Chapter 3

Spectral Embedding

In the previous chapter, we discussed algorithms that use S-T min-cut to perform segmentation ([116], [11], [13], [105]). These algorithms have several properties that make them attractive. First, they isolate the segmentation algorithm per se from the details about the particular similarity measure being used to compare pixels, this is useful because it means that the segmentation algorithm can be easily generalized to work on images from different domains, where particular cues may have different relative importance. Second, they are efficient; the minimum S-T cut for any given source and sink regions can be computed in nearly linear time (see Boykov and Kolmogorov [11]). Third, the cut obtained in this way is guaranteed to be globally optimal (see Wu and Leahy [116]), and for suitably defined source and sink regions, the cut will very likely correspond to a salient image boundary separating the source from the sink.

The problem, as we saw before, lies in identifying suitable regions to use as source and sink. Previous algorithms perform S-T cuts with individual pixels in the image and produce a segmentation from the resulting cuts, or they rely on user interaction to define the source and sink regions. Either of these choices has its drawbacks. Using individual pixels tends to produce small regions whose border does not correspond to any perceptually salient image boundary; it also requires a large number of calls to the min-cut procedure. Relying on user

interaction, on the other hand, preempts the creation of an automatic segmentation algorithm.

In this chapter, we will discuss the general properties of the min-cut algorithm, talk about the problem of defining suitable seed regions, and describe the properties these seed regions should have in order to be useful for minimum S-T cut segmentation. We will then lay the foundations for Spectral Embedding, a general data clustering technique. We will develop a connection between spectral embedding and anisotropic smoothing kernels, and show that the clustering offered by spectral embedding makes the technique ideal for obtaining candidate seed regions for S-T min-cut.

In the next chapter, we will describe a segmentation algorithm that uses seed regions obtained using spectral embedding to create source and sink combinations for S-T min-cut.

3.1 The Min-Cut Framework

The starting point of our method is the affinity matrix A that contains the similarities between neighboring pixels in the image. For a given image $I(\vec{x})$ of $n \times m$ pixels, A is an $nm \times nm$ symmetric matrix. The values in the affinity matrix are assumed to satisfy $A(i,j) \in [0,1]$. The value of 1 represents perfect similarity between two pixels, while 0 indicates the absence of a link between the corresponding pixels. The existence or absence of a link between two pixels is determined by the definition of a pixel's neighborhood, for example, if the neighborhood is taken to be the whole image, every pixel will be linked to every other pixel, and every entry in the affinity matrix will be non-zero. In practice, however, we use small neighborhoods centered on a particular pixel, and assume that all pixels beyond this neighborhood are not linked to the pixel in question. The advantage of using a small neighborhoods is that the resulting affinity matrix is sparse; this is helpful because (as long as the pixel neighborhood is reasonably small) it significantly reduces the amount of memory required to store the affinity matrix, and facilitates the eigen-decomposition of the matrix.

Our goal here is to study the segmentation process given such an affinity matrix. For this

reason, we will not explore the problem of defining the affinity measure. As discussed in the previous chapter, finding a good similarity measure is an important and difficult problem, and we can expect that better, more comprehensive similarity measures will only improve the segmentation results we present here. For the purposes of our algorithm, we use affinity matrices that are created using the standard 8-neighborhood structure. The similarity function is very simple, and is based entirely on the difference in gray-level intensity between two neighboring pixels x_i and x_j:

$$A_{i,j} = \exp -(I(x_i) - I(x_j))^2/(2\sigma^2), \tag{3.1}$$

where σ represents the typical gray-level variation between similar pixels due to image noise, and we set $A_{i,i} = 1$. We will discuss the choice of σ further on in this chapter. The analysis that follows requires only that the affinity matrix be non-negative, and that the entries along the main diagonal be non-zero.

Given this affinity matrix our task is to determine appropriate seed regions that can be used to create source and sink combinations for the min S-T cut algorithm. These regions must have several properties to ensure that the resulting segmentation will be meaningful. The first property is that a seed region must be sufficiently large, consider Figure 3.1b, the image illustrates what occurs when the source region is too small (only one pixel in this case). The minimum cut algorithm returns a segmentation in which the source region is simply cut from the rest of the image, this happens because even though the source is strongly linked to its neighbors, the total cost of cutting the few strong edges linking the source to its neighbors is smaller than the cost of cutting the many weak links that separate the source from the sink along a real image boundary.

The minimum cut procedure will only find a boundary when the cost of cutting along the boundary is lower than the cost of separating the source or sink from the rest of the image. Increasing the size of sources and sinks encourages the minimum cut procedure to cut along salient image boundaries, this is illustrated in Figure 3.1d.

The second condition is that individual seed regions must not cross perceptually significant

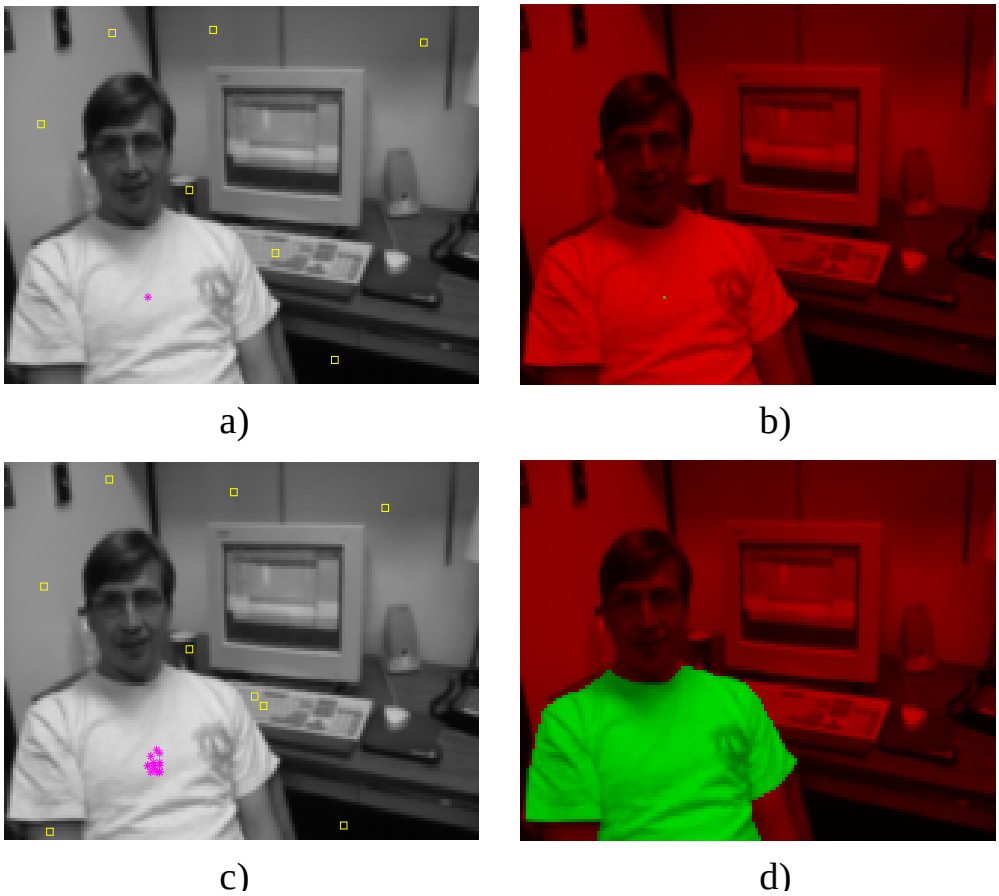

Figure 3.1: Effects of seed size on min-cut segmentation, a) 1 pixel source (magenta), and sink pixels (yellow), b) min-cut result, the pixel is cut out from its neighbors, c) enlarged source region (magenta) and sink pixels (yellow), d) min-cut result, this time the cut follows the actual image boundary. The original image is 160×120 pixels.

a) b)

Figure 3.2: Effect of seed crossing an image boundary, a) source region (magenta) and sink pixels (yellow), notice that the source spills across two separate regions. b) min-cut result, under-segmentation occurs. The original image is 160×120 pixels.

boundaries, Figure 3.2 illustrates the problem. Since the min-cut algorithm can't split a seed region, any seed regions that cross image boundaries will cause under-segmentation. As a result of this, source and sink combinations formed with subsets of seed regions should be such that the seed regions for the source, and the seed regions for the sink, come from disjoint unions of natural image segments (in other words, seed regions detected within the same natural image segment should go together into either the source or the sink).

The third and final condition is that the set of seed regions must be rich enough to allow for each natural image segment to be separated from the rest of the image. Given that the source and sink are formed with combinations of seed regions, any natural segment for which there is no corresponding seed will be joined to the regions extracted from other source/sink combinations, and will never appear on its own. In the following sections we will discuss how random walks and spectral embedding can be used to generate seed regions that satisfy the above conditions.

3.2 Random Walks Based on Pixel Similarity

We will now examine the properties of a random walk defined over the image using the affinities stored in the matrix A. Suppose that a particle is at pixel $\vec{x}_j$ at time t of the random walk, the probability $p_{i,j}$ that the particle jumps to pixel $\vec{x}_i$ at time $t+1$ is proportional to the affinity $A_{i,j}$ between the two pixels. In order to transform the affinity values into transition probabilities, we must normalize them so that for any $\vec{x}_j$, $\sum_{i=1}^{nm} p_{i,j} = 1$. From this condition it follows that $p_{i,j} = A_{i,j}/D_j$, where $D_j = \sum_{k=1}^{nm} A_{k,j}$.

We define a diagonal matrix D whose entries are the normalization factors D_j, and build a Markov matrix M with entries $M_{i,j} = A_{i,j}/D_j = p_{i,j}$ as

$$M = AD^{-1}, \tag{3.2}$$

this matrix is positive but generally not symmetric; notice that every column of M adds up to 1. Consider now a probability distribution $\vec{p}_t$ whose j^{th} entry $p_t(\vec{x}_j)$ represents the probability that the particle undergoing the random walk is at pixel $\vec{x}_j$ at time t, and pixels are stored in $\vec{p}_t$ in raster order. The probability distribution of the particle at time $t+1$ is given by

$$\vec{p}_{t+1} = M\vec{p}_t. \tag{3.3}$$

Given an initial distribution $\vec{p}_0$, it follows from (3.3) that the distribution at time t is given by

$$\vec{p}_t = M^t\vec{p}_0, \tag{3.4}$$

we shall see that there is a simple way of representing M^t using the eigenvectors and eigenvalues of M, but first, let us examine some of the properties of M and its eigen-decomposition.

In what follows, we will assume that the matrix M is irreducible; this means that for sufficiently large t, $p_{i,j}^t > 0$ for all i, j. A particle undergoing the random walk must be able to reach any pixel starting anywhere in the image within a finite time. Markov matrices that are not irreducible should first be processed to extract each connected subset of pixels, each of which can then be turned into an appropriate, irreducible Markov matrix (this can be accomplished

by performing connected-components analysis on the affinity matrix). Additionally, since the diagonal of the affinity matrix A contains only non-zero elements, the resulting Markov matrix will have positive trace. Under these conditions, the eigenvalues and eigenvectors of our Markov matrix have several interesting properties.

Given the fact that the columns of M add up to 1, it follows that $\vec{1}^T M = \vec{1}^T$. Thus, $\lambda = 1$ is always an eigenvalue of M. For an irreducible Markov matrix with positive trace, the Perron-Frobenius theorem (see Ch.8 in Meyer [72]) states that $\lambda = 1$ is the eigenvalue with largest magnitude. All other eigenvalues are real and their magnitude is less than 1 (see Ch.7 in Chennubhotla [17]). The Perron-Frobenius theorem also states that the largest eigenvalue is associated with a positive eigenvector. This eigenvector corresponds to the stationary distribution $\vec{\pi}$ of the random walk [55], which has the property that $M\vec{\pi} = \vec{\pi}$. The stationary distribution specifies the long-term behavior of the random walk, $\vec{\pi}(\vec{x}_j)$ is the probability that the particle will find itself at pixel $\vec{x}_j$ as $t \to \infty$. In other words, for very large t, $\vec{\pi} \simeq M^t \vec{p}_0$.

In our case, the eigenvectors and eigenvalues of M can be conveniently obtained from the *similar, symmetric* matrix $L = D^{-1/2} M D^{1/2} = D^{-1/2} A D^{-1/2}$. Since L is symmetric, its eigen-decomposition has the form $L = U \Delta U^T$, where U is an orthogonal matrix whose columns are the eigenvectors of L, and Δ is a diagonal matrix that contains the corresponding eigenvalues $\lambda_i, i = 1, \ldots, nm$. We assume that the eigenvalues have been sorted in decreasing order of absolute magnitude, so that $|\lambda_i| \geq |\lambda_{i+1}|$ for all $i \geq 1$. Since every eigenvalue λ_i of a the matrix must satisfy $|\lambda_i| \leq 1$, and as mentioned above, there is a unique eigenvalue equal to one, we can assume without loss of generality that $\lambda_1 = 1$. Finally, it is shown in [17] that the eigenvector of L associated with the eigenvalue $\lambda = 1$ is given by

$$\vec{u}_1^T = (u_{1,1}, u_{1,2}, \ldots, u_{1,nm}), \quad \text{with} \quad u_{1,j} = \frac{1}{\alpha} D_j^{1/2}, \quad \text{and} \quad \alpha = \sqrt{\sum_{k=1}^{nm} D_k}. \tag{3.5}$$

Using the above factorization for M and L, we can write

$$\vec{p}^{\,t} = M^t \vec{p}_0 = D^{1/2} L^t D^{-1/2} \vec{p}_0 = (D^{1/2} U) \Delta^t (U^T D^{-1/2}) \vec{p}_0. \tag{3.6}$$

The above construction has two additional properties that we will use. First, from (3.5)

$$\vec{1}^T D^{1/2} U = \alpha \vec{u}_1^T U = \alpha(1, 0, 0, \ldots, 0), \tag{3.7}$$

where $\vec{1}$ is the vector of all ones. Secondly, from (3.5) and (3.7) we have

$$\vec{1}^T \vec{p_t} = \alpha(1, 0, 0, \ldots, 0)\Delta^t U^T D^{-1/2} \vec{p_0} = \alpha u_1^T D^{-1/2} \vec{p_0} = \vec{1}\vec{p_0} = 1$$

so the distribution remains properly normalized.

Figure 3.3 shows two images, and the leading 7 eigenvectors for each image, the first eigenvector corresponds to the stationary distribution, while succeeding eigenvectors correspond to the dominant perturbations to this stationary distribution. The magnitude of the eigenvalue associated to each eigenvector determines how quickly the associated perturbation disappears as the number of iterations in the random walk increases. For a sufficiently large number of iterations, even the strongest perturbations die off, and the random walk approaches its stationary distribution. In general, the leading eigenvectors capture coarse properties of the random walk, while successive eigenvectors capture progressively finer interactions. Thus, it is the smallest, local perturbations that vanish first, while larger scale perturbations relax more slowly.

It is important to mention that the eigenvectors of the Markov matrix depend on the choice of σ in our affinity function. A large value of σ yields smoother eigenvectors (as sigma increases, the steady state distribution for the random walk gets closer and closer to a uniform distribution, and the perturbations to this distribution are weaker). A small sigma, on the other hand, yields many leading eigenvectors that capture strong, local perturbations caused by individual pixels that stand out from their immediate neighborhood. Figure 3.4 illustrates both of these undesirable situations. For practical purposes, we chose experimentally a value of σ that yields useful eigenvectors for images of the scale used in our experiments (between 80×80 and 200×200 pixels in size). We have found that for images such as the one in Figure 3.3i, and others of similar size, $\sigma = 7$ yields good results.

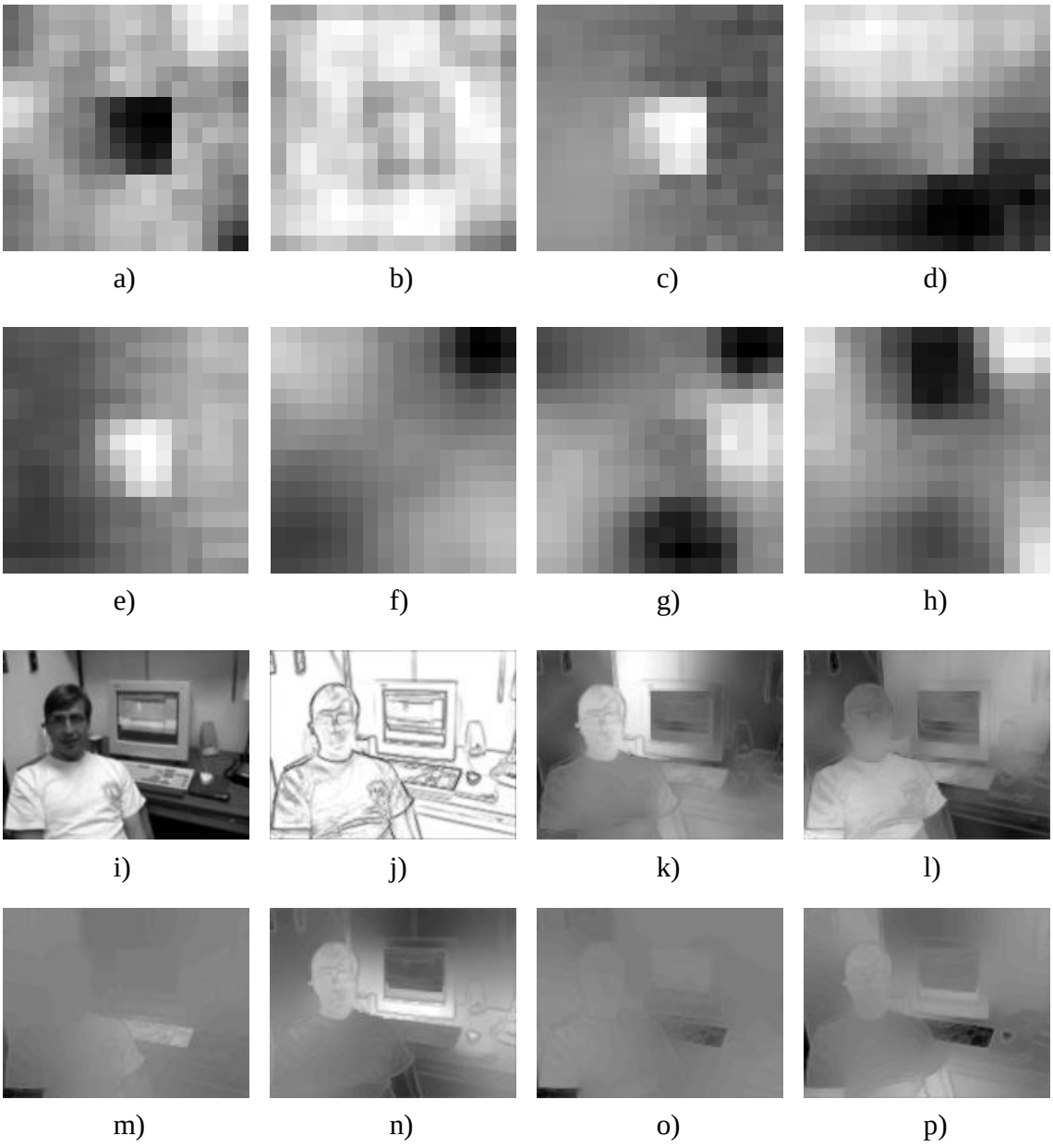

Figure 3.3: a) Original image (16×16 pixels), b) - h) Leading 7 eigenvectors sorted by the magnitude of the corresponding eigenvalue. The first eigenvector represents the stationary distribution, the succeeding eigenvectors capture the 6 most dominant perturbations to this distribution. i) Original image (160×120 pixels), j) - p) Leading 7 eigenvectors for the corresponding random walk.

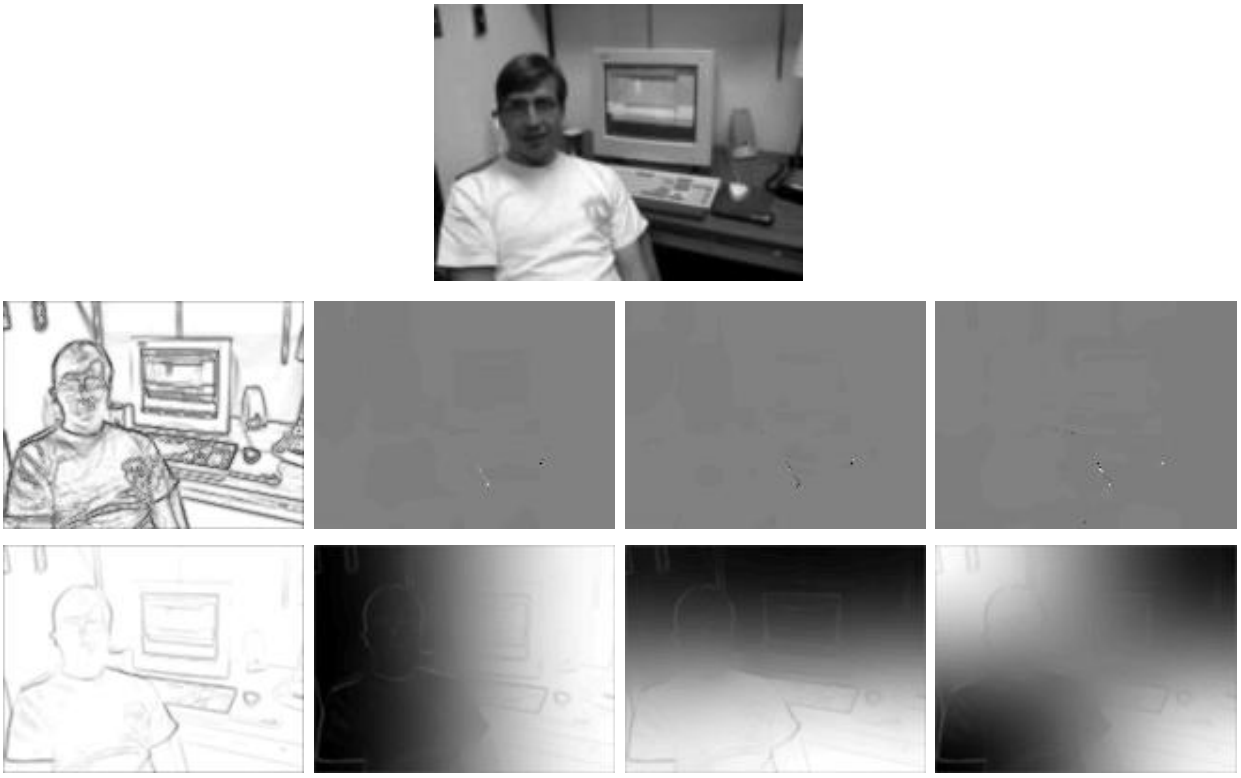

Figure 3.4: Top: Original image. Middle: Leading eigenvectors for $\sigma = 4$, with such a small σ the eigenvectors capture local perturbations created by a few pixels. Bottom: Leading eigenvectors for $\sigma = 25$. In this case σ is too large, and the eigenvectors are smoothed out. For such a large σ the eigenvectors resemble sinusoid functions.

3.3 Blur Kernels and Anisotropic Smoothing

Suppose now that we start the random walk from pixel $\vec{x}_i$, so that the initial distribution is given by $\vec{p}_{0,i} \equiv \vec{e}_i$, where $\vec{e}_i$ is the standard, nm-dimensional unit vector with 1 in the i^{th} row and zeros elsewhere. After t steps of the random walk we obtain the diffused distribution $\vec{p}_{t,i} = M^t \vec{e}_i$ which is just the i^{th} column of M^t. We call the vectors $\vec{p}_{t,i}$ averaging or blur kernels since, as will be discussed below, they can be used to anisotropically smooth the input image.

Figure 3.5 shows two of these blur kernels for different pixels $\vec{x}_i$, note that both kernels spread significantly across regions with similar brightness variations, however, the spread is much less noticeable across weak edges (e.g. see the bottom-left part of the dark square in Figure 3.5c), and almost non-existent across strong image edges (see the top-right side of the dark square in Figures 3.5b and 3.5c). From our original, rasterized image $\vec{I}$ we can form an anisotropically smoothed image $\vec{B}$ whose component B_j is a weighted average of the pixels in $\vec{I}$. The weights are given by the blur kernel initialized at pixel $\vec{x}_j$, that is, $B_j = \vec{I}^T \vec{p}_{t,j}$, where $\vec{p}_{t,j}$ is given by (3.6), and $\vec{p}_0 = \vec{e}_j$. Figure 3.5d shows the result of this averaging process. Notice that strong image discontinuities are preserved due to the anisotropic nature of the blur kernels, but weaker local brightness variations are smoothed out. This is similar to standard anisotropic smoothing [85], except that there the affinities are iteratively updated as the image is smoothed. Here, instead, the affinities are held fixed, according to the values dictated by the original image. It is also related to bilateral filtering [102], in that the blur kernels depend not only pixel similarity, but also on the distance between pixels. However, while [102] proposes a fixed size kernel, here the effect of distance is controlled by t. Larger values of t lead to blur kernels that typically cover larger regions of the image.

Due to the properties of the random walk, neighboring pixels in homogeneous regions of the image will have similar blur kernels for sufficiently large t, that is, the probability mass for their blur kernels is distributed over a similar subset of pixels. Conversely, neighboring pixels that belong to different regions (according to the chosen affinity measure) will have their probability

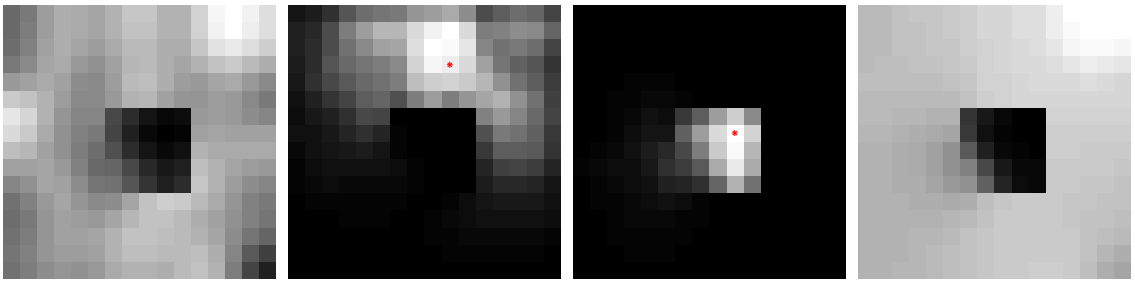

Figure 3.5: a) Original image (16×16 pixels), b) and c) Blur kernels for two different pixels (the starting pixels are marked in red), d) Anisotropically smoothed image. For these images $t = 40$.

mass distributed over different subsets of pixels, this is shown in Figure 3.6. The result is that given two blur kernels $\vec{p}_{t,i}$ and $\vec{p}_{t,j}$, their inner product $\vec{p}_{t,i}^T \vec{p}_{t,j}$ will be small if the pixels belong to different regions, and large if the pixels are near each other in a homogeneous region. In other words, the random walk induces a soft clustering of blur kernels that corresponds to the grouping of pixels into locally homogeneous regions. This property enables us to use blur kernels to determine candidate seed regions to use for segmentation. It is important to note here that the similarity between any two blur kernels increases with t, and that for very large t all blur kernels approximate the stationary distribution. For this reason, the choice of t becomes particularly important for preserving the properties of blur kernels that we have discussed up to this point. We will propose a suitable choice for t further on in this section.

The blur kernels themselves exist in an nm-dimensional space, and even for small images, finding clusters in such a high-dimensional space is no small task (in fact, for larger images, it becomes computationally infeasible to compute/store the complete set of blur kernels). Instead, we will use the spectral properties of the Markov matrix to our advantage. As mentioned above, the leading eigenvectors of the Markov matrix correspond to the dominant perturbations to the underlying random walk. In fact, most of the eigenvalues λ_i have magnitudes that are much smaller than one, and therefore, for sufficiently large t, the product $\Delta^t \vec{c}_0$ in (3.6) can be approximated using a projection comprised of the first few eigenvectors and eigenvalues of M.

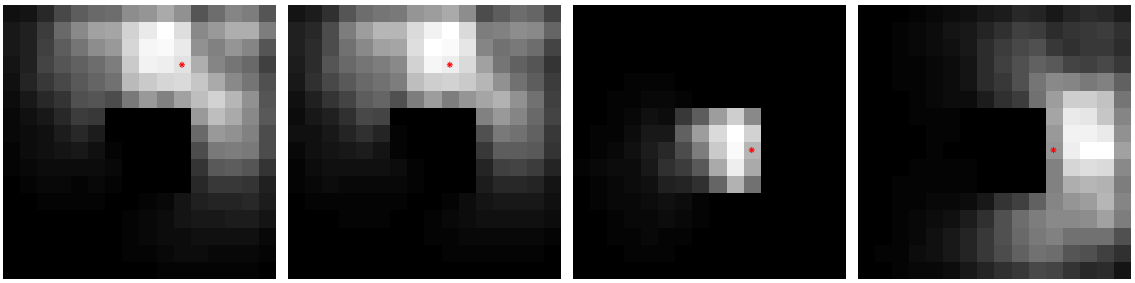

Figure 3.6: Blur kernel similarity for the image from Fig. 3.5. a) and b) Blur kernels for neighboring pixels (marked in red) in the same image region. c) and d) Blur kernels for neighboring pixels in different regions.

In particular, we define R_d to be a rectangular $d \times mn$ matrix with the $d \times d$ identity matrix in the left-most block. By neglecting the powers λ_i^t for $i > d$ and using (3.6), we find

$$\vec{p}_{t,i} \simeq \vec{q}_{t,i} \equiv (D^{1/2}U_d)\vec{w}_{t,i}, \quad \text{with } \vec{w}_{t,i} = \Delta_d^t U_d^T D^{-1/2}\vec{e}_i, \tag{3.8}$$

where $U_d = UR_d^T$ contains the first d columns of U, and Δ_d is the $d \times d$ diagonal matrix formed with the first d eigenvalues of the Markov matrix. The projected blur kernels $\vec{q}_{t,i}$ approximate the original blur kernels, and their clustering also corresponds to the clustering of image pixels into locally homogeneous regions. Furthermore, we have encoded the blur kernels using $\vec{w}_{t,i} \in \Re^d$ which is a significant improvement over $\vec{p}_{t,i} \in \Re^{nm}$.

We still have to choose a suitable value for d and t. To choose these values, we note that the error in the approximation of the blur kernels depends on the magnitudes of the terms $|\lambda_i|^t$ for $i > d$, the largest of which is $|\lambda_{d+1}|^t$. In practice, we find a good approximation when d and t are chosen such that $|\lambda_{d+1}|^t < 1/3$. Smaller values of d and t lead to artifacts in the projected blur kernels that resemble the ringing that occurs due to severe truncation of a Fourier Series. The use of $\vec{q}_{t,i}$ instead of $\vec{p}_{t,i}$ provides us with an efficient way for computing the anisotropically smoothed image $\vec{B}$. However, for segmentation purposes, we are more interested in the coefficient vectors $\vec{w}_{t,i}$.

3.4 Spectral Embedding

Given t and d, we can map each pixel $\vec{x}_i$ to the $d-$dimensional vector $\vec{w}_{t,i}$ which provides the coefficients for the projected blur kernel $\vec{q}_{t,i}$ centered at that pixel. For $t = 0$, this embedding is closely related to the standard Laplacian embedding [3]. Laplacian embedding is based on the eigenvectors and eigenvalues of the the generalized eigenvalue system

$$L\vec{x} = \lambda D\vec{x}, \tag{3.9}$$

where L is the Laplacian matrix $L = D - S$, S is a weight matrix that encodes the connections (and usually also the similarities) between nodes in a graph, and D is a diagonal matrix whose entries contain the row or column sums of S. Meila and Shi [71] show that given a standard eigenvalue problem

$$P\vec{x} = \lambda\vec{x}, \tag{3.10}$$

where P is a Markov matrix $P = SD^{-1}$, if λ and $\vec{x}$ are solutions to (3.10), then $1 - \lambda$ and $\vec{x}$ are solutions to (3.9). In other words, the two systems have the same eigenvectors, and their eigenvalues are related.

Our formulation is identical to that of Meila and Shi if we make $S = A$. Thus, for $t = 0$ the eigenvectors of our Markov matrix are the same as the eigenvectors of the Laplacian embedding, and the corresponding eigenvalues are related by $1 - \lambda$. Laplacian embedding is particularly useful for reducing the dimensionality of data that is expected to lie on a low-dimensional manifold contained within a high-dimensional space, it yields a low-dimensional representation of the data that best preserves the structure of the original manifold in the sense that points that are close to each other on the original manifold will also be close after embedding. At the same time, the embedding emphasizes clusters in the original data.

Here we extend Laplacian embedding for positive values of t, and make an explicit connection between the embedding and the projected blur kernels. Two additional properties of the embedded vectors $\vec{w}_{t,i}$ are of interest. First, it follows from (3.7) and (3.8) that the first component of $\vec{w}_{t,i}$ equals $1/\alpha \equiv 1/\sqrt{\sum_{k=1}^{nm} D_k}$, a constant. Secondly, as we have mentioned

before, the inner product of two blur kernels whose probability mass lies on different subsets of pixels will be small. This inner product can be approximated using the projected blur kernels as

$$\vec{p}_{t,i}^{\,T}\vec{p}_{t,j} \simeq \vec{q}_{t,i}^{\,T}\vec{q}_{t,j} = \vec{w}_{t,i}^{\,T}Q_d\vec{w}_{t,j}, \tag{3.11}$$

where $Q_d = U_d^T D U_d$. We can simplify this $\vec{w}$-space geometry by using the coordinate transform

$$\vec{z} = Q_d^{1/2}\vec{w}. \tag{3.12}$$

It follows that $\vec{q}_{t,i}^{\,T}\vec{q}_{t,j} = \vec{z}_{t,i}^{\,T}\vec{z}_{t,j}$. This indicates that the clustering of the blur kernels $\vec{p}_{t,i}$ in the nm-dimensional space is captured by the clustering of the embedded vectors $\vec{z}_{t,i}$ in the d-dimensional space.

Figure 3.7 shows the second and fourth components of $\vec{z}_{t,i}$ (the first component is nearly constant, and is omitted). There are several important observations to make here. First, small, isolated groups of pixels appear as tight clusters in the embedding indicating that the blur kernels for all the pixels in the group are very similar. Second, for larger homogeneous image regions, the corresponding points in the embedding cluster around 1D or 2D sets, in other words, the points in the embedding that correspond to different local groups of pixels within the same homogeneous region are joined by a smooth, densely populated path in the embedding. Conversely, image brightness discontinuities result in sparsely populated valleys separating two dense clusters.

If two groups of pixels are separated by a strong image boundary, their blur kernels will be almost orthogonal, and the width of the corresponding valley in $\vec{z}$-space will be nearly 90 degrees. Blur kernels for groups of pixels separated by weaker boundaries overlap more, and their relative angles are less than 90 degrees. Eventually, as the separating boundary becomes weaker, the corresponding blur kernels may blend together in the embedding. Finally, we note that the embedding $\vec{z}_{d,t,i}$ computed for a particular dimension d, scale t, and pixel $\vec{x}_i$ is related to the embedding $\vec{z}_{d',t',i}$ for some other pair (d', t') with $d' \leq d$ by a linear transformation

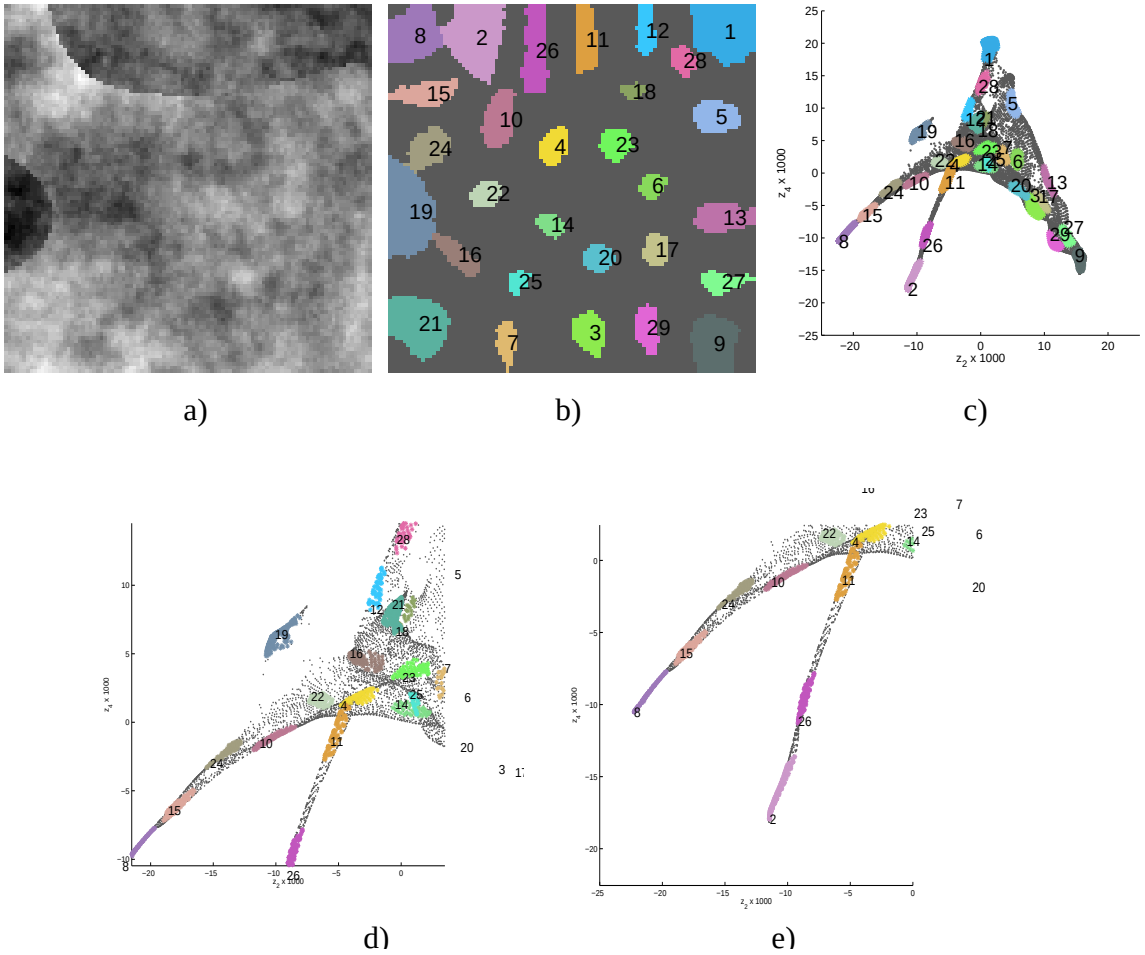

Figure 3.7: a) Fractal image (100×100 pixels) plus noise. b) Small groups of similar pixels (we shall see how these were generated in Section 5). c) Corresponding points in the second and fourth components of the $\vec{z}_{t,i}$ embedded vectors (chosen for clarity of illustration), notice that the corresponding points cluster tightly. d) Detail, notice that group 19, which is in a different image region than groups 16, 22, and 24 is isolated from them in the embedding. e) Detail, notice that neighboring groups 8 and 2 are separated by an image boundary, they are also isolated from each other in the embedding. Conversely, group 8 is connected in the embedding to group 15 which lies inside the same image region, similarly, groups 26 and 2 which are also in the same image region are connected in the embedding. For this figure, $d = 15$, and the constraint that $|\lambda_{d+1}|^t \simeq 1/3$ yields $t = 430$

$\vec{z}_{d',t',i} = Z\vec{z}_{d,t,i}$ where Z is a $d' \times d$ matrix that does not depend on i

$$Z = Q_{d'}^{1/2} \Delta_{d'}^{t-t'} W_{d \to d'} Q_d^{-1/2} \tag{3.13}$$

where $W_{d \to d'}$ is a $d' \times d$ matrix whose leftmost $d' \times d'$ block is the identity matrix, and whose last $d - d'$ columns are zero vectors. This implies that the $\vec{z}$-space is only distorted in an affine manner when the scale t and dimension d are changed. Thus, although the relative angles between various points may change with t and d, the global arrangement of valleys, ridges, and clusters in the general distribution can only change in a linear manner. Notice that Z is in general not invertible.

Summing up the above, under spectral embedding the projected vectors of similar, spatially proximal pixels project to dense clusters in $\vec{z}$-space, while the angle between $\vec{z}$ vectors for pixels separated by an image boundary increases toward 90 degrees as the strength of the boundary grows. These two conditions make spectral embedding ideal for selecting groups of pixels that can be used as seed regions for S-T min-cut. It is worth noting that this is just one possible application of the embedding technique described above, the framework itself is general and can be used for other data clustering applications. In the next chapter, we will present a segmentation algorithm that generates seed regions using spectral embedding, and uses these regions along with S-T min-cut to partition the image into homogeneous looking regions.