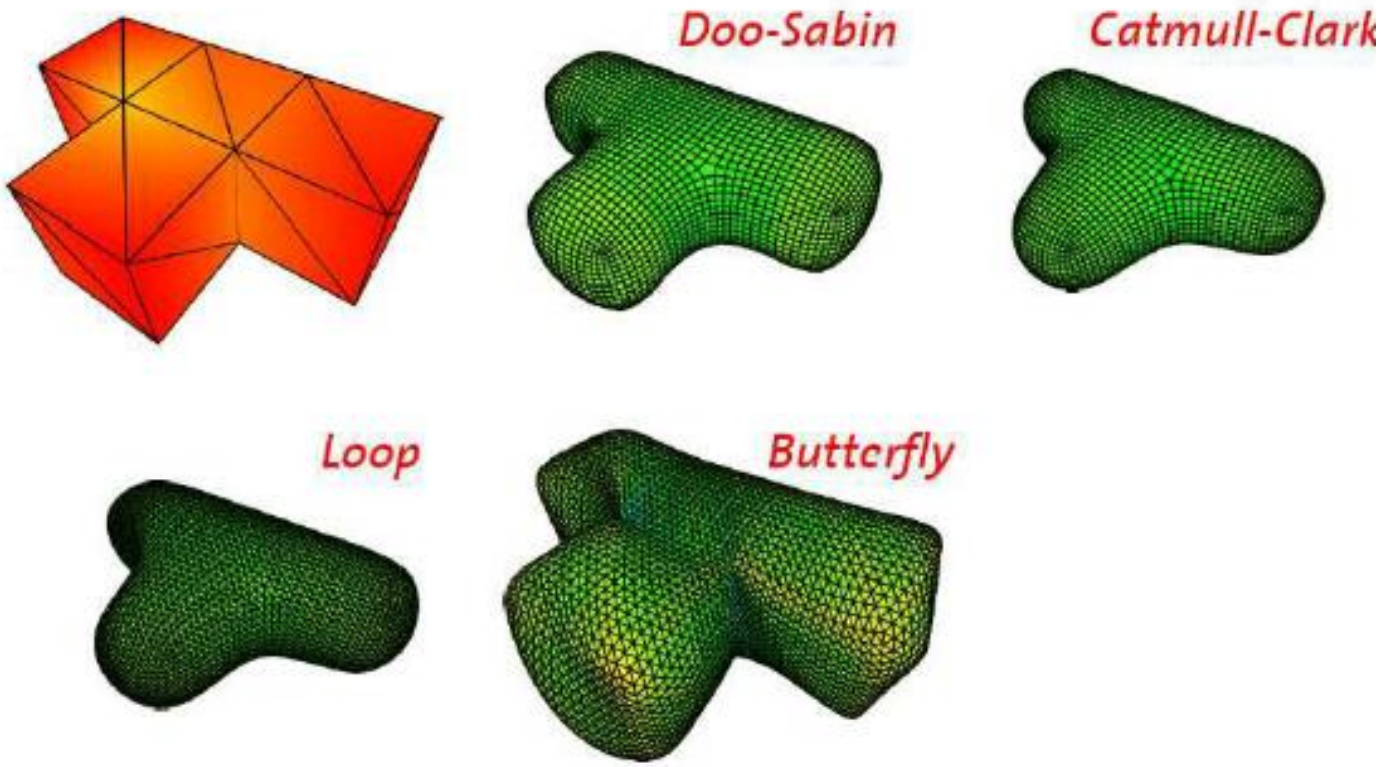
The original control net

After 1st iteration

After 3rd iteration

The limit surface

Comparison of Scheme



Subdivision Surface Summary

- **Advantages**
  - Simple method for describing complex surfaces
  - Relatively easy to implement
  - Arbitrary topology
  - Local support
  - Guaranteed continuity (C^1 or C^2 continuous)
  - Multi-resolution
  - Be fit for animation, rapid rendering
- **Difficulties**
  - No analytic expression at irregular vertices (奇异点处没有解析表达，难以计算微分量)
  - It is difficult to construct surfaces of higher order continuity

Mesh Simplification

- Surface mesh simplification is the process of reducing the number of faces used in a surface mesh while keeping the overall shape, volume and boundaries preserved as much as possible. It is the opposite of subdivision.

Why do we need Mesh Simplification?

- Remove redundant geometry
  - Eg. A flat region with many small, coplanar triangles. Merging these triangles to large polygons could decrease model's complexity.
- Reduce mode size
  - Reduce the size to store or transmit it.
- Improve run-time performance
  - Simplification can be essential to efficient rendering
  - Generating levels of detail of objects in a scene. Presenting distant objects with a lower LOD and near object with a higher LOD.

How to simplify?

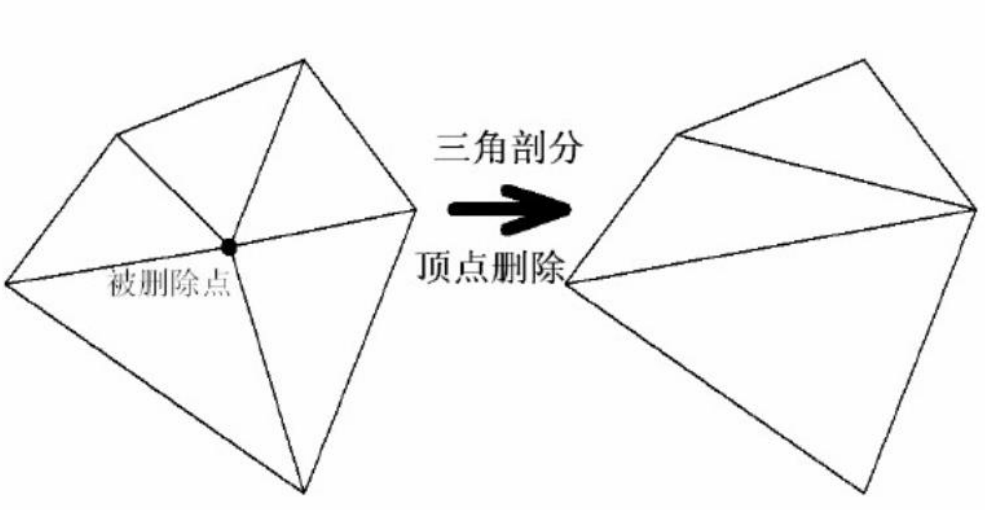
- Nearly every simplification technique in the literature uses some variation or combination of four basic polygon removal mechanisms below:
  - Sampling
  - Decimation
  - Vertex merging

Mechanisms

- Sampling
  - Sampling algorithms sample the initial model's geometry, with points on the model's surface.
  - They may have trouble to sample the high frequency features accurately. These algorithms usually work best on smooth surfaces with no sharp corners.

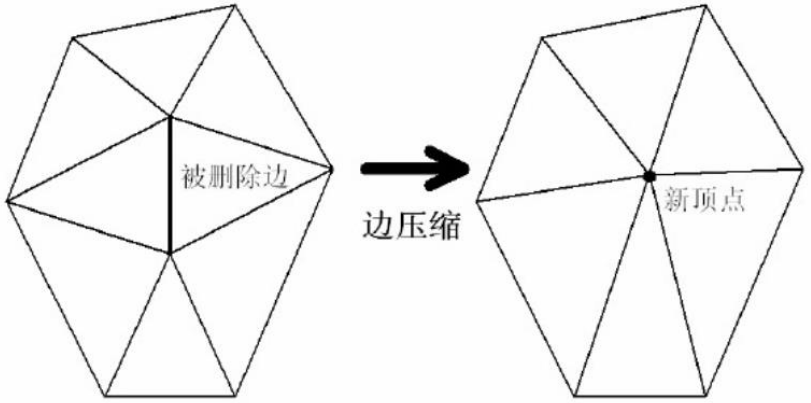
Mechanisms

- Decimation（去除）
  - Decimation techniques iteratively remove vertices or faces from the mesh, re-triangulating the resulting hole after each step.
  - These algorithms are relatively simple to code and can be very fast. Most use strictly local changes that tend to preserve the genus. These algorithms are good at removing redundant geometry such as coplanar polygons.



Mechanisms

- Vertex-merging
  - Vertex-merging schemes operate by collapsing two or more vertices of a triangulated model together into a single vertex, which in turn can be merged with other vertices.
  - Vertex-merging is a simple and easy-to-code mechanism, but algorithms use various techniques to determine which vertices to merge in what order.
  - Edge-collapse algorithms, which always merge two vertices sharing an edge, tend to preserve local topology, but algorithms permitting general vertex-merge operations can modify topology and aggregate objects.



Category

- Static simplification
- Dynamic simplification
- View-dependent simplification

Static simplification

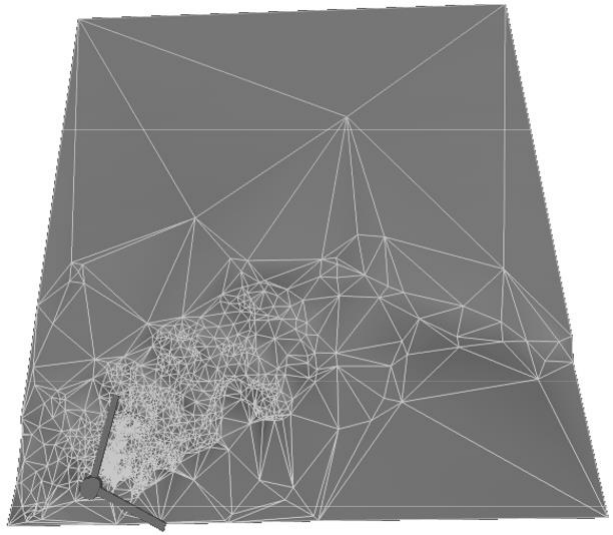
- Creates several discrete simplified versions of each model in a preprocess, each at a different level of detail. At runtime, rendering algorithms choose the appropriate LOD to represent the object.
- Static simplification has many advantages. Decoupling simplification and rendering makes this simplest model to program. The simplification algorithm generates LODs without regard to real-time rendering constraints, and the rendering algorithm simply chooses which LODs to render.

Dynamic simplification

- Dynamic polygonal simplification is different from the traditional static approach. Whereas as a static simplification algorithm creates individual LODs during the preprocessing stage, a dynamic simplification system creates a data structure encoding a continuous level of detail.
- A major advantage of this approach is better continuity and granularity, rather than choosing from a few pre-created simplified models, which would produce gap when switching from one model to another one.

View-dependent simplification

- Extends dynamic simplification by using view-dependent criteria to select the most appropriate LOD for the current view.
- In a view-dependent system, a single object can span multiple levels of simplification. For Instance, nearby portions of the object may appear at a higher resolution than distance portions, or silhouette（轮廓）regions of the object may appear at a higher resolution than interior regions.



Simplification Algorithms

- We will introduce two algorithms for simplification in details:
  - Vertex decimation
  - Edge contraction

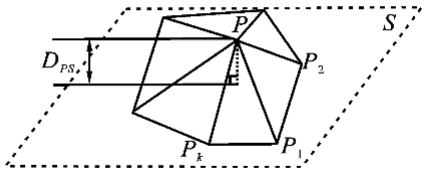
Vertex decimation

- First proposed by [Schroeder et.al. 1992], operates on a single vertex by deleting that vertex and re-tessellating the resulting hole.
- Typically consists of:
  - select the vertex to delete by some error metric
  - based on the adjacent information of the selected vertex, determine how to re-tessellate the hole
  - Recursively delete vertices using step 1&2

Vertex decimation

- Error metric to select vertex
  - For each vertex v
    - Consider the triangles in v's neighbourhood, and compute an approximating plane, based on the area-weighted average of the triangle normals n, centers x, and areas A:

$\text{Import}(P) = D_{PS}$



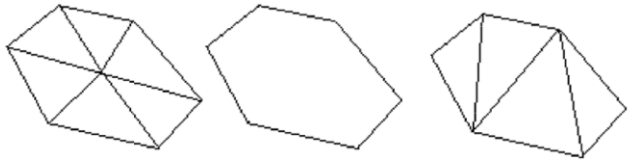
$$d = |\mathbf{n} \cdot (v - \mathbf{x})|$$

$$\mathbf{n} = \frac{\sum_k A_k \mathbf{n}_k}{\sum_k A_k} \qquad \hat{\mathbf{n}} = \frac{\mathbf{n}}{|\mathbf{n}|} \qquad \mathbf{x} = \frac{\sum_k A_k \mathbf{x}_k}{\sum_k A_k}$$

Vertex decimation

- Error metric to select vertex
  - Then calculate the distance from vertex to this plane as the "error metric"
  - Select the vertex with the minimal distance and remove this vertex
  - Re-tessellate the hole caused by deleting of the vertex
- This metric ensures that vertices in smooth regions will be decimated before vertices that define sharp features

- Illustration



  - left: before deleting vertex
  - middle: after deleting vertex
  - right: re-tessellate
- Refer to paper: [Schroeder92] "Decimation of triangle meshes "

Edge contraction

- Originally proposed in [Hoppe et al. 1993], is the most common simplification operation used in computer graphics.
- An edge contraction operates on a single edge{v_i,v_j} and contracts that edge to a single vertex {v_h}, updating all edges previously connected on {v_i} and {v_j} to reference {v_h}.

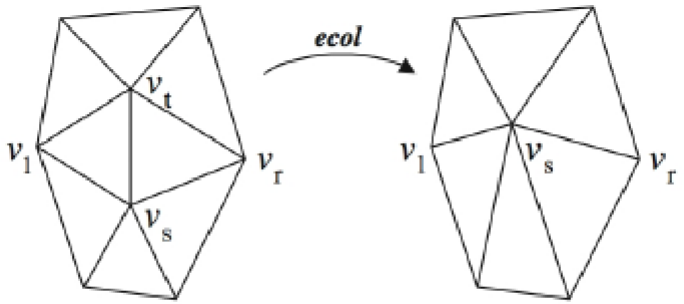


Fig. 1. The edge contraction operation [Hoppe 1993].

Edge contraction

- Select an edge: based on the length of the edge.
- New vertex placement
  - However, for a given contraction edge {i,j}→ vertex {h}, it is not immediately clear what value should be assigned to vertex {h}, i.e. where the resulting vertex should be placed.
  - Obvious choices such as

$$\mathbf{v}_h = \mathbf{v}_i, \ \mathbf{v}_h = \mathbf{v}_j, \text{ or } \mathbf{v}_h = (\mathbf{v}_i + \mathbf{v}_j)/2$$

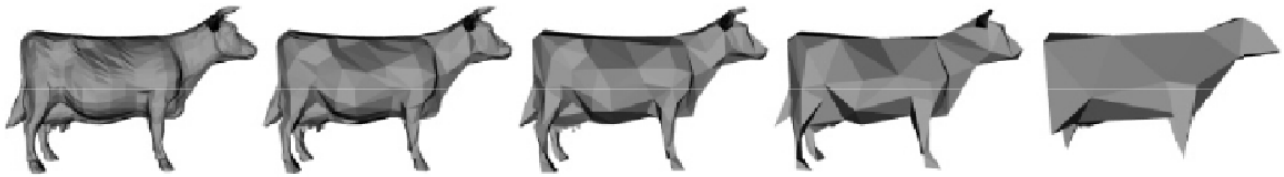
are convenient, but not necessarily optimal (最优)

Edge contraction

- Error metric for new vertex placement [Garland and Heckbert 1997]
  - Consider the newly generated triangles around the new vertex
  - Calculate the sum S of square of distances from vertex v_i and v_j to these triangles
  - Find the v_h that makes that the sum minimal
- Refer to the paper [Garland and Heckbert 1997] “Surface simplification using quadric error metrics”

Edge contraction

- Result

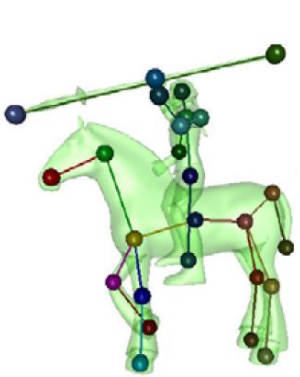


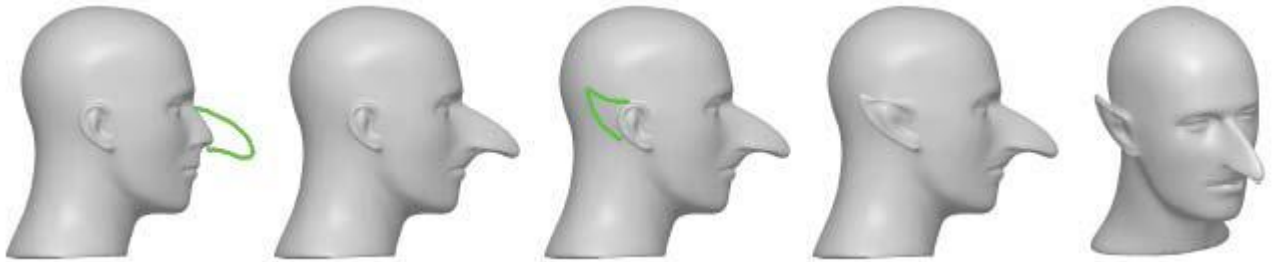
  - A sequence of simplified models generated by an edge contraction algorithm [Garland and Heckbert 1997]

Shapes and Deformations

- Why deformations?
  - Sculpting, customization
  - Character posing, animation

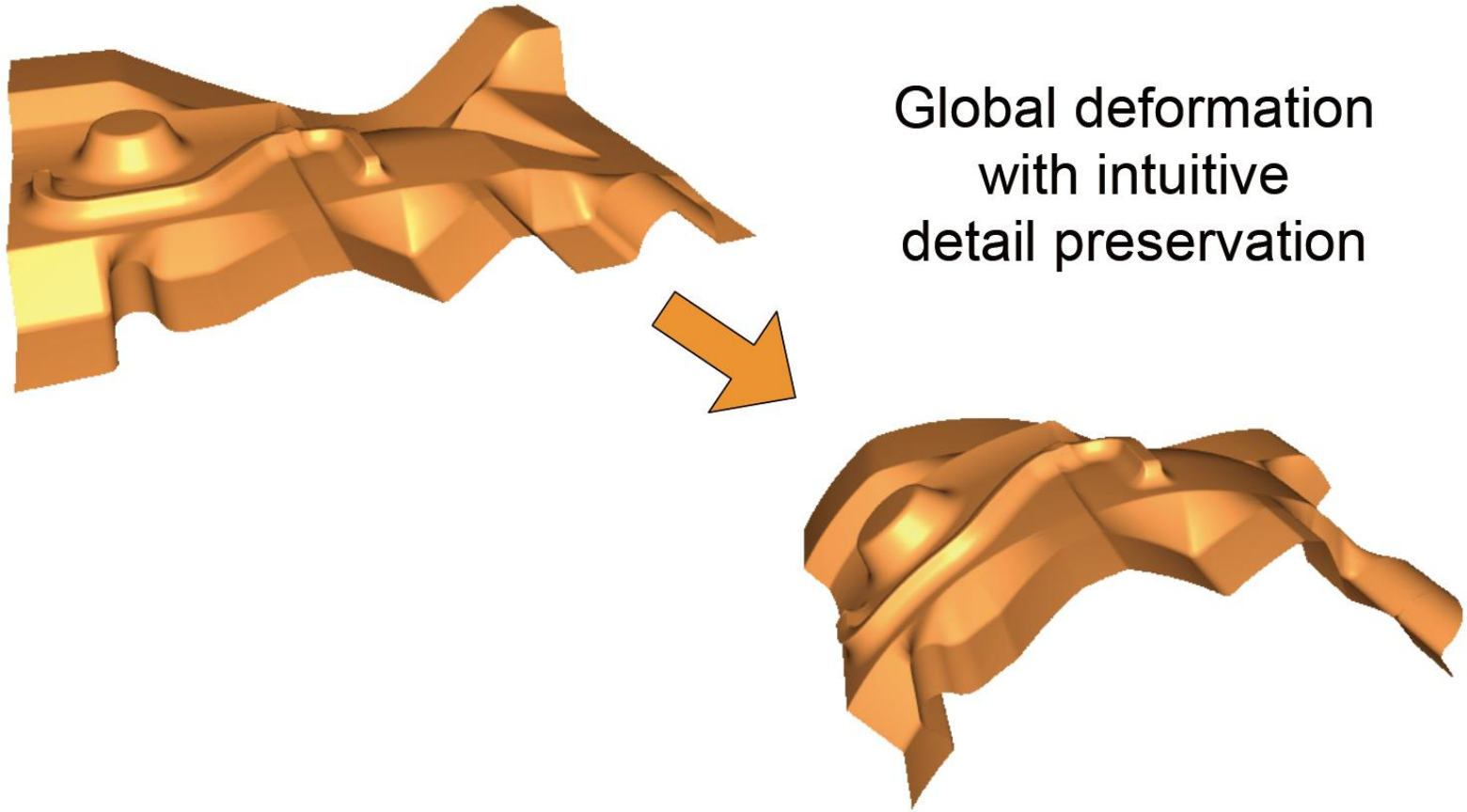
- Criteria?
  - Intuitive behavior and interface
  - Interactivity



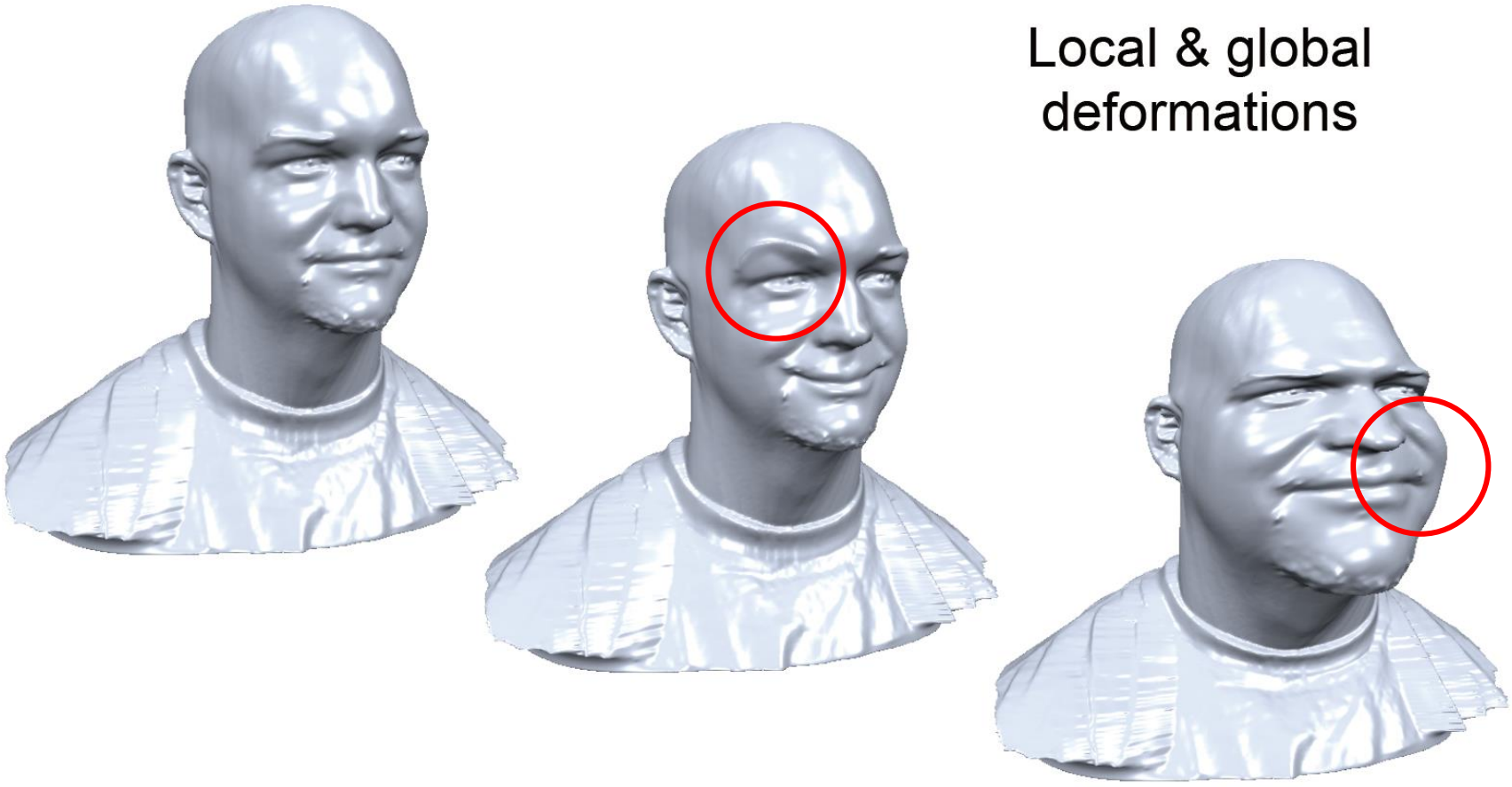


Linear Surface-Based Deformation

- Mesh Deformation



Mesh Deformation



Polygon Mesh Processing

- http://www.pmp-book.org/
- “Geometric Modeling Based on Polygonal Meshes”
- https://hal.inria.fr/inria-00186820/document