Learning with Meshes

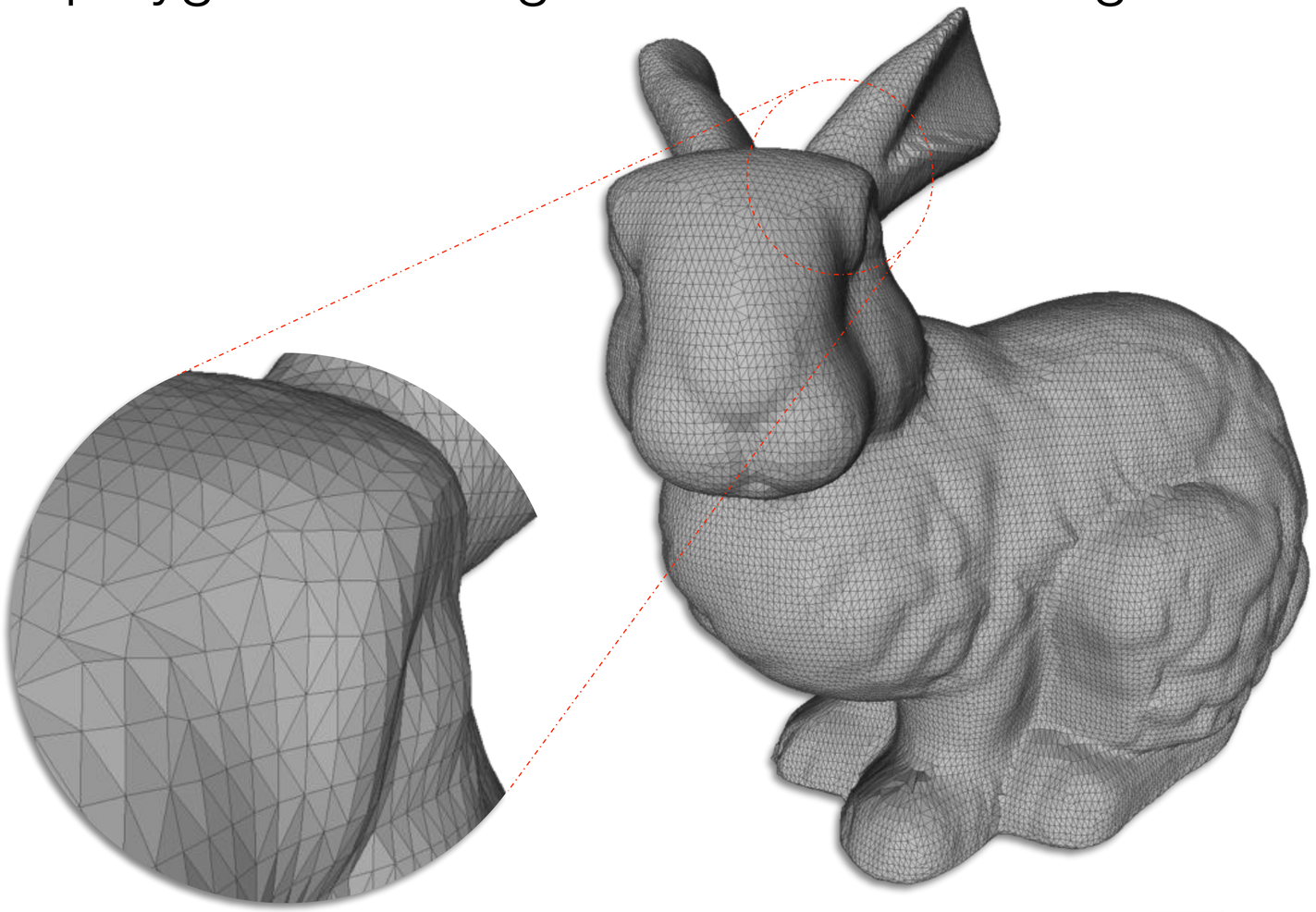
Alexey Artemov

MLSS 2019 tutorial

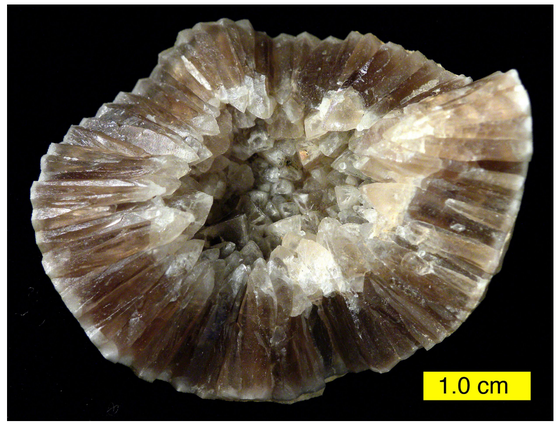
Acknowledgement: Enrico Puppo, Daniele Panozzo,
Michael Bronstein, and Albert Matveev

What is a mesh?

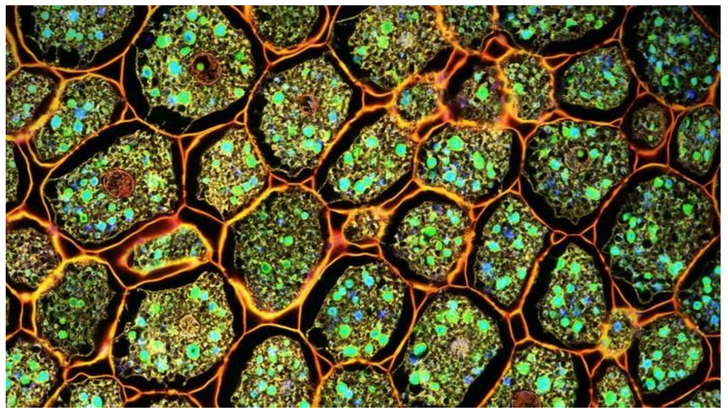
A surface made of polygonal faces glued at common edges

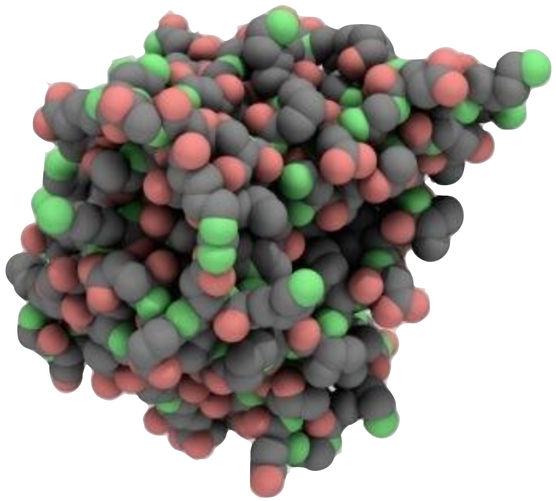


Origin of Meshes



- 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*



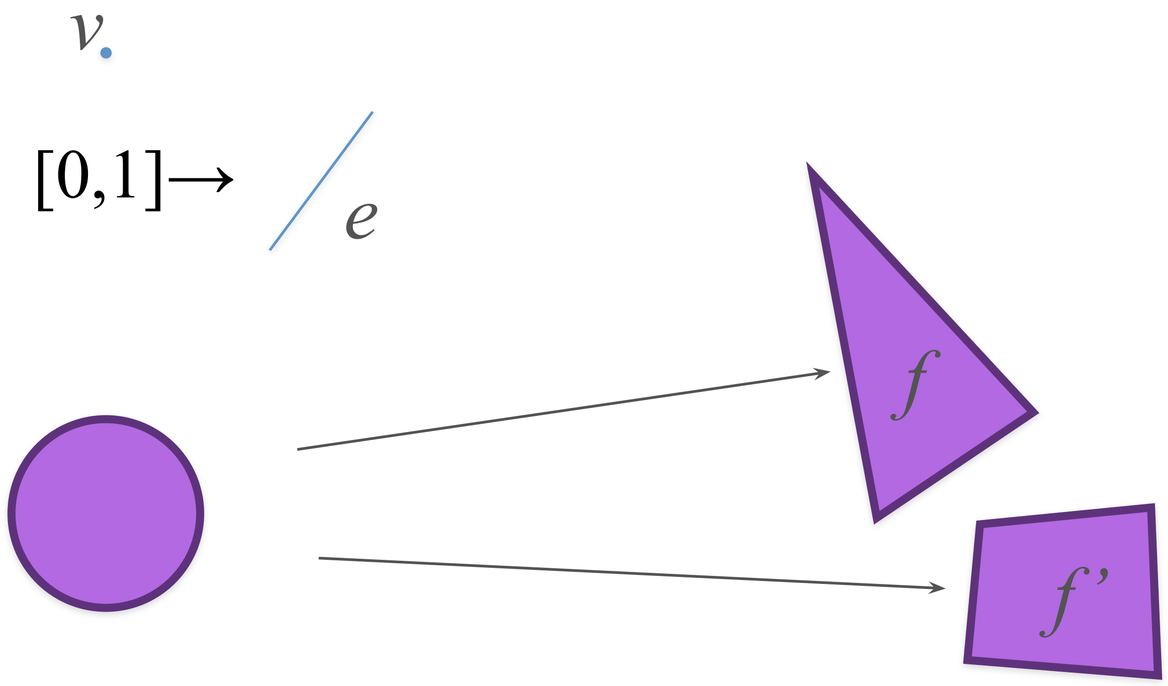


Credit: Fernan Federici Moment Getty Images

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

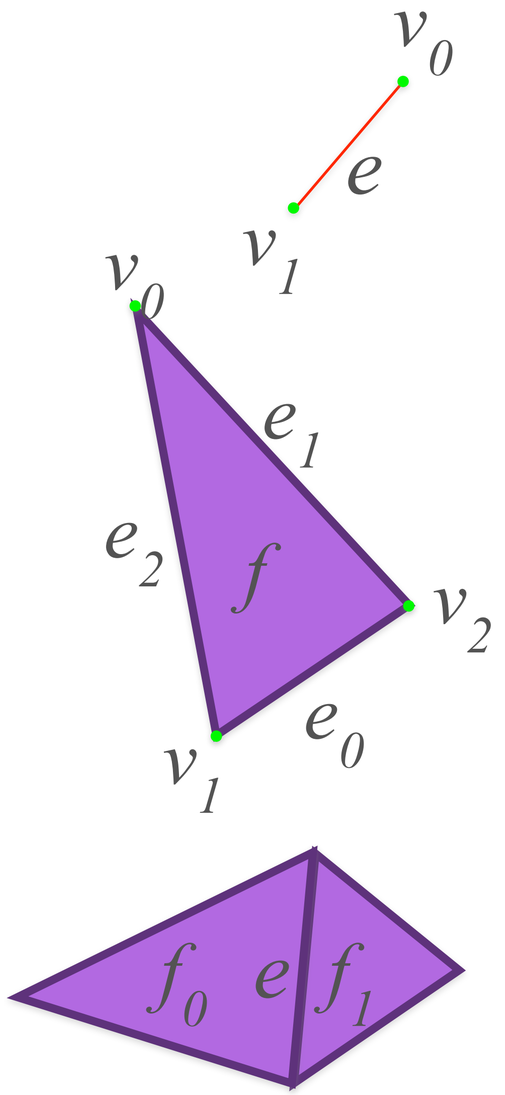
Basic Math of Meshes

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



Structure

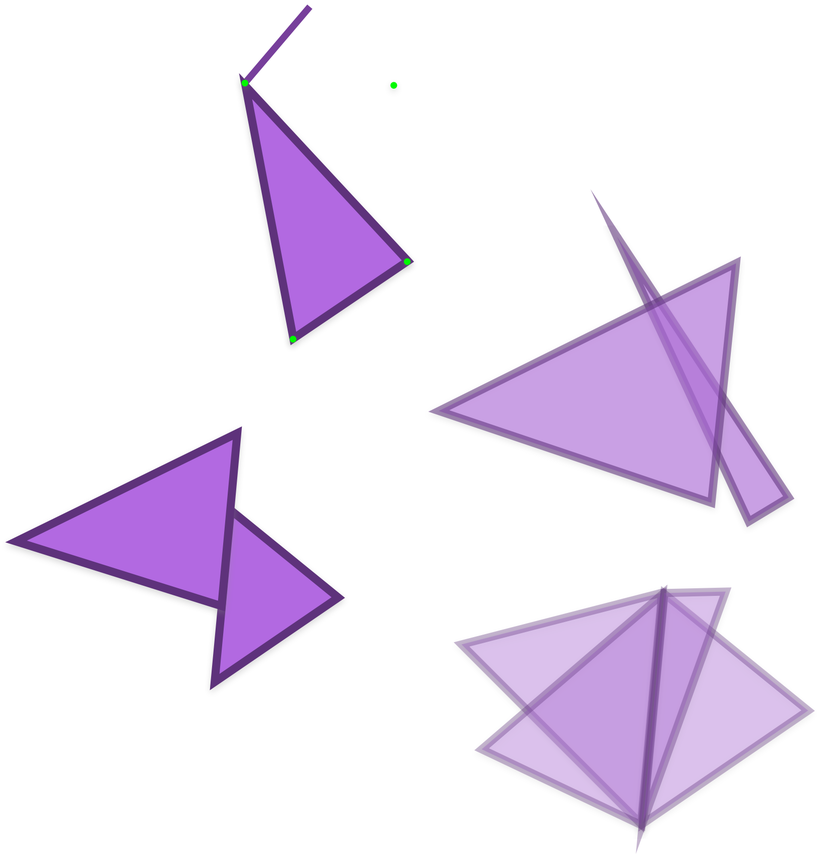
- A ***mesh*** $M=(V,E,F)$ of dimension 2 is made of a collection of k-cells for k = 0, 1, 2:
  - 0-cells of V lie on the boundary of 1-cells of E
  - 1-cells of E lies on the boundary of 2-cells of F
    - **(manifoldness)** each 1-cell of E lies on the boundary of either one or two 2-cells of F
  -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



Structure

Forbidden configurations:

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



Topological Informations

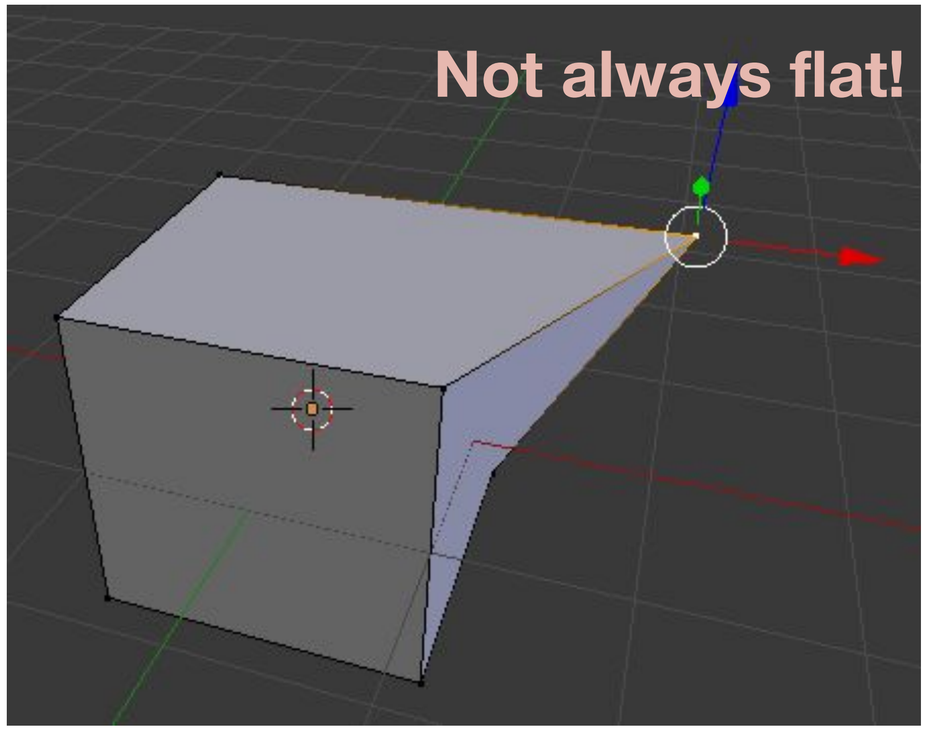
- A mesh can be treated as a purely combinatorial structure

$M = (V,E,F)$

- 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)

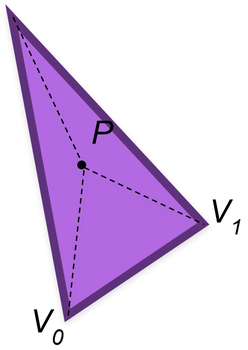
Geometrical Information

- 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*



V_2

Triangle Meshes



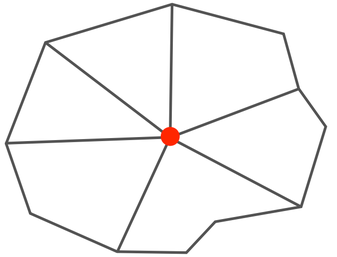
- A ***triangle mesh*** **is a polygonal mesh with all triangular faces**
  - all faces are flat (there exist a unique plane for three points)
- All cells are ***simplices***, i.e., they are the convex combinations of their vertices

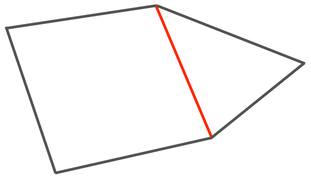
$$P = \lambda_0 V_0 + \lambda_1 V_1 + \lambda_2 V_2 \qquad \lambda_i \in [0,1] \qquad \lambda_0 + \lambda_1 + \lambda_2 = 1$$

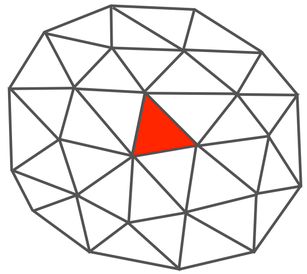
  - embedding of vertices + combinatorial structure characterize the embedding of the whole mesh

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







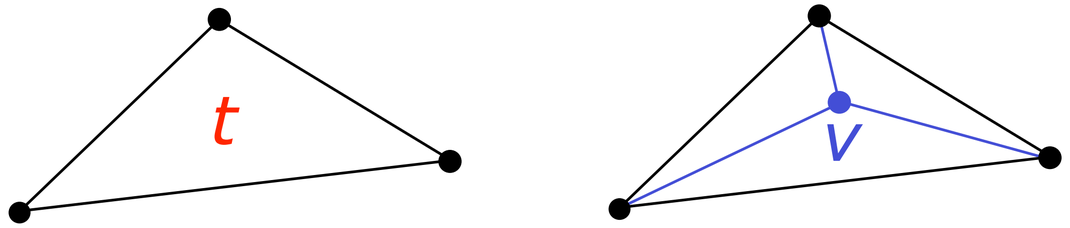
Editing Operators

Operators for Triangle Meshes

- Specific operators for triangle meshes include
- **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

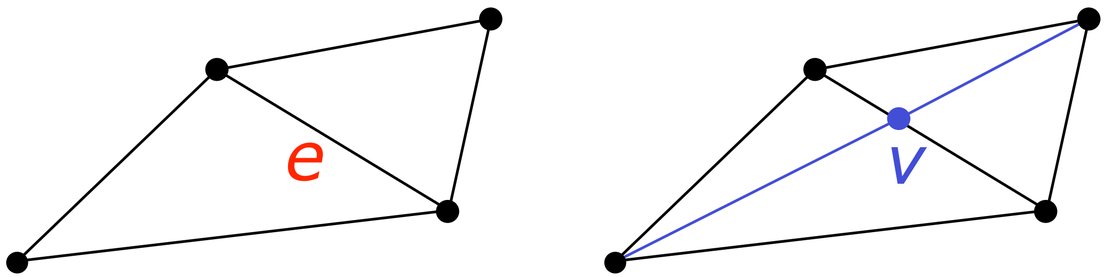
Refinement Operators

- **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



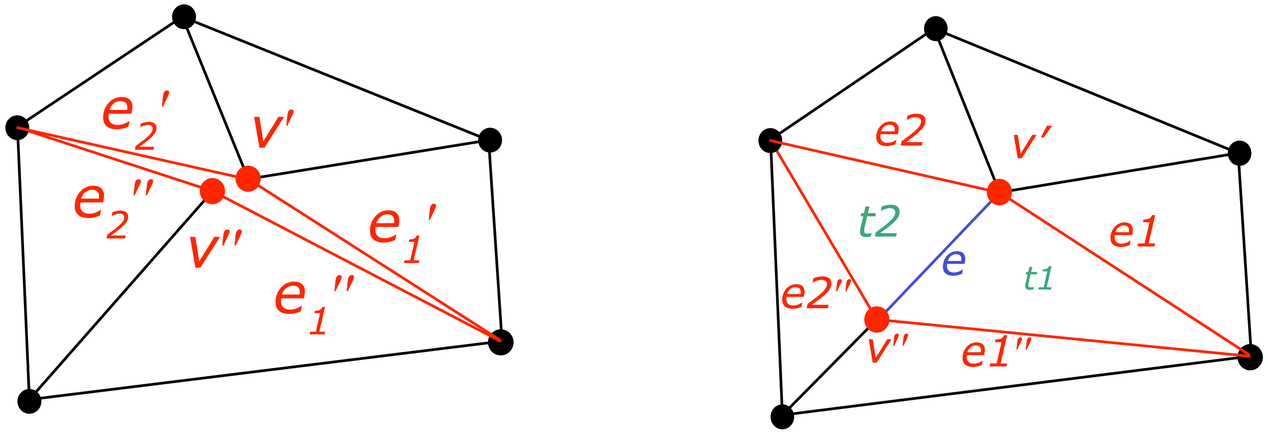
Refinement Operators

- **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



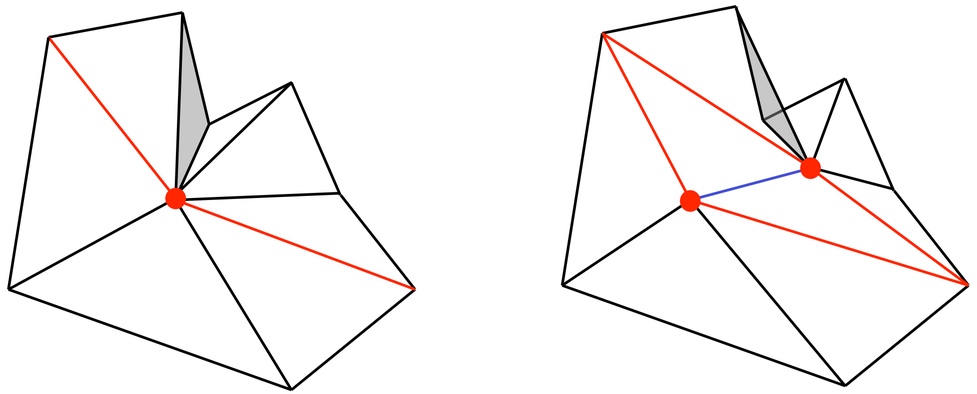
Refinement Operators

- **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
  - fill the quadrangular hole with two new triangles and an edge joining the two copies of v



Refinement Operators

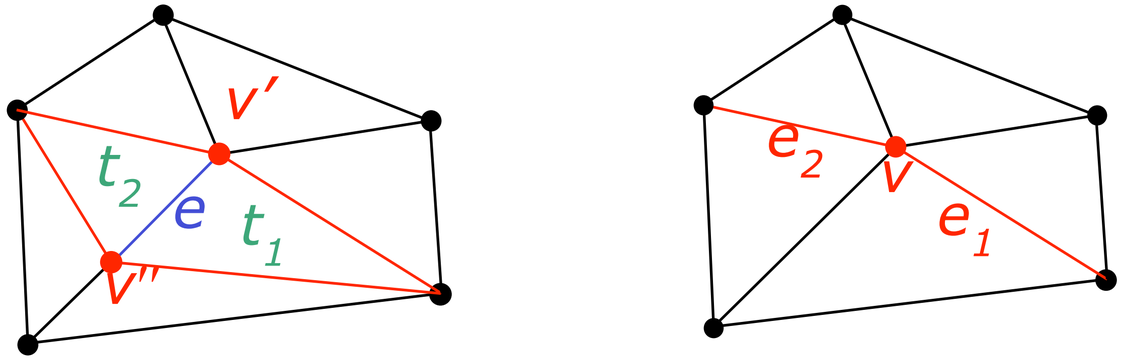
- **Vertex split:**
  - possible inconsistencies because of *triangle flip*



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

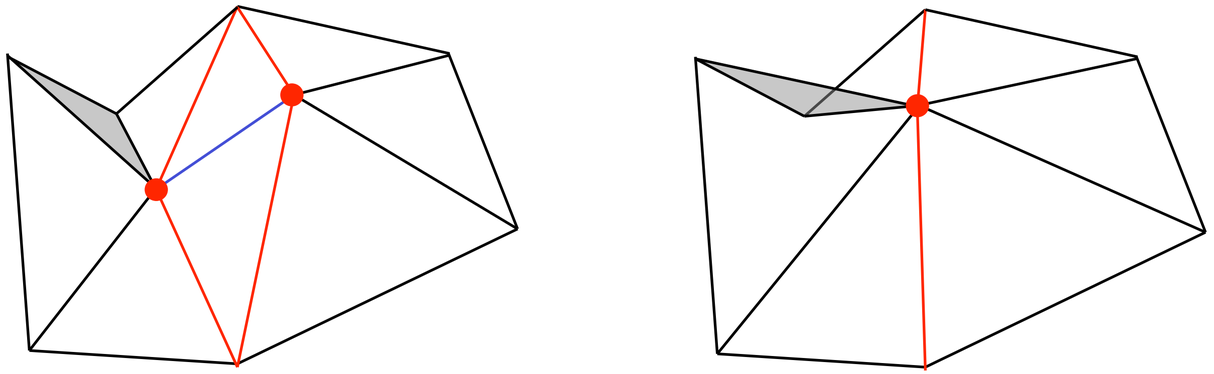
Simplification Operators

- **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

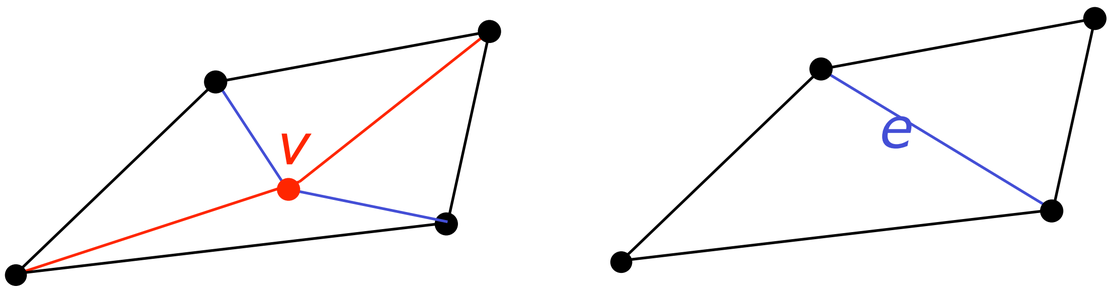
- **Edge collapse:**
  - possible inconsistencies because of *triangle flip*



  - consistency check analogous to vertex split

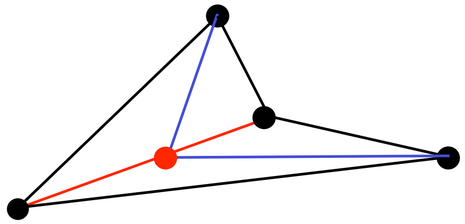
Simplification Operators

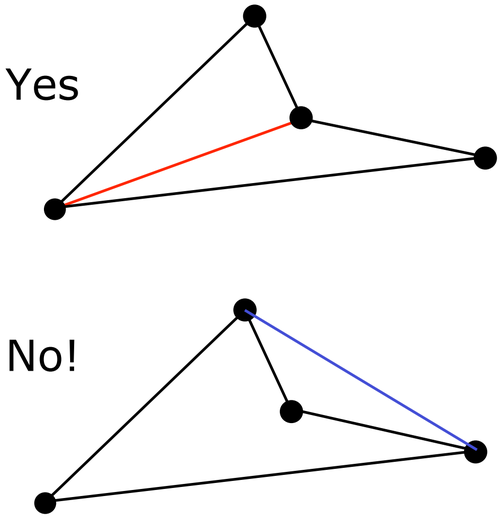
- **Edge merge (reverse of edge split):**
  - take an internal vertex ***v*** with valence 4
  - delete ***v*** together with its incident triangles and edges and fill the hole with two new triangles sharing a new edge ***e***



Simplification Operators

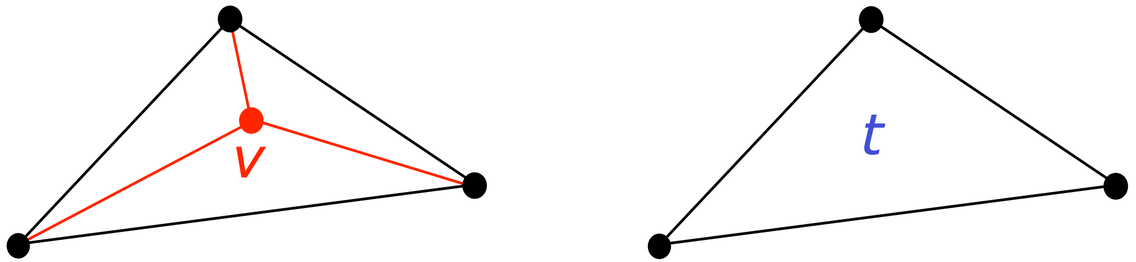
- **Edge merge:**
  - if the hole is not convex, only one diagonal edge can be inserted





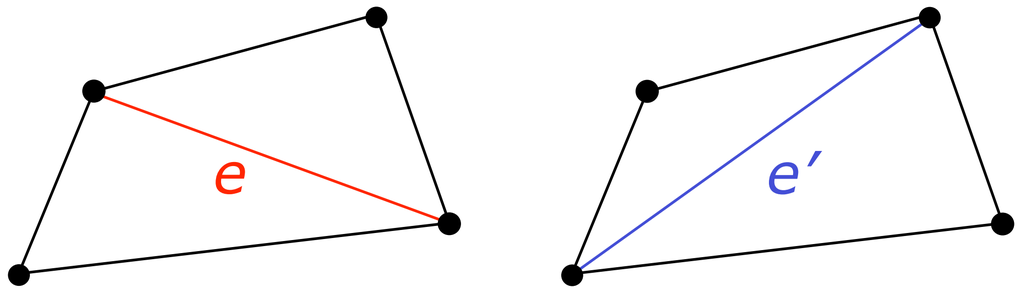
Simplification Operators

- **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



Mesh Data Structures

Data Structures

- 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

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

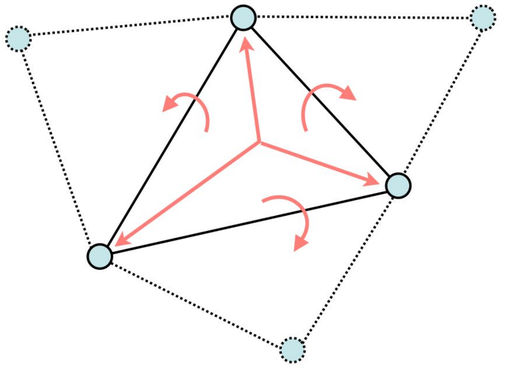
- Avoid replication of vertices - OBJ, OFF, PLY files
- Maintain a vector of vertices and a vector of faces
  - For each vertex: position
  - For each face: references to positions of its vertices (FV) in the vector
  - in general, the number of vertices of each face must also be stored
  - number implicit 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}

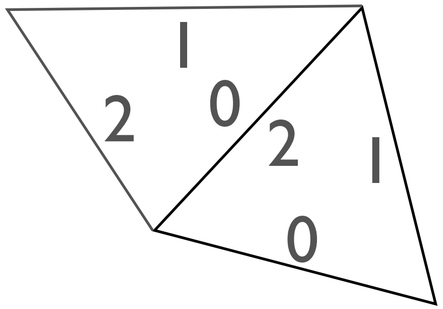
Indexed Structure with Adjacencies

- 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)



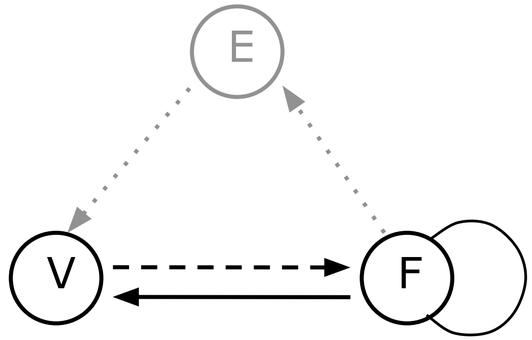
Indexed Structure with Adjacencies

- 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



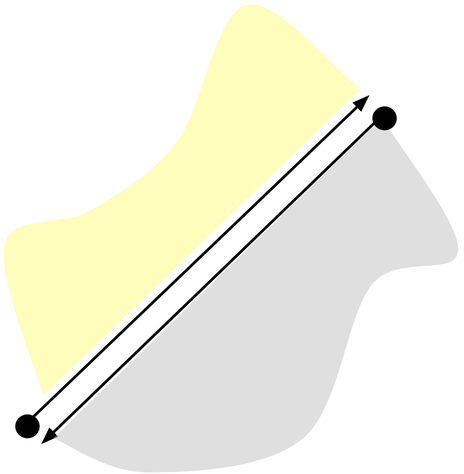
Indexed Structure with Adjacencies

- 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



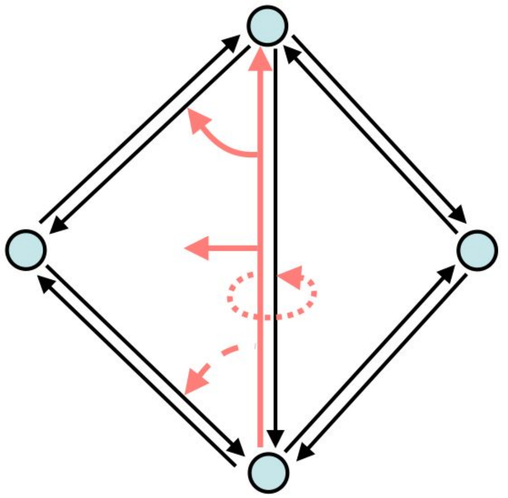
Half-Edge Data Structure

- 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

- 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*)



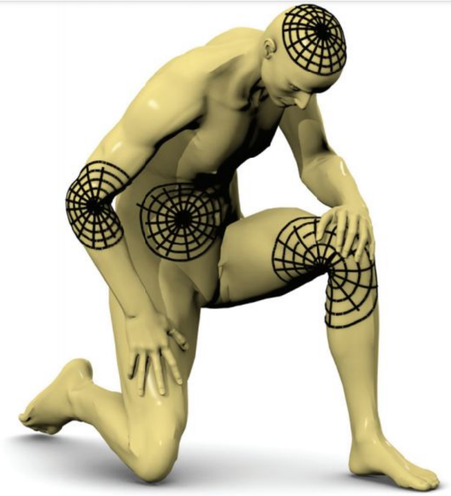
Half-Edge Data Structure

- 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

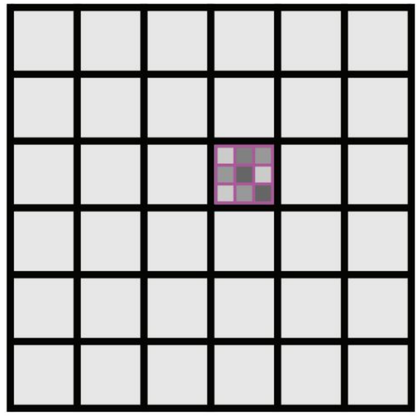
Mesh Neural Networks

Formulations of non-Euclidean CNNs

Spectral domain

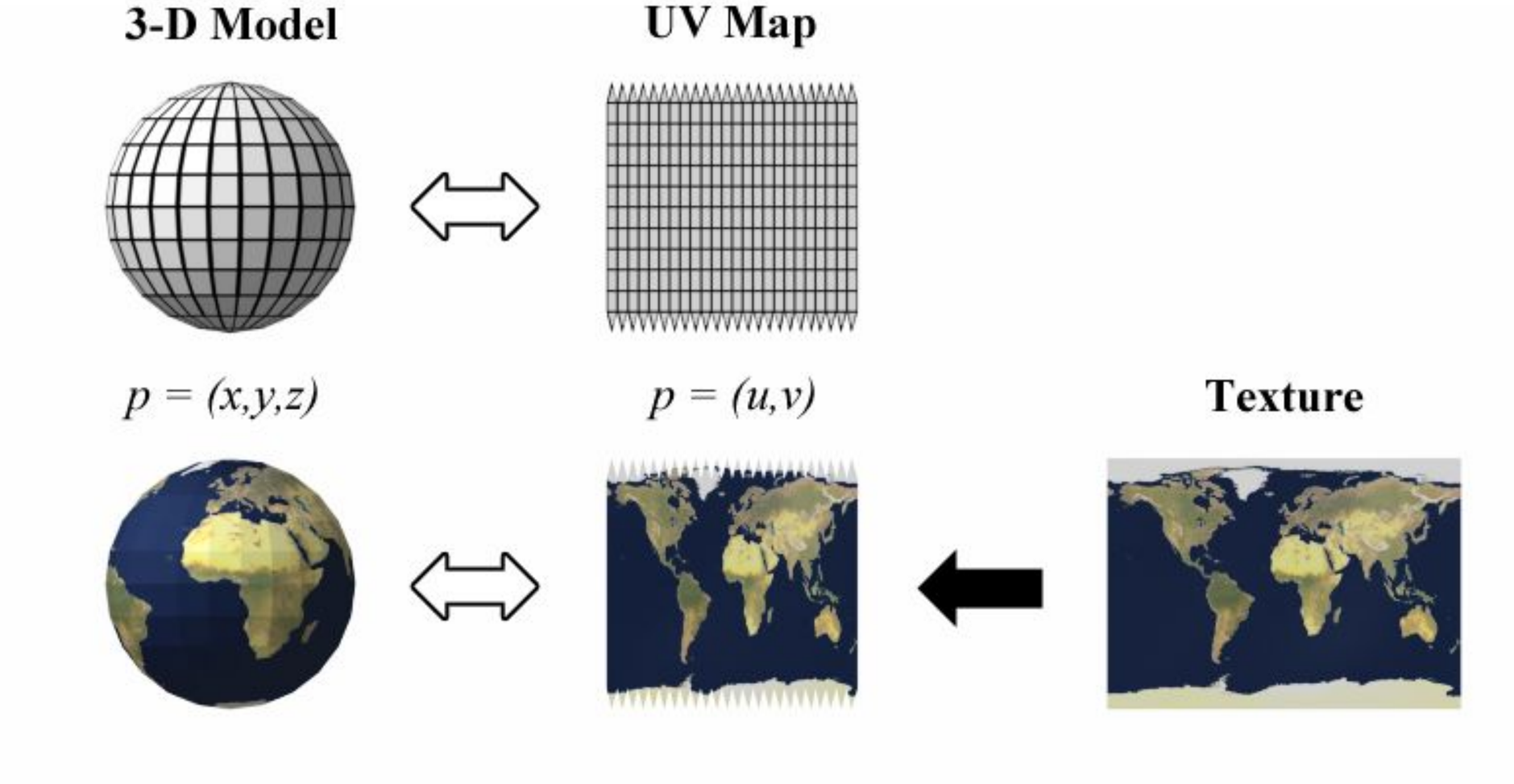


Spatial domain

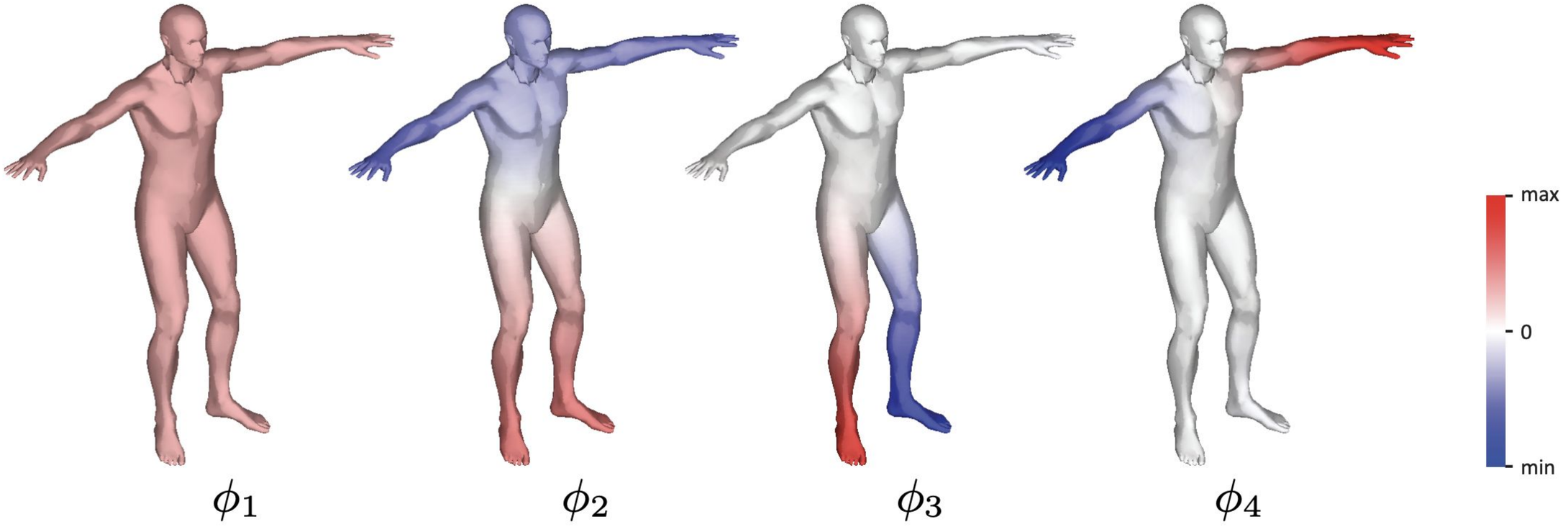


Parametric domain

Non-Euclidean CNNs via conventional CNNs



Laplacian eigenfunctions and eigenvalues



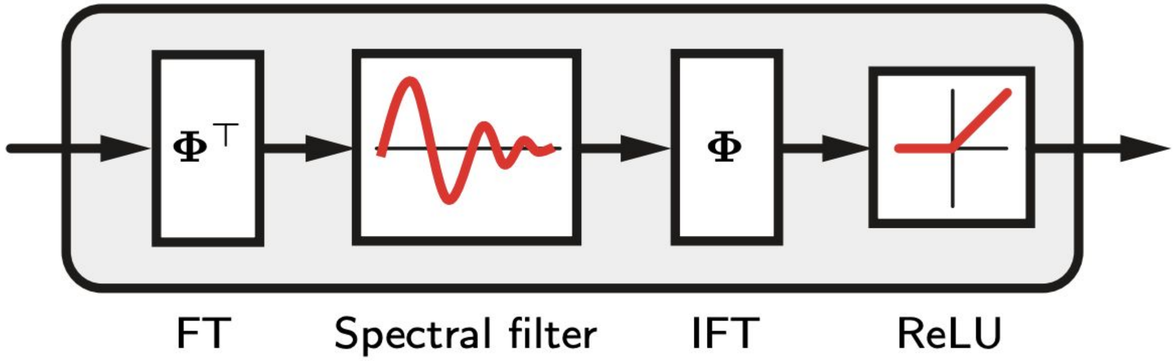
First eigenfunctions of a manifold Laplacian

Spectral CNN (2014)

Convolution expressed in the spectral domain:

$$\mathbf{g} = \mathbf{\Phi W \Phi}^\top \mathbf{f}$$

where $\mathbf{W}$ is n x n diagonal matrix of learnable spectral filter coefficients



Convolutional filter of a Spectral CNN

Joan Bruna et al. Spectral Networks and Locally Connected Networks on Graphs

Spectral CNN with polynomial filters (2016)

Represent spectral transfer function as a polynomial of order r

$$\hat{g}_{\boldsymbol{\theta}}(\boldsymbol{\Delta}) = \sum_{\ell=0}^{r} \theta_\ell \boldsymbol{\Delta}^\ell$$

where $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_r)^\top$ > is the vector of filter parameters

- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ No explicit computation of $\boldsymbol{\Phi}^\top, \boldsymbol{\Phi} \Rightarrow \mathcal{O}(nr)$ complexity
- ☺ Filters have guaranteed r-ring support
- ☹ Mesh dependent

Defferrard M., Bresson X., Vandergheynst P. Convolutional neural networks on graphs with fast localized spectral filtering

Spectral CNN with rational filters (2017)

Represent spectral transfer function as a Cayley polynomial of order r

$$\hat{g}_{\mathbf{c},h}(\lambda) = c_0 + 2\mathrm{Re}\Big\{ \sum_{\ell=1}^{r} c_\ell (h\lambda - \mathrm{i})^\ell (h\lambda + \mathrm{i})^{-\ell} \Big\}$$

where the filter parameters are the vector of real/complex coefficients $\mathbf{c} = (c_0, \ldots, c_r)^\dot{\top}$ and the spectral zoom h

Levie R. et al. Cayleynets: Graph convolutional neural networks with complex rational spectral filters

Spectral CNN with rational filters (2017)

- ☺ $\mathcal{O}(1)$ parameters per layer
- ☺ Filters have guaranteed exponential spatial decay
- ☺ Spectral zoom property allowing to better localize in frequency
- ☺ Richer class of filters than polynomials for the same order
- ☺ Scale invariance
- ☺ $\mathcal{O}(n)$ computational complexity with **Jacobi approximate matrix inversion**

Levie R. et al. Cayleynets: Graph convolutional neural networks with complex rational spectral filters

Spatial convolutions on meshes

Local system of coordinates u_{ij} around i (e.g. geodesic polar)

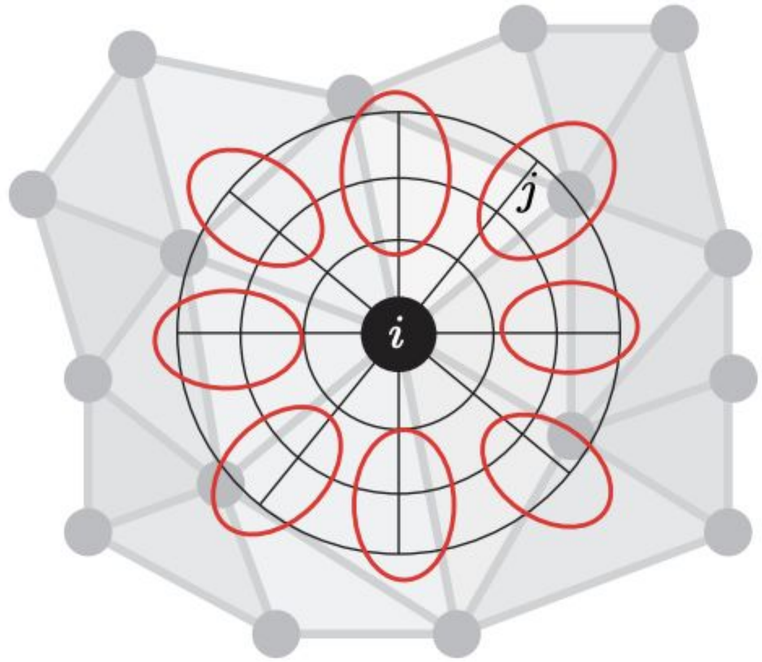
Local weights $w_1(\mathbf{u}),\ldots,w_L(\mathbf{u})$ w.r.t. $\mathbf{u}$, e.g. Gaussians

$$w_\ell = \exp\left(-(\mathbf{u}-\boldsymbol{\mu}_\ell)^\top \boldsymbol{\Sigma}_\ell^{-1}(\mathbf{u}-\boldsymbol{\mu}_\ell)\right)$$

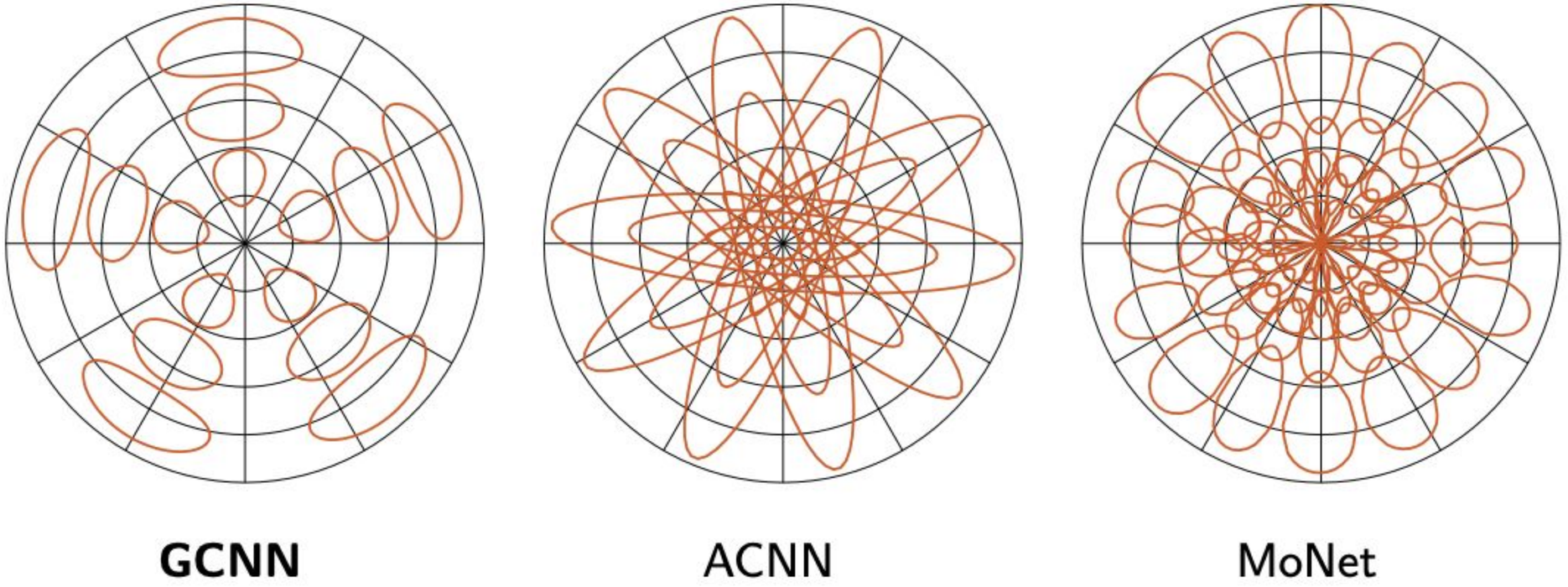
Spatial convolution with filter g

$$\mathbf{x}'_i \propto \sum_{\ell=1}^{L} g_\ell \underbrace{\sum_{j=1}^{n} w_\ell(\mathbf{u}_{ij})\mathbf{x}_j}_{\text{patch operator}}$$

where $\mathbf{x}_i \in \mathbb{R}^d$ is feature at vertex i



Patch operator weight functions



Masci et al. 2015 (GCNN); Boscaini et al. 2016 (ACNN); Monti et al. 2017 (MoNet)

MoNet (2017)

Geodesic polar coordinates $\mathbf{u}_{ij} = (\rho_{ij}, \theta_{ij})$

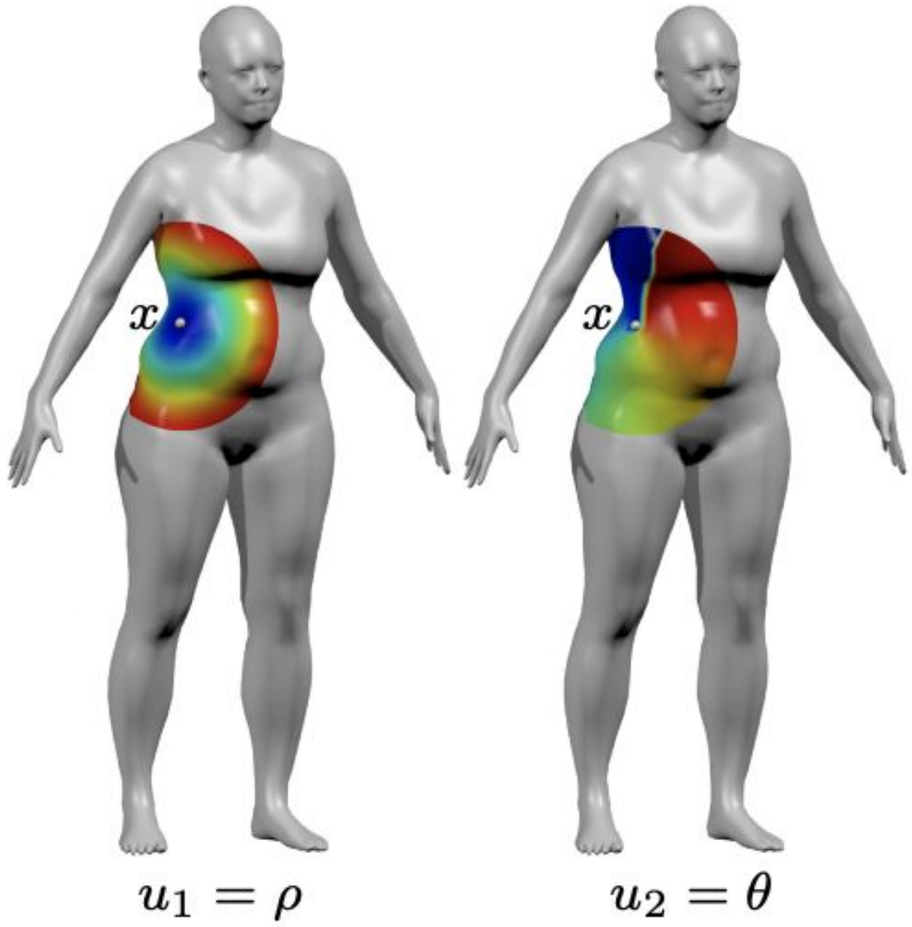
Gaussian weighting functions

$$w_{\boldsymbol{\mu},\boldsymbol{\Sigma}}(\mathbf{u}) = \exp\left(-\tfrac{1}{2}(\mathbf{u}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{u}-\boldsymbol{\mu})\right)$$

with learnable covariance $\boldsymbol{\Sigma}$ and mean $\boldsymbol{\mu}$

Spatial convolution

$$\mathbf{x}'_i \propto \sum_{j=1}^{n} \underbrace{\sum_{\ell=1}^{L} g_\ell w_{\boldsymbol{\mu}_\ell,\boldsymbol{\Sigma}_\ell}(\mathbf{u}_{ij})}_{\text{Gaussian mixture}} \mathbf{x}_j$$



Monti F. et al. Geometric deep learning on graphs and manifolds using mixture model cnns

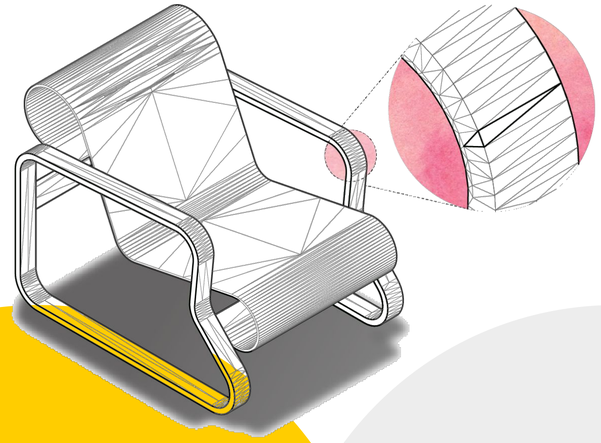
MoNet as a generalization

Method	**Coordinates** $\mathbf{u}_{ij}$	**Weight function** $w_{\boldsymbol{\Theta}}(\mathbf{u})$
CNN[1]	$\mathbf{u}_j - \mathbf{u}_i$	$\delta(\mathbf{u} - \mathbf{v})$
GCNN[2]	ρ_{ij}, θ_{ij}	$\exp\left(-\frac{1}{2}(\mathbf{u}-\mathbf{v})^\top \begin{pmatrix} \sigma_\rho^2 & \\ & \sigma_\theta^2 \end{pmatrix}^{-1} (\mathbf{u}-\mathbf{v})\right)$
ACNN[3]	ρ_{ij}, θ_{ij}	$\exp\left(-t\mathbf{u}^\top \mathbf{R}_\varphi \begin{pmatrix} \alpha & \\ & 1 \end{pmatrix} \mathbf{R}_\varphi^\top \mathbf{u}\right)$
MoNet[4]	ρ_{ij}, θ_{ij}	$\exp\left(-\frac{1}{2}(\mathbf{u}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{u}-\boldsymbol{\mu})\right)$
		learnable parameters $\boldsymbol{\Theta} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$

Some CNN models can be considered as particular settings of MoNet with weighting functions of different form

Methods: 1 LeCun et al. 1998; 2 Masci et al. 2015; 3 Boscaini et al. 2016; 4 Monti et al. 2016

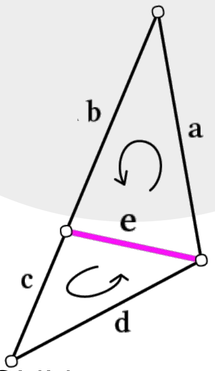
MeshCNN (2019)

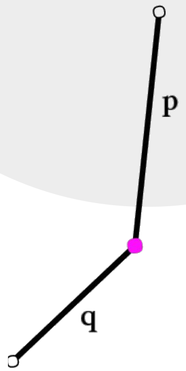


Mesh Convolution

MeshCNN

Mesh Pooling





Rana Hanocka et al. MeshCNN

Mesh Convolution

Face normal

➔ Consistent ordering in each face

Two *valid* orderings

➔ (a, b, c, d) or (c, d, a, b)

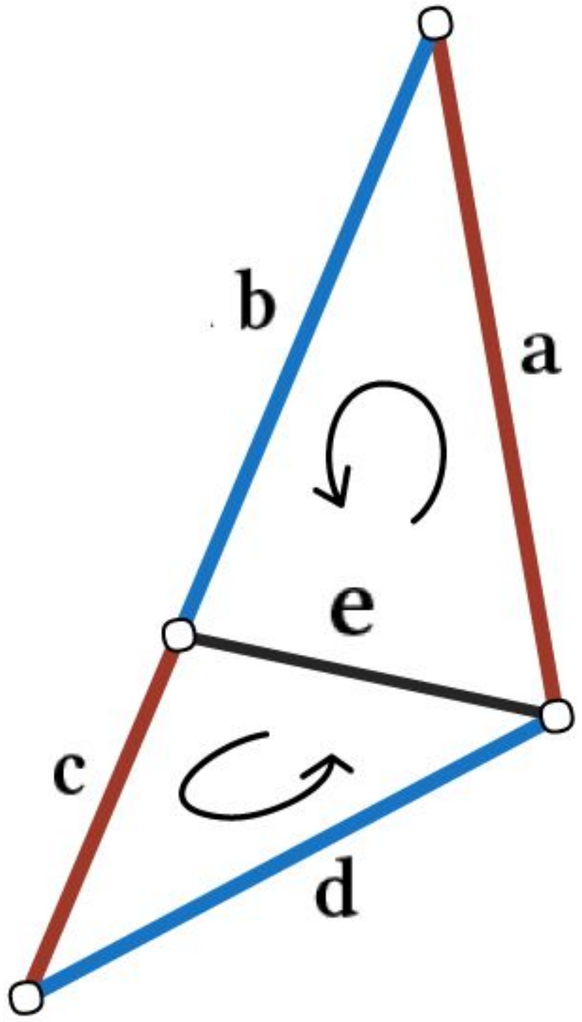
How to handle ambiguity?

Build symmetric features

➔ e-> (a+c, |a-c|, b+d, |b-d|)

Conv Filter

➔ (k_0, k_1, k_2, k_3, k_4)



Rana Hanocka et al. MeshCNN

Mesh Pooling

Delete edge with smallest feature activations

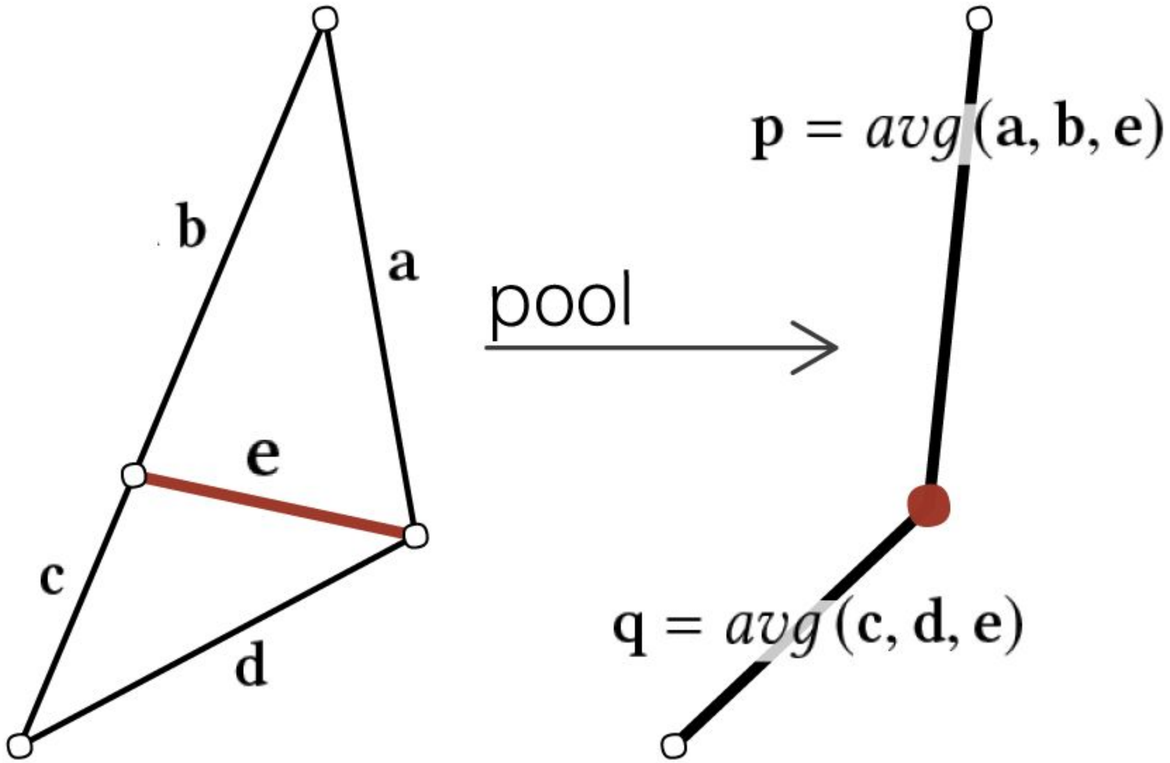
➔ Aggregate features (average pooling)

➔ Update topology

Average pooling aggregation

➔ 5 edges to 2 edges

➔ Average per-channel



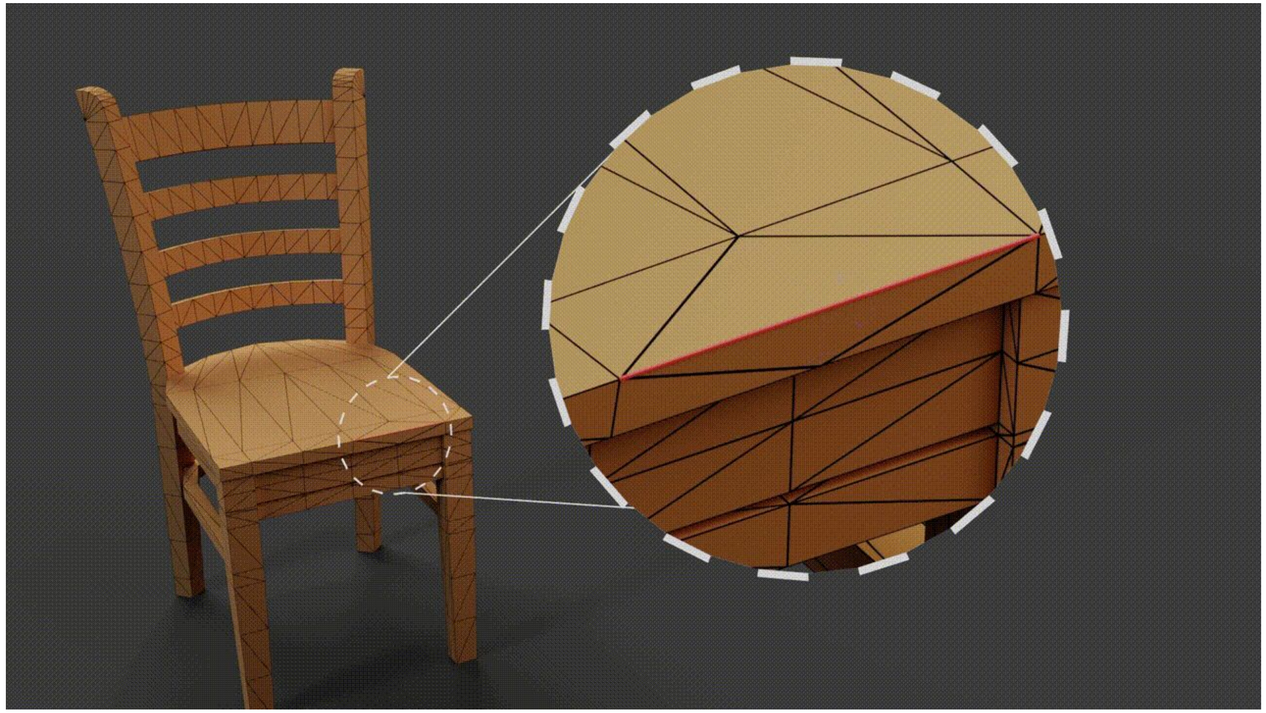
Rana Hanocka et al. MeshCNN

Update Mesh Topology

New mesh is generated

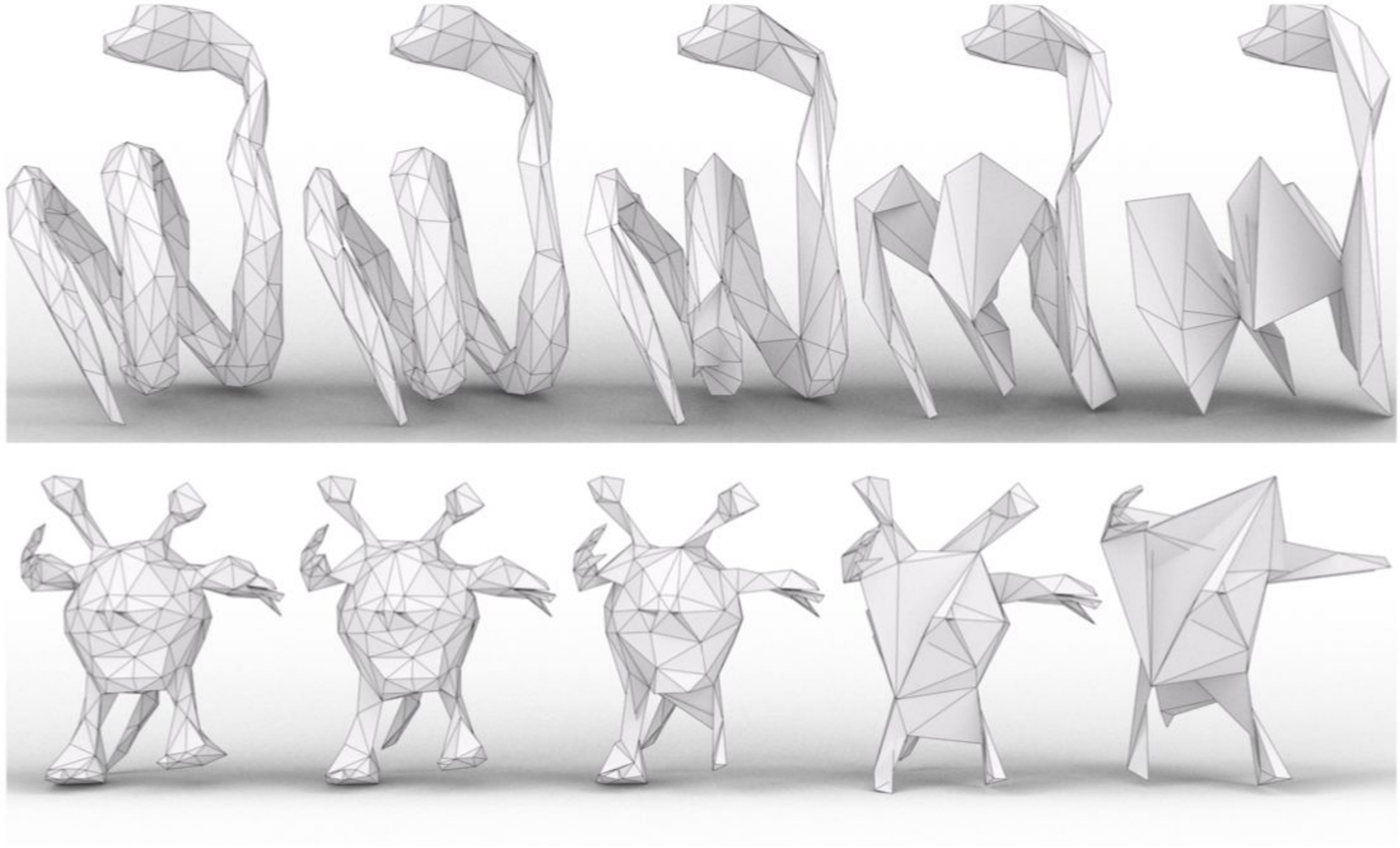
➔ Defines new neighbors in next convolution

➔ Update relevant data structures



Rana Hanocka et al. MeshCNN

Common pooling results



Rana Hanocka et al. M

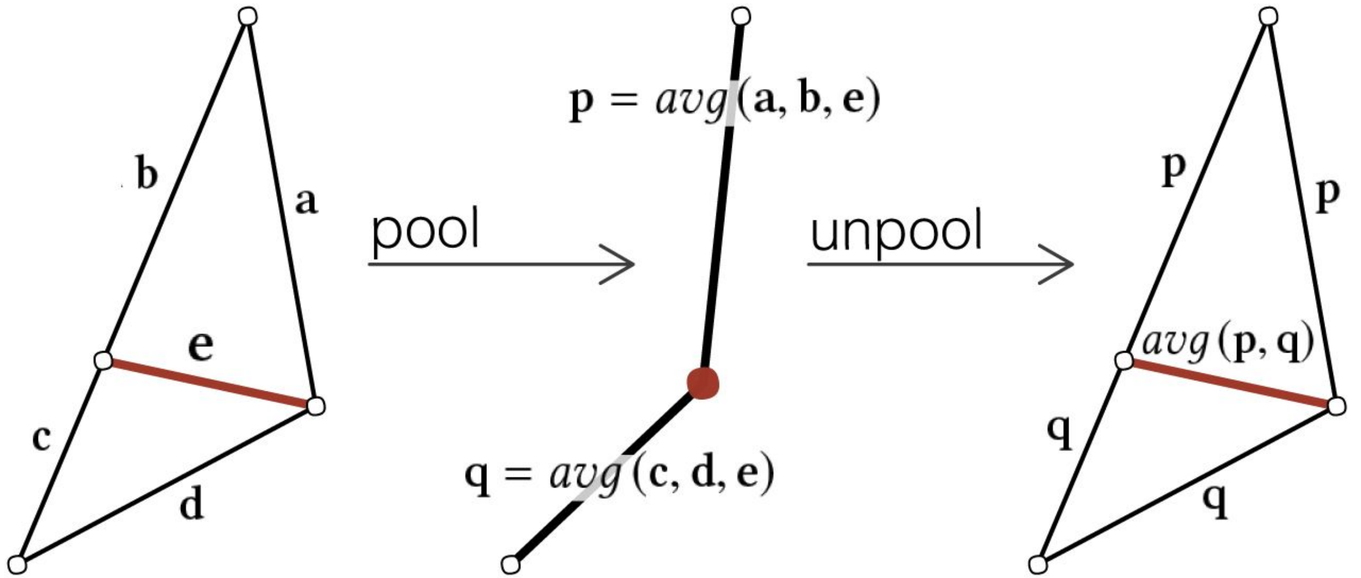
input 600 450 300 150

Mesh Unpooling

Partial Inverse of Pooling

➔ Restores upsampled topology (reversible)

➔ Unpooled features weighted combination of pooled features



Rana Hanocka et al. MeshCNN

Key Attributes of Mesh Pooling

- ➔ Spatially adapts to the task
- ➔ Network decides collapse
- ➔ Flexible input mesh resolution
- ➔ Strengthens the learned feature representation
- ➔ Visual insights from the network

Rana Hanocka et al. MeshCNN

Next part: Spiral CNN

See the notebook!

Rana Hanocka et al. MeshCNN