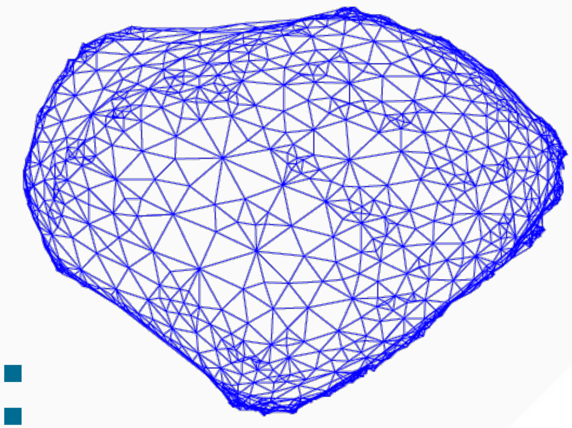
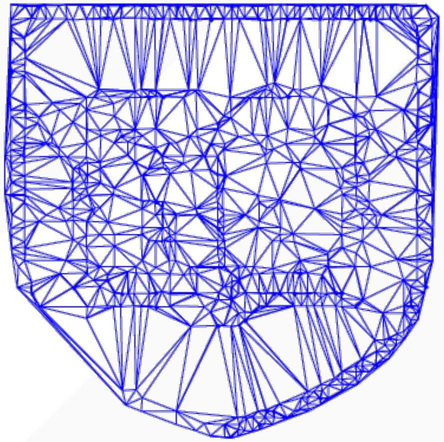
UC San Diego



Lecture 14/15:

Graph Laplacian



Instructor: Hao Su

Feb 27, 2018

Slides ack: Leo Guidas, Radu Horaud, Dan Spielman
http://perception.inrialpes.fr/
http://www.cs.yale.edu/homes/spielman/561/

Agenda

- Some Guidelines for the Final Project
- Graph Laplacian Theory

Agenda

- **Some Guidelines for the Final Project**

- Graph Laplacian Theory

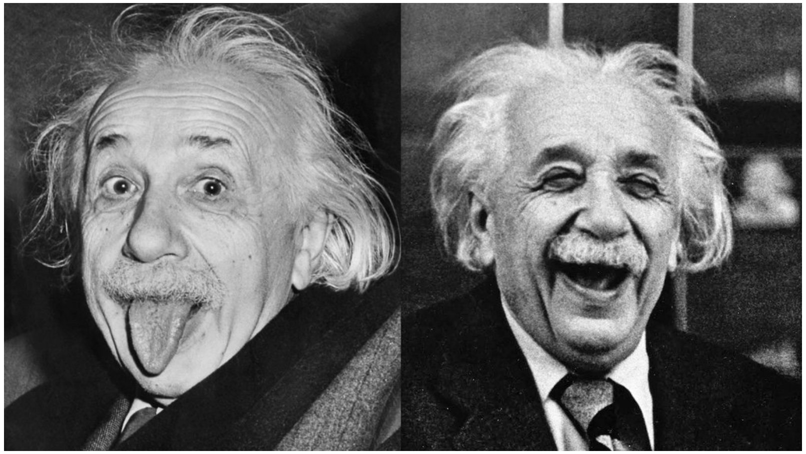
Topic Selection

- Can be analytical
  - **Systematically** analyze when a published work would fail
  - **Conclude** the causes or provide bounds
  - **Suggest** possible improvements

- Can be Algorithmic
  - **Propose** a new idea based upon existing work
  - Or, **combine** the best of existing approaches
  - Or, **improve** the “state-of-the-art” with solid experiments

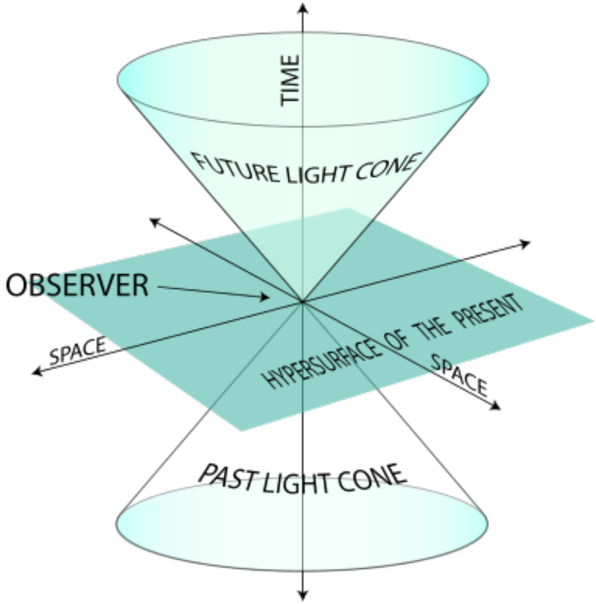
Characteristics of a Good Research

- Surprising results/discoveries
  length contraction, time dilation, mass–energy equivalence, relativistic mass, a universal speed limit and relativity of simultaneity



- Inspiring to others that will breed follow-up work

"theory of special relativity"



Possible Strategies towards Good Research

- **Sharp**
  - Well-defined problem so that everyone understands
  - Tactically designed setting so that
    - Crisp conclusion is reachable
    - But still generalizable to broad cases

- **Simple** on paper, but **sophisticated** in mind
  - Simple so that extensible
  - Need extensive experiments and sufficient reasoning to find the simple setting and solution

How to do Experiments?

- Experiments are the log of **conclusions,** but **not numbers**
- Take iterations — from simple to complicated
  - Simple enough to build understanding and form solid **conclusions**
  - Make small but solid steps to expand
- Simple means:
  - Small data, to allow more iterations
  - Synthetic data, so that you can control variables
  - e.g. Point Set Generation Network

Scoring Rubric of the Project

- Based upon your presentation and write-up
- Novelty
  - problem, approach, discovery
- Intellectual depth
  - technical strength
- The key is to show your "**commitment**" and "**understanding**" to the problem and results
  - Can be incomplete upon deadline
  - As long as you can **insightfully** explain the motivation, idea, approach, and progress

Schedule for Final Presentation

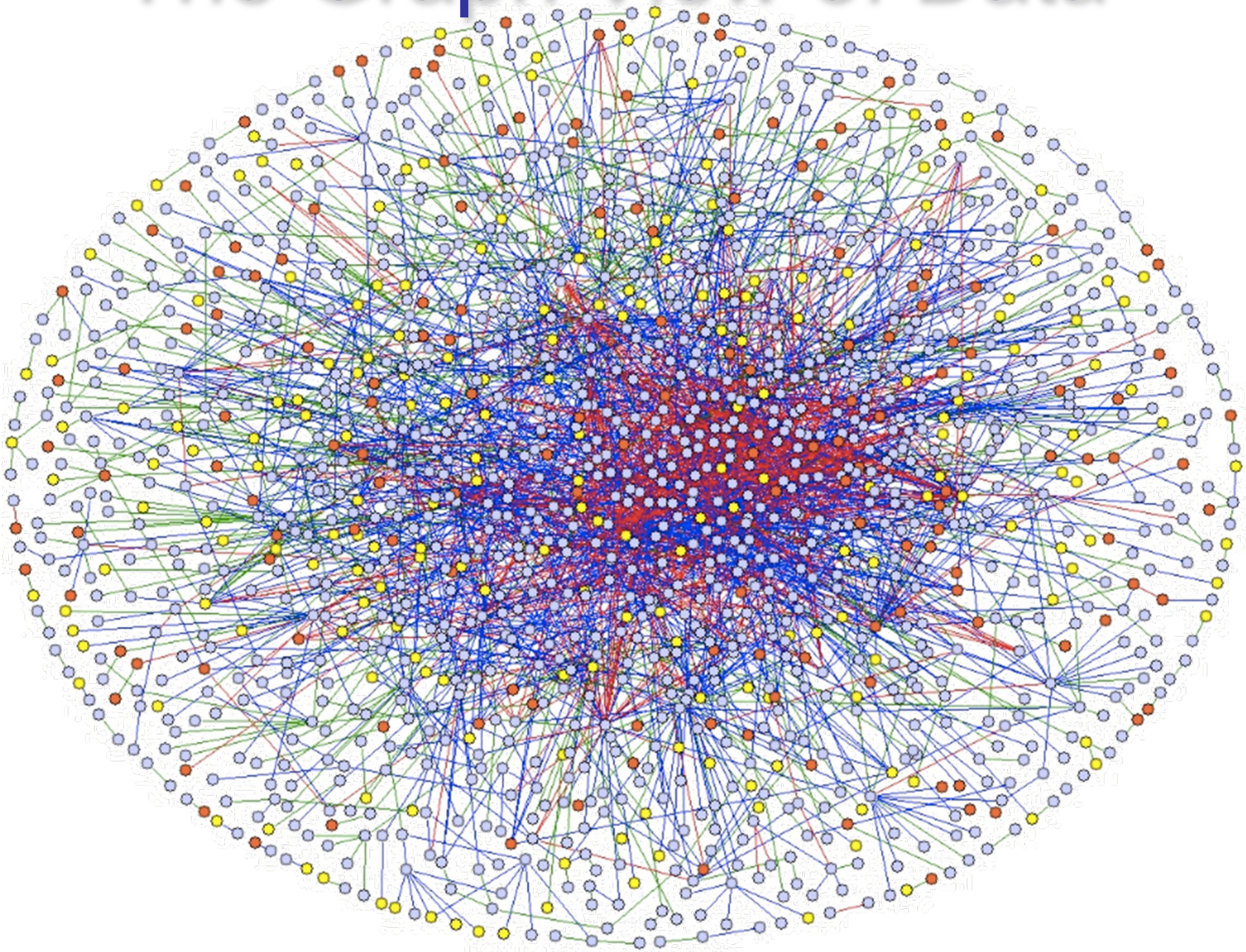
- Time: March 20, 2018, 3:00pm to 7:00pm
  - If you have any conflict with the schedule, let me know in advance no later than March 15
- Form: TBD
  - Presentation only (~15 min for each team)
  - Or spotlight presentation (~5 min) + poster session
- Three best papers will be generated

Agenda

- Some Guidelines for the Final Project

- **Graph Laplacian Theory**

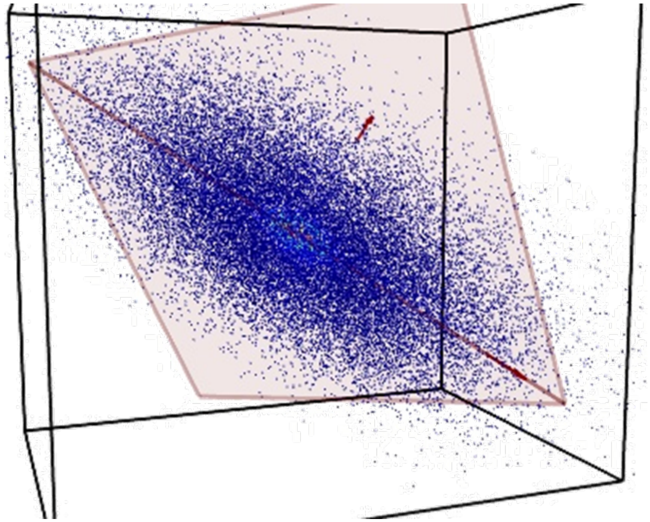
The Graph View of Data

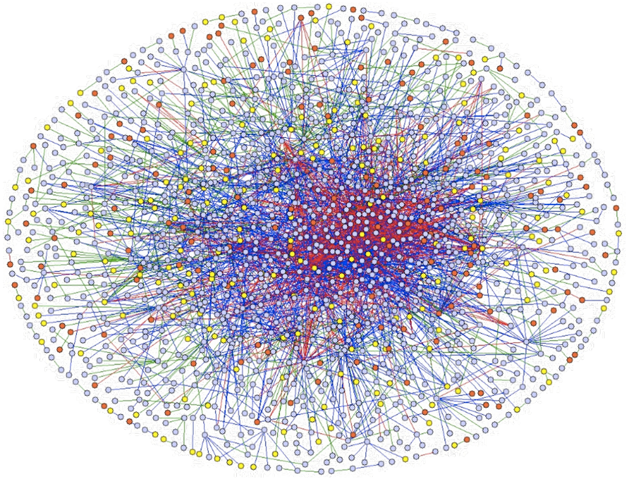


Social Networks

Connect Points in R^d and Graph Views of Data

- Points in R^d
  - via near-neighbor graphs
- Graph
  - via matrix representations of graphs





Spectral Graph Theory

- The *spectral graph theory* studies the properties of graphs via the eigenvalues and eigenvectors of their associated graph matrices: the *adjacency matrix* and the *graph Laplacian* and its variants.
- Both matrices have been extremely well studied from an algebraic point of view.
- The Laplacian allows a natural link between discrete representations, such as graphs, and continuous representations, such as vector spaces and manifolds.
- The most important application of the Laplacian is *spectral clustering* that corresponds to a computationally tractable solution to the *graph partitionning problem*.
- Another application is *spectral matching* that solves for *graph matching*.

More Applications

- *Spectral partitioning*: automatic circuit placement for VLSI (Alpert et al 1999), image segmentation (Shi & Malik 2000),
- *Text mining and web applications*: document classification based on semantic association of words (Lafon & Lee 2006), collaborative recommendation (Fouss et al. 2007), text categorization based on reader similarity (Kamvar et al. 2003).
- *Manifold analysis*: Manifold embedding, manifold learning, mesh segmentation, etc.

Graph Notations and Definitions

We consider *simple graphs* (no multiple edges or loops), $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$:

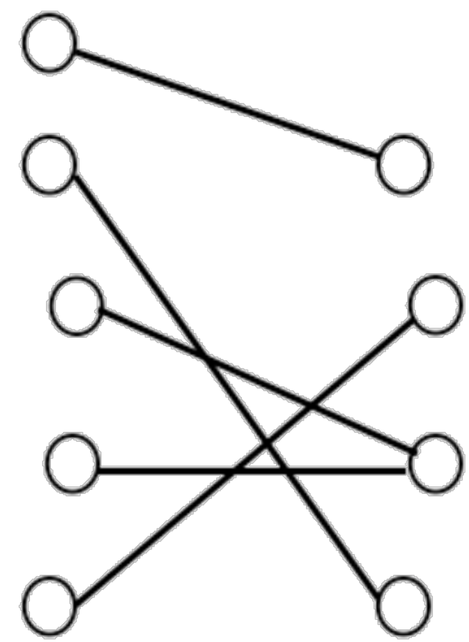
- $\mathcal{V}(\mathcal{G}) = \{v_1, \ldots, v_n\}$ is called the *vertex set* with $n = |\mathcal{V}|$;
- $\mathcal{E}(\mathcal{G}) = \{e_{ij}\}$ is called the *edge set* with $m = |\mathcal{E}|$;
- An edge e_{ij} connects vertices v_i and v_j if they are adjacent or neighbors. One possible notation for adjacency is $v_i \sim v_j$;
- The number of neighbors of a node v is called the *degree* of v and is denoted by $d(v)$, $d(v_i) = \sum_{v_i \sim v_j} e_{ij}$. If all the nodes of a graph have the same degree, the graph is *regular*; The nodes of an *Eulerian* graph have even degree.
- A graph is *complete* if there is an edge between every pair of vertices.

Subgraphs

- $\mathcal{H}$ is a *subgraph* of $\mathcal{G}$ if $\mathcal{V}(\mathcal{H}) \subseteq \mathcal{V}(\mathcal{G})$ and $\mathcal{E}(\mathcal{H}) \subseteq \mathcal{E}(\mathcal{G})$;
- a subgraph $\mathcal{H}$ is an *induced subgraph* of $\mathcal{G}$ if two vertices of $\mathcal{V}(\mathcal{H})$ are adjacent if and only if they are adjacent in $\mathcal{G}$.
- A *clique* is a complete subgraph of a graph.
- A *path* of k vertices is a sequence of k distinct vertices such that consecutive vertices are adjacent.
- A *cycle* is a connected subgraph where every vertex has exactly two neighbors.
- A graph containing no cycles is a *forest*. A connected forest is a *tree*.

k-Partite Graphs

- A graph is called *k-partite* if its set of vertices admits a partition into k classes such that the vertices of the same class are not adjacent.
- An example of a *bipartite* graph.

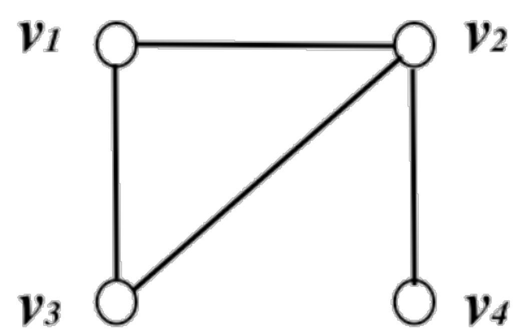


Adjacency Matrices

- For a graph with n vertices, the entries of the $n \times n$ adjacency matrix are defined by:

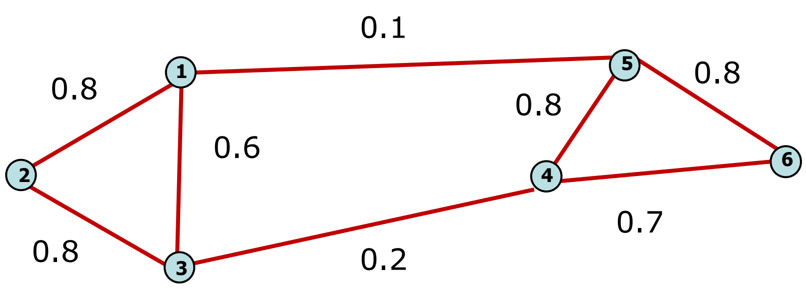
$$\mathbf{A} := \begin{cases} A_{ij} = 1 & \text{if there is an edge } e_{ij} \\ A_{ij} = 0 & \text{if there is no edge} \\ A_{ii} = 0 & \end{cases}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$



Weighted Matrices

- Adjacency matrix (A)
  - n x n matrix
  - $A = [w_{ij}]$ edge weight between vertex x_i and x_j



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0.8	0.6	0	0.1	0
x_2	0.8	0	0.8	0	0	0
x_3	0.6	0.8	0	0.2	0	0
x_4	0	0	0.2	0	0.8	0.7
x_5	0.1	0	0	0.8	0	0.8
x_6	0	0	0	0.7	0.8	0

- Important properties:
  - Symmetric matrix

  $\Rightarrow$ Eigenvalues are <u>real</u>

  $\Rightarrow$ Eigenvector could span <u>orthogonal base</u>

Eigenvalues and Eigenvectors

- $\mathbf{A}$ is a real-symmetric matrix: it has n real eigenvalues and its n real eigenvectors form an orthonormal basis.
- Let $\{\lambda_1, \ldots, \lambda_i, \ldots, \lambda_r\}$ be the set of *distinct* eigenvalues.
- The eigenspace S_i contains the eigenvectors associated with λ_i:

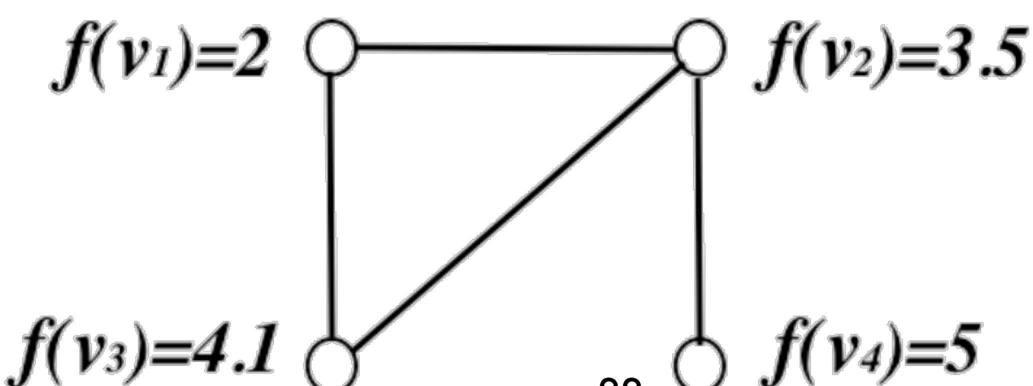
$$S_i = \{\boldsymbol{x} \in \mathbb{R}^n | \mathbf{A}\boldsymbol{x} = \lambda_i \boldsymbol{x}\}$$

- For real-symmetric matrices, the algebraic multiplicity is equal to the geometric multiplicity, for all the eigenvalues.
- The dimension of S_i (geometric multiplicity) is equal to the multiplicity of λ_i.
- If $\lambda_i \neq \lambda_j$ then S_i and S_j are mutually orthogonal.

Order the eigenvalues from small to large

Functions on Graphs

- We consider real-valued functions on the set of the graph's vertices, $\boldsymbol{f} : \mathcal{V} \longrightarrow \mathbb{R}$. Such a function assigns a real number to each graph node.
- $\boldsymbol{f}$ is a vector indexed by the graph's vertices, hence $\boldsymbol{f} \in \mathbb{R}^n$.
- Notation: $\boldsymbol{f} = (f(v_1), \dots, f(v_n)) = (f(1), \dots, f(n))$.
- The eigenvectors of the adjacency matrix, $\mathbf{A}\boldsymbol{x} = \lambda\boldsymbol{x}$, can be viewed as *eigenfunctions*.



Operators and Quadratic Forms

- The adjacency matrix can be viewed as an operator

$$\boldsymbol{g} = \mathbf{A}\boldsymbol{f}; g(i) = \sum_{i \sim j} f(j)$$

- It can also be viewed as a quadratic form:

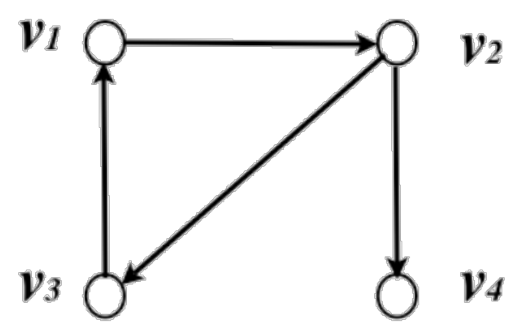
$$\boldsymbol{f}^\top \mathbf{A}\boldsymbol{f} = \sum_{e_{ij}} f(i) f(j)$$

Incidence Matrix

- Let each edge in the graph have an arbitrary but fixed orientation;
- The incidence matrix of a graph is a $|\mathcal{E}| \times |\mathcal{V}|$ $(m \times n)$ matrix defined as follows:

$$\nabla := \begin{cases} \nabla_{ev} = -1 & \text{if } v \text{ is the initial vertex of edge } e \\ \nabla_{ev} = 1 & \text{if } v \text{ is the terminal vertex of edge } e \\ \nabla_{ev} = 0 & \text{if } v \text{ is not in } e \end{cases}$$

$$\nabla = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & +1 \end{bmatrix}$$



Discrete Differential Operator

- The mapping $\boldsymbol{f} \longrightarrow \bigtriangledown \boldsymbol{f}$ is known as the *co-boundary mapping* of the graph.
- $(\bigtriangledown \boldsymbol{f})(e_{ij}) = f(v_j) - f(v_i)$

$$\begin{pmatrix} f(2) - f(1) \\ f(1) - f(3) \\ f(3) - f(2) \\ f(4) - f(2) \end{pmatrix} = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & +1 \end{bmatrix} \begin{pmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \end{pmatrix}$$

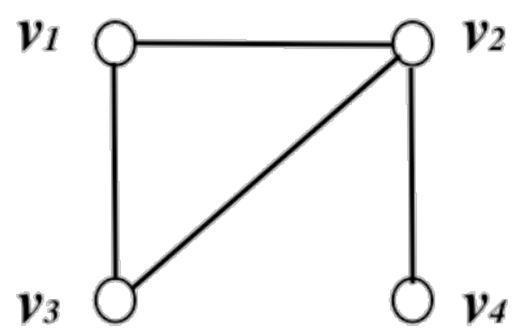
Graph (Unnormalized) Laplacian

- $\mathbf{L} = \nabla^\top \nabla$
- $(\mathbf{L}\boldsymbol{f})(v_i) = \sum_{v_j \sim v_i} (f(v_i) - f(v_j))$
- Connection between the Laplacian and the adjacency matrices:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

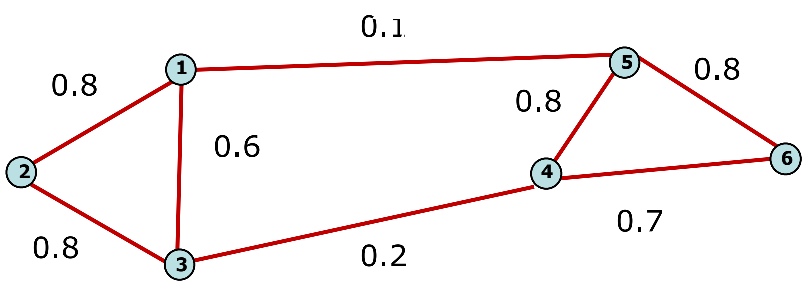
- The degree matrix: $\mathbf{D} := D_{ii} = d(v_i)$.

$$\mathbf{L} = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$



Degree Matrix

- **Degree matrix (D)**
  - $n\ x\ n$ diagonal matrix
  - $D(i,i) = \sum_j w_{ij}$: total weight of edges incident to vertex x_i



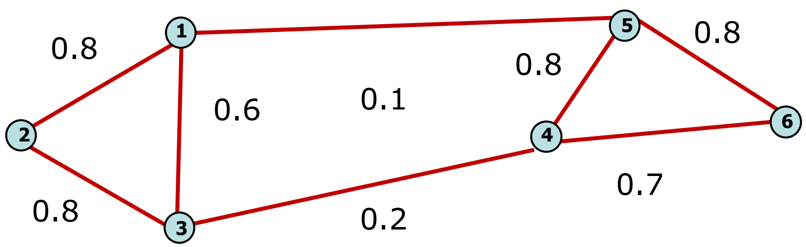
	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1.5	0	0	0	0	0
x_2	0	1.6	0	0	0	0
x_3	0	0	1.6	0	0	0
x_4	0	0	0	1.7	0	0
x_5	0	0	0	0	1.7	0
x_6	0	0	0	0	0	1.5

- Important application:
  - Normalize adjacency matrix

Laplacian Matrix

- **Laplacian matrix (L)**
  - $n \ x \ n$ symmetric matrix

$$L = D - A$$



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1.5	-0.8	-0.6	0	-0.1	0
x_2	-0.8	1.6	-0.8	0	0	0
x_3	-0.6	-0.8	1.6	-0.2	0	0
x_4	0	0	-0.2	1.7	-0.8	-0.7
x_5	-0.1	0	0	0.8-	1.7	-0.8
x_6	0	0	0	-0.7	-0.8	1.5

- Important properties:
  - Eigenvalues are non-negative real numbers (Gershgorin circle theorem)
  - Eigenvectors are real and orthogonal
  - Eigenvalues and eigenvectors provide an insight into the connectivity of the graph…

Laplacian Defines Natural Quadratic Form of Graphs

$$x^T L x = \sum_{(i,j)\in E} (x(i) - x(j))^2$$

$L = D - A$ where D is diagonal matrix of degrees

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

Undirected Weighted Graphs

- We consider *undirected weighted graphs*: Each edge e_{ij} is weighted by $w_{ij} > 0$.
- The Laplacian as an operator:

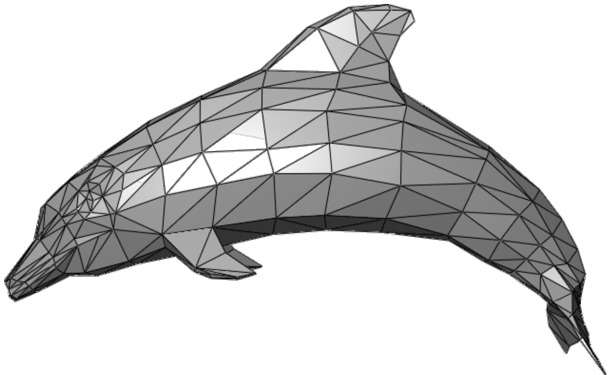
$$(\mathbf{L}\boldsymbol{f})(v_i) = \sum_{v_j \sim v_i} w_{ij}(f(v_i) - f(v_j))$$

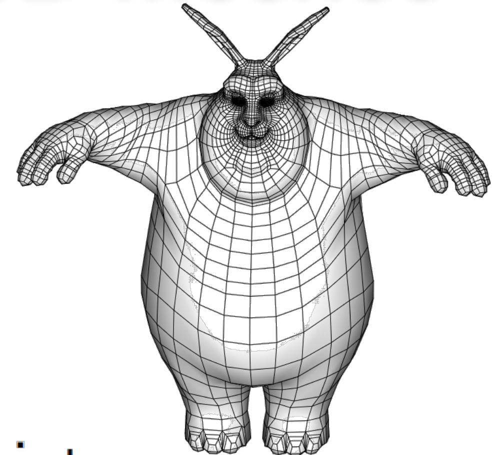
- As a quadratic form:

$$\boldsymbol{f}^\top \mathbf{L}\boldsymbol{f} = \frac{1}{2}\sum_{e_{ij}} w_{ij}(f(v_i) - f(v_j))^2$$

- $\mathbf{L}$ is symmetric and positive semi-definite.
- $\mathbf{L}$ has n non-negative, real-valued eigenvalues: $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$.

Discrete Surface Laplacians: 3D Meshes





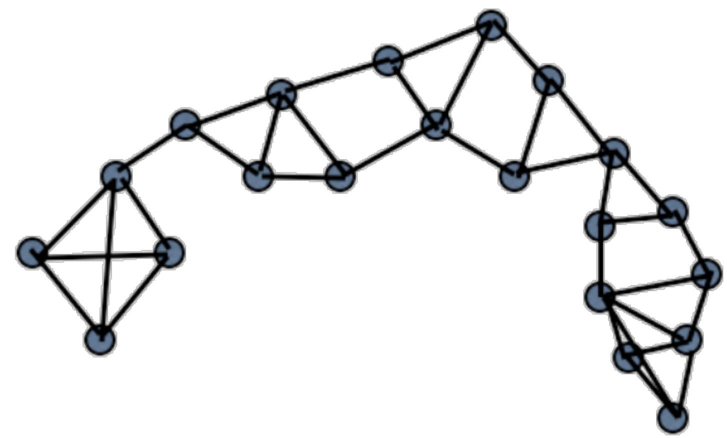
- A graph vertex v_i is associated with a 3D point $\boldsymbol{v}_i$.
- The weight of an edge e_{ij} is defined by the Gaussian kernel:

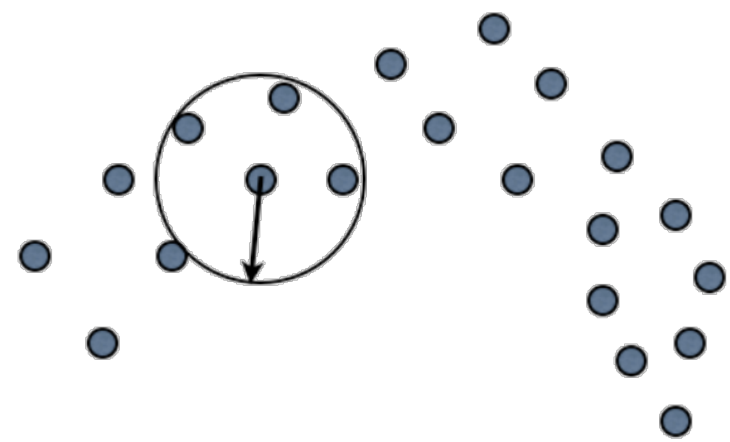
$$w_{ij} = \exp\left(-\|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2 / \sigma^2\right)$$

- $0 \leq w_{\min} \leq w_{ij} \leq w_{\max} \leq 1$
- Hence, the geometric structure of the mesh is encoded in the weights.
- Other weighting functions were proposed in the literature.

Point Cloud Laplacians

- 3-nearest neighbor graph
- ε-radius graph
- KNN may guarantee that the graph is connected (depends on the implementation)
- ε-radius does not guarantee that the graph has one connected component





Connected Graph Laplacians

- $\mathbf{L}\boldsymbol{u} = \lambda\boldsymbol{u}$.
- $\mathbf{L}\mathbf{1}_n = \mathbf{0}$, $\lambda_1 = 0$ is the smallest eigenvalue.
- The *one* vector: $\mathbf{1}_n = (1 \dots 1)^\top$.
- $0 = \boldsymbol{u}^\top \mathbf{L}\boldsymbol{u} = \sum_{i,j=1}^{n} w_{ij}(u(i) - u(j))^2$.
- If any two vertices are connected by a path, then $\boldsymbol{u} = (u(1), \dots, u(n))$ needs to be constant at all vertices such that the quadratic form vanishes. Therefore, a graph with one connected component has the constant vector $\boldsymbol{u}_1 = \mathbf{1}_n$ as the only eigenvector with eigenvalue 0.

A Graph with *k* Connected Components

- Each connected component has an associated Laplacian. Therefore, we can write matrix $\mathbf{L}$ as a *block diagonal matrix*:

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_1 & & \\ & \ddots & \\ & & \mathbf{L}_k \end{bmatrix}$$

- The spectrum of $\mathbf{L}$ is given by the union of the spectra of $\mathbf{L}_i$.
- Each block corresponds to a connected component, hence each matrix $\mathbf{L}_i$ has an eigenvalue 0 with multiplicity 1.
- The spectrum of $\mathbf{L}$ is given by the union of the spectra of $\mathbf{L}_i$.
- The eigenvalue $\lambda_1 = 0$ has multiplicity k.

The Eigenspace of $\lambda_1 = 0$

- The eigenspace corresponding to $\lambda_1 = \ldots = \lambda_k = 0$ is spanned by the k mutually orthogonal vectors:

$$\boldsymbol{u}_1 = \mathbf{1}_{L_1}$$
$$\ldots$$
$$\boldsymbol{u}_k = \mathbf{1}_{L_k}$$

- with $\mathbf{1}_{L_i} = (0000111110000)^\top \in \mathbb{R}^n$
- These vectors are the *indicator vectors* of the graph's connected components.
- Notice that $\mathbf{1}_{L_1} + \ldots + \mathbf{1}_{L_k} = \mathbf{1}_n$

The Fiedler Vector

- The first non-null eigenvalue λ_{k+1} is called the Fiedler value.
- The corresponding eigenvector $\boldsymbol{u}_{k+1}$ is called the Fiedler vector.
- The multiplicity of the Fiedler eigenvalue is always equal to 1.
- The Fiedler value is the *algebraic connectivity of a graph*, the further from 0, the more connected.
- The Fidler vector has been extensively used for *spectral bi-partioning*
- Theoretical results are summarized in Spielman & Teng 2007: `http://cs-www.cs.yale.edu/homes/spielman/`

Laplacian Eigenvectors for Connected Graphs

- $\boldsymbol{u}_1 = \mathbf{1}_n, \mathbf{L}\mathbf{1}_n = \mathbf{0}$.
- $\boldsymbol{u}_2$ is the *the Fiedler vector* with multiplicity 1.
- The eigenvectors form an orthonormal basis: $\boldsymbol{u}_i^\top \boldsymbol{u}_j = \delta_{ij}$.
- For any eigenvector $\boldsymbol{u}_i = (\boldsymbol{u}_i(v_1) \ldots \boldsymbol{u}_i(v_n))^\top$, $2 \leq i \leq n$:

$$\boldsymbol{u}_i^\top \mathbf{1}_n = 0$$

- Hence the components of $\boldsymbol{u}_i$, $2 \leq i \leq n$ satisfy:

$$\sum_{j=1}^{n} \boldsymbol{u}_i(v_j) = 0$$

- Each component is bounded by:

$$-1 < \boldsymbol{u}_i(v_j) < 1$$

λ_2 = algebraic connectivity, monotone under graph inclusion

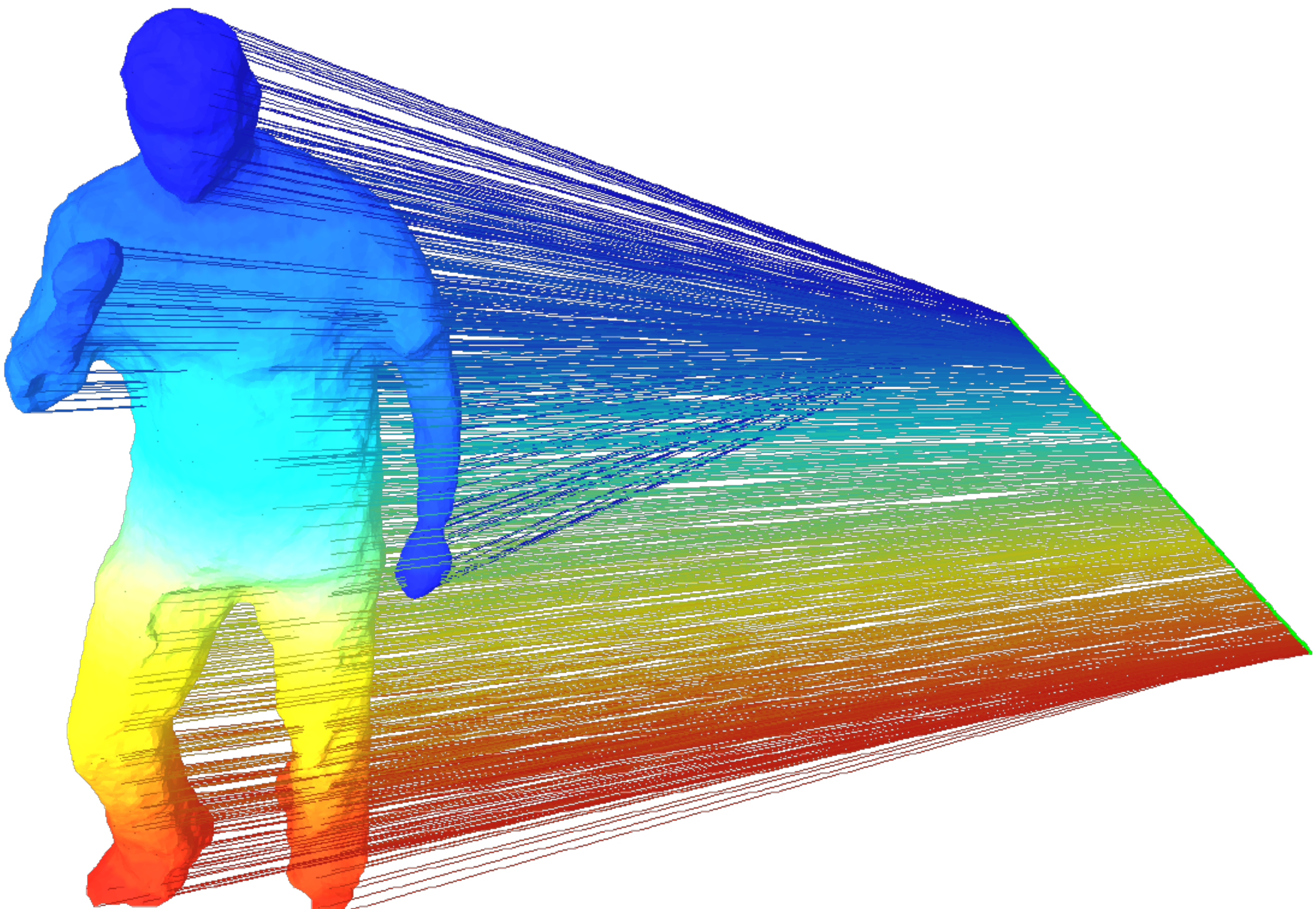
1-d Laplacian Embedding

- Map a weighted graph onto a line such that connected nodes stay as close as possible, i.e., minimize $\sum_{i,j=1}^{n} w_{ij}(f(v_i) - f(v_j))^2$, or:

$$\arg\min_{\boldsymbol{f}} \boldsymbol{f}^\top \mathbf{L} \boldsymbol{f} \text{ with: } \boldsymbol{f}^\top \boldsymbol{f} = 1 \text{ and } \boldsymbol{f}^\top \mathbf{1} = 0$$

- The solution is the eigenvector associated with the smallest nonzero eigenvalue of the eigenvalue problem: $\mathbf{L}\boldsymbol{f} = \lambda \boldsymbol{f}$, namely the Fiedler vector $\boldsymbol{u}_2$.
- For more details on this minimization see Golub & Van Loan *Matrix Computations*, chapter 8 (The symmetric eigenvalue problem).

1-d Embedding Example



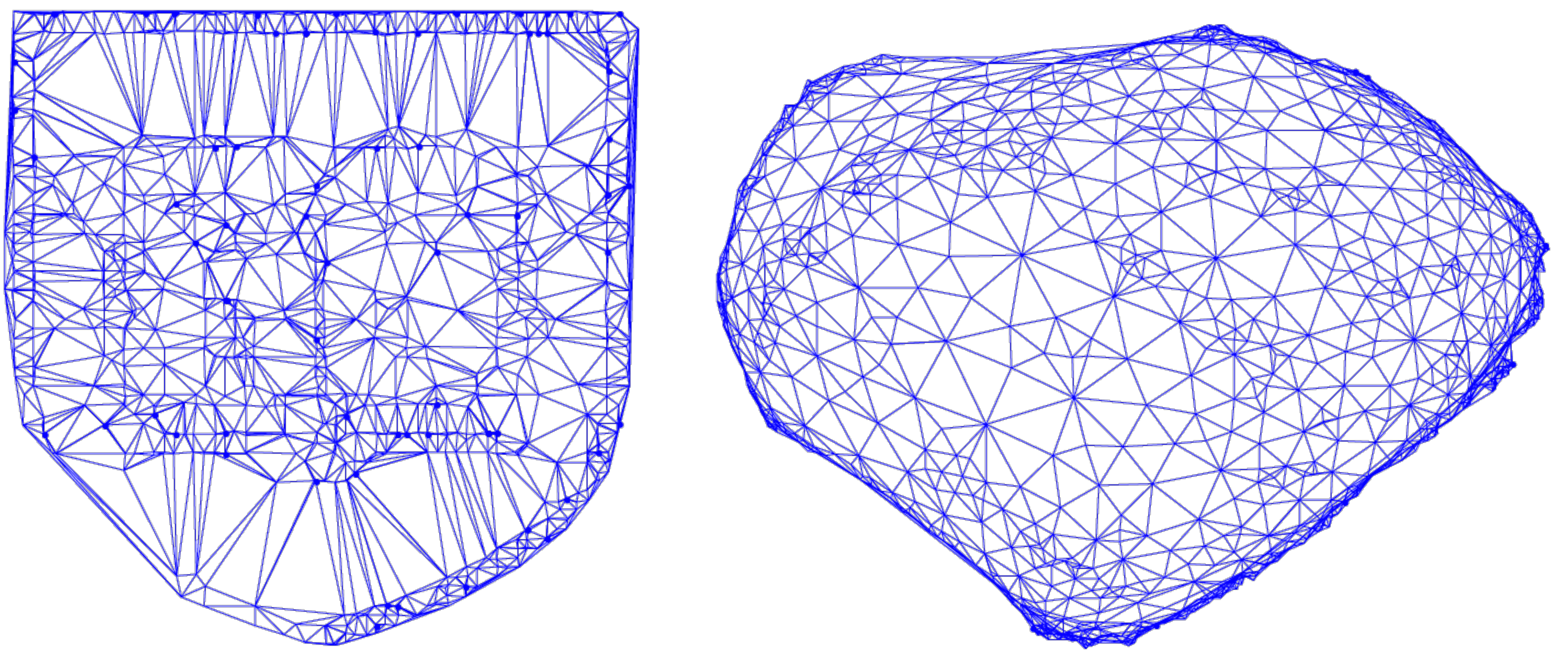
Higher-d Embeddings

- Embed the graph in a k-dimensional Euclidean space. The embedding is given by the $n \times k$ matrix $\mathbf{F} = [\boldsymbol{f}_1 \boldsymbol{f}_2 \ldots \boldsymbol{f}_k]$ where the i-th row of this matrix – $\boldsymbol{f}^{(i)}$ – corresponds to the Euclidean coordinates of the i-th graph node v_i.
- We need to minimize (Belkin & Niyogi '03):

$$\arg \min_{\boldsymbol{f}_1 \cdots \boldsymbol{f}_k} \sum_{i,j=1}^{n} w_{ij} \|\boldsymbol{f}^{(i)} - \boldsymbol{f}^{(j)}\|^2 \text{ with: } \mathbf{F}^\top \mathbf{F} = \mathbf{I}.$$

- The solution is provided by the matrix of eigenvectors corresponding to the k lowest nonzero eigenvalues of the eigenvalue problem $\mathbf{L}\boldsymbol{f} = \lambda \boldsymbol{f}$.

2-d Embeddings



Spectral Graph Drawing

Condition for eigenvector $Lx = \lambda x$

Gives $x(i) = \frac{1}{d_i - \lambda} \sum_{j \sim i} x(j)$ for all i

λ small says x(i) near average of neighbors

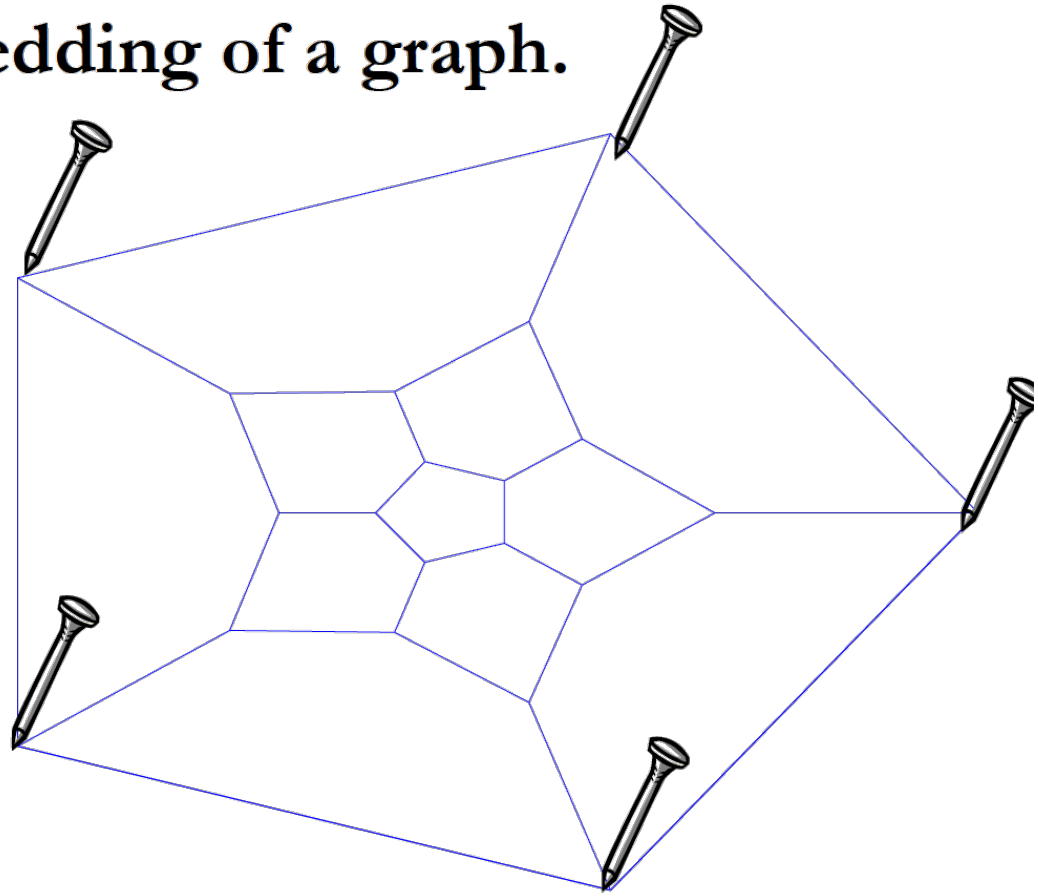
Tutte '63: If fix outside face, and let every other vertex be average of neighbors, get planar embedding of planar graph.

Tutte Embedding

Tutte '63 embedding of a graph.

Fix outside face.
Edges -> springs.

Vertex at center
of mass of nbrs.



3-connected -> get planar embedding

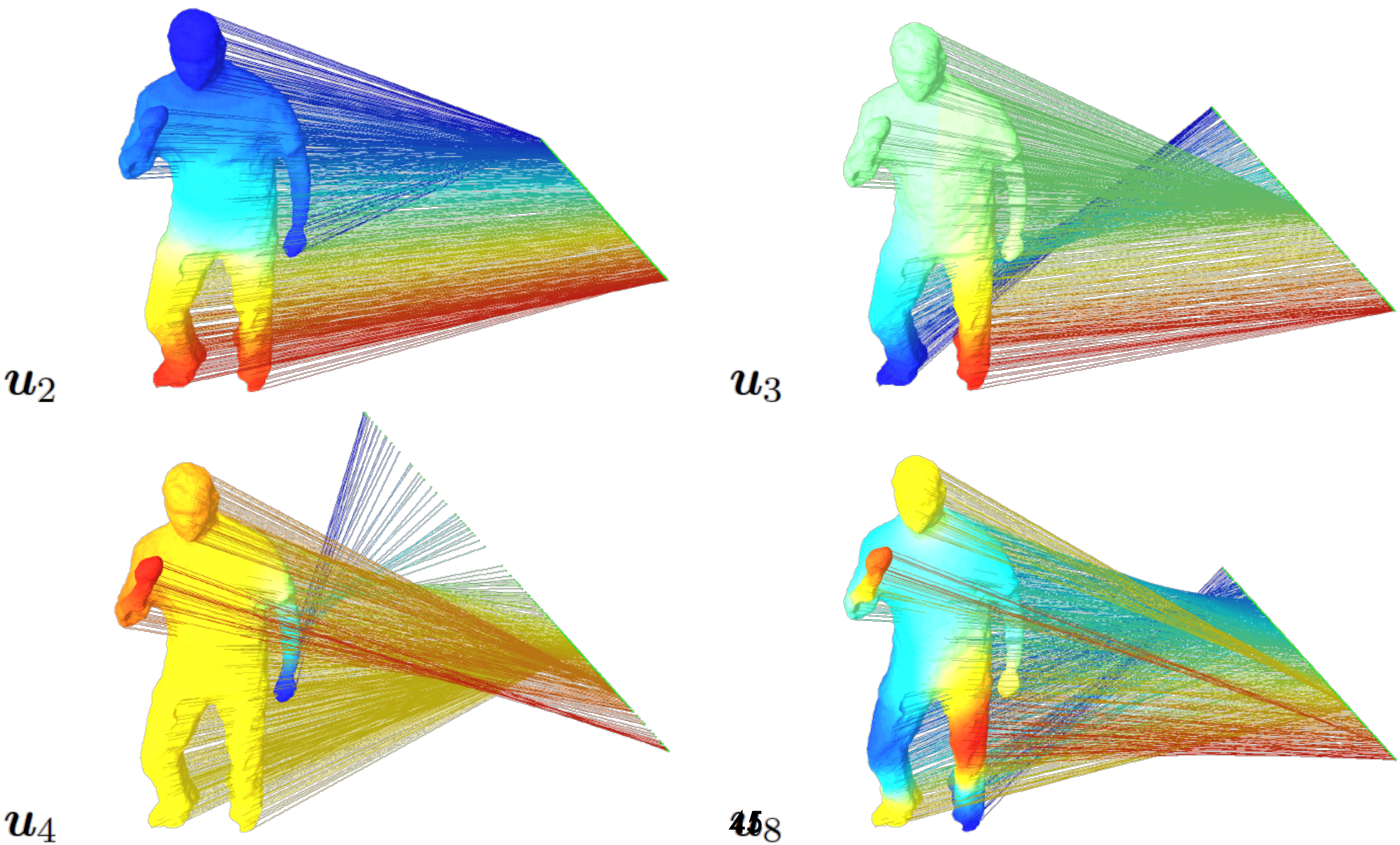
Spectral Embedding Using Unnormalized Laplacian

- Compute the eigendecomposition $\mathbf{L} = \mathbf{D} - \mathbf{A}$.
- Select the k smallest non-null eigenvalues $\lambda_2 \leq \ldots \leq \lambda_{k+1}$
- $\lambda_{k+2} - \lambda_{k+1} =$ **eigengap**.
- We obtain the $n \times k$ matrix $\mathbf{U} = [\boldsymbol{u}_2 \ldots \boldsymbol{u}_{k+1}]$:

$$\mathbf{U} = \begin{bmatrix} \boldsymbol{u}_2(v_1) & \ldots & \boldsymbol{u}_{k+1}(v_1) \\ \vdots & & \vdots \\ \boldsymbol{u}_2(v_n) & \ldots & \boldsymbol{u}_{k+1}(v_n) \end{bmatrix}$$

- $\boldsymbol{u}_i^\top \boldsymbol{u}_j = \delta_{ij}$ (orthonormal vectors), hence $\mathbf{U}^\top \mathbf{U} = \mathbf{I}_k$.
- Column i $(2 \leq i \leq k+1)$ of this matrix is a mapping on the eigenvector $\boldsymbol{u}_i$.

More Eigenvectors, More 1-d Embeddings



The Normalized Spectral Embedding of a Graph

- (Euclidean) $\mathbf{L}$-embedding of a graph:

$$\mathbf{X} = \mathbf{\Lambda}_k^{-\frac{1}{2}} \mathbf{U}^\top = [\boldsymbol{x}_1 \ \ldots \ \boldsymbol{x}_j \ \ldots \ \boldsymbol{x}_n]$$

  The coordinates of a vertex v_j are:

$$\boldsymbol{x}_j = \begin{pmatrix} \frac{\boldsymbol{u}_2(v_j)}{\sqrt{\lambda_2}} \\ \vdots \\ \frac{\boldsymbol{u}_{k+1}(v_j)}{\sqrt{\lambda_{k+1}}} \end{pmatrix}$$

Why the Scaling?

Both

- the *commute-time distance* (CTD) and
- the *principal-component analysis* of a graph (graph PCA)

are two important concepts; They allow to reason "statistically" on a graph. They are both associated with the *unnormalized* Laplacian matrix.

Commute-Time Distance (CTD)

- The CTD is a well known quantity in Markov chains;
- It is the average number of (weighted) edges that it takes, starting at vertex v_i, to randomly reach vertex v_j for the first time and go back;
- The CTD decreases as the number of connections between the two nodes increases;
- It captures the connectivity structure of a small graph volume rather than a single path between the two vertices – such as the shortest-path geodesic distance.
- The CTD can be computed in closed form:

$$\text{CTD}^2(v_i, v_j) = \text{vol}(\mathcal{G})\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$$

Graph PCA

- The mean (remember that $\sum_{j=1}^{n} \boldsymbol{u}_i(v_j) = 0$):

$$\overline{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_j = \boldsymbol{\Lambda}_k^{-\frac{1}{2}} \begin{pmatrix} \sum_{j=1}^{n} \boldsymbol{u}_2(v_j) \\ \vdots \\ \sum_{j=1}^{n} \boldsymbol{u}_{k+1}(v_j) \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

- The covariance matrix:

$$\mathbf{S} = \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{x}_j \boldsymbol{x}_j^\top = \frac{1}{n}\mathbf{X}\mathbf{X}^\top = \frac{1}{n}\boldsymbol{\Lambda}_k^{-\frac{1}{2}}\mathbf{U}^\top\mathbf{U}\boldsymbol{\Lambda}_k^{-\frac{1}{2}} = \frac{1}{n}\boldsymbol{\Lambda}_k^{-1}$$

- The vectors $\boldsymbol{u}_2, \ldots, \boldsymbol{u}_{k+1}$ are the directions of *maximum variance* of the graph embedding, with $\lambda_2^{-1} \geq \ldots \geq \lambda_{k+1}^{-1}$.

Laplacian Variants

- The normalized graph Laplacian (symmetric and semi-definite positive):

$$\mathbf{L}_n = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$$

- The transition matrix (allows an analogy with Markov chains):

$$\mathbf{L}_t = \mathbf{D}^{-1}\mathbf{A}$$

- The random-walk graph Laplacian:

$$\mathbf{L}_r = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{L}_t$$

- These matrices are similar:

$$\mathbf{L}_r = \mathbf{D}^{-\frac{1}{2}}\mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}\mathbf{D}^{\frac{1}{2}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}_n\mathbf{D}^{\frac{1}{2}}$$

Eigenvectors/Eigenvalues for L_n, L_r

- $\mathbf{L}_r \boldsymbol{w} = \lambda \boldsymbol{w} \iff \mathbf{L}\boldsymbol{w} = \lambda \mathbf{D}\boldsymbol{w}$, hence:

$$\mathbf{L}_r : \quad \lambda_1 = 0; \quad \boldsymbol{w}_1 = \mathbf{1}$$

- $\mathbf{L}_n \boldsymbol{v} = \lambda \boldsymbol{v}$. By virtue of the similarity transformation between the two matrices:

$$\mathbf{L}_n : \quad \lambda_1 = 0 \quad \boldsymbol{v}_1 = \mathbf{D}^{\frac{1}{2}} \mathbf{1}$$

- More generally, the two matrices have the same eigenvalues:

$$0 = \lambda_1 \leq \ldots \leq \lambda_i \ldots \leq \lambda_n$$

- Their eigenvectors are related by:

$$\boldsymbol{v}_i = \mathbf{D}^{\frac{1}{2}} \boldsymbol{w}_i, \ \forall i = 1 \ldots n$$

Graph Partitioning

- **The graph-cut problem**: Partition the graph such that:
  1. Edges between groups have very low weight, and
  2. Edges within a group have high weight.

$$\text{cut}(A_1, \ldots, A_k) := \frac{1}{2}\sum_{i=1}^{k} W(A_i, \overline{A}_i) \text{ with } W(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

- **Ratio cut**: (Hagen & Kahng 1992)

$$\text{RatioCut}(A_1, \ldots, A_k) := \frac{1}{2}\sum_{i=1}^{k} \frac{W(A_i, \overline{A}_i)}{|A_i|}$$

- **Normalized cut**: (Shi & Malik 2000)

$$\text{NCut}(A_1, \ldots, A_k) := \frac{1}{2}\sum_{i=1}^{k} \frac{W(A_i, \overline{A}_i)}{\text{vol}(A_i)}$$

Spectral Clustering

- Both ratio-cut and normalized-cut minimizations are NP-hard problems
- Spectral clustering is a way to solve relaxed versions of these problems:
  1. The smallest non-null eigenvectors of the *unnormalized Laplacian* approximate the RatioCut minimization criterion, and
  2. The smallest non-null eigenvectors of the *random-walk Laplacian* approximate the NCut criterion.

Spectral Clustering Using the Random-Walk Laplacian

- For details see (von Luxburg '07)
- Input: Laplacian $\mathbf{L}_r$ and the number k of clusters to compute.
- Output: Cluster $C_1, \ldots, C_k$.

1. Compute $\mathbf{W}$ formed with the first k eigenvectors of the random-walk Laplacian.
2. Determine the spectral embedding $\mathbf{Y} = \mathbf{W}^\top$
3. Cluster the columns $\boldsymbol{y}_j, j = 1, \ldots, n$ into k clusters using the K-means algorithm.

k-Means Clustering

See Bishop'2006 (pages 424–428) for more details.

- What is a cluster: a group of points whose inter-point distance are small compared to distances to points outside the cluster.
- Cluster centers: $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$.
- Goal: find an assignment of points to clusters as well as a set of vectors $\boldsymbol{\mu}_i$.
- Notations: For each point $\boldsymbol{y}_j$ there is a *binary indicator variable* $r_{ji} \in \{0, 1\}$.
- Objective: minimize the following *distorsion measure*:

$$J = \sum_{j=1}^{n} \sum_{i=1}^{k} r_{ji} \|\boldsymbol{y}_j - \boldsymbol{\mu}_i\|^2$$

k-Means Algorithm

1. Initialization: Choose initial values for $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$.
2. First step: Assign the j-th point to the closest cluster center:

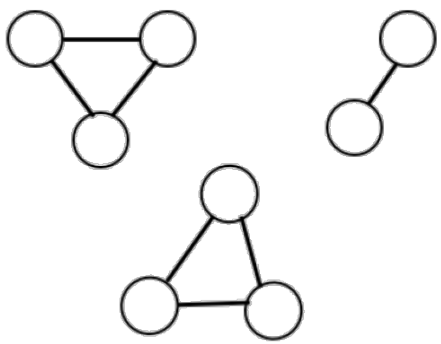
$$r_{ji} = \begin{cases} 1 & \text{if } i = \arg\min_l \|\boldsymbol{y}_j - \mu_l\|^2 \\ 0 & \text{otherwise} \end{cases}$$

3. Second Step: Minimize J to estimate the cluster centers:

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^{n} r_{ji} \boldsymbol{y}_j}{\sum_{j=1}^{n} r_{ji}}$$

4. Convergence: Repeat until no more change in the assignments.

Spectral Clustering: The Ideal Case

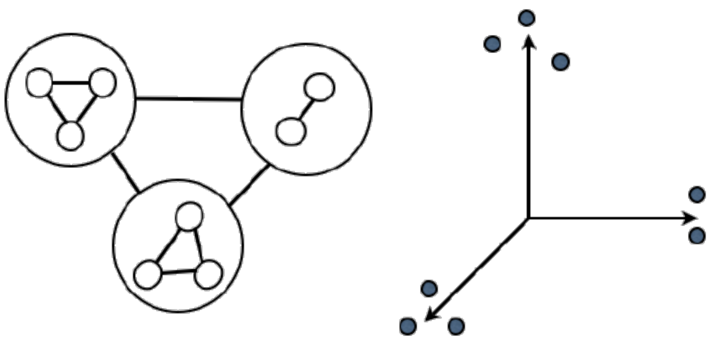


$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

- $\lambda_1 = \lambda_2 = \lambda_3 = 0$
- $\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3$ form an orthonormal basis.
- The connected components collapse to $(100), (010), (001)$.
- Clustering is trivial in this case.

$$\mathbf{Y} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Spectral Clustering: The Perturbed Case



- See (von Luxburg '07) for a detailed analysis.
- The connected components are no longer *disconnected*, but they are only connected by few edges with low weight.

- The Laplacian is a perturbed version of the ideal case.
- Choosing the first k nonzero eigenvalues is easier the larger the eigengap between λ_{k+1} and λ_{k+2}.
- The fact that the first k eigenvectors of the perturbed case are approximately piecewise constant depends on $|\lambda_{k+2} - \lambda_{k+1}|$.
- Choosing k is a crucial issue.

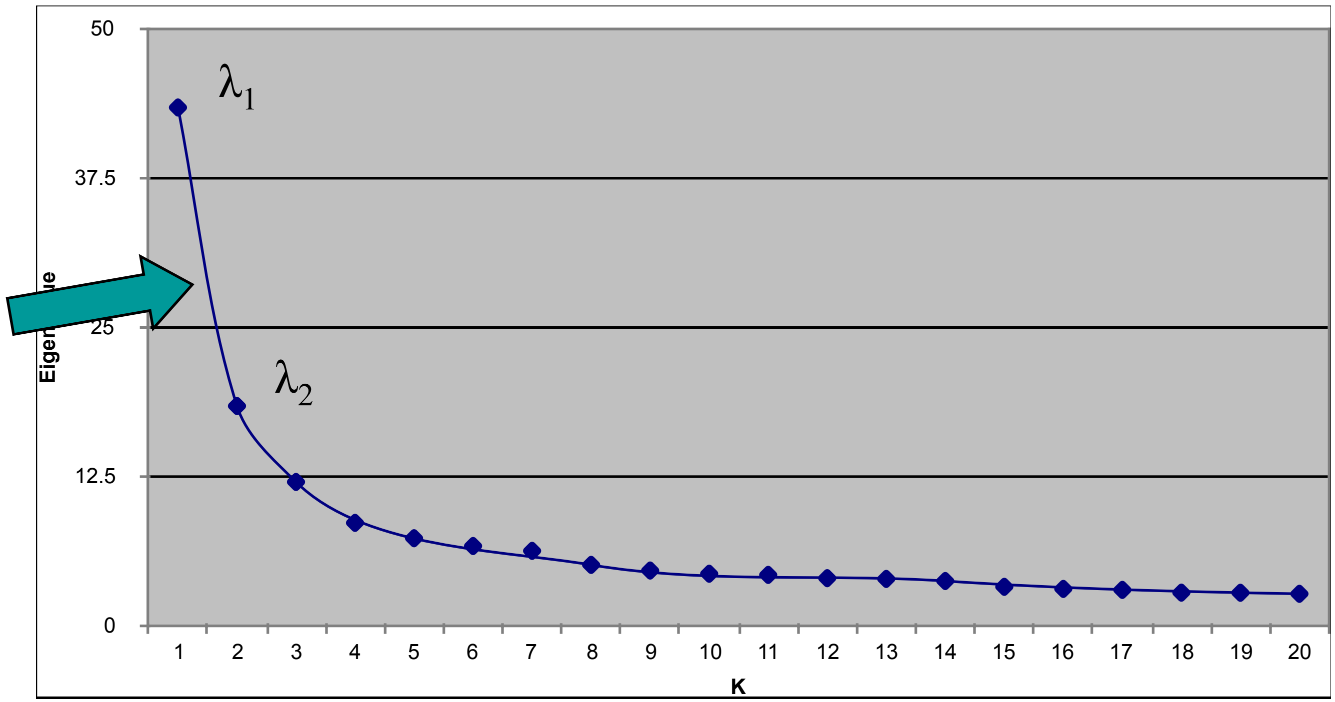
Spectral Gap: Selecting *k*

- *Eigengap*: the difference between two consecutive eigenvalues.
- Most stable clustering is generally given by the value k that maximizes the expression

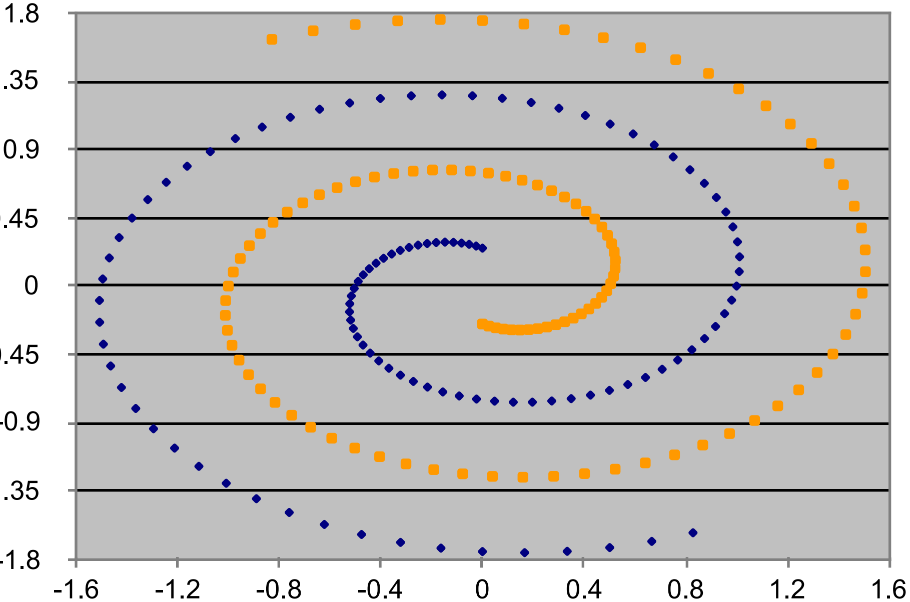
$$\Delta_k = \left|\lambda_k - \lambda_{k-1}\right|$$

$$\max \Delta_k = \left|\lambda_2 - \lambda_1\right|$$

$\Rightarrow$ Choose $k=2$



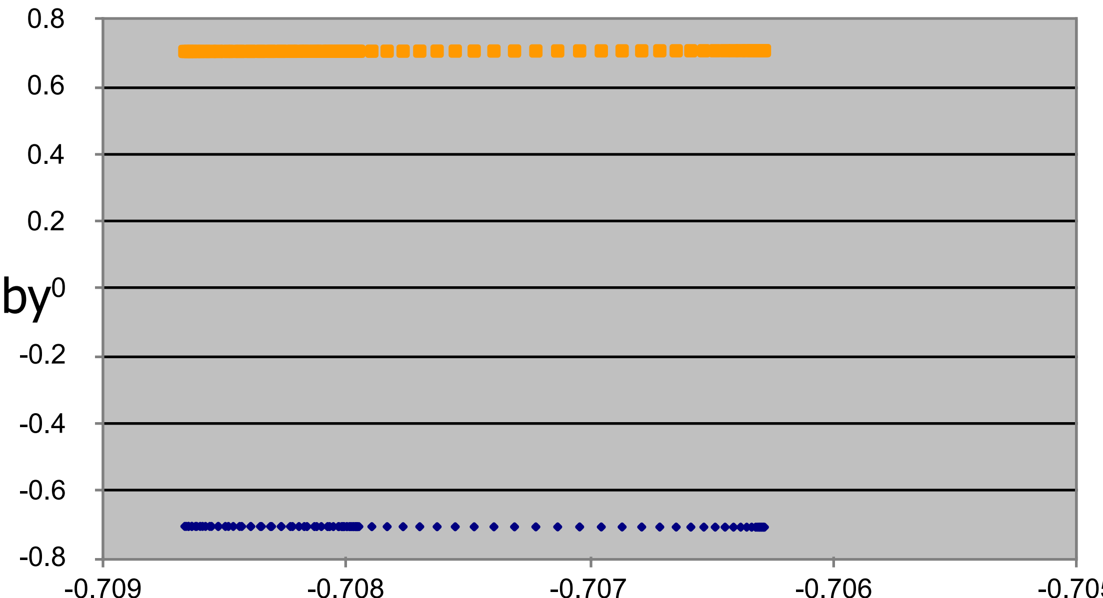
Spirals Again



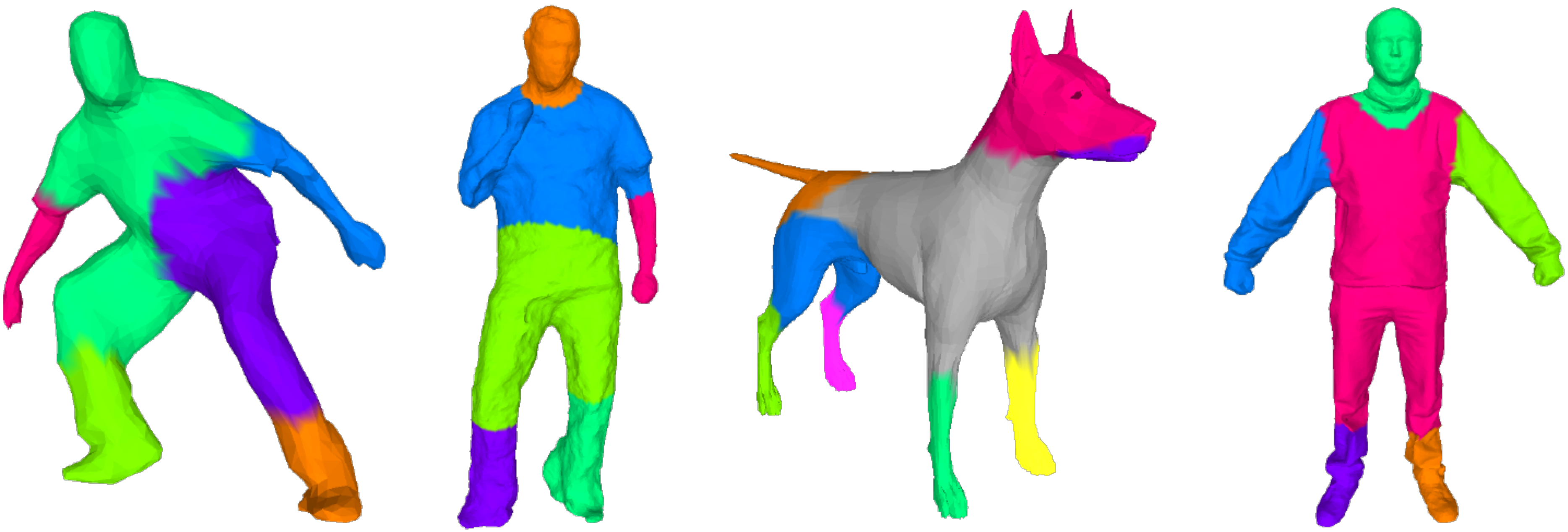
Dataset exhibits complex cluster shapes

⇒ Direct k-means performs very poorly in this space due to bias toward dense spherical clusters.

In the embedded space given by two leading eigenvectors, clusters are trivial to separate.



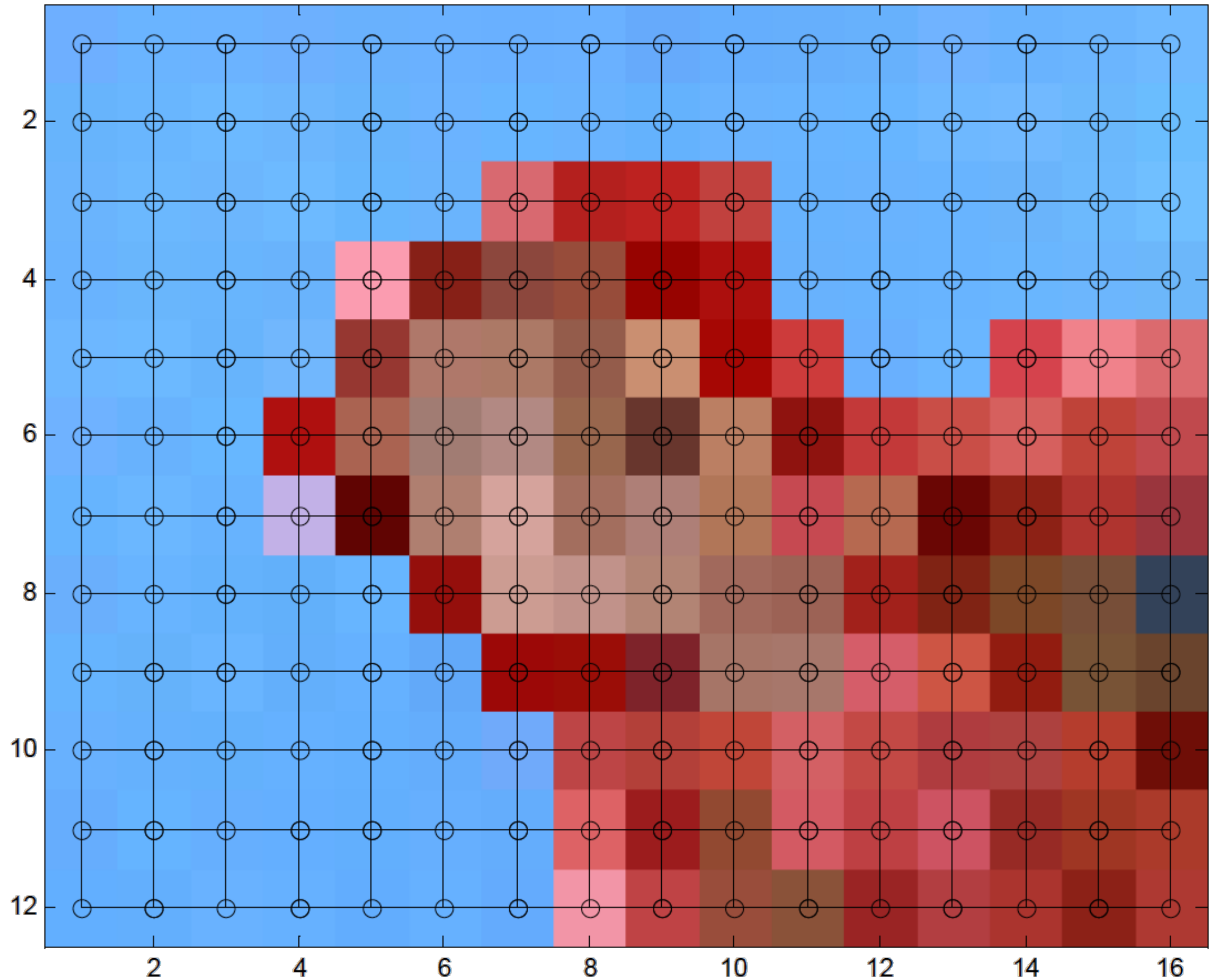
Mesh Segmentation Using Spectral Clustering



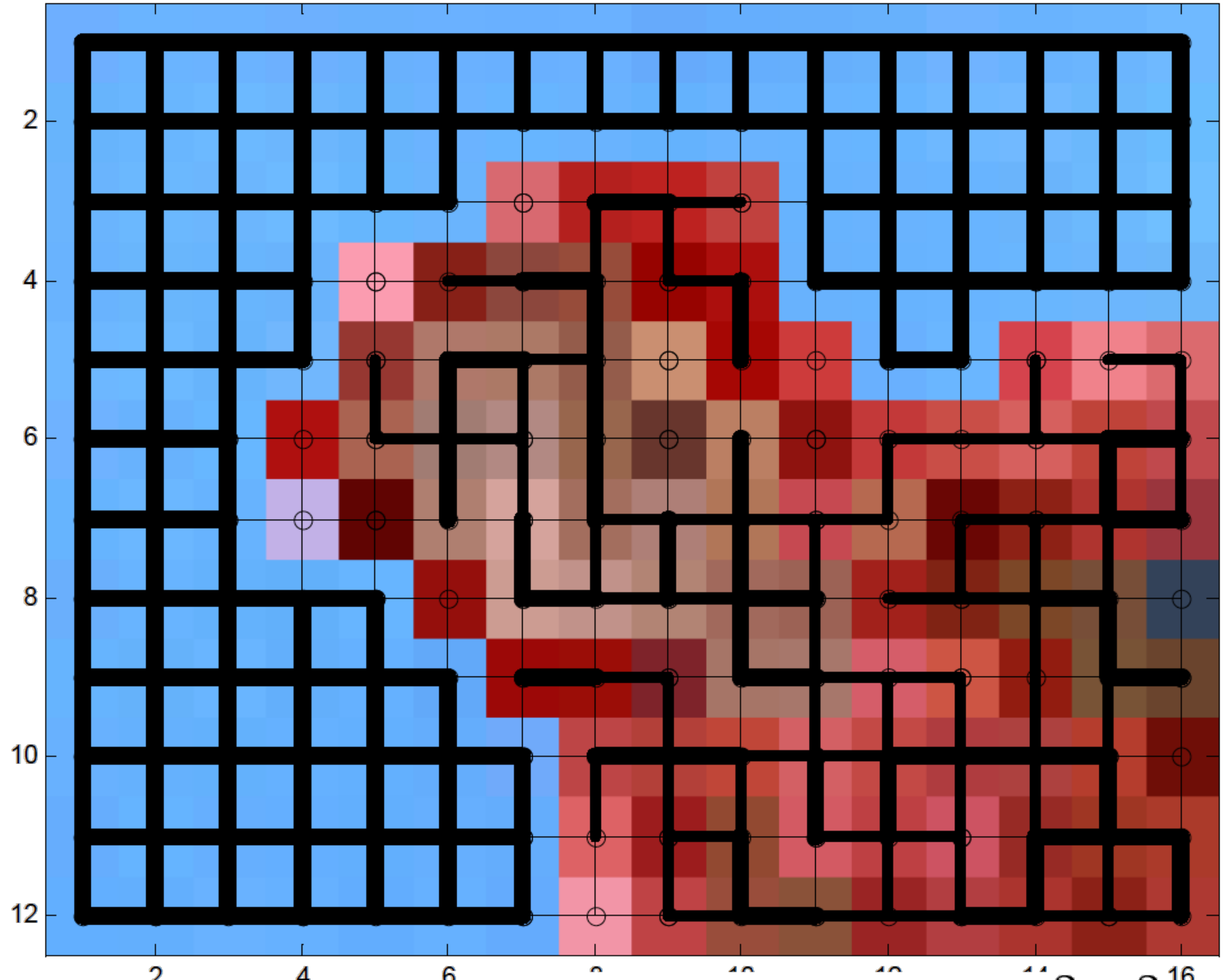
K=6 K=6 K=9 K=6

Spectral Image Segmentation

Spectral Image Segmentation (Shi-Malik '00)

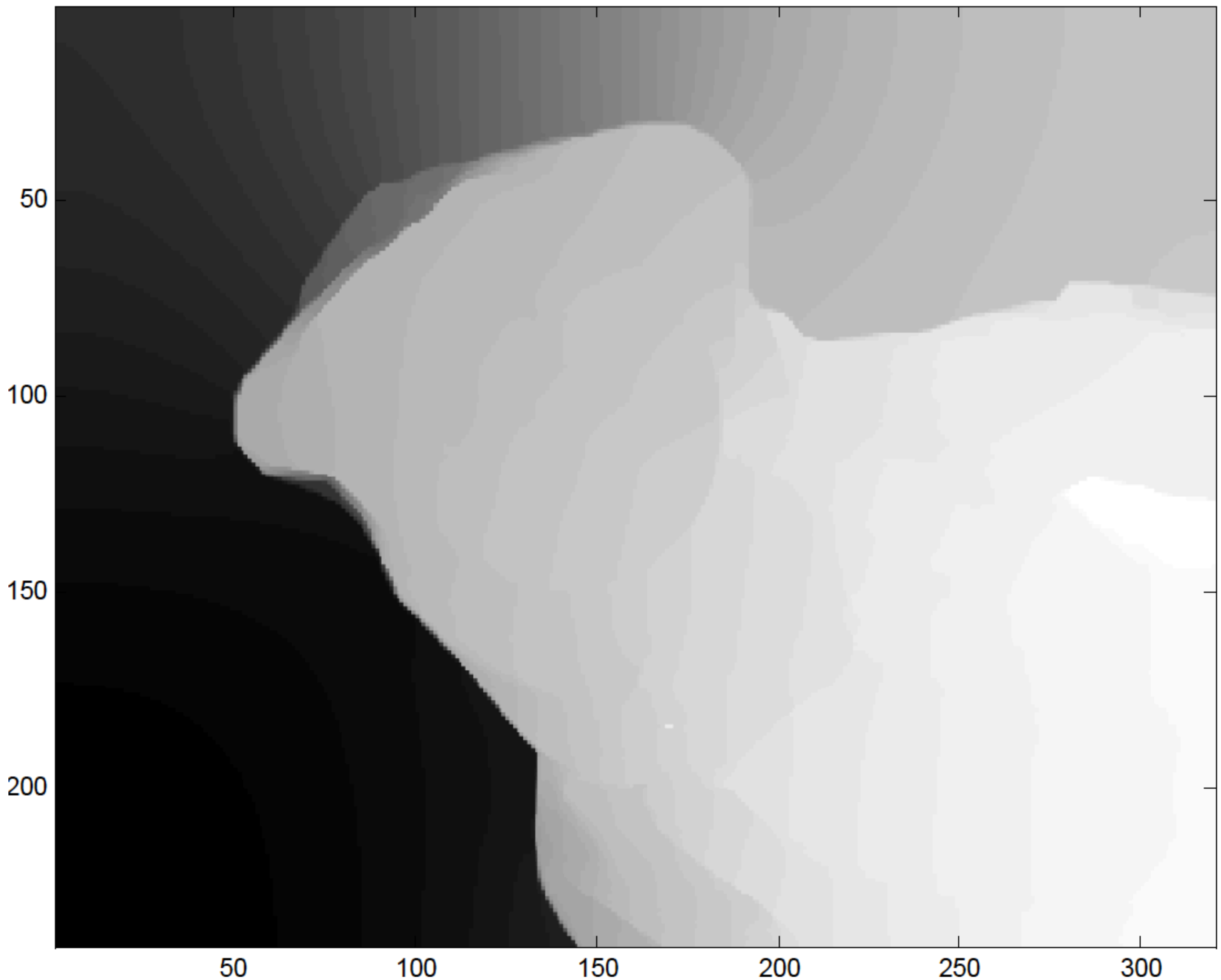


Spectral Image Segmentation
(Shi-Malik '00)



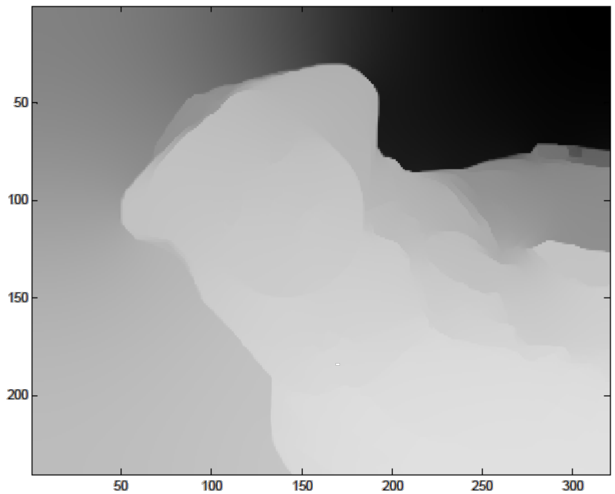
edge weight $e^{-\mathrm{diff}(\mathrm{pixel}\,i,\mathrm{pixel}\,j)^2/t^2}$

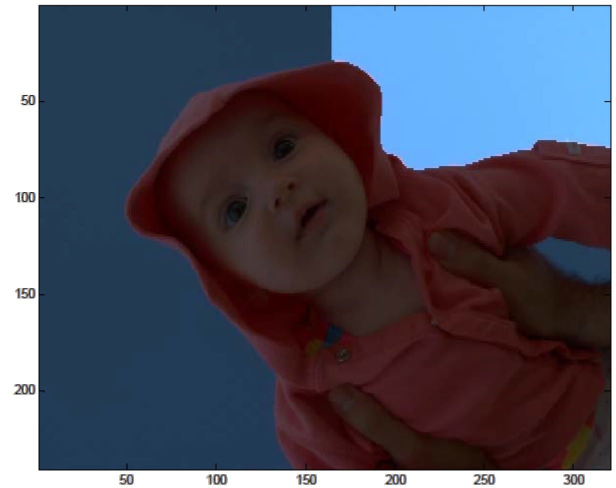
Second Eigenvector

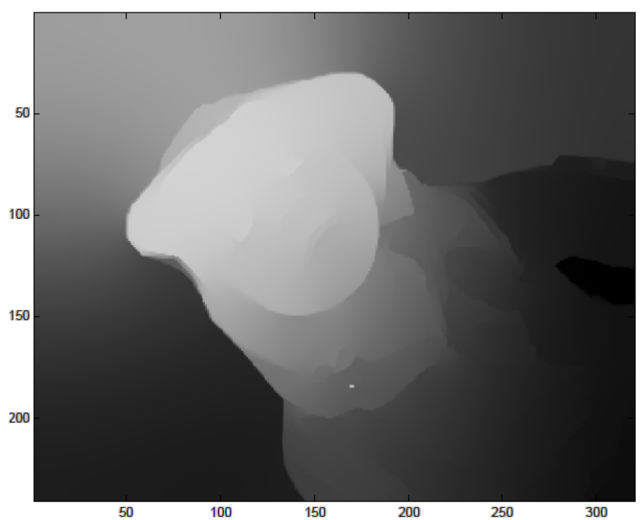


Second Eigenvector Sparsest Cut

3rd and 4th Eigenvectors







Conclusion

- Spectral graph embedding based on the graph Laplacian is a very powerful tool;
- Allows links between graphs and Riemannian manifolds
- There are strong links with Markov chains and random walks
- It allows clustering (or segmentation) under some conditions

The End