1.2 Surface Representation & Data Structures

Hao Li

http://cs621.hao-li.com

Administrative

- No class next Tuesday, due to **Siggraph deadline**
- Introduction to first programming exercise on Jan 25th

Siggraph Deadline 2013@ILM!

Last Time

Geometry Processing

Capture

Reconstruction

Analysis

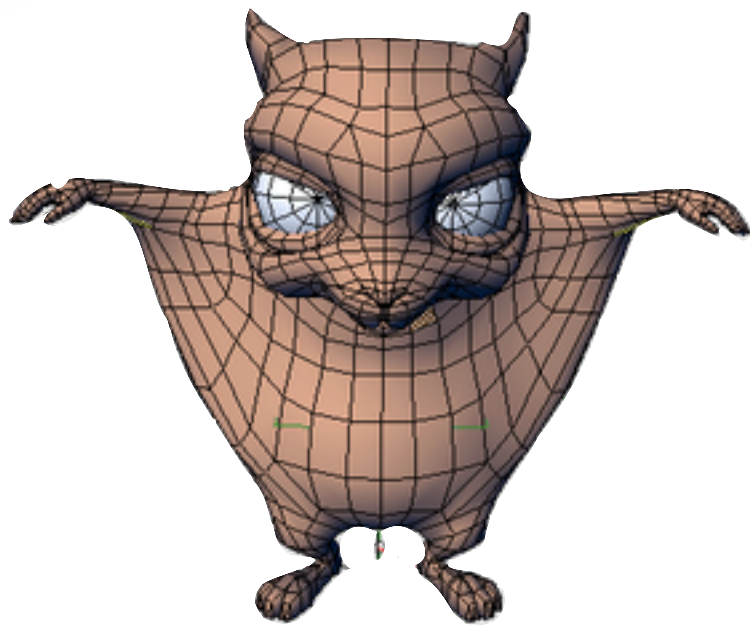
Manipulation

Rendering

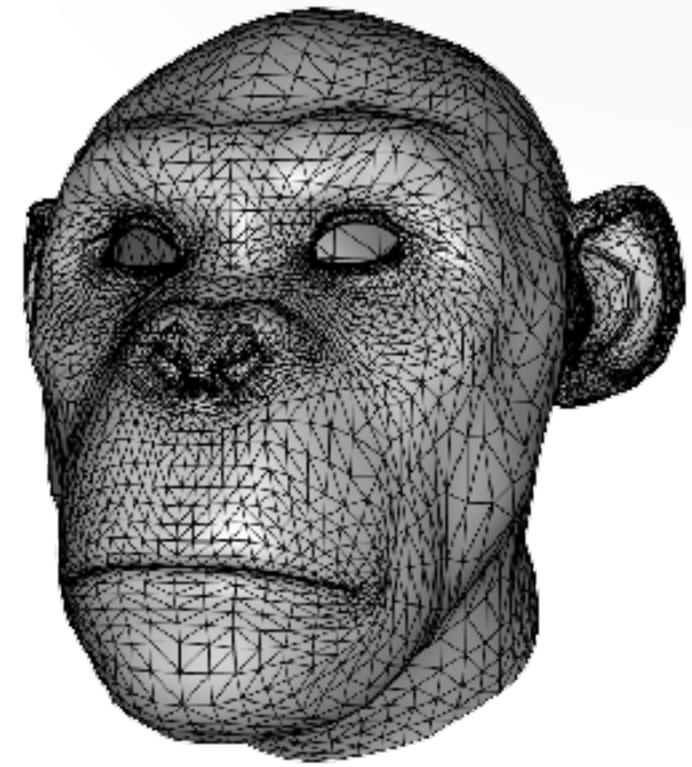
Reproduction

Geometric **Representations**

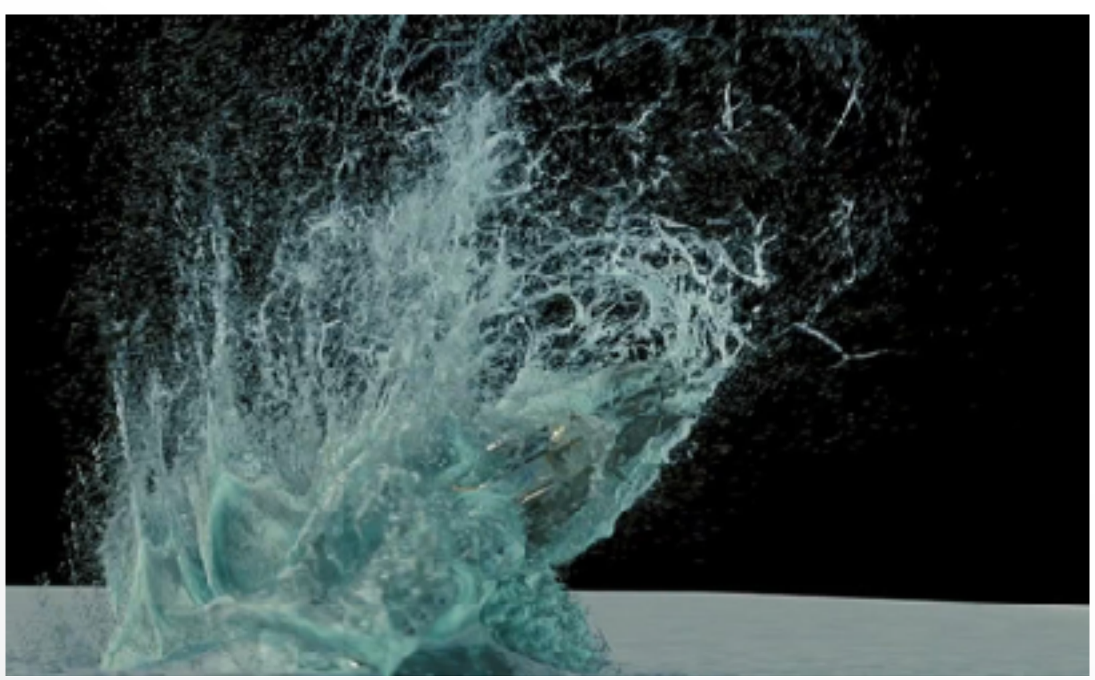
point based



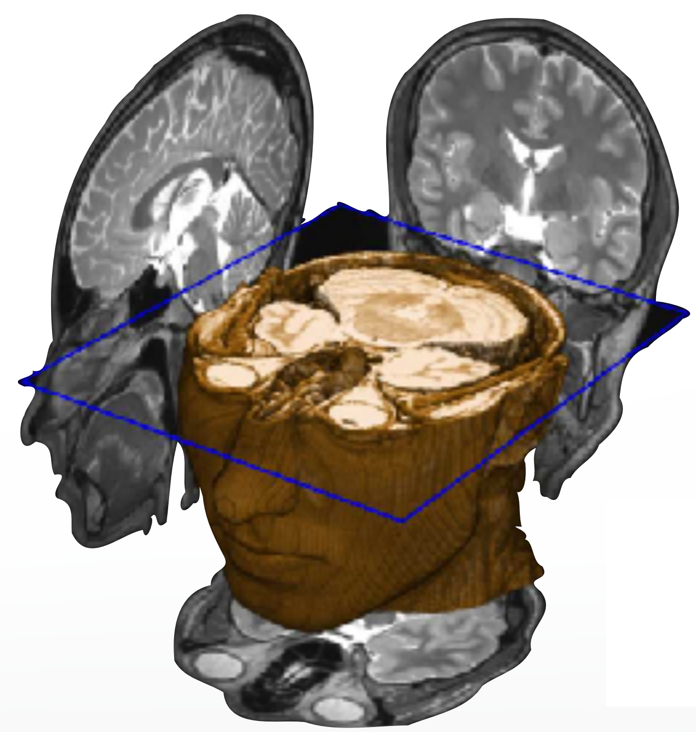
quad mesh



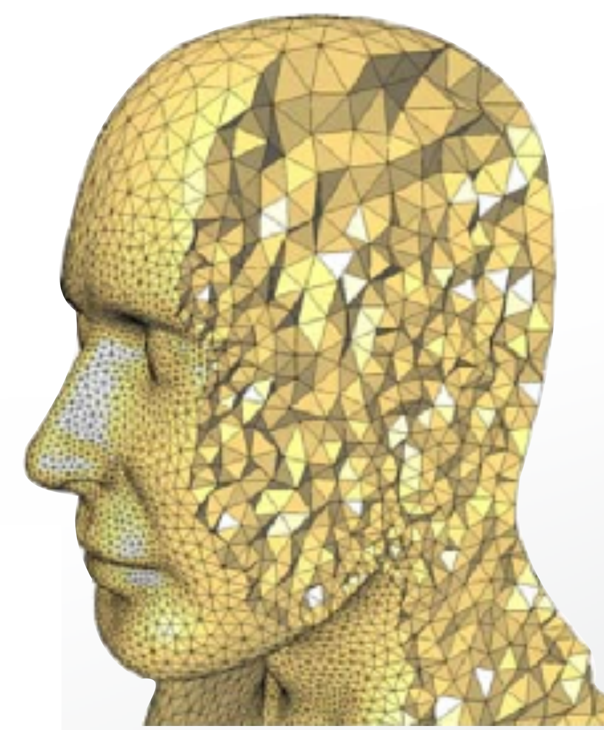
triangle mesh



implicit surfaces / particles



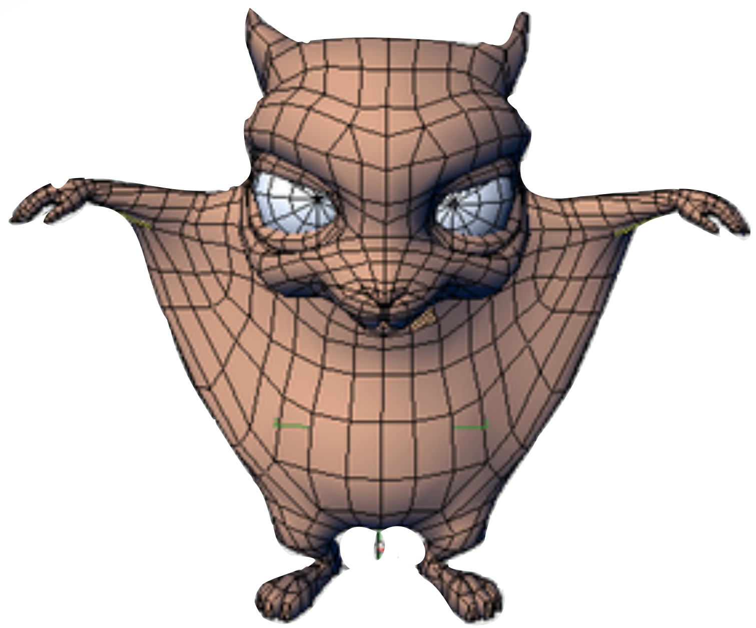
volumetric



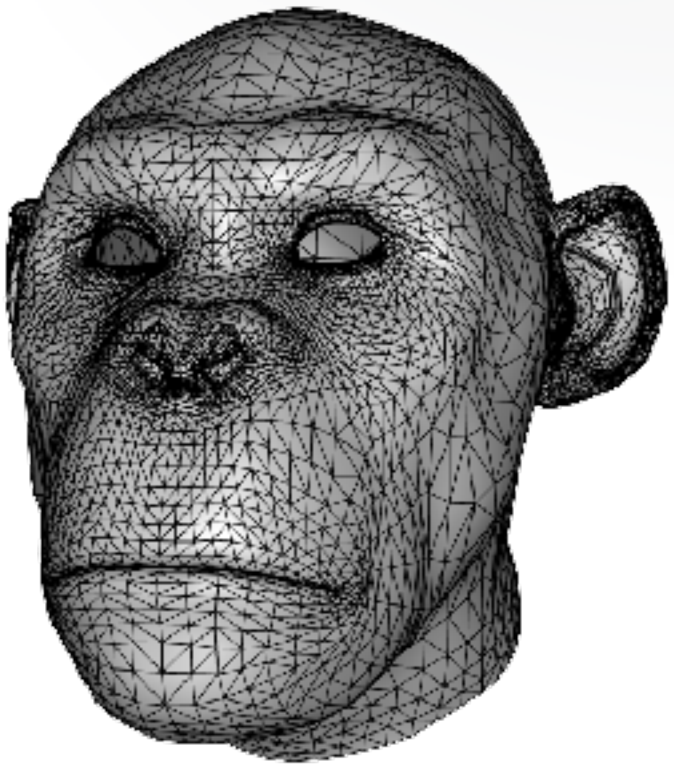
tetrahedrons

Geometric **Representations**

point based



quad mesh



triangle mesh

Surface Representations

implicit surfaces / particles

volumetric

tetrahedrons

High Resolution

Large scenes

Outline

- **Parametric Approximations**
- Polygonal Meshes
- Data Structures

Parametric Representation

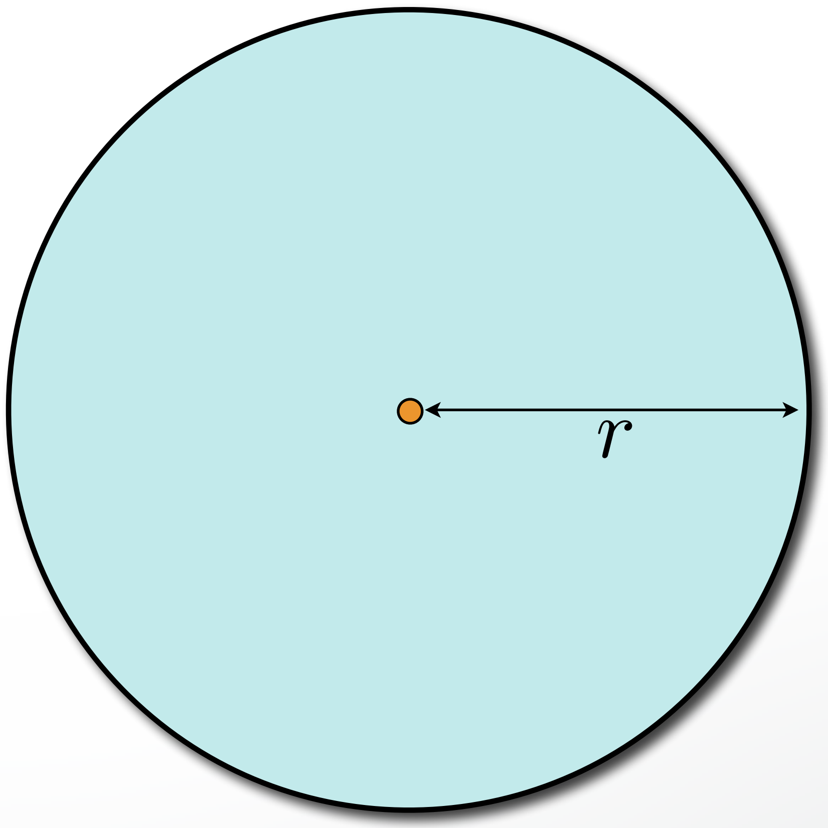
Surface is the range of a function

$$\mathbf{f} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3, \quad \mathcal{S}_\Omega = \mathbf{f}(\Omega)$$

2D example: A Circle

$$\mathbf{f} : [0, 2\pi] \to \mathbb{R}^2$$

$$\mathbf{f}(t) = \begin{pmatrix} r\cos(t) \\ r\sin(t) \end{pmatrix}$$



Parametric Representation

Surface is the range of a function

$$\mathbf{f} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3, \quad \mathcal{S}_\Omega = \mathbf{f}(\Omega)$$

2D example: Island coast line

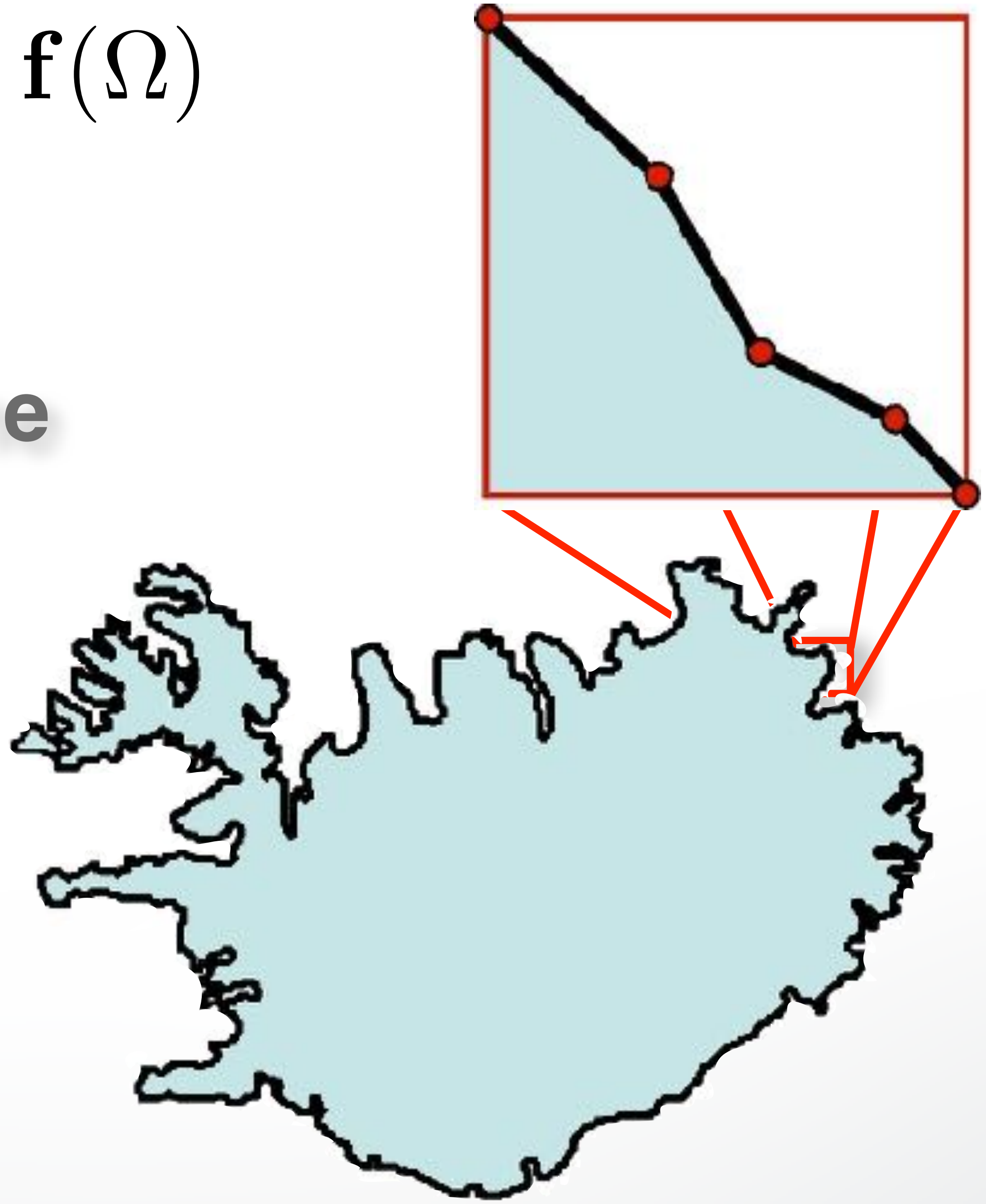
$$\mathbf{f} : [0, 2\pi] \to \mathbb{R}^2$$

$$\mathbf{f}(t) = \begin{pmatrix} \textbf{?} \\ \textbf{?} \end{pmatrix}$$

Piecewise Approximation

Surface is the range of a function

$$\mathbf{f} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3, \quad \mathcal{S}_\Omega = \mathbf{f}(\Omega)$$



2D example: Island coast line

$$\mathbf{f} : [0, 2\pi] \to \mathbb{R}^2$$

$$\mathbf{f}(t) = \begin{pmatrix} ? \\ ? \end{pmatrix}$$

Polynomial Approximation

Polynomials are computable functions

$$f(t) \;=\; \sum_{i=0}^{p} c_i\, t^i \;=\; \sum_{i=0}^{p} \tilde{c}_i\, \phi_i(t)$$

Taylor expansion up to degree p

$$g(h) \;=\; \sum_{i=0}^{p} \frac{1}{i!}\, g^{(i)}(0)\; h^i \;+\; O\big(h^{p+1}\big)$$

Error for approximation g by polynomial f

$$f(t_i) = g(t_i)\,, \quad 0 \le t_0 < \cdots < t_p \le h$$

$$|f(t) - g(t)| \;\le\; \frac{1}{(p+1)!}\, \max f^{(p+1)} \prod_{i=0}^{p} (t - t_i) \;=\; O\Big(h^{(p+1)}\Big)$$

Polynomial Approximation

Approximation error is $O(h^{p+1})$

Improve approximation quality by

- increasing p… higher order polynomials
- decreasing h … shorter / more segments

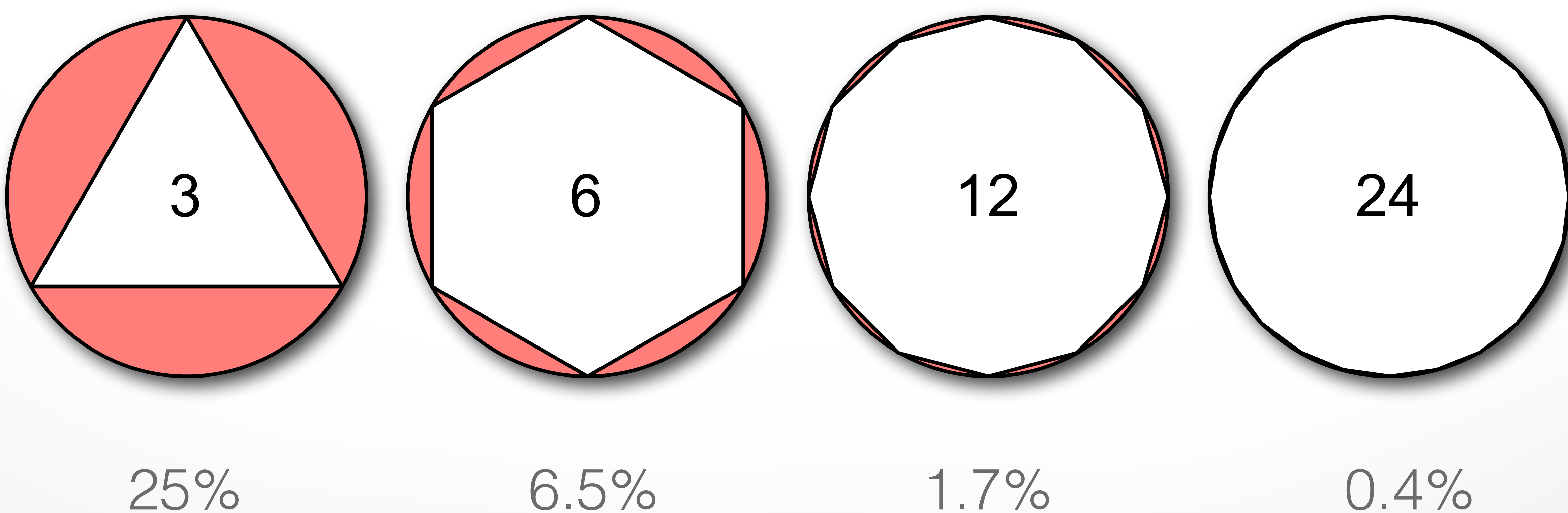
Issues

- smoothness of the target data ($\max_t f^{(p+1)}(t)$)
- smoothness condition between segments

Polygonal Meshes

Polygonal meshes are a good compromise

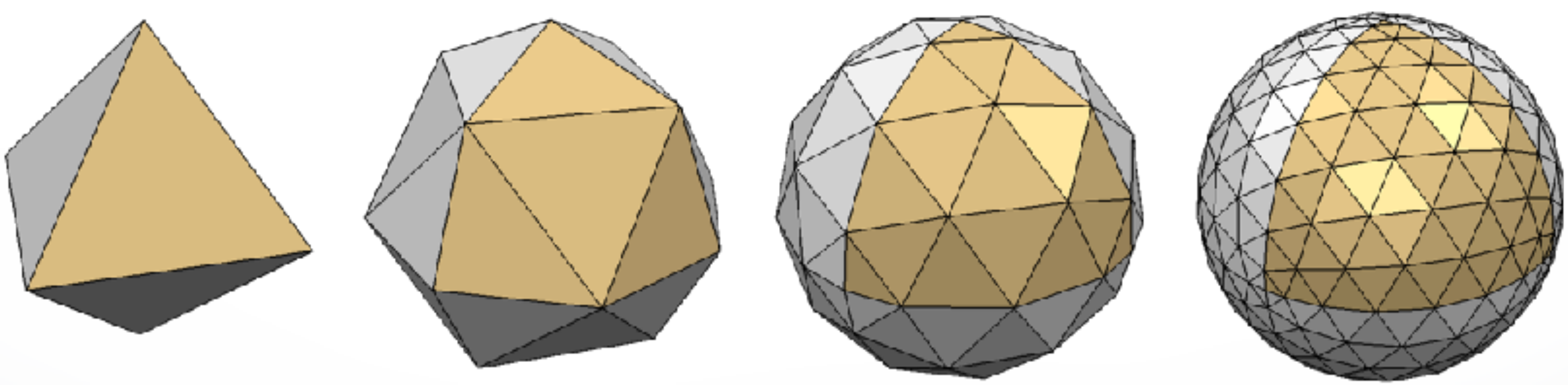
- Piecewise linear approximation → error is $O(h^2)$



Polygonal Meshes

Polygonal meshes are a good compromise

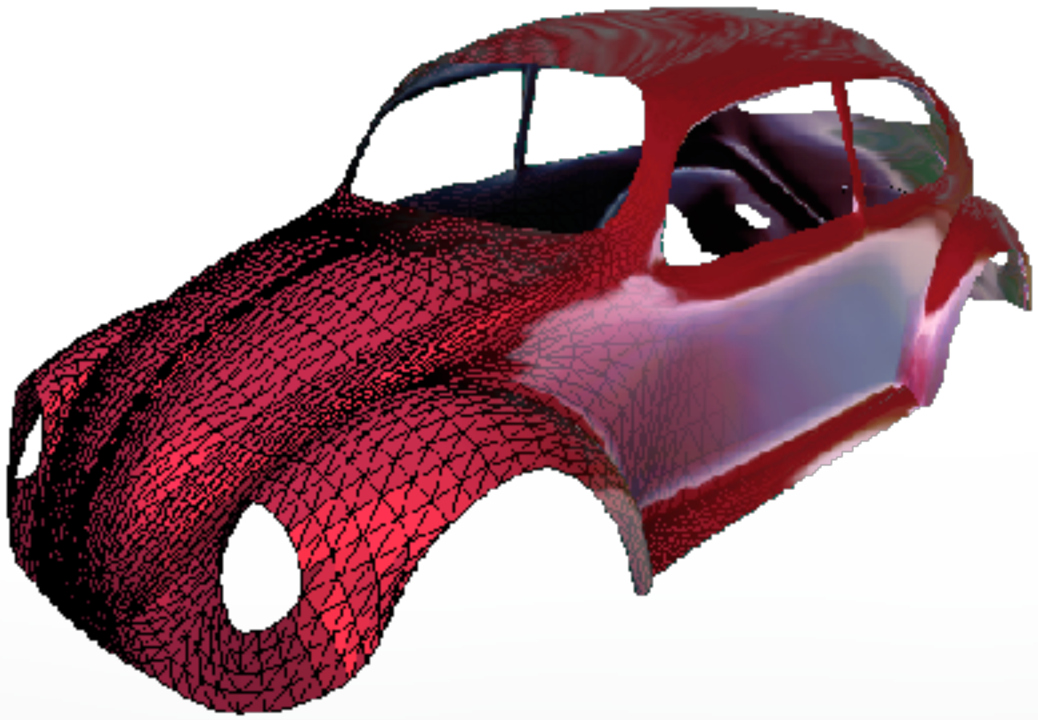
- Piecewise linear approximation → error is $O(h^2)$
- Error inversely proportional to #faces



Polygonal Meshes

Polygonal meshes are a good compromise

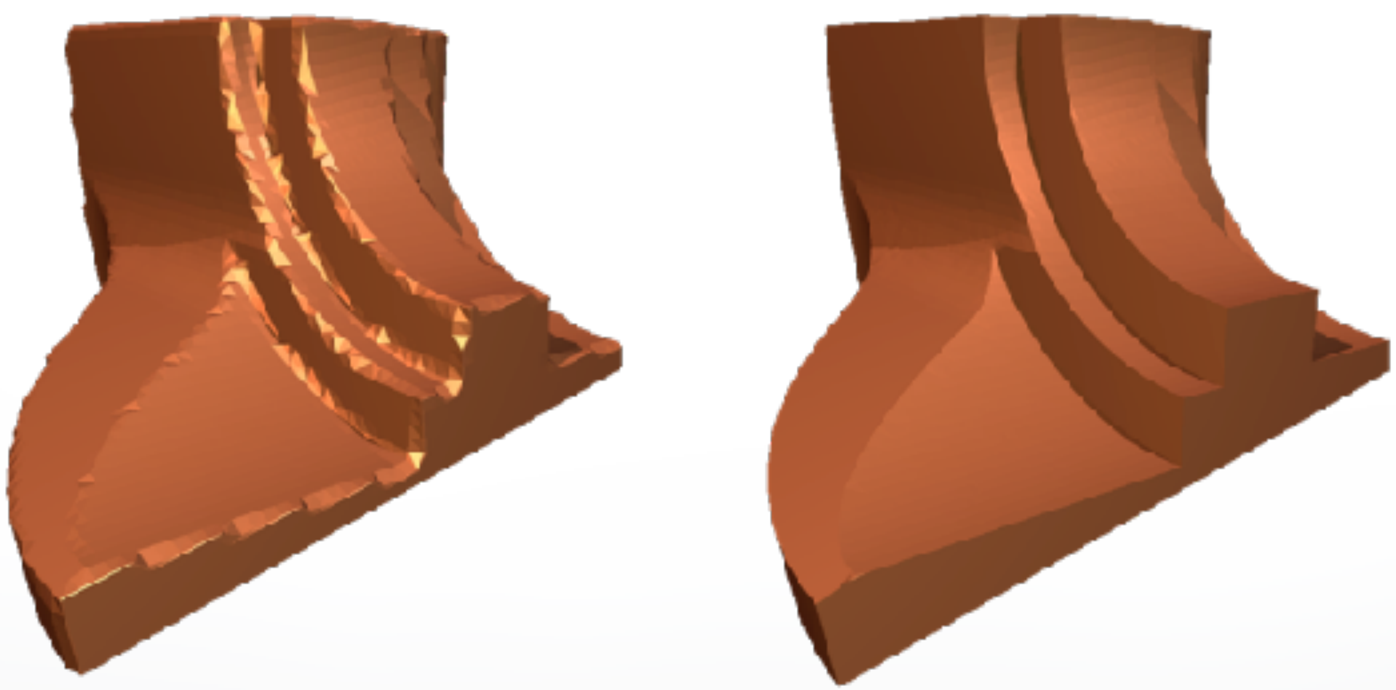
- Piecewise linear approximation → error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces



Polygonal Meshes

Polygonal meshes are a good compromise

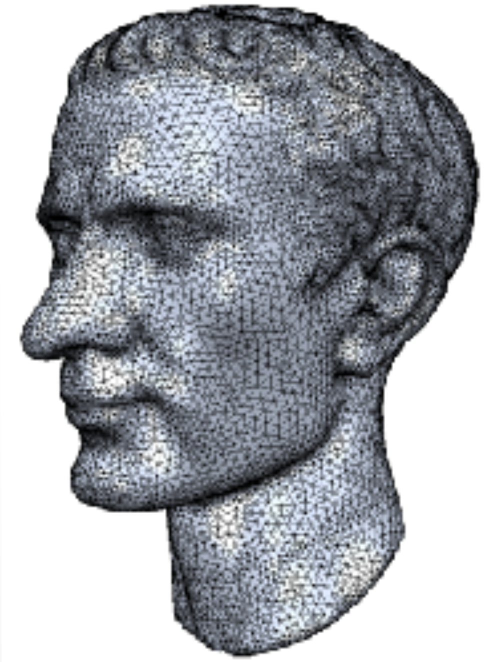
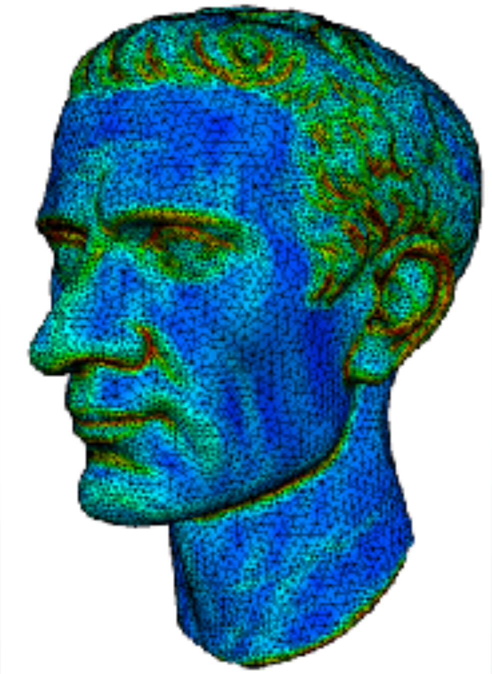
- Piecewise linear approximation → error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces
- Piecewise smooth surfaces



Polygonal Meshes

Polygonal meshes are a good compromise

- Piecewise linear approximation → error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces
- Piecewise smooth surfaces
- Adaptive sampling

Polygonal Meshes

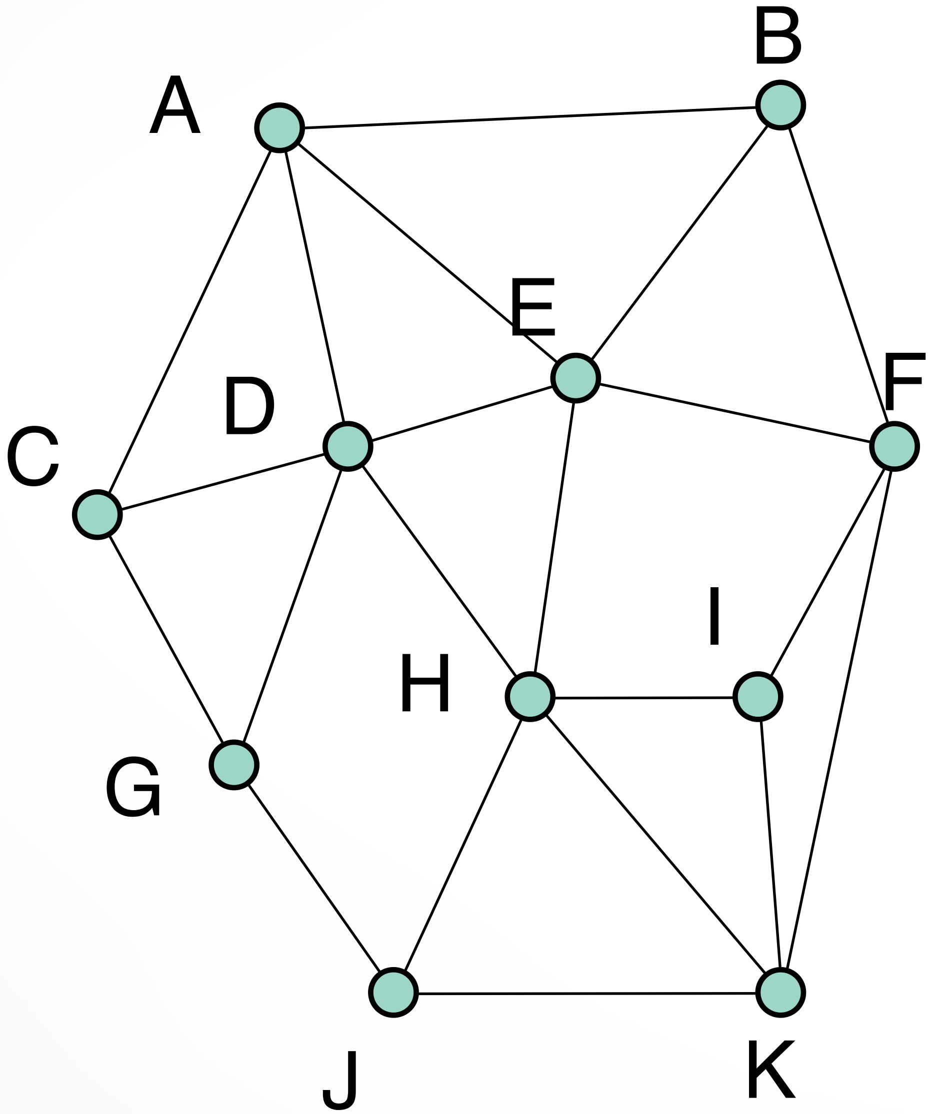
Polygonal meshes are a good compromise

- Piecewise linear approximation → error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces
- Piecewise smooth surfaces
- Adaptive sampling
- Efficient GPU-based rendering/processing

Outline

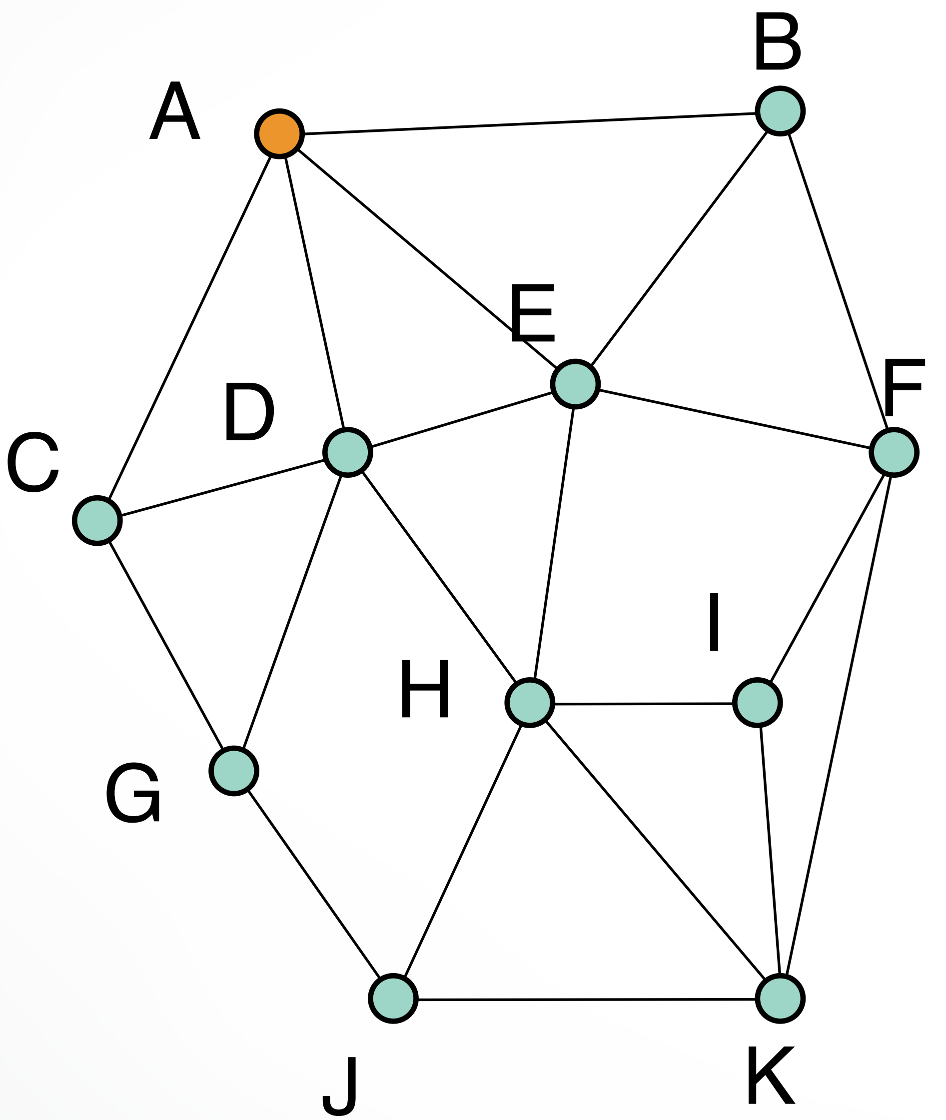
- Parametric Approximations
- **Polygonal Meshes**
- Data Structures

Graph Definitions



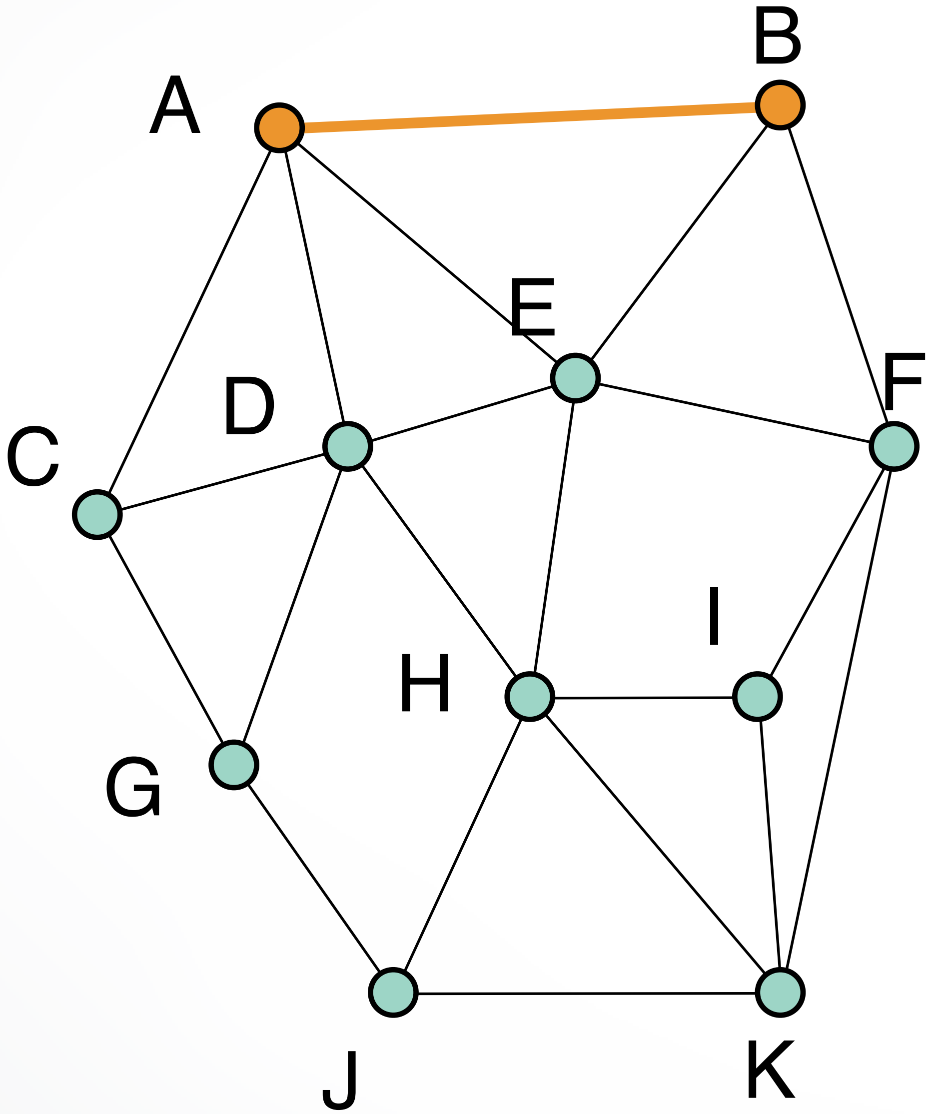
- Graph $\{V,E\}$

Graph Definitions



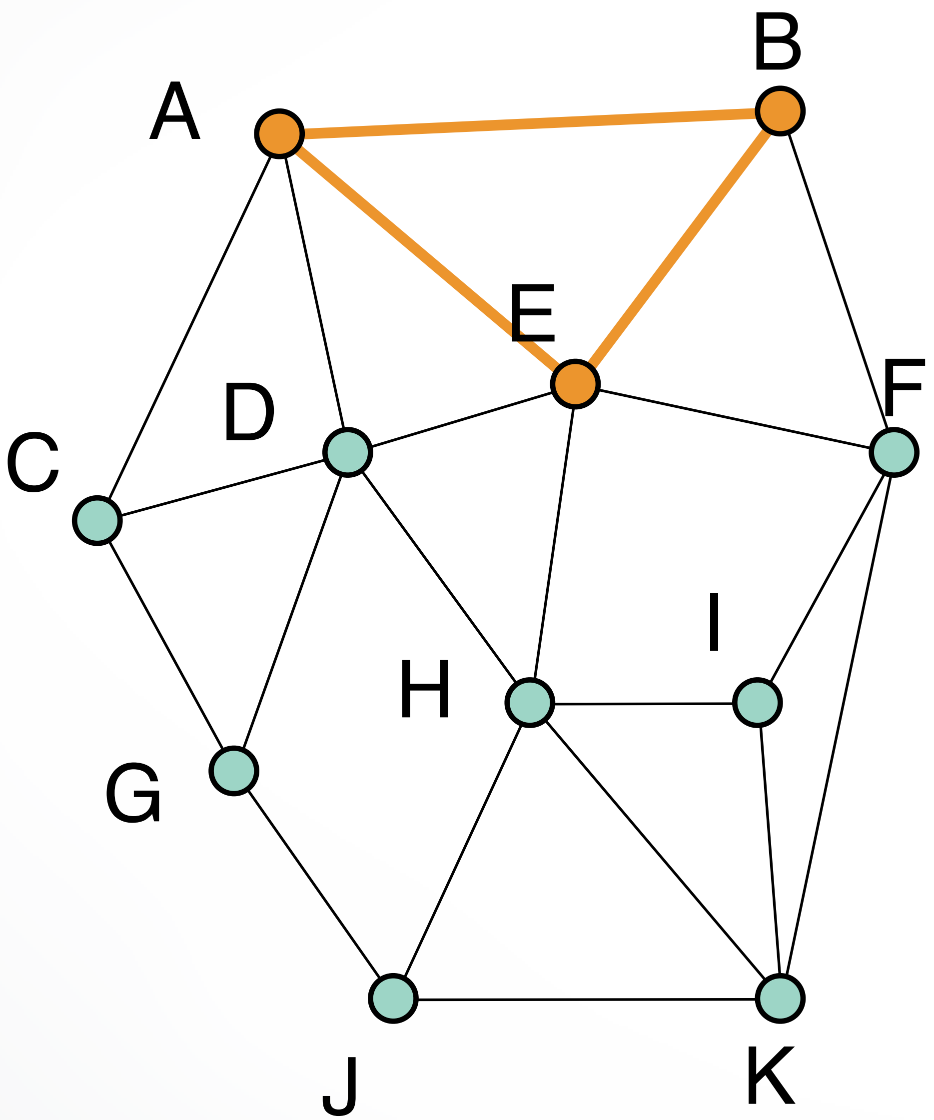
- Graph $\{V,E\}$
- Vertices $V = \{A,B,C,\ldots,K\}$

Graph Definitions



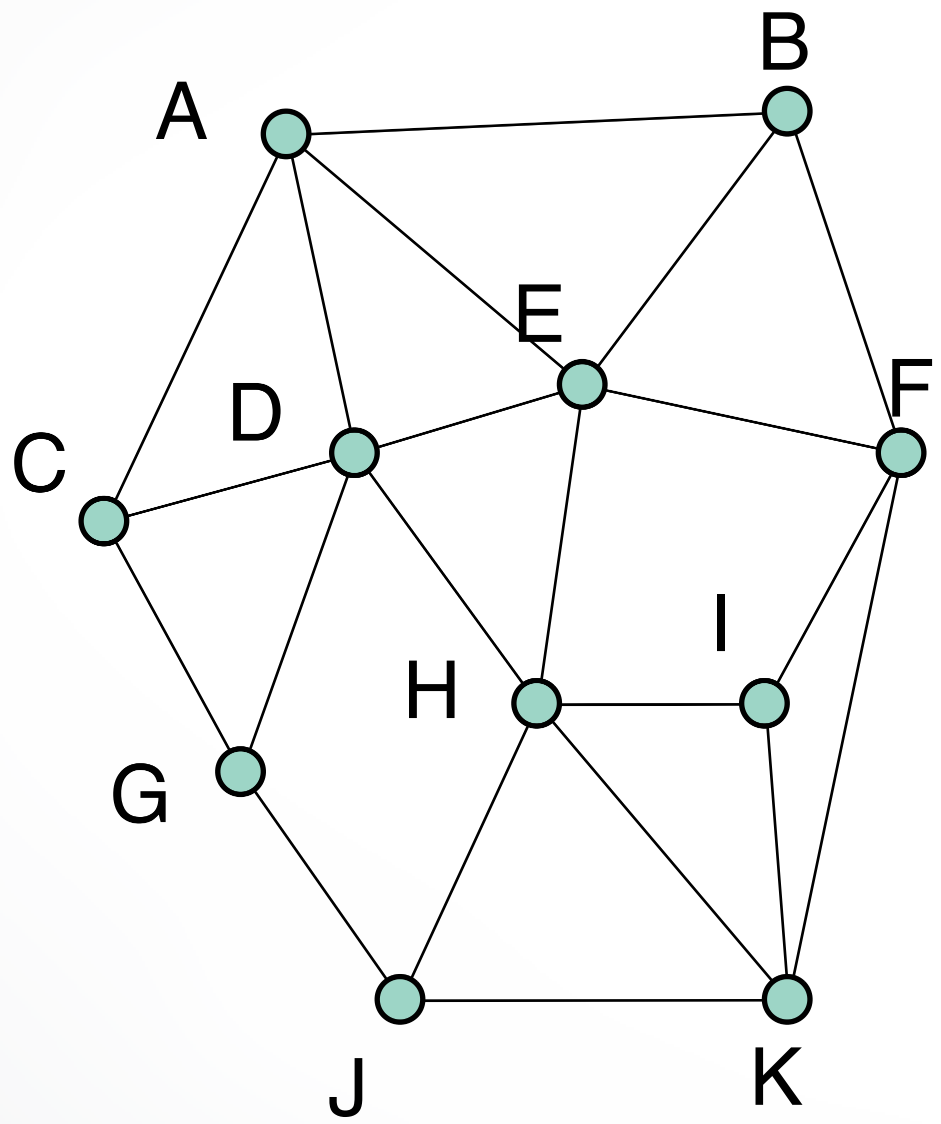
- Graph $\{V,E\}$
- Vertices $V = \{A,B,C,\ldots,K\}$
- Edges $E = \{(AB),(AE),(CD),\ldots\}$

Graph Definitions



- Graph $\{V,E\}$
- Vertices $V = \{A,B,C,\ldots,K\}$
- Edges $E = \{(AB),(AE),(CD),\ldots\}$
- Faces $F = \{(ABE),(EBF),(EFIH),\ldots\}$

Graph Definitions

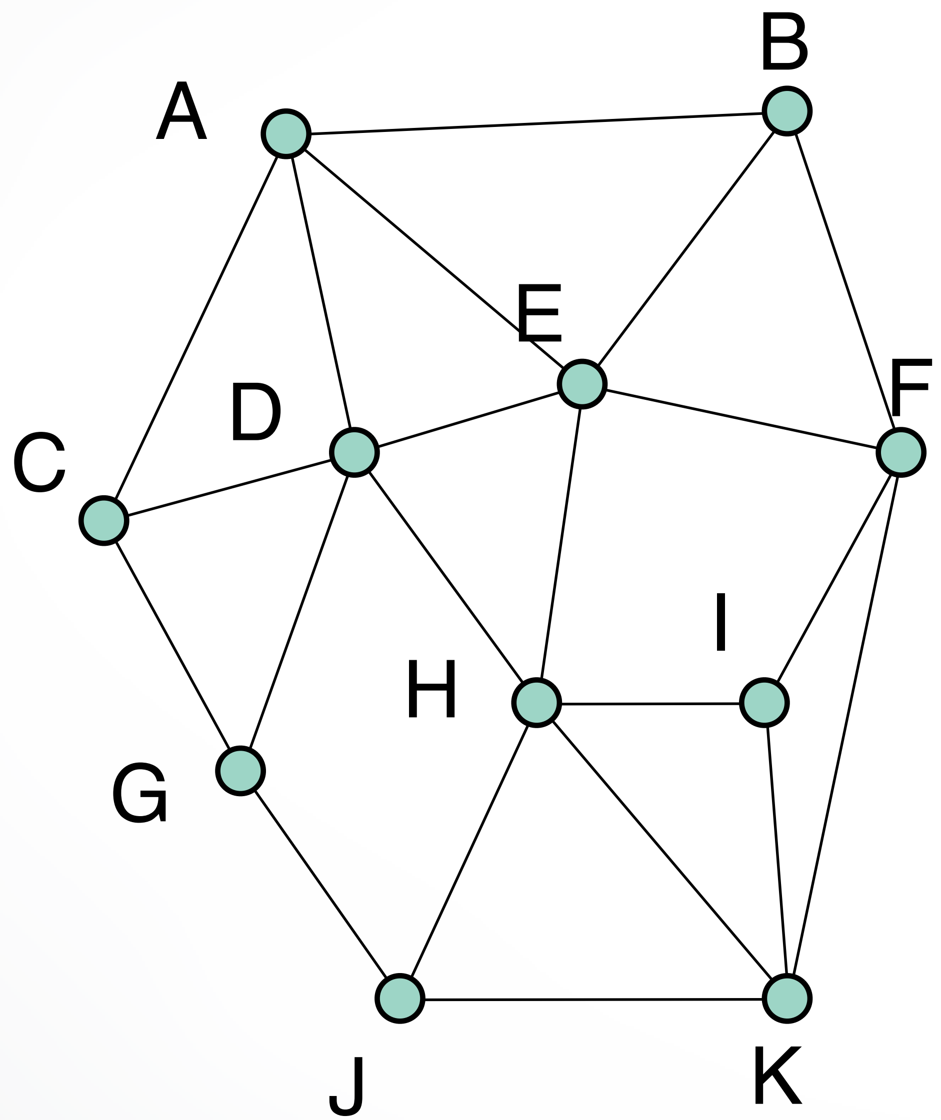


Vertex degree or valence:

number of incident edges

- deg(A) = 4
- deg(E) = 5

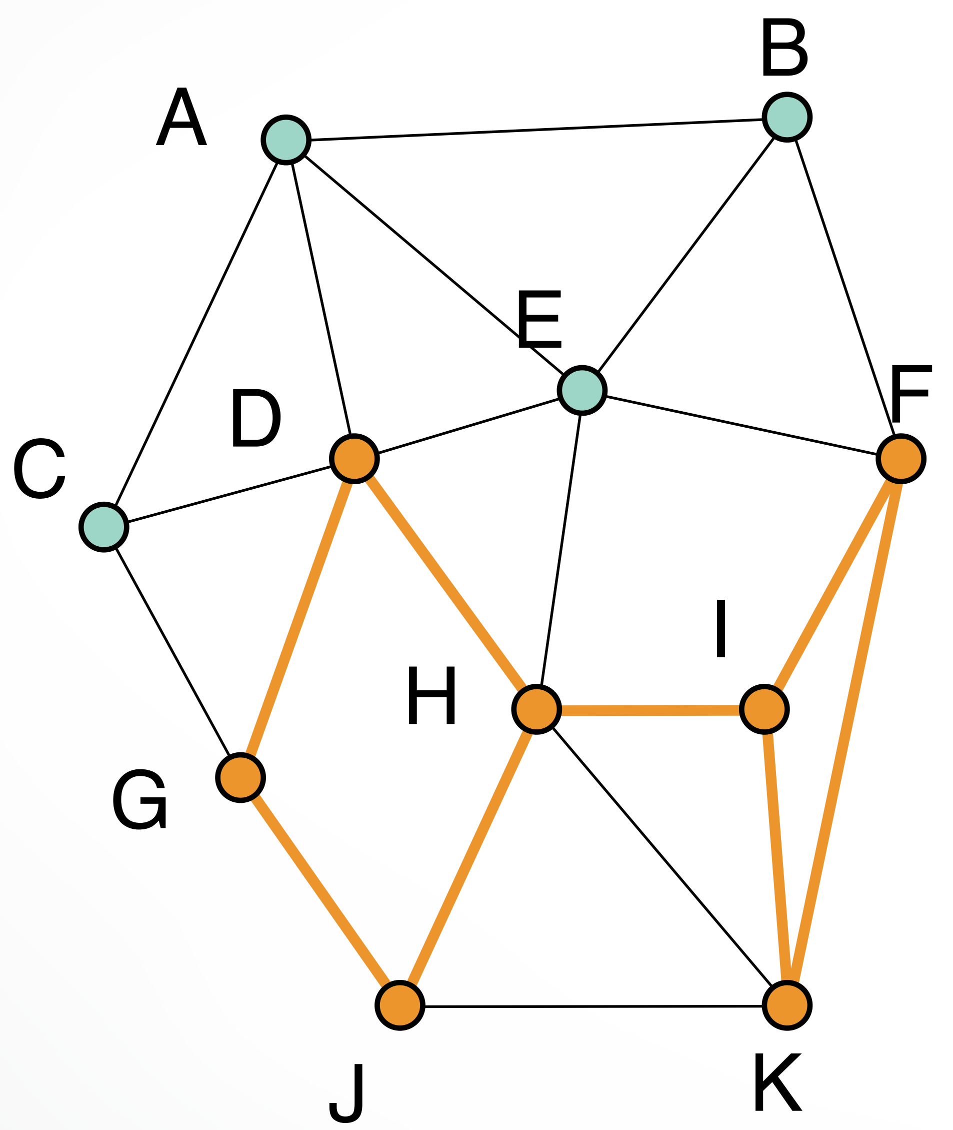
Connectivity



Connected:

Path of edges connecting every two vertices

Connectivity



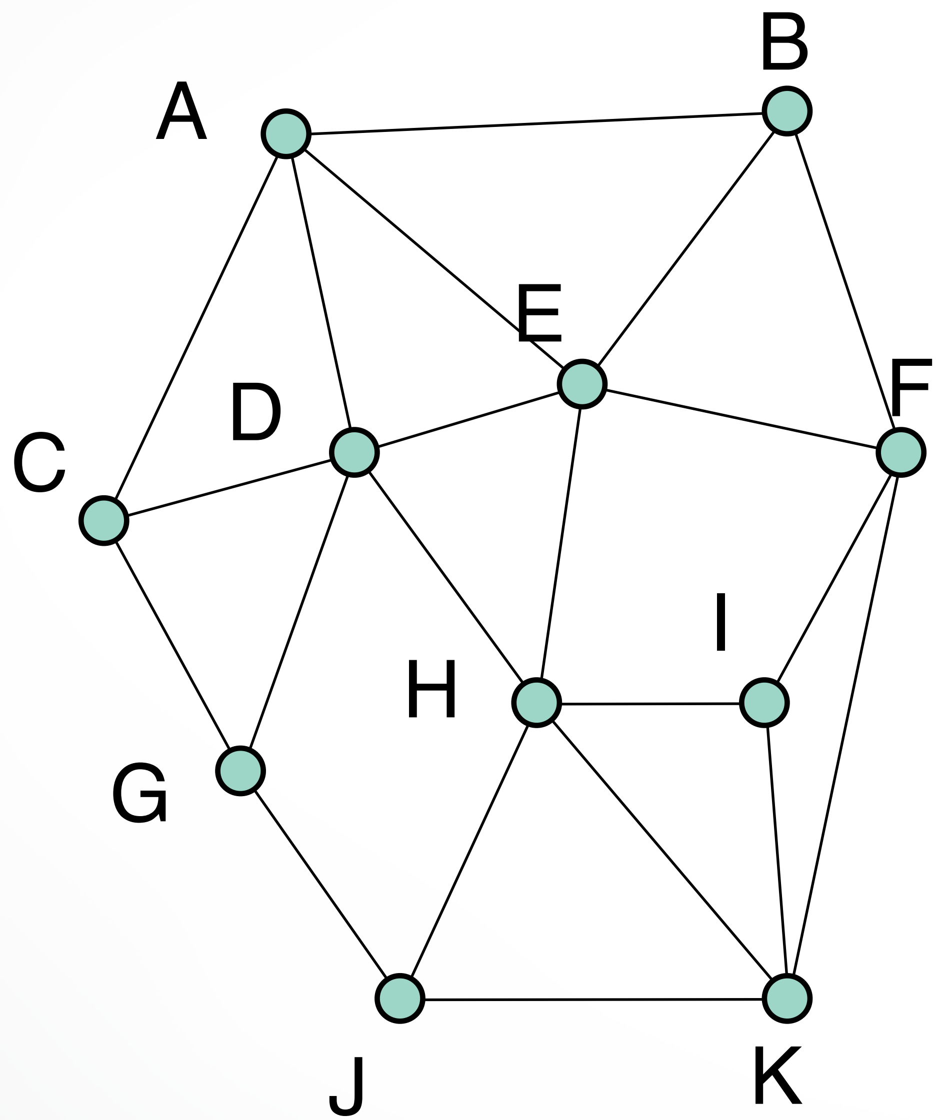
Connected:

Path of edges connecting every two vertices

Subgraph:

Graph $\{V',E'\}$ is a subgraph of graph $\{V,E\}$ if V' is a subset of V and E' is a subset of E incident on V'.

Connectivity



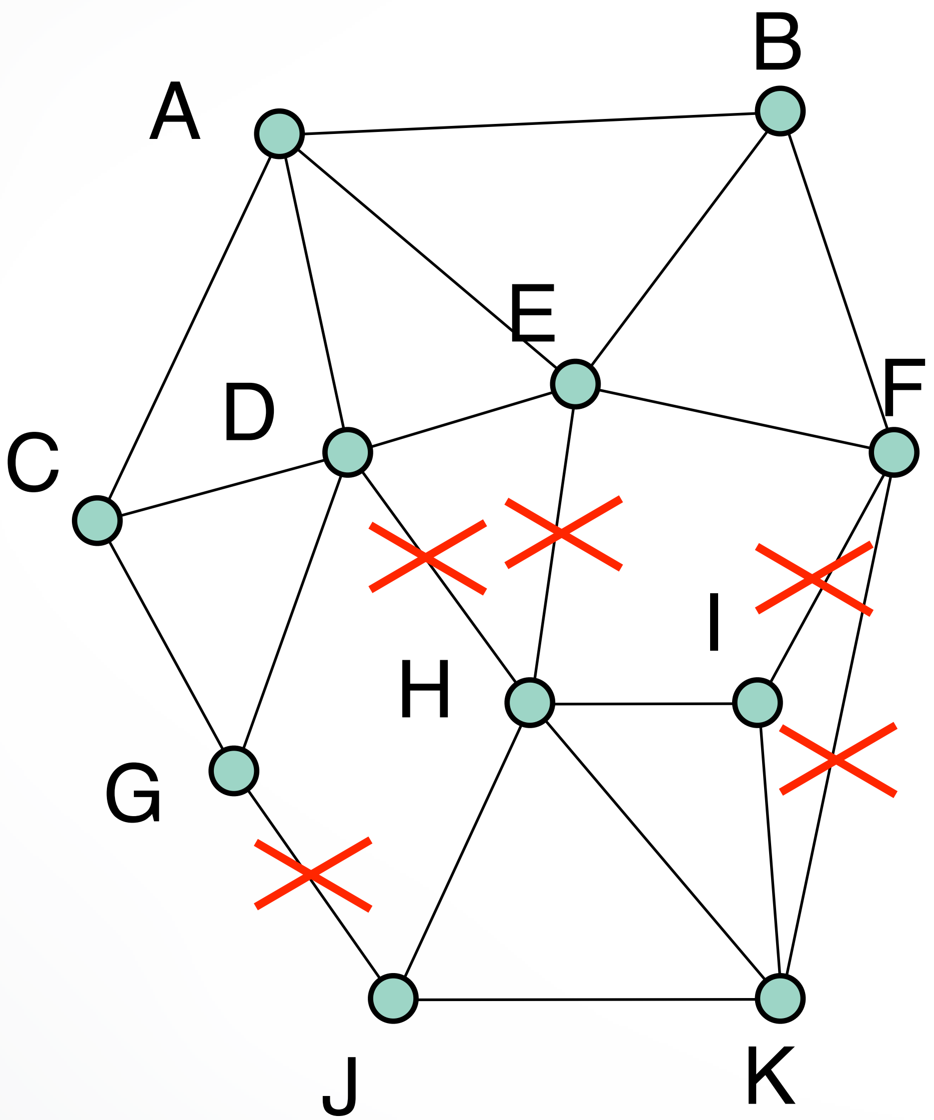
Connected:

Path of edges connecting every two vertices

Subgraph:

Graph {*V'*,*E'*} is a subgraph of graph {*V*,*E*} if *V'* is a subset of *V* and *E'* is a subset of *E* incident on *V'*.

Connectivity



Connected:

Path of edges connecting

every two vertices

Subgraph:

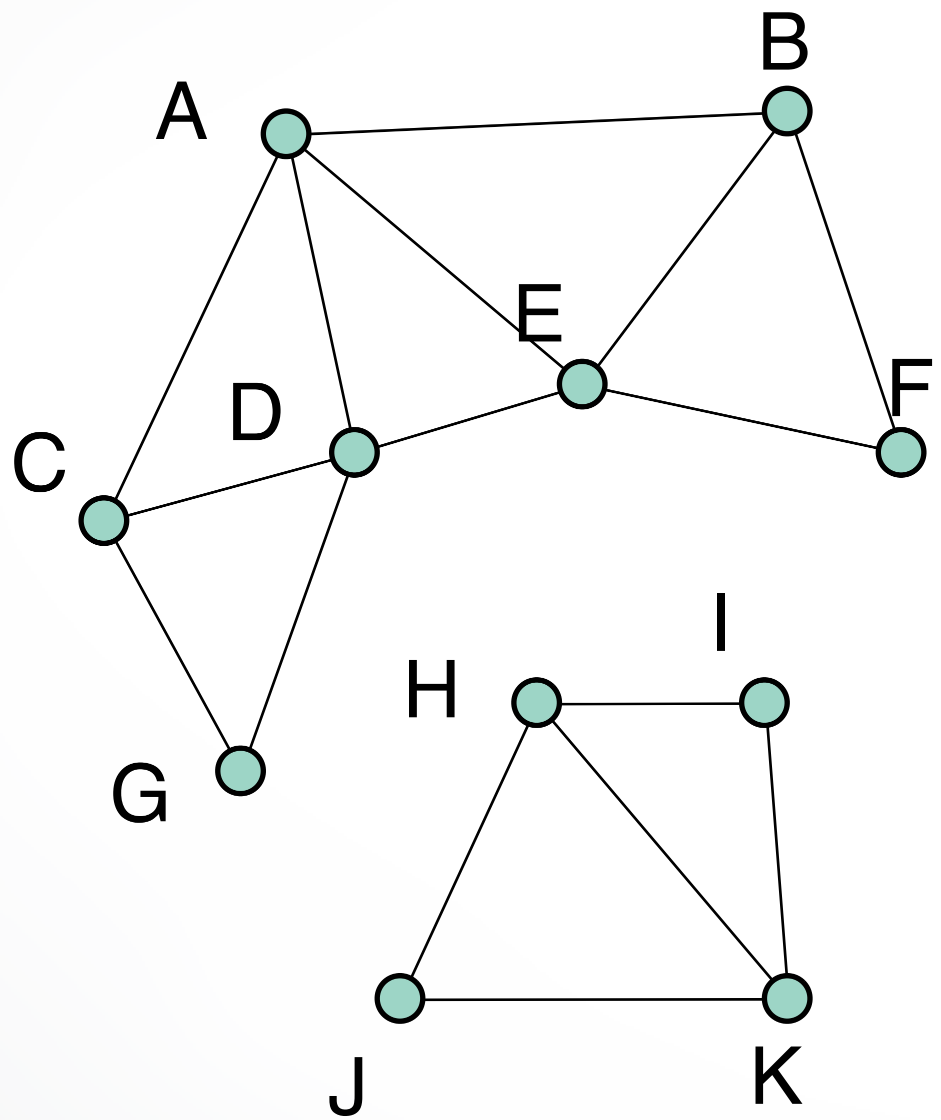
Graph {*V'*,*E'*} is a subgraph of graph

{*V*,*E*} if *V'* is a subset of *V* and *E'* is a subset of *E* incident on *V'*.

Connected Components:

Maximally connected subgraph

Connectivity



Connected:

Path of edges connecting

every two vertices

Subgraph:

Graph {*V'*,*E'*} is a subgraph of graph

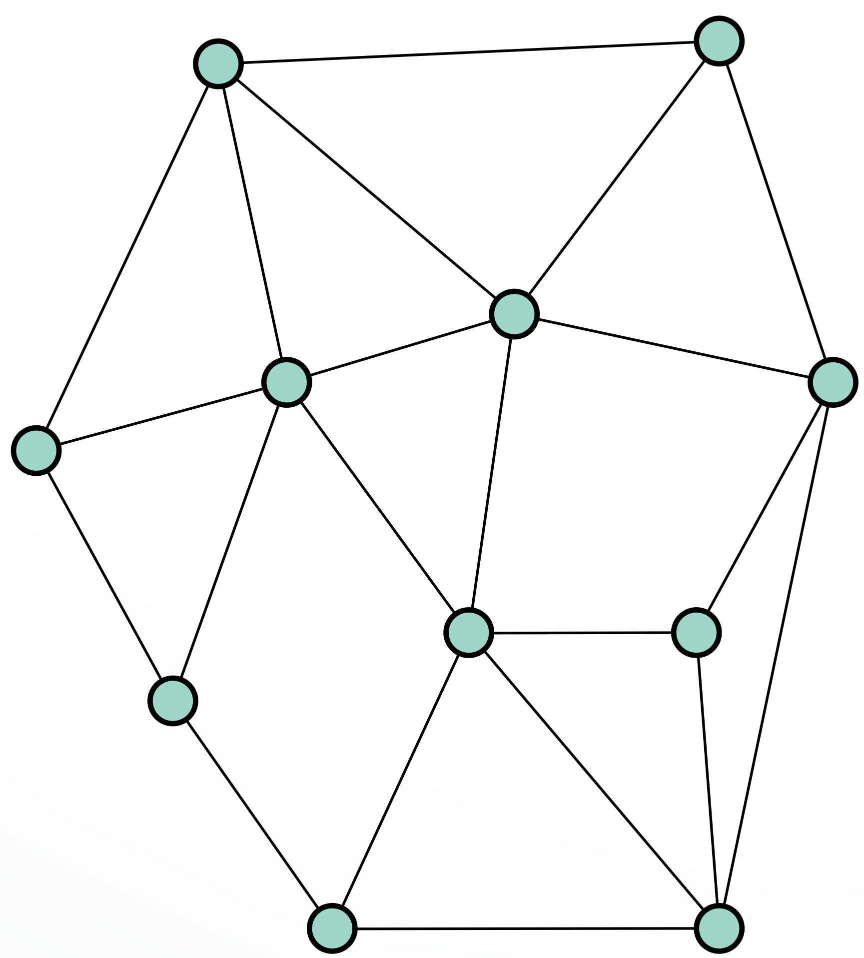
{*V*,*E*} if *V'* is a subset of *V* and *E'* is a subset of *E* incident on *V'*.

Connected Components:

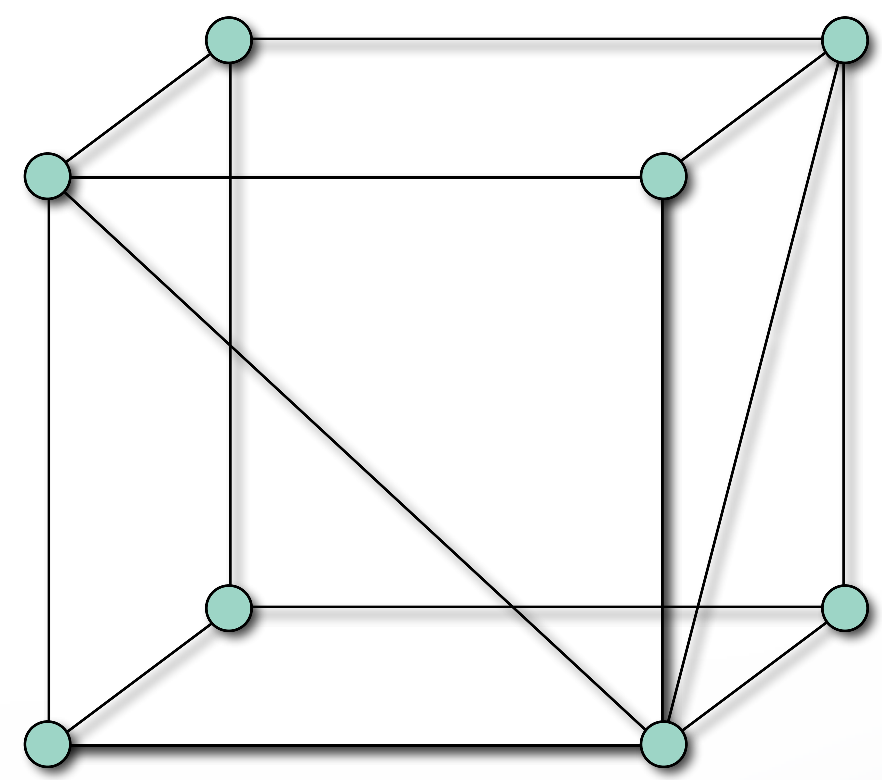
Maximally connected subgraph

Graph Embedding

Embedding: Graph is **embedded** in $\mathbb{R}^d$, if each vertex is assigned a position in $\mathbb{R}^d$.



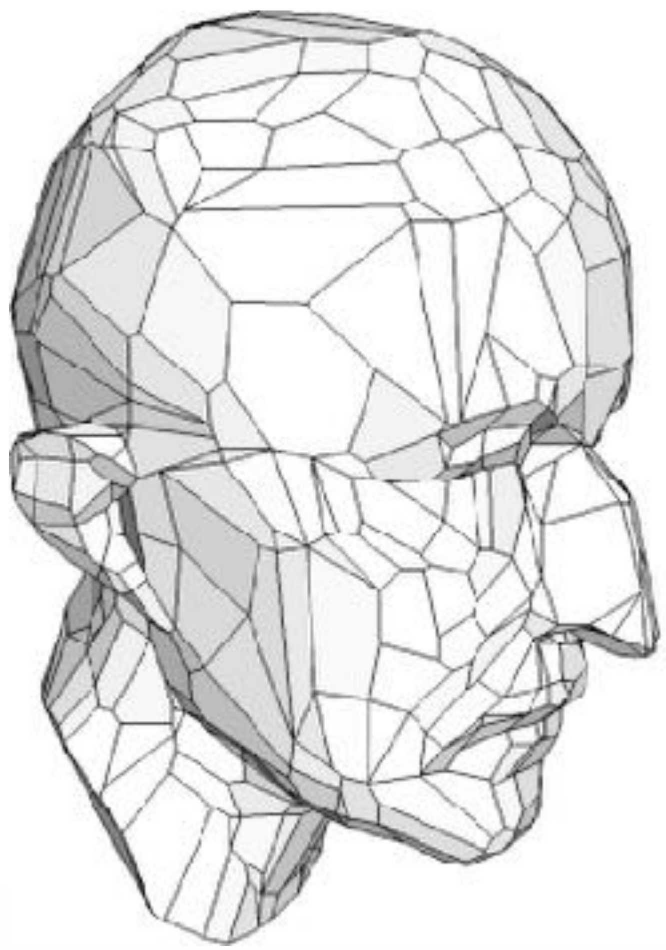
Embedding in $\mathbb{R}^2$



Embedding in $\mathbb{R}^3$

Graph Embedding

Embedding: Graph is **embedded** in $\mathbb{R}^d$, if each vertex is assigned a position in $\mathbb{R}^d$.

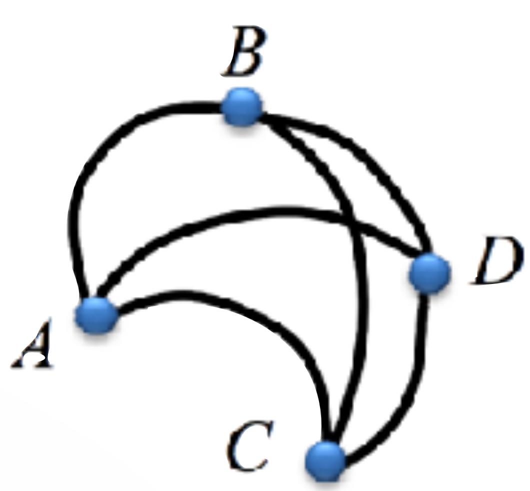


Embedding in $\mathbb{R}^3$

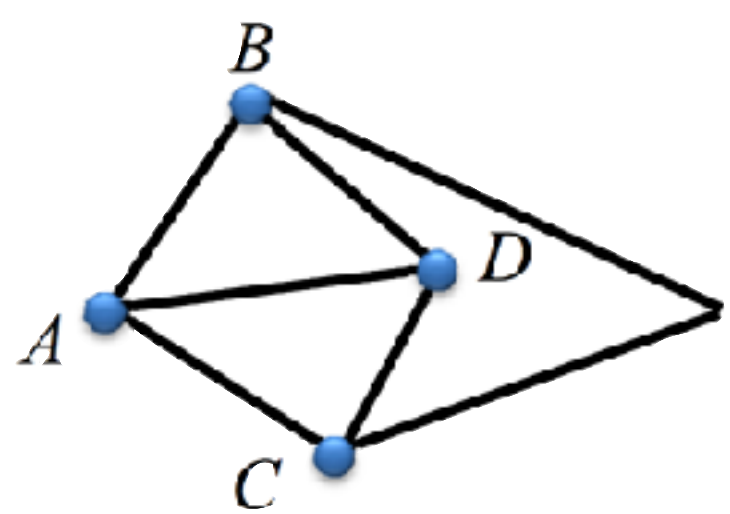
Planar Graph

Planar Graph

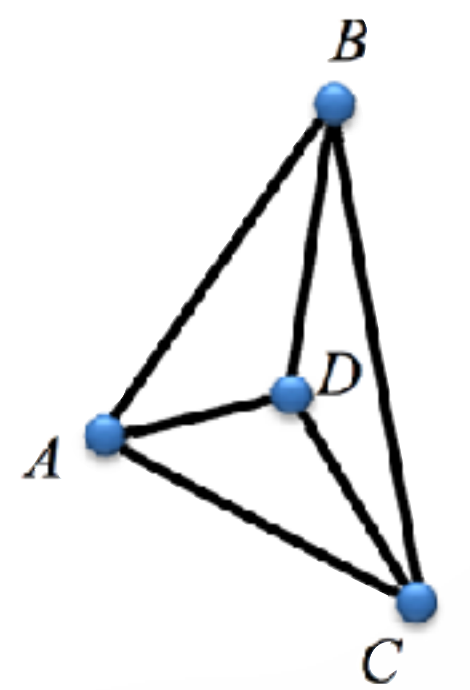
Graph whose vertices and edges **can be** embedded in $\mathbb{R}^2$ such that its edges do not intersect



Planar Graph

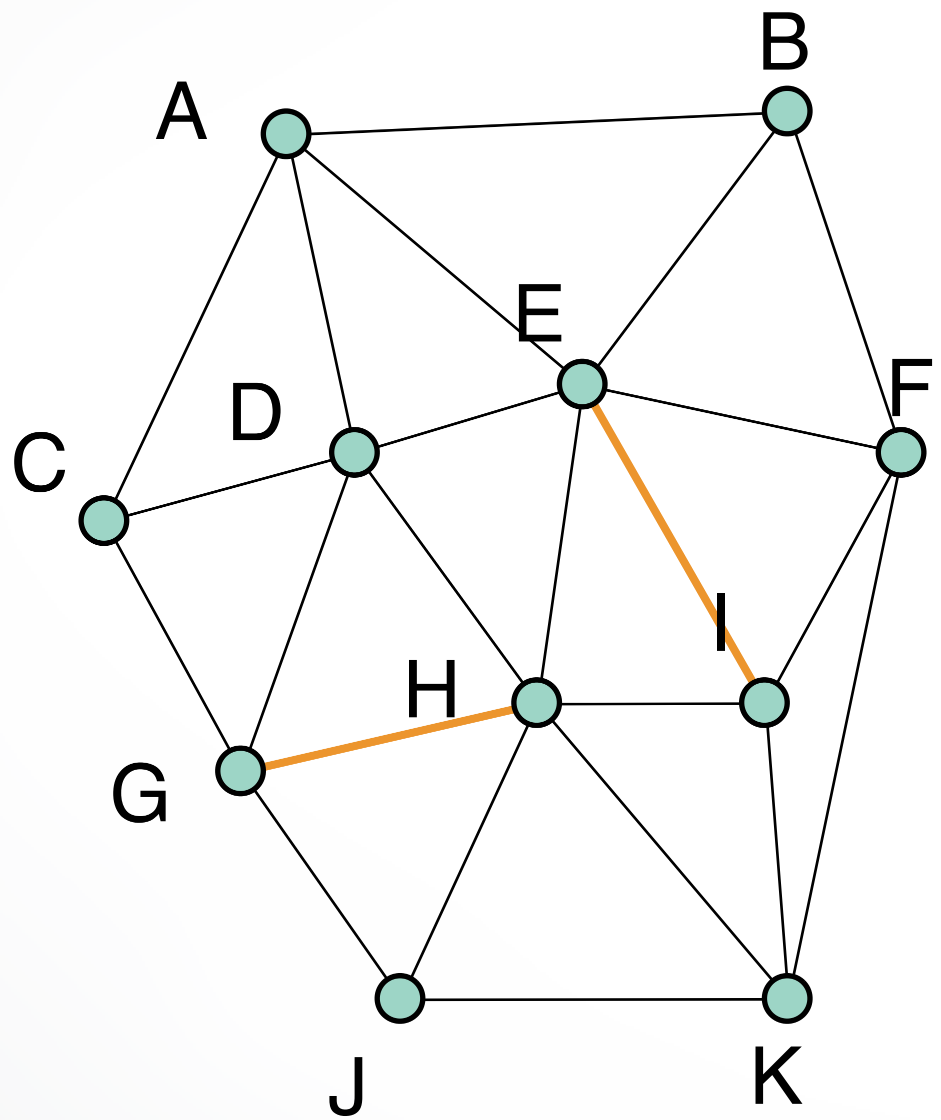


Plane Graph



Straight Line
Plane Graph

Triangulation



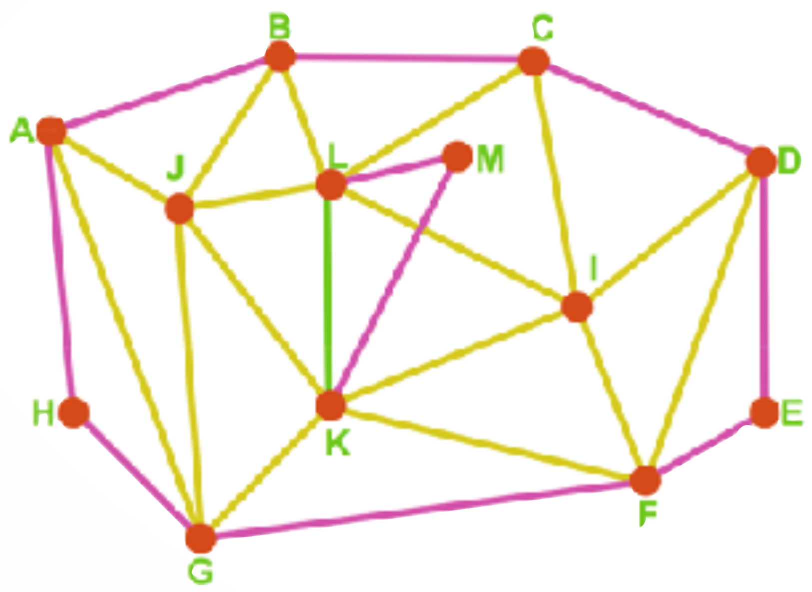
Triangulation:

Straight line plane graph where every face is a triangle

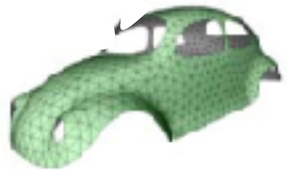
Why?

- simple homogenous data structure
- efficient rendering
- simplifies algorithms
- by definition, triangle is planar
- any polygon can be triangulated

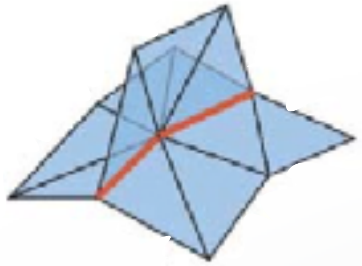
Mesh



- **Mesh**: straight-line graph embedded in $\mathbb{R}^3$
- **Boundary edge:** adjacent to exactly 1 face



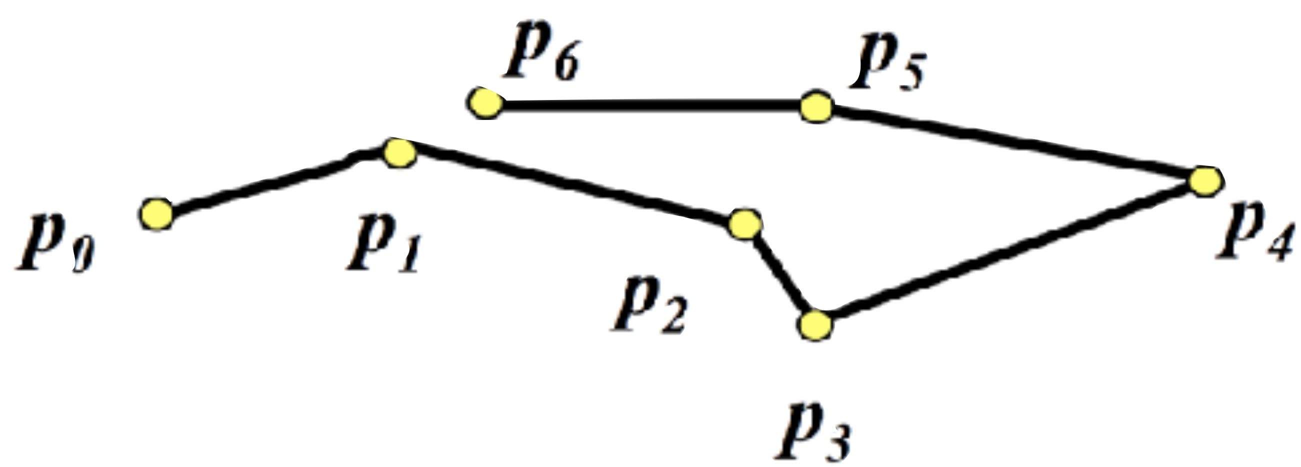
- **Regular edge:** adjacent to exactly 2 faces
- **Singular edge:** adjacent to more than 2 faces



- **Closed mesh:** mesh with no boundary edges

Polygon

A geometric graph $Q = (V, E)$
with $V = \{\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}\}$ in $\mathbb{R}^d$, $d \geq 2$
and $E = \{(\mathbf{p}_0, \mathbf{p}_1) \dots (\mathbf{p}_{n-2}, \mathbf{p}_{n-1})\}$
is called a **polygon**



A polygon is called

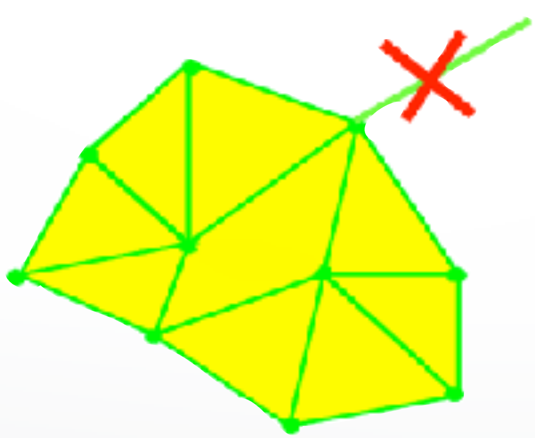
- flat, if all edges are on a plane
- closed, if $\mathbf{p}_0 = \mathbf{p}_{n-1}$

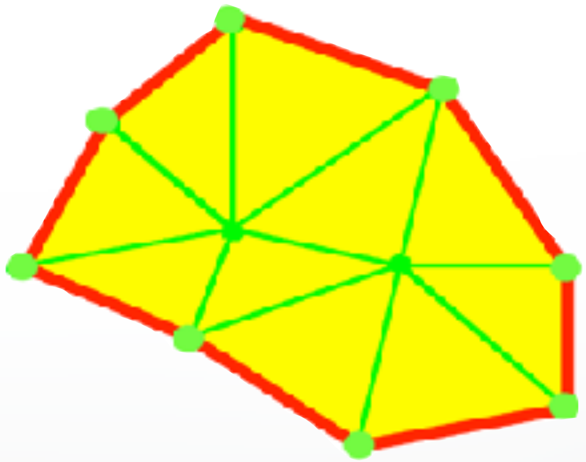
While digital artists call it **Wireframe, ...**

Polygonal Mesh

A set M of finite number of closed polygons Q_i if:

- Intersection of inner polygonal areas is empty
- Intersection of 2 polygons from M is either empty, a point $p \in P$ or an edge $e \in E$
- Every edge $e \in E$ belongs to at least one polygon
- The set of all edges which belong only to one polygon are called edges of the polygonal mesh and are either empty or form a single closed polygon

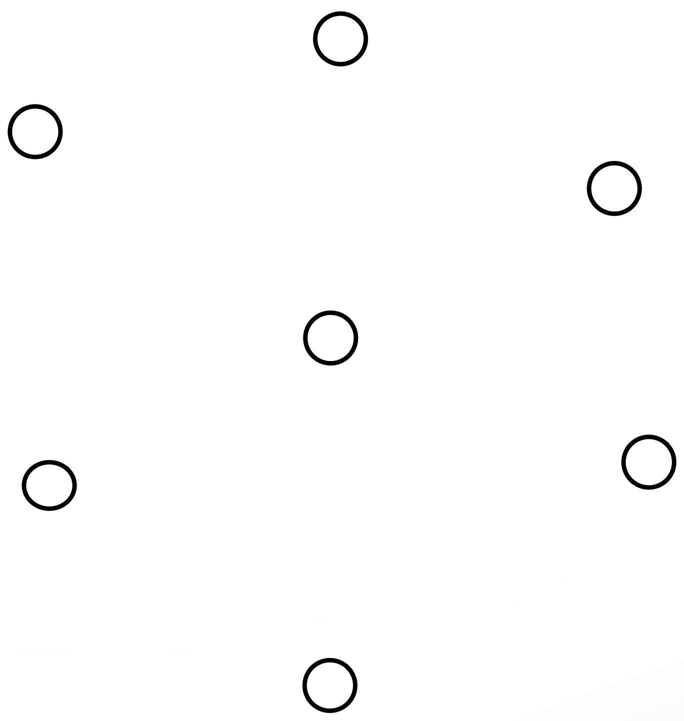




Polygonal Mesh Notation

$$\mathcal{M} = (\{\mathbf{v}_i\}, \{e_j\}, \{f_k\})$$

geometry $\mathbf{v}_i \in \mathbb{R}^3$

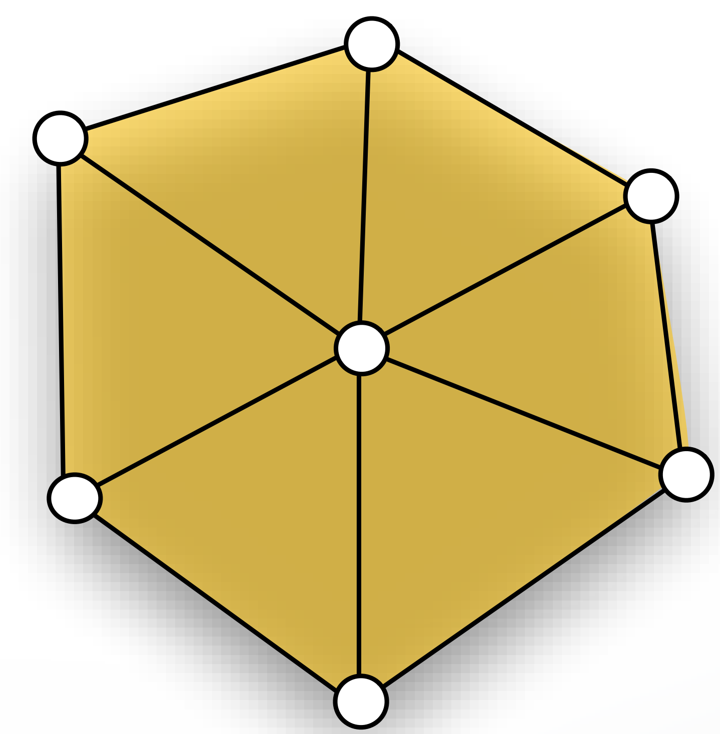


Polygonal Mesh Notation

$$\mathcal{M} = (\{\mathbf{v}_i\}, \{e_j\}, \{f_k\})$$

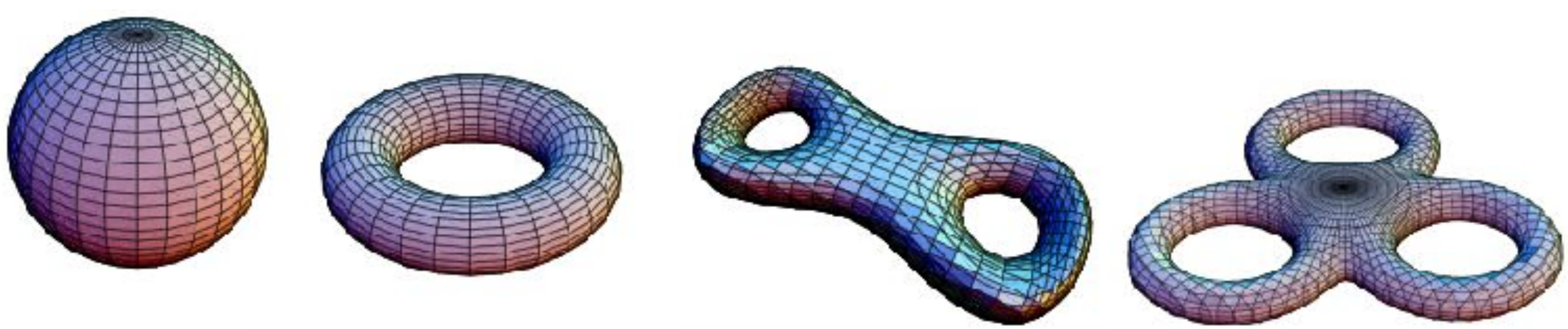
geometry $\mathbf{v}_i \in \mathbb{R}^3$

topology $e_i, f_i \subset \mathbb{R}^3$



Global Topology: **Genus**

- **Genus:** Maximal number of closed simple cutting curves that do not disconnect the graph into multiple components.
- Or half the maximal number of closed paths that do no disconnect the mesh
- Informally, the number of **holes** or **handles**



Genus 0 Genus 1 Genus 2 Genus 3

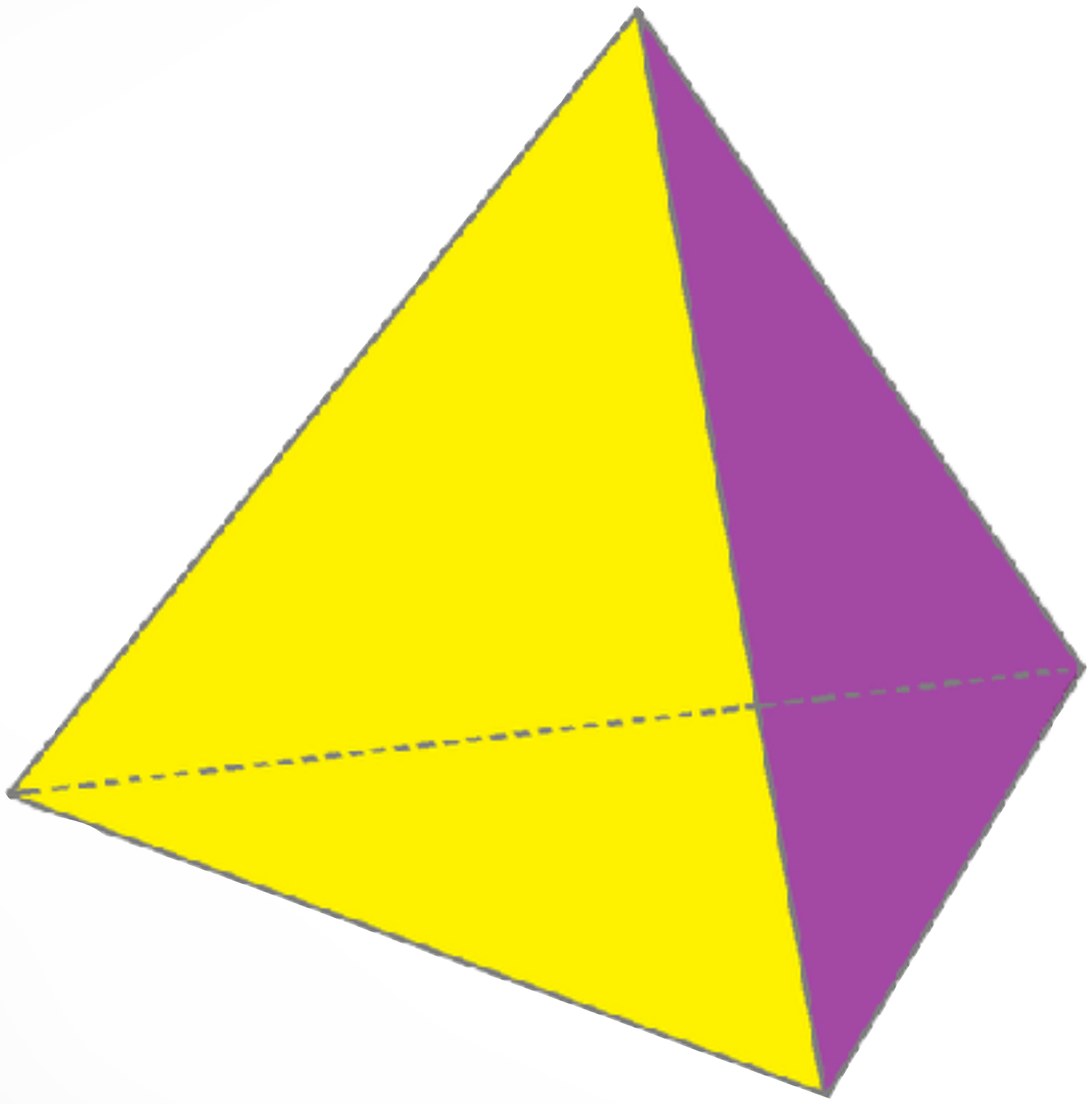
Euler Poincaré Formula

- For a closed polygonal mesh of **genus** g , the relation of the number V of vertices, E of edges, and F of faces is given by **Euler's formula:**

$$V - E + F = 2(1 - g)$$

- The term $2(1-g)$ is called the **Euler characteristic** χ

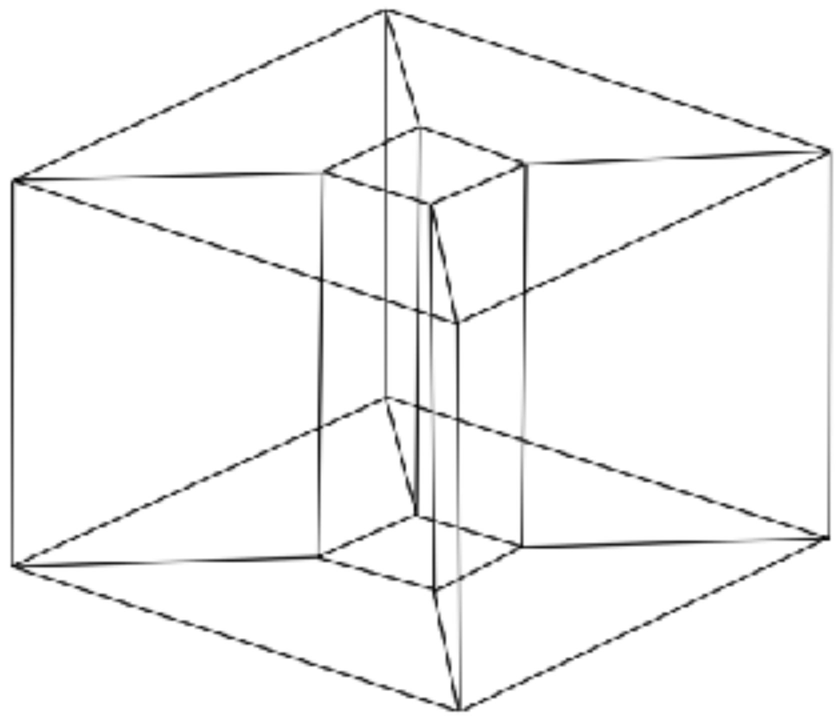
Euler Poincaré Formula



$$V - E + F = 2(1 - g)$$

$$4 - 6 + 4 = 2(1 - 0)$$

Euler Poincaré Formula



$$V - E + F = 2(1 - g)$$

$$16 - 32 + 16 = 2(1 - 1)$$

Average Valence of Closed Triangle Mesh

Theorem: Average vertex degree in a closed manifold triangle mesh is ~6

Proof: Each face has 3 edges and each edge is counted twice: $3F = 2E$

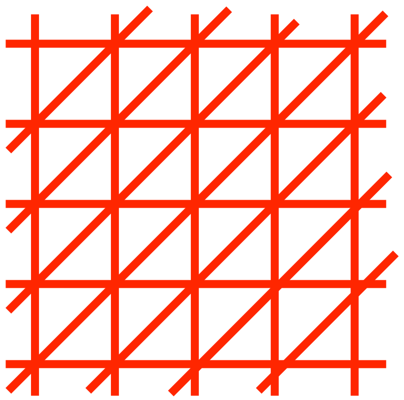
by Euler's formula: $V+F-E = V+2E/3-E = 2-2g$
Thus $E = 3(V-2+2g)$

So average degree $= 2E/V = 6(V-2+2g)/V \sim 6$ for large V

Euler Consequences

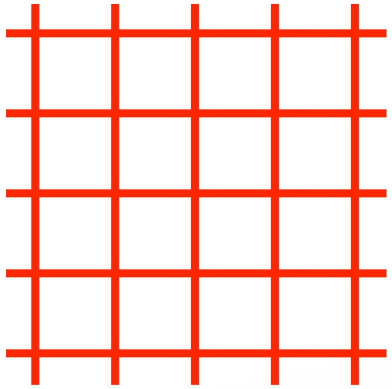
Triangle mesh statistics

- $F \approx 2V$
- $E \approx 3V$
- Average valence = 6



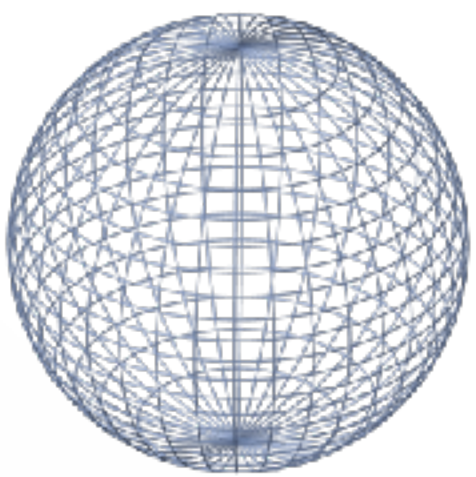
Quad mesh statistics

- $F \approx V$
- $E \approx 2V$
- Average valence = 4



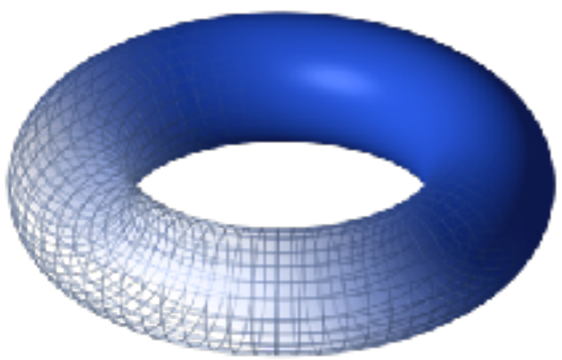
Euler Characteristic

Sphere



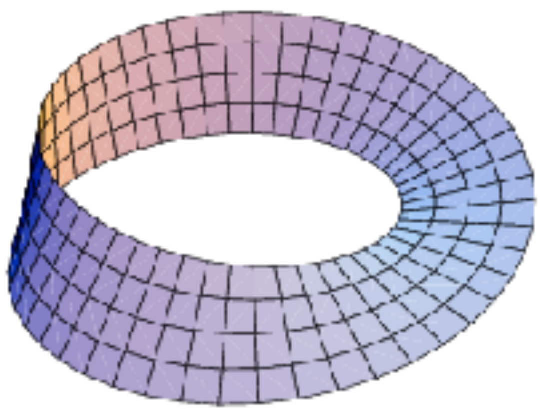
$$\chi = 2$$

Torus



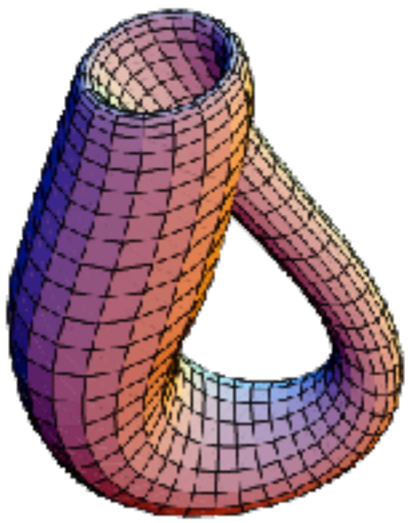
$$\chi = 0$$

Moebius Strip



$$\chi = 0$$

Klein Bottle



$$\chi = 0$$

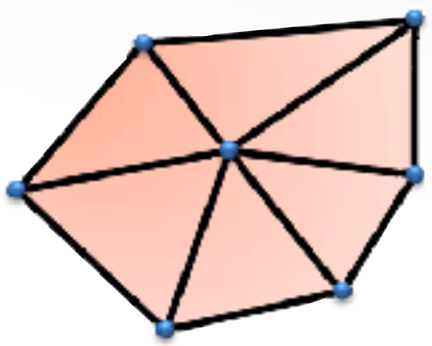
How many pentagons?

How many pentagons?

Any **closed surface** of **genus 0** consisting only of **hexagons** and **pentagons** and where every **vertex has valence 3** must have exactly **12 pentagons**

Two-Manifold Surfaces

Local neighborhoods are disk-shaped

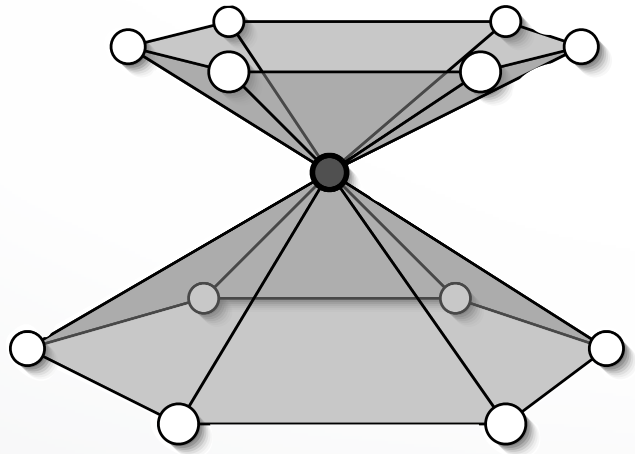


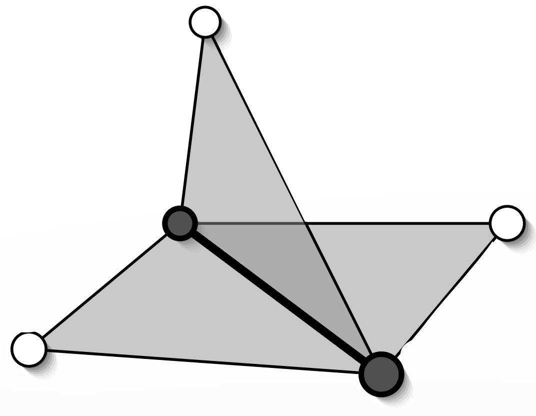
$$\mathbf{f}(D_\epsilon[u,v]) = D_\delta[\mathbf{f}(u,v)]$$

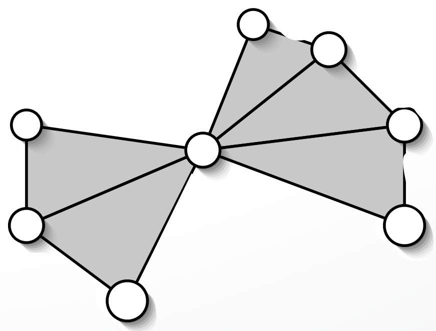
Guarantees meaningful neighbor enumeration

- required by most algorithms

Non-manifold Examples:







Outline

- Parametric Approximations
- Polygonal Meshes
- **Data Structures**

Mesh Data Structures

- How to store geometry & connectivity?
- compact storage and file formats
- Efficient algorithms on meshes
  - Time-critical operations
  - All vertices/edges of a face
  - All incident vertices/edges/faces of a vertex

Data Structures

What should be stored?

- Geometry: 3D vertex coordinates
- Connectivity: Vertex adjacency
- Attributes:
  - normals, color, texture coordinates, etc.
  - Per Vertex, per face, per edge

Data Structures

What should it support?

- Rendering
- Queries
  - What are the vertices of face #3?
  - Is vertex #6 adjacent to vertex #12?
  - Which faces are adjacent to face #7?
- Modifications
  - Remove/add a vertex/face
  - Vertex split, edge collapse

Data Structures

Different Data Structures:

- Time to construct (preprocessing)
- Time to answer a query
    - Random access to vertices/edges/faces
    - Fast mesh traversal
    - Fast Neighborhood query
- Time to perform an operation
    - split/merge
- Space complexity
- Redundancy

Data Structures

Different Data Structures:

- Different topological data storage
- Most important ones are face and edge-based (since they encode connectivity)
- Design decision ~ memory/speed trade-off

Face Set (STL)

Face:

- 3 vertex positions

Triangles		
x_{11} y_{11} z_{11}	x_{12} y_{12} z_{12}	x_{13} y_{13} z_{13}
x_{21} y_{21} z_{21}	x_{22} y_{22} z_{22}	x_{23} y_{23} z_{23}
...	...	...
x_{F1} y_{F1} z_{F1}	x_{F2} y_{F2} z_{F2}	x_{F3} y_{F3} z_{F3}

9*4 = 36 B/f (single precision)
72 B/v (Euler Poincaré)

No explicit connectivity

Shared Vertex (OBJ,OFF)

Indexed Face List:

- Vertex: position
- Face: Vertex Indices

Vertices
x_1 y_1 z_1
...
x_V y_V z_V

Triangles
i_{11} i_{12} i_{13}
...
...
...
...
i_{F1} i_{F2} i_{F3}

12 B/v + 12 B/f = 36B/v

No explicit adjacency info

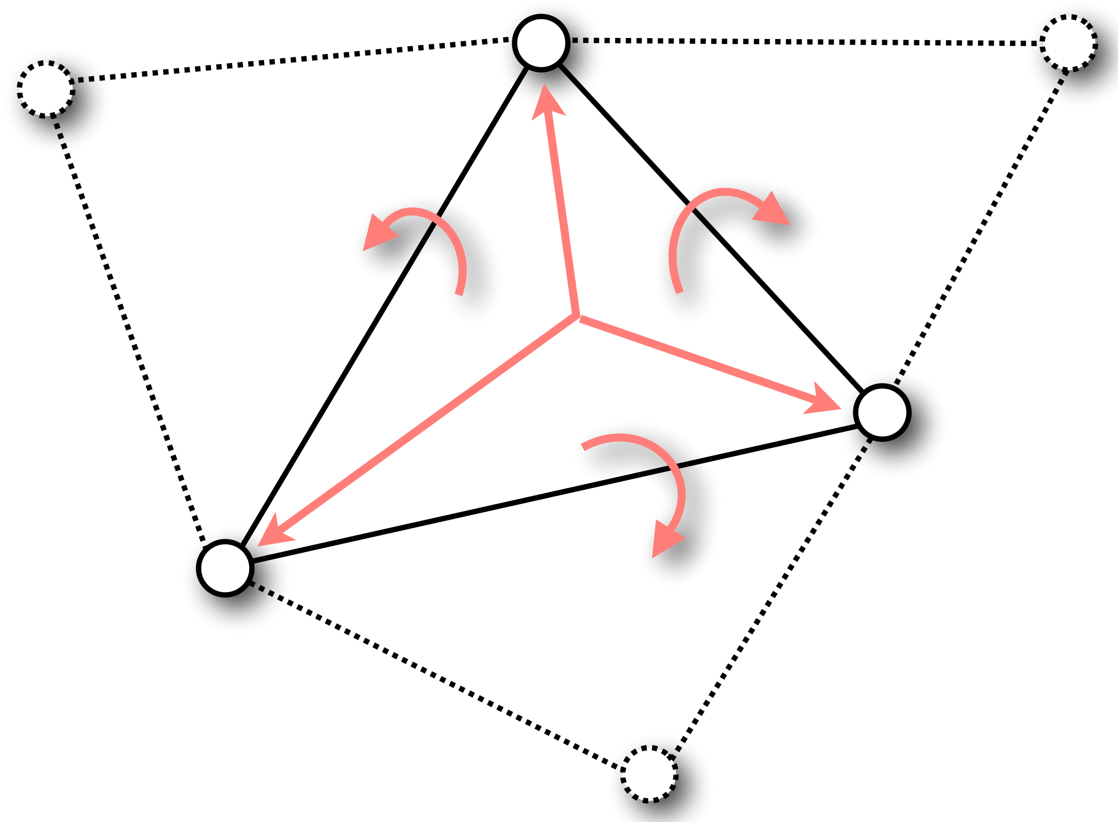
Face-Based Connectivity

Vertex:

- position
- 1 face

Face:

- 3 vertices
- 3 face neighbors



64 B/v

No edges: Special case handling for arbitrary polygons

Edges always have the same topological structure

↓

Efficient handling of polygons with variable valence

(Winged) **Edge-Based Connectivity**

Vertex:

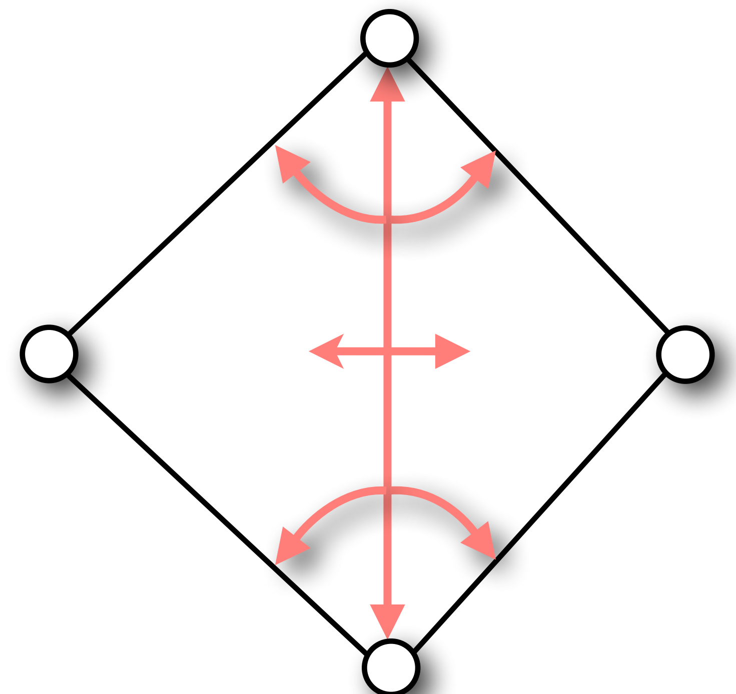
- position
- 1 edge

Edge:

- 2 vertices
- 2 faces
- 4 edges

Face:

- 1 edges



120 B/v

Edges have no orientation: special case handling for neighbors

Halfedge-Based Connectivity

Vertex:

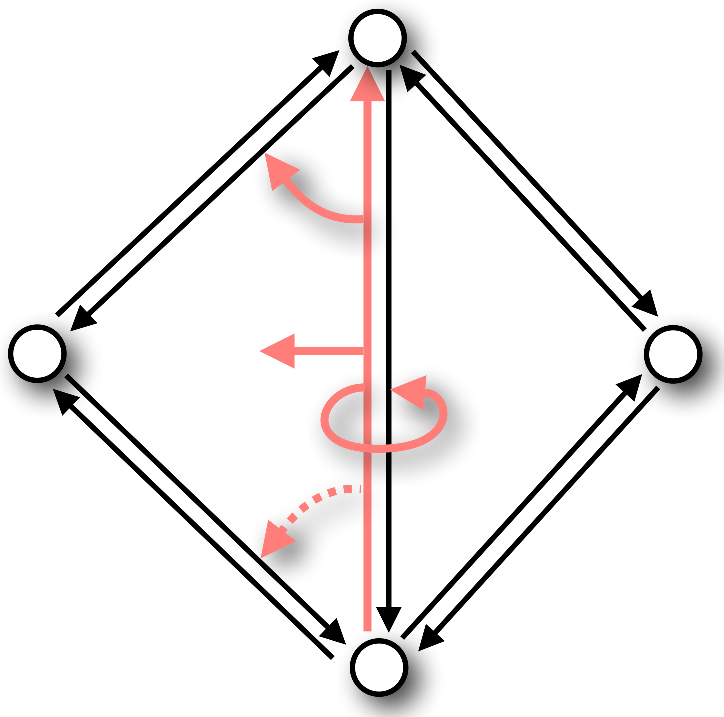
- position
- 1 halfedge

Edge:

- 1 vertex
- 1 face
- 1, 2, or 3 halfedges

Face:

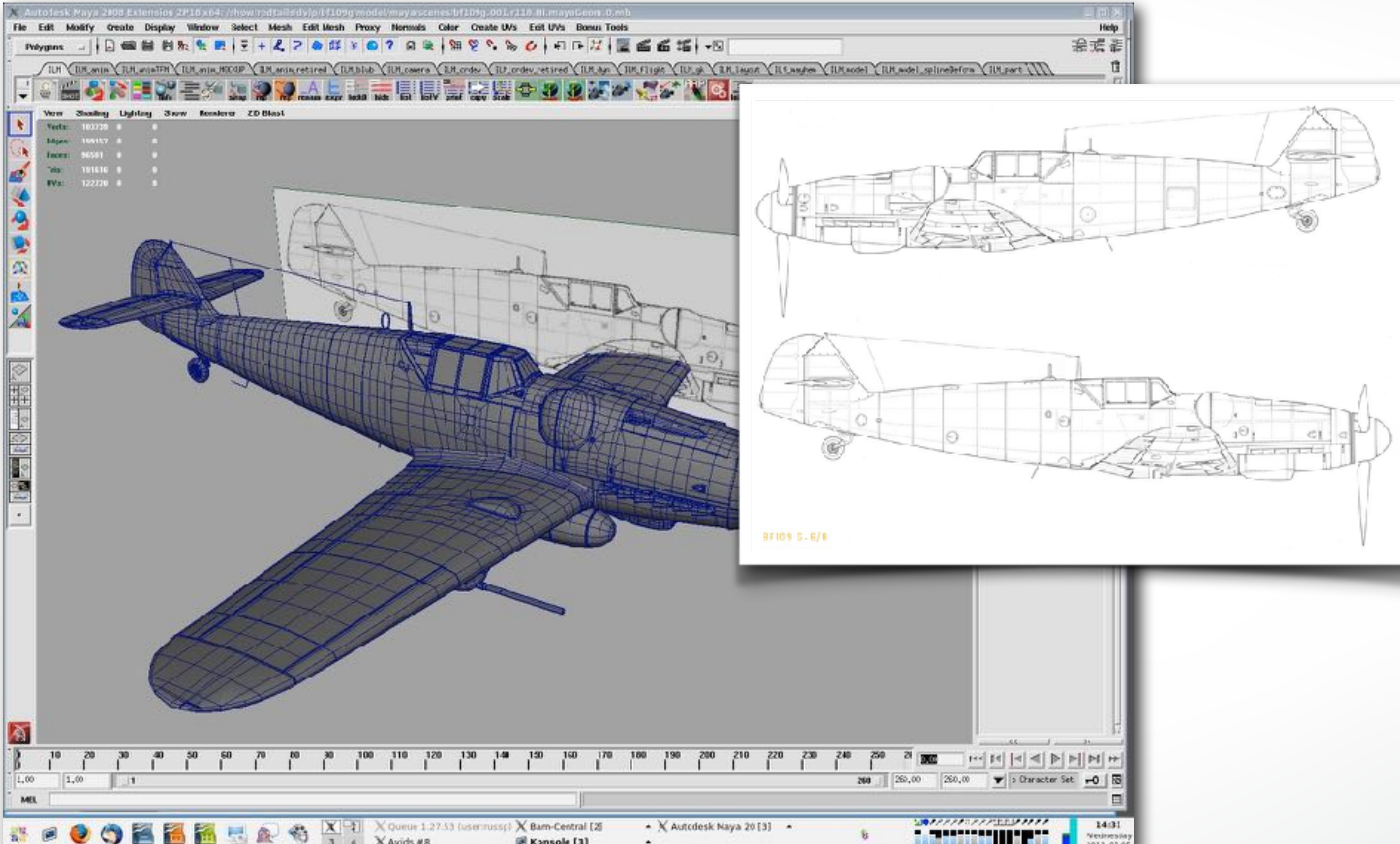
- 1 halfedge



96 to 144 B/v

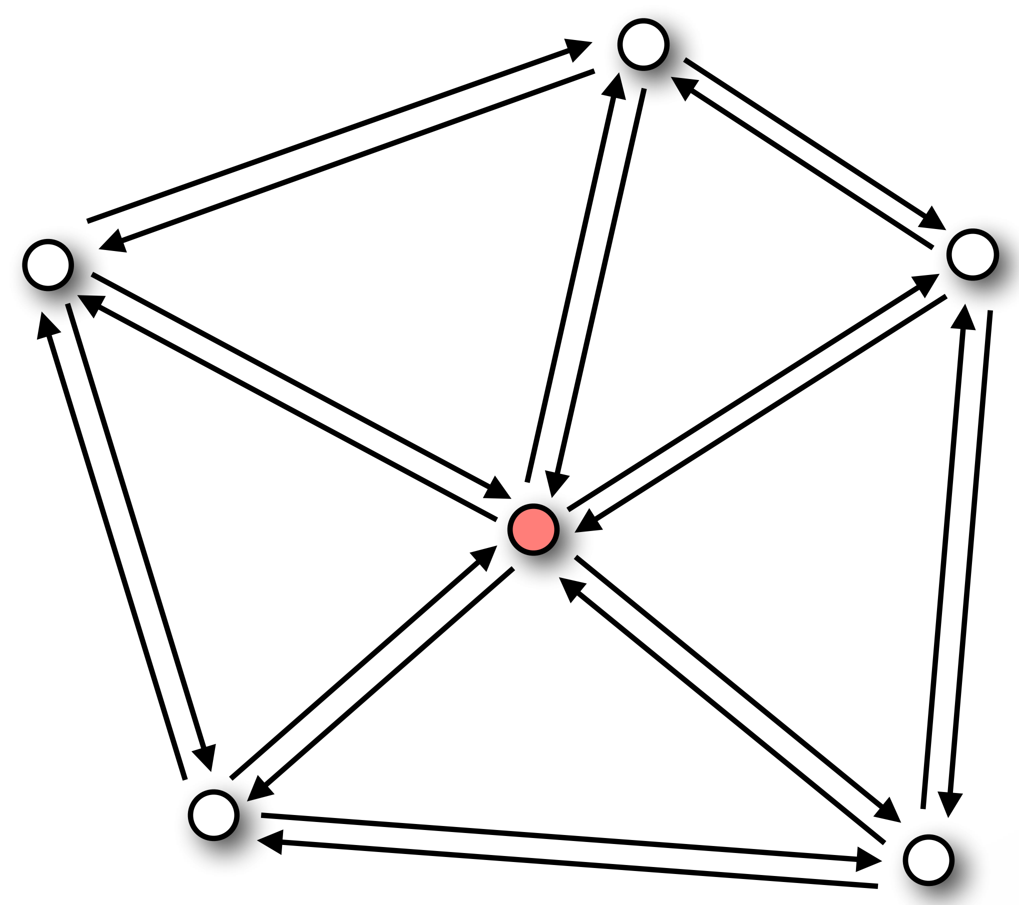
Edges have orientation: No-runtime overhead due to arbitrary faces

Arbitrary Faces during Modeling



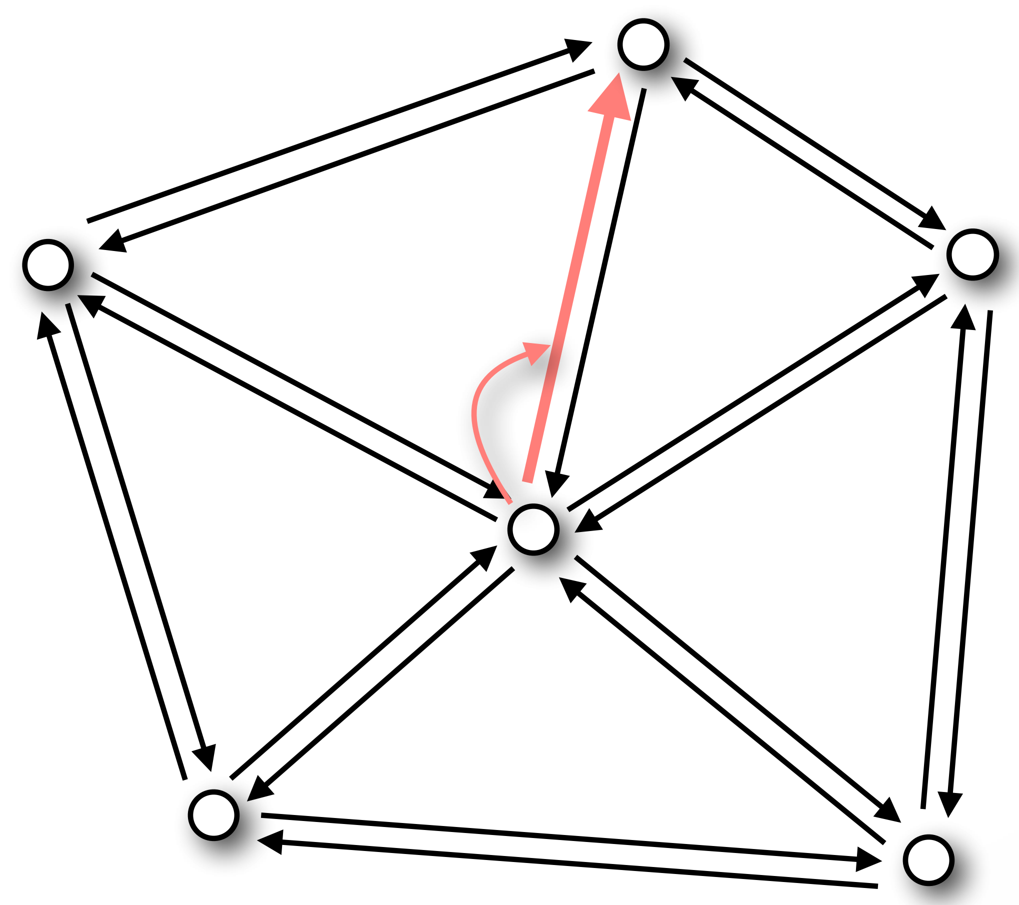
One-Ring Traversal

1. Start at vertex



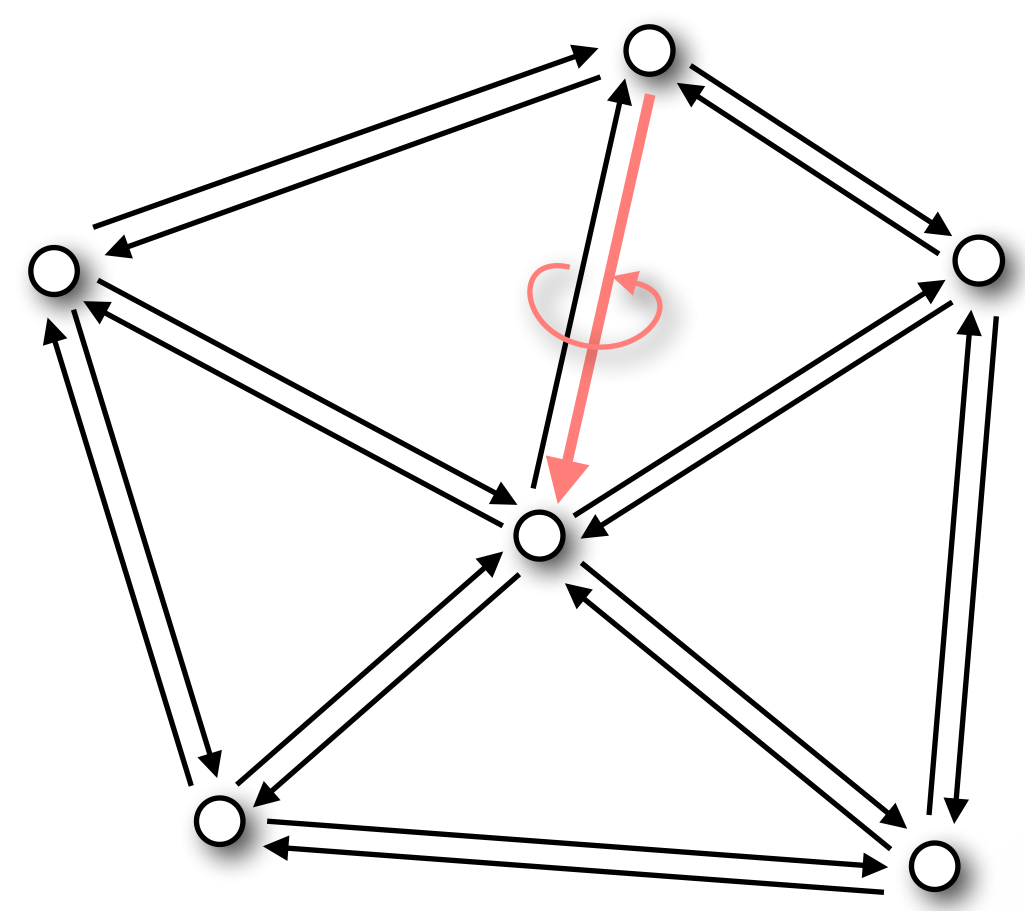
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge



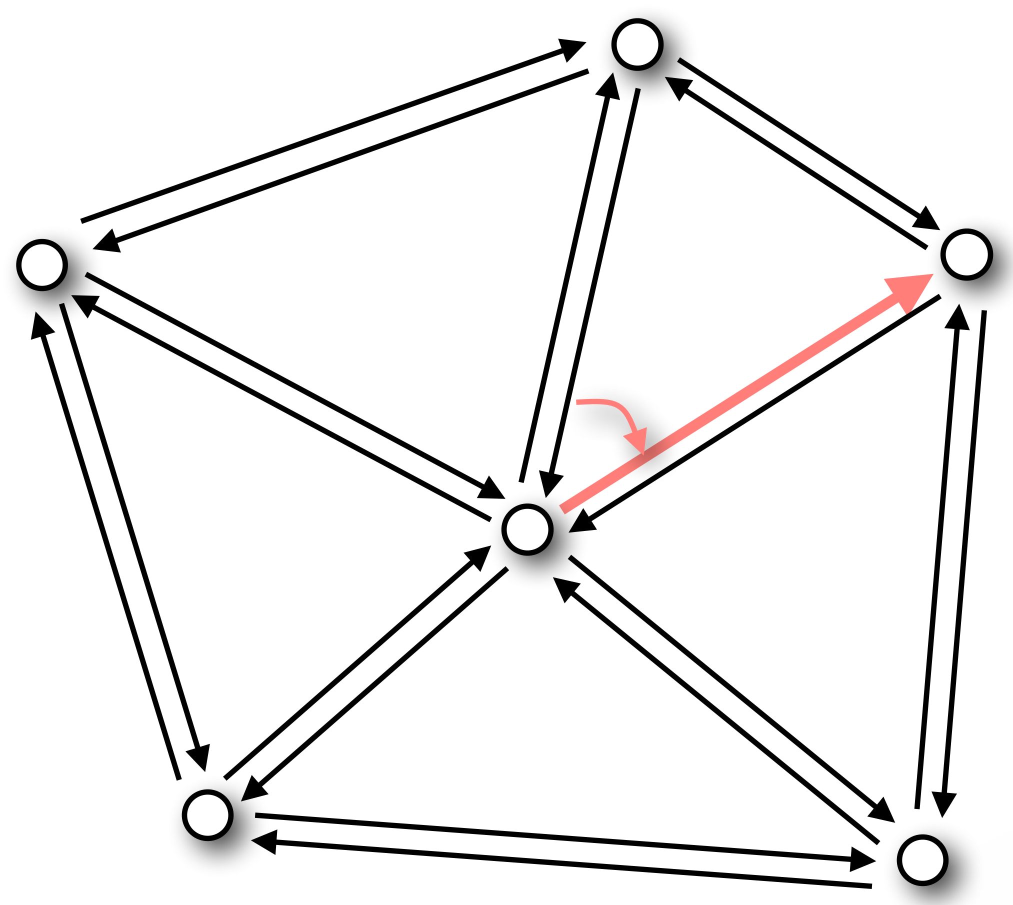
One-Ring Traversal

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



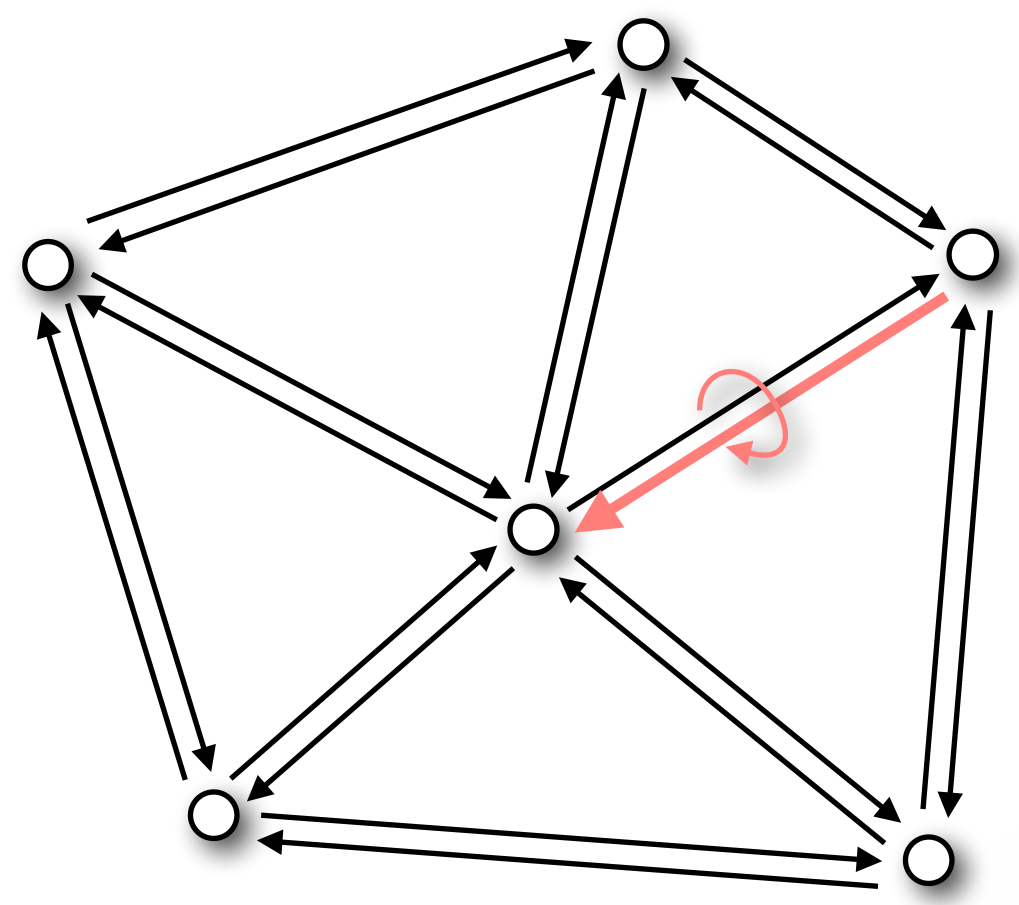
One-Ring Traversal

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



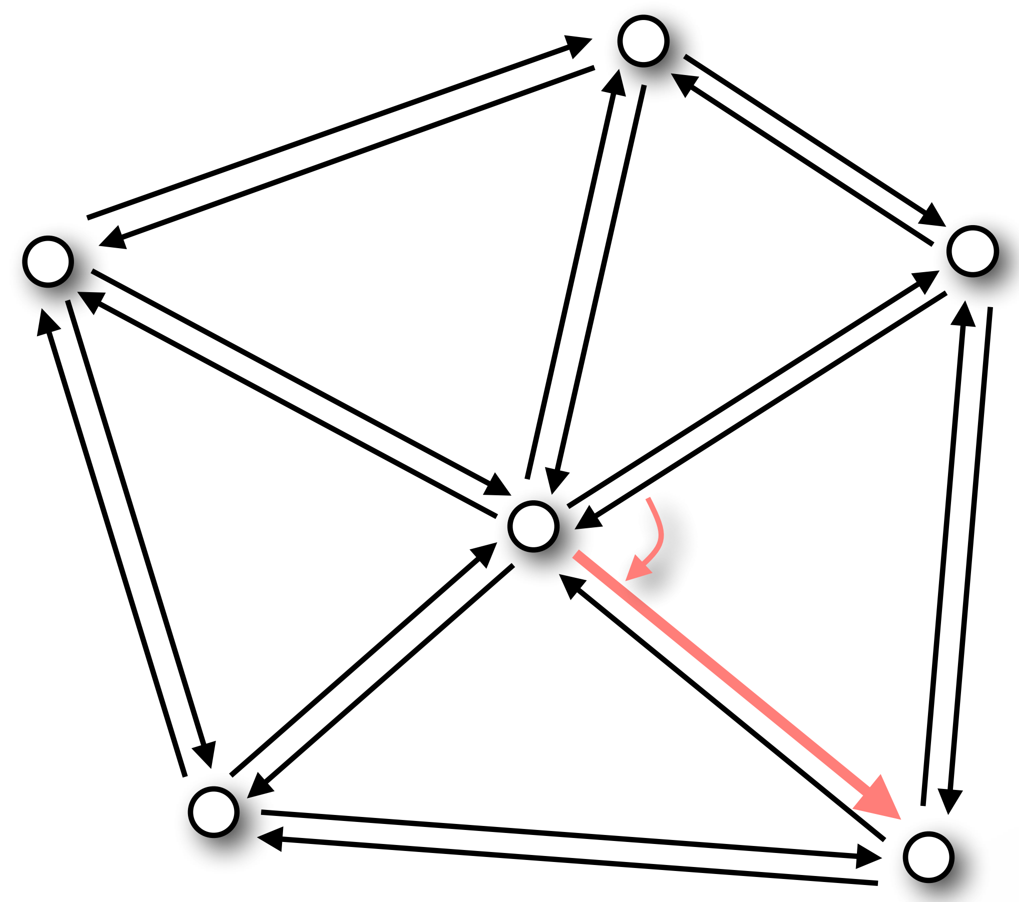
One-Ring Traversal

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. …



Halfedge datastructure Libraries

CGAL

- www.cgal.org
- Computational Geometry
- Free for non-commercial use

OpenMesh

- www.openmesh.org
- Mesh processing
- Free, LGPL license

Why *Open*mesh?

Flexible / Lightweight

- Random access to vertices/edges/faces
- Arbitrary scalar types
- Arrays or lists as underlying kernels

Efficient in space and time

- Dynamic memory management for array-based meshes (not in CGAL)
- Extendable to specialized kernels for non-manifold meshes (not in CGAL)

Easy to Use

Literature

- Textbook: Chapter
- http://www.openmesh.org
- Kettner, Using generic programming for designing a data structure for polyhedral surfaces, Symp. on Comp. Geom., 1998
- Campagna et al., Directed Edges - A Scalable Representation for Triangle Meshes, Journal of Graphics Tools 4(3), 1998
- Botsch et al., OpenMesh - A generic and efficient polygon mesh data structure, OpenSG Symp. 2002

TODO

Learn the **terms** and **notations**

Next Next Time

- **Explicit & Implicit Surfaces**
- **Exercise 1**: Getting Started with Mesh Processing

http://cs621.hao-li.com

Thanks!