The Graph Analysis Toolbox: Image Processing on Arbitrary Graphs

Leo Grady[1] and Eric L. Schwartz[2]

8/14/03

[1]L. Grady is with the Department of Cognitive and Neural Systems, Boston University, Boston, MA 02215, E-mail: lgrady@cns.bu.edu

[2]E. L. Schwartz is with the Departments of Cognitive and Neural Systems and Electrical and Computer Engineering, Boston University, Boston, MA 02215, E-mail: eric@bu.edu

Abstract

We present a theoretical and computational framework for the analysis of data associated with the node set of an arbitrary graph. The algorithms described here are collected in a ™MATLAB software package named the Graph Analysis Toolbox. Our purpose is to describe the functionality of the Graph Analysis Toolbox and the associated theoretical framework in the context of biologically inspired, space-variant computer vision.

1 Introduction

For various reasons, many researchers have in the past been interested in freeing themselves of the constraints of a uniformly sampled, Cartesian data representation for images. Motivations include a dependence on nonuniform sensors, feature extraction, data reduction, or a desire to process even in the absence of some samples (or regions). Another group of researchers have been interested in modeling biological sensory systems or applying the architecture to computer vision tasks.

Biological vision systems contrast sharply with standard video inputs. Although the sampling of visual space varies widely between species in both the visual angle subtended and the sampling arrangement (see, for example [72] or [38]), a striking feature is that all systems non-uniformly sample visual space, usually anisotropically, with different types of receptors. A comparison of the retinal ganglion cell density for various species may be seen in Figure 1.1.

The main reason for employing a space-variant architecture is to process with dramatically lower bandwidth while retaining a high resolution in part of the visual scene. However, the difference in visual sampling across species suggests that there is a relationship between features of the sampling regime and the animal's visual ecology. An example of such a relationship is the belief that the "horizontal streak" seen in many animals (e.g., the rabbit in Figure 1.1) is helpful to species that live in open (i.e., non-occlusive) visual environments [38]. This belief has been supported by the strong correlation between species with less occlusive visual ecologies and those possessing a horizontal streak. Uncovering relationships of this nature help the engineer of a computer vision system design an architecture that is optimized to match the design constraints for the "visual ecology" of the artificial system. It

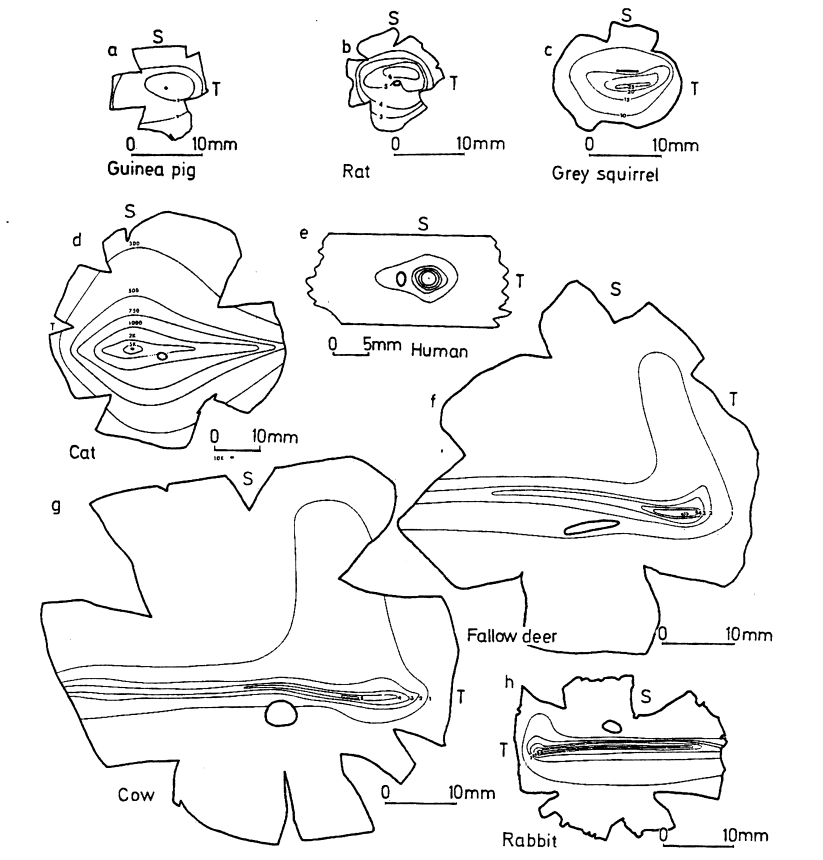


Figure 1.1: Isodensity lines for retinal ganglion cell distribution. Reprinted from [38], with permission.

is necessary that a data structure exists with sampling-independent (i.e., generalized) computer vision algorithms in order for the designer to be free to craft the visual sampling to the purpose of the system.

Traditional methods of space variant processing have focused largely on the *mapping* from a regular Cartesian grid to an alternate space [10, 6]. Therefore, the primary interest has been on the properties of the transform and not the necessary data structure. The notion of a "connectivity graph" was introduced by [71] to address the question of data structure. Unfortunately, the idea was not fully developed in the sense that the problem of visualization was not addressed, nor were many image processing algorithms developed that could operate on this structure. Finally, the structure was not generalized for an arbitrary sampling and no link with algebraic graph theory [5] was made. More recent approaches to space-variant vision appear to have abandoned this idea (e.g., [43]). The purpose of the Graph Analysis Toolbox is to provide a algorithms and data structures intended to facilitate computer vision on arbitrary visual sampling arrangements, even if a space-variant sensor is unavailable.

Work on the sampling of visual space employed by the upper primate visual system may be divided into two sections:

1. *Properties of the mapping*

   In addition to the considerable data reduction, the discretized log-polar transform of a Cartesian grid has a number of properties that make it interesting to researchers in computer vision. Casasent and Psaltis [10] discovered that objects in the fovea which are scaled, rotated or translated have a roughly invariant spectral signature. Their method is to employ the log-polar transform of an image, apply the discrete Fourier transform and find the log-polar transform of the spectrum (i.e., a Mellin transform). This invariance may be used for object recognition (see [43] for a recent application of this idea). The drawback to this idea is that the properties of scale and rotation invariance only apply if the fovea is directed to the center of the object. Furthermore, a translated object against a static background will not have the same spectrum as a translated object against a translated background, which means that the utility of this approach is limited to simple cases of an object moving against a uniform background (or uniformly translated background). Another approach to exploiting the structure of the log-polar mapping is that of Bonmassar and Schwartz [6], who developed a

specialized Fourier transform called the "exponential chirp transform" (ECT). The ECT finds the Fourier transform of an image in log-polar format and uses the structure of the logarithmic function to obtain an algorithm that is even faster than what would be obtained simply by the data reduction of an ordinary log-polar transform.

2. *Image Processing in a Space-Variant Domain*

   The notion of a "connectivity graph" was introduced by [71] to allow for image processing on a foveal sensor. This notion is introduced specifically to treat the sampling of the macaque retina [62]. However, standard computer vision tasks (e.g., edge finding, filtering) were not developed for this structure. Neither visualization of the connectivity graph nor how one simulates a space-variant sensor, given access to an acquisition device based on a Cartesian sensor array, is addressed. Furthermore, the generalization of the connectivity graph to other biological sampling schemes was not discussed. Chen [11] used a mesh-based representation to perform image processing. His data structure does not allow for an arbitrary topology (i.e., the graph must be planar and polygonal). Furthermore, his operators are discretized continuum operators, instead of combinatorial operators (see below). Significantly, Chen proposed the use of a method from computer graphics [34] for *resampling* an image to allow the determination of pixel values for an arbitrary sampling based on a given, Cartesian sampled, image. Unfortunately, this method only applies to resamplings for which the Jacobian is known. For the nearly log-polar mapping known to exist in macaque [62], this resampling works well. However, the method for resampling does not apply for most species, since the sampling known to exist does not currently have a mapping describing it.

The Graph Analysis Toolbox follows in the tradition of Wallace [71] and Chen [11] by providing methods to perform computer vision on graph-based architectures. However, since data processing on graphs appears in other fields such as computer graphics [68], 3D surface flattening [73] and data clustering [40], we hope that this toolbox will find a larger audience. Specifically, there are three problems that we hope to address with this toolbox:

1. *Importing images*

Since space-variant sensors [60, 59] are rare, tools must be developed for transferring image data acquired with a standard, Cartesian sensor array to a desired space-variant sampling arrangement. We use the phrase **importing an image** to refer to the process of transferring image data from a Cartesian sensor array to a space-variant arrangement. Our strategy is to extend the resampling work of Heckbert [34] by removing the requirement that the resampling is performed by a known differentiable function.

2. *Visualization*

   Additional tools are required in order to visualize space-variant images on a standard raster CRT display. A Voronoi cell and interpolation method are provided to allow visualization of image data on graphs.

3. *Processing*

   At the core of the Graph Analysis Toolbox are methods for performing image processing on data associated with each node in a graph. Some of the methods represent original work, while others are collected from other researchers. The theory underpinning much of this work was developed by Roth [58], Branin [7] and classic research in circuit theory [41, 78].

Since Zahn's classic paper [84], graph processing algorithms have become increasingly popular in the context of computer vision [83, 64, 74, 61, 53]. Typically, pixels are associated with the nodes of a graph and edges are derived from a 4- or 8-connected lattice topology. Some authors have also chosen to associate higher level features with nodes [61, 53]. For purposes of importing images to space-variant architectures, we adopt the conventional view that each node corresponds to a pixel.

Graph theoretic algorithms often translate naturally to the proposed space-variant architecture. Unfortunately, algorithms that employ convolution (or correlation) implicitly assume a shift-invariant topology. Although shift-invariance may be the natural topology for a lattice, a locally connected space-variant sensor array (e.g., obtained by connecting to K-nearest-neighbors) will typically result in a shift-variant topology. Therefore, a reconstruction of computer vision algorithms for space-variant architectures requires the use of additional theory to generalize these algorithms.

Data acquired from sensors may be viewed as samples of an exterior, continuous world, which must be analyzed from the limited information given by the samples. An alternate approach is to view the sensor data itself as the object about which conclusions must be drawn. For reasons that will become clear below, the former view of sensor data will be referred to as the **sampling paradigm** and the second as the **combinatorial paradigm**.

The difference between these paradigms may appear to be purely academic, since the primary output of many computer vision tasks (e.g., face detection) makes no comment on whether the result pertains to the pixels or the "real-world". However, some algorithms do operate under an implicit paradigm. For example, shape analysis [44, 85] typically adopts the approach of the sampling paradigm, while morphological analysis [65] employs the combinatorial paradigm. One practical difference between these two approaches is that algorithms developed to output statements about the continuous world should *improve performance* with increasing samples, while algorithms developed to output statements about the pixels should *decrease performance* due to the increased processing required. The difference between these two approaches is amplified when the sensor arrangement is space-variant, since the sampling theory is not as well developed for nonuniform samples [70], despite the fact that some authors have adopted the sampling viewpoint [6]. Furthermore, since a typical motivation for employing a space-variant architecture is the ability to employ a small number of pixels (while maintaining a high peak resolution), algorithms developed in the sampling paradigm are expected to decrease accuracy while combinatorial algorithms are expected to significantly increase in speed. For these reasons, the algorithms collected and developed for this toolbox adopt the combinatorial paradigm. Specifically, we approach the combinatorial paradigm through combinatorial analogs of vector calculus, since operators such as the gradient [57] and Laplacian [45] play such a prominent role in computer vision. The mathematical foundation for this viewpoint will be reviewed in the next section.

First we will introduce the basic mathematics and notation used in this paper for developing and applying combinatorial algorithms. A discussion of the data structures and implementational approach to these issues will then be addressed. The remainder of this paper describes and demonstrates the various tools for importing, visualizing and processing data on graphs.

2 Mathematical background

Most of the early work on the algebraic properties of graphs was done in the context of linear circuit theory. This section is essentially a short review of Branin's exposition on the algebraic-topological basis for analogy between graphs and vector calculus [7].

A **graph** is a pair $G = (V, E)$ with vertices (nodes) $v \in V$ and edges $e \in E \subseteq V \times V$. An edge, e, spanning two vertices, v_i and v_j, is denoted by e_{ij}. Let $n = |V|$ and $m = |E|$ where $|\cdot|$ denotes cardinality. A **weighted graph** has a value (typically nonnegative and real) assigned to each edge called a **weight**. The weight of edge e_{ij}, is denoted by $w(e_{ij})$ or w_{ij}. Since weighted graphs are more general than unweighted graphs (i.e., $w(e_{ij}) = 1$ for all $e_{ij} \in E$ in the unweighted case), we will develop all our results for weighted graphs. Define the **degree** of a vertex v_i, denoted d_i, as

$$d_i = \sum_{e_{ij}} w(e_{ij}) \ \ \forall\, e_{ij} \in E. \tag{2.1}$$

A graph may be defined from a linear electrical circuit by identifying the wire between circuit components with the node set, and the components bridging nodes (i.e., branches) as the edge set with weights equal to the admittance of each component (or the conductance, in the case of resistors). In this way, every linear circuit has an equivalent graph and *vice versa*. The explicit connection between circuits, graphs and algebraic topology was made in Roth's fundamental paper [58]. Roth showed that Kirchhoff's Current Law corresponds to a homology sequence in topology, while Kirchhoff's Voltage Law corresponds to a cohomology sequence. Roth then proposed Ohm's Law as a bridge between the sequences. Largely adopting the notation of Strang [66], we may write the fundamental equations of circuit theory in matrix form.

Define the $m \times n$ **edge-node incidence matrix** as

$$A_{e_{ij}v_k} = \begin{cases} +1 & \text{if } i = k, \\ -1 & \text{if } j = k, \\ 0 & \text{otherwise,} \end{cases} \tag{2.2}$$

for every vertex v_k and edge e_{ij}, where e_{ij} has been arbitrarily assigned an orientation. The notation $A_{e_{ij}v_k}$ is used to indicate that the rows of A are indexed by edge e_{ij} and the columns of A are indexed by node v_k.

Define the $m \times m$ **constitutive matrix**, C, as the diagonal matrix with the weights of each edge along the diagonal.

The three main laws governing circuit theory may be written as

$$A^T y = f \qquad \text{(Kirchhoff's Current Law)}, \tag{2.3}$$

$$Cp = y \qquad \text{(Ohm's Law)}, \tag{2.4}$$

$$p = Ax \qquad \text{(Kirchhoff's Voltage Law)}, \tag{2.5}$$

where f represents current sources at the nodes, p is the potential drop (voltage) across a branch, x is the potential at a node and y is the current through a branch.

By viewing the incidence matrix as a linear operator, it may be seen that application of that operator to a set of numbers assigned to each node induces a related set of numbers on the edges. Kirchhoff's Voltage Law is an example of this operation, in which electric potentials at each node are converted to voltages across edges by application of the incidence matrix. In a similar manner, the application of the operator A^T to a set of numbers on the edge set of a graph induces a related set of values defined on the node set. Kirchhoff's Current Law is an example of this operation, since the application of A^T to the currents through each branch yields the values of the current sources at each node. Application of C may be viewed as bridging the voltages and currents defined on each edge.

When an edge is added to a tree, the unique closed path so formed is called a **loop**. The set of loops, Q, formed by the addition of edges to a tree consists of elements, q_i, such that $q_i \in Q$. Note that $|Q| = |E| - |V| + 1$ (a variation of the Euler formula), since the number of edges in a tree of a connected graph is $|V| - 1$ [5]. Define the **loop-edge incidence matrix**

$$K_{m_k e_{ij}} = \begin{cases} +1 & \text{if } e_{ij} \text{ is crossed positively in a clockwise traversal of } q_k, \\ -1 & \text{if } e_{ij} \text{ is crossed negatively in a clockwise traversal of } q_k, \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

Similar to the edge-node incidence matrix, the application of the loop-edge incidence matrix to a set of numbers defined on the edges returns a related set of values defined on each loop.

Branin [7] identified the A, A^T and K operators with the familiar gradient, divergence and curl operators from vector calculus. This analogy holds for familiar identities such as $KA = 0$ (i.e., the curl of the gradient is zero) and

Operator	Vector calculus	Combinatorial
Gradient	∇	A
Divergence	$\nabla\cdot$	A^T
Curl	$\nabla \times \nabla$	K
Laplacian	$\nabla \cdot \nabla$	$A^T A$
Beltrami	$\nabla C \cdot \nabla$	$A^T C A$

Table 2.1: Correspondence between continuum differential operators and combinatorial differential operators on graphs. C represents a constitutive matrix relating flux to flow, e.g., a conductivity tensor, a diffusion tensor, a thermal conductivity, a stress-strain tensor, or, in the context of differential geometry, a metric tensor. A is the edge-node incidence matrix of the graph representing the topology of the problem and K is the loop-edge incidence matrix of the graph.

allows definition of other operators, such as the **Laplacian**, $L = A^T A$. The generalization of the Laplace operator to the Laplace-Beltrami operator [75] fits well with this analogy, where $L = A^T C A$. Since the constitutive matrix defines a weighted inner product of edge values (i.e., $\langle y, Cy \rangle$ for a vector of edge values, y), it may be considered as representing metric information. As a matrix, the Laplacian may be derived directly from knowledge of V and E by letting

$$L_{v_i v_j} = \begin{cases} d_i & \text{if } i = j, \\ -w(e_{ij}) & \text{if } e_{ij} \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

The notation $L_{v_i v_j}$ is used to indicate that the matrix L is being indexed by vertices v_i and v_j. This matrix is also known as the *admittance matrix* in circuit theory, and a good review of the properties of this matrix is given in [47]. More than one representation of the combinatorial Laplacian operator has been developed, depending on the choice of metric and normalization [24, 25, 48, 13]. However, unless otherwise noted, the above formulation will be referred to as the Laplacian. A summary of the analogies between operators in vector calculus and graph theory is given in Table 2.1, while additional relationships are listed in Table 2.2.

There is both a conceptual and practical difference between the graph

Equation	Continuum	Combinatorial
KVL	$\nabla V = E$	$Ax = p$
KCL	$\nabla \cdot J = \frac{d\rho}{dt}$	$A^T y = f$
Ohm's Law	$\sigma^{-1} E = J$	$Cp = y$
Dirichlet Integral	$\frac{1}{2} \int_\Omega \lvert\nabla u\rvert^2 d\Omega$	$\frac{1}{2} x^T A^T C A x$

Table 2.2: Correspondence between continuum differential equations and combinatorial differential equations on graphs. Kirchhoff's current law is a quasi-static ($\frac{\partial B}{\partial t} = 0$) approximation to Maxwell's Equation $\nabla \times E = \frac{\partial B}{\partial t}$. Kirchhoff's voltage law follows from the definition of electric field as the gradient of potential. Ohm's Law is a constitutive (phenomenological) law asserting a presumed linear dependence between voltage and current.

theoretic analog of a concept from mathematical physics and a discretization of the standard continuum representation of that same concept. Consider solving Poisson's equation [20] on a continuous domain with a digital computer (e.g., through use of finite elements). The objective of such a solution would be that the values assumed at any point in the domain could be determined, not just those points used in the calculation. In contrast, solving Poisson's equation on a graph, $Lx = f$, returns values *only for the node set*. Furthermore, the number of nodes and the graph topology (i.e., edge set) directly affects the solution to Poisson's equation. In contrast, a major design goal of a discretization procedure is that the solution for points in the domain are *invariant* to changes in the meshing.

Conceptually, one can understand the difference between solving for the charge distribution at the nodes of a planar electrical circuit given initially charged capacitors (i.e., the combinatorial diffusion equation [54]), and solving for the values taken at discrete samples of a planar conductive material with an initial heat distribution (i.e., the discretized continuous diffusion equation). Therefore, the solution of a continuous problem by use of a digital computer is referred to here as a **discrete** approach, while the solution of a problem using the graph-theoretic analogies shown above is referred to as a **combinatorial** approach.

All functions in this toolbox operate under the graph-theoretic, combinatorial paradigm. By this we mean that the operators we are concerned with are represented by matrices and the quantities of interest are defined by

vectors associating values to nodes, edges or meshes. Typically, the values associated with nodes are image values (e.g., grayscale, RGB color channels) or coordinate values, while those associated with edges and loops are dependent on the nodal values (e.g., the result of applying the gradient operator).

2.1 Adjacency matrix

Another fundamental matrix in graph theory is the $n \times n$ adjacency matrix defined as

$$W_{v_i v_j} = \begin{cases} w(e_{ij}) & \text{if } e_{ij} \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

The adjacency matrix has also appeared in applications (e.g., [53]), and its spectral properties have been thoroughly analyzed [22]. By noting that the adjacency matrix, Laplacian matrix and edge-node incidence matrix all completely specify the topology of a graph, it is not surprising that these matrices are related. Specifically, it is well known [5] that $D - W = L = A^T C A$, where D is the diagonal matrix with $D_{ii} = d_i$.

2.2 Indicator vectors

Another important type of values associated with nodes, edges and loops is an **indicator vector**. Indicator vectors are used to indicate membership of a node, edge or loop in a set. A common goal [55, 64] is to determine which components of a graph belong to a set (e.g., which pixels belong to a segment), and therefore the indicator vector represents the solution. Another important use of an indicator vector is to perform set operations (e.g., union, intersection) or determine properties of the set (e.g., cardinality, boundary nodes). We develop this section in the context of a nodal indicator vector. Indicator vectors on the set of edges or loops follow an identical development. For a set of nodes, $S \subset V$ an indicator column vector, x, may be defined as

$$x_i = \begin{cases} 0 & \text{if } v_i \notin S, \\ 1 & \text{if } v_i \in S. \end{cases} \tag{2.9}$$

For two sets, S_1 and S_2, with corresponding indicator vectors x_1 and x_2,

the usual set operations may be performed with

$$|S_1| = x_1^T x_1 \qquad \text{(Set cardinality)}, \tag{2.10}$$

$$S_1 \cap S_2 = x_1 \wedge x_2 \qquad \text{(Set intersection)}, \tag{2.11}$$

$$S_1 \cup S_2 = x_1 \vee x_2 \qquad \text{(Set union)}, \tag{2.12}$$

where $\wedge$ and $\vee$ denote the logical (binary) "and", "or" operations.

The matrices W and L may be used to determine useful properties of a vertex set, S, through operations with its indicator vector, x, in the following manner

$$\frac{1}{2}x^T W x = \sum_{e_{ij}, v_i \in S, v_j \in S} w(e_{ij}) \qquad \text{(Sum of the weights internal to } S\text{)},$$

$$x^T L x = \sum_{e_{ij}, v_i \in S, v_j \in \overline{S}} w(e_{ij}) \quad \text{(Sum of the weights on the boundary of } S\text{)},$$

where $\overline{S}$ indicates the set complement of S.

3 Implementation

We chose to implement the toolbox in ™MATLAB for several reasons:

1. *Numerical linear algebra* is at the core of the combinatorial approach to space-variant vision outlined above. Since ™MATLAB is well equipped with a numerical linear algebra package and a sparse matrix package [30], ™MATLAB is a natural environment for the toolbox.

2. *Rapid prototyping* of new algorithms is facilitated by the extensive set of tools available in ™MATLAB. Since this toolbox is intended for a research audience, the ability to rapidly prototype new algorithms is essential.

3. *Visualization* of space-variant images associated with graphs is a major design objective of the Graph Analysis Toolbox. ™MATLAB provides an excellent ability to visualize data.

However, ™MATLAB also has several drawbacks:

1. *™MATLAB is proprietary* and licenses for the standard package and additional toolboxes may cost hundreds to thousands of dollars. This fact limits the accessibility of the Graph Analysis Toolbox.

2. *Speed* of computation in ™MATLAB can be very slow for certain types of operations (e.g., code loops). Although it is possible to use the MEX package to speed up some of this computation, the portability of the code suffers. Fortunately for the Graph Analysis Toolbox, the numerical linear algebra package in ™MATLAB is relatively fast.

A full listing of the functions in the Graph Analysis Toolbox is given in Appendix A and a list of demos is given in Appendix B. For reasons of consistency, ease of readability and agreement with publications, the same variable names were used to refer to the same variables across functions and demos. A listing of standardized variable names used in the Graph Analysis Toolbox is given by Appendix C.

4 Data structures

There are different ways of representing a graph on a computer (e.g., lists, matrices). The choice of representation is often dependent on the particular application. The guiding principle in defining data structures for the Graph Analysis Toolbox is that functions should exist for switching between different representations and that information which may not be bound together should not be forced together (e.g., in a `struct`). Since ™MATLAB uses a pass-by-value system, this latter principle is especially important. Consequently, there is no all-purpose `struct` that contains all possible information about a graph. A full listing of standardized variable names is given in Appendix C.

4.1 Topological information

The most fundamental description of a graph is the topology, since none of the matrix representations may be defined without it. The most space efficient representation of the graph topology is given by a list, `edges`, that contains pairs of integers indicating nodes joined by an edge. However, the matrices A, L and W may be generated from the edge list with the functions `incidence.m`, `laplacian.m` and `adjacency.m`, respectively. The function

`adjtoedges.m` allows conversion from a adjacency matrix representation of topology to an edge set.

4.2 Nodal information

Numbers associated with nodes in the Graph Analysis Toolbox typically have two separate meanings: coordinates and image values. Aside from the semantics, there is no difference in the way in which these values may be represented or processed. However, since one may be interested in keeping notation of these quantities separate, two different N-dimensional lists are kept to refer to these quantities. The list called `points` refers to coordinate values, while the list `vals` typically refers to image values (e.g., RGB, grayscale).

4.3 Structs

Two quantities are kept in ™MATLAB `structs`, since their component values are never used separately. The information necessary to perform importing of a graph is kept in a `struct` called `imgGraph`. Voronoi visualization information (see below) is kept in a struct called `voronoiStruct`.

5 Generating graphs

When building a graph from a sensor array (or simulated sensor array), it is common to assign the value of each sensor to a node. However, the choice of connectivity (i.e., edge set) is much less clear. Typically, one wants the nodes to be locally connected. Two functions are provided for locally connecting a point set in arbitrary dimensions, `knn.m` and `triangulatepoints.m`. An N-dimensional Delaunay triangulation is implemented by `triangulatepoints.m` and K-nearest-neighbors is implemented by `knn.m`. For 2-dimensional points, `triangulatepoints.m` calls the MEX version of Shewchuck's `triangle.c` [63], if installed. Recent interest in the use of "small-world" networks [76, 77, 67] prompts inclusion of the function `addrandedges.m` to randomly add a specified number of edges to the graph. The effect of using `addrandedges.m` is to dramatically decrease the diameter of the graph [76] while only minimally increasing the number of edges.

Due to the prominence of the Cartesian lattice in traditional computer vision, the function `lattice.m` generates a lattice with three different topologies: 4-connected, 8-connected or a radially connected topology (as in [64]).

The use of a multi-scale image representation to enhance image analysis algorithms has a long history dating back to Burt [9] and Witkin [80]. Typically [82, 12, 1, 17, 51], a multi-resolution representation is employed both for speed and robustness against noise by performing the analysis at the coarsest level and projecting the solution back to the original image. Multiresolution approaches to general graphs have also been proposed [4, 28]. The Graph Analysis Toolbox includes one function `latticepyramid.m` that draws on this literature by returning a pyramid-shaped graph with 3-dimensional coordinates, such that every node at a higher level is connected to four (non-overlapping) nodes on the lower level. Although the pyramid graph is returned as a unit (i.e., a single node and edge set), an index is also returned indicating the level of each node and a list of its four children on the lower level. Some of the graphs described in this section are displayed in Figure 5.2.

Finally, the function `roach.m` generates the "roach" graph of Guattery and Miller [33] for purposes of testing.

5.1 Biological datasets

Included in the extended (demo) release of the Graph Analysis Toolbox is a precomputed (i.e., saved in `.mat` files) set of filters corresponding to the visual sampling associated with 22 different species. Following the retinal diagrams in [38] of ganglion cell isodensity lines (see Figure 1.1), topographical maps of other species have been published. Specifically, we have included node sets (and filters) corresponding to the visual sampling (as determined by ganglion cell counts) for baboon [79], beagle [52], bottlenosed dolphin [46], cat [37], cheetah [38], cow [38], deep-sea bass [15], deer [38], German shepherd [52], harlequin tusk fish [16], labrador [38], pig [38], pigeon [79], plains kangaroo [36], rabbit [35], sacred kingfisher [50], tree kangaroo [36], two-toed sloth [18], squirrel [38], wolf [52] and yellow-finned trevally [14]. A sampling reflecting the macaque retina is also provided by using the retinotopic function $w = \log(z + a)$ of Schwartz [62]. Generating sampling points according to this distribution is accomplished by the function `logz.m`.

Graphs were generated from topographic maps by interpolating the contours across the extent of the topographic image, treating this interpolation

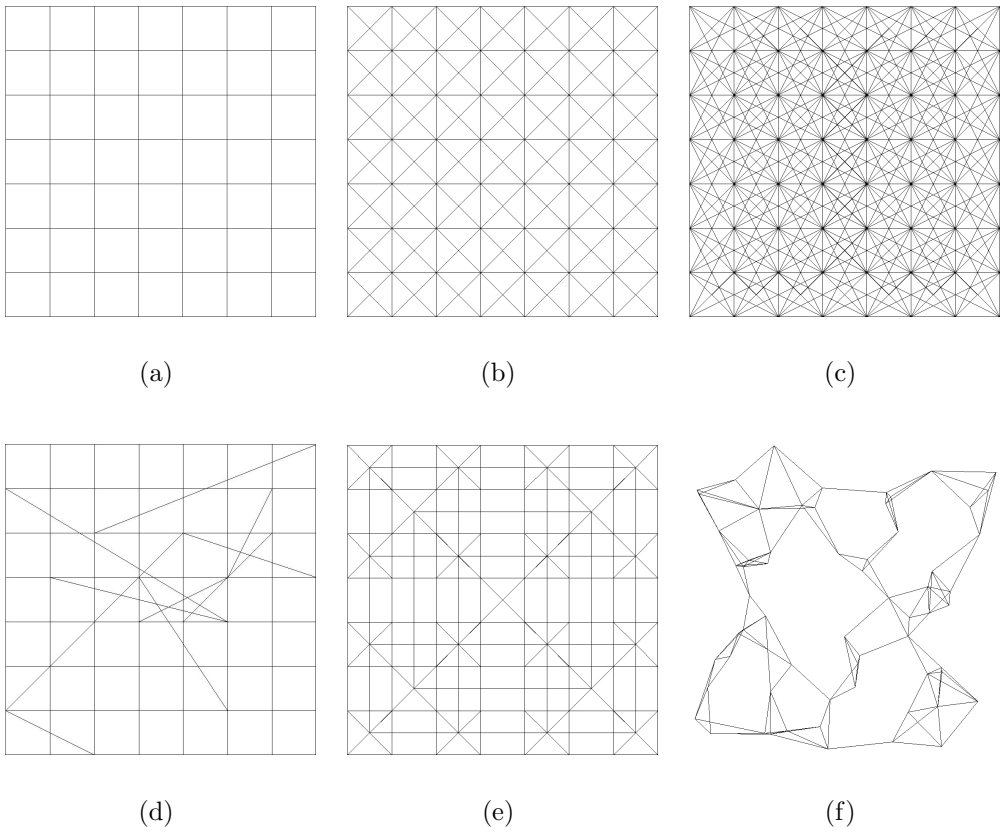


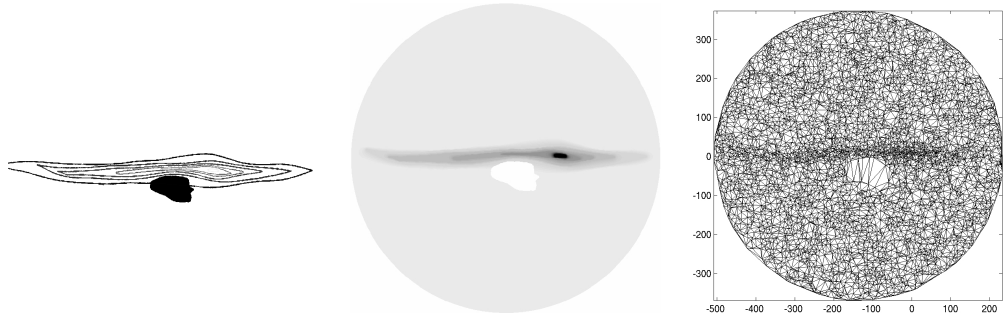
Figure 5.2: Examples of the structured graphs (i.e., node coordinates and edge sets) generated by functions in the Graph Analysis Toolbox. (a) A 4-connected lattice. (b) An 8-connected lattice. (c) A radially connected lattice. (d) A 4-connected "small world" lattice. (e) A flattened 4-connected pyramid lattice. (f) A K-nearest-neighbors graph with a randomly generated set of coordinates.

as a probability density function (PDF), sampling a predetermined number of points and scaling the coordinates of the points to fit the desired input image size. In the case of the precomputed filters included with the Demos package, the desired input image size was 256×256 ($65,536$ pixels) and the number of samples was $6,400$, resulting in less than an order of magnitude fewer pixels than in the original image. The function `contour2graph.m` accepts contour images where the background is white, the contours are shades of gray proportional to the values of the isodensity lines and the blind spot (or pecten, in the case of Aves) is colored red and returns an interpolated image (i.e., the PDF) by solving the corresponding Dirichlet problem (see below). Sampling points from the PDF is accomplished with the function `pdf2graph.m`. The process of converting a contour to a graph is shown in Figure 5.3. For purposes of comparison, Appendix D illustrates the PDFs of all the species included in the Graph Analysis Toolbox.

6 Importing images

The problem of importing an image is to transfer an image taken with a conventional lattice sensor array to a space-variant graph (usually with many less nodes than pixels in the original) with minimal aliasing. The approach proposed by Chen [11] was to treat the problem of importing as a *resampling* problem by calculating appropriate filters [34] to apply to a neighborhood of pixels in the original image in order to output a value for each node. The method of Heckbert [34] for defining filters requires the definition of a differential resampling function that inputs points in the original sampling and outputs points in the new sampling. This formulation was sufficient for Chen, since he was using the $w = \log(z + a)$ formulation of the mapping given by Schwartz [62] to model macaque retinotopy. However, we would like to make use of the visual sampling strategies of other species, for which no retinotopic function is known.

Since space-variant sampling arrangements typically have some areas of high acuity and some areas of low acuity, they require an active vision system to allow the high acuity regions to sample different regions of a visual scene. By analogy with the high acuity foveal pit found in most vertebrate vision systems, we refer to the region of highest acuity as the **fovea** and the process of addressing the high acuity area to a region of an image as **foveation**. In order to simulate the active vision aspects of a space-variant visual sampling



(a) (b) (c)

Figure 5.3: Demonstration of conversion from a retinal topography to a graph. The example chosen here is based on the retinal topography of the cheetah [38]. (a) Isodensity contours of the cheetah retinal topography. Darker contours represent a lower ganglion cell density, while lighter contours represent a higher ganglion cell density. The blind spot is shown in black for publication, although the function `contour2graph.m` requires it to be colored red. (b) The interpolated and normalized probability density function determined from the contours (see text for details). Darker areas correspond to areas of greater probability. (c) The graph obtained by sampling $6,400$ nodes from the distribution in (b) and connecting with a Delaunay triangulation. Note that the fovea is not at the center of the image (i.e., the contours are taken from the left eye).

regime, we would like to be able to specify a point in a large image for the system to fixate on. Therefore, a further design criteria for an importing procedure is that we want the importing procedure to be relative to its own internal coordinate system that may be "aimed" at different areas of a large image. Our method for accomplishing this is to give the nodes coordinates such that the fovea is at the origin and each unit represents one pixel in a standard raster image. The advantage of this design is that the filters may be precomputed for a graph relative to the origin and simply shifted to different areas of the image, resulting in fast on-line importation. The drawback to this design is that the extent of the "visual field" must be fixed prior to computing the filters.

Heckbert's Elliptical Weighted Average filters are ellipses computed for each new sample such that the axes of the ellipse lay along the eigenvectors of the Jacobian matrix and the weights for each point in the ellipse are given by an elliptical Gaussian function. In other words, the image value assigned to each resampled point is the weighted sum of image values on the original points lying within the computed ellipse. Our approach to computing the ellipses for the resampled points is to perform a least-squares fit of an ellipse to the Voronoi cell of each node and compute Gaussian weights. The ellipses computed for a small randomly generated set of Gaussian distributed points in the plane are seen in Figure 6.4.

The filters for a point set are stored in a ™MATLAB `struct` named `imgGraph`. An `imgGraph` has three fields: `pntMap`, `breakpoints` and `filtWeights`. In order for the importation of images to be fast in ™MATLAB the three fields are used to avoid code loops. The `breakpoints` field contains a list of indices to `pntMap` and `filtWeights` that indicate the start and end of a block of pixels or weights corresponding to each new node. The `pntMap` field is a set of two vectors containing the x- and y-coordinates for the nodes corresponding to those pixels that lie within the ellipse for each point. The `filtWeights` field contains the weight to be applied to each pixel in `pntMap`. Therefore, the size of `breakpoints` is the same as the cardinality of the node set, while the size of `pntMap` and `filtWeights` are larger than the node set (since more than one pixel typically maps to each node), but equal to each other. Figure 6.5 demonstrates the importing of an image onto a set of nodes that were randomly distributed in the plane with a uniform probability. Figure 6.6 demonstrates the simulated active vision system by importing a large image at multiple fixation points.

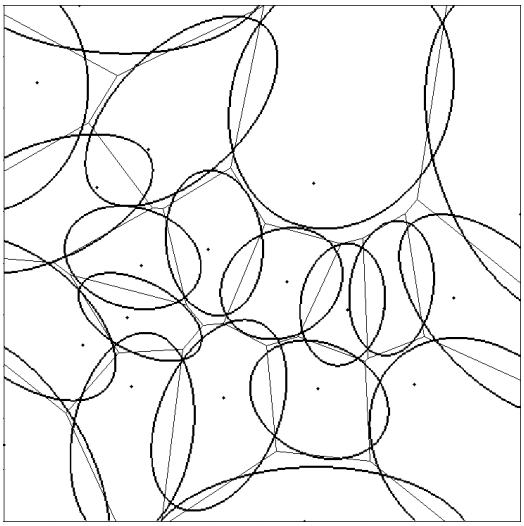


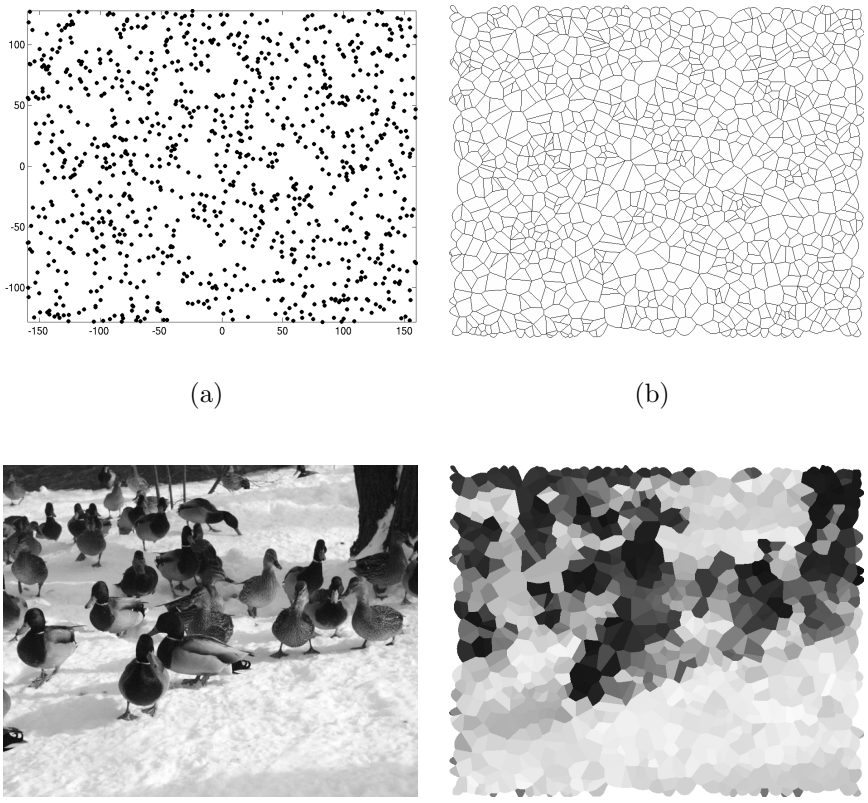
Figure 6.4: Voronoi cells for a point set and the corresponding ellipses fit with least squares error used to generate the Elliptical Weighted Average filters of Heckbert [34].

7 Visualization

Visualization of arbitrary, nonuniformly 2D sampled data is a difficult issue. Typical "stick-and-ball" representations of graphs poorly convey the content of an image associated with the nodes. Two methods have been implemented for visualizing an image on an arbitrary architecture.

The first of these methods interpolates image values at a node across the *faces* of a graph. If the graph is planar, the interior faces will be polygons, with image data at each point on the polygon. If the graph is nonplanar, a Delaunay triangulation of the points may be found quickly for purposes of the visualization. However, it should be noted that not all graphs will have faces that produce a good image (e.g., if the nodes were collinear). A common "shading" technique from computer graphics is to perform a bilinear interpolation across vertices (i.e., Gourand shading). Applying this technique to the internal faces provides a smooth change across the image. Unfortunately, the polygonal faces can introduce artifacts into the visualization that degrade the quality. The function `showmesh.m` generates a display in this manner.

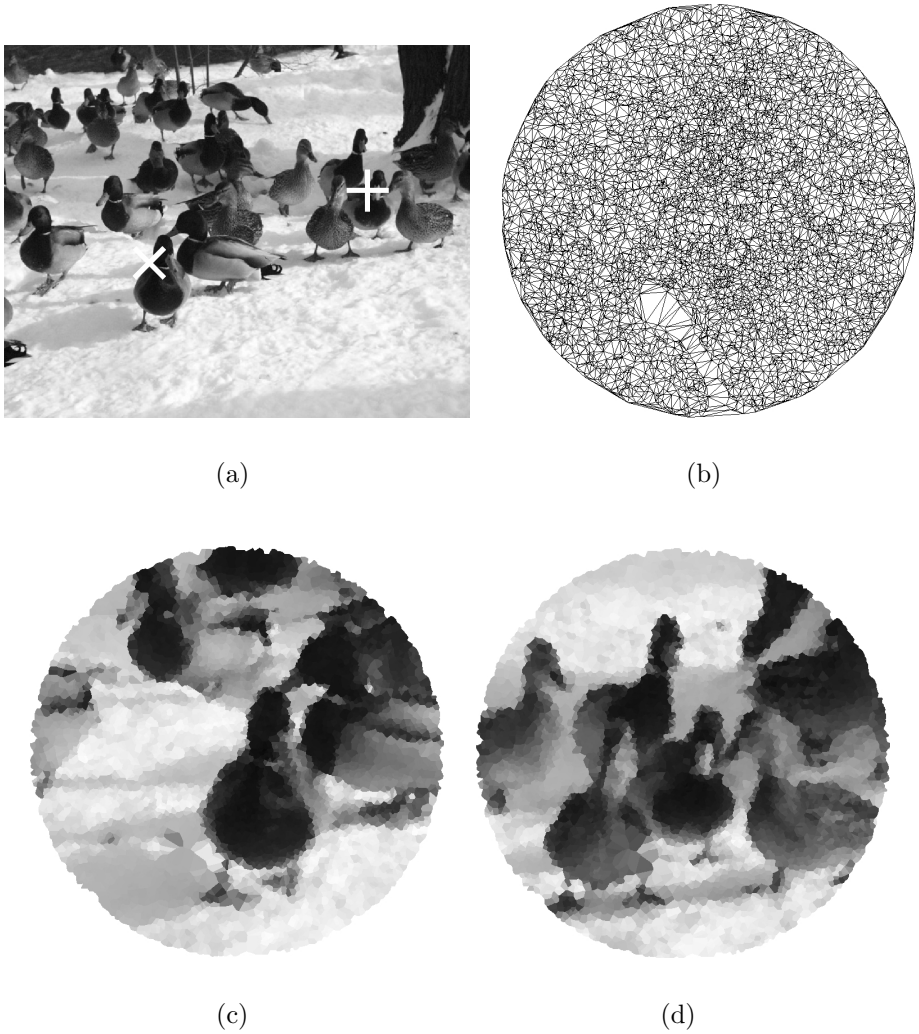
A second visualization method is implemented in the Graph Analysis Toolbox that uses the Voronoi diagram of the vertex set. This technique



(a) (b)

(c) (d)

Figure 6.5: Demonstration of importing an image onto a set of points distributed randomly in the plane with a uniform distribution. Note that the resolution of the sampling is almost two orders of magnitude smaller than the resolution of the original. (a) Nodes randomly placed in the plane with a uniform distribution. (b) Voronoi cells for the node set upon which the filters were generated. (c) The original raster image: `ESLab0043.jpg`. (d) The image imported onto the node set.



(a) (b)

(c) (d)

Figure 6.6: Demonstration of importing a large image onto a smaller graph at different points (i.e., multiple fixation points). The example chosen here is based on the retinal topography of the pigeon [79]. (a) Original image: `ESLab0043.jpg`. The two fixation points are marked with a white '×' and a white '+'. (b) The graph corresponding to the retinal topography of the pigeon. (c) Result of importing the image in (a) at the point marked with a white '×'. (d) Result of importing the image in (a) at the point marked with a white '+'.

simply assigns each Voronoi cell the color (or grayscale value) of its corresponding vertex. The visualization benefits from its independence of the planar and internal faces requirements of the previous technique. However, the visualization can sometimes look blocky. Furthermore, since the boundary nodes have Voronoi cells that extend to infinity, a dense set of phantom nodes is used to make finite, appropriately sized cells for the boundary nodes. The Voronoi information for a graph is stored in a ™MATLAB `struct` called `voronoiStruct`. A `voronoiStruct` is produced from a node set by the function `voronoicells.m` and consists of the three fields `pts`, `faces` and `index`. The field `voronoiStruct.pts` contains coordinates for the vertices of the Voronoi cells for the node set. Faces intended for use by `patch.m` are contained in `voronoiStruct.faces` and an index referencing nodes with Voronoi cells contained inside the convex hull is given by `voronoiStruct.index`.

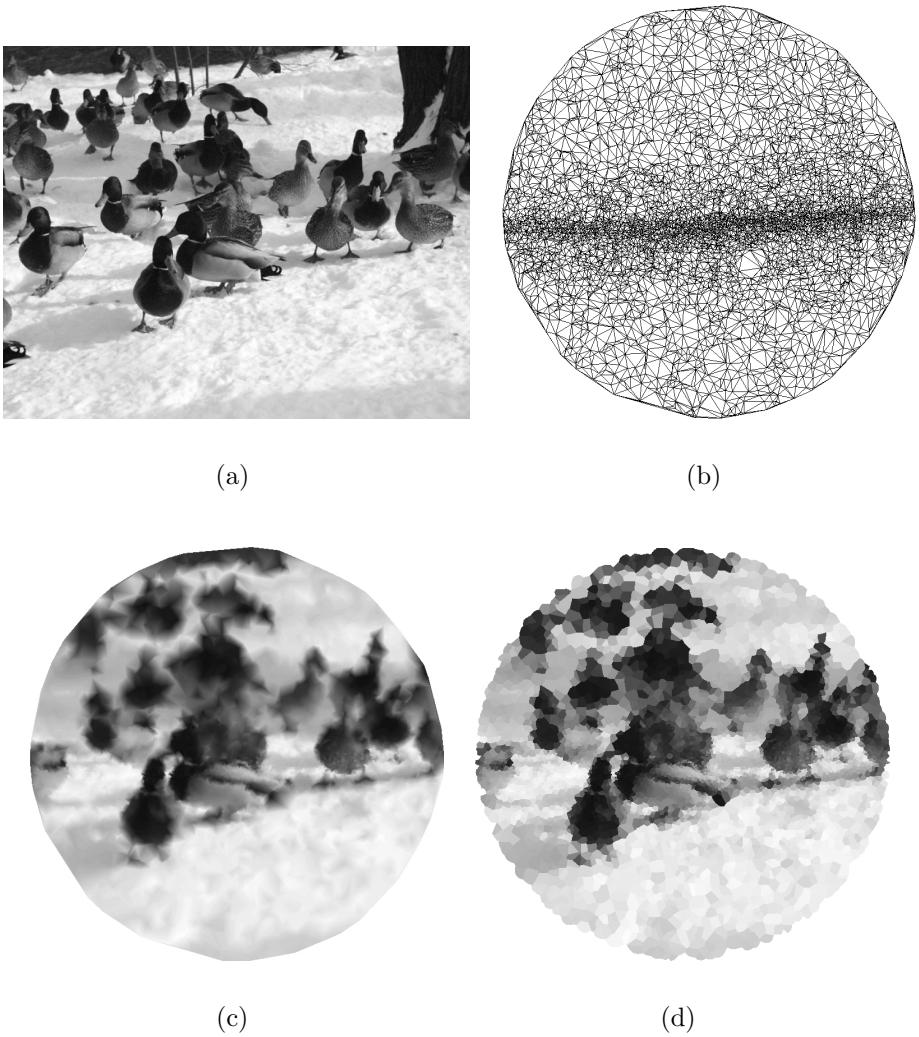
The two visualization techniques are illustrated in Figure 7.7 for a grayscale image. Despite the smoothness given by the face interpolation method of visualization, the Voronoi cells method offers a better notion of the structure of the image distribution on the nodes, and we therefore prefer it for visualization of space-variant images in the remainder of this document, as well as in the space-variant examples shown above.

It is important to distinguish between *sampling aliasing* and *visual aliasing*. Sampling aliasing refers to the inadequacy of the local sampling density to satisfy the image frequency (as detailed above). On the contrary, visual aliasing refers to the displeasing visual artifacts induced as a result of the visualization technique. Sampling aliasing is minimized by the precomputation of Heckbert's resampling filters. Visual aliasing, however, depends on the visualization method employed and in no way reflects an inadequacy of the data obtained nor its internal representation.

8 Processing

Processing data on graphs is a recurring theme that extends beyond space-variant vision systems. There are four main types of processing implemented in the Graph Analysis Toolbox: Interpolation, filtering, edge finding and segmentation.

Isotropic and anisotropic versions of all the processing methods exist in the sense that the edge weights all take unity value in the isotropic case and the weights assume different values in the isotropic case. For example,



(a) (b)

(c) (d)

Figure 7.7: Comparison of the two visualization techniques implemented in the Graph Analysis Toolbox. (a) Original image: `ESLab0043.jpg`. (b) The graph corresponding to the retinal topography of the plains kangaroo [36]. (c) Visualization of the space-variant image performed by interpolating across the Delaunay triangles of the graph implemented in `showmesh.m`. (d) Visualization of the space-variant image performed by assigning a uniform grayscale value to the Voronoi cells of the node set, implemented in `showvoronoi.m`.

use of uniform or nonuniform weights in building the Laplacian matrix is the difference between the isotropic diffusion of Koenderink [42] and the anisotropic diffusion of Perona and Malik [54]. Therefore, the theme is to encode data information (e.g., intensity changes for images, distances for a point set) in the structure of edge weights, and then build the operator in accordance with the weights. This procedure provides the difference between isotropic or anisotropic diffusion, interpolation, filtering and edge finding, as well as affording the structure necessary for some segmentation algorithms (e.g., [64]).

The functions used to build the important matrix operators listed above are `incidence.m`, `laplacian.m` and `adjacency.m`. The function `incidence.m` generates the edge-node incidence matrix, `laplacian.m` generates the Laplacian matrix and `adjacency.m` generates the adjacency matrix. In order to allow the user to choose between isotropic and anisotropic operators, all of the operator generating functions allow specification of edge weights in order to produce anisotropic operators, but default to generating isotropic operators if weights are not specified.

The important task of generating a weight set is handled by the function `makeweights.m`. Define the vector of data changes, c_{ij}, as the Euclidean distance between the fields (e.g., coordinates, image RGB channels, image grayscale, etc.) on nodes v_i and v_j. For example, if we represent grayscale intensities defined on each node with vector b, then $c = Ab$. If the fields are nodal coordinates in the plane, then c represents the Euclidean distance in the plane. In order to make one choice of β applicable to a wide range of data sets, we have found it helpful to normalize the vector c.

Although we typically treat coordinates as any other data field (e.g., to filter, interpolate, etc.), in the context of space-variant vision we may want to treat spatial differences between the nodes separately from image-derived differences. Therefore, `makeweights.m` optionally accepts both data values and coordinate values for the node set and generates a corresponding (normalized and histogram equalized) c^1 and c^2 for data and coordinate values, respectively. Accordingly, associated with the data and coordinates is a separate parameter, β^1 and β^2 we call **scale**. Setting either parameter to zero nullifies the effects of the corresponding data or coordinates. A common weighting function is implemented in `makeweights.m`, defined by

$$w_{ij} = \exp\left(-|\beta^1 c_{ij}^1 + \beta^2 c_{ij}^2|\right). \tag{8.1}$$

8.1 Interpolation

The method of interpolation implemented in the Graph Analysis Toolbox is to solve the combinatorial Laplace equation with Dirichlet boundary conditions given by the known values [31]. A solution to the combinatorial Laplace equation has several desirable properties in the context of an interpolation method (see below). Both isotropic and anisotropic interpolation are handled similarly. Furthermore, use of the algorithm is independent of the dimension in which a graph is embedded.

Solving the Laplace equation in order to "fill-in" missing values has been described in the context of digital elevation models [8, 81], image editing [27], and is even used by the ™MATLAB function `roifill.m` to fill in regions of missing data in images.

Solutions to the Laplace equation with specified boundary conditions are harmonic functions, by definition. Finding a harmonic function that satisfies the boundary conditions may be viewed as a method for finding values on the interior of the volume that interpolate between the boundary values in the "smoothest" possible fashion [21]. In this section, we discuss the properties of harmonic functions that make them useful for interpolation, defining smoothness in terms of extremal solutions to the Dirichlet integral.

From a physical standpoint, one may think of a heat source with a fixed temperature at the center of a copper plate and a second heat source with fixed temperature on the boundary of the copper plate. The temperature values taken by the plate at every point are those assumed by a harmonic function subject to the internal and external boundaries imposed by the heat sources. In this analogy, the temperatures measured on the inside of the copper plate may be viewed as smoothly interpolated between the temperature on the internal heat source and the external heat source. The internal and external heat sources are considered to be *boundary* points, while points on the copper plate for which temperature values are found are *interior* points. Three characteristics of harmonic functions are attractive qualities for generating a "smooth" interpolation:

1. The *mean value theorem* states that the value at each point in the interior (i.e., not a boundary point) is the average value of its neighbors [2].

2. The *maximum principle* follows from the mean value theorem. It states that harmonic functions may not take values on interior points that are

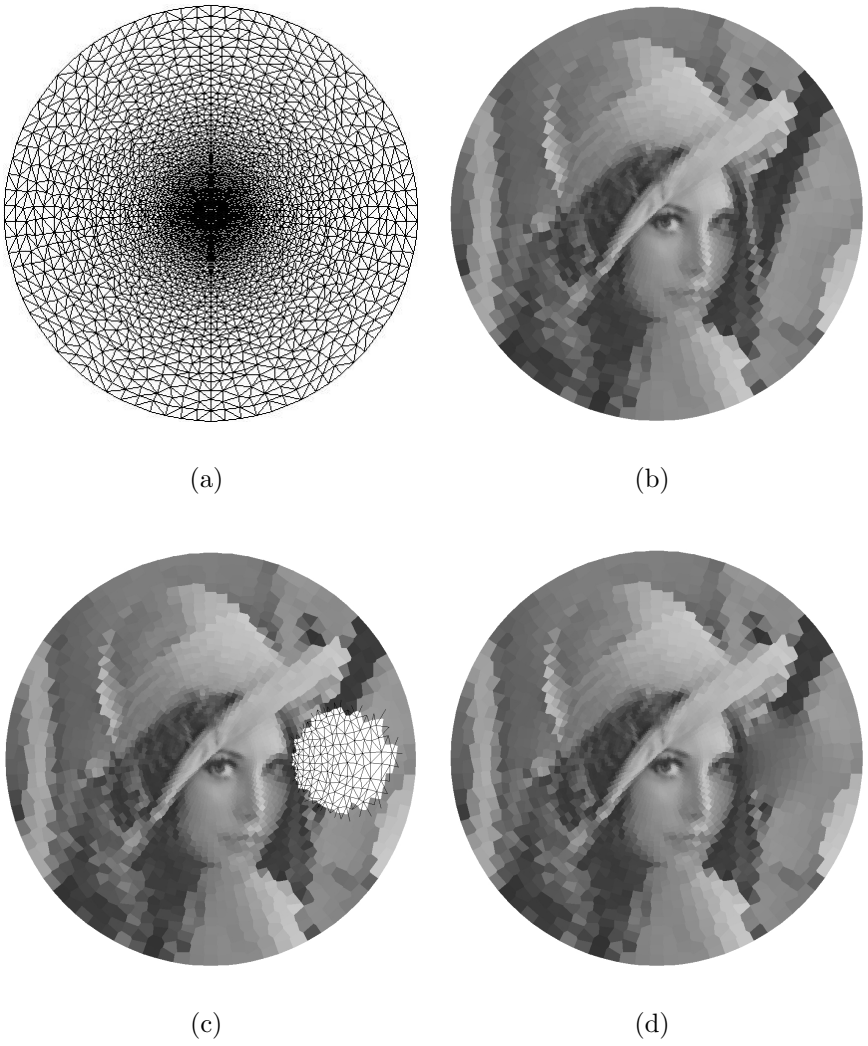


Figure 8.8: Interpolation of image data on a space-variant graph from which a hole has been cut out. (a) A space-variant graph patterned after the retinotopic map of the macaque [62]. (b) The Lena image imported onto the graph. (c) Space-variant image with a hole arbitrarily cut out of it. Underlying graph structure is shown inside the hole. (d) Foveal image with isotropically interpolated data in the hole.

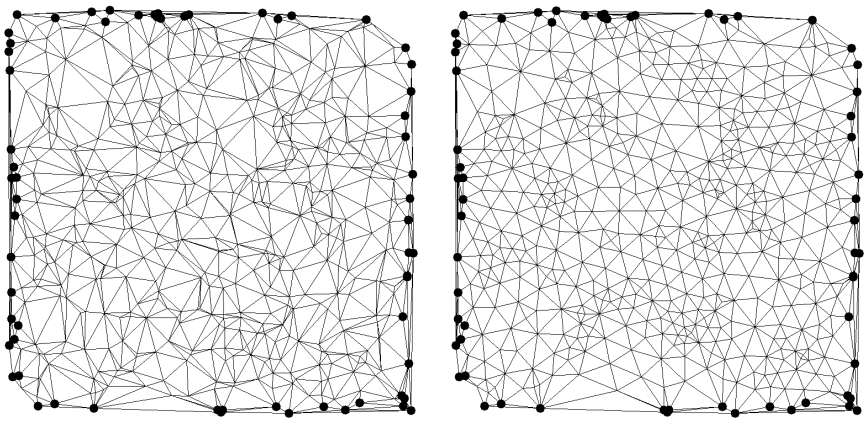
Figure 8.9: Anisotropic interpolation of image data on the macaque-based graph where the same region as in Figure 8.8 has been removed. Weights were determined using $\beta^1 = 30$ (see text for details).

greater (or less) than the values taken on the boundary [2].

3. The Dirichlet integral is minimized by harmonic functions [19]. This means that the integral of the gradient magnitudes for the system will be minimized, subject to fixed boundary conditions.

The function `dirichletboundary.m` inputs an index of boundary nodes and their values and solves the combinatorial Dirichlet problem for a graph with arbitrary connectivity, producing a combinatorial harmonic function. An example of using `dirichletboundary.m` to perform isotropic interpolation on a space-variant image with a region of missing values is displayed in Figure 8.8. By generating weights corresponding to the image intensities, anisotropic interpolation may also be used to find the missing values in a region, as displayed in Figure 8.9.

Graph drawing is another task in which `dirichletboundary.m` is useful. By treating the extremal nodes as boundary nodes, the coordinates of the interior nodes may be interpolated in order to produce a more regular representation in the sense that each interior node is placed at the average of its neighbors (by the mean value theorem). This usage of `dirichletboundary.m` is displayed in Figure 8.10.



(a) (b)

Figure 8.10: An example of using the interpolation method for graph drawing. (a) A graph was created by randomly generating the coordinates with a uniform distribution and connecting the points with a 2D Delaunay triangulation. The black dots represent the extremal points chosen to represent the boundary (i.e., to have their coordinates fixed). (b) The graph generated by interpolating the coordinates of the interior nodes.

9 Filtering

In the context of surface fairing, Taubin has already taken a combinatorial approach to filtering on a graph [68, 69]. Taubin treats the coordinates of a vertex set of a 3D mesh as a signal for which low-pass filters may be designed in order to smooth a noisy surface. The signal processing treatment in Taubin's work follows standard signal processing approaches, except that Taubin wants to apply the same techniques to a shift-variant topology. Generally the eigenfunctions of the Laplacian operator define the surface harmonics [20]. In the combinatorial setting, the Laplacian operator is represented by the Laplacian matrix, although Taubin chooses a different combinatorial representation of the Laplacian operator than the definition given in (2.7). For shift-invariant topologies, the Laplacian matrix is circulant and the complex exponential basis vectors (functions) are the eigenvectors [66]. In the general case of a shift-variant topology, the Laplacian is not circulant, requiring a different set of (usually unknown) eigenvectors in order to perform signal filtering. Taubin's method of filtering circumvents the need to compute the eigenvectors explicitly in order to modify the spectral coefficients of an input signal (e.g., the coordinates of a graph or an image on a graph). The function `filtergraph.m` implements Taubin's $\lambda - \mu$ filtering technique as well as standard mean filtering. Both the mean filter and the $\lambda - \mu$ filter are low-pass filters. However, a high pass filter may be generated by subtracting the low-pass filtered signal from the original. A band-pass filter may be generated by using the difference of two low-pass filters. Figure 9.11 demonstrates image filtering and Figure 9.12 demonstrates coordinate filtering.

The spectrum of the Laplacian matrix has been thoroughly investigated [49, 13, 47, 3, 29]. It is well known that the eigenvalues of the Laplacian matrix are nonnegative and ordered such that the smallest eigenvalue corresponds to the lowest frequency harmonic (i.e., the DC component) and the largest eigenvalue corresponds to the highest frequency harmonic. This view of the spectral characteristics of the Laplacian matrix predicts its use as an edge detector since, as an operator, it will clearly have the effect of a high-pass filter. Another implication of the spectral properties of the Laplacian matrix is that an iteration on the signal x of the form

$$x_1 = x_0 - \alpha L x_0, \tag{9.1}$$

will have the effect of creating a low-pass signal, since a high-pass form of the

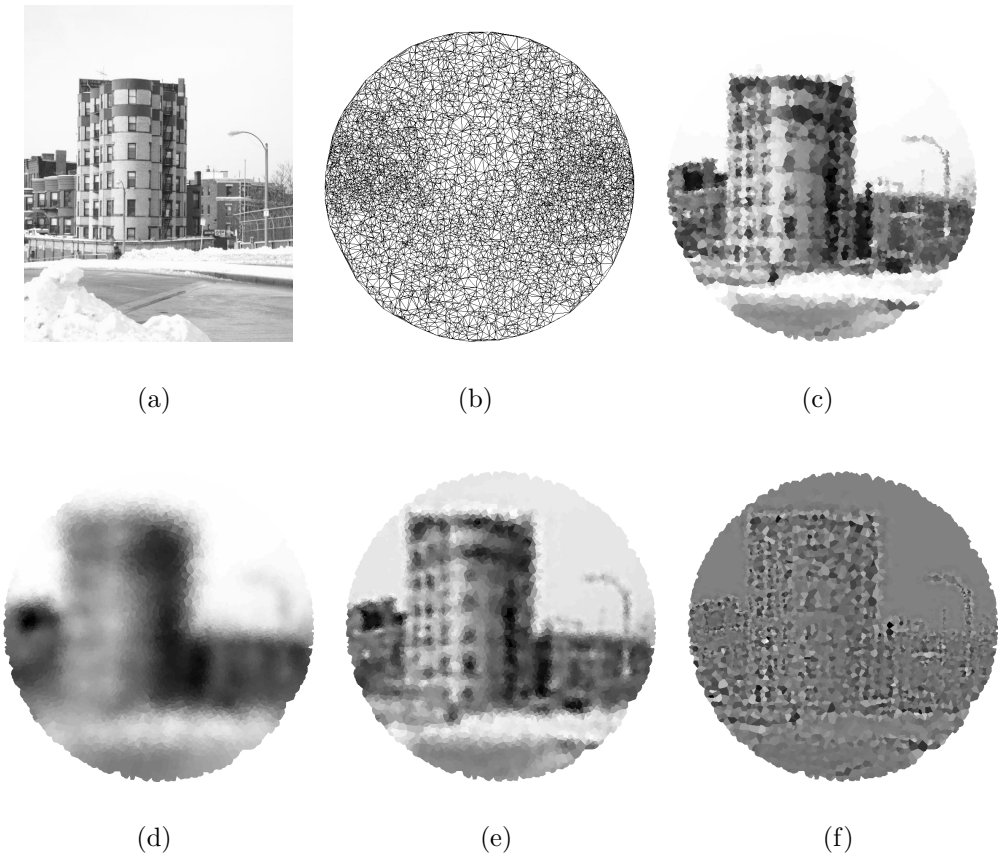


Figure 9.11: Filtering image data on a space-variant image. (a) The original image: `ESLab0059.jpg` (b) A space-variant graph patterned after the retinal ganglion cell distribution of the bottlenosed dolphin [46]. (c) The imported image, before any processing. (d) Result of the mean filter applied to the image in (c). (e) The low-pass $\lambda - \mu$ filter [68] applied to the image in (c). (f) A high-pass filter of (c), produced by differencing the low-pass signal of (e) with the original.

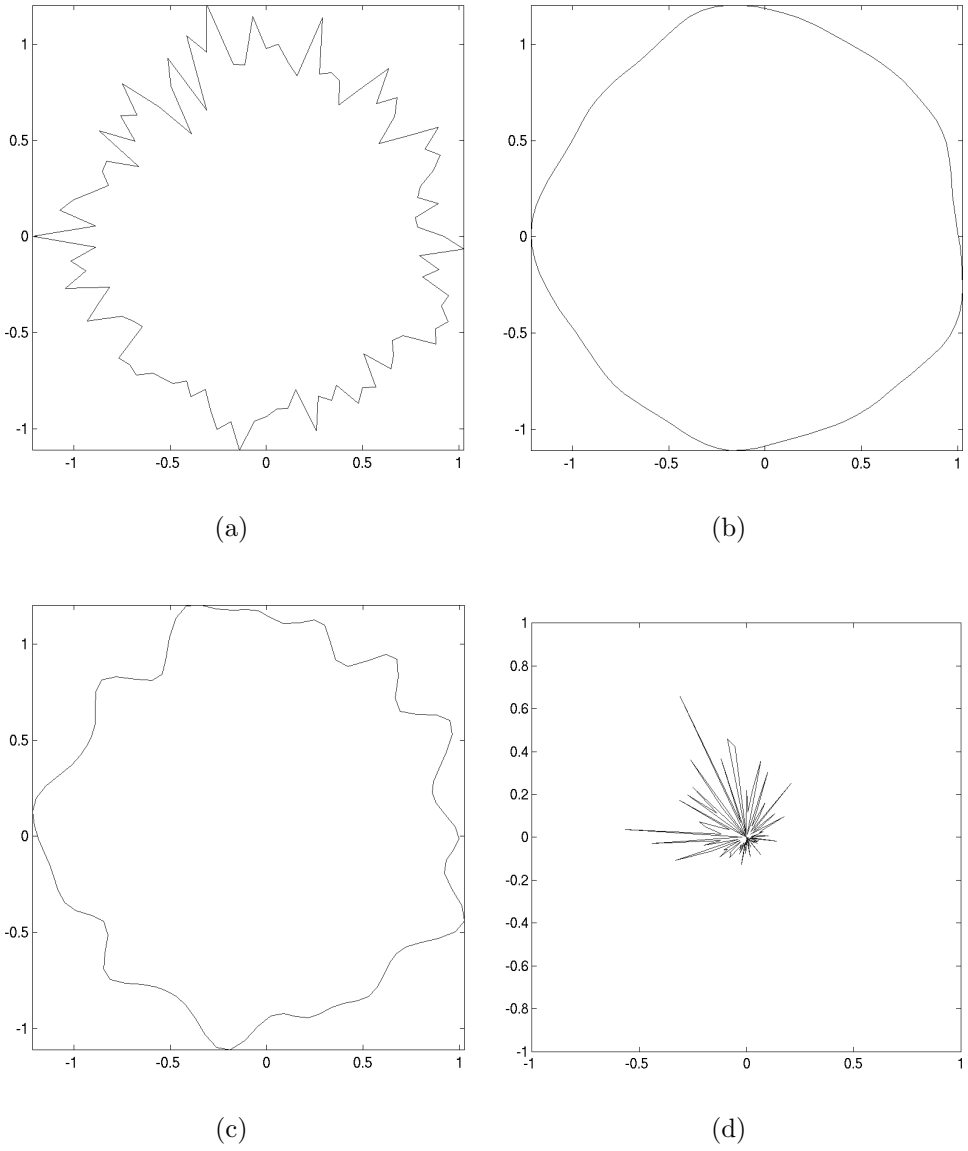


Figure 9.12: Filtering coordinate data on a ring graph. (a) A noisy ring graph produced by adding radial Gaussian distributed random to nodes arranged in a perfect circle. (b) The effect of applying the mean filter to the coordinates of the graph in (a). (c) The low-pass $\lambda-\mu$ filter [68] applied to the coordinates of the graph in (a). (d) A high-pass filter of the coordinates in (c), produced by differencing the low-pass signal of (c) with the original.

(a) (b) (c)

Figure 9.13: Diffusion filtering on an image. (a) The original image: `ES-Lab0059.jpg` (b) The effect of performing isotropic diffusion on the 4-connected lattice representing image (a). (c) The effect of performing anisotropic diffusion on the 4-connected lattice representing image (a).

signal is subtracted from the original. Since (9.1) represents one iteration of the diffusion equation

$$\frac{dx}{dt} = Lx, \tag{9.2}$$

both isotropic [42] and anisotropic [54] diffusion may be viewed as a low-pass filter. The function `diffusion.m` performs diffusion on a signal. Figure 9.13 shows an example of isotropic and anisotropic diffusion on an image.

The combinatorial Dirichlet problem method of interpolation presented above may also be viewed as a low-pass filter, since it requires the solution to a system of equations corresponding to the Laplace equation, constrained by Dirichlet boundary conditions. A solution to a system of equations $Lx = b$ may be viewed (if not computed) as $x = L^{-1}b$. The inverse of a matrix retains the same eigenvectors as the original, but the corresponding eigenvalues of the inverse matrix are the reciprocal of the eigenvalues of the original. Therefore, the solution to the constrained Laplace equation may be viewed as a low-pass filter since the lowest frequencies of the inverse of the (constrained) Laplacian matrix will correspond to the largest eigenvalues and *vice versa*. The relationship of the solution to a constrained Laplace equation to low-

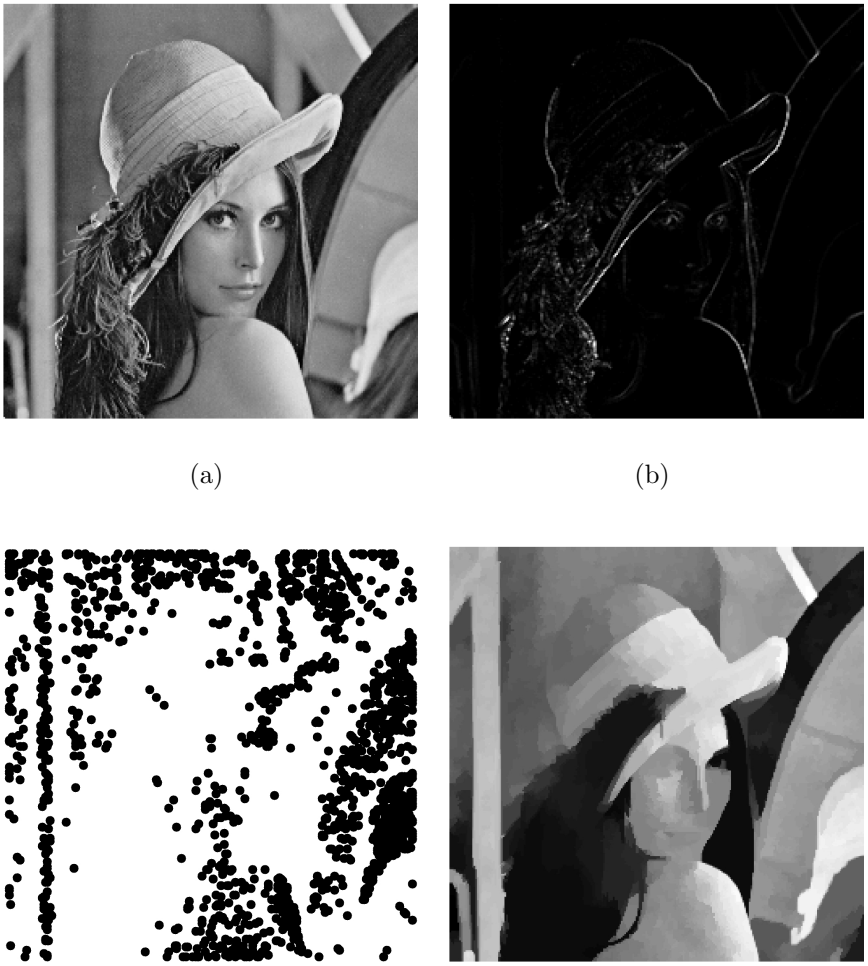
pass filtering and steady state diffusion with boundary conditions (see [26]) justifies its inclusion as a filtering method.

Figure 9.14 demonstrates anisotropic interpolation applied to low-pass filtering (i.e., smoothing) an image. To generate Figure 9.14, a 4-connected lattice was generated with the weight function of (8.1), based on the Lena image. Samples were chosen from relatively uniform areas by computing the sum of the edge gradients incident on each node. All nodes with gradient sums below a threshold were selected as sample nodes to have their values fixed. The remaining nodes were anisotropically interpolated, given the fixed set. One can see that sharp boundaries are maintained, due to the encoding of image information with weights. Areas of the image with high variability (e.g., the feathers) are smoothed considerably since very few samples were taken, while areas with initially low variability remain uniform.

10 Edge finding

Edge detection is a common goal of low-level computer vision. Common edge detection approaches [39] make use of gradient [57] or Laplacian [45] operators. An interesting feature of these operators is that although they both operate on values associated with the node set, the combinatorial gradient operator (the edge-node incidence matrix) returns values on the *edge set* while the combinatorial Laplacian returns values on the *node set*. This difference is analogous to 3D vector calculus in which the application of both the gradient and Laplacian operators to a scalar field results in a scalar field for the Laplacian and a vector field for the gradient. Standard gradient operators used for edge detection are applied to pixels and return values on pixels [57, 56, 23], causing the output of a typical gradient or Laplacian-based edge detection algorithm to be a set of edge-pixels.

The function `findedges.m` keeps with tradition by returning a set of edge pixels, regardless of whether a combinatorial gradient or Laplacian-based edge detection scheme is chosen. In the context of space-variant edge detection, we have found that better edge detection results are obtained by using an anisotropic edge operator, where the weights are based on the Euclidean distance of the point coordinates. The reason for this is that sensors (nodes) that are located more distant from each other are more likely to have an intensity change that crosses threshold, even if the continuous light distribution varies smoothly across a single object. Weighting the edge



(a) (b)

(c) (d)

Figure 9.14: Anisotropic interpolation used to low-pass filter an image. (a) Original Lena image. (b) Magnitude of summed image gradients. (c) Samples taken from lowest magnitude points. (d) Anisotropically interpolated image.

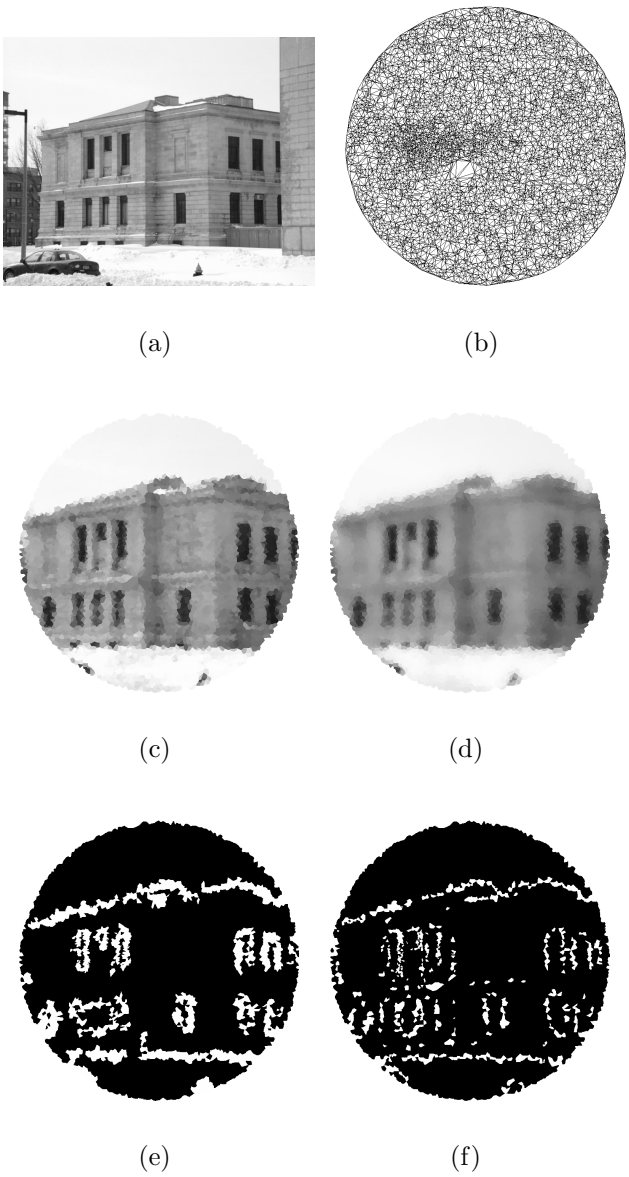


Figure 10.15: Edge detection on image data on a space-variant image. (a) The original image: `ESLab0032.jpg`. (b) A space-variant graph patterned after the retinal ganglion cell distribution of the labrador [38]. (c) The imported image, before any processing. (d) Result of anisotropic diffusion preprocessing used to sharpen edges and blur noise. (e) Result of gradient-based edge detection. (f) Result of Laplacian-based edge detection.

operator by distance compensates for this effect. Figure 10.15 demonstrates gradient and Laplacian-based edge detection on a space-variant image. In order to reduce noise but preserve edges, anisotropic diffusion was performed before the edge operator was applied. Note that different nodes correspond to different sized Voronoi regions in the visualization, due to the space-variance, which results in varying edge width.

11 Segmentation

The use of graph theory for data clustering and image segmentation may be traced to the work of Zahn on Gestalt clustering [84]. By framing the segmentation problem in the context of graph partitioning, Wu and Leahy developed the minimum cut algorithm [83]. Graph partitioning approaches to segmentation has led to several other algorithms [64, 53, 74, 61]. The function `partitiongraph.m` performs a graph bipartition using the isoperimetric algorithm [32], normalized cuts [64] or spectral partitioning [55]. Converting a graph bipartitioning algorithm to a complete segmentation may be accomplished by recursively applying the bipartitioning algorithm to each new segment and stopping the recursion when a specified metric of partition quality fails to be satisfied. The function `recursivepartition.m` recursively applies `partitiongraph.m` and returns integer labels of each node such that nodes sharing the same label are considered to be in the same partition. Since one often wants to apply these algorithms to standard Cartesian images, the functions `imgsegment.m` and `imgsegpyr.m` input standard images and return segmentations by using an underlying lattice or pyramid topology, respectively. The function `isosolve.m` performs the calculations for computing the potentials for a single application of the isoperimetric algorithm. An example of applying the segmentation algorithm to a space-variant image is given in figure 11.16.

12 Miscellaneous functions

This section details the minor functions included in the Graph Analysis Toolbox used to manipulate and visualize data.

Three functions used to perform minor graph and matrix manipulation are `adjtoedges.m`, `circulant.m` and `removeisolated.m`. The function `adj-`

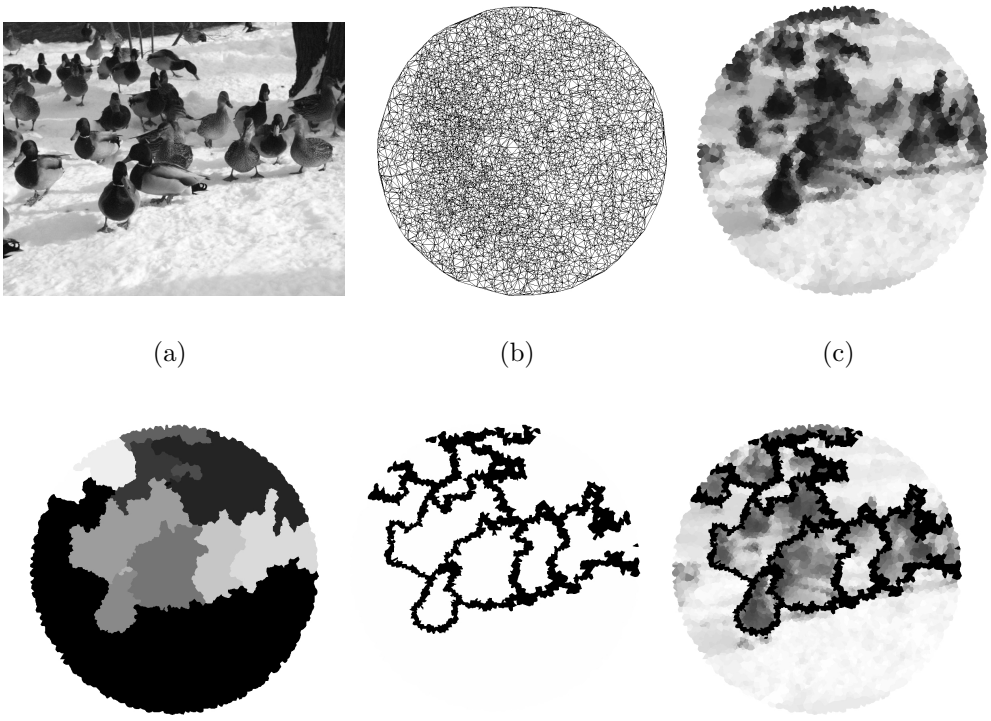


Figure 11.16: Segmentation of image data in a space-variant image. (a) The original image: `ESLab0043.jpg` (b) A space-variant graph patterned after the retinal ganglion cell distribution of the two-toed sloth [18]. (c) The imported image, before any processing. (d) Segment labels obtained using the isoperimetric algorithm [32] ($\beta^1 = 20, \beta^2 = 5, \text{stop} = 1 \times 10^{-8}$). (e) Segmentation result displayed as the outline of segments against a white background. (f) Segmentation result displayed as black outlines against the faded image.

`toedges.m` converts an adjacency matrix to an edge set. Since shift-invariant graphs correspond to circulant adjacency and Laplacian matrices, the function `circulant.m` is used to generate a sparse circulant matrix from one row that details the topology of a single node (which suffices to define the topology of the whole graph, since the graph is shift-invariant). The function `removeisolated.m` is used to remove any nodes which are not connected to any other node.

The functions `binarysearch.m`, `equalize.m` and `normalize.m` provide additional data processing. The function `binarysearch.m` implements a divide-and-conquer search algorithm of a sorted vector for the closest value of a given input. Normalization of the columns of a matrix to a specified range (defaulting to [0,1]) is accomplished by the function `normalize.m`. Histogram equalization of a vector is performed by the function `equalize.m`.

Four functions are available to aid in visualization of a segmentation and the production of segmentation results for publication. Three representations of a segmentation are returned by `segoutput.m` and `segoutputSV.m`: integer node labels, black outlines of segments against a white background and red outlines of segments superimposed on the original image. The special case of Cartesian images is handled by `segoutput.m`, while the general case of space-variant images is handled by `segoutputSV.m`. Since one may want to use the red-outlined segmentations returned by `segoutput.m` and `segoutputSV.m` in publication, the functions `colorseg2bwseg.m` and `colorseg2bwsegSV.m` convert the red-outlined segmentation to a black-outlined segmentation superimposed on a faded out copy of the original (so as not to confuse the segmentation lines with image features). Cartesian and space-variant segmentations are handled by `colorseg2bwseg.m` and `colorseg2bwsegSV.m` respectively.

Since there is a frequent need in the Graph Analysis Toolbox for vectorizing an RGB image, the function `rgbimg2vals.m` exists for this purpose.

13 Conclusion

The Graph Analysis Toolbox was developed to provide tools for space-variant computer vision that are independent of sampling regime or a chosen topology. Graph theory comprises the primary data structure, and combinatorial methods are the tools used to process the data. Since graph theoretic data structures appear in other disciplines with problems of analysis similar to

computer vision (e.g., segmentation, edge finding), we hope that the tools developed here may find a wider audience.

Despite the ubiquity of space-variant sensors in biological systems, it is rare to find artificial space-variant acquisition devices or displays. For this reason, the Graph Analysis Toolbox allows the simulation of a space-variant sensor, by providing tools to transfer images acquired with a Cartesian sensor array to an arbitrary space-variant representation. Likewise, tools are also available to allow the display of space-variant images on a standard CRT monitor. In addition to simulation of a space-variant sensor, the Graph Analysis Toolbox provides the simulation of an active vision system that allows the simulated space-variant architecture to be directed to different points in a larger image.

We hope that the Graph Analysis Toolbox will aid researchers in space-variant computer vision and other disciplines that take a combinatorial approach to graph theoretic structures.

1 Function list

This appendix is a reproduction of the `Contents.m` file.

Generating filters for space-variant graphs.	
`contour2pdf.m`	Convert a contour map to a probability density function.
`ellipsefit.m`	Fit an ellipse to a polygon with least-square error.
`findfilter.m`	Compute resampling filters for a point set.
I/O on space-variant graphs.	
`importimg.m`	Import a Cartesian (standard) image to a space-variant architecture.
`showmesh.m`	Visualize 2D data (e.g., an image) on a graph by interpolating data across the faces of the nodes.
`showvoronoi.m`	Visualize 2D data (e.g., an image) on a graph by uniformly filling the Voronoi cell of each node with its value.
`voronoicells.m`	Compute Voronoi information of a graph for visualization.
Data processing on graphs.	
`diffusion.m`	Diffuse data on a graph.
`dirichletboundary.m`	Solve the combinatorial Dirichlet problem on a graph (e.g., interpolate missing data).
`filtergraph.m`	Filter data on a graph.
`findedges.m`	Detect edges in data on a graph.
`imgsegment.m`	Segment a Cartesian (standard) image using a lattice.

`imgsegpyr.m`	Segment a Cartesian (standard) image using a pyramid.
`isosolve.m`	Perform the calculations required by the isoperimetric algorithm.
`makeweights.m`	Convert nodal graph data to edge weights.
`partitiongraph.m`	Segment data on an arbitrary graph.
`recursivepartition.m`	Recursively segment data on an arbitrary graph.

Generating node/edge sets for graphs.

`addrandedges.m`	Add random edges to "small worldify" a graph.
`latticepyramid.m`	Generate a connected pyramid from a Cartesian lattice.
`knn.m`	Connect nodes to their nearest neighbors.
`lattice.m`	Generate a Cartesian lattice with varying connectivity.
`logz.m`	Generate a point set using the $w = \log(z + a)$ function describing the macaque retinotopic map.
`roach.m`	Generate the "roach" graph of Guattery and Miller.
`triangulatepoints.m`	Compute an triangulated edge set for an input node set.

Graph matrix generation.

`adjacency.m`	Generate the adjacency matrix for a node/edge set.
`incidence.m`	Generate the incidence matrix for a node/edge set.
`laplacian.m`	Generate the Laplacian matrix for a node/edge set.

Support functions.

`adjtoedges.m`	Convert an adjacency matrix to an edge list.
`binarysearch.m`	Perform a binary search of a vector.
`circulant.m`	Generate a circulant matrix (similar to `toeplitz.m`).
`colorseg2bwseg.m`	Convert a segmentation indicated with color to a publishable (B&W) format.
`colorseg2bwsegSV.m`	Convert a space-variant segmentation indicated with color to a publishable (B&W) format.
`equalize.m`	Perform histogram equalization of a data vector.
`normalize.m`	Normalize data (columnwise) to a specified range.
`removeisolated.m`	Remove any isolated nodes in a graph.
`rgbimg2vals.m`	Vectorize an RGB image.
`segoutput.m`	Convert a segmentation labeling of a lattice to a better visualization.
`segoutputSV.m`	Convert a segmentation labeling of an arbitrary graph to a better visualization.

2 Demo scripts

This section provides a list of demo scripts included in the extended package. The information presented here is also included in the `ContentsDemo.m` file.

Edge finding.

`findEdgesDemo.m`	Compute edges for a Cartesian image using the gradient and Laplacian edge detectors.
`findEdgesDemoSV.m`	Compute edges for a space-variant image using the gradient and Laplacian edge detectors.

Graph filtering.

`diffusionDemo.m`	Compute iso/anisotropic diffusion on a Cartesian image.
`filterCoordDemo.m`	Filter coordinate data.
`filterImageDemo.m`	Filter a space-variant image.
`interpolationFilterDemo.m`	Use anisotropic interpolation as an image filter.

Graph drawing.

`drawGraphDemo.m`	Use isotropic interpolation to smooth a graph drawing.

Image interpolation.

`fovealAnisotropicDemo.m`	Perform anisotropic interpolation on a missing image region.
`fovealIsotropicDemo.m`	Perform isotropic interpolation on a missing image region.
`cartesianAnisotropicDemo.m`	Perform anisotropic interpolation based on sampling different regions of the image.

Importing/Visualization.

`buildFiltersDemo.m`	Generate importing filters for a random point set.
`contour2graphDemo.m`	Generate a graph from a retinal topography contour image.
`differentFoveationDemo.m`	Foveate on different points in a larger image.
`ellipseDisplayDemo.m`	Fit ellipses to Voronoi cells of a randomly generated point set.
`generateSVgraphsDemo.m`	Generate graphs and filters from the existing set of retinal topography images.
`importVisualizationDemo.m`	Import a Cartesian image to an `imgGraph` and visualize.

Segmentation.

`clusterPointsDemo.m`	Cluster a point set (segmentation on coordinates).

`segmentationSVDemo.m`	Segment an `imgGraph`.
`segmentationCompareDemo.m`	Compare segmentation of a Cartesian image generated by different algorithms.
Pyramids.	
`pyramidSegmentationDemo.m`	Compute a segmentation using a pyramid architecture.
Graph generation.	
`connectGraphDemo.m`	Computes and compares graphs with different topology and geometric arrangement.

3 Standardized variable names

Throughout the functions, documentation and demos, a set of standardized variable names are used. The list of variable names and their meanings is given below, and a reproduction of this list is included in the file `variableNames.txt`.

Scalars.	
`Q`	Cardinality of faces set in a graph.
`K`	Dummy constant.
`N`	Cardinality of node set in a graph.
`M`	Cardinality of edge set in a graph.
`P`	Number of coordinate dimensions of a node set.
`scale`	The weighting function parameter.
`stop`	The recursion stop parameter.
`X/Y/Z`	Dimensions of an image or region (e.g., `[X Y Z]=size(img)`).
Matrices.	
`W`	The $N \times N$ adjacency matrix.
`D`	The $N \times N$ diagonal matrix of node degrees.
`img`	The current image.
`L`	The $N \times N$ Laplacian matrix.
`A`	The $M \times N$ edge-node incidence matrix.

Vectors.

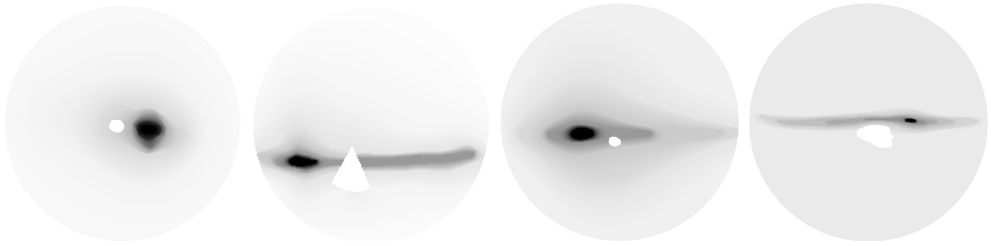
`d`	The $N \times 1$ vector of node degrees.
`vals`	The $N \times K$ vector of $K-$dimensional nodal values (e.g., RGB, with $K = 3$).
`weights`	The $M \times 1$ vector of edge weights.

Graph components.

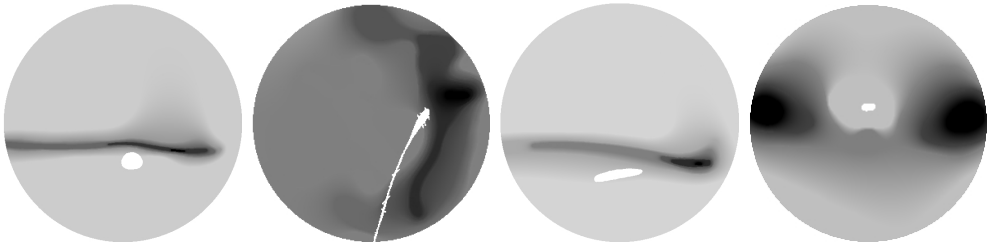
`points`	An $N \times P$ list of node coordinates.
`edges`	An $M \times 2$ list of edges (containing indices to the node set).
`faces`	A $Q \times K$ list of polygonal faces with order $\leq K$.

Structs.

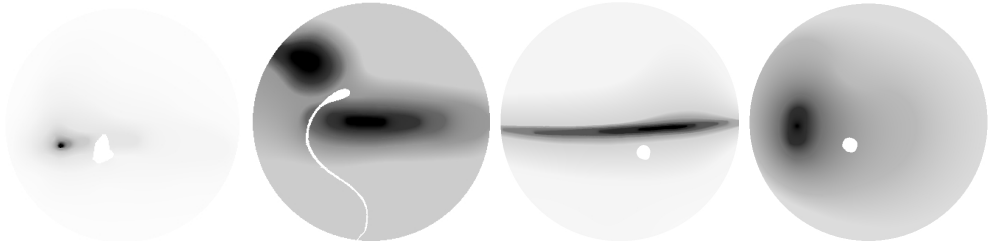
`imgGraph`	Struct containing the filters for importing an image to a space-variant graph.
`imgGraph.pntMap`	A $K \times 2$ list of the K points in the image plane used to filter an image for importing.
`imgGraph.breakpoints`	A $1 \times N$ list of the breakpoints in `imgGraph.pntMap` referring to the points corresponding to different nodes.
`imgGraph.filtWeights`	A $K \times 2$ list of the filter weights for each of the K points in `imgGraph.pntMap`.
`voronoiStruct`	Struct containing the information necessary to perform Voronoi visualization on a space-variant image.
`voronoiStruct.pts`	$K \times 2$ list of coordinates for the vertices of the Voronoi cells for the node set, where $K > N$.
`voronoiStruct.index`	List of nodes that are represented in the visualization (i.e., nodes with a Voronoi cell within the convex hull of the node set).
`voronoiStruct.faces`	List of faces representing the Voronoi cells, to be used by `patch.m`.



(a) Baboon [79] (b) Beagle [52] (c) Cat [37] (d) Cheetah [38]



(e) Cow [38] (f) Deep-sea bass [15] (g) Deer [38] (h) Bottlenosed dolphin [46]

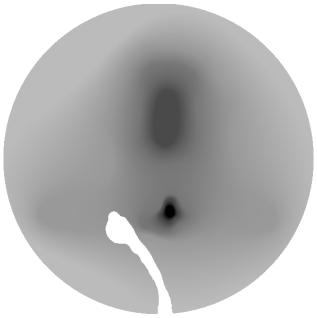


(i) German shepherd [52] (j) Harlequin tusk fish [16] (k) Plains kangaroo [36] (l) Tree kangaroo [36]

(m) Sacred kingfisher [50]

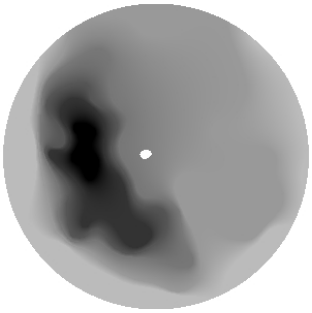
(n) Labrador [38]

(o) Pig [38]

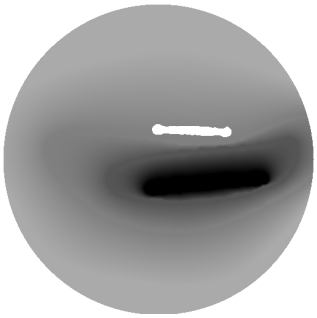


(p) Pigeon [79]

(q) Rabbit [35]



(r) Two-toed sloth [18]



(s) Squirrel [38]

(t) Wolf [52]

(u) Yellow-finned trevally [14]

4 Included retinal topographies

This section shows the probability density function (PDF) obtained from the retinal topographies of ganglion cell density for the species included in the Graph Analysis Toolbox. Darker areas represent higher ganglion cell density, while lighter areas represent a lower ganglion cell density. In addition to having different visual sampling arrangements, different species have varying degrees of nonuniformity in the sense that the discrepancy between the most dense and most sparse regions of ganglion cells may be $2:1$ in some species and $100:1$ in others. Since each image given below is normalized to have unity sum, the species with a greater ratio of dense to sparse areas are displayed as nearly white with a small dark region, while species having a lower ratio of dense to sparse regions are displayed as a more uniform gray. However, since the topographic maps for different species were studied by different researchers who stopped counting cell densities at different points in the retinal periphery, the topographies (and hence the images here) may or may not represent an actual comparison between species as regards the discrepancy between the region of highest cell density and the region of lowest cell density.

5 Acknowledgments

The authors would like to thank Jonathan Polimeni for many fruitful discussions and suggestions.

This work was supported in part by the Office of Naval Research (ONR N00014-01-1-0624).

Bibliography

[1] S. T. Acton, *A pyramidal edge detector based on anisotropic diffusion*, in 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings, vol. 4, Signal Process. Soc. IEEE, IEEE, May 1996, pp. 2215–2218.

[2] L. Ahlfors, *Complex Analysis*, McGraw-Hill, New York, 1966.

[3] W. N. J. Anderson and T. D. Morley, *Eigenvalues of the laplacian of a graph*, Tech. Report TR 71-45, University of Maryland, October 1971.

[4] S. T. Barnard and H. D. Simon, *A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, Tech. Report RNR-92-033, NASA Ames Research Center, Moffett Field, CA, March 1994.

[5] N. Biggs, *Algebraic Graph Theory*, no. 67 in Cambridge Tracts in Mathematics, Cambridge University Press, 1974.

[6] G. Bonmassar and E. Schwartz, *Space variant fourier analysis: The exponential chirp transform*, IEEE Pattern Analysis and Machine Intelligence, 19 (1997), pp. 1080–1089.

[7] F. H. Branin Jr., *The algebraic-topological basis for network analogies and the vector calculus*, in Generalized Networks, Proceedings, Brooklyn, N.Y., April 1966, pp. 453–491.

[8] P. A. Burrough, *Principles of geographical information systems for land resources assessment*, no. 12 in Monographs on soil and resources survey, Clarendon Press, Oxford, 1986.

[9] P. J. Burt, T.-H. Hong, and A. Rosenfeld, *Segmentation and estimation of image region properties through cooperative hierarchical computation*, IEEE Transactions on Systems, Man and Cybernetics, SMC-11 (1981), pp. 802–809.

[10] D. Casasent and D. Psaltis, *Position, rotation and scale invariant optical correlation*, Applied Optics, (1976).

[11] G.-N. Chen, *Fundamental Algorithms of Space-Variant Vision: Non-Uniform Sampling, Triangulation, and Foveal Scale-Space*, PhD thesis, Boston University, 2001.

[12] W. Chen and S. Acton, *Morphological pyramids for multiscale edge detection*, in 1998 IEEE Southwest Symposium on Image Analysis and Interpretation, Univ. Arizona, Univ. Arizona Found., Arizona State Univ., Tucson Sect. IEEE, IEEE, April 1998, pp. 137–141.

[13] F. R. K. Chung, *Spectral Graph Theory*, no. 92 in Regional conference series in mathematics, American Mathematical Society, Providence, R.I., 1997.

[14] S. P. Collin, *Behavioural ecology and retinal cell topography*, in Adaptive Mechanisms in the Ecology of Vision, S. Archer, M. Djamgoz, E. Loew, J. Partridge, and D. S. Vallerga, eds., Kluwer Academic Publishers, 1999, pp. 509–535.

[15] S. P. Collin and J. C. Partridge, *Retinal specializations in the eyes of deep-sea teleosts*, Journal of Fish Biology, 49 (Supplement A) (1996), pp. 157–174.

[16] S. P. Collin and J. D. Pettigrew, *Retinal ganglion cell topography in teleosts: A comparison between nissl-stained material and retrograde labelling from the optic nerve*, The Journal of Comparative Neurology, 276 (1988), pp. 412–422.

[17] M. L. Comer and E. J. Delp, *Multiresolution image segmentation*, in 1995 International Conference on Acoustics, Speech, and Signal Processing. Conference Proceedings, vol. 4, Signal Process. Soc. IEEE, IEEE, May 1995, pp. 2415–2418.

[18] B. L. S. A. Costa, V. F. Pessoa, J. D. Bousfield, and R. J. Clarke, *Ganglion cell size and distribution in the retina of the two-toed sloth (choleopus didactylus l.)*, Brazilian Journal of Medical and Biological Research, 22 (1989), pp. 233–236.

[19] R. Courant, *Dirichlet's Principle*, Interscience, 1950.

[20] R. Courant and D. Hilbert, *Methods of Mathematical Physics*, vol. 1 of Wiley Classics Libary, John Wiley and Sons, 1989.

[21] ——, *Methods of Mathematical Physics*, vol. 2 of Wiley Classics Libary, John Wiley and Sons, 1989.

[22] D. M. Cvetković, M. Doob, and H. Sachs, *Spectra of Graphs: Theory and Applications*, Johann Ambrosious Barth Verlag, Heidelberg–Leipzig, Germany, 3rd revised and enlarged edition ed., March 1995.

[23] L. S. Davis, *A survey of edge detection techniques*, Computer Graphics and Image Processing, 4 (1975), pp. 248–270.

[24] J. Dodziuk, *Difference equations, isoperimetric inequality and the transience of certain random walks*, Transactions of the American Mathematical Soceity, 284 (1984), pp. 787–794.

[25] J. Dodziuk and W. S. Kendall, *Combinatorial laplacians and the isoperimetric inequality*, in From local times to global geometry, control and physics, K. D. Ellworthy, ed., vol. 150 of Pitman Research Notes in Mathematics Series, Longman Scientific and Technical, 1986, pp. 68–74.

[26] P. Doyle and L. Snell, *Random walks and electric networks*, no. 22 in Carus mathematical monographs, Mathematical Association of America, Washington, D.C., 1984.

[27] J. H. Elder and R. M. Goldberg, *Image editing in the contour domain*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 23 (2001), pp. 291–296.

[28] D. Eppstein, *Sparsification–a technique for speeding up dynamic graph algorithms*, in Proceedings 33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, PA, October 1992, IEEE, AT&T Bell Lab., IBM, Istituto di Analisi dei Sistemi Ed Inf., Xerox, IEEE Computer Soceity Press, pp. 60–69.

[29] M. Fiedler, *Eigenvalues of acyclic matrices*, Czechoslovak Mathematical Journal, 25 (1975), pp. 607–618.

[30] J. Gilbert, C. Moler, and R. Schreiber, *Sparse matrices in matlab: Design and implementation*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 333–356.

[31] L. Grady and E. Schwartz, *Anisotropic interpolation on graphs: The combinatorial dirichlet problem*, Tech. Report CAS/CNS-TR-03-014, Boston University, Boston, MA, July 2003. Submitted to IEEE Pattern Analysis and Machine Intelligence.

[32] ——, *Isoperimetric graph partitioning for data clustering and image segmentation*, Tech. Report CAS/CNS-TR-03-015, Boston University, Boston, MA, July 2003. Submitted to IEEE Pattern Analysis and Machine Intelligence.

[33] S. Guattery and G. Miller, *On the quality of spectral separators*, SIAM Journal on Matrix Analysis and Applications, 19 (1998), pp. 701–719.

[34] P. Heckbert, *Fundamentals of texture mapping and image warping*, master's thesis, University of California at Berkeley, 1989.

[35] A. Hughes, *Topographical relationships between the anatomy and physiology of the rabbit visual system*, Documents Ophthalmology, 30 (1971), pp. 33–159.

[36] ——, *A comparison of retinal ganglion cell topography in the plains and tree kangaroo*, Journal of Physiology, 244 (1974), pp. 61–63P.

[37] ——, *A quantitative analysis of cat retinal ganglion cell topography*, Journal of Comparative Neurology, 163 (1975), pp. 107–128.

[38] ——, *The topography of vision in mammals of contrasting life style: Comparative optics and retinal organization*, in The Handbook of Sensory Physiology, vol. 7, New York: Academic Press, 1977.

[39] A. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, Inc., 1989.

[40] A. K. Jain, M. N. Murty, and P. J. Flynn, *Data clustering: a review*, ACM Computing Surveys, 31 (1999), pp. 264–323.

[41] G. Kirchhoff, *Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen verteilung galvanischer ströme geführt wird*, Poggendorf's Annalen der Physik Chemie, 72 (1847), pp. 497–508.

[42] J. Koenderink, *The structure of images*, Biological Cybernetics, 50 (1984), pp. 363–370.

[43] C.-Y. Lin and e. a. Wu, Min, *Rotation, scale and translation reslilient watermarking for images*, IEEE Transactions on Image Processing, (2001).

[44] S. Loncaric, *A survey of shape analysis techniques*, Pattern Recognition, 31 (1998), pp. 983–1001.

[45] D. Marr and E. C. Hildreth, *Theory of edge detection*, Proceedings of the Royal Soceity of London, 207 (1980), pp. 187–217.

[46] A. M. Mass and A. Y. Supin, *Ganglion cell topography of the retina in the bottlenosed dolphin, tursiops truncatus*, Brain Behavior and Evolution, 45 (1995), pp. 257–265.

[47] R. Merris, *Laplacian matrices of graphs: A survey*, Linear Algebra and its Applications, 197,198 (1994), pp. 143–176.

[48] B. Mohar, *Isoperimetric inequalities, growth and the spectrum of graphs*, Linear Algebra and its Applications, 103 (1988), pp. 119–131.

[49] ——, *The laplacian spectrum of graphs*, in Graph Theory, Combinatorics, and Applications, Y. Alavi, G. Chartrand, O. R. Oellermann, and A. J. Schwenk, eds., vol. 2, Wiley, 1991, pp. 871–898.

[50] M. K. Moroney and J. D. Pettigrew, *Some observations in the visual optics of kingfishers (aves, coraciformes, alcedinidae)*, Journal of Comparative Physiology A, 160 (1987), pp. 137–149.

[51] C. e. a. Pachai, *A pyramidal approach for automatic segmentation of multiple sclerosis lesion in brain mri*, Computerized Medical Imaging and Graphics, 22 (1998), pp. 399–408.

[52] L. Peichl, *Topgraphy of ganglion cells in the dog abnd wolf retina*, The Journal of Comparative Neurology, 324 (1992), pp. 603–620.

[53] P. Perona and W. Freeman, *A factorization approach to grouping*, in Computer Vision - ECCV'98. 5th European Conference on Computer Vision. Proceedings, B. Burkhardt, H.; Neumann, ed., vol. 1, Freiburg, Germany, 2–6 June 1998, Springer-Verlag, pp. 655–670.

[54] P. Perona and J. Malik, *Scale-space and edge detection using anisotropic diffusion*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 12 (1990), pp. 629–639.

[55] A. Pothen, H. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal of Matrix Analysis Applications, 11 (1990), pp. 430–452.

[56] J. M. S. Prewitt, *Object enhancement and extraction*, in Picture Processing and Psychopictorics, B. S. Lipkin and A. Rosenfeld, eds., Academic Press, New York, 1970, pp. 75–149.

[57] L. G. Roberts, *Machine perception of three-dimensional solids*, in Optical and Electro-optical Information Processing, J. T. Tippet, ed., MIT Press, Cambridge, MA, May 1965, pp. 159–197.

[58] J. Roth, *An application of algebraic topology to numerical analysis: On the existence of a solution to the network problem*, Proceedings of the National Academy of Science of America, (1955), pp. 518–521.

[59] G. Sandini, F. Bosero, F. Bottino, and A. Ceccherini, *The use of an anthropomorphic visual sensor for motion estimation and object tracking*, in Image Understanding and Machine Vision 1989. Technical Digest Series, Vol.14. Conference Edition, vol. 14 of Technical Digest Series, North Falmouth, MA, June 1989, Optical Society of America, AFOSR, Optical Society of America, pp. 68–72.

[60] G. Sandini, P. Questa, D. Scheffer, and A. Mannucci, *A retina-like cmos sensor and its applications*, in IEEE Sensor Array and Multichannel Signal Processing Workshop, Cambridge, March 16-17 2000, IEEE.

[61] S. Sarkar and P. Soundararajan, *Supervised learning of large perceptual organization: Graph spectral partitioning and learning automata*, IEEE Trans. on Pattern Analysis and Machine Intelligence, 22 (2000), pp. 504–525.

[62] E. L. Schwartz, *Spatial mapping in the primate sensory projection: analytic structure and relevance to perception*, Biological Cybernetics, 25 (1977), pp. 181–194.

[63] J. R. Shewchuk, *Triangle: engineering a 2d quality mesh generator and delaunay triangulator*, in Applied Computational Geometry. Towards Geometric Engineering. FCRC'96 Workshop, WACG'96. Selected Papers, M. C. Lin and D. Manocha, eds., Philadelphia, PA, 27–28 May 1996, ACM, Springer-Verlag, pp. 203–222.

[64] J. Shi and J. Malik, *Normalized cuts and image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22 (2000), pp. 888–905.

[65] P. Soille, *Morphological Image Analysis: Principles and Applications*, Springer-Verlag, 2nd ed., June 1999.

[66] G. Strang, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, 1986.

[67] S. H. Strogatz, *Exploring complex networks*, nature, 410 (2001), pp. 268–276.

[68] G. Taubin, *A signal processing approach to fair surface design*, in Computer Graphics Proceedings. SIGGRAPH 95, R. Cook, ed., Los Angeles, CA, Aug. 1995, ACM, ACM, pp. 351–358.

[69] G. Taubin, T. Zhang, and G. Golub, *Optimal surface smoothing as filter design*, Tech. Report RC-20404, IBM, March 1996.

[70] M. Unser, *Sampling - 50 years after shannon*, Proceedings of the IEEE, 68 (2000), pp. 569–587.

[71] R. Wallace, P.-W. Ong, and E. Schwartz, *Space variant image processing*, International Journal of Computer Vision, 13 (1994), pp. 71–90.

[72] G. Walls, *The Veterbrate Eye*, Michigan: The Cranbrook Press, 1942.

[73] B. Wandell, S. Chial, and B. T. Backus, *Visualization and measurement of the cortical surface*, Journal of Cognitive Neuroscience, 12 (2000), pp. 739–752.

[74] S. Wang and J. M. Siskund, *Image segmentation with ratio cut*, IEEE Trans. on Pattern Analysis and Machine Intelligence, 25 (2003), pp. 675–690.

[75] F. W. Warner, *Foundations of Differentiable Manifolds and Lie Groups*, Graduate Texts in Mathematics, Spring-Verlag, 1983.

[76] D. Watts and S. Strogatz, *Collective dynamics of 'small-world' networks*, Nature, 393 (1998), pp. 440–442.

[77] D. J. Watts, *Small worlds: the dynamics of networks between order and randomness*, Princeton studies in complexity, Princeton University Press, Princeton, N.J., 1999.

[78] H. Weyl, *Reparticion de corriente en una red conductora*, Revista Matemática Hispano-Americans, 5 (1923), pp. 153–164.

[79] D. Whitteridge, *Geometrical relations between the retina and the visual cortex*, in Mathematics and computer science in biology and medicine; proceedings of conference held by Medical Research Council in association with the health departments: Oxford, July 1964, Medical Research Council, London, H.M. Stationery Off., 1965, pp. 269–276.

[80] A. Witkin, *Scale-space filtering*, in Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Aug 8–12 1983, pp. 1019–1022.

[81] J. Wood and P. Fisher, *Assessing interpolation accuracy in elevation models*, IEEE Computer Graphics and Applications, 13 (1993), pp. 48–56.

[82] A. S. Wright and S. T. Acton, *Watershed pyramids for edge detection*, in Proceedings. International Conference on Image Processing, vol. 2, IEEE Signal Process. Soc, IEEE Comput. Soc, 1997, pp. 578–581.

[83] Z. Wu and R. Leahy, *An optimal graph theoretic approach to data clustering: theory and its application to image segmentation*, IEEE Pattern Analysis and Machine Intelligence, 11 (1993), pp. 1101–1113. 546.

[84] C. Zahn, *Graph theoretical methods for detecting and describing gestalt clusters*, IEEE Transactions on Computation, 20 (1971), pp. 68–86.

[85] R. Zhang, P.-S. Tsai, J. E. Cryer, and M. Shah, *Shape-from-shading: a survey*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 21 (1999), pp. 690–706.