Skoltech
Skolkovo Institute of Science and Technology

Surface modalities

Geometric Computer Vision

GCV v2021.1, Module 7

Alexey Artemov, Spring 2021

Lecture Outline

§1. Mesh data structures [30 min]

1.1. What is a mesh?

1.2. Level of detail

1.3. Editing operators

1.4. Mesh data structures

§2. Meshing: constructing meshes [15 min]

2.1. Marching cubes

§3. Defining convolutions on meshes [20 min]

3.1. MeshCNN architecture

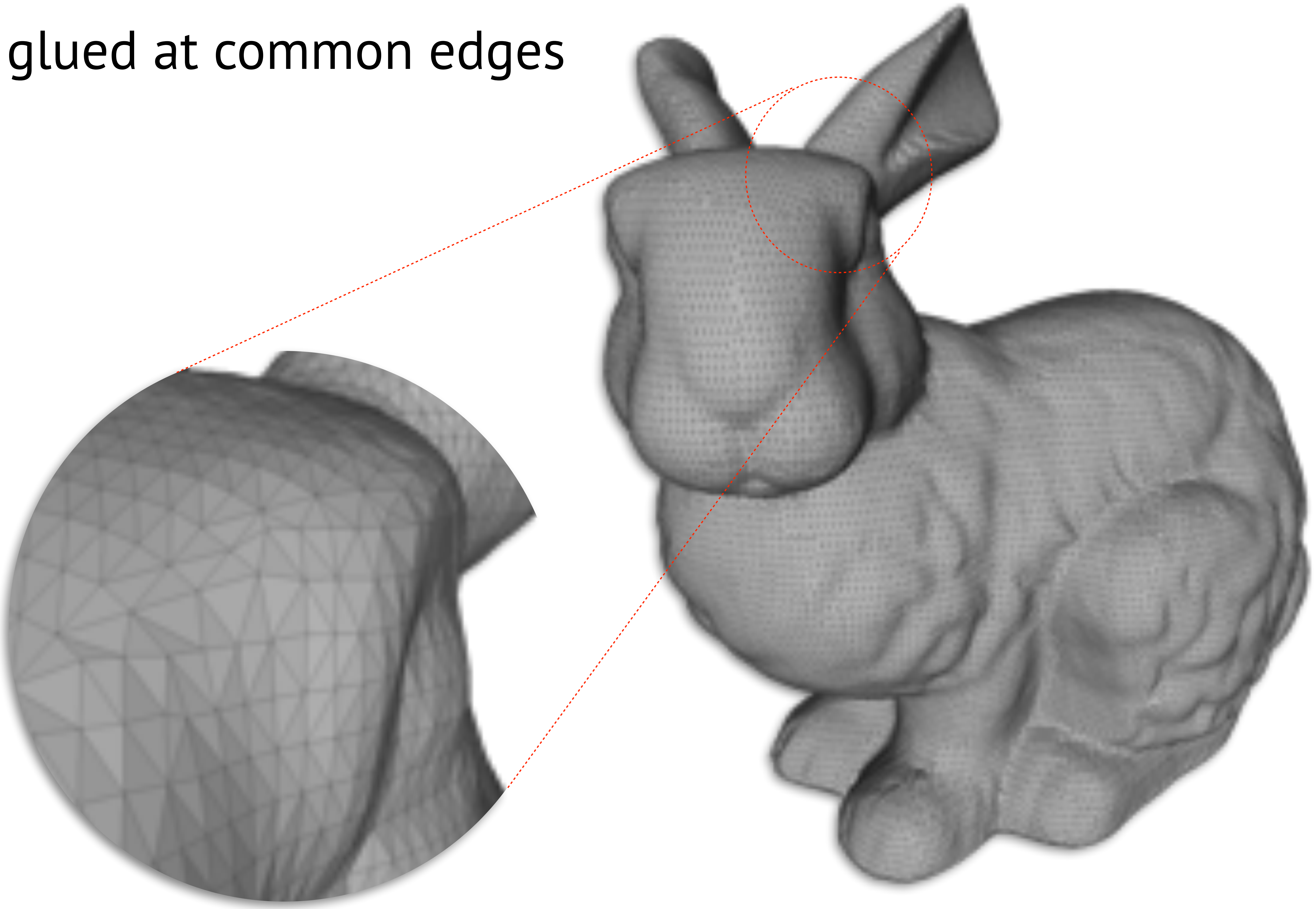
§1. Mesh data structures

What is a mesh?

What is a mesh?

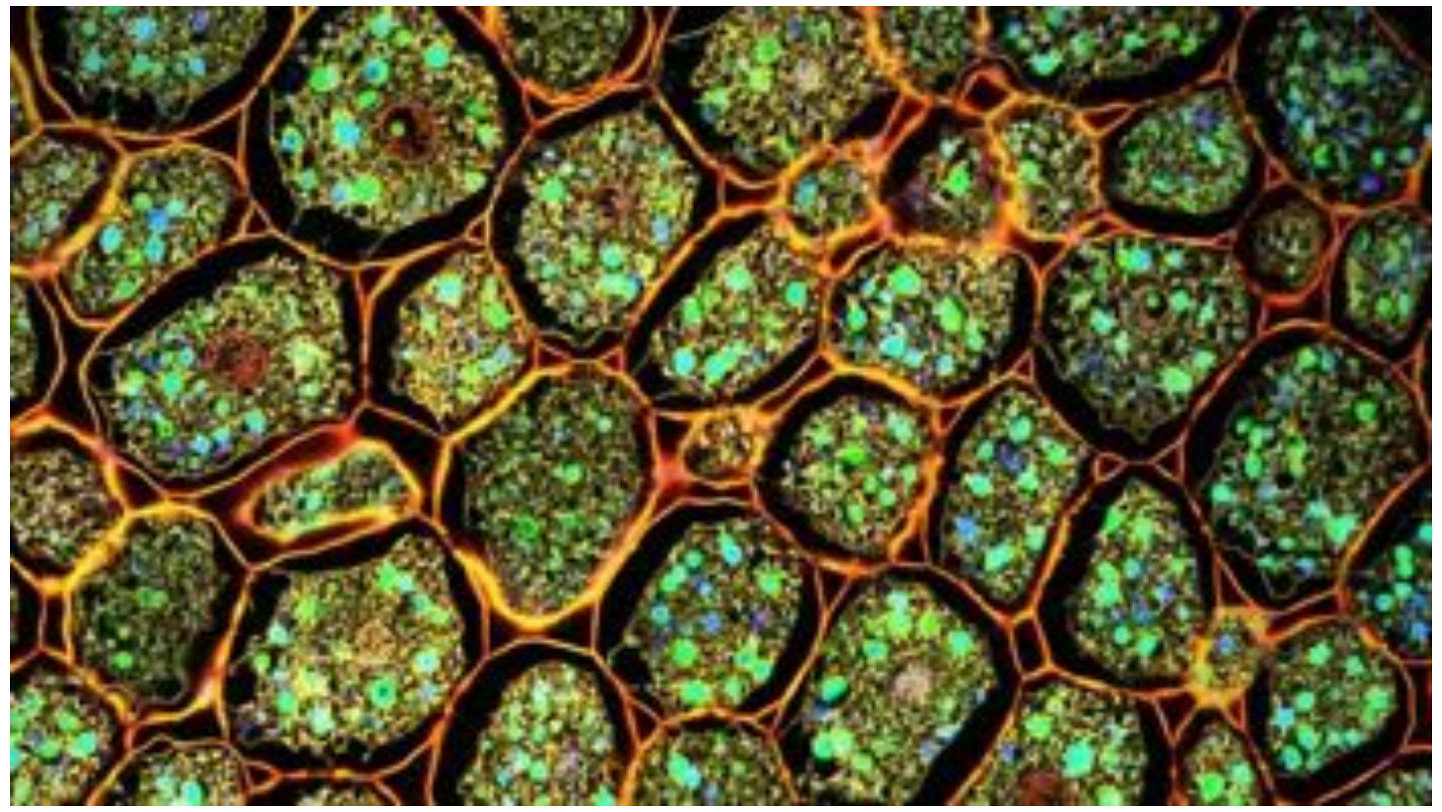
Skoltech
Skolkovo Institute of Science and Technology

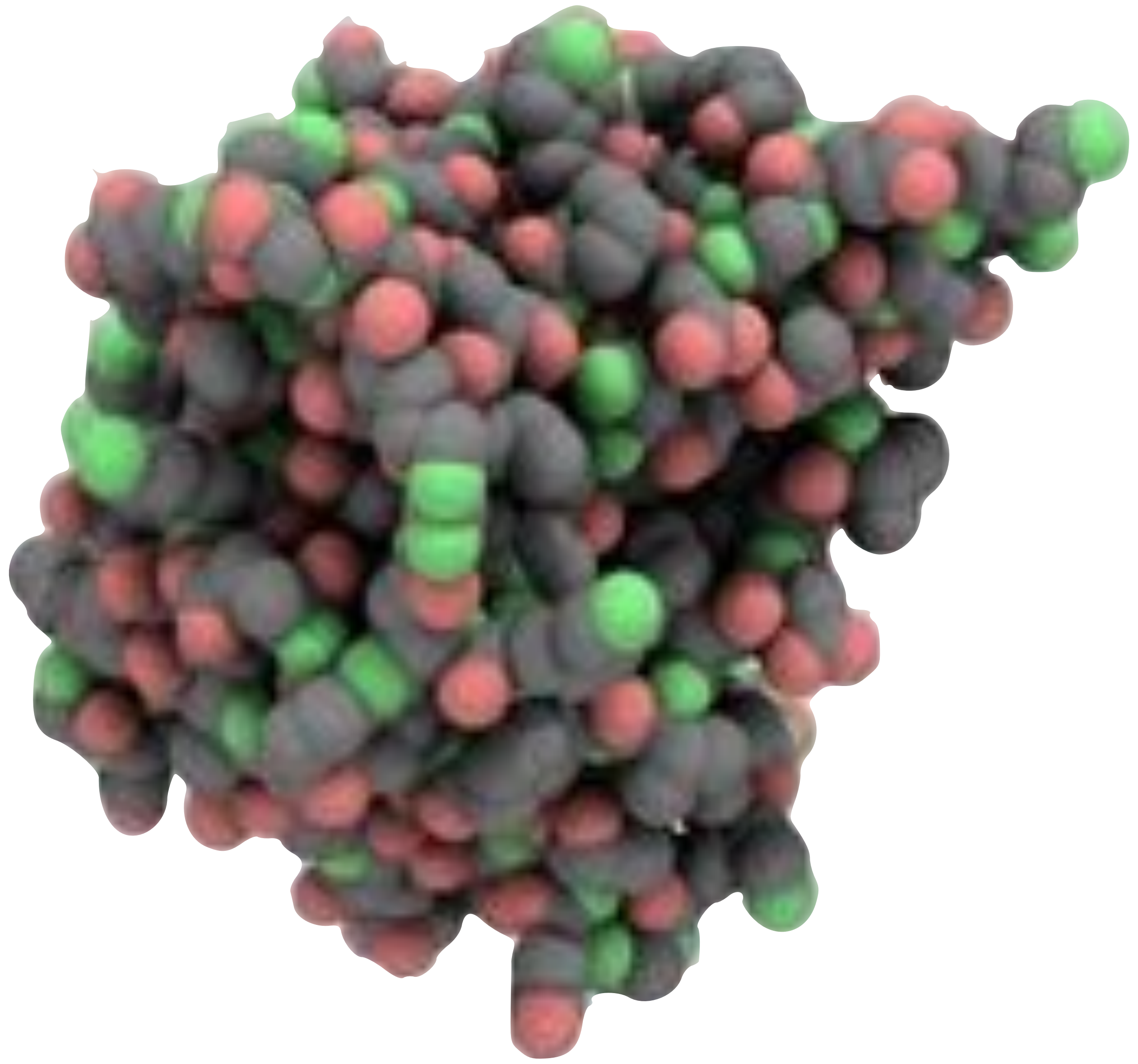
A surface made of polygonal faces glued at common edges



Origin of Meshes

- In nature, meshes arise in a variety of contexts:
  - Cells in organic tissues
  - Crystals
  - Molecules
  - Mostly *convex* but *irregular* cells
  - Common concept: *complex* shapes can be described as *collections* of *simple building blocks*



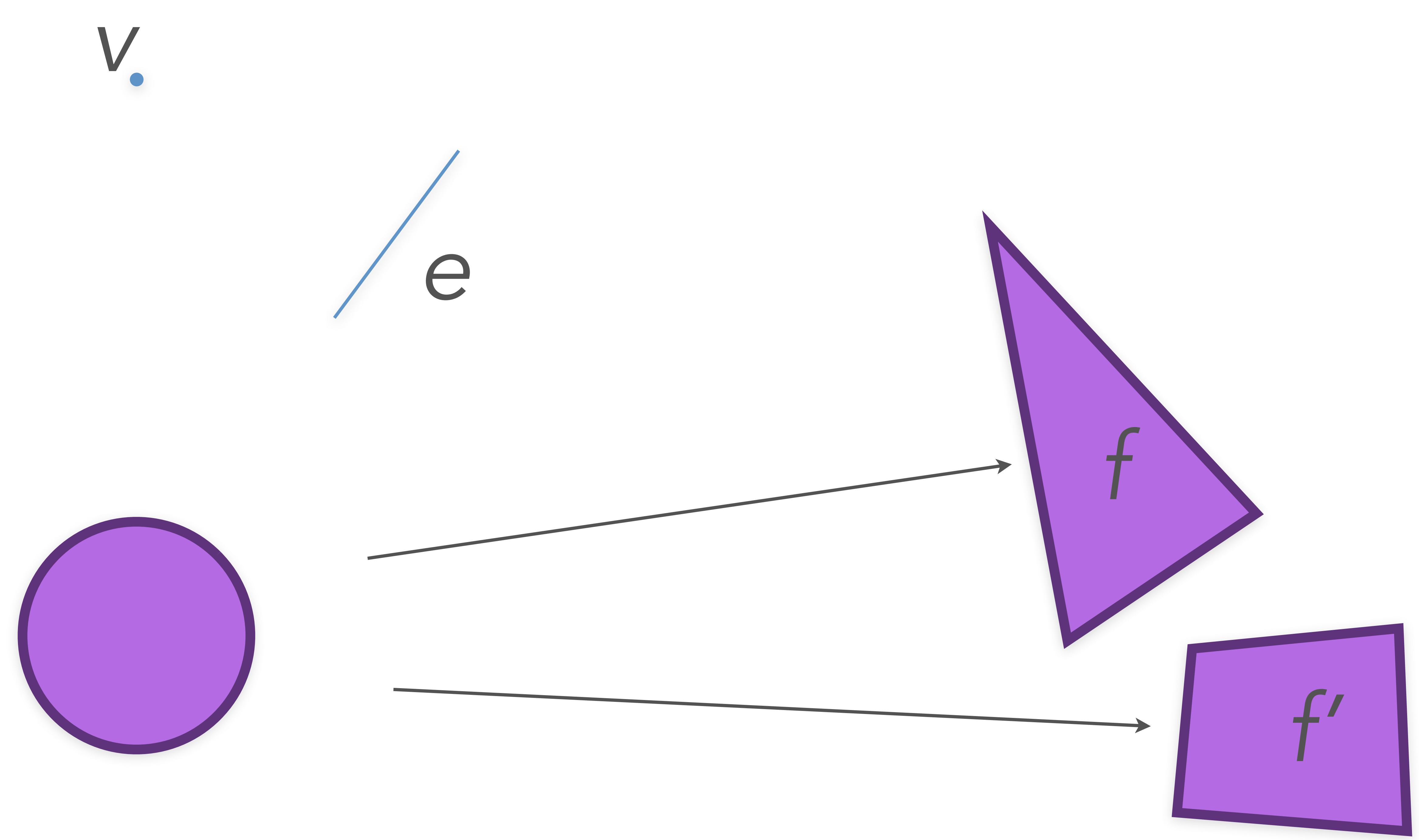


Credit: Fernan Federici Moment Getty Images

By Mark A. Wilson (Department of Geology, The College of Wooster).[1] - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=12666593

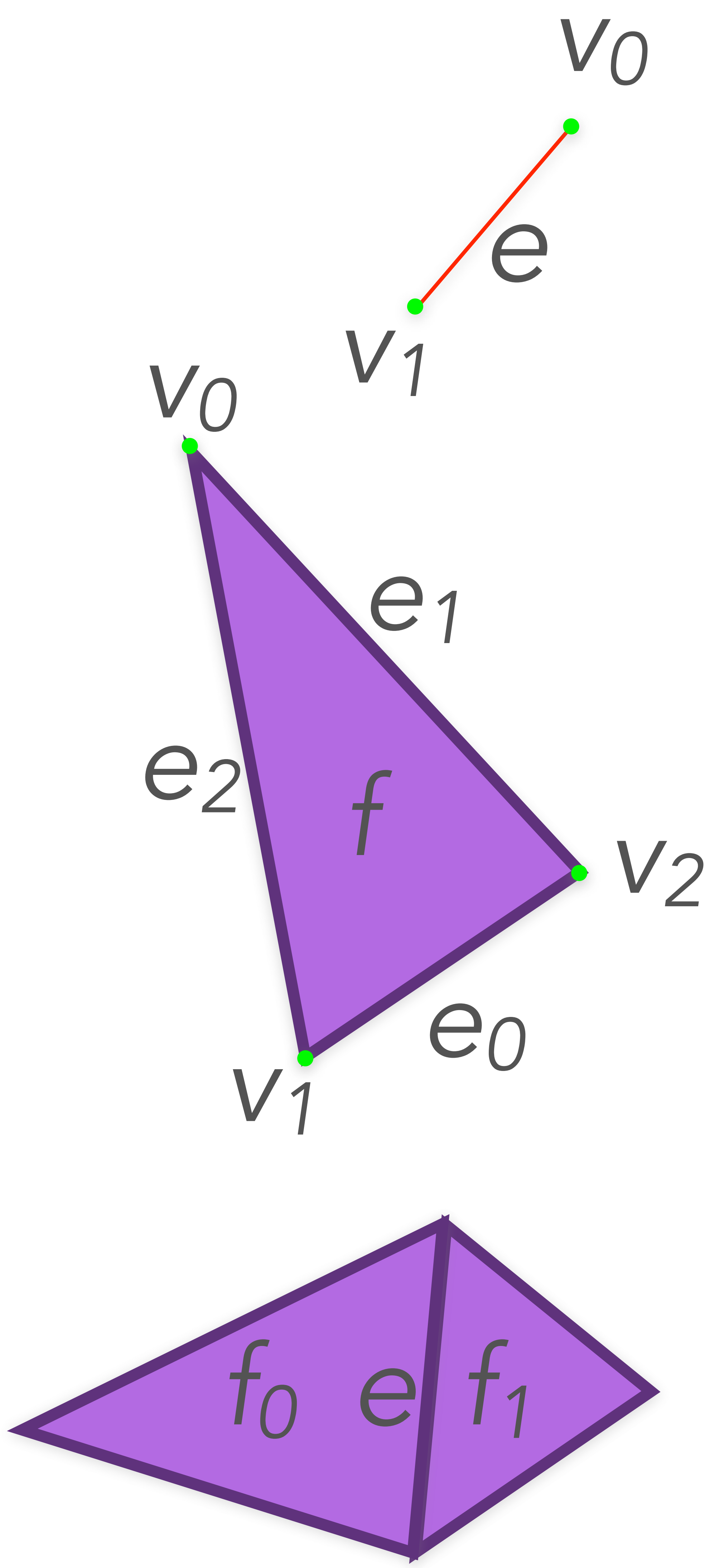
Basic Math of Meshes

- A *n-cell* is a set homeomorphic to a Euclidean disc of dimension *n*:
  - 0-cell: *vertex*
  - 1-cell: *edge*
  - [0,1]→
  - 2-cell: *face*



Structure

- A *mesh M=(V,E,F)* of dimension 2 is made of a collection of *k*-cells for $k = 0, 1, 2$:
  - 0-cells of *V* lie on the boundary of 1-cells of *E*
  - 1-cells of *E* lies on the boundary of 2-cells of *F*
    - (manifoldness) each 1-cell of *E* lies on the boundary of either one or two 2-cells of *F*
  - the intersection of two distinct 1- / 2-cells is either empty or it coincides with a collection of 0- / 1-cells



Structure

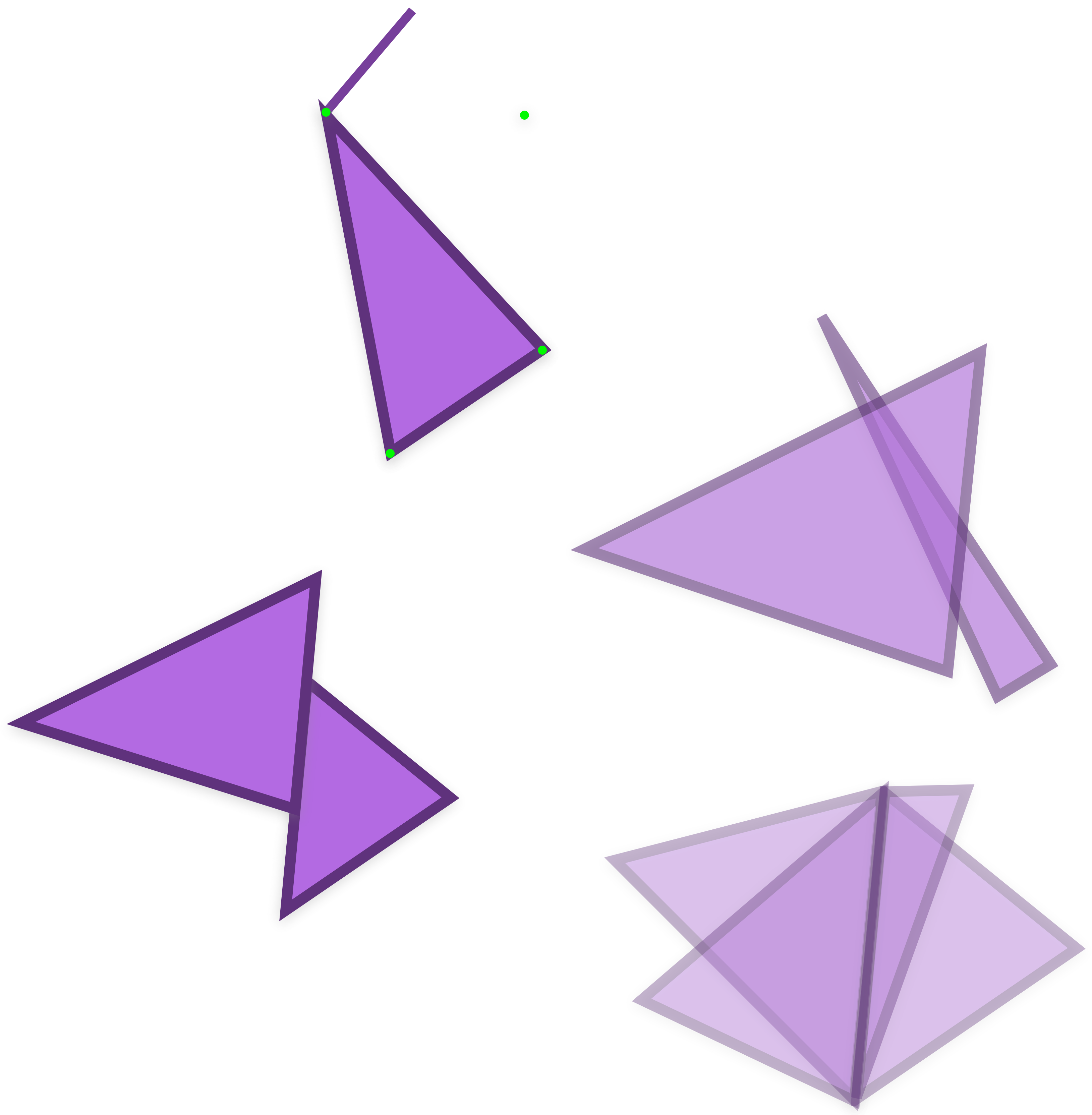
Skolkovo Institute of Science and Technology

- Properties 1 and 2 guarantee that there are no “dangling” edges and isolated vertices
- Property 3 guarantees that faces abut properly
- Property 2.1 extends property 2. to guarantee that the *carrier* of the mesh (i.e., the union of all its cells) is a manifold (i.e., a surface)

Structure

Forbidden configurations:

- Dangling edges and isolated vertices
- Intersecting faces
- Non-conforming adjacency
- Non-manifold edges



Topological Informations

- A mesh can be treated as a purely combinatorial structure

  $M = (V,E,F)$

- For some applications, geometry of edges and faces it not relevant. Just encode:
  - vertices as singletons (V)
  - how vertices are connected among them (E)
  - how cycles of vertices bound faces (F)

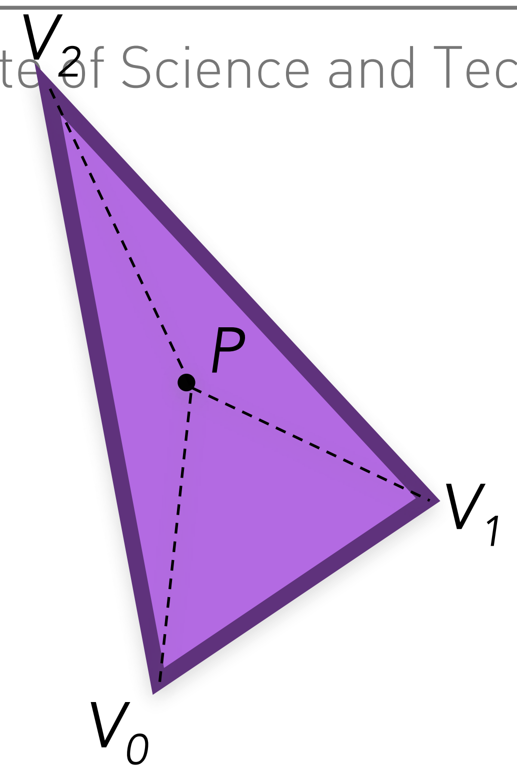
Geometrical Information

Skoltech
Skolkovo Institute of Science and Technology

- Geometric embedding:
  - position in space for each 0-cell (vertex - point)
  - geometry for each 1-cell (edge - line) and 2-cell (face - disk-like surface)
- Polygonal meshes are embedded:
  - edges are straight-line segments
  - are faces flat? not always true: vertices of a face might be *not coplanar*

Triangle Meshes

Skoltech
Skolkovo Institute of Science and Technology



- A *triangle mesh* is a polygonal mesh with all triangular faces
  - all faces are flat (there exist a unique plane for three points)
- All cells are *simplices*, i.e., they are the convex combinations of their vertices

$P = \lambda_0 V_0 + \lambda_1 V_1 + \lambda_2 V_2 \qquad \lambda_i \in [0,1] \qquad \lambda_0 + \lambda_1 + \lambda_2 = 1$

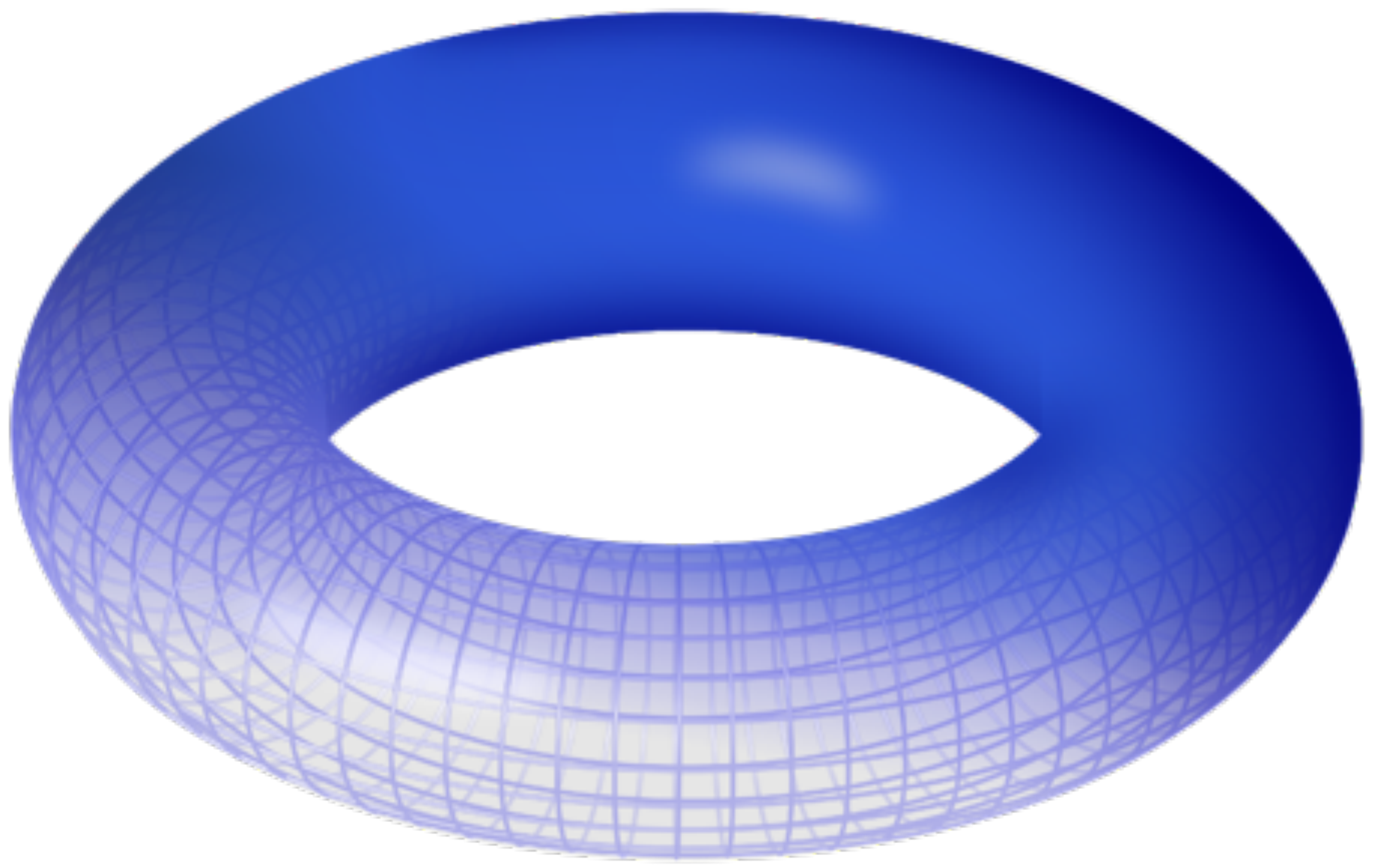
  - embedding of vertices + combinatorial structure characterize the embedding of the whole mesh

Euler-Poincaré Formula

- Relates the number of cells in a mesh with the characteristics of the surface it represents:

  $v\text{-}e\text{+}f = 2s\text{-}2g\text{-}h$

- Shells s = # connected components
- Genus g = # handles (ex.: sphere: genus 0; torus: genus 1)
- Holes h = # boundary loops (watertight: h = 0)



Euler-Poincaré Formula

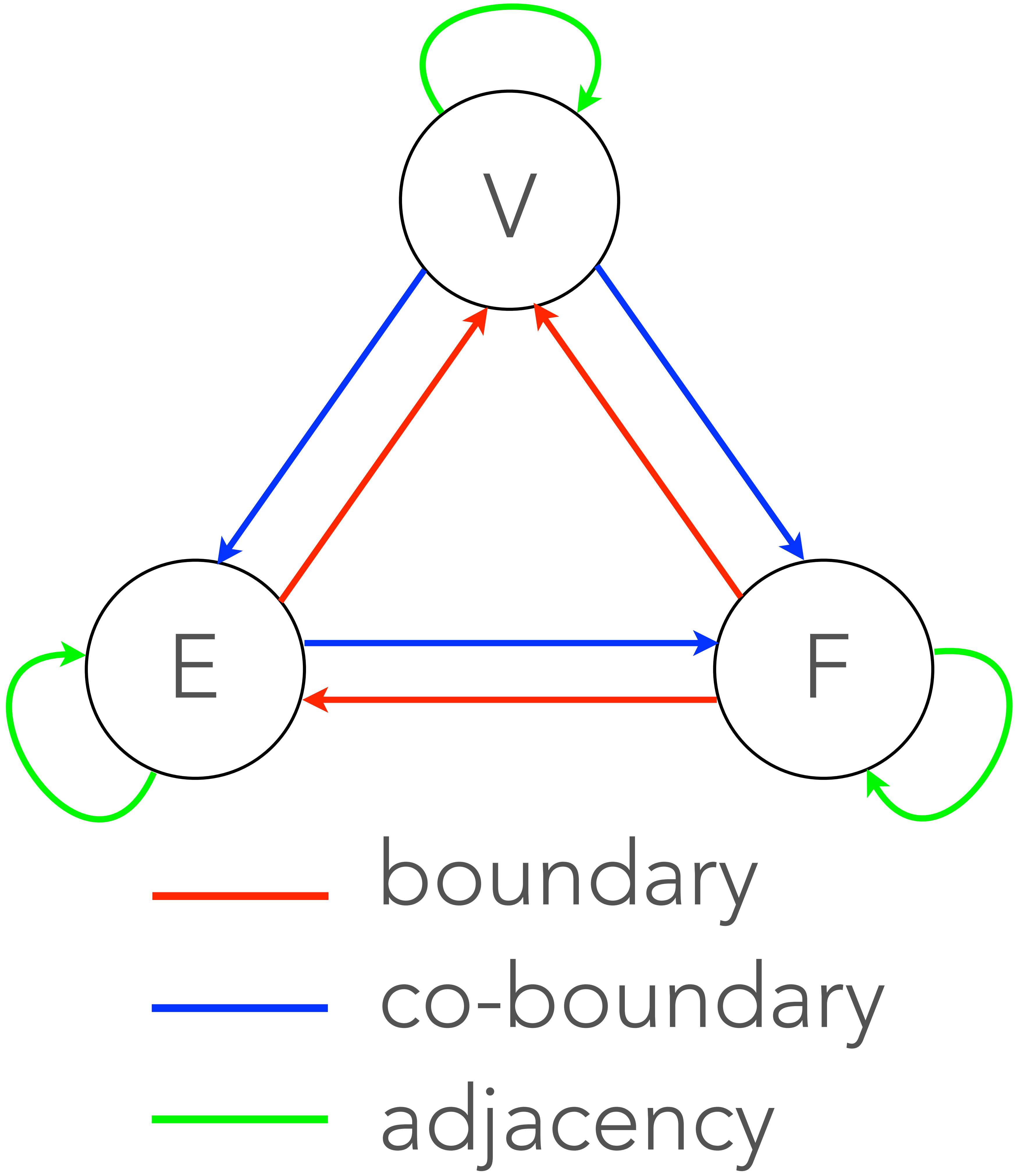
$$v\text{-}e+f = 2s\text{-}2g\ \text{-}h$$

- In a (watertight manifold) triangle mesh:
  - each face has three edges, each edge is shared by two faces:
    - $e = 3f/2$
    - $e = 3v + 6g - 6 \approx 3v$
    - $f = 2v + 4g - 4 \approx 2v$
- The formula can be adapted to bordered surfaces to take into account boundary loops and edges

Topological Relations

Skoltech
Skolkovo Institute of Science and Technology

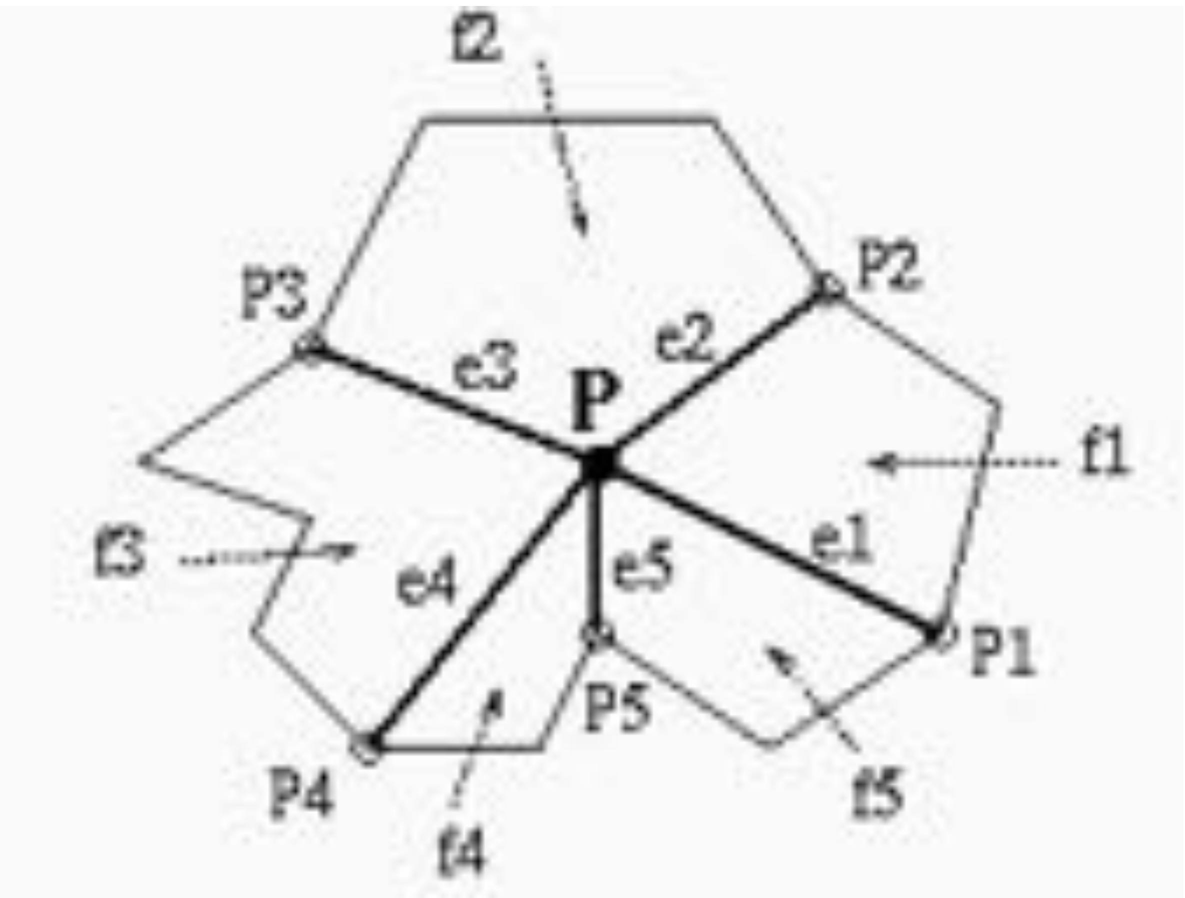
- Boundary relations:
  - for each cell *c* of dimension n, all cells of dimension <n that belong to its boundary
- Co-boundary relations:
  - for each cell *c* of dimension n, all cells of dimension >n such that *c* belongs to their boundary
- Adjacency relations:
  - for each cell *c* of dimension n=1,2, all cells of dimension =n such that share some of their boundary with *c*



Vertex-Based Relations

Skoltech
Skolkovo Institute of Science and Technology

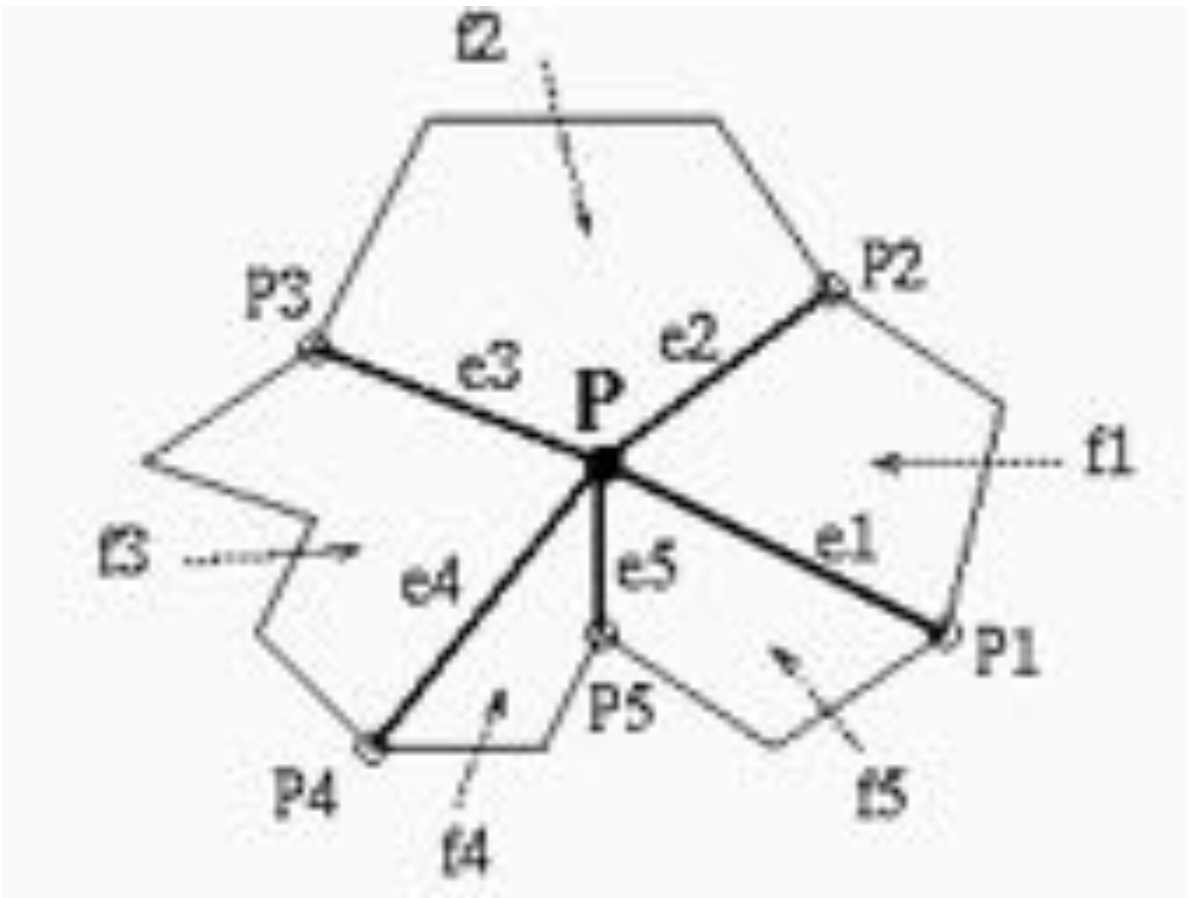
- VE (Vertex-Edge):
  - for each vertex v, the list of edges (e_1,e_2,...,e_r) having an endpoint in v (*incident edges*) arranged in counter-clockwise radial order around v
  - list is circular: initial vertex e_1 is arbitrarily chosen



Slide Credit: Daniele Panozzo

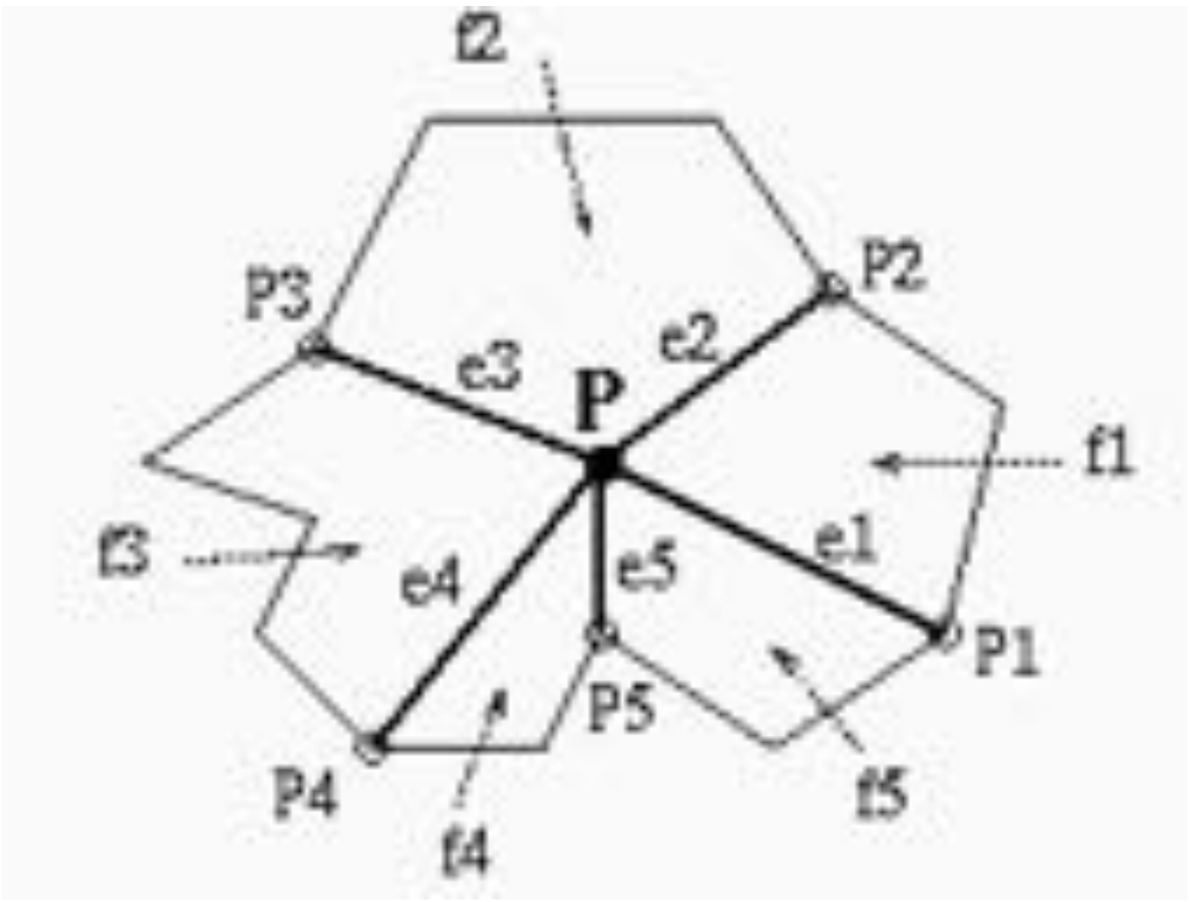
Vertex-Based Relations

- VV (Vertex-Vertex):
  - for each vertex v, the list of vertices (v_1,v_2,...,v_r) connected to v with an edge (*adjacent vertices*) arranged in counter-clockwise radial order around v
  - consistency rule: vertex v_i in VV(v) is an endpoint of edge e_i in VE(v)



Vertex-Based Relations

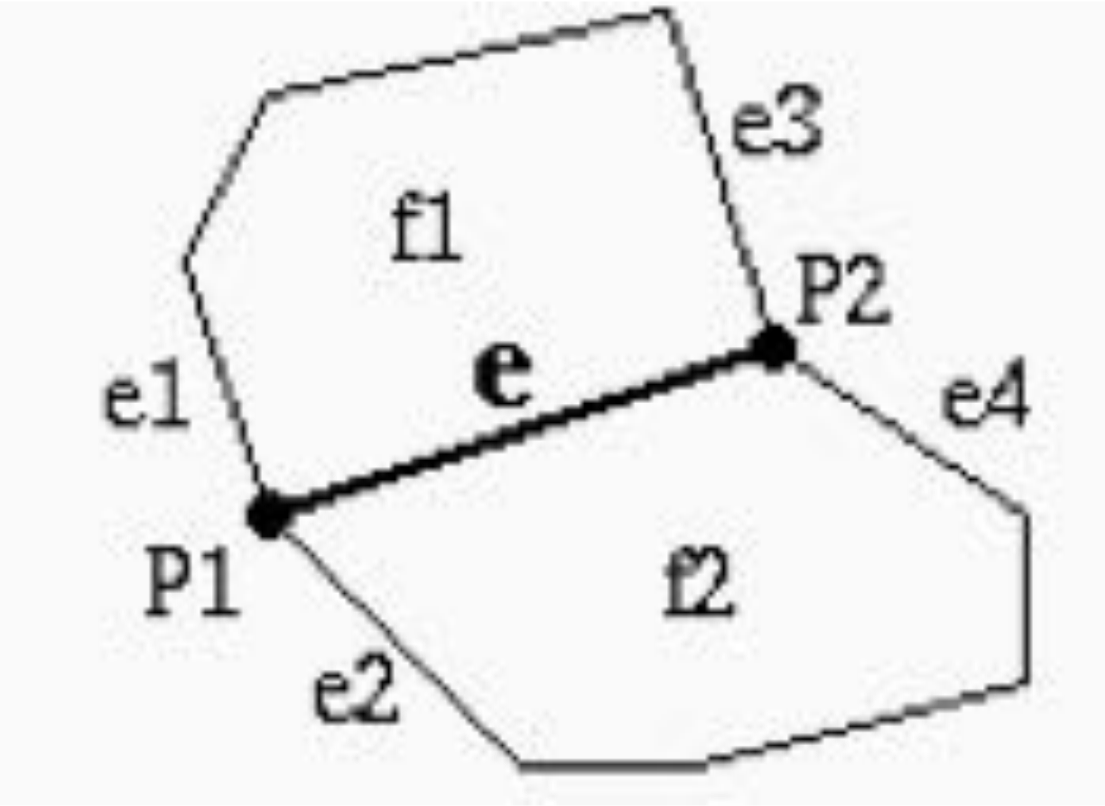
- VF (Vertex-Face):
  - for each vertex v, the list of faces (f_1,f_2,…,f_r) having v on their boundary (*incident faces*) arranged in counter-clockwise radial order around v
  - consistency rule: face f_i in VF(v) is bounded by edges e_i and e_{i+1} in VE(v)



Edge-Based Relations

Skoltech
Skolkovo Institute of Science and Technology

- EV (Edge-Vertex):
  - for each edge e, the two endpoints (v_1,v_2) of e (*incident vertices*)
- EF (Edge-Face):
  - for each edge e, the two faces (f_1,f_2) having e on their boundary (*incident faces*)
- Consistency rule: face f_1 [f_2] is on the left [right] of the oriented line from v_1 to v_2

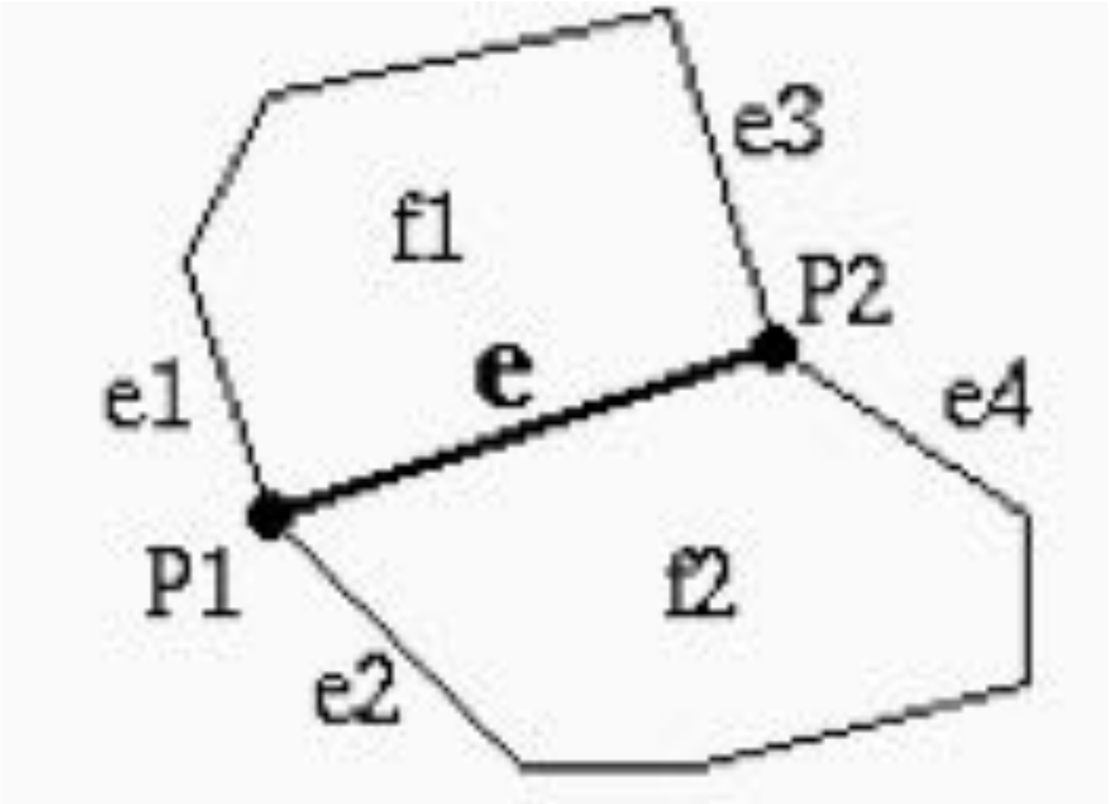


Slide Credit: Daniele Panozzo

Edge-Based Relations

Skoltech
Skolkovo Institute of Science and Technology

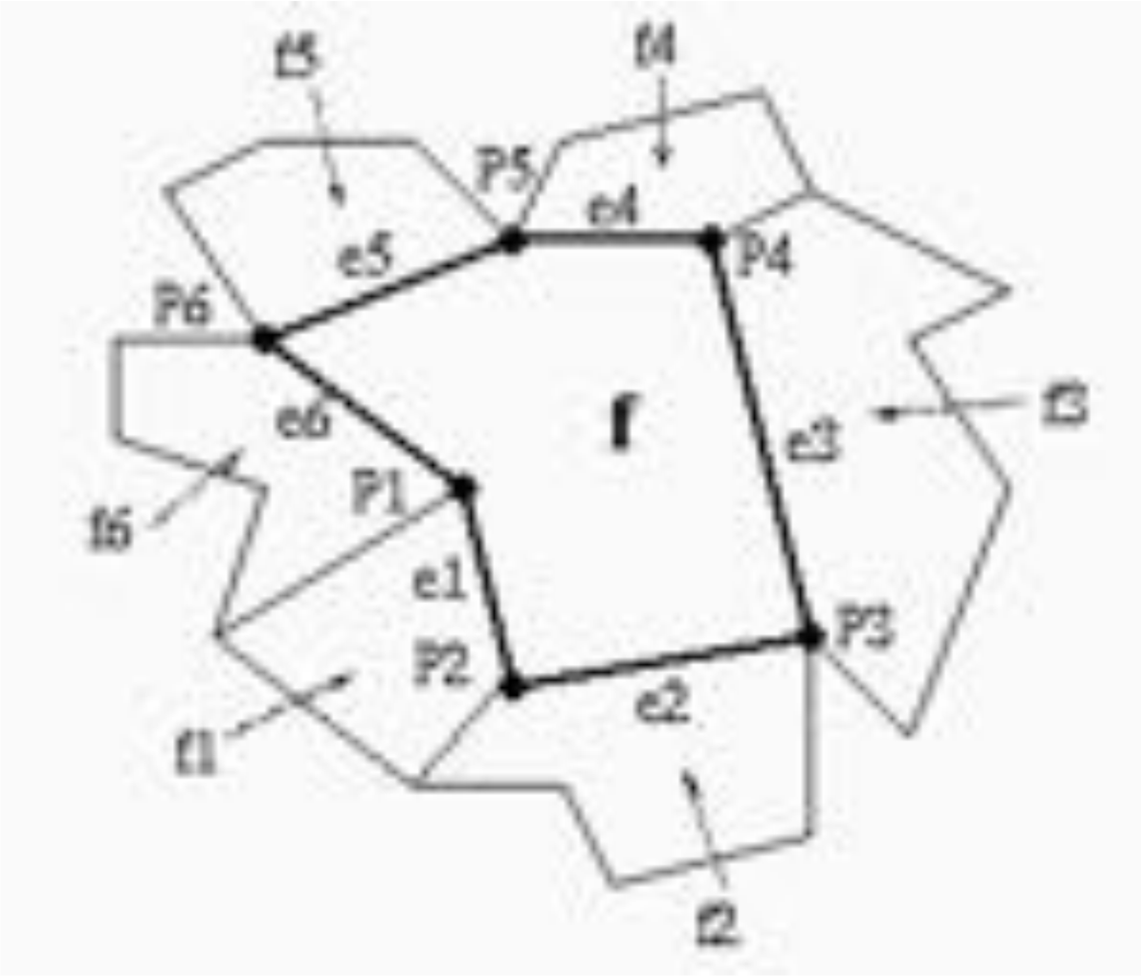
- EE (Edge-Edge):
  - for each edge e, two pairs of edges $((e_1,e_2), (e_3,e_4))$ that share a vertex and a face with e (*adjacent edges*)
- Consistency rule:
  - e_1 is incident on v_1 and f_1
  - e_2 is incident on v_1 and f_2
  - e_3 is incident on v_2 and f_1
  - e_4 is incident on v_2 and f_2



Face-Based Relations

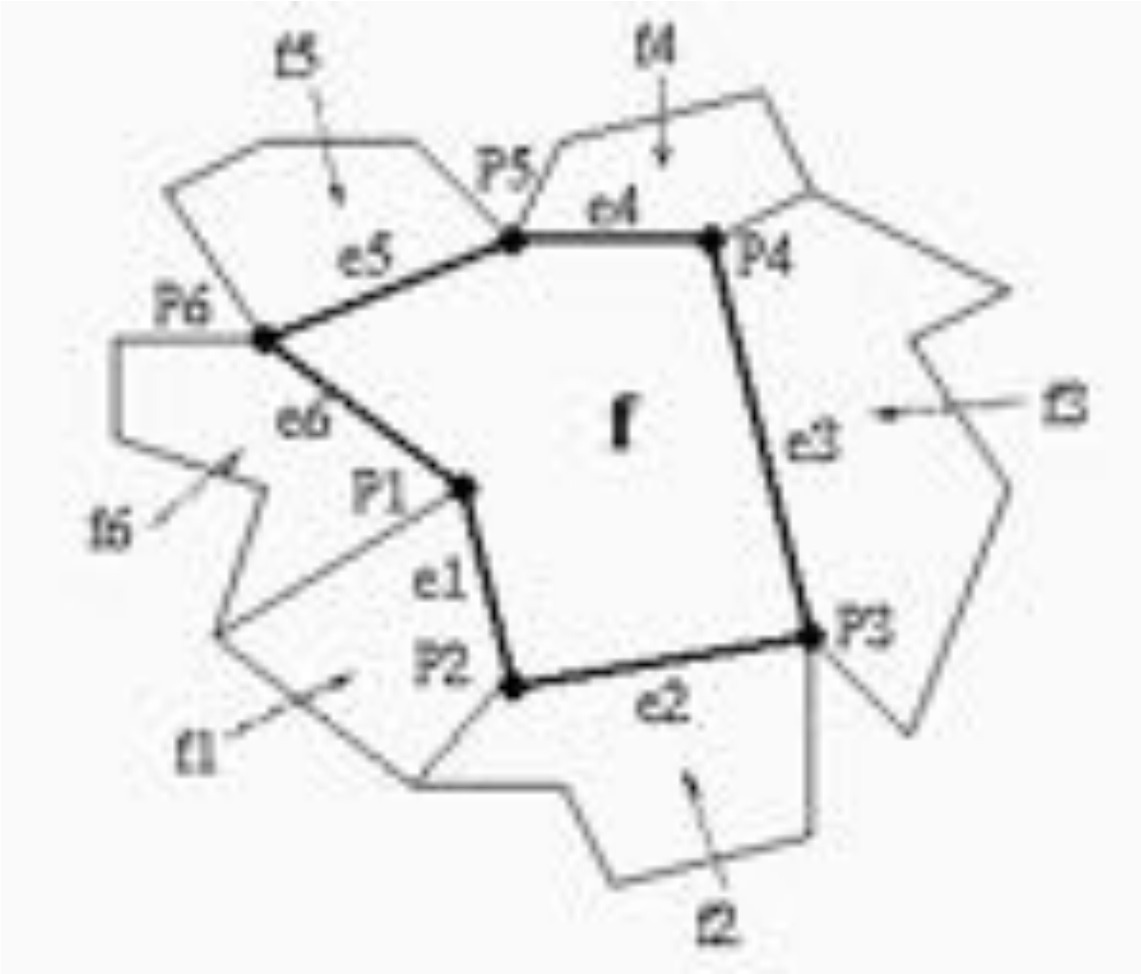
Skoltech
Skolkovo Institute of Science and Technology

- FE (Face-Edge):
  - for each face f, the list (e_1,e_2,...,e_m) of edges of its boundary (*incident edges*), in counter-clockwise order about f
  - list is circular: initial vertex e_1 is arbitrarily chosen



Face-Based Relations

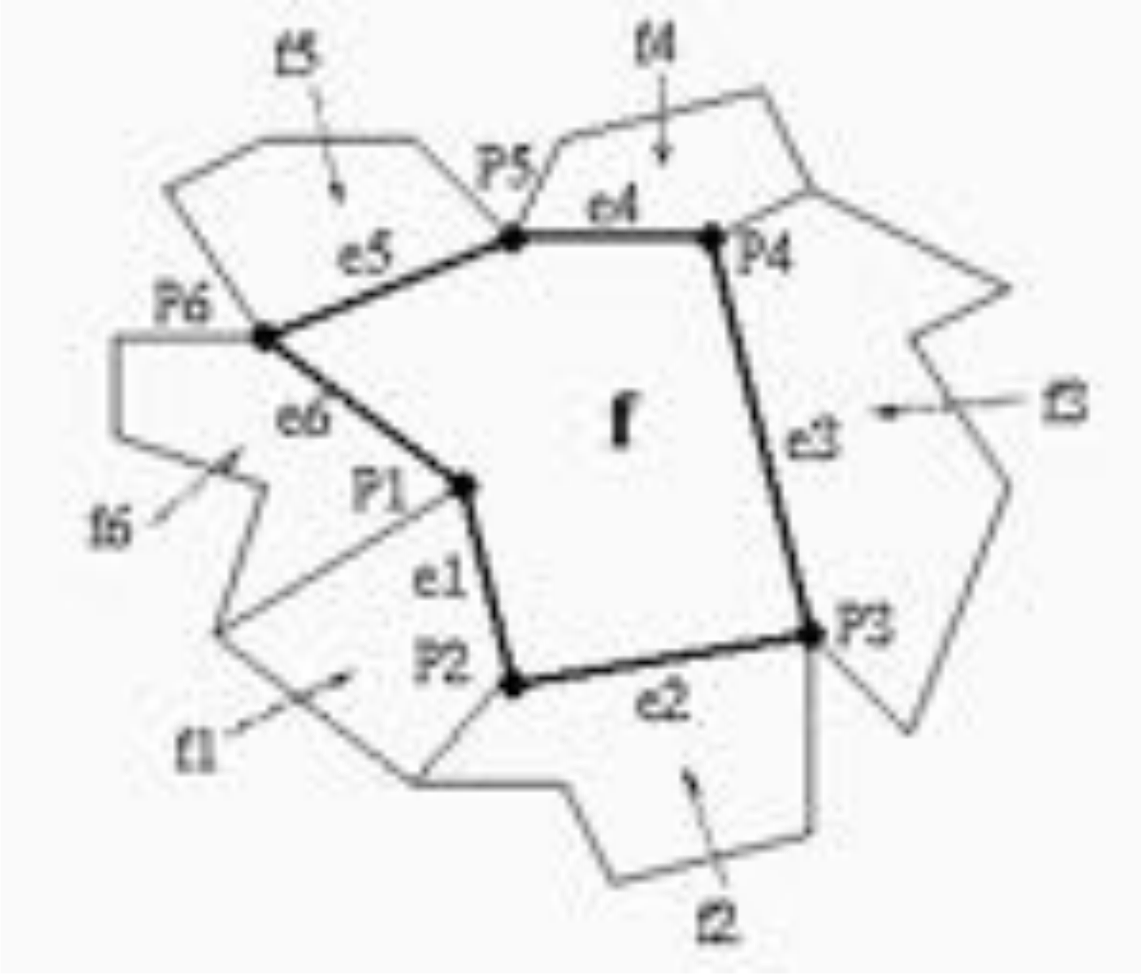
- FV (Face-Vertex):
  - for each face f, the list (v_1,v_2,...,v_m) of vertices of its boundary (*incident vertices*), in counter-clockwise order about f
  - consistency rule: edge e_i in FE(f) has endpoints v_i and v_{i+1}



Face-Based Relations

Skoltech
Skolkovo Institute of Science and Technology

- FF (Face-Face):
  - for each face f, the list (f_1,f_2,...,f_m) of faces that share an edge with f (*adjacent faces*), in counter-clockwise order about f
  - consistency rule: face f_i in FF(f) shares edge e_i in FE(f)

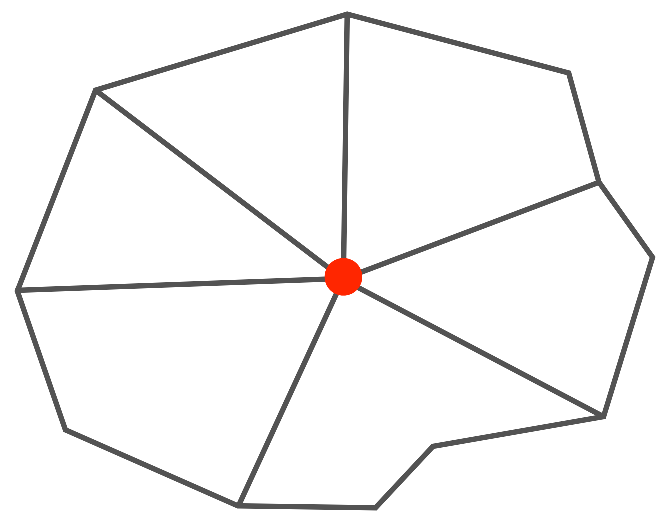


Topological Relations

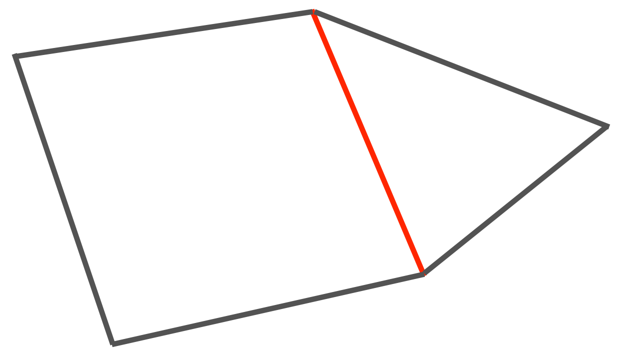
- Constant relations return a constant number of elements:
  - EV (each edge has two endpoints)
  - EE (each edge has four adjacent edges)
  - EF (each edge has two incident faces)
- Variable relations return a variable number of elements:
  - VV, VE, VF, FV, FE, FF: the number of vertices/edges/faces incident/adjacent to a given vertex/face is not constant and it can be of the same order of the total number of vertices/edges/faces

Stars and Rings

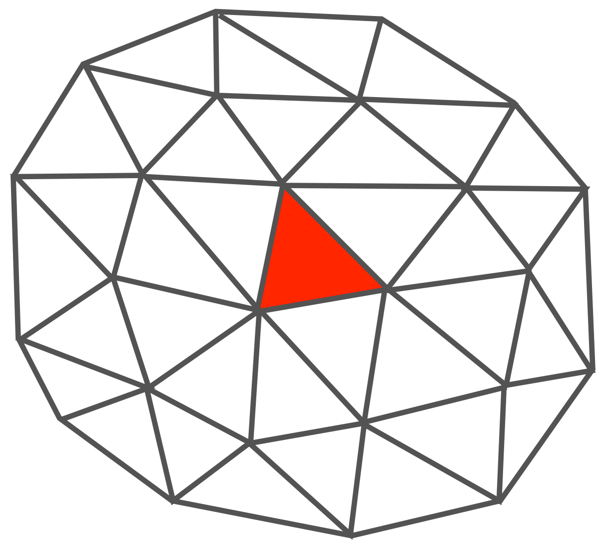
- The star of a vertex v is formed by v plus the set of cells incident at v (edges and faces of its co-boundary)



- The star of an edge e is formed by e plus the set of faces incident at e (faces of its co-boundary)



- The 1-ring of a face f is the formed by the union of the stars of its boundary vertices



- The k-ring of a face f, for k>1 is the formed by the union of the 1-rings of faces in its (k-1)-ring

Level of Details

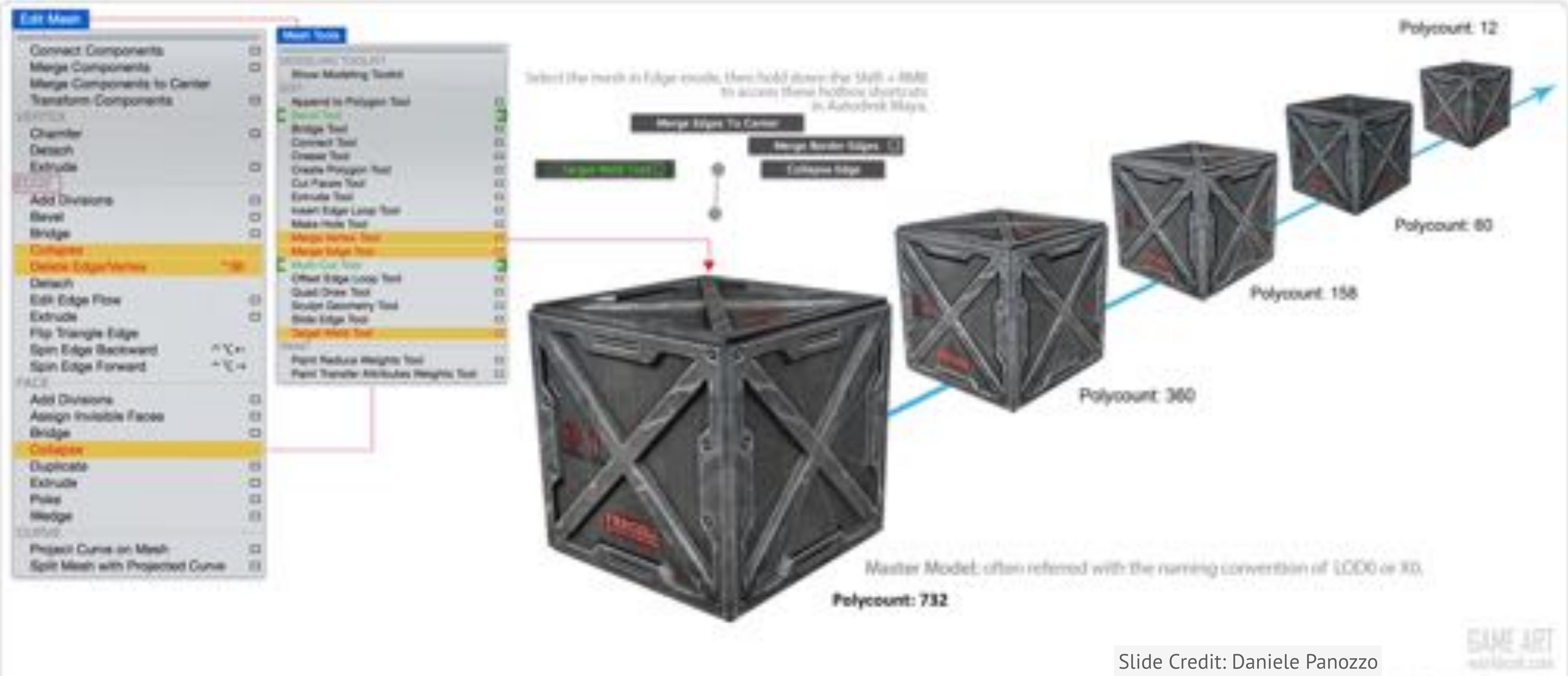
Continuous Methods

- Widely used for rendering terrains

Image					
Vertices	~5500	~2880	~1580	~670	140
Notes	Maximum detail, for closeups.				Minimum detail, very far objects.

Source: http://en.wikipedia.org/wiki/Level_of_detail

Discrete LOD



Slide Credit: Daniele Panozzo

Editing Operators

Euler Operators

Skolkovo Institute of Science and Technology

- Change a mesh while fulfilling the Euler formula $v - e + f = 2s - 2g - h$
- Challenging to implement
- Examples:
  - MVS - MakeVertexShell: creates a new connected component composed of a single vertex
  - MEV - MakeEdgeVertex: creates a new vertex and a new edge, joining it to an existing vertex
  - MEF - MakeEdgeFace: connects two existing vertices with an edge creating a new face - this can either make and fill a loop, or split an existing face into two
  - KHMF - KillHoleMakeFace: fills a hole loop with a face
  - ...

Operators for Triangle Meshes

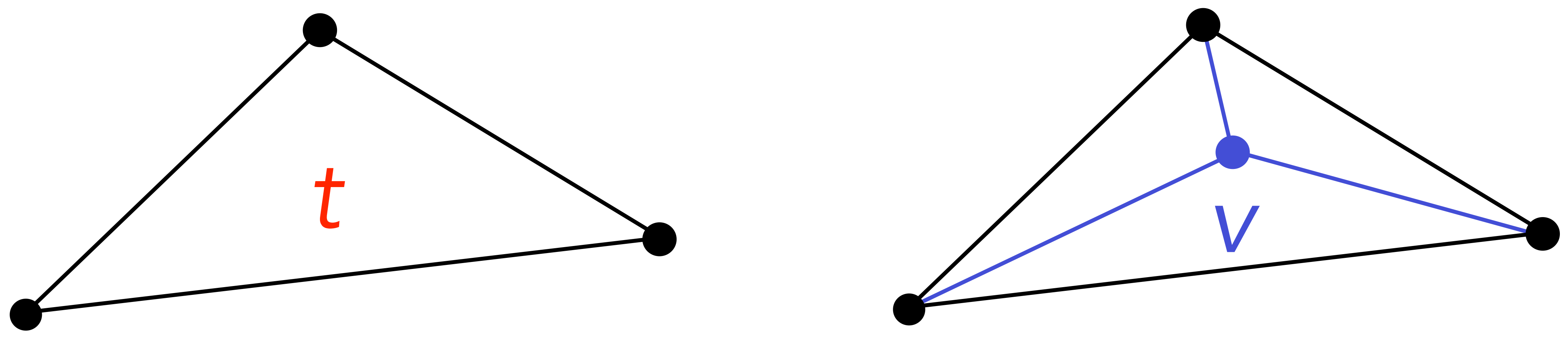
- Specific operators that specific for triangle meshes
- **Refinement operators**: produce a mesh with more vertices/edges/faces
- **Simplification operators**: produce a mesh with less vertices/edges/faces
- Can be implemented on any topological data structure for triangle meshes

Refinement Operators

Skoltech

Skolkovo Institute of Science and Technology

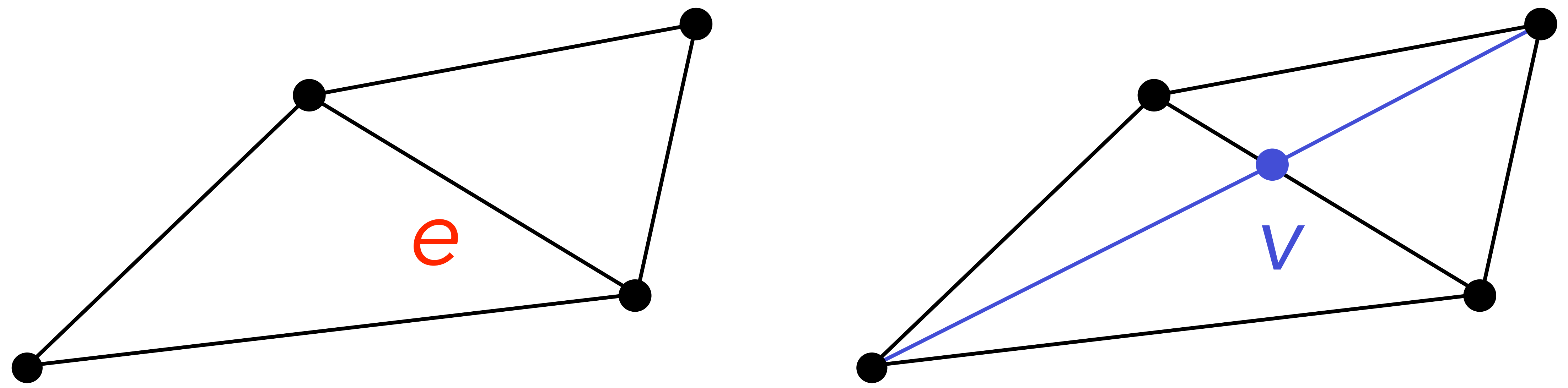
- Triangle split:
  - insert a new vertex *v* in a triangle *t* and connect *v* to the vertices of *t* by splitting it into three triangles



Refinement Operators

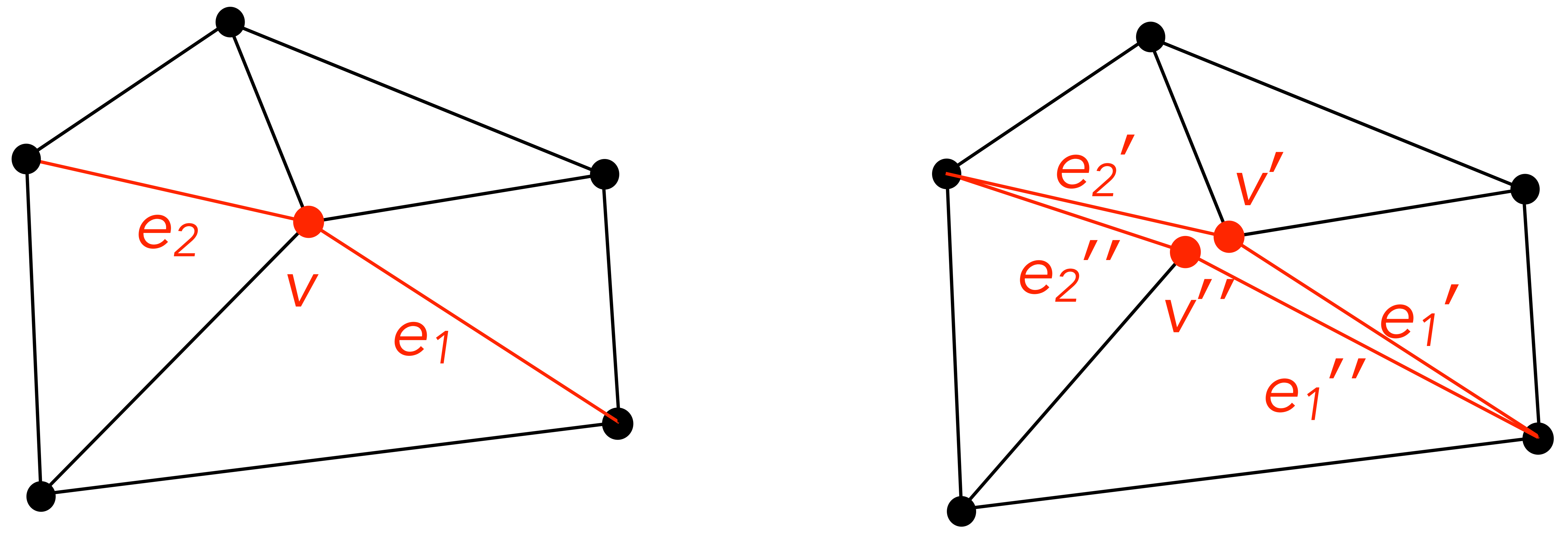
Skoltech
Skolkovo Institute of Science and Technology

- Edge split:
  - insert a new vertex v on an edge e and connect v to the opposite vertices of triangles incident at e by splitting e as well as each such triangle into two



Refinement Operators

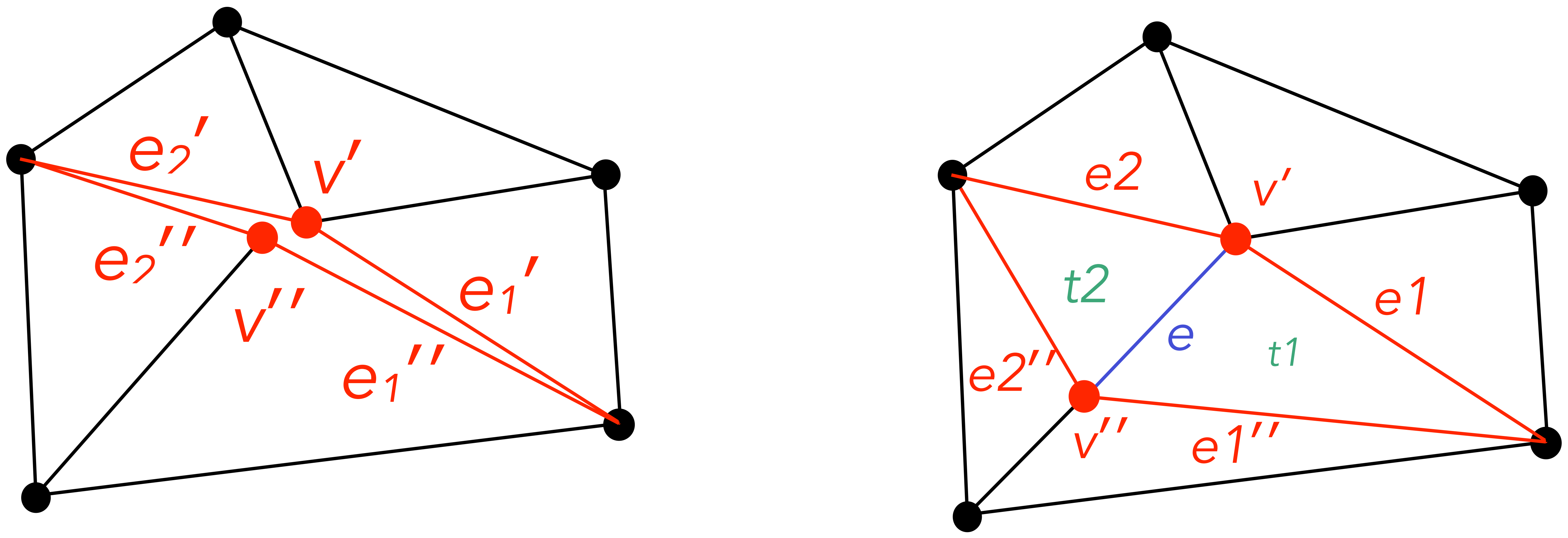
- Vertex split:
    - cut open the mesh along two edges e_1 and e_2 incident at a common vertex v, by duplicating such edges as well as v



Refinement Operators

Skoltech
Skolkovo Institute of Science and Technology

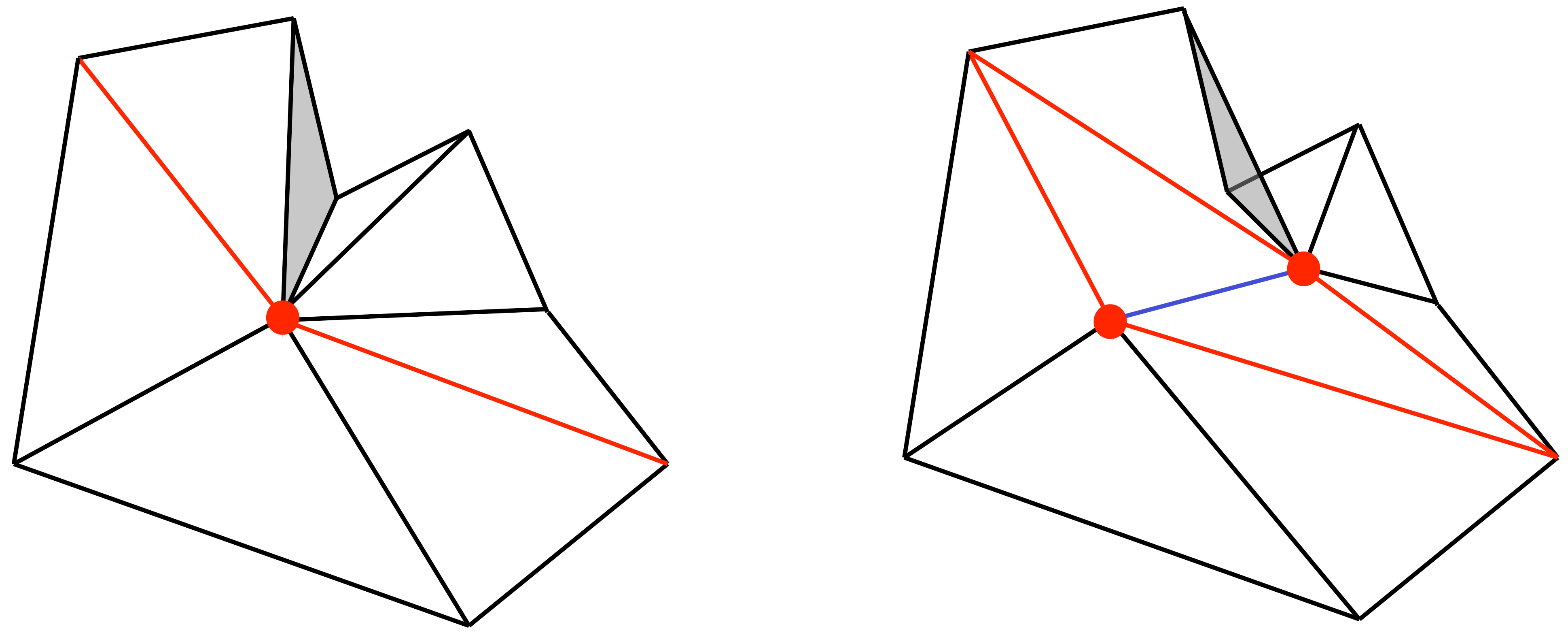
- Vertex split:
  - cut open the mesh along two edges e_1 and e_2 incident at a common vertex v, by duplicating such edges as well as v
  - fill the quadrangular hole with two new triangles and an edge joining the two copies of v



Slide Credit: Daniele Panozzo

Refinement Operators

- Vertex split:
  - possible inconsistencies because of *triangle flip*

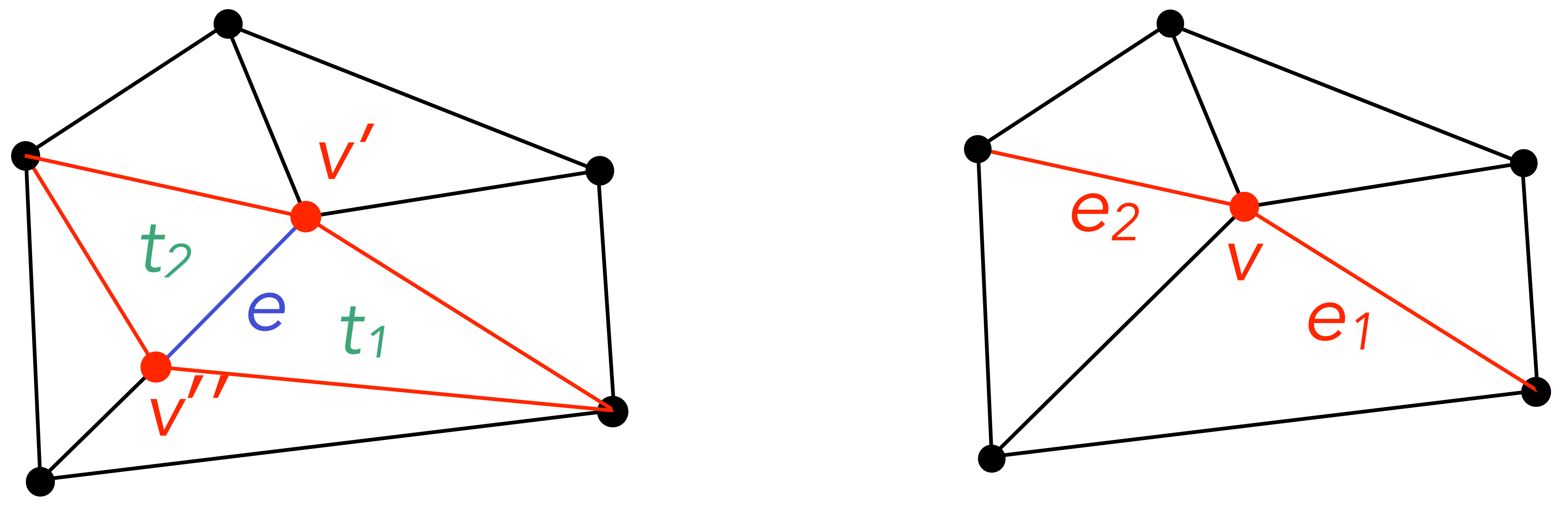


  - flips can be detected by a local test on the orientation of faces: flips changes orientation from clockwise to counter-clockwise and vice-versa

Simplification Operators

Skoltech
Skolkovo Institute of Science and Technology

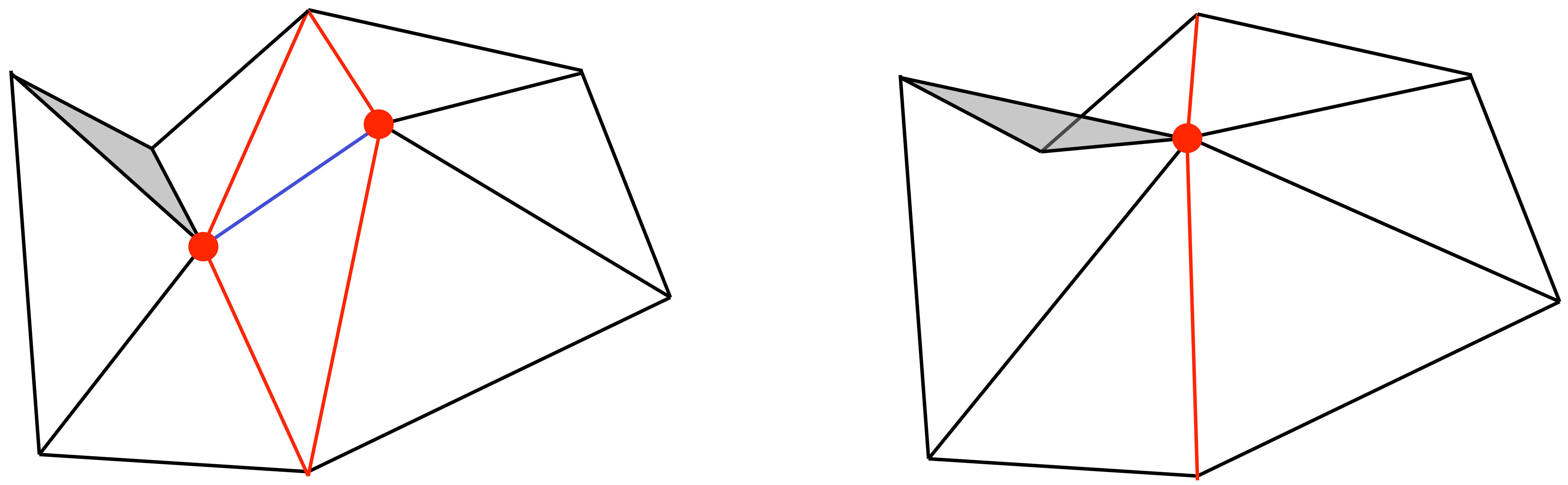
- Edge collapse (reverse of vertex split):
  - collapse an edge *e* to a single point
  - *e* is removed together with its two incident triangles
  - the endpoints of *e* are identified
  - the other edges bounding the deleted triangles are pairwise identified
-



Simplification Operators

Skoltech
Skolkovo Institute of Science and Technology

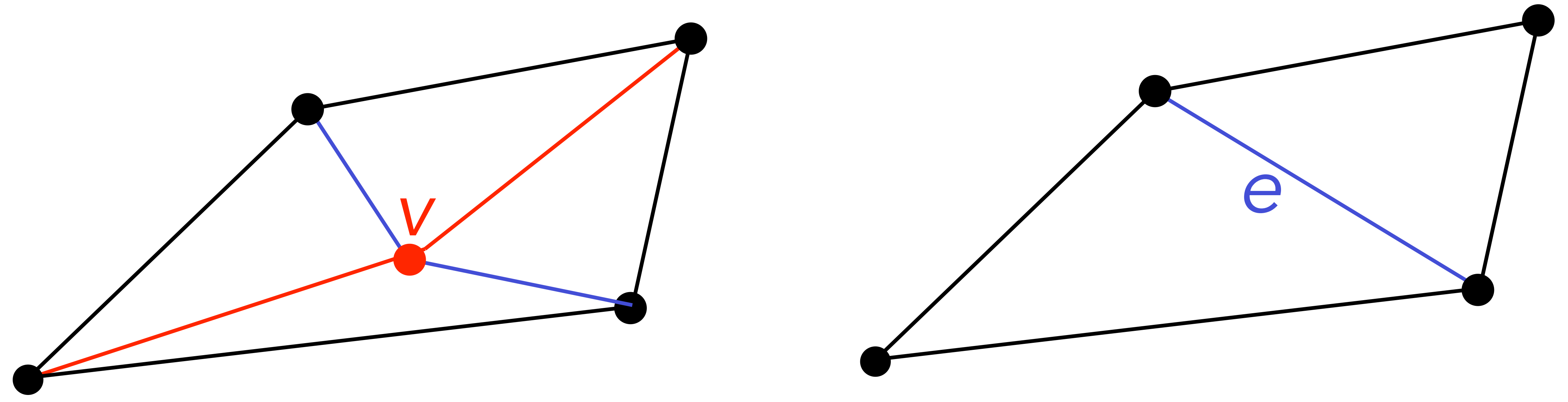
- Edge collapse:
  - possible inconsistencies because of *triangle flip*



  - consistency check analogous to vertex split

Simplification Operators

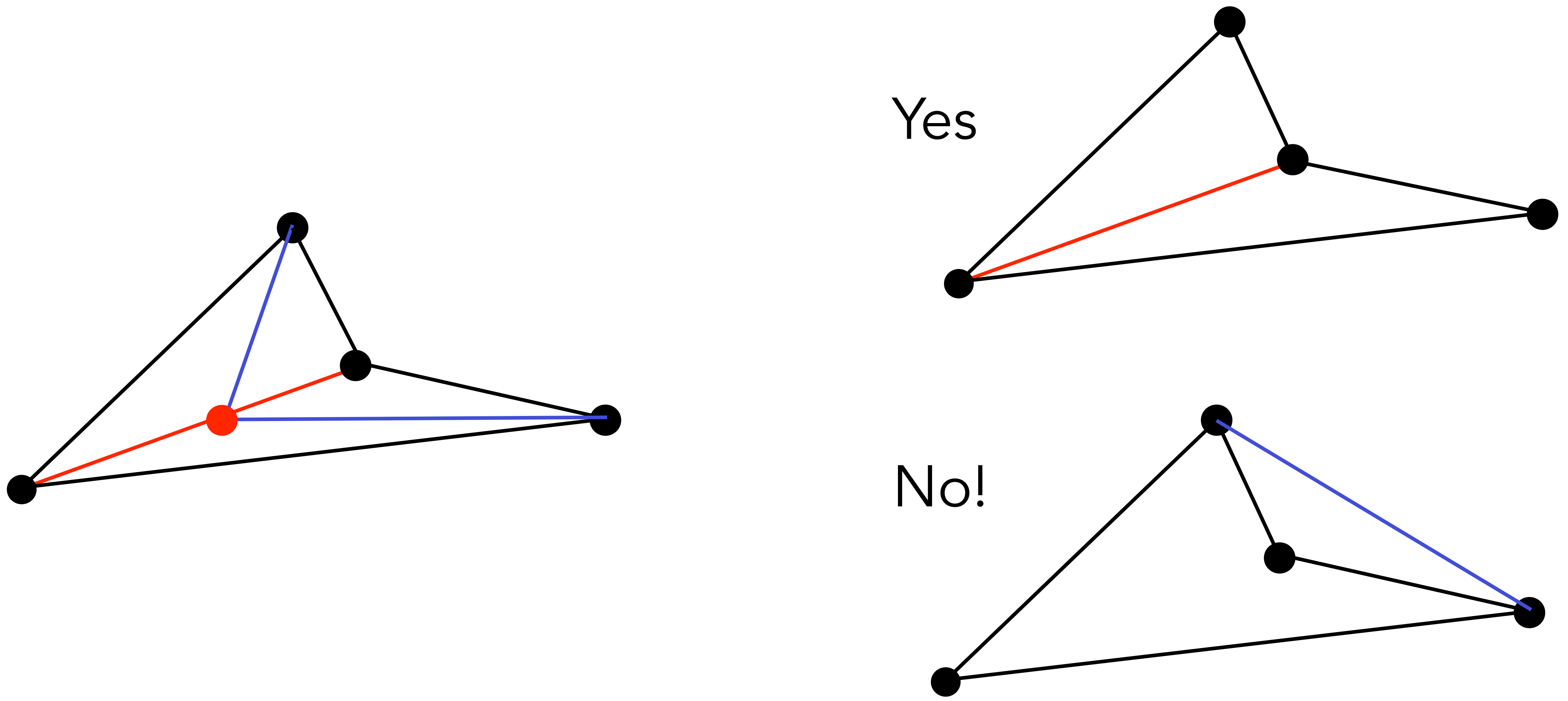
- Edge merge (reverse of edge split):
  - take an internal vertex *v* with valence 4
  - delete *v* together with its incident triangles and edges and fill the hole with two new triangles sharing a new edge *e*



Slide Credit: Daniele Panozzo

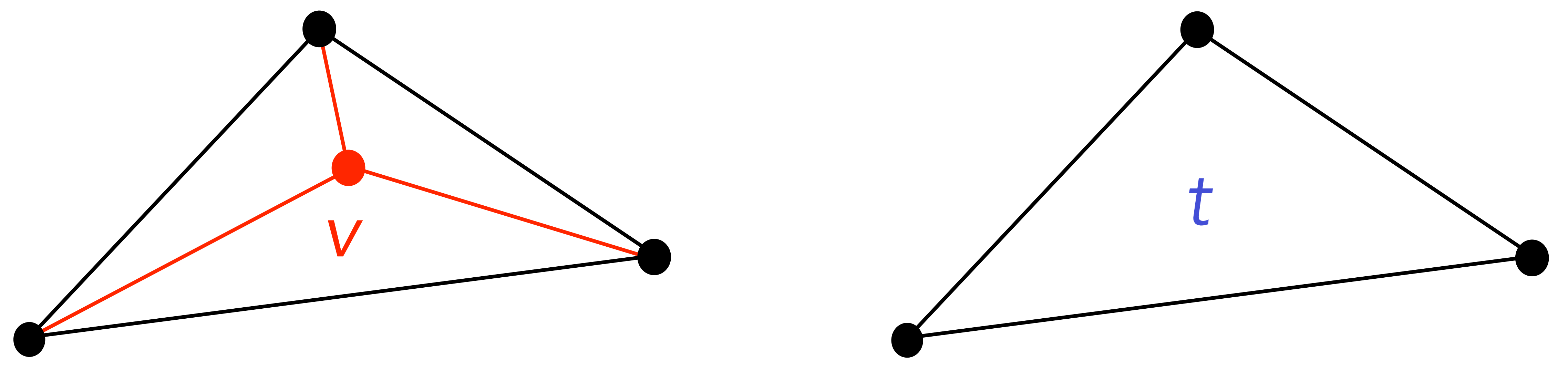
Simplification Operators

- Edge merge:
  - if the hole is not convex, only one diagonal edge can be inserted



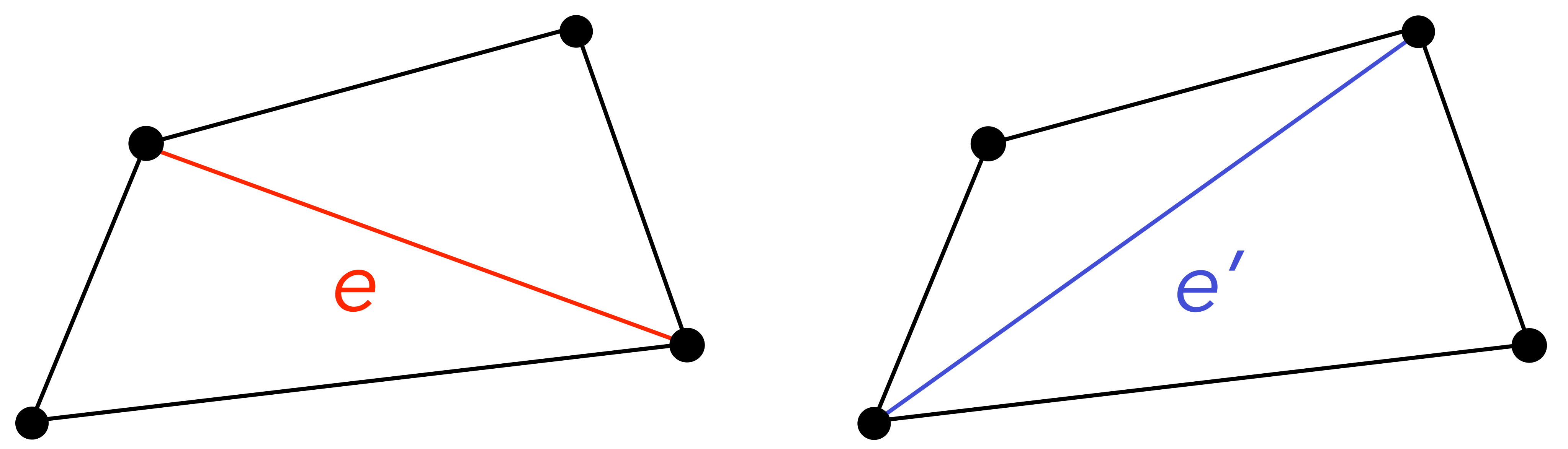
Simplification Operators

- Delete vertex (reverse of triangle split):
  - remove an internal vertex *v* of valence 3 together with its incident triangles and edges and fille the hole with a new triangle



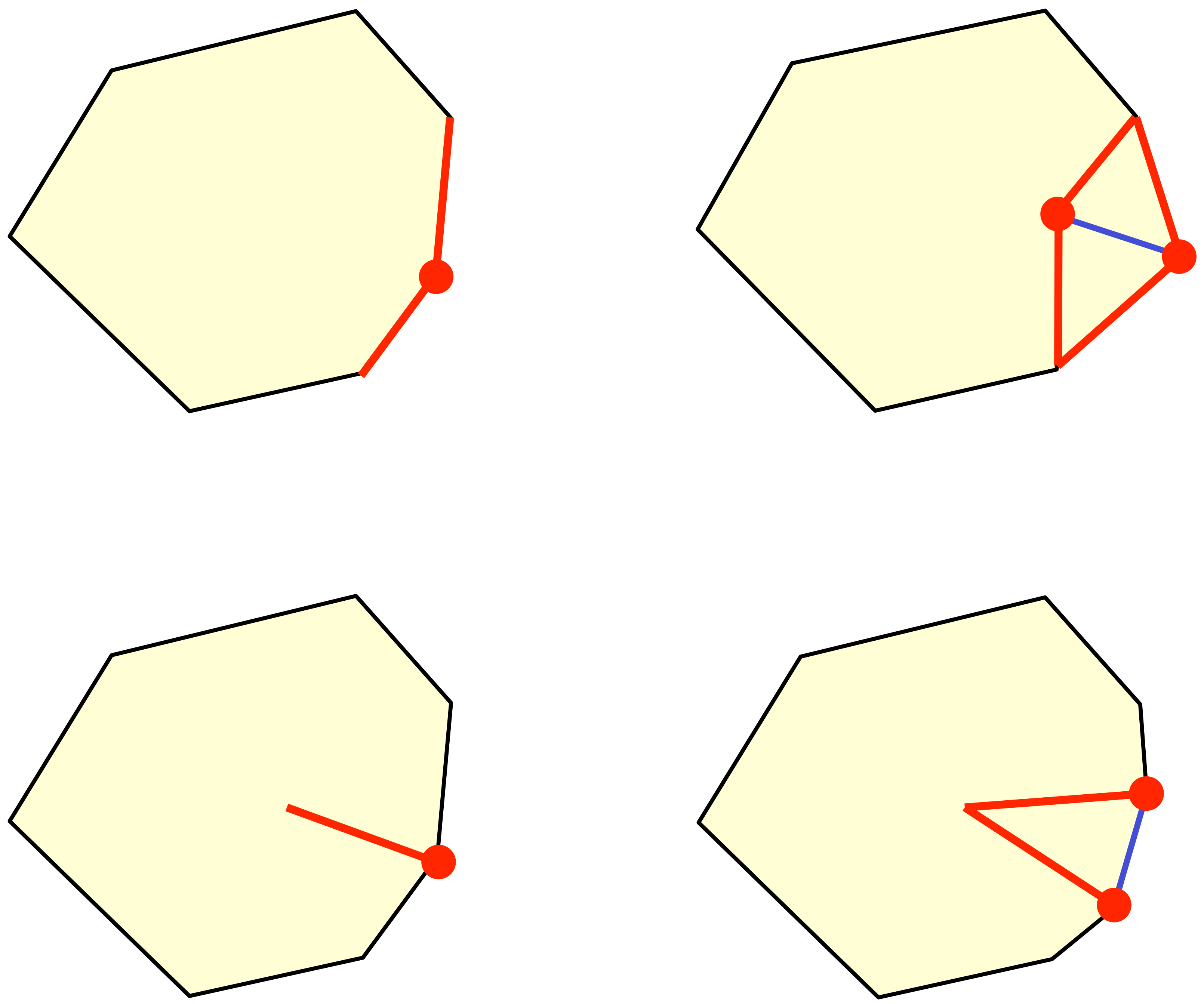
Neutral Operator

- Edge swap:
  - consider an edge *e* such that its two incident triangles form a convex quadrilateral
  - replace *e* with the opposite diagonal of the quadrilateral, rearranging the two incident triangles accordingly



Boundary Cases

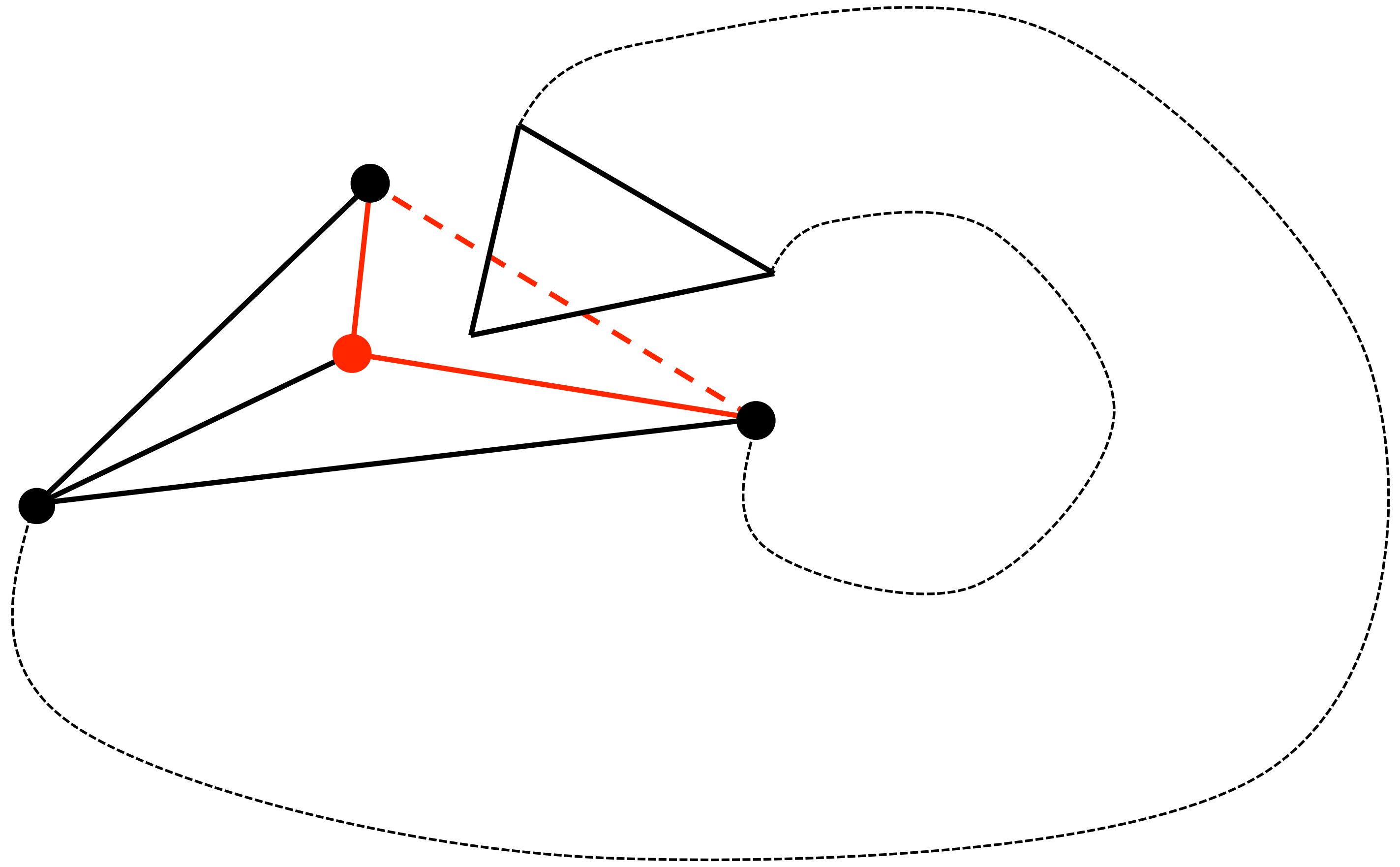
- Vertex split / Edge collapse:



Boundary Cases

Skoltech
Skolkovo Institute of Science and Technology

- Edge merge on a concave boundary may cause self-intersection of the mesh
- It is a global check! intersecting parts may be far on the mesh



- Similar problems with edge collapse on concave boundary and vertex split on convex boundary

Mesh Data Structures

Data Structures

Skoltech
Skolkovo Institute of Science and Technology

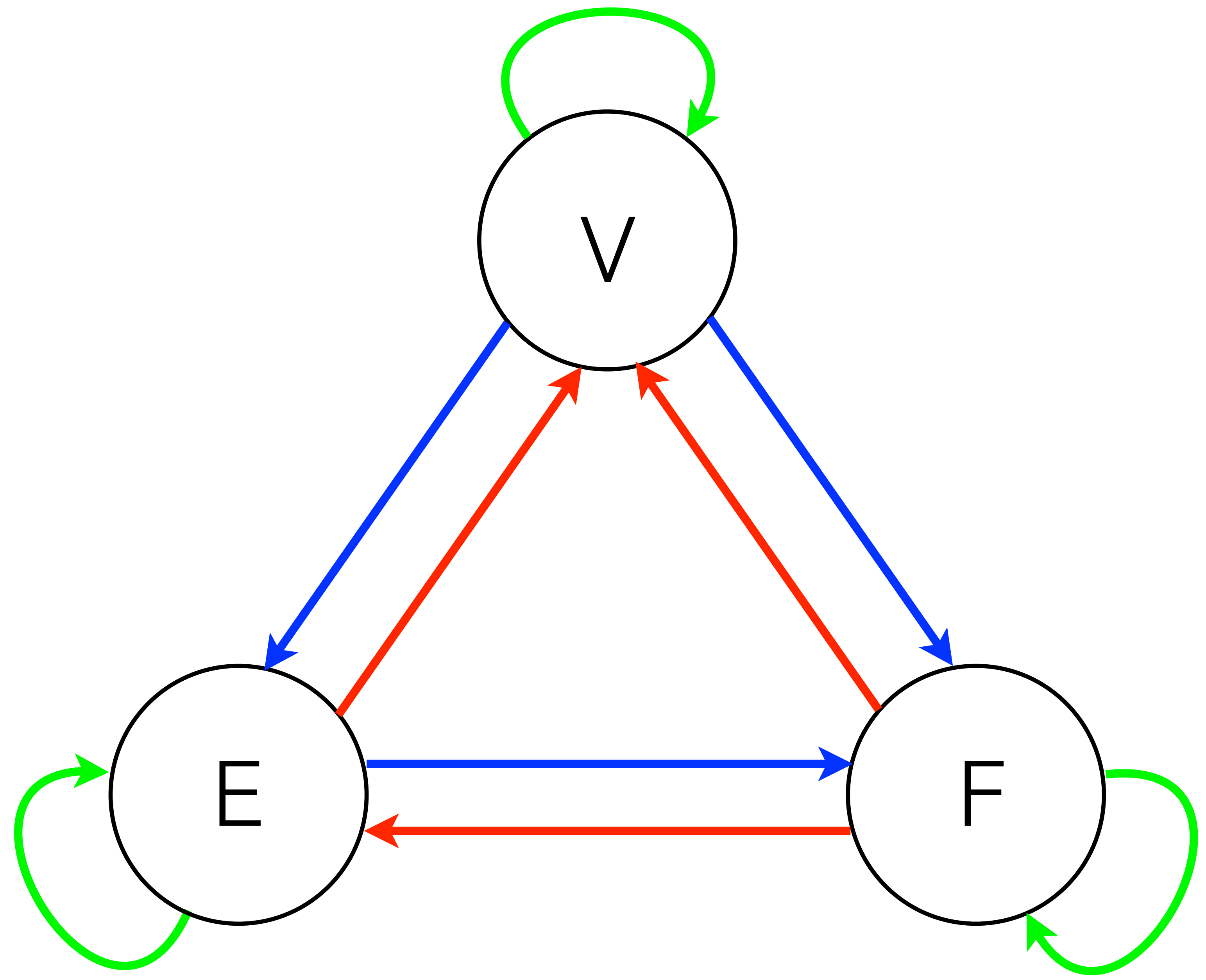
- Storing a mesh:
  - geometry: position of vertices
  - connectivity: edges, faces
  - topology: topological relations
- Compactness vs efficiency trade-off: data structures should be compact, while efficiently supporting time-critical operations
  - storage requirements
  - traversal operations
  - update operations

Data Structures

Skoltech

Skolkovo Institute of Science and Technology

- Information to store/retrieve:
  - Entities: vertices, edges, faces
  - Relations: VV, VE, VF, EV, EE, EF, FV, FE, FF
  - Additional properties attached to any entity (Positions, Normals, Colors)
- The data structure to use is application-dependent



Polygon Soup

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}

- a.k.a. Face set - STL files
- Just store a list of faces
- For each face, store positions of its vertices
  - For general polygonal mesh, the number of vertices of each face must also be stored
  - Number implicit for triangle meshes
- For a triangle mesh: 36 bytes/face ≈ 72 bytes/vertex
- No connectivity! just a collection of polygons
- Streaming structure: no need to store it all in memory to render the mesh

Indexed Structure

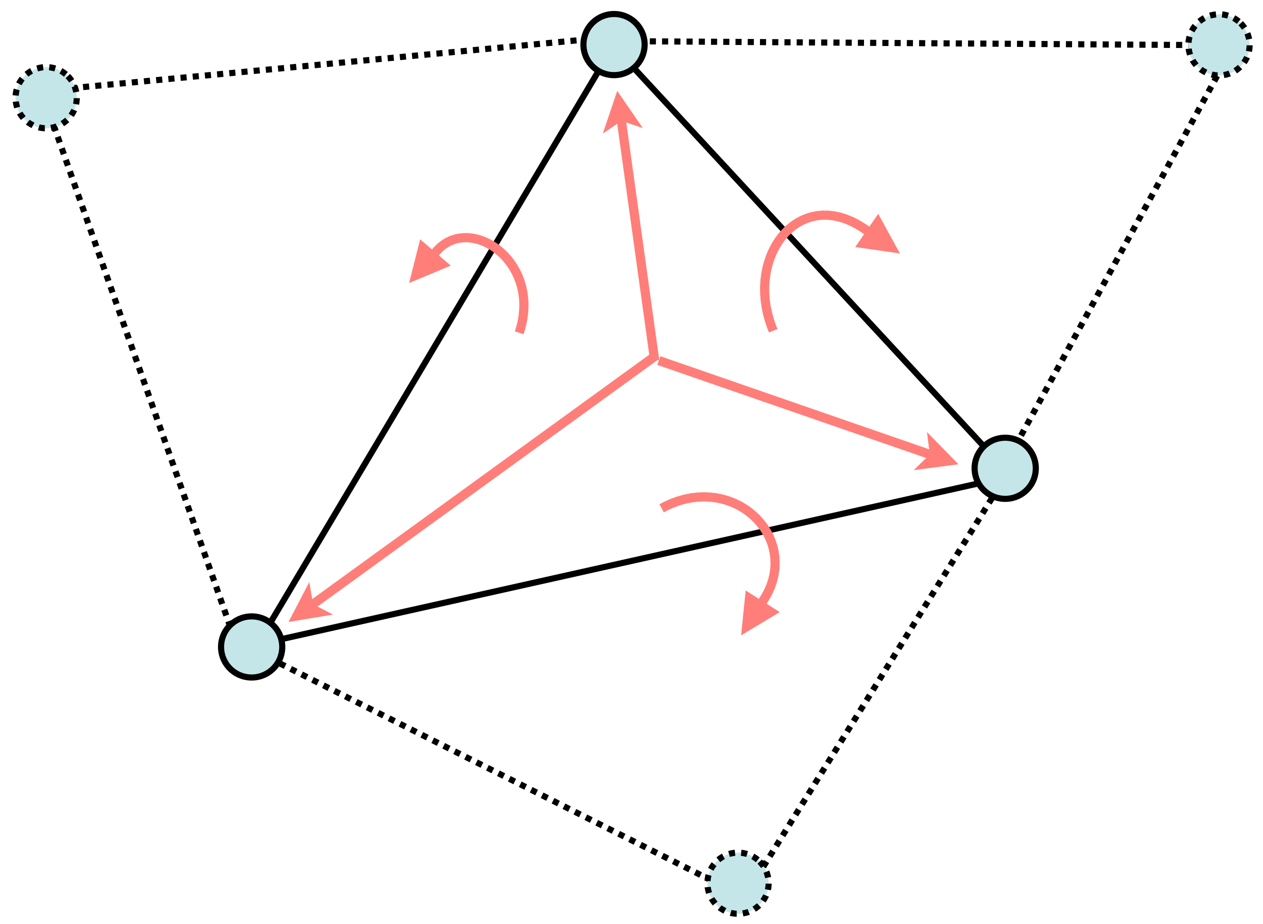
- Avoid replication of vertices - OBJ, OFF, PLY files
- Maintain a vector of vertices and a vector of faces
  - For each vertex: position
  - For each face: references to positions of its vertices (FV) in the vector
  - in general, the number of vertices of each face must also be stored
  - number implicit for triangle meshes
- For a triangle mesh: 12 bytes/vertex +12 bytes/face ≈ 36 bytes/vertex
- Encodes connectivity through the FV relation
- Main memory structure: need to store in memory at least the whole list of vertices

Vertices
x_1 y_1 z_1
...
x_V y_V z_V

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

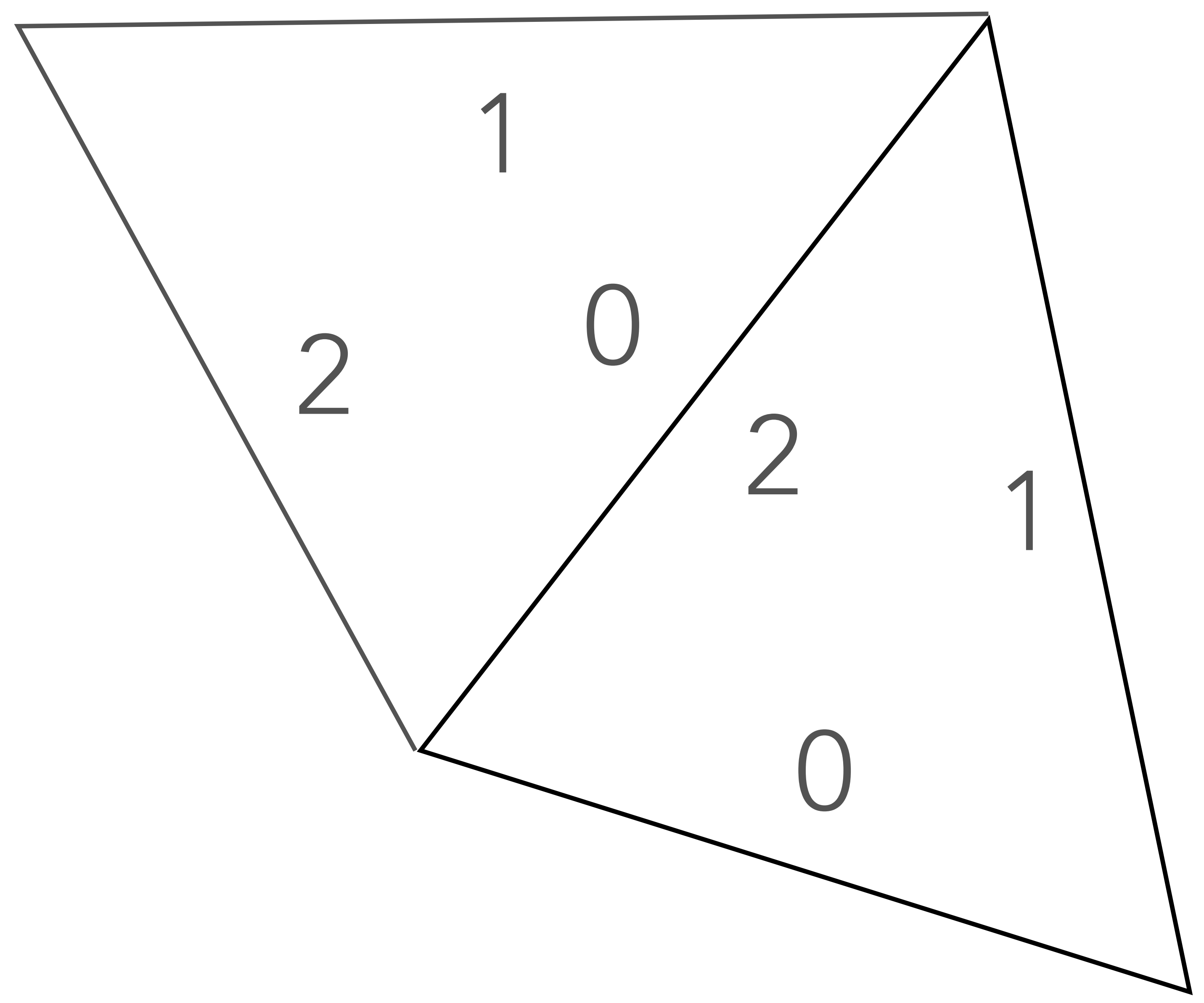
Indexed Structure with Adjacencies

- Just for triangle meshes
- Extends indexed structure with some topological relations:
  - For each vertex:
    - position
    - one reference to an incident triangle (VF* relation)
  - For each face:
    - references to its three vertices (FV)
    - references to its three adjacent triangles (FF)



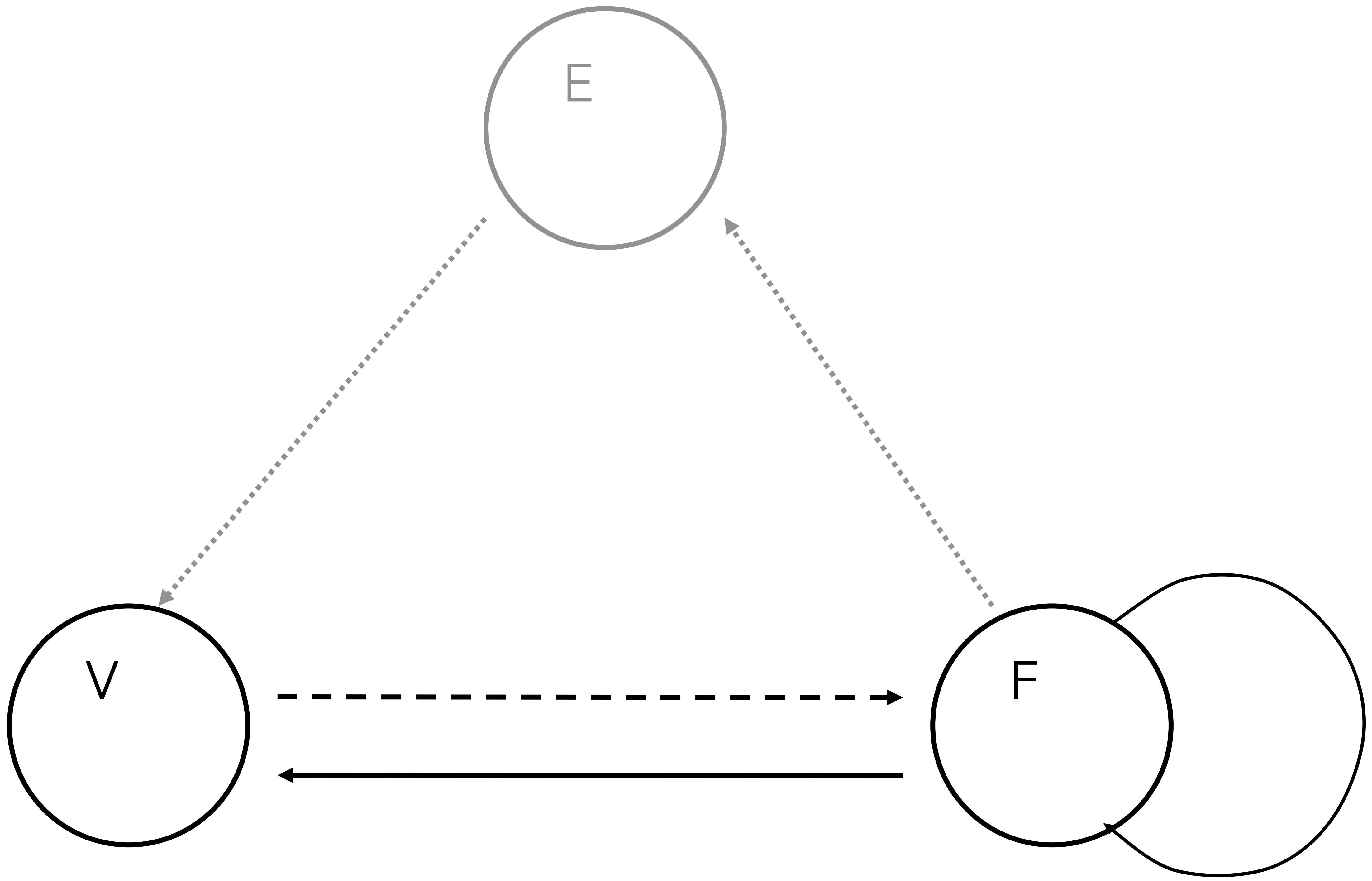
Indexed Structure with Adjacencies

- No explicit edges!
  - implicitly defined as pairs of vertices (unique)
  - or as pairs (f,i) with f triangle and i index in 0,1,2 (not unique!)
- Attributes for edges require care, especially when modifying the mesh with editing operators



Indexed Structure with Adjacencies

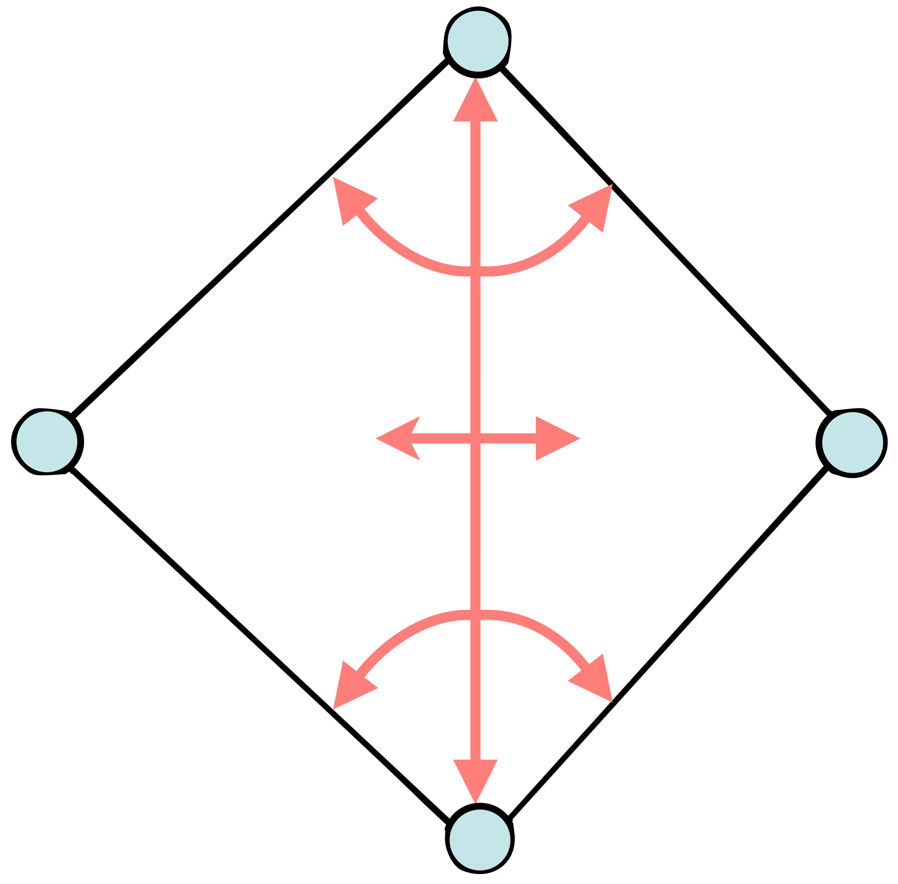
- Evaluation of topological relations:
  - FV, FF: encoded - optimal
  - VF = VF* + FF + FV optimal
  - VV analogous to VF
- Relations involving edges are either implicit or they can be evaluated in optimal time by the (f,i) encoding

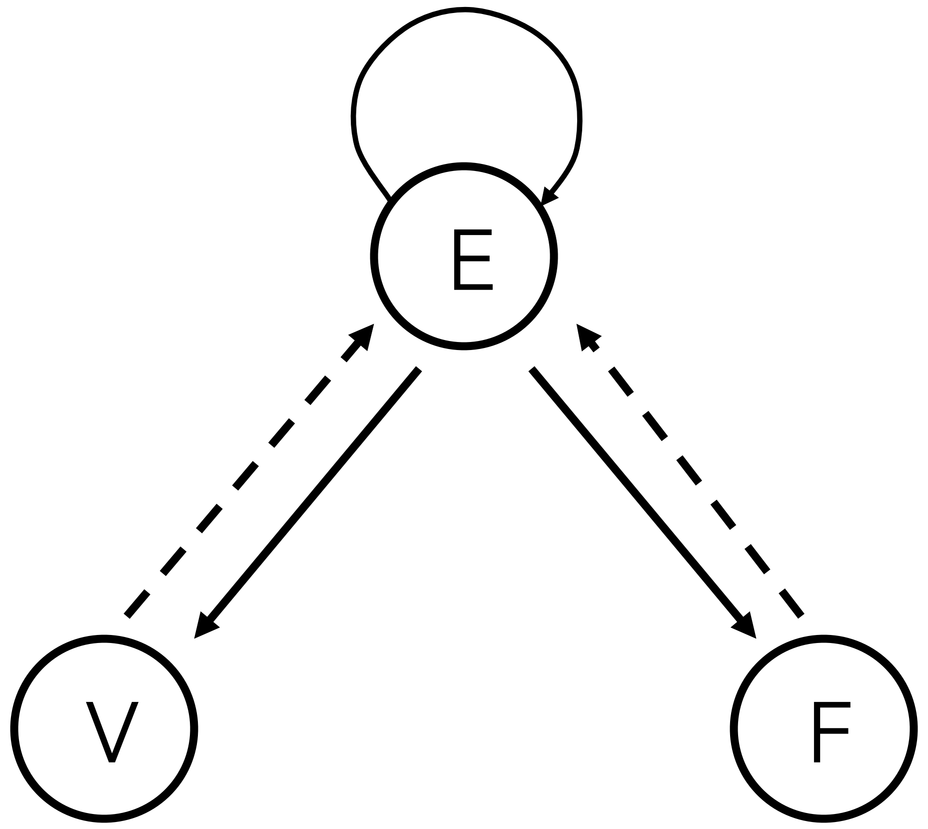


Winged Edge Data Structure

Skoltech
Skolkovo Institute of Science and Technology

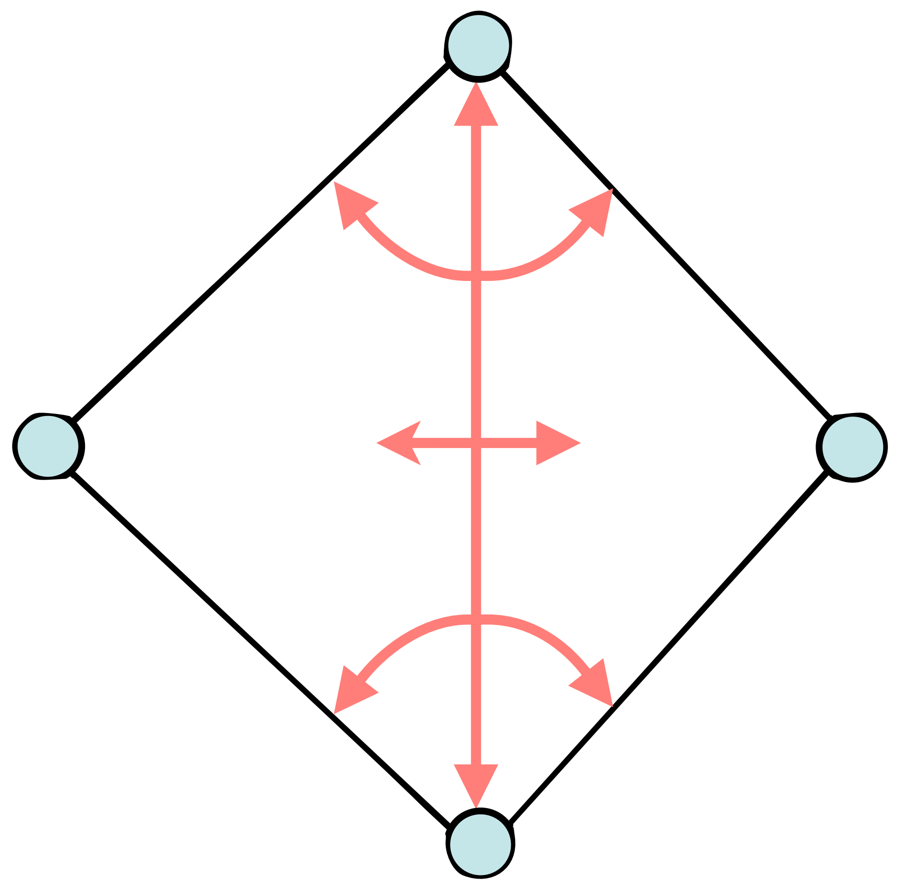
- Topological structure for general polygonal meshes
  - For each vertex:
    - position
    - one reference to an incident edge (VE*)
  - For each edge:
    - references to its two vertices (EV)
    - references to its two incident faces (EF)
    - references to its four adjacent edges (EE)
  - For each face:
    - one reference to an incident edge (FE*)

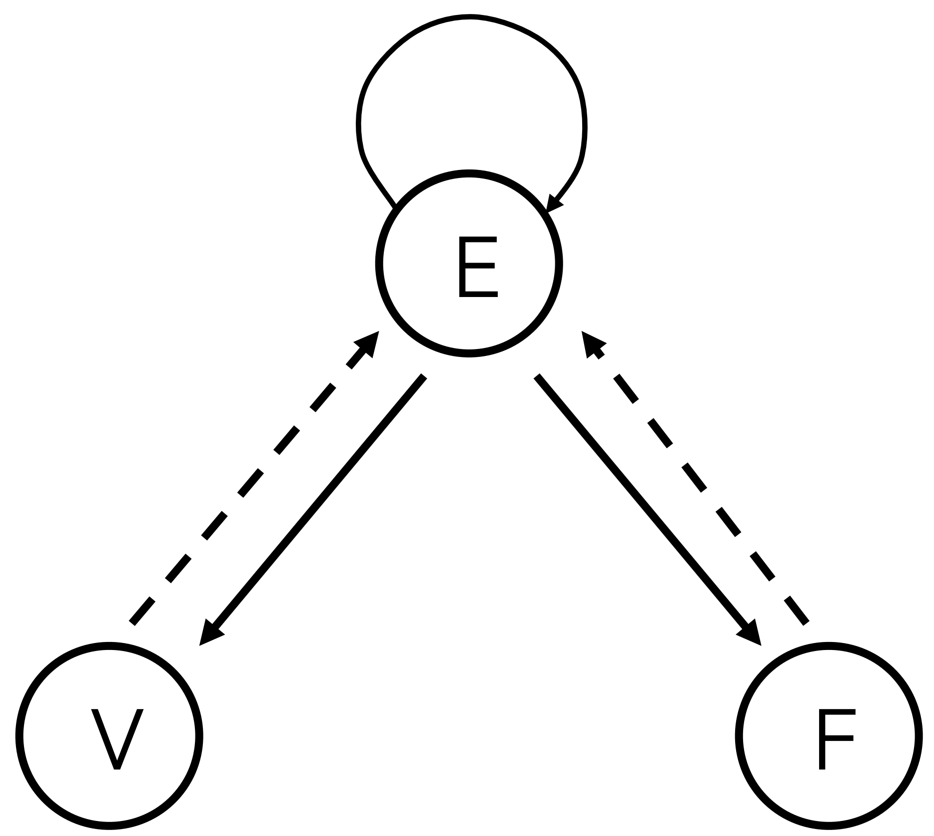




Winged Edge Data Structure

- 120 bytes/vertex
- all entities are represented explicitly
- all topological relations can be retrieved in optimal time
- ambiguous orientation of edges:
  - does EV(e) return (v_i,v_j) or (v_j,v_i)?

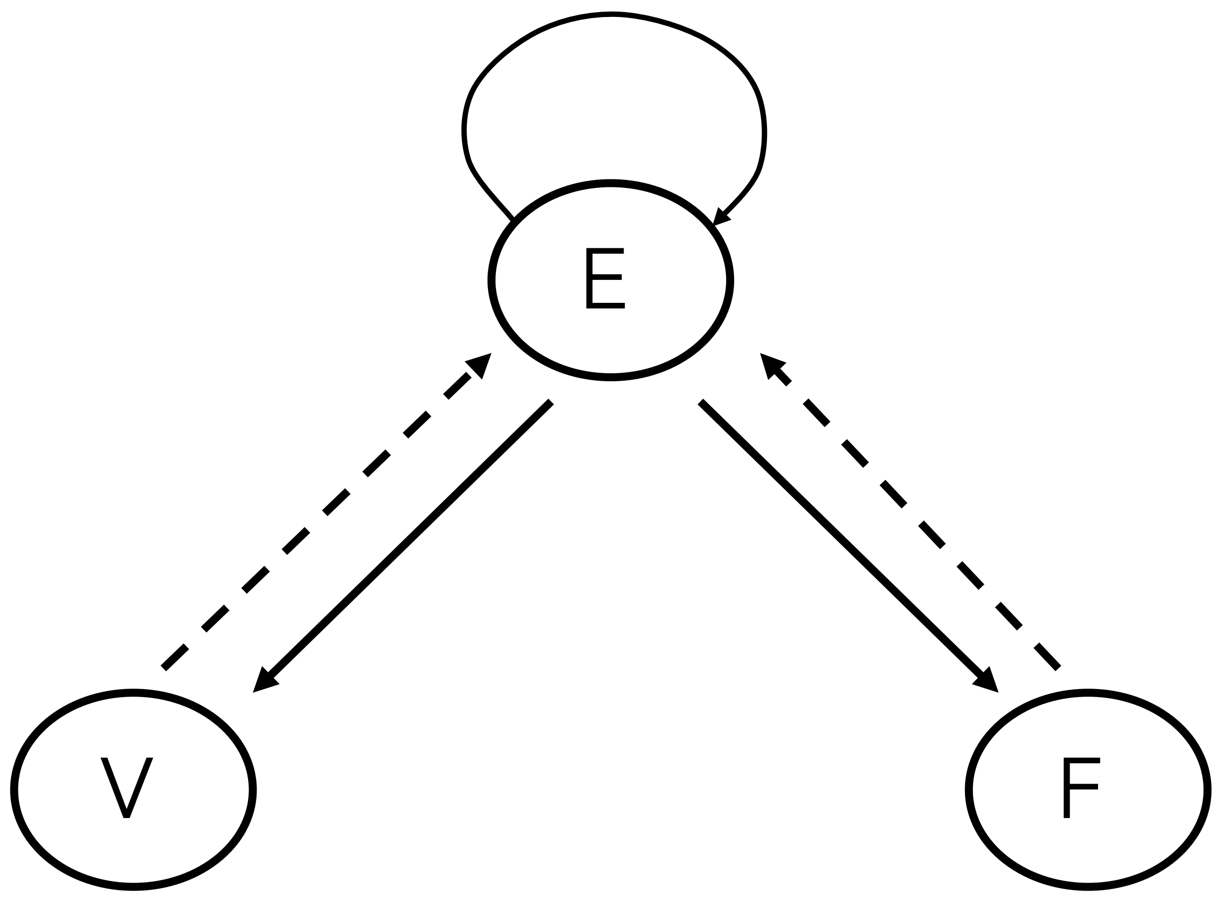


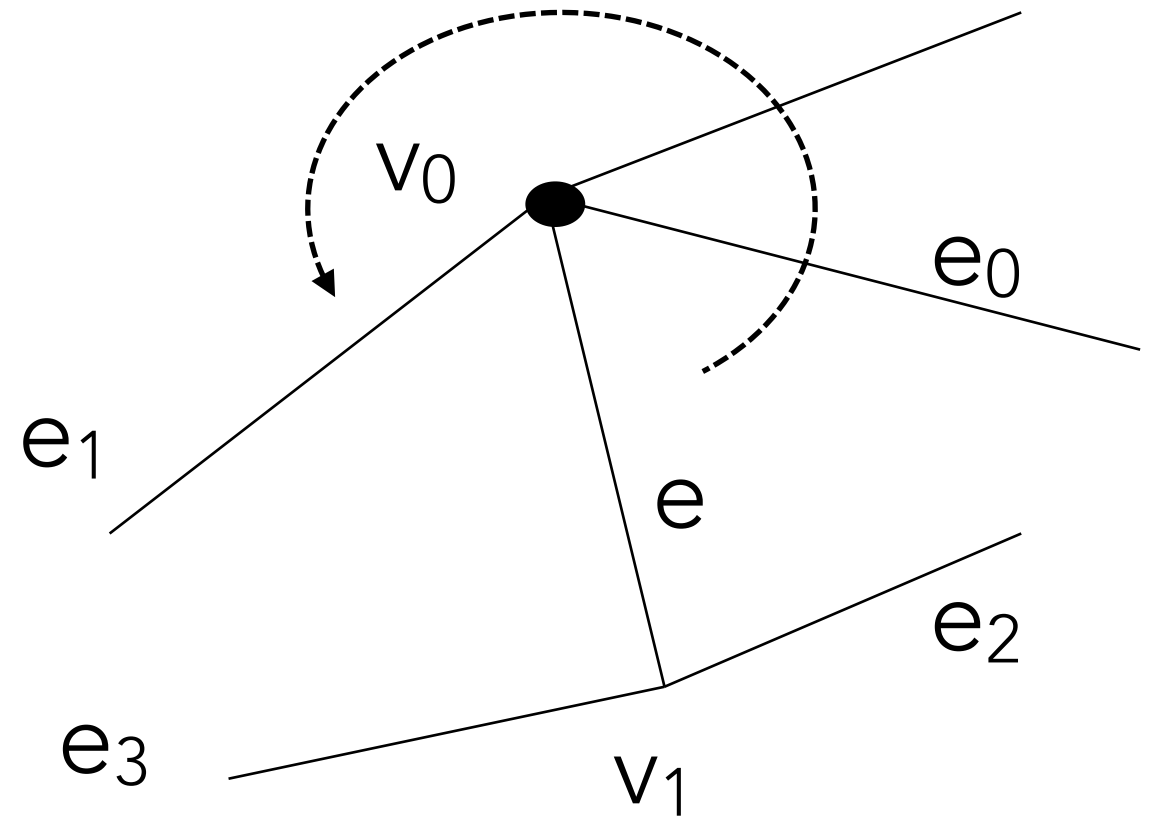


Winged Edge Data Structure

Skoltech
Skolkovo Institute of Science and Technology

- Example: evaluation of VE(v):
  - get first edge e = VE*(v)
  - evaluate (e_0,e_1,e_2,e_3) = EE(e)
  - evaluate (v_0,v_1) = EV(e)
  - in counter-clockwise order, set next edge at either e_0 or e_3, depending on whether $v=v_0$ or $v=v_1$
  - cycle until hitting e

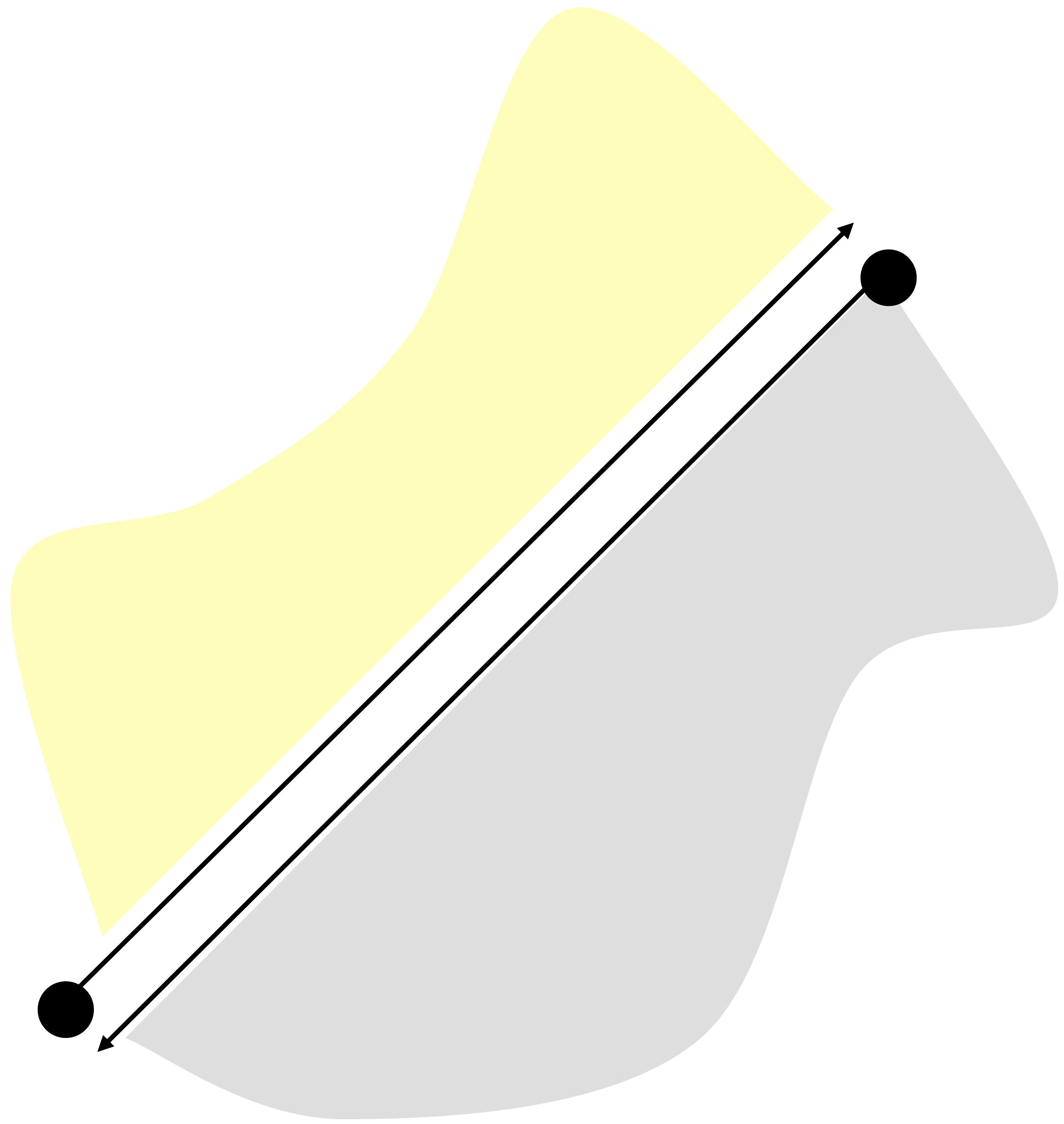




Slide Credit: Daniele Panozzo

Half-Edge Data Structure

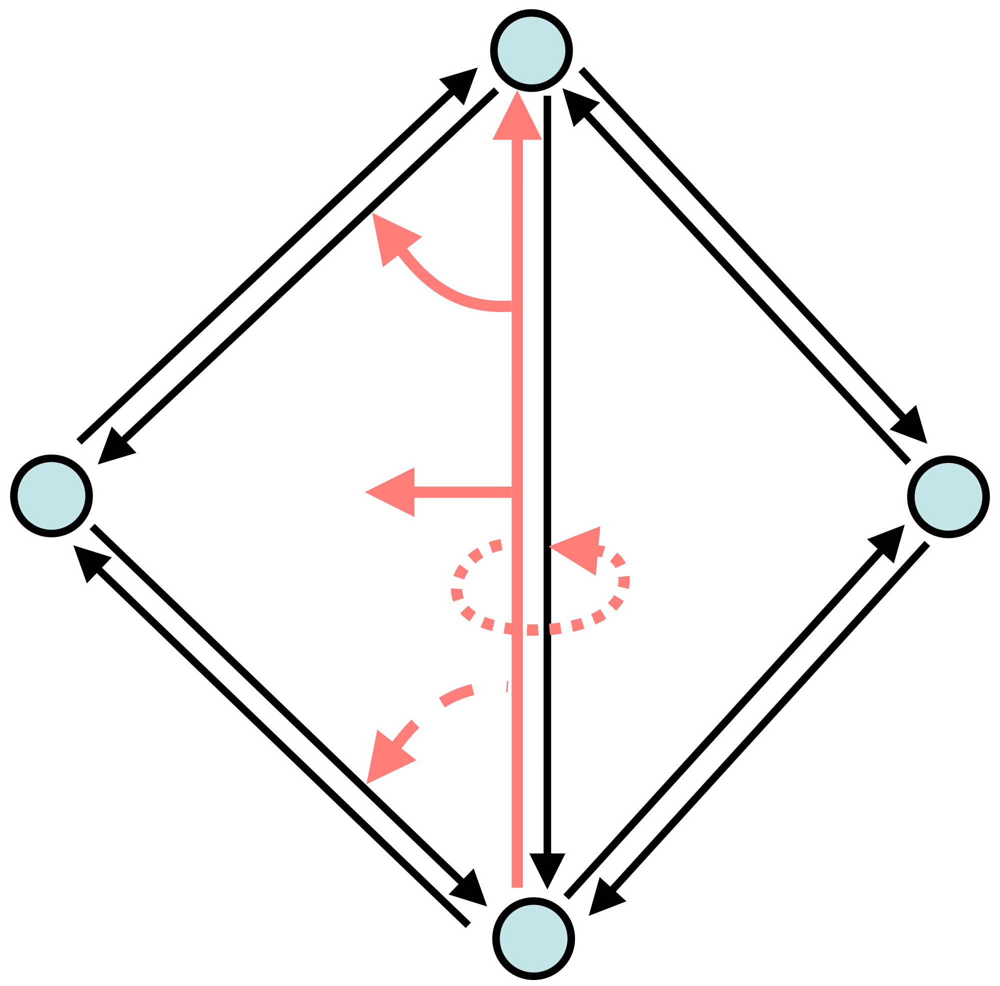
- Half-edge: each edge is duplicated by also considering its orientation
- An edge corresponds to a pair of sibling half-edges with opposite orientations
- Each half-edge stores half topological information concerning the edge



Half-Edge Data Structure

Skoltech
Skolkovo Institute of Science and Technology

- For each vertex:
  - position
  - one reference to an incident half-edge
- For each half-edge:
  - reference to one its two vertices (origin)
  - references to one of its two incident faces (face on its left)
  - references to: next/previous half-edge on the same face, sibling half-edge
- For each face:
  - one reference to an incident half-edge (FE*)



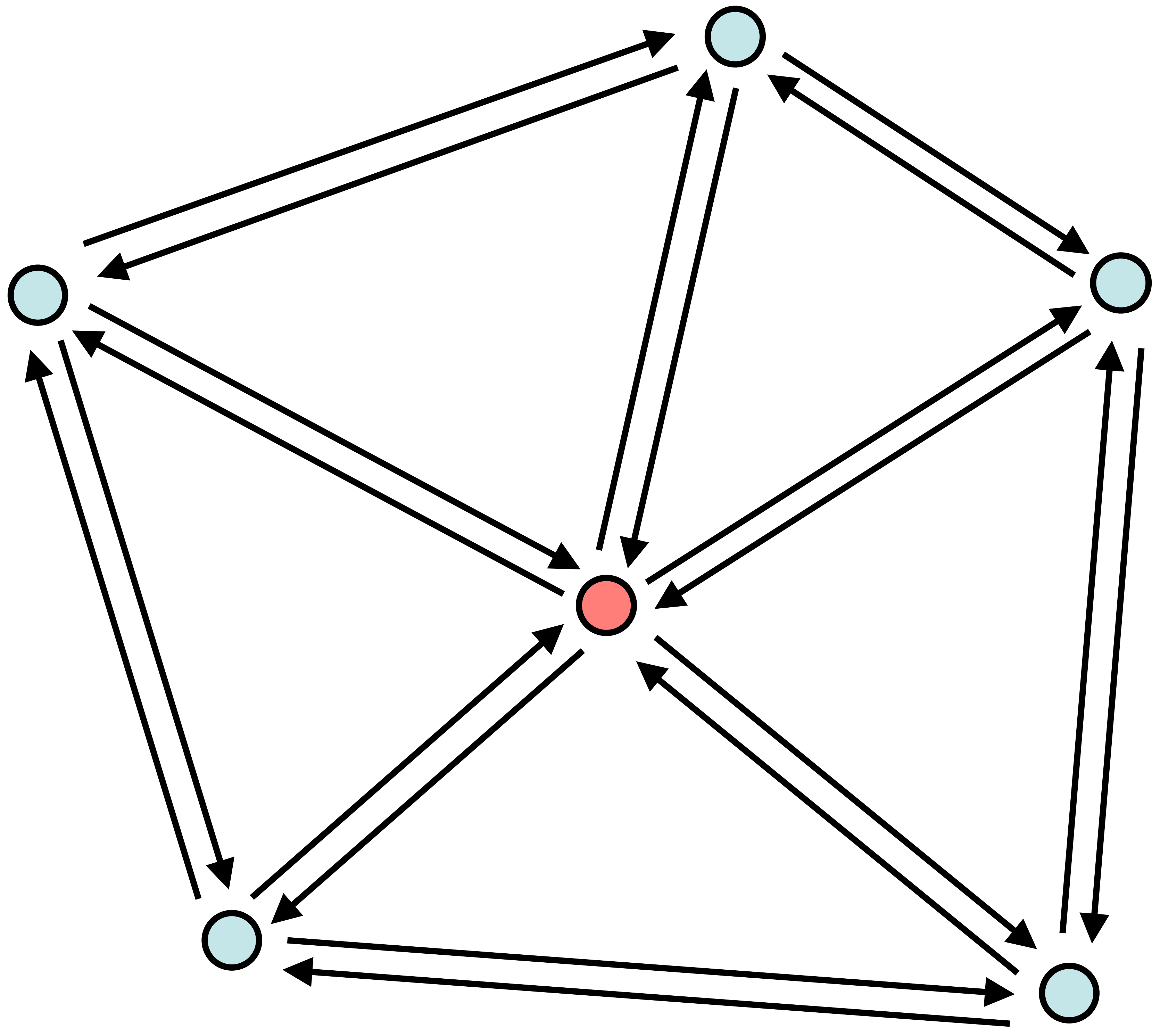
Half-Edge Data Structure

- 96-144 bytes/vertex depending on number of references to adjacent edges
  - reference to sibling half-edge can be avoided by storing siblings at consecutive entries of a vector
  - for triangle meshes, just one reference to either next or previous half-edge is sufficient
- Efficient traversal and update operations
- Attributes for edges must be stored separately

Half-Edge Data Structure

- One-ring traversal (V* relations):

  1.start at vertex

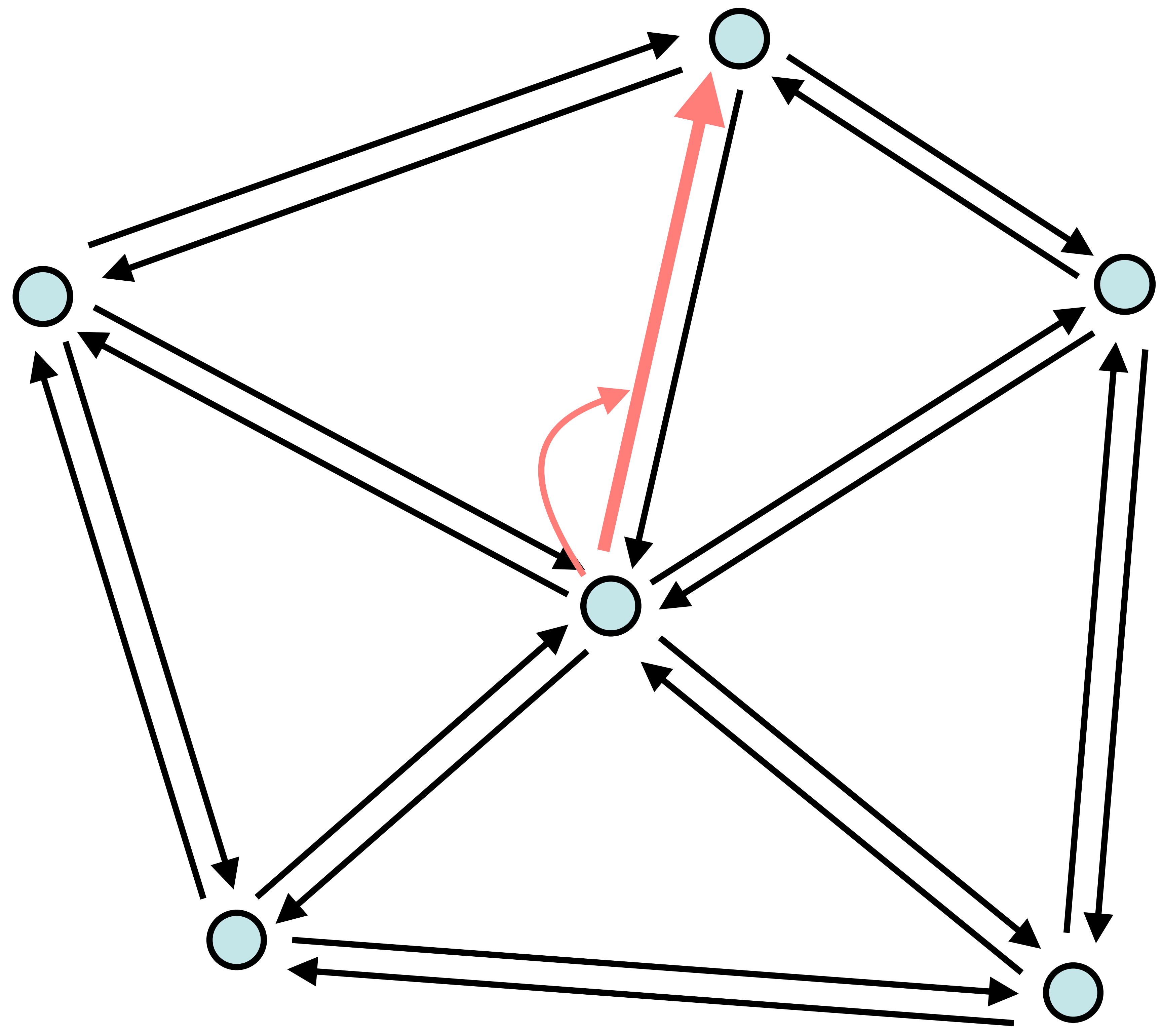


Half-Edge Data Structure

- One-ring traversal (V^* relations):

  1.start at vertex

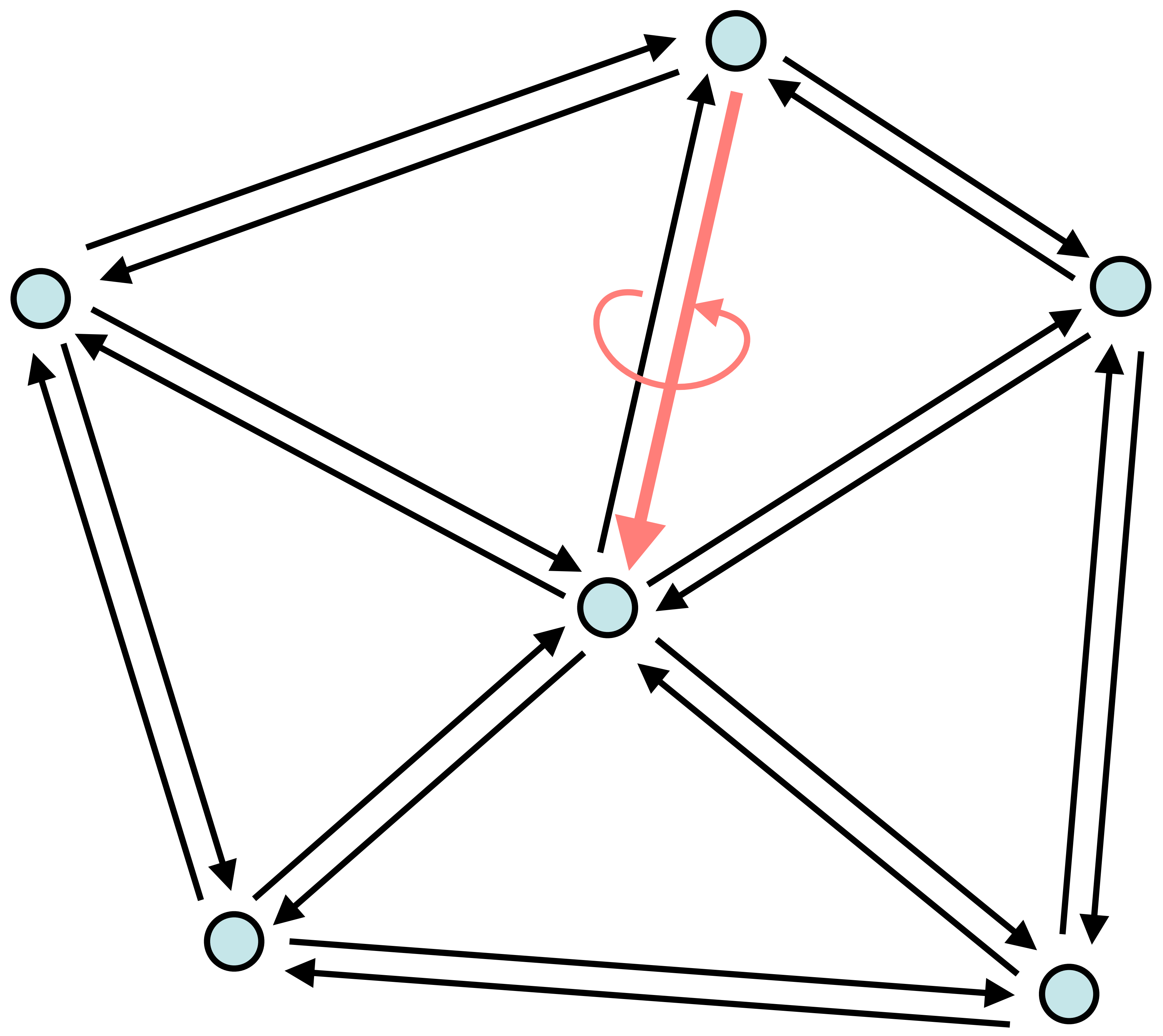
  2.outgoing half-edge



Skoltech
Skolkovo Institute of Science and Technology

Half-Edge Data Structure

- One-ring traversal (V* relations):
  1. start at vertex
  2. outgoing half-edge
  3. opposite half-edge



Half-Edge Data Structure

Skoltech
Skolkovo Institute of Science and Technology

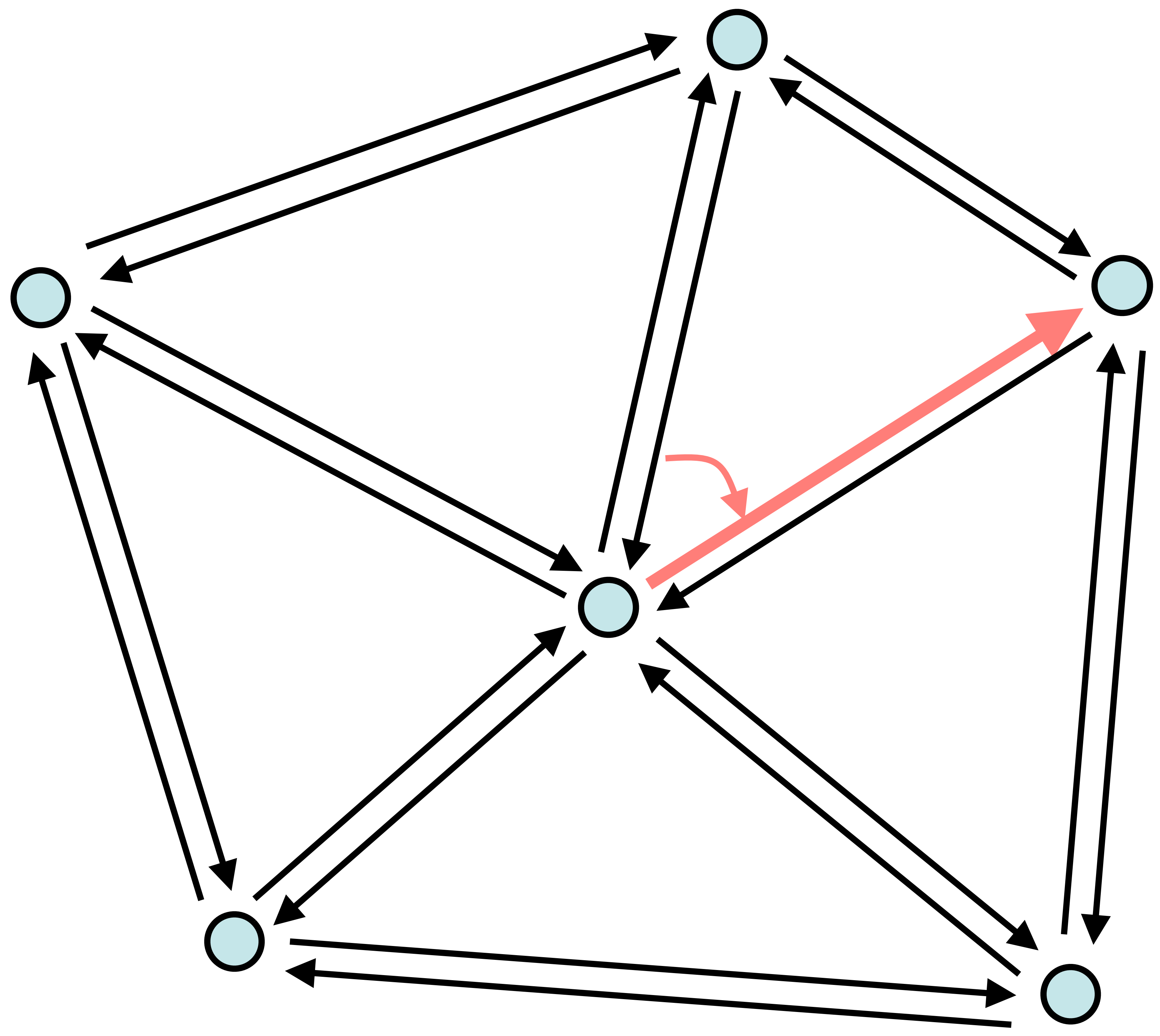
- One-ring traversal (V* relations):

  1. start at vertex

  2. outgoing half-edge

  3. opposite half-edge

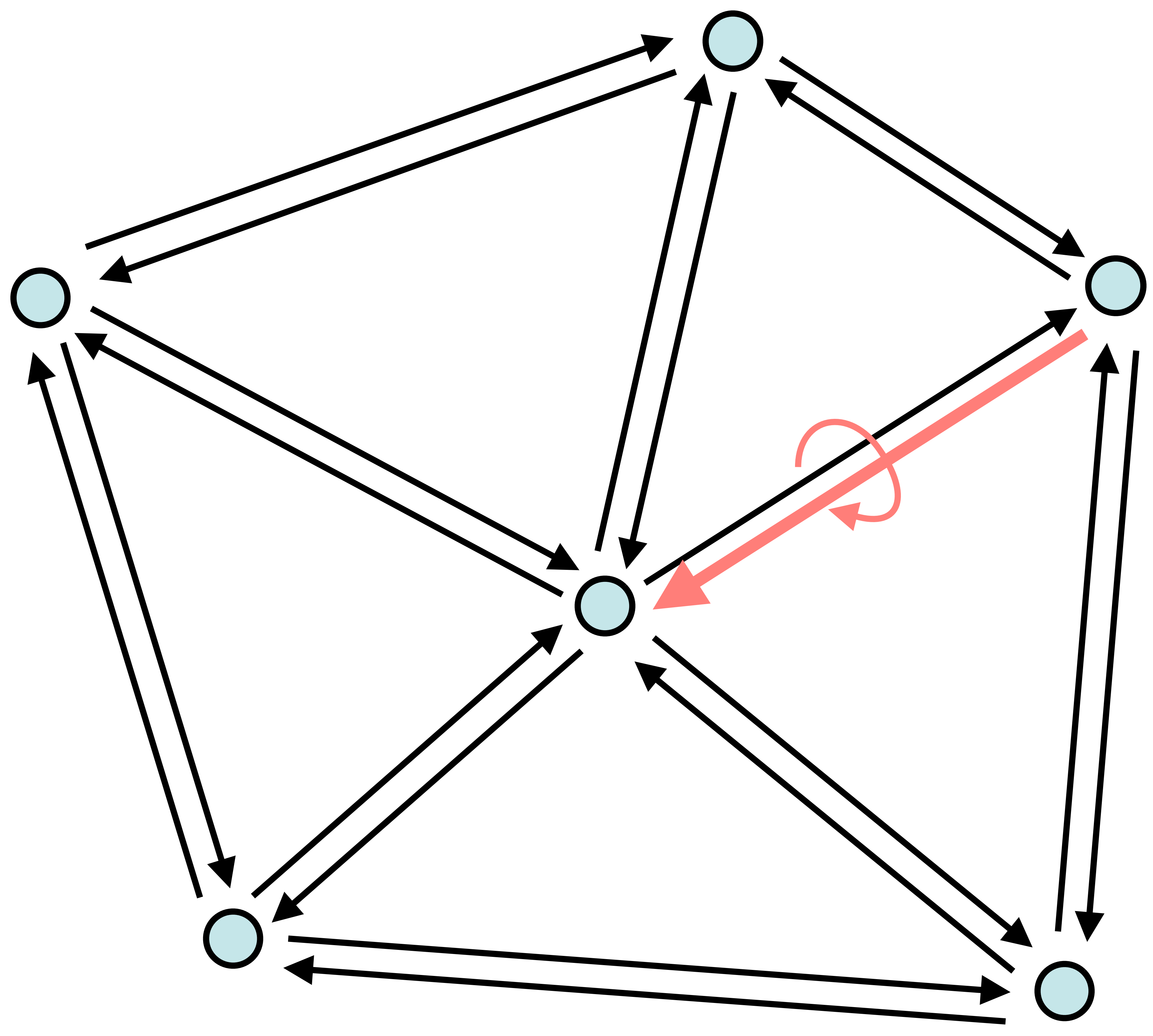
  4. next half-edge



Half-Edge Data Structure

Skoltech
Skolkovo Institute of Science and Technology

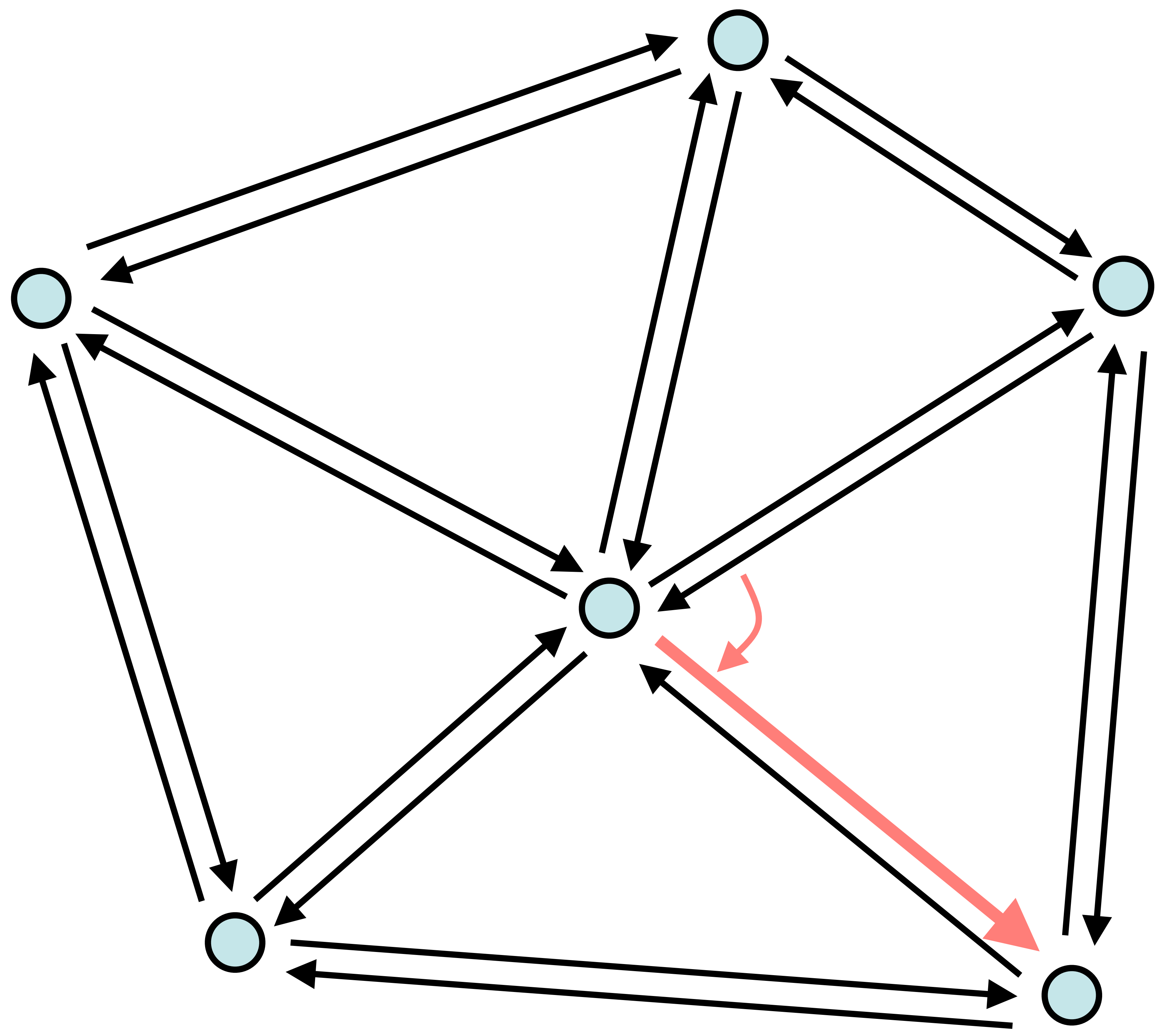
- One-ring traversal (V* relations):
  1. start at vertex
  2. outgoing half-edge
  3. opposite half-edge
  4. next half-edge
  5. opposite



Half-Edge Data Structure

Skoltech
Skolkovo Institute of Science and Technology

- One-ring traversal (V* relations):
  1. start at vertex
  2. outgoing half-edge
  3. opposite half-edge
  4. next half-edge
  5. opposite
  6. next.....



C++ libraries

Skoltech
Skolkovo Institute of Science and Technology

- Indexed based:
  - libigl (igl.ethz.ch/projects/libigl/):
    - light mesh representation compatible with numerical software (e.g., Eigen, Matlab)
    - topological relations are computed on-the-fly and can be stored for later use
    - several geometry processing algorithms algorithms
    - free, MPL2 licence

C++ libraries

- Adjacency based:
  - VCGlib (vcg.sourceforge.net):
    - optimized for triangle and tetrahedral meshes
    - extensions with half-edge for more general meshes
    - several geometry processing algorithms and spatial data structures
    - free, LGPL licence

C++ libraries

Skolkovo Institute of Science and Technology

- Half-edge based:
  - CGAL (www.cgal.org):
    - rich, complex
    - computational geometry algorithms
    - free for non-commercial use
  - OpenMesh (www.openmesh.org):
    - mesh processing algorithms
    - free, LGPL licence

Tools

- Meshlab (meshlab.sourceforge.net) - free:
  - triangle mesh processing with many features
  - based on the VCGlib
- OpenFlipper (www.openflipper.org) - free:
  - polygon mesh modeling and processing
  - based on OpenMesh
- Graphite (alice.loria.fr) - free:
  - polygon mesh modeling, processing and rendering
  - based on CGAL

References

Skolkovo Institute of Science and Technology

Fundamentals of Computer Graphics, Fourth Edition
4th Edition **by Steve Marschner, Peter Shirley**
Chapter 12

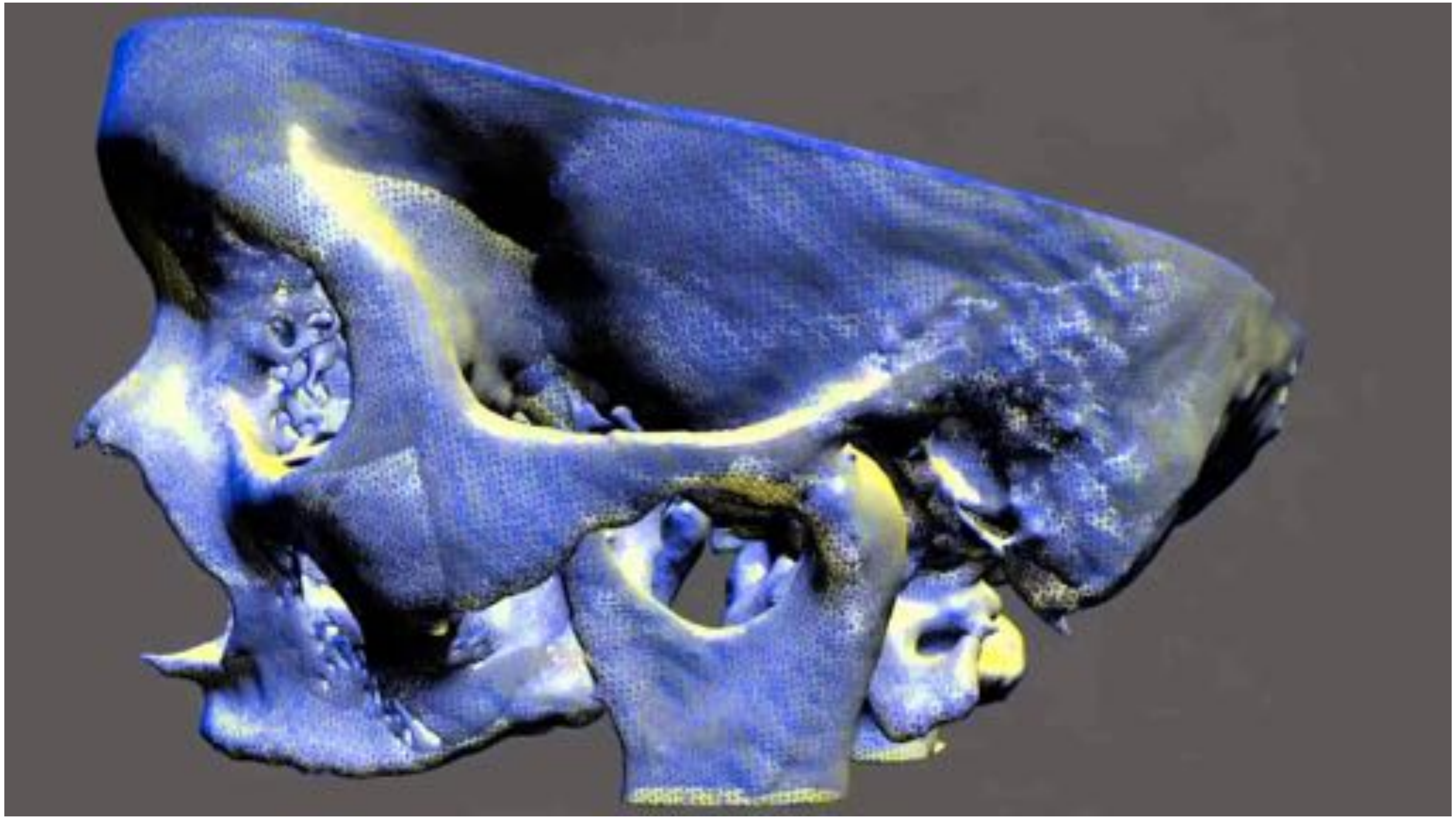
§2. Meshing: constructing meshes

Marching cubes

Meshing: Motivation

Skoltech
Skolkovo Institute of Science and Technology

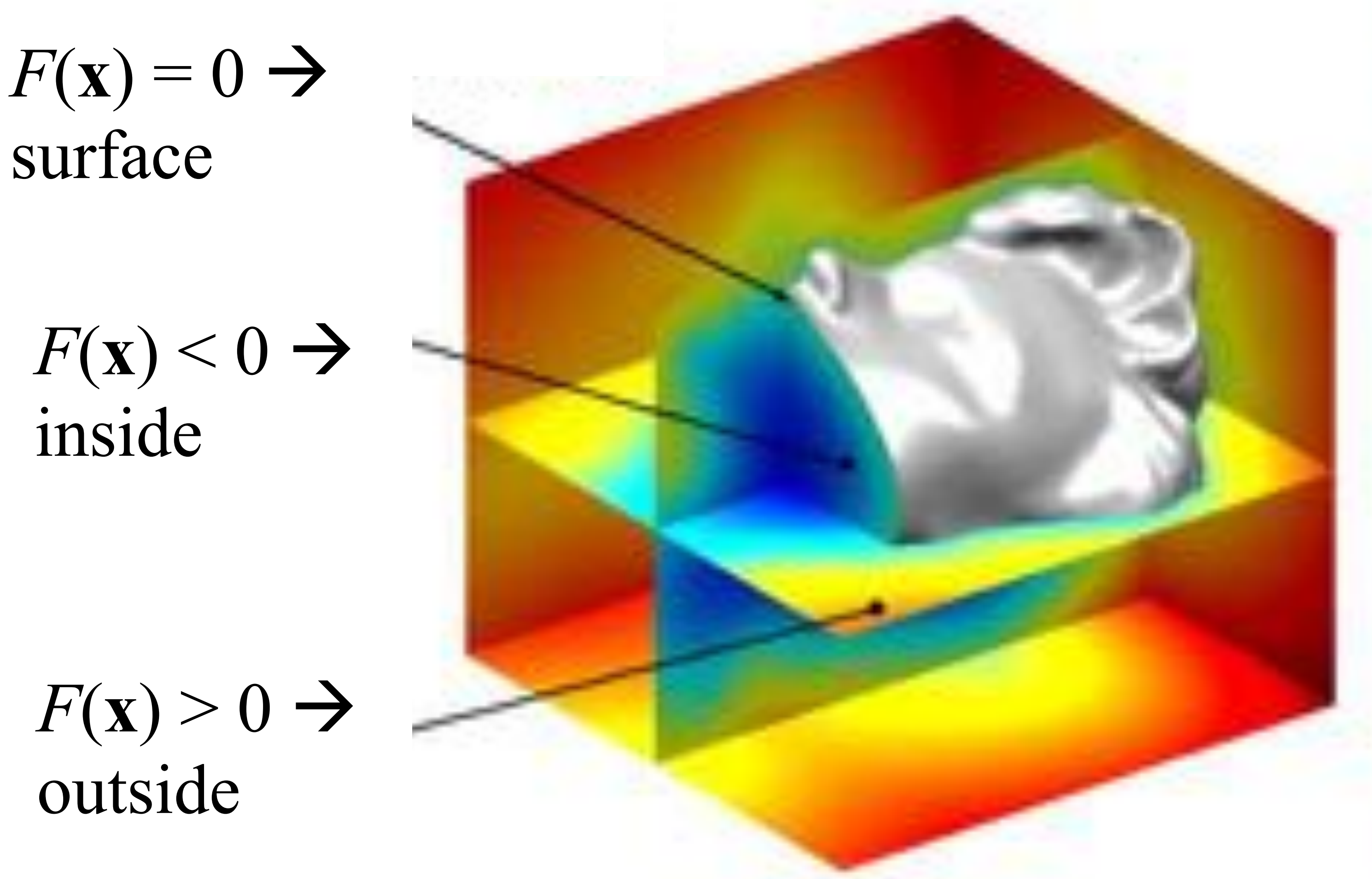
§2. Meshing: constructing meshes



Meshing: Extracting the Surface

§2. Meshing: constructing meshes

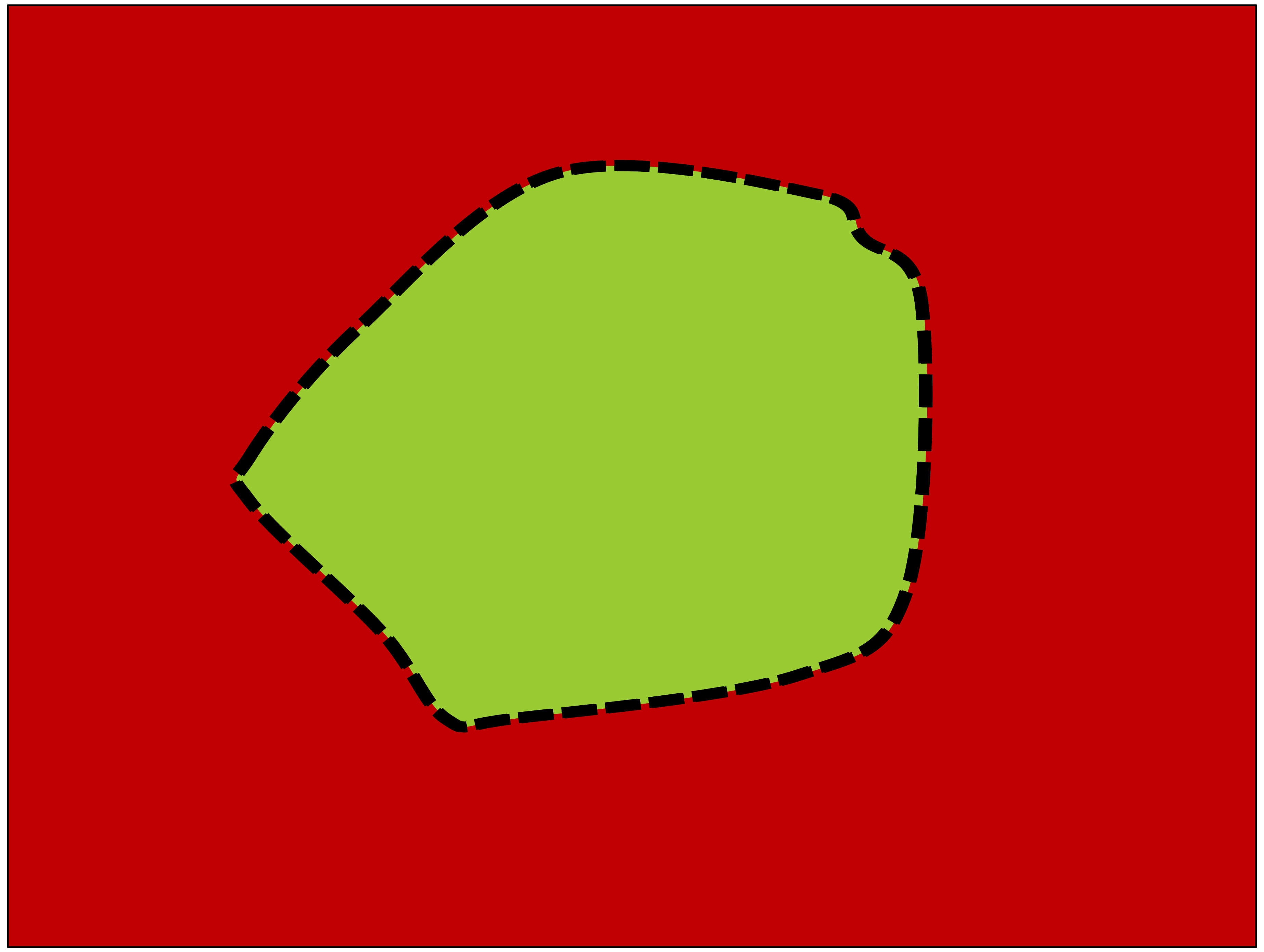
- Wish to compute a manifold mesh of the level set
- **Mesh:** a subdivision of a continuous geometric space into discrete geometric and topological cells (often simplicial surface is constructed)
- **Meshing:** implicit surface → simplicial surface



Slide Credit: Denis Zorin

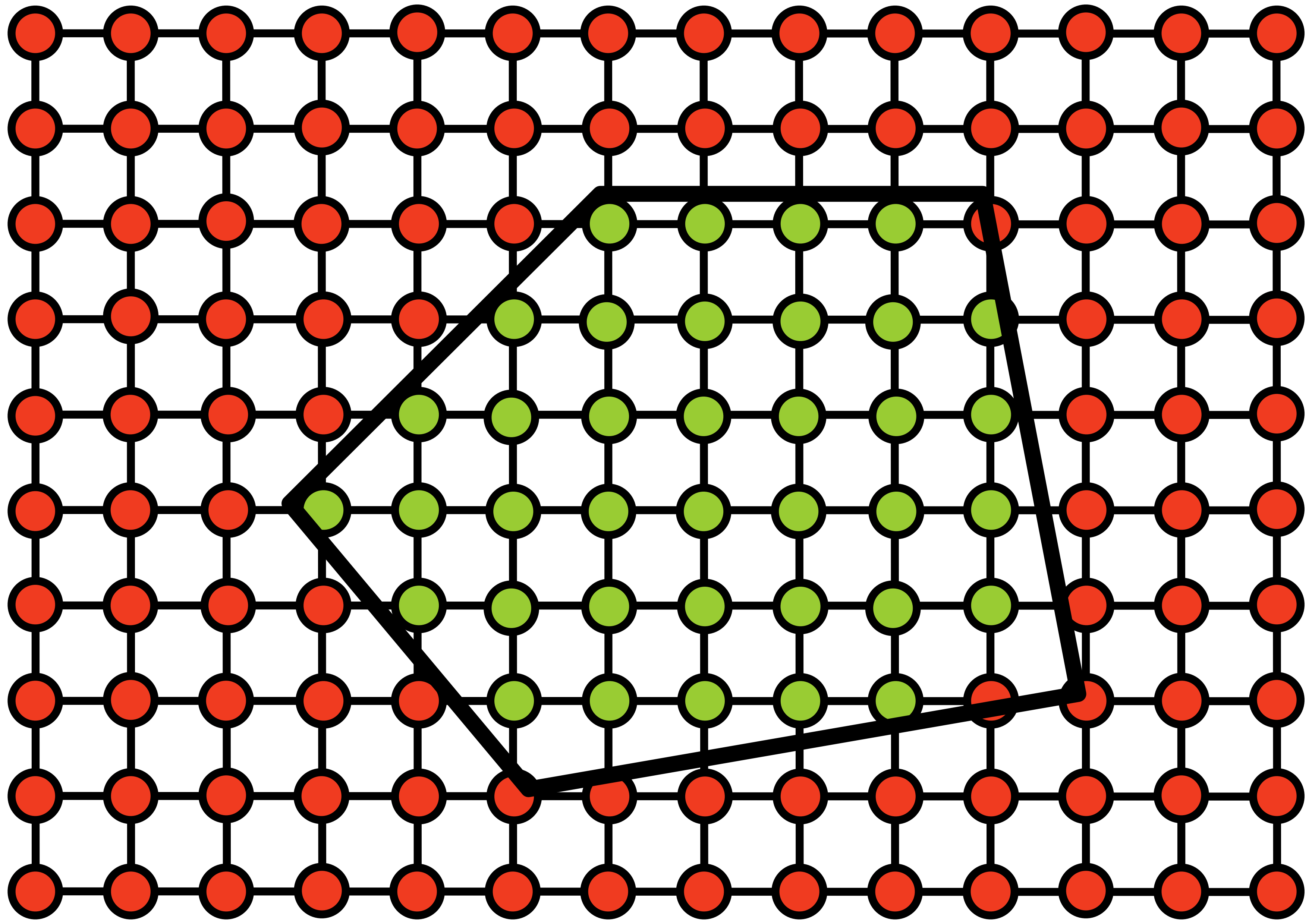
Meshing: Sample the SDF

§2. Meshing: constructing meshes



Meshing: Sample the SDF

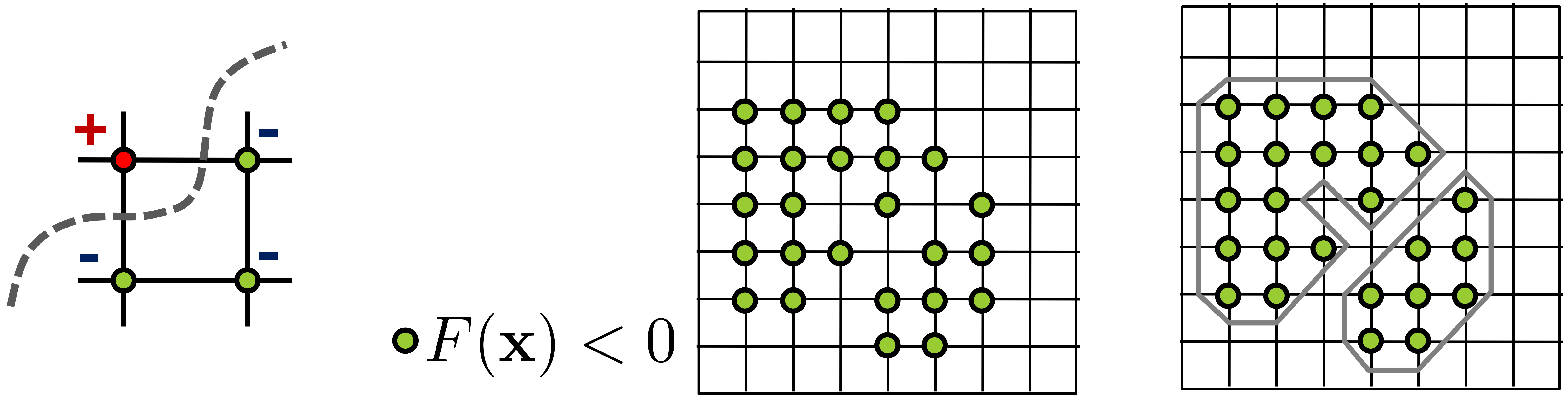
§2. Meshing: constructing meshes



Meshing: Tessellation

§2. Meshing: constructing meshes

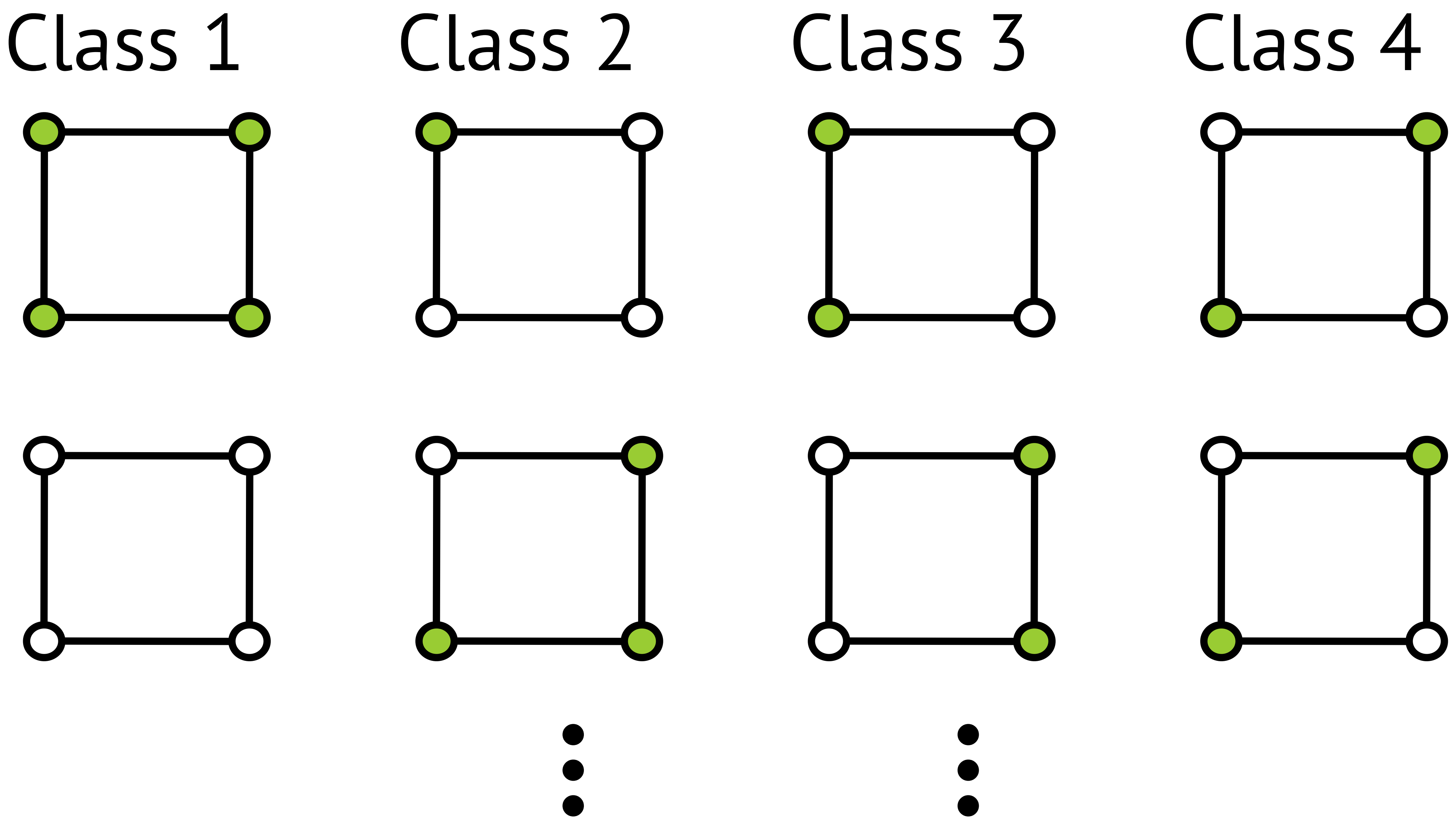
- Want to approximate an implicit surface with a mesh
- Can‘t explicitly compute all the roots
  - Sampling the level set is difficult (root finding)
- Solution: find approximate roots by trapping the implicit surface in a grid (lattice)



Meshing: Marching Squares

§2. Meshing: constructing meshes

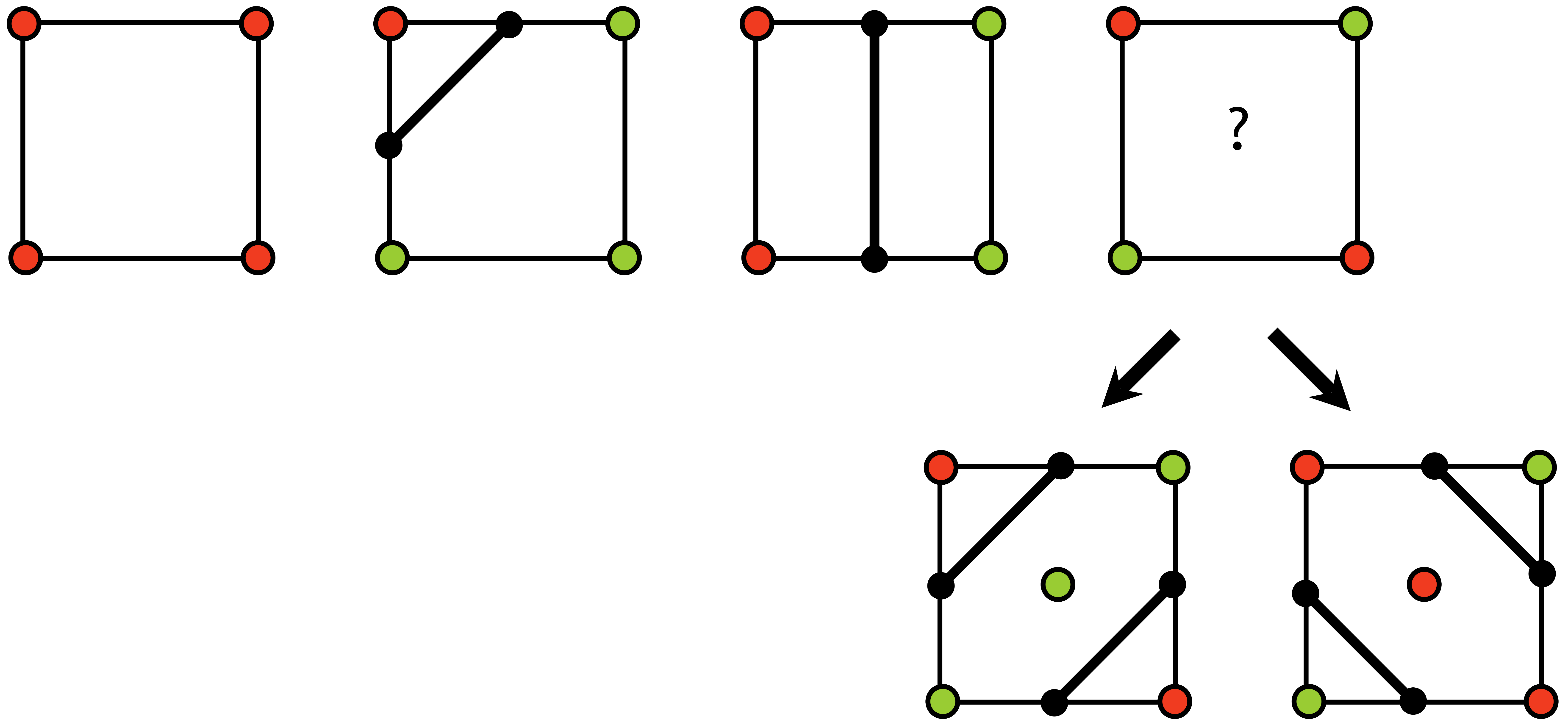
- 16 different configurations in 2D
- 4 equivalence classes (up to rotational and reflection symmetry + complement)



Meshing: Tessellation in 2D

§2. Meshing: constructing meshes

- 4 equivalence classes (up to rotational and reflection symmetry + complement)

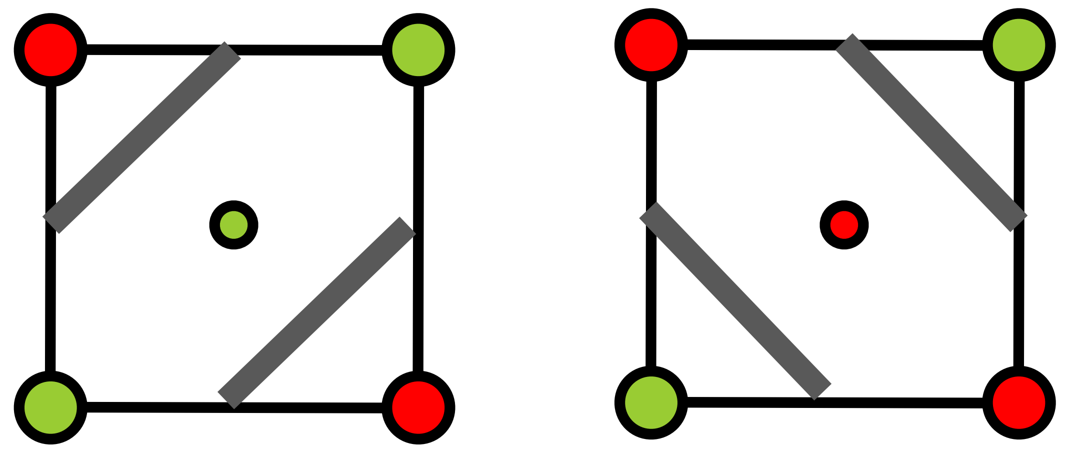


Meshing: Tessellation in 2D

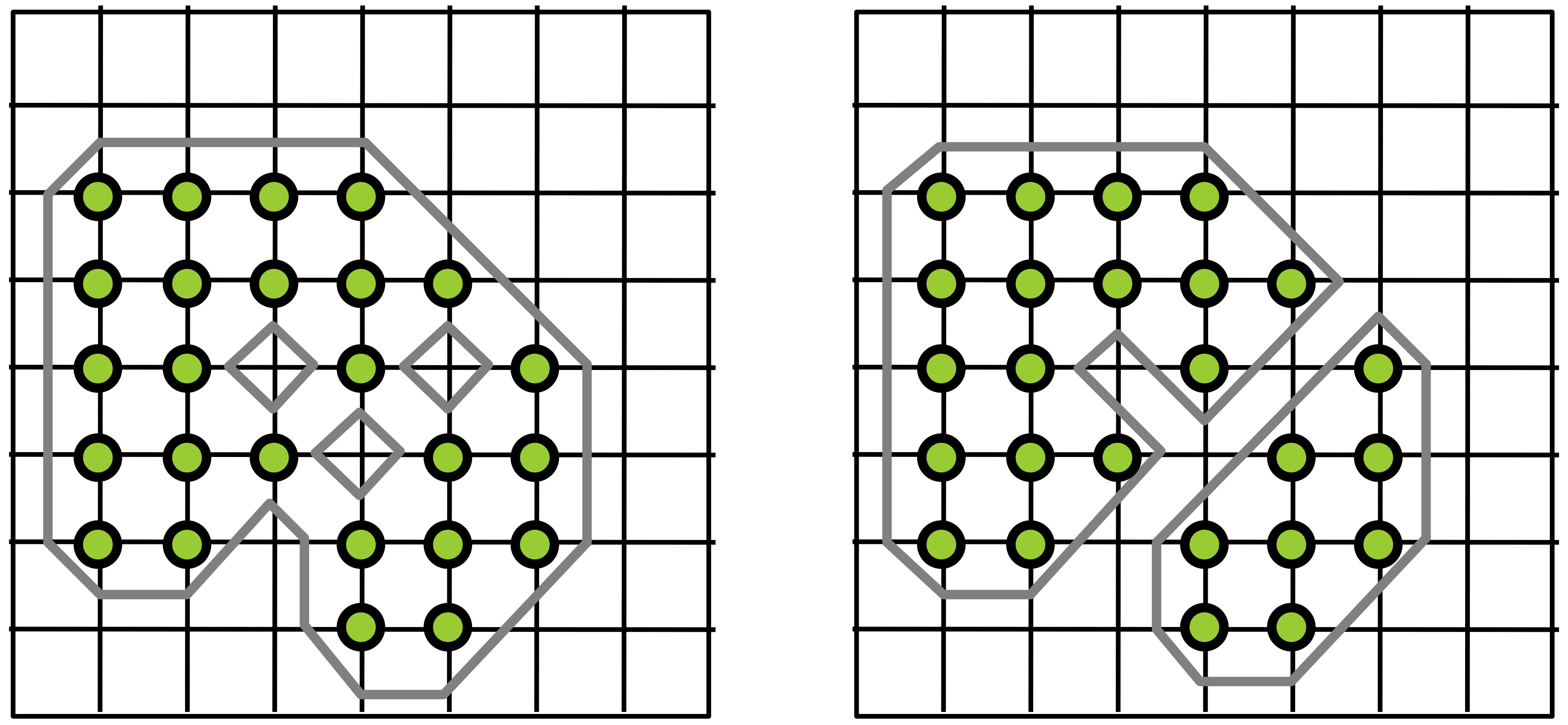
Skoltech
Skolkovo Institute of Science and Technology

§2. Meshing: constructing meshes

- Case 4 is ambiguous:



- Always pick consistently to avoid problems with the resulting mesh

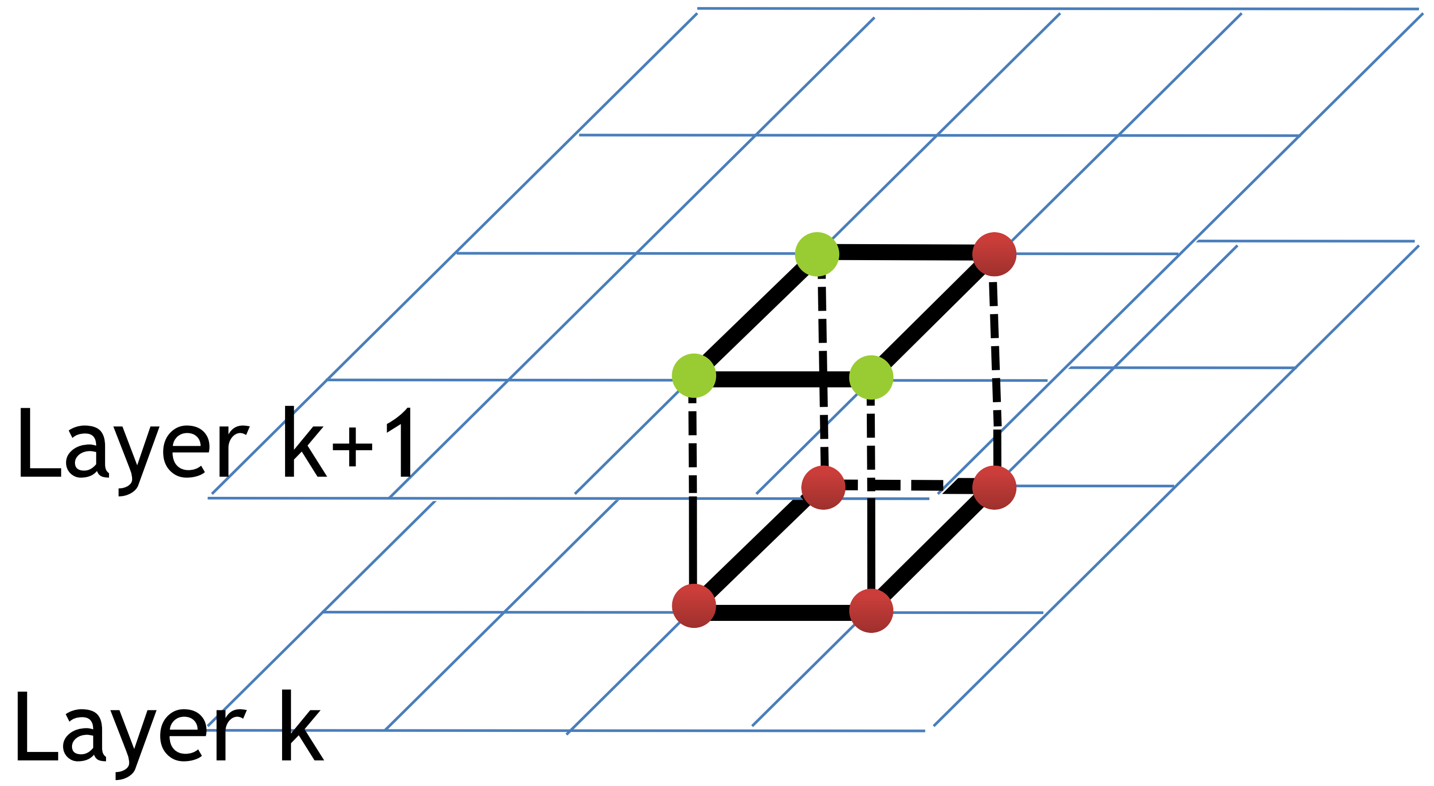


Slide Credit: Daniele Panozzo

Meshing: 3D Marching Cubes

§2. Meshing: constructing meshes

- Marching Cubes (Lorensen and Cline 1987)
  1. Load 4 layers of the grid into memory
  2. Create a cube whose vertices lie on the two middle layers
  3. Classify the vertices of the cube according to the implicit function (inside, outside or on the surface)

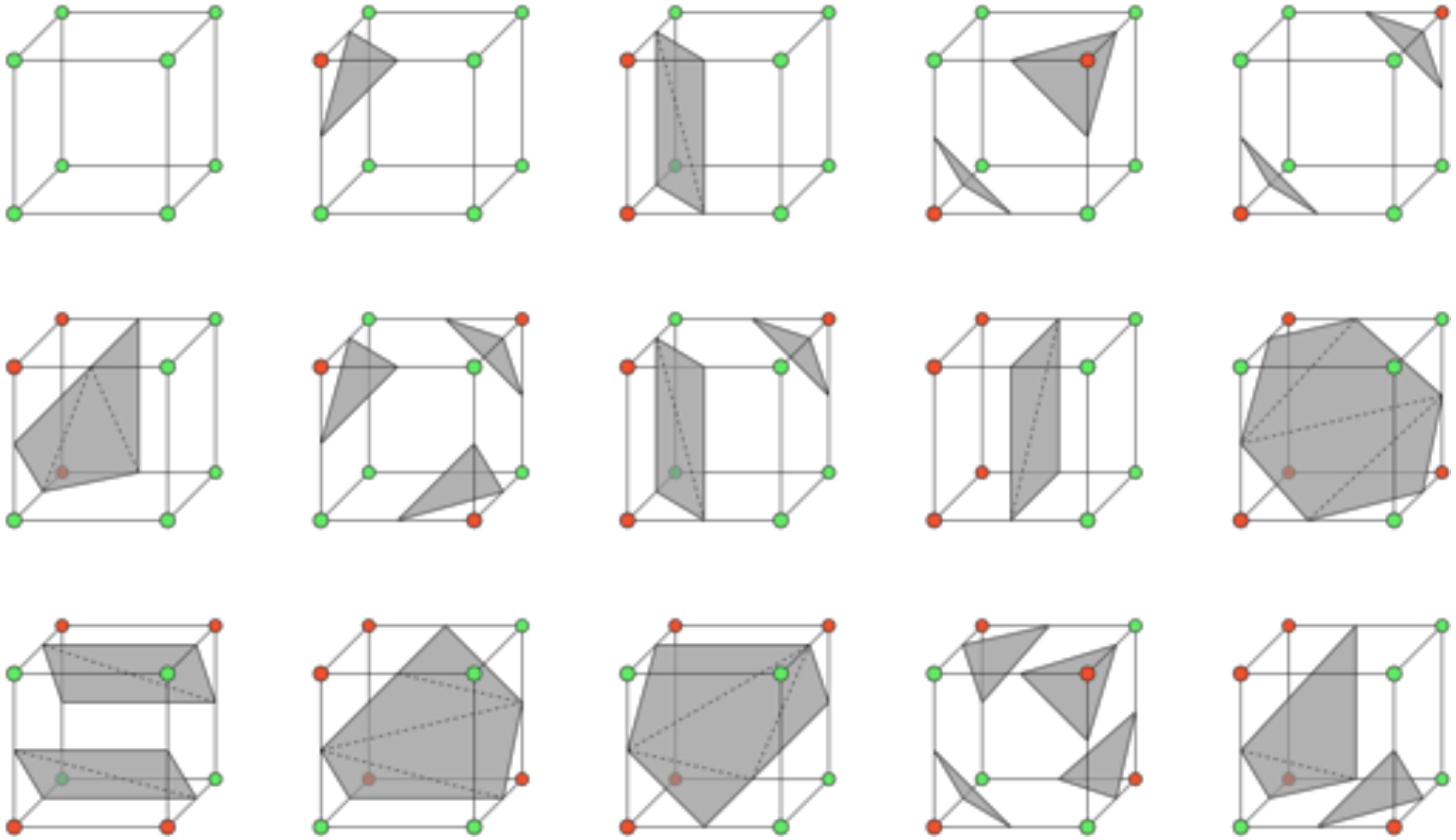


Meshing: 3D Marching Cubes

Skoltech
Skolkovo Institute of Science and Technology

§2. Meshing: constructing meshes

- Unique cases (by rotation, reflection and complement)

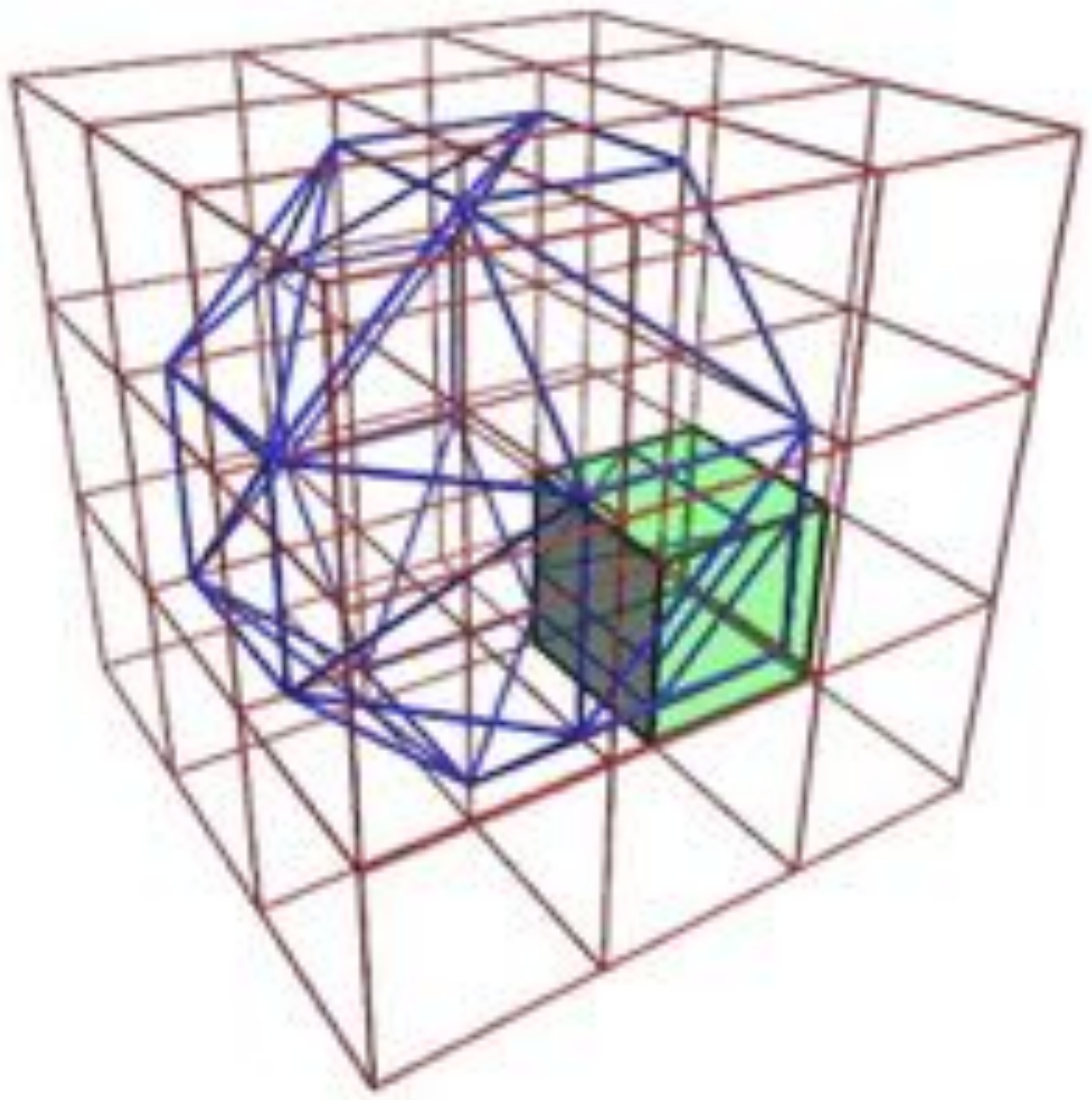


Meshing: 3D Marching Cubes

Skoltech
Skolkovo Institute of Science and Technology

§2. Meshing: constructing meshes

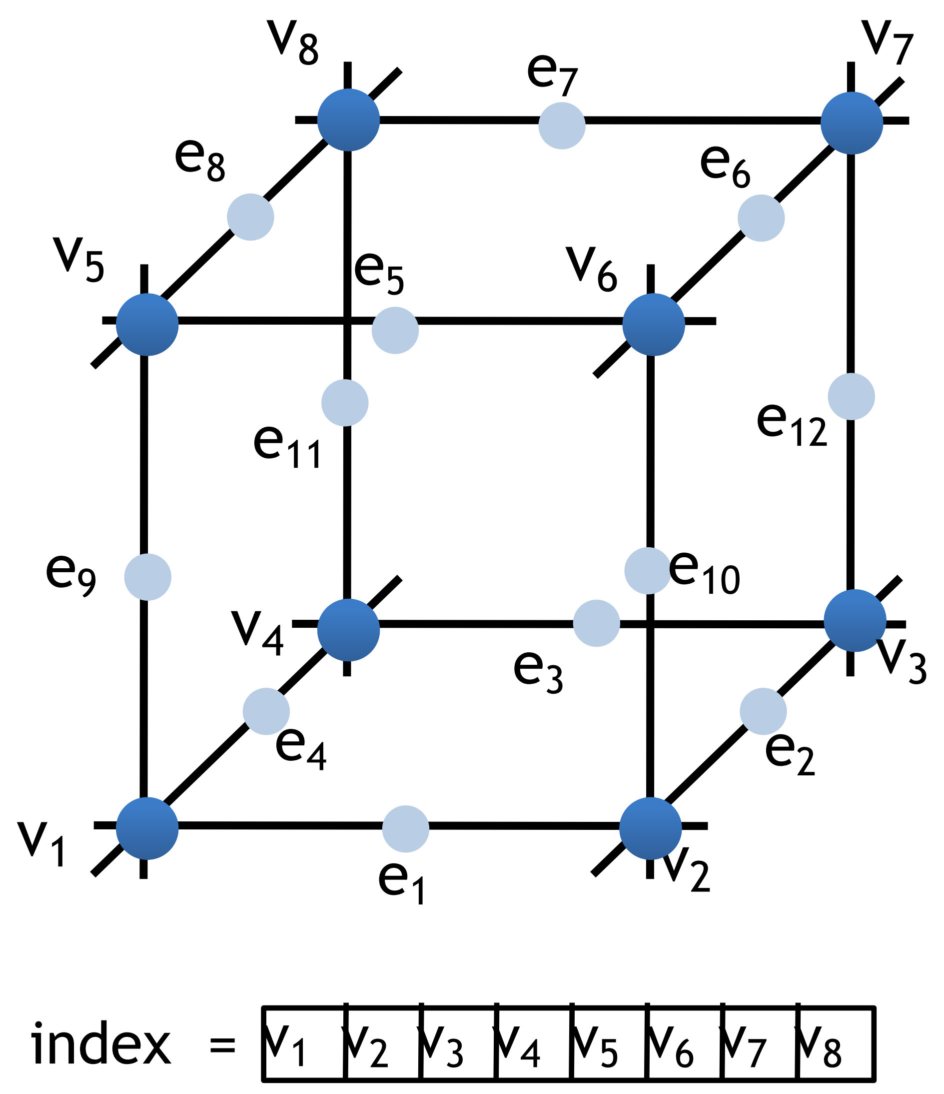
Implementation

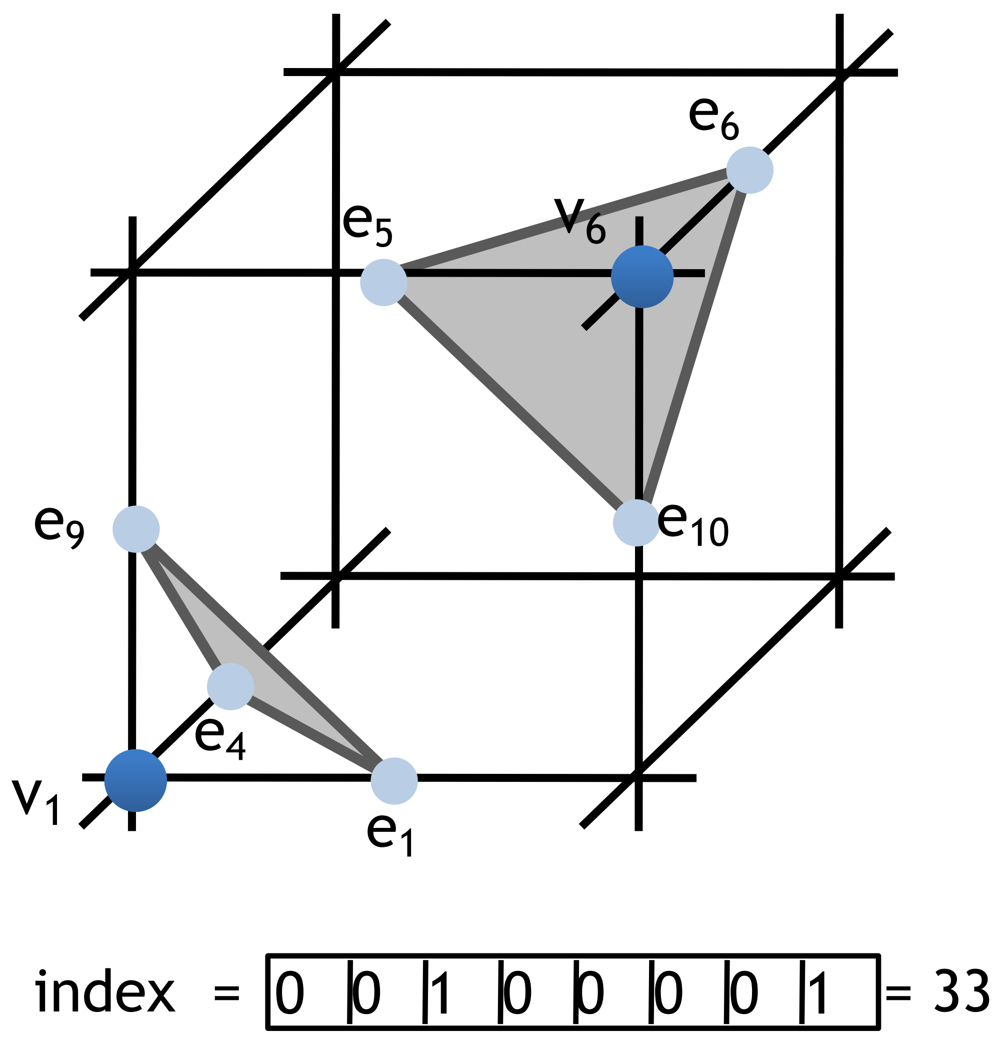


Marching Cubes

§2. Meshing: constructing meshes

- Compute case index. We have $2^8 = 256$ cases (0/1 for each of the eight vertices) – can store as 8 bit (1 byte) index.

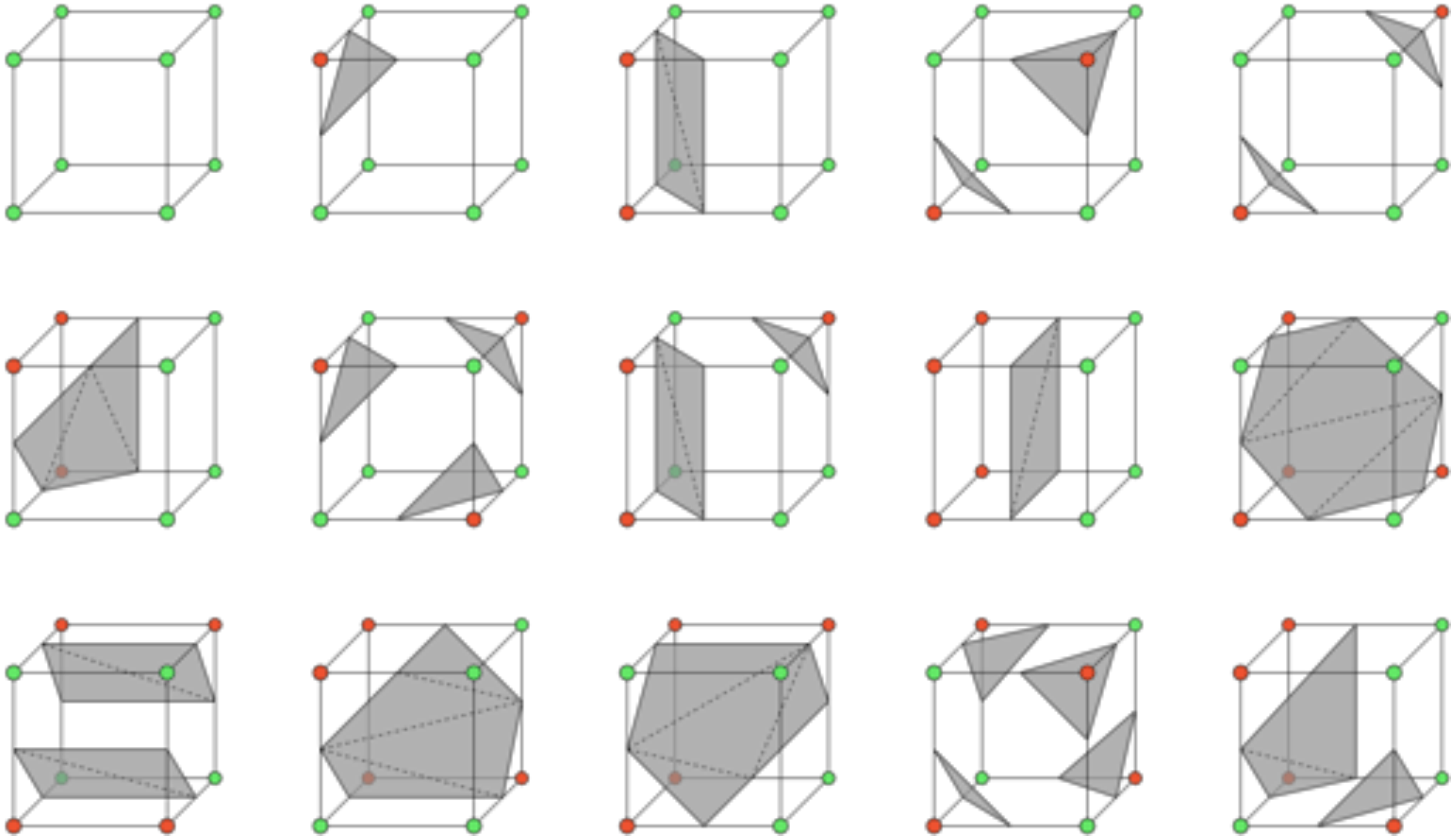




Marching Cubes

§2. Meshing: constructing meshes

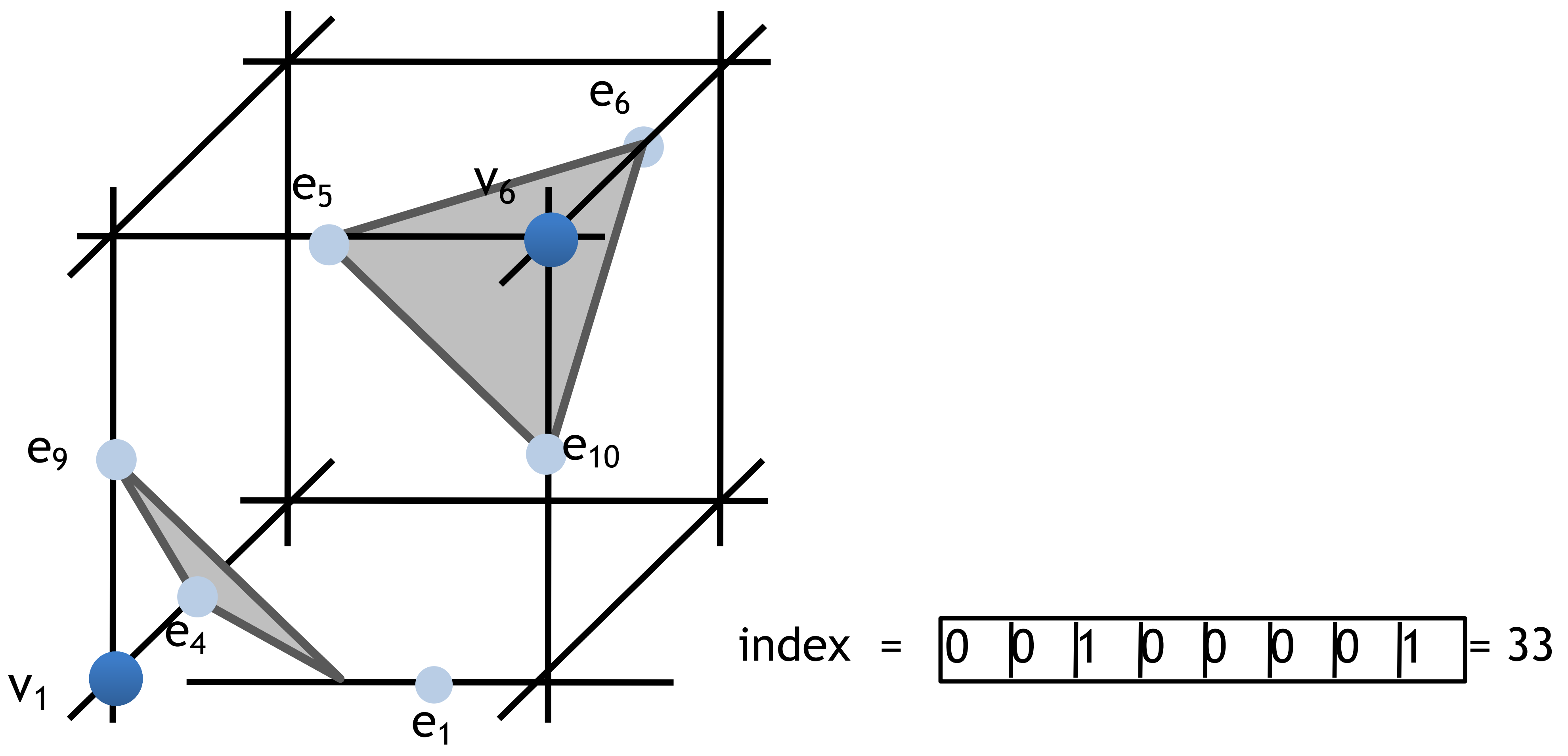
- Unique cases (by rotation, reflection and complement)



Tessellation 3D – Marching Cubes

§2. Meshing: constructing meshes

- Using the case index, retrieve the connectivity in the look-up table
- Example: the entry for index 33 in the look-up table indicates that the cut edges are e_1; e_4; e_5; e_6; e_9 and e_{10} ; the output triangles are (e_1; e_9; e_4) and (e_5; e_{10}; e_6).



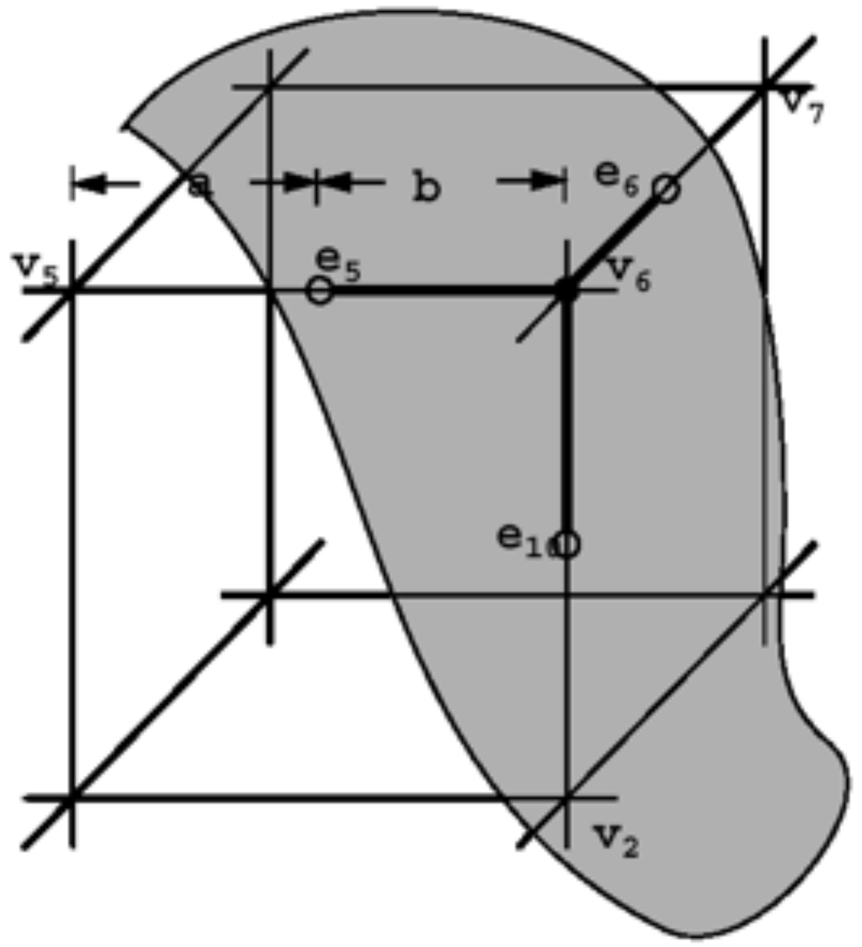
Marching Cubes

§2. Meshing: constructing meshes

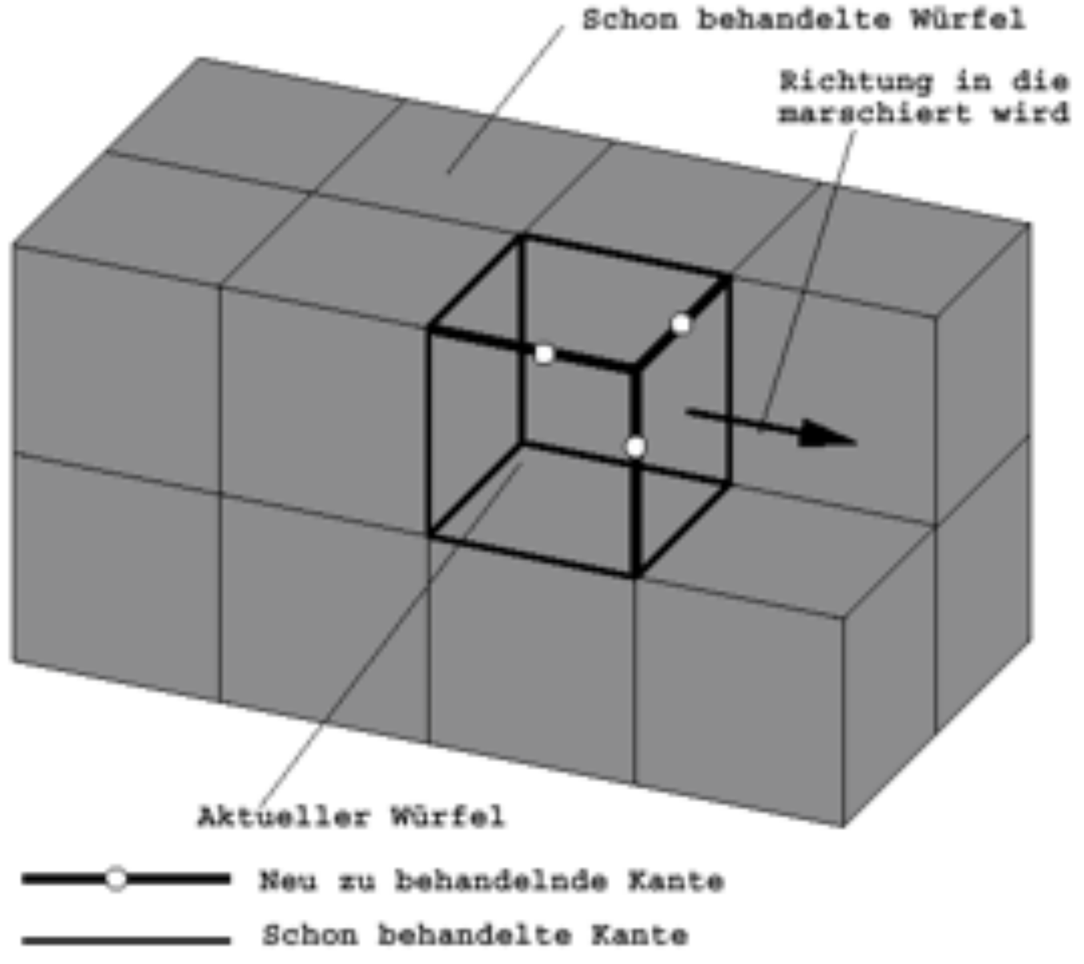
- Compute the position of the cut vertices by linear interpolation:

$$\mathbf{v}_s = t\mathbf{v}_a + (1-t)\mathbf{v}_b$$

$$t = \frac{F(\mathbf{v}_b)}{F(\mathbf{v}_b) - F(\mathbf{v}_a)}$$



- Move to the next cube

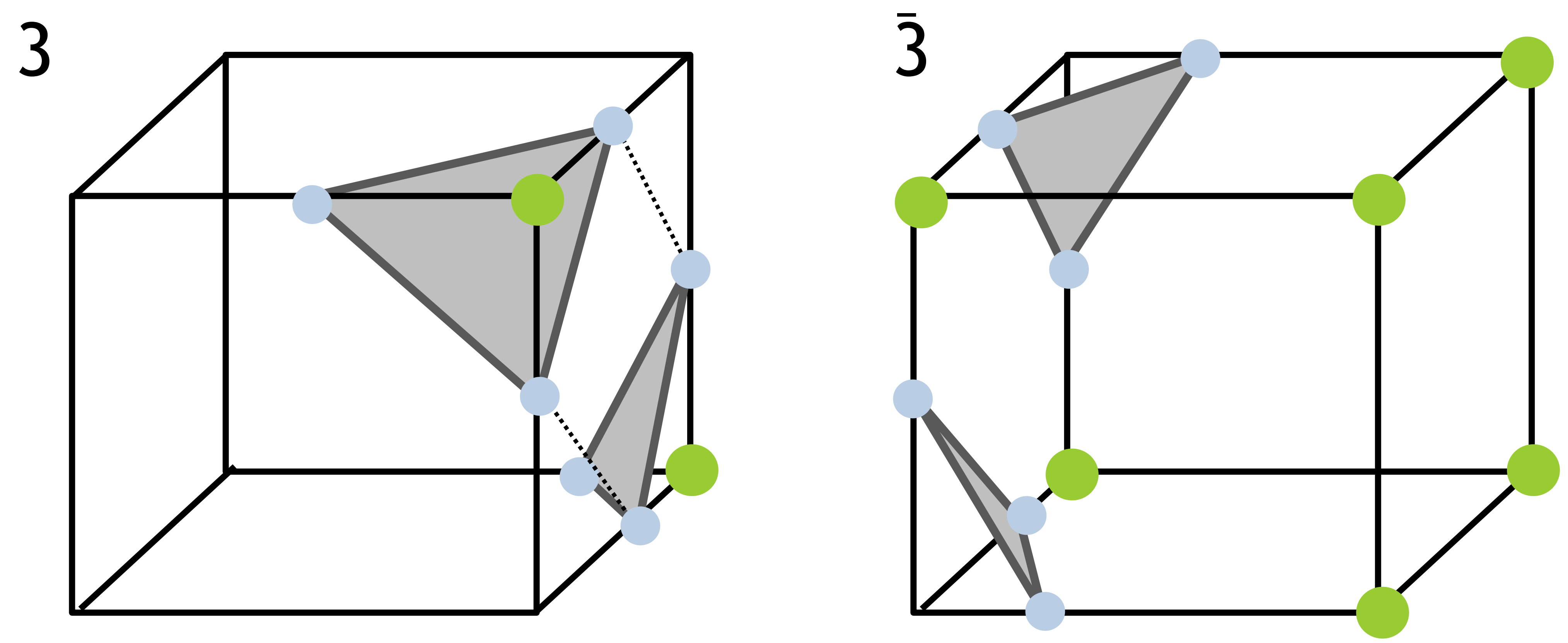


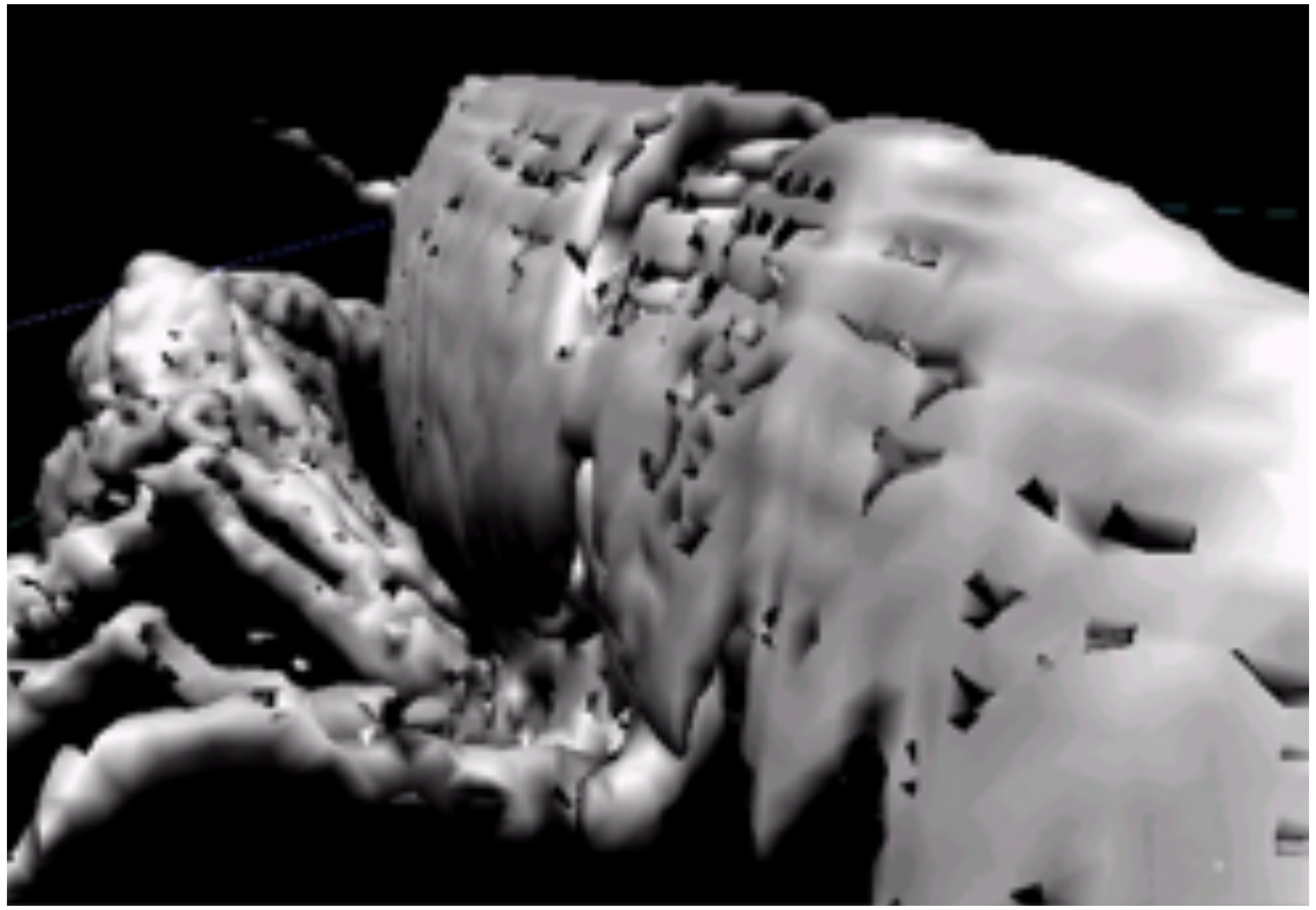
Marching Cubes – Problems

Skoltech
Skolkovo Institute of Science and Technology

§2. Meshing: constructing meshes

- Have to make consistent choices for neighboring cubes – otherwise get holes



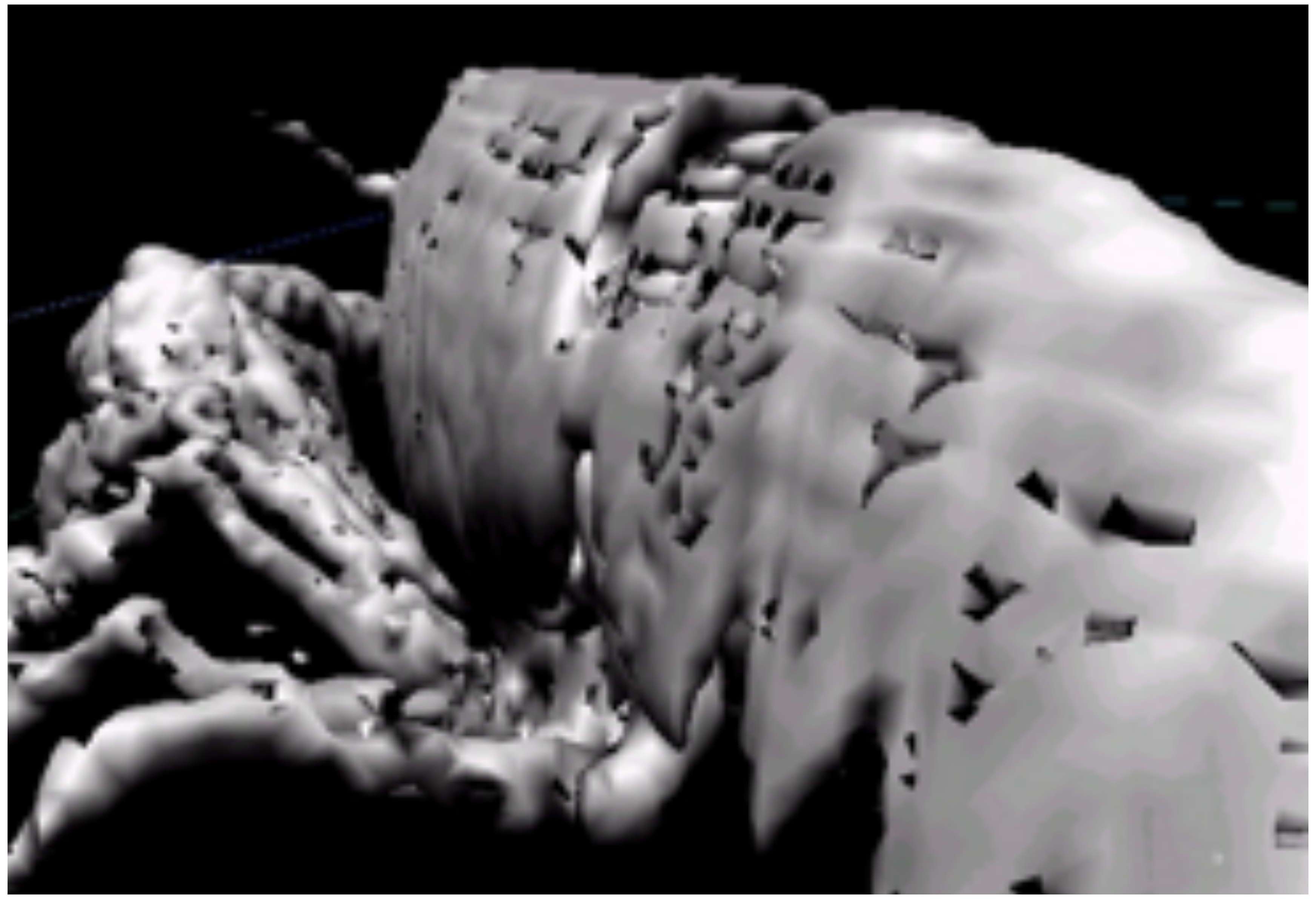


Marching Cubes – Problems

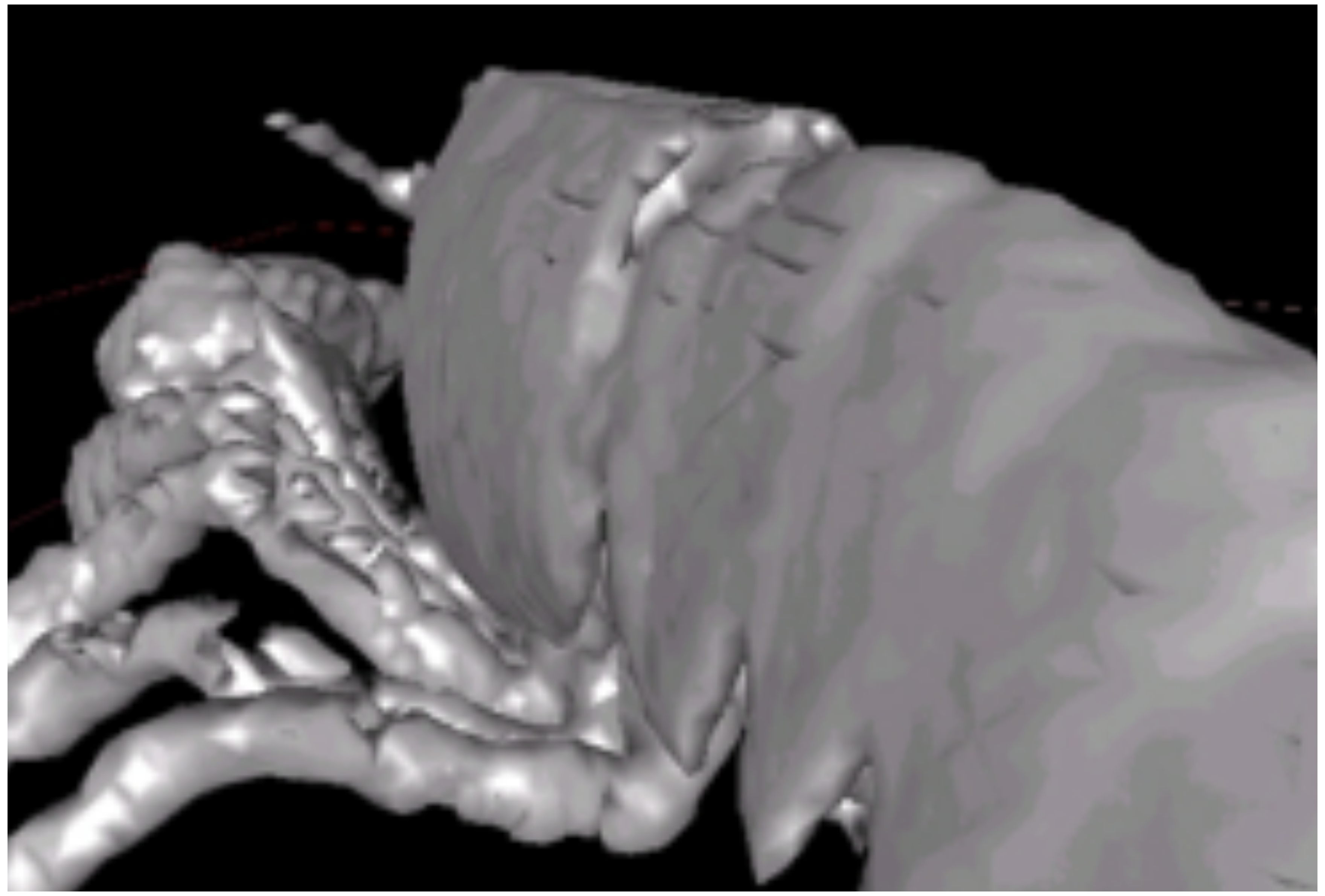
Skoltech
Skolkovo Institute of Science and Technology

§2. Meshing: constructing meshes

- Resolving ambiguities



Ambiguity



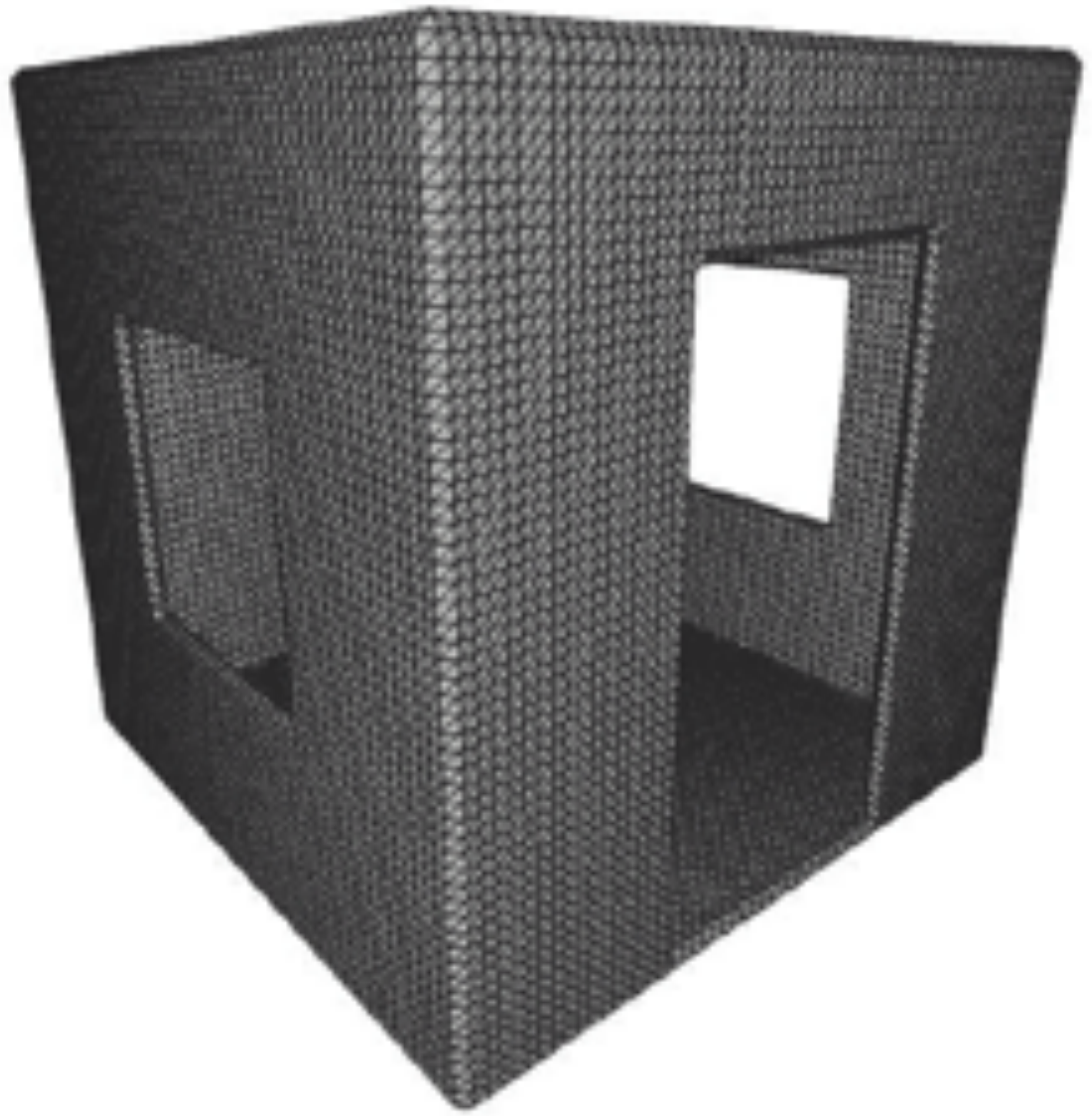
No Ambiguity

Marching Cubes – Problems

Skolkovo Institute of Science and Technology

§2. Meshing: constructing meshes

- Grid not adaptive
- Many polygons required to represent small features

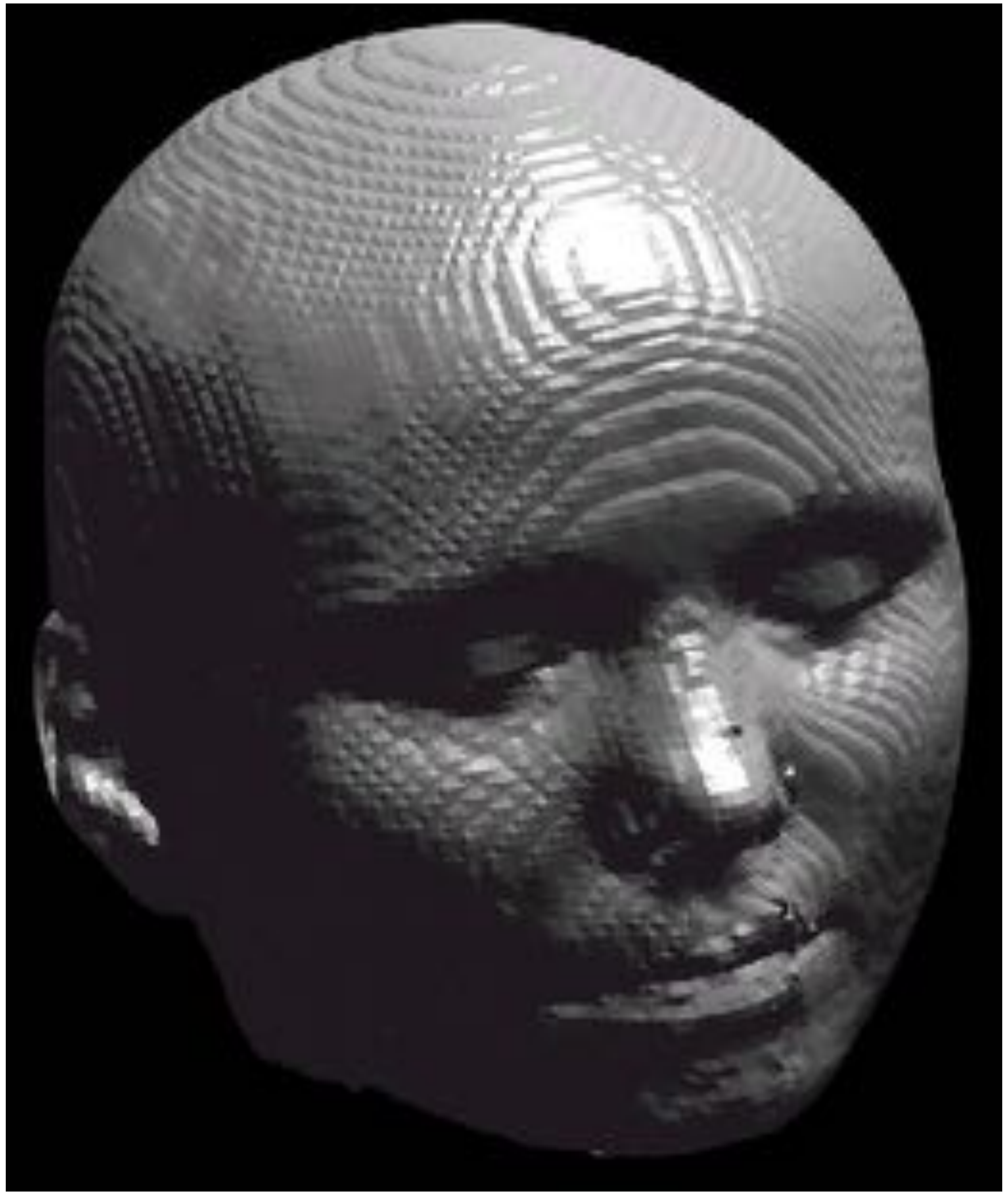


Images from: “Dual Marching Cubes: Primal Contouring of Dual Grids” by Schaeffer et al.

Marching Cubes – Problems

Skoltech
Skolkovo Institute of Science and Technology

§2. Meshing: constructing meshes

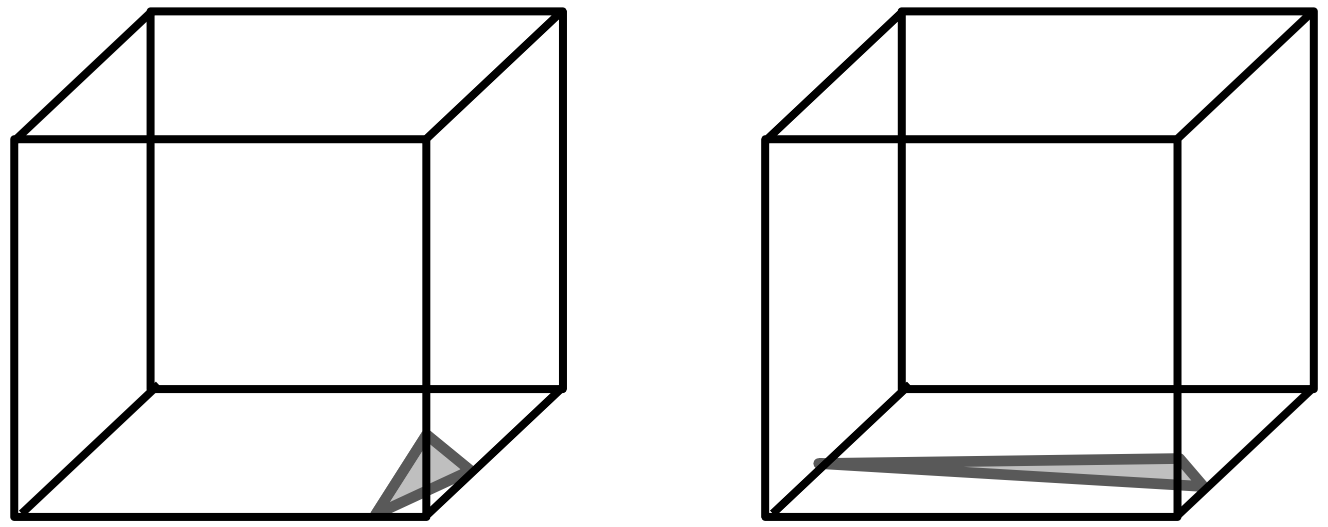


Marching Cubes – Problems

Skoltech
Skolkovo Institute of Science and Technology

§2. Meshing: constructing meshes

- Problems with short triangle edges
  - When the surface intersects the cube close to a corner, the resulting tiny triangle doesn‘t contribute much area to the mesh
  - When the intersection is close to an edge of the cube, we get skinny triangles (bad aspect ratio)
- Triangles with short edges waste resources but don‘t contribute to the surface mesh representation



Grid Snapping

Skoltech
Skolkovo Institute of Science and Technology

§2. Meshing: constructing meshes

- Solution: threshold the distances between the created vertices and the cube corners
- When the distance is smaller than d_{snap} we snap the vertex to the cube corner
- If more than one vertex of a triangle is snapped to the same point, we discard that triangle altogether

Skoltech
Skolkovo Institute of Science and Technology

Grid Snapping

§2. Meshing: constructing meshes

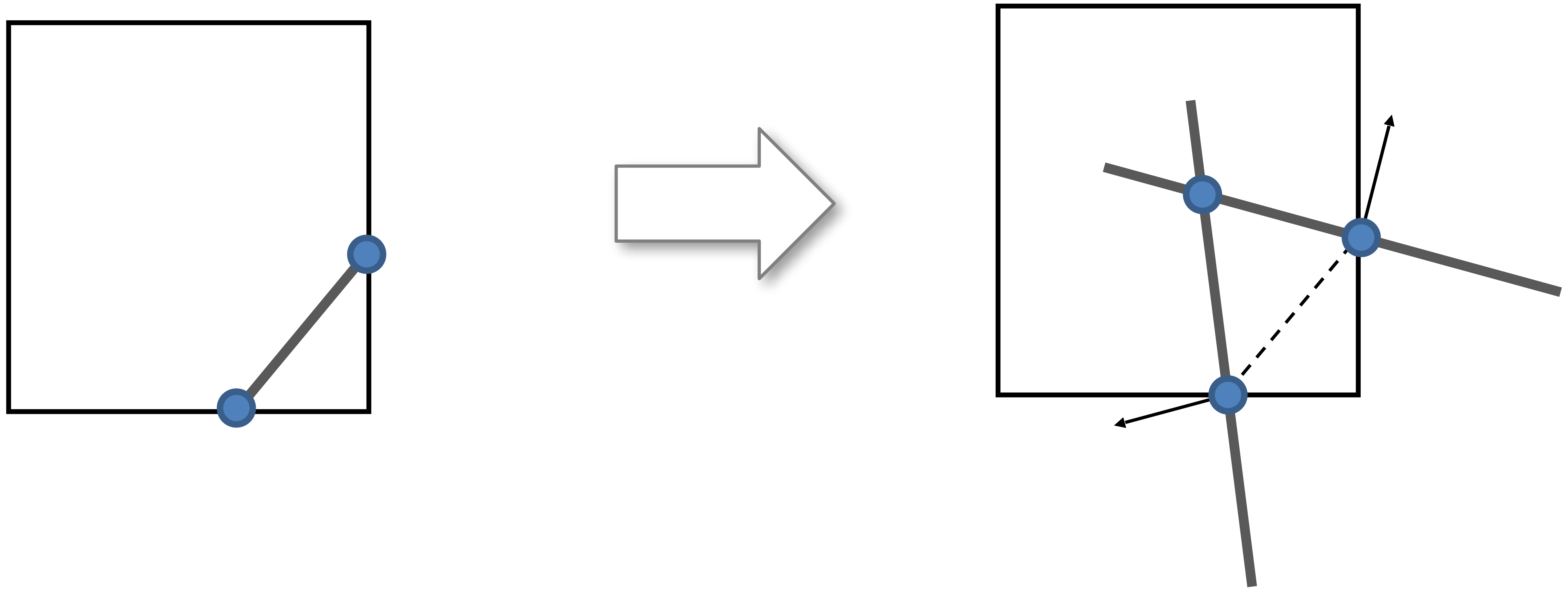
- With Grid-Snapping one can obtain significant reduction of space consumption

Parameter	0	0,1	0,2	0,3	0,4	0,46	0,495
Vertices	1446	1398	1254	1182	1074	830	830
Reduction	0	3,3	13,3	18,3	25,7	42,6	42,6

Sharp Corners and Features

§2. Meshing: constructing meshes

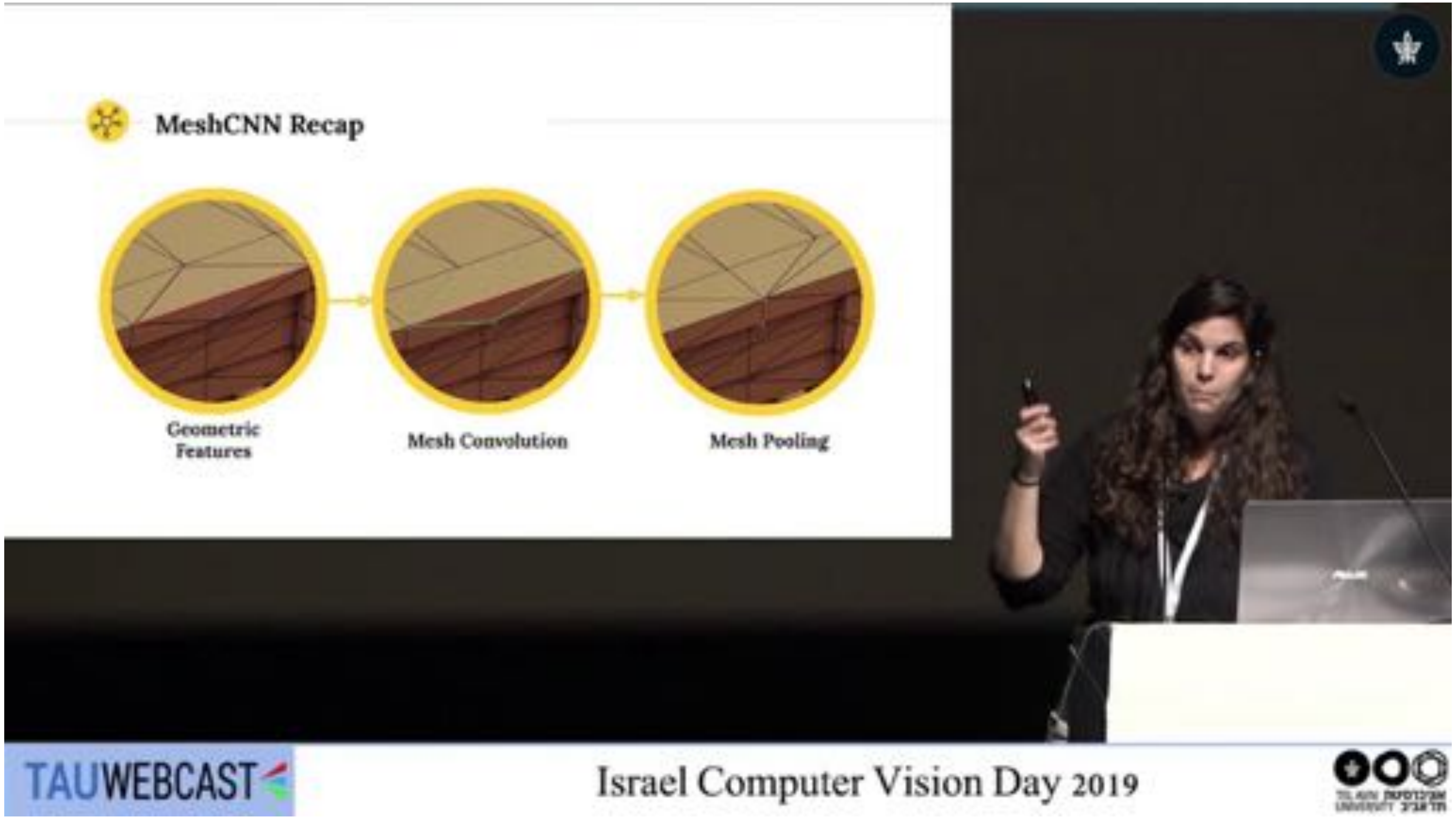
- (Kobbelt et al. 2001):
  - Evaluate the normals (use gradient of *F*)
  - When they significantly differ, create additional vertex



§3. Defining convolutions on meshes

3.1. MeshCNN: convolutions on edges

§3. Defining convolutions on meshes



References

1. Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., & Lévy, B. (2010). *Polygon mesh processing*. CRC press.