Multilevel space-time aggregation for cell segmentation and tracking

by

Tiffany C. Inglis

A report
presented to the University of Waterloo
in fulfillment of the
report requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Supervisor: Hans De Sterck

Waterloo, Ontario, Canada, 2009

© Tiffany Inglis 2009

I hereby declare that I am the sole author of this report. This is a true copy of the report, including any required final revisions, as accepted by my examiners.

I understand that my report may be made electronically available to the public.

Abstract

We introduce a multilevel algorithm for the purpose of segmenting brightfield cell images. The V-cycle algorithm has two stages. The first stage takes all pixels in an image and repeatedly groups them into larger overlapping blocks until all blocks become salient. The second stage then takes these segments and determine exactly which pixels belong to each segment. The algorithm is implemented recursively and makes use of algebraic multigrid (AMG) coarsening. At first, using only intensity as the segmentation criterion, the algorithm performs poorly on textured examples. So we include a multilevel isotropic texture measure as a separate criterion, and the result improves greatly. Then, the algorithm is extended for 3D problems and consequently applied to a series of cell images taken over small time intervals. The result is a tube-shaped segment that tracks the cell's location in space-time.

Acknowledgements

I am deeply thankful to my supervisor, Hans De Sterck, for guiding me through this report. When teaching new concepts and offering insights to a problem, he was always very patient and tried to explain in the most intuitive way possible.

I would also like to thank Killian Miller for helping me optimize the algorithm implemented in Matlab, by providing the C code for the Algebraic Multigrid (AMG) coarsening.

Geoff Sanders was very helpful in clarifying some ideas from AMG including how to compute certain coarse-level block attributes and why it works.

Lastly, I am grateful to Adley Au for sending me many useful online resources on image segmentation and offering a different perspective on the same problem.

Dedication

To my husband and my parents for their unending support in whatever I do.

Contents

Chapter 1

Introduction

The goal in image segmentation is to partition an image into meaningful segments so that it is easier to analyze. Figure 1.1 is shown here with 2 segments separating the subject (the mallard duck) from the background (the water).

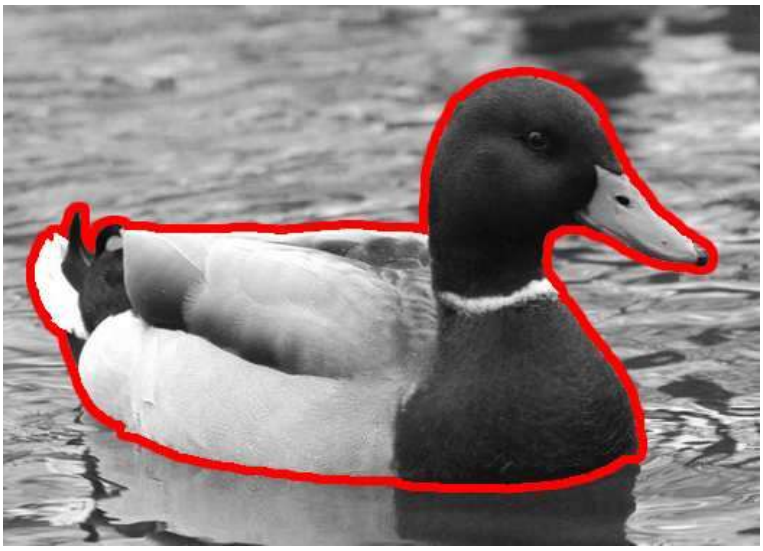


Figure 1.1: An image segmented into subject and background

In general, a segmentation algorithm takes the input image and groups the pixels according to features such as intensity, colour, texture, shape, etc. Certain algorithms may be ran multiple times to improve the final result.

1.1 Current methods

Because there are various mathematical interpretations to what a segment really is, many different approaches have been developed for segmenting an image. With point-based methods, pixels are treated as points in a data set and they are grouped according to some statistics. The K-means algorithm [4] is one such method where clusters of pixels are formed so as to minimize the variance within each cluster.

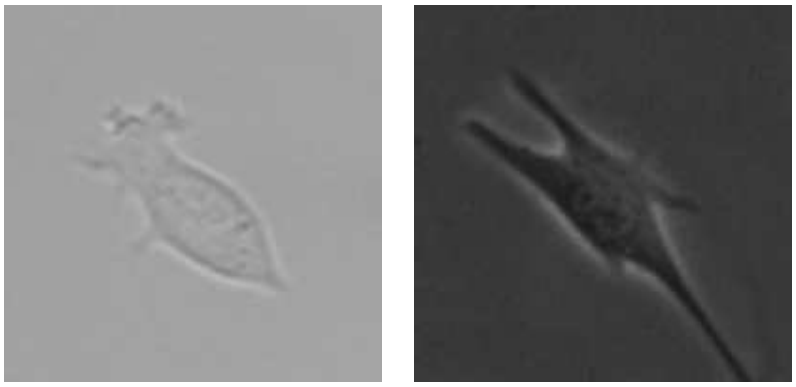
Instead of grouping pixels, we can also try to locate the edges within an image. Edge detection methods [1] work the best with images containing distinct boundaries between segments. Otherwise, there are remedies for connecting broken boundaries but they require more work, such as analyzing the boundary continuity between segments.

Graph-partitioning methods involve constructing weighted undirected graphs out of images (with pixels as nodes and pixel connections as edges) and applying graph-theoretic methods to find good segments. One such method is the normalized cut algorithm [8], which partitions an image into segments by removing edges in the graph. The goal is to find the optimal partition that maximizes the connection *within* each segment while minimizing the connections *between* segments.

The method we want to consider is a multilevel segmentation that progressively groups smaller blocks into larger blocks. It does so by looking at the similarity between each pair of blocks at different scales. By taking into account both local and global features of the image, the segmentation can be performed more accurately. This method also has the added advantage of being more intuitive. Instead of working with a constraint optimization problem, which is computationally difficult, we simply calculate some summary statistics for each block, then merge ones that are closedly related.

1.2 Application to cell tracking

The type of images we want to segment are cell images taken via brightfield microscopes. These images typically have low contrast between the cells and the background (see Figure 1.2(a)) . To aid the observer, halos are added around cells to makes the boundaries clearer (see Figure 1.2(b)).



(a) Without halo (b) With halo

Figure 1.2: Brightfield cell images

Having a good segmentation algorithm is particularly important when there are too many images to be segmented manually. So our first goal is to segment an image into different cells plus the background. After that is accomplished, we would like to apply the algorithm to a series of cell images taken sequentially over small time intervals. Ideally, the stack of images will then be segmented into several tubes that trace the movement of multiple cells.

Certain cell behaviour might make this problem more challenging. For example, cells can merge, split, change shape, or pass over one another from above or below. This can cause great difficulty when attempting to distinguish between different cells.

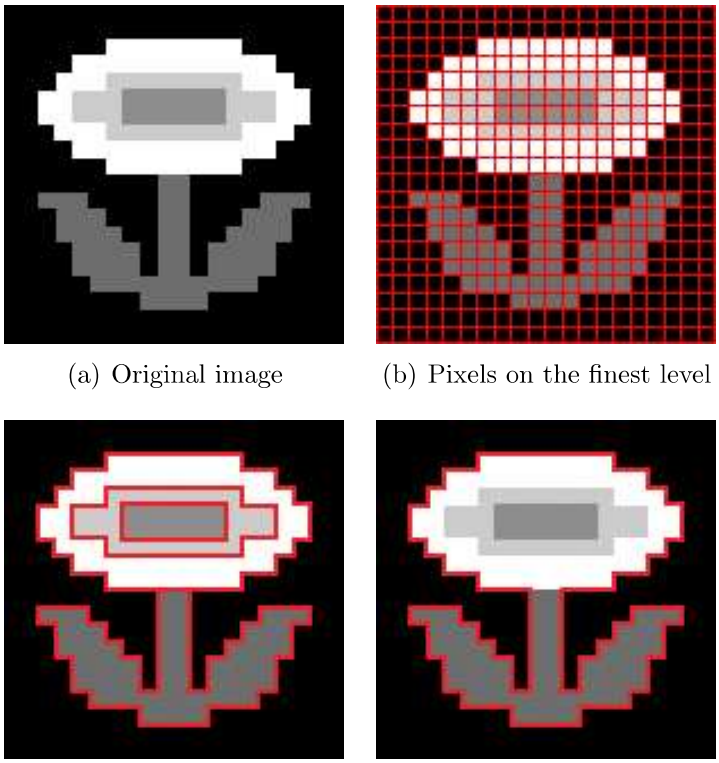
Chapter 2

Description and motivation

Our multilevel image segmentation algorithm is mainly based on an article by Sharon et al [5]. The calculation of the saliency measure is similar to the form given in a different article [7] except we normalize both the boundary and the internal connections. To account for textural elements, the paper by Galun et al [3] was consulted. We will now give a brief outline of the algorithm.

2.1 Overview

Our algorithm is multilevel, which means segmentation occurs on different levels. For an image such as Figure 2.1(a), the finest level contains individual pixels (see Figure 2.1(b)), the coarsest level contains the final segments detected (see Figure 2.1(d)), and all the levels in between contains blocks accumulated along the way (see Figure 2.1(c)).



(a) Original image (b) Pixels on the finest level

(c) Blocks (intermediate level) (d) Segments (coarsest level)

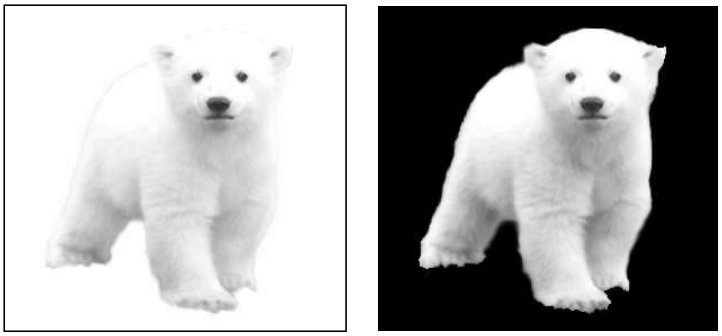
Figure 2.1: The difference between pixels, blocks and segments

On the finest level, we want to group the pixels by similarity. Start by forming small blocks with pixels that have similar intensity and are close together. Then take these small blocks to form medium blocks, and finally large blocks. Note that we allow the formation of slightly overlapping blocks since, at this stage in the algorithm, we are simply locating roughly where the blocks are and not worrying about their exact boundaries. So a pixel may partially belong to several blocks at once. Note also that, in general, there will be more than three levels of blocks.

Now if we continue blindly forming the blocks, the end result would be one giant block containing the entire image, which is useless. We need a stopping criterion so that when a block is an actual segment in the image, it stops merging with other blocks. This criterion is called saliency, and for each block, its saliency tells us how much it has in common with its neighbours.

Figure 2.2(a) shows an example of the polar bear being a high-saliency segment with respect to the background. It has high saliency because the subject and the background are similar in intensity. In contrast, Figure 2.2(b) shows the bear as a low-saliency segment

since it is starkly difference from the black background. The segment with low saliency is more likely to be detected as a segment, which fits with our intuition that a polar bear on a black background is much easier to spot.



(a) High-saliency segment (b) Low-saliency segment

Figure 2.2: The polar bear as both high and low saliency segments

An important segment in an image should have low saliency because it should be sufficiently different from its surrounding. Therefore, at each stage of the block formation, we check if any blocks are salient (*i.e.* have low saliency). All blocks found salient are to remain inert while non-salient blocks merge to form larger blocks. This multilevel block formation continues until there are no more blocks available for merging. At this point, we have found all the segments within the image.

The block formation is only the first stage of the algorithm. Currently, we only know, between two adjacent level, which smaller blocks belong to which larger blocks. To determine exactly which pixels belong to each of the segments, we do the following (keep in mind there are usually more levels than presented here):

1. Determine which large blocks belong to each segment.
2. Determine which medium blocks belong to each large block.
3. Determine which small blocks belong to each medium block.
4. Determine which pixels belong to each small block.
5. Combine the above information (which we learned through block formation in the first part of the algorithm) to determine which pixels belong to each segment.
6. In the case of overlapping segments, that is, a pixel belonging to more than one segment, we examine the degrees to which the pixel belongs to each of the segments, and assign it to the segment it most likely belongs to.

This completes the second stage of the algorithm.

To summarize, the algorithm contains two stages. The first stage involves forming overlapping blocks from the finest level down to the coarsest level, eventually ending with a few overlapping segments. The second stage determines exactly which pixels belong to each segment by going back up to the finest level. This type of algorithm is called a *V-cycle* as it involves going down to the coarsest level (bottom of the "V") then back up to the finest level (top of the "V").

Figure 2.3 shows an example of what the algorithm does for this 5×5 image. The left branch of the "V" shows the first stage, in which overlapping blocks are formed. At each level, a block is represented by a red dot. As the level becomes coarser, we have fewer but larger blocks. Eventually, two salient segments are found. We do not coarsen any further because there are no more non-salient blocks to merge. Next is the second stage of the algorithm, which makes up the right branch of the "V". Here we refine the edges of the segments by tracing back to the finest level which pixels belong each segment. At each level, we connect nodes that fell into the same block one level coarser. The end result is 2 disconnected graphs containing all pixels from the finest level. These two graphs represent the salient segments found, with every pixel belonging to exactly 1 segment. The actual implemention of this algorithm will be done recursively, as we will describe in the next chapter.

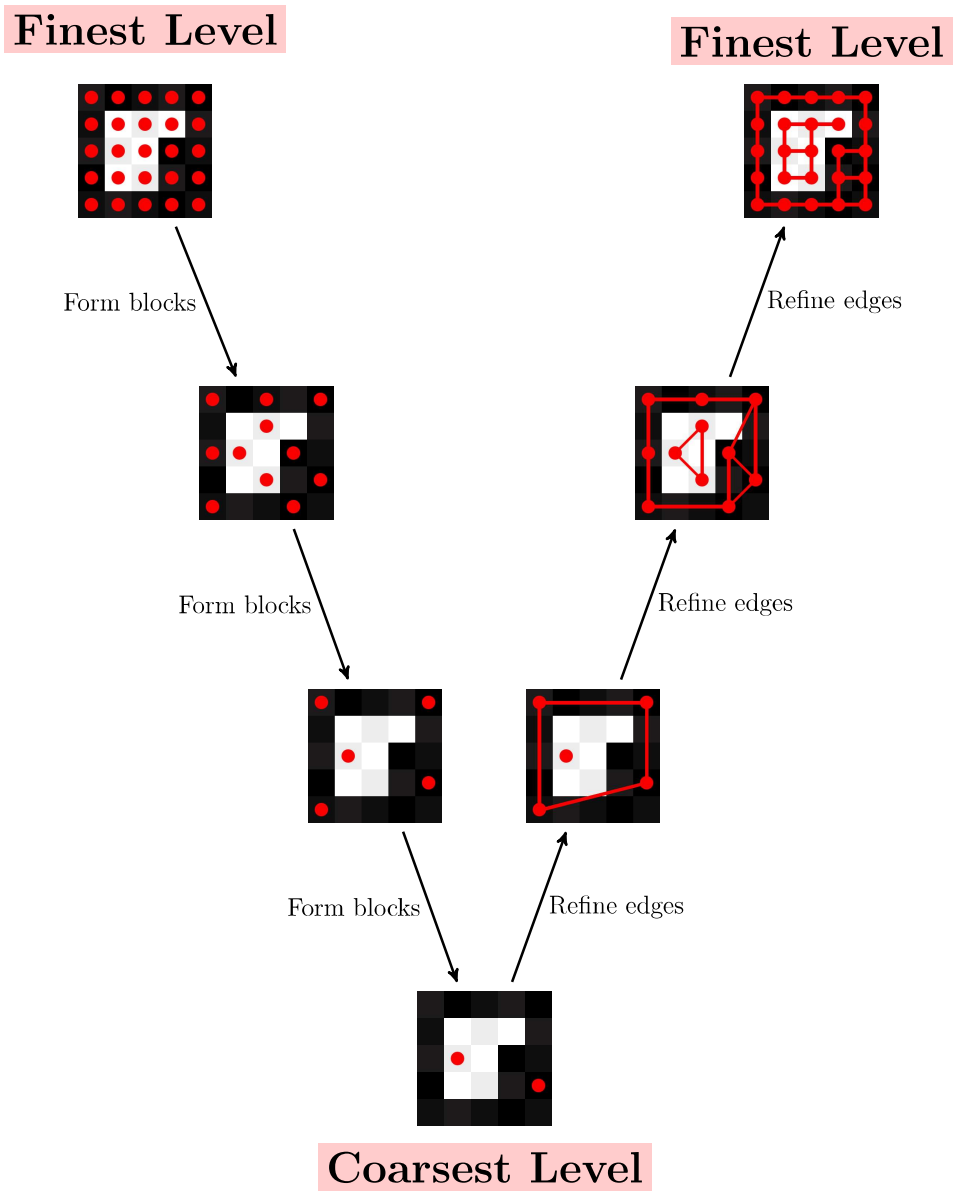
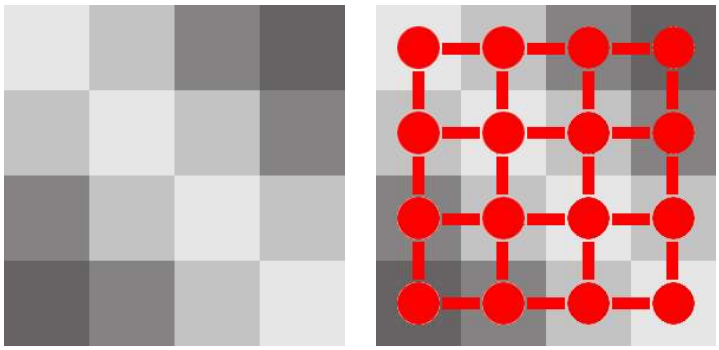


Figure 2.3: Example of a V-cycle segmentation

2.2 Variables on the finest level

We begin the algorithm with an $n \times n$ input image containing $N = n^2$ pixels. For simplicity, assume the image is grayscale. Then construct an undirected graph with nodes being the pixels and edges being connections or similarities between neighbouring pixels.

Define two pixels to be neighbours if and only if they are next to each other either horizontally or vertically in the image. For the graph, we will only consider connections between neighbouring pixels, so an edge exists between node i and j if and only if pixel i and j are neighbours. Figure 2.4 shows a sample image and the underlying graph.



(a) Original image (b) Corresponding graph

Figure 2.4: An image and its corresponding graph

Each pixel, or node, has an intensity value between 0 (black) and 1 (white). Since the goal is to group similar nodes together, we want to assign high edge weights to neighbours with similar intensities and low edge weights to neighbours with different intensities. We call these edge weights *coupling weights* because they describe the coupling strengths between nodes.

Let I_i be the intensity of pixel i. Then an obvious choice for the coupling weight A_{ij} between pixel i and j would be $|I_i - I_j|$. However, this linear function is not so easy to work with. Ideally, we want closely related neighbours to merge much more easily than weakly related neighbours. A nonlinear scaling such as

$$A_{ij} = e^{-\alpha |I_i - I_j|},$$

where α is predetermined scaling constant, allows the algorithm to perform much more efficiently and accurately [7].

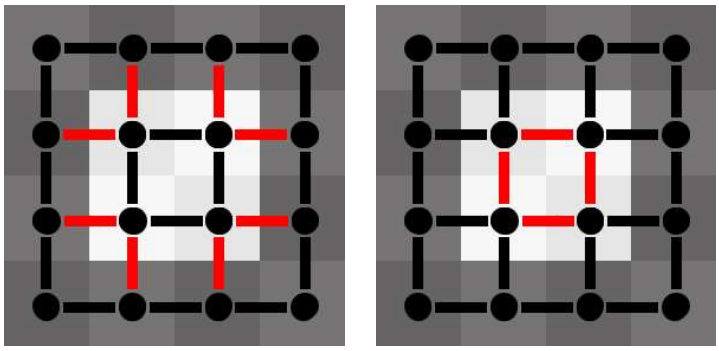
So far we have only discussed the graph on the finest level, constructed from individual pixels. As we proceed with the segmentation, these pixels will be grouped together in

overlapping blocks, forming a coarse-level graph with much fewer nodes. Now that each node represent a block, we want measure its saliency so that we know when a salient segment has been found. Let Γ be the saliency measure such that a low value implies a block is salient. To define Γ, we need to consider what properties make a block salient.

A salient block should be weakly connected to others and strongly connected to itself. So Γ would roughly take the form

$$\Gamma = \frac{\text{connection to neighbours}}{\text{connection to self}}.$$

Recall that coupling weights measure connections between pairs of nodes. So we can calculate a block's connection to its neighbours by adding up the coupling weights along the block's boundary. Similarly, a block's connection to itself can be calculated as the total coupling weights for edges within the block. Figure 2.5 shows an example of a light block on a dark background. In Figure 2.5(a), the coupling weights on the red edges contribute to the numerator (*i.e.* connection to neighbours) while the ones in Figure 2.5(b) contribute to the denominator (*i.e.* connection to self).



(a) Connection to neighbours (b) Connection to self

Figure 2.5: Calculating saliency using coupling weights

The proposed saliency measure has two problems, both concerning scalability. First, the numerator increases with the boundary length of a block. Second, the denominator scales according to the area of the block. These properties are undesirable for Γ because the saliency measure is not supposed to depend on the size or shape of the block.

We can fix the scalability problems by normalizing both the numerator and the denominator. Let L be the total coupling weights across the boundary of the block and W be the total coupling weights within the block. Earlier, we proposed that $\Gamma = L/W$. Now to normalize this measure, introduce two more quantities. Let G be the length of the

boundary and V be the area of the block. Then a good saliency measure without scaling problems would be

$$\Gamma = \frac{L/G}{W/V}.$$

Sometimes we need to manipulate the values of Γ in order to guide the algorithm into segmenting the image correctly. For example, setting $\Gamma = \infty$ ensures that a block will not be declared salient. This is useful when we want to prevent, say, smaller blocks from becoming salient segments. On the other hand, setting $\Gamma = 0$ is useful when a block has already been declared salient and we want it to remain salient for all coarser levels (and therefore be excluded from any merging activities). We will show in the next chapter how to include these ideas into the actual implementation of the algorithm.

2.3 L, G, W and V as matrices

In the last section, we gave an overly simplified description of what L, G, W and V are, for the purpose of explaining how they relate to the saliency, Γ. Now we will discuss them in greater detail.

The $N \times N$ coupling matrix A is the basis from which L, G, W and V are calculated. On the finest level, let A_1, A_2, A_3 and A_4 denote coupling weights between a node and each of its 4 neighbours, then L, G, W and V are also $N \times N$ matrices with the following 5-point stencils:

$$L = \begin{bmatrix} & -A_1 & \\ -A_3 & \sum A_i & -A_4 \\ & -A_2 & \end{bmatrix}, \quad G = \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix},$$

$$W = \begin{bmatrix} & A_1 & \\ A_3 & & A_4 \\ & A_2 & \end{bmatrix}, \qquad V = \begin{bmatrix} & 1 & \\ 1 & & 1 \\ & 1 & \end{bmatrix}.$$

Now let the superscript c indicate that a variable is defined on a coarser level. So we want to determine L^c, G^c, W^c and V^c, the coarse-level counterparts of L, G, W and V. Once that is done, the saliency of block i on the coarse level may be calculated using the formula

$$\Gamma_i^c = \frac{L_{ii}^c / G_{ii}^c}{W_{ii}^c / V_{ii}^c},$$

which follows the heuristic we discussed in the previous section.

Note that we are only interested in saliency values on levels coarser than the finest level, because coarser-level nodes represent blocks rather than pixels. However, it is necessary to calculate L, G, W and V on the finest level because they are needed to determine L^c, G^c, W^c, V^c and hence Γ^c on coarser levels. We will discuss how this is done in Section 2.5.

On the finest level, we set Γ to an $N \times 1$ vector of ∞'s so that no nodes can be found salient. The reason is that, in all practical cases, individual pixels do not represent salient segments so we do not want any nodes on the finest level to be declared salient.

2.4 Coarsening

The coarsening step occurs between two adjacent levels, which we will refer to as the fine level and the coarse level. Basically, we take all the nodes on the fine level and picks out important ones (called *coarse-level points*, or *C-points*) to construct the coarse-level graph. Think of this as forming overlapping blocks with the fine-level nodes, where each C-point acts as a seed node for each block.

Coarsening is treated as a function that takes some inputs–the set of fine-level nodes, some threshold θ as the strength-of-connection parameter, and another threshold γ used to detect salient segments–and returns an output C containing the C-points chosen. For example, given fine-level nodes $\{1, 2, 3, 4, 5\}$, $\theta = 0.1$ and $\gamma = 1$, the function may return $C = \{1, 3, 5\}$.

The C-points should be a good representation of all the fine-level nodes because we want to lose as little information as possible when going down to a coarser level. In particular, they should include all salient segments found up to this level because these segments contain important information about the image. So basically, the C-points will be chosen so that they are either salient or, failing that, have strong influence on many of their neighbours (to be explained below).

The coarsening algorithm consists of two steps. The first step is to set all salient nodes as C-points. To prevent them from forming larger blocks and to ensure that they remain salient on all coarser levels, we set their saliency values to 0. The second step is to apply algebraic multigrid (AMG) coarsening [2] to select nodes that strongly influence many of their neighbours. The AMG coarsening algorithm we use has been modified to fit the image segmentation problem.

From the coupling matrix A, we select entries representing strong influences. Consider row i, for instance. The j-th entry of row i represents the influence of node j on node i. The sum of all entries in row i, excluding the diagonal term A_{ii}, is then equal to the total influence of other nodes on node i. So we define node j to be a strong influence if and only

if the j-th entry exceeds some fraction of the total influence. In other words, A_{ij} is strong if

$$A_{ij} \geq \theta \cdot \sum_{k \neq i} A_{ik},$$

where $\theta \in (0, 1)$ is the coarsening threshold, usually chosen in the range $(0.1, 0.2)$. Smaller values of θ result in too few segments while larger values not only give us too many segments but also dramatically increases the computation time.

Next we count, for each node, the number of strong influences it has on other nodes. Store these counts in a vector λ. Then perform an iterative process where we systematically assign the nodes to be either C-points or F-points (*i.e.* nodes that are *not* chosen as C-points) until all nodes have been assigned. At each iteration, pick the most influential point (*i.e.* one with the highest λ value) to be a C-point. Scan all neighbours it strongly influences, and label any unassigned ones as F-points. Note that after a node is assigned to be either a C- or an F-point, its λ value is dropped to 0 so that we do not revisit it. Lastly, for all the newly assigned F-points, we scan their neighbours, and for any unassigned nodes that strongly influence these F-points, we increase their λ value by 1. This step increases the importance of all nodes strong influencing nodes that are already assigned, and increases their chance of being selected as the next C-point. Repeat this process until all λ values become zeroes, which occurs when all nodes have been assigned. The result of this coarsening is a set of C-points containing normally less than half the nodes from the original set.

2.5 Coarse-level variables

The C-points we found represent overlapping blocks, some of which may be salient segments. If we then apply the coarsening step on the set of C-points, and repeat this for several levels, eventually all the overlapping blocks will be salient segment, and we will have completed the first stage of the V-cycle algorithm. But since the algorithm is recursive, in order to perform further coarsening, we need to redefine some input variables on the coarse level for each block (or C-point). These variables are

- coarse intensity vector I^c
- coarse coupling matrix A^c
- coarse saliency vector Γ^c and its component matrices L^c, G^c, W^c, V^c.

To compute these, we need to know how the nodes and blocks relate. Let us define the interpolation matrix P as a N by K matrix such that P_{ij} indicates how much node

i belongs to the j-th block. This definition implies that P should satisfy the following properties:

1. Each P_{ij} ranges from 0 to 1.
2. Each row sum $\sum_k P_{ik}$ is equal to 1 (because P_{ij} is the fraction of node i that belongs to j and together these fractions should add up to 1).
3. If the i-th node is the j-th C-point, then $P_{ij} = 1$ while $P_{ik} = 0$ for all $k \neq j$.

To determine how much a node belongs to a block, one only needs to examine and normalize the coupling weights. So P can be easily constructed from the coupling matrix A, using the formula

$$P_{ij} = \begin{cases} 1 & \text{if node } i \text{ is a C-point and } i = C_j \\ 0 & \text{if node } i \text{ is a C-point but } i \neq C_j \\ \dfrac{A_{iC_j}}{\sum\limits_{k \in C} A_{ik}} & \text{if node } i \text{ is an F-point.} \end{cases}$$

Now that we have the interpolation matrix P, the necessary variables can be recomputed for the coarse level. Let $\tilde{P}$ be P column-normalized, so that

$$\tilde{P}_{ij} = \frac{P_{ij}}{\sum\limits_k P_{kj}}.$$

Then the coarse-level attributes are calculated as

- Coarse intensity vector:
$$I^c = \tilde{P}^T I$$
- Coarse coupling matrix:
$$A^c = P^T A P$$
- Coarse boundary coupling matrix:
$$L^c = P^T L P$$
- Coarse boundary length matrix:
$$G^c = P^T G P$$
- Coarse internal coupling matrix:
$$W^c = P^T W P$$

- Coarse internal area matrix:

$$V^c = P^T V P$$

- Coarse saliency vector:

$$\Gamma_i^c = \frac{L_{ii}^c / G_{ii}^c}{W_{ii}^c / V_{ii}^c}$$

The coarse intensity I^c of a block is calculated as a weighted sum of the intensities of the nodes it contains. The coarse saliency Γ^c is calculated using diagonals of the matrices L^c, G^c, W^c and V^c, as discussed earlier. Now the way we calculate L^c, G^c, W^c, V^c involves multiplying each of L, G, W, V by P^T on the left and P on the right. Since the reason for this is more complicated, we will demonstrate with an example. Consider a 5×5 image whose i-th block is consists of the 9 white pixels, as shown in Figure 2.6:

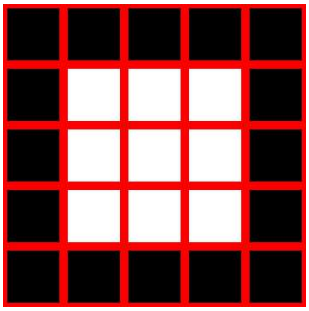


Figure 2.6: An image with a white block

Then the 2D representation of P_i, the i-th row of P, would be given by

$$P_i = \left[\begin{array}{c|ccc|c} 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \end{array}\right].$$

First, we want to show that the diagonal element $G_{ii}^c = P_i^T G P_i$ equals the boundary length of block i, which in this example is $3 \times 4 = 12$. In stencil notation,

$$G = \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix},$$

so applying P_i^T to G gives us

$$P_i^T G = \left[\begin{array}{c|ccc|c} 0 & -1 & -1 & -1 & 0 \\ \hline -1 & 2 & 1 & 2 & -1 \\ -1 & 1 & 0 & 1 & -1 \\ -1 & 2 & 1 & 2 & -1 \\ \hline 0 & -1 & -1 & -1 & 0 \end{array}\right]$$

Then applying this to P_i results in a scalar value of

$$P_i^T G P_i = 12,$$

as desired.

Next we want to show that $V_{ii}^c = P_i^T V P_i$ equals the number of internal connections within block i. In stencil notation,

$$V = \begin{bmatrix} & 1 & \\ 1 & & 1 \\ & 1 & \end{bmatrix},$$

First apply P_i^T to V to get

$$P_i^T V = \left[\begin{array}{c|ccc|c} 0 & 1 & 1 & 1 & 0 \\ \hline 1 & 2 & 3 & 2 & 1 \\ 1 & 3 & 4 & 3 & 1 \\ 1 & 2 & 3 & 2 & 1 \\ \hline 0 & 1 & 1 & 1 & 0 \end{array}\right]$$

Then apply this to P_i to get

$$P_i^T V P_i = 24.$$

Here, 24 is the total number of node pairs (i, j) within the block. Since this counts each edge twice, we must divide it by 2 to get 12, the number of internal connections. So V_{ii}^c is really *twice* the number of internal connections in block i. However, it is unnecessary to divide it by 2 because V^c is simply used to calculate the saliency, Γ^c. Having a factor of 2 does not affect any saliency comparison between blocks.

So we have shown that the method used to calculate G^c and V^c yields correct results. A similar analysis will show that the method works for calculating L^c and W^c as well, but we will not demonstrate here.

Notice that the example we gave involves non-overlapping blocks. This is an important property because in such cases, P contains only binary values (*i.e.* 0's and 1's), and the

formulae given for L^c, G^c, W^c, V^c all work correctly. However, if the blocks are overlapping, then P contains a range of values from 0 to 1, and we have found through experimentation that sometimes the formulae can produce nonsensical results. Specifically, it is possible to obtain positive off-diagonal values in G^c if we calculate it as

$$G^c = P^T G P.$$

This makes no sense considering G^c is the graph Laplacian for a set of nodes, and its off-diagonal elements are defined to be nonpositive. This problem is also observed in L^c for certain input images. To fix this, we need to slightly alter the way G^c and L^c are calculated. For G^c, we do the following:

1. Let $\hat{G}$ be G without its diagonal.
2. Set $G^c = P^T \hat{G} P$.
3. Replace the diagonal of G^c by ithe negative of ts off-diagonal row sums. That is,

$$G^c_{ii} \leftarrow -\sum_{k \neq i} G^c_{ik}.$$

The same fix is applied to L^c:

1. Let $\hat{L}$ be L without its diagonal.
2. Set $L^c = P^T \hat{L} P$.
3. Replace the diagonal of L^c by ithe negative of ts off-diagonal row sums. That is,

$$L^c_{ii} \leftarrow -\sum_{k \neq i} L^c_{ik}.$$

2.6 On the coarsest level

The segmentation algorithm will recursively coarsen the current set of nodes and recompute the variables on the coarser level. At each level, we get a new set of blocks and a saliency value associated with each.

Every node that has been found salient at some level remains salient throughout the algorithm, while every non-salient node continues to participate in the block formation at each level. This process continues until all nodes become salient (that is, the saliency vector is identically zero), at which point we terminate the first part of the algorithm.

2.7 Gathering nodes for each segment

Currently we have a small set of nodes on the coarsest level that represent significant segments in the image. Between every two adjacent levels, each coarse-level node is associated with some subset of the fine-level nodes. These fine-level nodes may belong to more than one coarse-level node. What we really want is to have each segment (*i.e.* coarsest-level node) be associated with some subset of the pixels (*i.e.* finest-level node), where each pixel belongs to *exactly* one segment.

The information we need comes from the interpolation matrices P calculated earlier. Between every two adjacent levels, P tells us how the fine-level nodes relate to the coarse-level nodes, or blocks. By multiplying all these interpolation matrices together, we can obtain the relationship between the the nodes on the finest level and those on the coarsest level.

Let us introduce the state matrix U. It keeps track of which nodes on a particular level belong to which segments. If m is the number of segments found on the coarsest level and M is the number of nodes on the current level, then U has dimensions $M \times m$. Much like P, the entries of U are defined such that U_{ij} indicates how much node i belongs to the j-th segment. In fact, U can be considered an interpolation matrix, only that it relates the current level to the coarsest level rather than to the immediate coarser level.

Starting from the coarsest level, every nodes belongs to itself, so U is an m by m identity matrix. On each subsequent (finer) level, we compute U by

$$U = PU^c,$$

where U is the current-level state matrix and U^c is from one level coarser. Eventually, we obtain U for the finest level.

Calculating U this way does not guarantee the final result to have binary values, since the product of all the P's may not be binary. Nevertheless, once on the finest level, we can simply assign each pixel to the segment that influences it the most. Along the way, we can also sharpen the image at each level so that the end result is close to being binary. Basically, at each level, after U is calculated, we promote values in U greater than 0.8 to 1, and demote all values less than 0.2 to 0. The thresholds 0.2 and 0.8 were set arbitrarily to reflect the amount of sharpening desired, but in generally they should add up to 1 to retain symmetry in the sharpening.

Chapter 3

The algorithm

3.1 Terminology review

In this multilevel algorithm, we have a graph connecting the nodes on each level. Any two adjacent levels are referred to as the fine level and the coarse level. A subset of the fine-level nodes are chosen as C-points (sometimes referred to as blocks because they represent overlapping blocks of the fine-level nodes). These C-points then become nodes for the coarse-level graph. Nodes on the finest level are referred to as pixels, while the nodes on the coarsest level (all of which are salient) are called segments. The entire algorithm is a 2-stage V-cycle. The first stage goes from the finest level to the coarsest level, finding C-points along the way. The second stage goes from the coarsest level back to the finest level in order to determine which pixels belong to each segment.

3.2 Variables

To keep the actual algorithm brief, we will first define all the variables used. Many are listed twice–once for the fine level and once for the caorse level. The use of these variables will become clear as they are introduced in the algorithm.

n = dimension of the square image
N = number of pixels in the image
M = number of nodes (fine level)
M^c = number of nodes (coarse level)
m = number of segments found on the coarsest level
I = intensity vector (fine level)
I^c = intensity vector (coarse level)
A = coupling matrix (fine level)
A^c = coupling matrix (coarse level)
L = boundary connection matrix (fine level)
L^c = boundary connection matrix (coarse level)
G = boundary length matrix (fine level)
G^c = boundary length matrix (coarse level)
W = internal connection matrix (fine level)
W^c = internal connection matrix (coarse level)
V = internal area matrix (fine level)
V^c = internal area matrix (coarse level)
Γ = saliency vector (fine level)
Γ^c = saliency vector (coarse level)
P = interpolation matrix
$\tilde{P}$ = column-normalized interpolation matrix
U = state matrix (fine level)
U^c = state matrix (coarse level)
α = initial intensity scaling factor for coupling weights
$\tilde{\alpha}$ = subsequent intensity scaling factor for coupling weights
θ = coarsening threshold
γ = saliency threshold
d_1 = lower sharpening threshold
d_2 = upper sharpening threshold
ℓ = current level (*i.e.* number of levels removed from the finest level)
σ = number of levels to keep saliency values at ∞

3.3 Non-recursive part

As mentioned before, the algorithm is implemented recursively. For clarity, we will describe the non-recursive part first.

1. Read in the grayscale image as an $n \times n$ intensity matrix with values from 0 (black)

to 1(white). Let $N = n^2$.

2. Define global parameters α, $\tilde{\alpha}$, θ, γ, d_1, and σ. For example,

$$(\alpha, \tilde{\alpha}, \theta, \gamma, d_1, \sigma) = (10, 10, 0.1, 0.1, 0.15, 2).$$

   Set $d_2 = 1 - d_1$.

3. Initialize variables ℓ, M, I, A, L, G, W, V and Γ for the finest level as follows:

   (a) Set $\ell = 1$.

   (b) Set $M = N$.

   (c) Obtain the $M \times 1$ intensity vector I by reshaping the $n \times n$ intensity matrix.

   (d) A is an $M \times M$ matrix with

$$A_{ij} = \begin{cases} e^{-\alpha|I_i - I_j|} & \text{if node } i \text{ and } j \text{ are neighbours} \\ 0 & \text{otherwise (including } i = j). \end{cases}$$

   Note that each node has 2 to 4 neighbours, depending on its location within the image.

   (e) L is defined by

$$L_{ij} = \begin{cases} -A_{ij} & \text{if } i \neq j \\ \sum\limits_k A_{ik} & \text{if } i = j. \end{cases}$$

   (f) G is defined by

$$G_{ij} = \begin{cases} -1 & \text{if } i \neq j,\ L_{ij} \neq 0 \\ 0 & \text{if } i \neq j,\ L_{ij} = 0 \\ -\sum\limits_k G_{ik} & \text{if } i = j. \end{cases}$$

   (g) Set $W = A$.

   (h) V is defined by

$$V_{ij} = \begin{cases} 0 & \text{if } A_{ij} = 0 \\ 1 & \text{if } A_{ij} \neq 0. \end{cases}$$

   (i) Γ is an $M \times 1$ vector with $\Gamma_i = \infty$ for all i.

4. Call the recursive function *imageVCycle* to get U:

$$U = imageVCycle(\ell, M, I, A, G, L, W, V, \Gamma).$$

   This function will be described in more detail in the next section.

5. Make the entries of U binary by setting

$$U_{ij} = \begin{cases} 1 & \text{if } j \text{ is the smallest value satisfying } U_{ij} = \max_k U_{ik}, \\ 0 & \text{otherwise.} \end{cases}$$

Now U_{ij} means pixel i belongs entirely to segment j. So each column of U corresponds to a segment, and the 1's within that column tell us which pixels belong to that segment.

3.4 Recursive part

The recursive part of the algorithm consists of the function *imageVCycle*. Its inputs and output are

- Inputs: ℓ, M, I, A, G, L, W, V, Γ, σ
- Output: U

And here are the steps within the function:

1. If $\ell \leq \sigma$, set $\Gamma_i = \infty$ for all $i = 1, 2, \cdots, M$.
2. Apply $coarsenAMG(A, \Gamma, \gamma, \theta)$ to get a vector C containing indices of the C-points chosen. Again, we will defer detailing the *coarsenAMG* function until the next section.
3. Let M^c be the length of vector C, which is the number of C-points selected.
4. Increment ℓ by 1.
5. If $M = M^c$, then output U as an $M \times M$ identity matrix since the current level is the coarsest. Otherwise continue to the next step.
6. Define the interpolation P by

$$P_{ij} = \begin{cases} 1 & \text{if node } i \text{ is a C-point and } i = C_j \\ 0 & \text{if node } i \text{ is a C-point but } i \neq C_j \\ \dfrac{A_{iC_j}}{\sum_{k \in C} A_{ik}} & \text{if node } i \text{ is an F-point.} \end{cases}$$

7. Define the column-normalized interpolation matrix $\tilde{P}$ by

$$\tilde{P}_{ij} = \frac{P_{ij}}{\sum_k P_{kj}}.$$

8. Define the intensity vector I^c by

$$I^c = \tilde{P}_{ij}^T I.$$

9. Define A^c in two steps:
   (a) Set $A^c = P^T A P$.
   (b) Modify A^c_{ij} by a factor of $e^{-\tilde{\alpha}|I^c_i - I^c_j|}$. That is,

$$A^c_{ij} \leftarrow A^c_{ij} e^{-\tilde{\alpha}|I^c_i - I^c_j|}.$$

10. Define L^c in two steps:
    (a) Set $L^c = -A^c$.
    (b) Replace the diagonal of L^c by its negative off-diagonal row sums. That is,

$$L^c_{ii} \leftarrow -\sum_{k \neq i} L^c_{ik}.$$

11. Define G^c in three steps:
    (a) Let $\hat{G}$ be G without its diagonal.
    (b) Set $G^c = P^T \hat{G} P$.
    (c) Replace the diagonal of G^c by its negative off-diagonal row sums. That is,

$$G^c_{ii} \leftarrow -\sum_{k \neq i} G^c_{ik}.$$

12. Let $W^c = A^c$.
13. Let $V^c = P^T V P$.
14. The saliency Γ^c is then determined in two steps:
    (a) Set $\Gamma^c_{ii} = \dfrac{L^c_{ii}/G^c_{ii}}{W^c_{ii}/V^c_{ii}}$.

(b) Replace all Γ^c_{ii} less than γ by 0.

15. Recursively call the function

$$U^c = imageVCycle(\ell, M^c, I^c, A^c, L^c, G^c, W^c, V^c, \Gamma^c).$$

16. Obtain the fine-level state matrix by

$$U = PU^c.$$

17. Sharpen the image by replacing all $U_{ij} < d_1$ by 0 and all $U_{ij} > d_2$ by 1.

18. Return U.

3.5 AMG coarsening

The function *coarsenAMG* has these inputs and output:

- Inputs: A, Γ, γ, θ
- Output: C

and follows these steps:

1. Let M be the number of rows (or columns) in A.

2. Define the strength matrix A^s by

$$A^s_{ij} = \begin{cases} A_{ij} & \text{if } i \neq j \text{ and } A_{ij} \geq \theta \cdot \sum_{k \neq i} A_{ik}, \\ 0 & \text{otherwise.} \end{cases}$$

   So A^s retains only strong connections in A. Keep in mind that A^s is not a symmetric matrix. This is important because $A^s_{ij} > 0$ means node i is *strongly influenced by* node j, but $A^s_{ji} > 0$ implies that node i *strongly influences* nodes j.

3. Let λ be an $M \times 1$ vector whose i-th entry λ_i equals the number of nonzero entries in column i of A^s. This value indicates how many nodes are strongly influenced by node i.

4. Initialize T as an $M \times M$ zero vector. This vector will keep track of the assignment status of each node so that $T_i = 0$ means node i is unassigned, $T_i = 1$ means node i has been assigned as a C-point, and $T_i = 2$ means node i has been assigned as an F-point. Note that T can only contain 0's, 1's and 2's.

5. For $i = 1, 2, \cdots, M$, if $\Gamma_i < \gamma$ then set $T_i \leftarrow 1$ and $\lambda_i \leftarrow 0$. This assigns all salient nodes as C-points.

6. While λ is not identically zero, do the following:

   (a) Find the unassigned node with the largest λ value (if there are many such nodes, choose the one with the small index). Index this node by j.

   (b) Make node j a C-point by setting $T_j \leftarrow 1$ and $\lambda_j \leftarrow 0$.

   (c) Let K be the indices of all unassigned nodes *strongly influenced* by node j. For $k \in K$, set $T_k \leftarrow 2$ and $\lambda_k \leftarrow 0$. This assigns all nodes in K as F-points.

   (d) Finally, we want to prioritize certain nodes so that they are more likely to be chosen as the next C-point. For each $k \in K$, let H be the indices of all unassigned nodes that *strongly influence* node k, and then for each $h \in H$, increment λ_h by 1.

7. Let C be the indices of all C-point.

8. Return C.

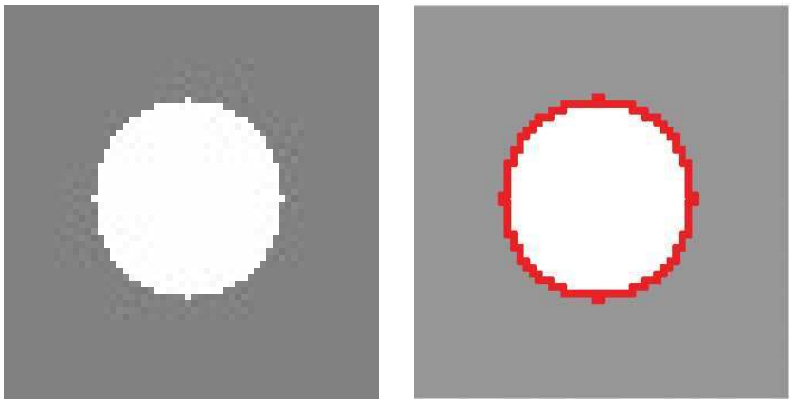
Chapter 4

Performance using only intensity

Currently the algorithm uses only the intensities of nodes to segment an image. Let us see how well it performs on some artificially constructed examples as well as actual cell images. For each example that follows, we have chosen the best parameters found through trial and error, and the parameters are listed in the caption for each figure. Each segment the algorithm finds will be outlined in red along its boundary. It is understood that the boundary of the image also contains some segment boundaries, so in general, we will not outline it.

4.1 Artificial examples

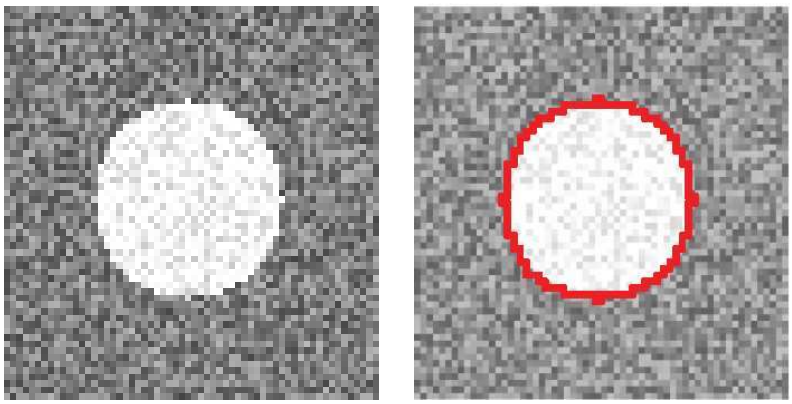
For a noiseless image with a sharp boundary such as Figure 4.1(a), the algorithm segments it perfectly, as Figure 4.1(b) shows.



(a) Original image (b) 2 segments found

Figure 4.1: $(\alpha, \tilde{\alpha}, \theta, \gamma, d_1, \sigma) = (10, 10, 0.1, 0.1, 0.15, 5)$

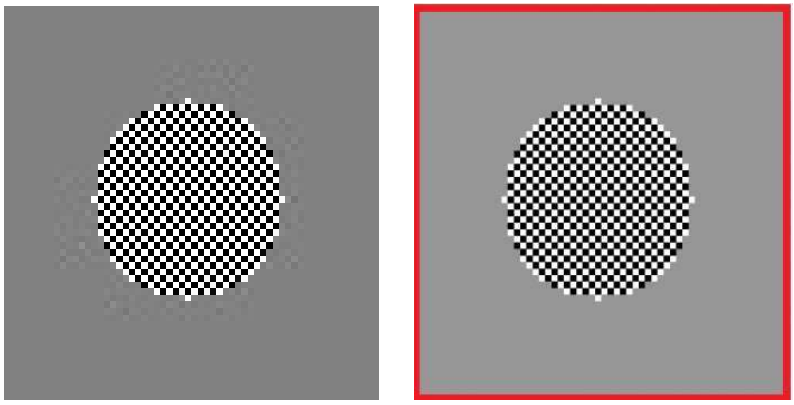
Now take the same image and add some random noise between -0.2 to 0.2 to get Figure 4.2(a). The algorithm can still segment it perfectly, as we can see in Figure 4.2(b), since there is enough intensity difference between the 2 segments.



(a) Original image (b) 2 segments found

Figure 4.2: $(\alpha, \tilde{\alpha}, \theta, \gamma, d_1, \sigma) = (10, 10, 0.1, 0.1, 0.15, 5)$

Now let us try an example, shown in Figure 4.3(a), with two segments that differ in texture but are identical in average intensity. As we see in Figure 4.3(b), the result is a poor segmentation in which only one segment is found. Since our algorithm uses only intensity as the distinguishing factor between segments, it cannot tell apart two segments of different textures but same average intensity, which is why they merged into one giant segment.

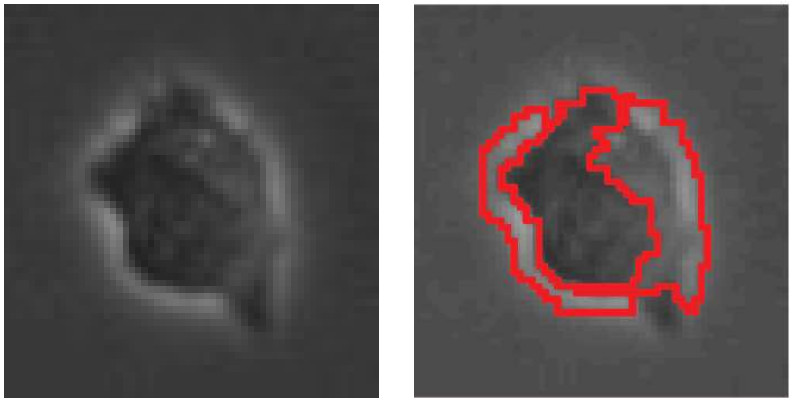


(a) Original image (b) 1 segment found

Figure 4.3: $(\alpha, \tilde{\alpha}, \theta, \gamma, d_1, \sigma) = (10, 10, 0.1, 0.1, 0.15, 5)$

4.2 Cell image examples

Having seen what the algorithm is capable of, we now apply it to some cell images. Using intensity differences, Figure 4.4(a) is found to have 4 segments, as shown in Figure 4.4(b).



(a) Original image (b) 4 segments found

Figure 4.4: $(\alpha, \tilde{\alpha}, \theta, \gamma, d_1, \sigma) = (100, 10, 0.18, 1.2, 0.15, 6)$

The halo around the cell is originally intended to create an outline of the cell, but parts of it became salient segments because they are distinctly brighter than the rest of the image. The intensity variation within the cell also causes the interior of the cell to split into 2 segments. As a result, we get 4 separate segments instead of just the two we want.

One way to differentiate between the cell and the background is to take into account the different textures of the two regions. While the background is quite homogeneous, the cell's interior exhibits more variation. So the next step is to try to improve our algorithm by incorporating texture in the form of variance.

Chapter 5

Incorporating texture

Texture can be viewed as variation within a region. So we will start by looking at how variation is calculated. Between any two adjacent levels, we have overlapping blocks each containing some subset of the fine-level nodes. For each block, we can measure its variance as the variance of the node intensities. For instance, if a block contains nodes with intensities $I_1, I_2, \cdots, I_r$, then the variance of the block, denoted by v^c, is calculated using the standard variance formula

$$v^c = \sum_{i=1}^{r} p_i (I_i - \bar{I})^2,$$

where p_i is the weight given to node i in the block and

$$\bar{I} = \sum_{i=1}^{r} p_i I_i$$

is the weighted mean of the node intensities.

To determine p_i, simply study the interpolation matrix P. Suppose the block we are looking at is the j-th block. Since P_{ij} tells us how much of node i contributes to block j, if we divide this value by total contributions of the nodes towards block j, the resulting value is the fraction of all contribution towards block j that comes from node i, or simply p_i. Basically p_i can be calculated as

$$p_i = \frac{P_{ij}}{\sum_{k=1}^{r} P_{kj}}.$$

Going back to the variance, we can use the identity

$$Var[X] = E[X^2] - E[X]^2,$$

where X is a random variable, to formulate a simpler expression for the variance of a block. Namely,

$$v^c = \sum_{i=1}^{r} p_i I_i^2 - \left(\sum_{i=1}^{r} p_i I_i \right)^2$$

So far, v^c describes the variance of the nodes within a block. But in our multilevel framework, the nodes also have variance within themselves since they are composed of even finer nodes. We should also take in account these finer-level variances. In fact, at any level, we want information for the variances at all previous finer levels. So we should associate each block with a vector that contains all its variances from the finest level down to the current level.

On the finest level, since all the nodes are just pixels with no inner structure, the variance within each is simply 0. Now suppose we are between two adjacent levels. On the fine level, let S be a matrix that contains the variance vectors of all the nodes. It is defined such that each row of S is a variance vector for a fine-level node, and each column corresponds to a level between the fine level and the finest level. Hence S_{i1} is the average variance of node i on the finest level (which is just 0), S_{i2} is the average variance of node i on the second finest level, and so on up to the last entry in row i, which is the average variance of node i on the current fine level.

Now given the variance matrix S for the fine level, we would like to compute S^c, the variance matrix for the coarse level. The variables needed for the calculation are S, I (fine-level intensity vector), I^c (coarse-level intensity vector), and $\tilde{P}$ (column-normalized interpolation matrix). These give us

$$S^c = \begin{bmatrix} \tilde{P}^T S & \tilde{P}^T I^2 - (I^c)^2 \end{bmatrix},$$

where I^2 and $(I^c)^2$ are the vectors I and I^c squared componentwise. Basically, the coarse-level variance vectors in S^c are calculated using the intensities I and I^c, while for all the other finer levels, the variance vectors in S^c are calculated from S as weighted sums.

At each level, we now have multilevel variance vectors for every node, given as rows of the matrix S. To include variance in the segmentation process, we need to modify the coupling matrix A so that A_{ij}–the coupling weight between node i and node j–depends not only on their difference in intensity but also on their difference in variance. Before, the difference in intensity was simply calculated as the absolute distance $|I_i - I_j|$. With variance, however, we need to calculate the difference between two variance vectors. A good way to do this is to take their Euclidean distance, or the 2-norm of their difference, given by

$$\sqrt{\sum_k (S_{ik} - S_{jk})^2} = ||\mathbf{s_i} - \mathbf{s_j}||_2,$$

where $\mathbf{s_i}$ and $\mathbf{s_i}$ are respectively the i-th and j-th row of S.

Now suppose we are between two adjacent levels, then the coarse coupling matrix A^c can be defined in two steps:

1. Set $A^c = P^T AP$.

2. Rescale A^c_{ij} by setting
$$A^c_{ij} \leftarrow A^c_{ij} e^{\tilde{\alpha}|I_i - I_j|} e^{\beta ||\mathbf{s_i} - \mathbf{s_j}||_2},$$
where $\tilde{\alpha}$ and β are predetermined scaling factors. They reflect, respectively, the importance of intensity and variance to the segmentation.

So incorporating texture into the segmentation requires changing only two things in the algorithm:

1. Define β.

2. Rescale A_{ij} by $e^{\beta ||\mathbf{s_i} - \mathbf{s_j}||_2}$ in addition to $e^{\tilde{\alpha}|I_i - I_j|}$.

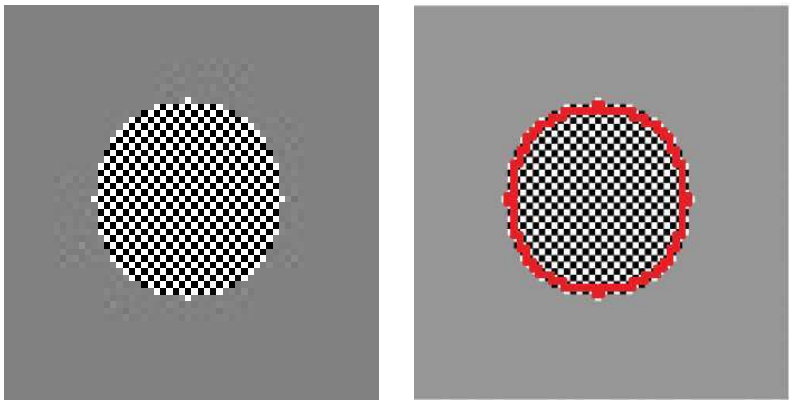
Chapter 6

Performance using intensity and texture

Now apply the algorithm to more images to evaluate its performance.

6.1 Artificial examples

Try the textural example that did not work earlier. As we can see, Figure 6.1(a) can now be segmented correctly, as shown in Figure 6.1(b).

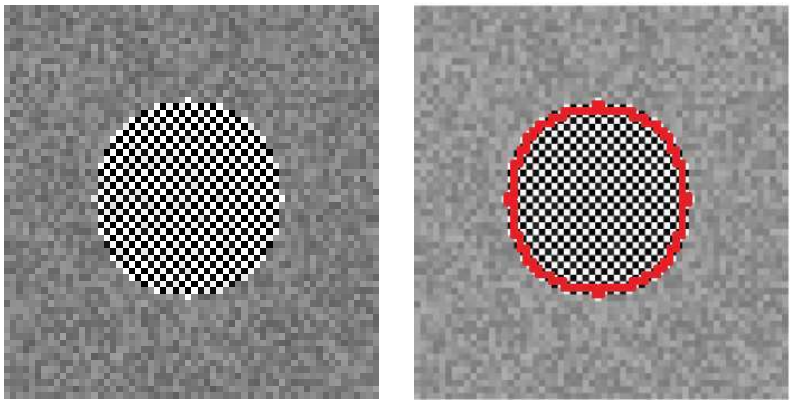


(a) Original image (b) 2 segments found

Figure 6.1: $(\alpha, \tilde{\alpha}, \beta, \theta, \gamma, d_1, \sigma, \rho) = (10, 10, 10, 0.1, 0.1, 0.15, 5, 1)$

To test the robustness of the algorithm, we will introduce some random noise between

-0.1 to 0.1 to Figure 6.1(a) to get Figure 6.2(a). It turns out that the image can still be segmented correctly, as shown in Figure 6.2(b).

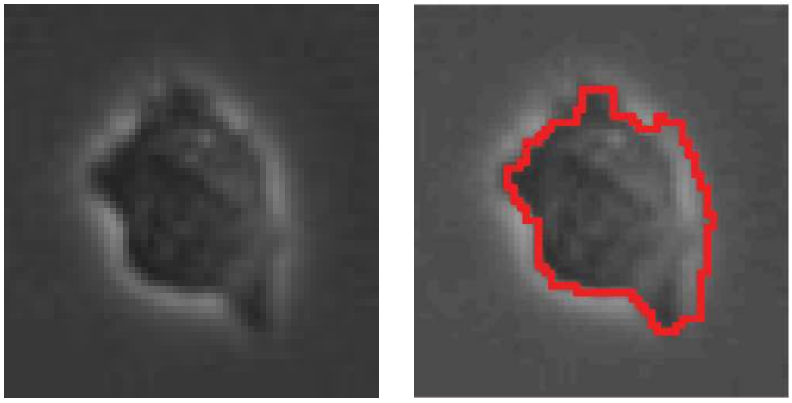


(a) Original image (b) 2 segments found

Figure 6.2: $(\alpha, \tilde{\alpha}, \beta, \theta, \gamma, d_1, \sigma, \rho)) = (10, 10, 10, 0.1, 0.1, 0.15, 5, 1)$

6.2 Cell image examples

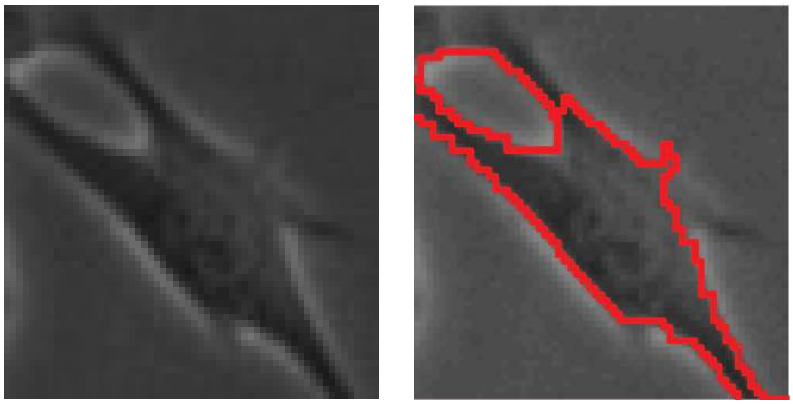
The results from the previous examples show promise. Now apply the algorithm to the cell images. First take Figure 6.3(a), an image that was segmented incorrectly earlier. The result is successful, as shown in Figure 6.3(b).



(a) Original image (b) 2 segments found

Figure 6.3: $(\alpha, \tilde{\alpha}, \beta, \theta, \gamma, d_1, \sigma, \rho) = (100, 4, 100, 0.18, 1.2, 0.15, 7, 1)$

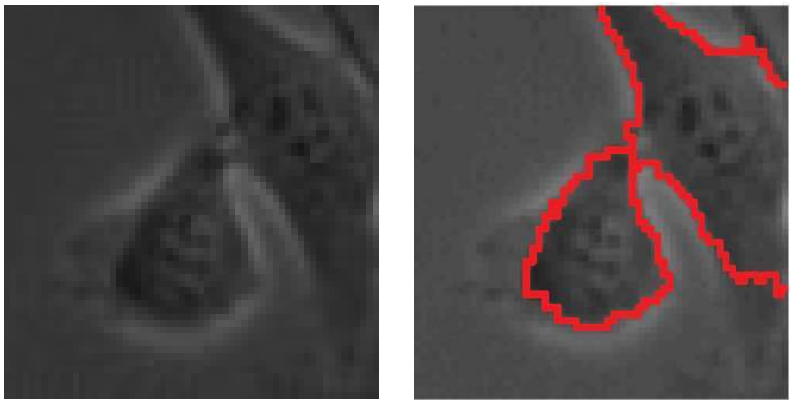
An image of a more irregularly-shaped cell (see Figure 6.4(a)) can also be segmented fairly well (see Figure 6.4(b)).



(a) Original image (b) 4 segments found

Figure 6.4: $(\alpha, \tilde{\alpha}, \beta, \theta, \gamma, d_1, \sigma, \rho) = (210, 0, 25, 0.138, 0.8, 0.15, 6, 1)$

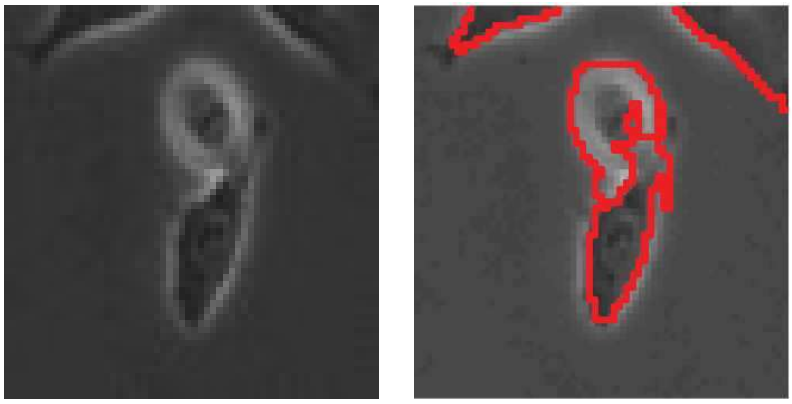
With 2 cells in the image such as Figure 6.5(a), the algorithm is still able to correctly identify them, as shown in Figure 6.5(b).



(a) Original image (b) 4 segments found

Figure 6.5: $(\alpha, \tilde{\alpha}, \beta, \theta, \gamma, d_1, \sigma, \rho) = (190, 0, 110, 0.13, 0.8, 0.15, 5, 1)$

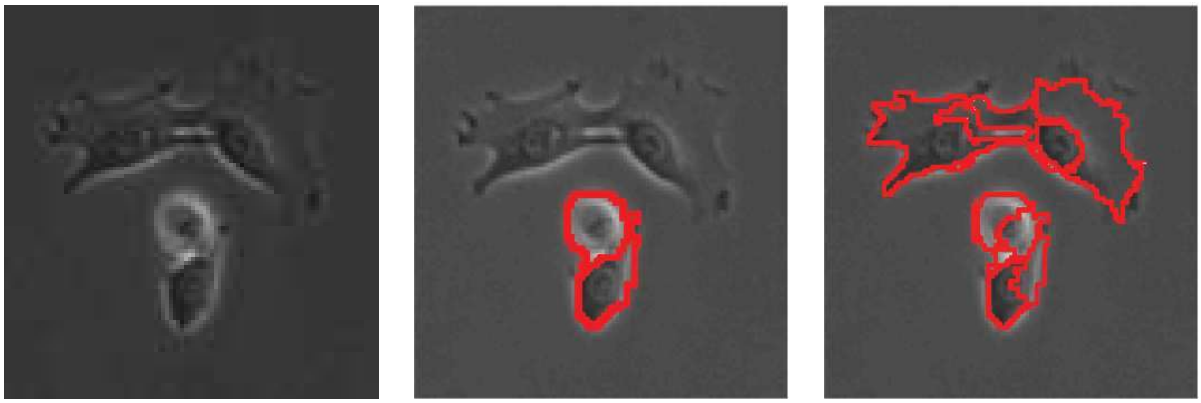
The next example in Figure 6.6(a) contains 2 whole cells and 2 partially cropped cells. Figure 6.6(b) shows that we are able to find the 5 segment (4 cells plus background), although there is a little "leakage" from one cell segment to another, because the boundaries are not so clear-cut.



(a) Original image (b) 5 segments found

Figure 6.6: $(\alpha, \tilde{\alpha}, \beta, \theta, \gamma, d_1, \sigma, \rho) = (200, 0.9, 90, 0.15, 0.5, 0.15, 5, 1)$

Let us try Figure 6.7(a), a more difficult example involving 4 whole cells at once. With one choice of parameters, we get Figure 6.7(b), which correctly identifies the 2 cells at the bottom with sharper boundaries but completely misses the top 2 cells. With another choice of parameters (simply changing σ from 6 to 5), we get Figure 6.7(c), where all 4 cells are found, but instead of 4 segments, we get 13 segments. Neither results are ideal.



(a) Original image (b) 3 segments found, $\sigma = 6$ (c) 13 segments found, $\sigma = 5$

Figure 6.7: $(\alpha, \tilde{\alpha}, \beta, \theta, \gamma, d_1, \rho) = (400, 0, 80, 0.2, 0.65, 0.15, 1)$

Chapter 7

Extending to 3D problems

In this chapter, we consider the problem of segmenting multiple images at once. These images should be taken over a period of time with small time intervals between consecutive ones so that there is a sense of continuity. For example, we could be given 100 images showing the movement of a cell. Since the cell does not move much between frames, there is a lot of overlap (or similarity) between consecutive images. This suggests that we can actually apply the image segmentation in 3D, with time as the third dimension.

7.1 3D modifications

It is not actually difficult to modify the algorithm to suit a 3D problem because the algorithm is already designed to coarsen arbitrary graphs, which can represent geometric grids of any dimension. The only part we need to change is the variable initialization.

Let k be the number of consecutive images we wish to segment. Then here are a list of things that need to be changed from the non-recursive part of the algorithm. To highlight the changes, we will not rewrite any steps that do not change.

1. Read in the k grayscale images as $n \times n$ intensity matrices, $I_1, I_2, \cdots, I_k$, taking values from 0 (black) to 1 (white). Let $N = n^2$.
2. (The global parameters are defined as before)
3. Initialize variables ℓ, M, I, A, L, G, W, V and Γ for the finest level as follows:
   (a) Set $M = Nk$.

(b) Obtain the $M \times 1$ intensity vector I by reshaping the $n \times nk$ intensity matrix given by

$$\begin{bmatrix} I_1 & I_2 & \cdots & I_{k-1} & I_k \end{bmatrix}.$$

(c) A is an $M \times M$ matrix with

$$A_{ij} = \begin{cases} e^{-\alpha|I_i - I_j|} & \text{if node } i \text{ and } j \text{ are neighbours} \\ 0 & \text{otherwise (including } i = j\text{).} \end{cases}$$

Note that nodes now have extra neighbours in the third dimension (the time direction). So each node has anywhere from 3 to 6 neighbours, depending on its location in space-time.

(d) (The other variables are defined as before).

4. (Call the recursive function as before)

5. (U is defined as before)

Everything else in the algorithm, namely the recursive part, remains unchanged.

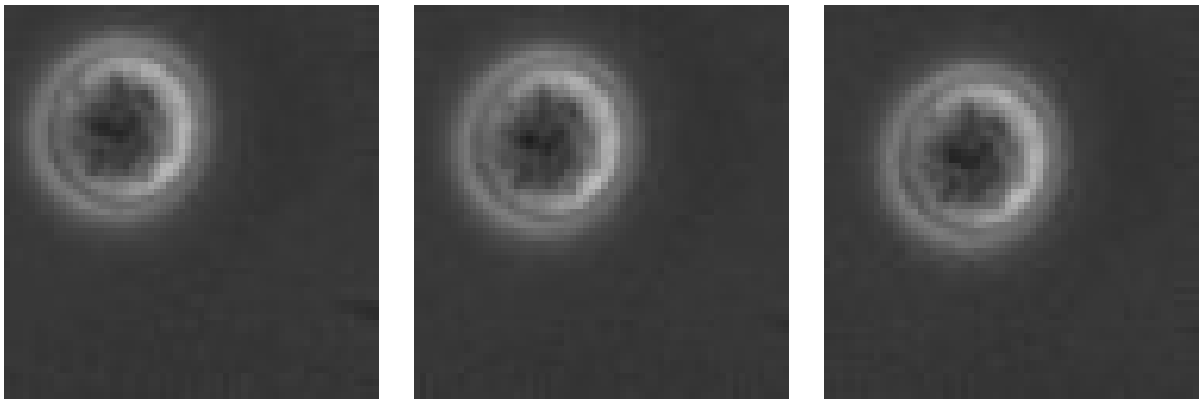
Chapter 8

Performance on 3D examples

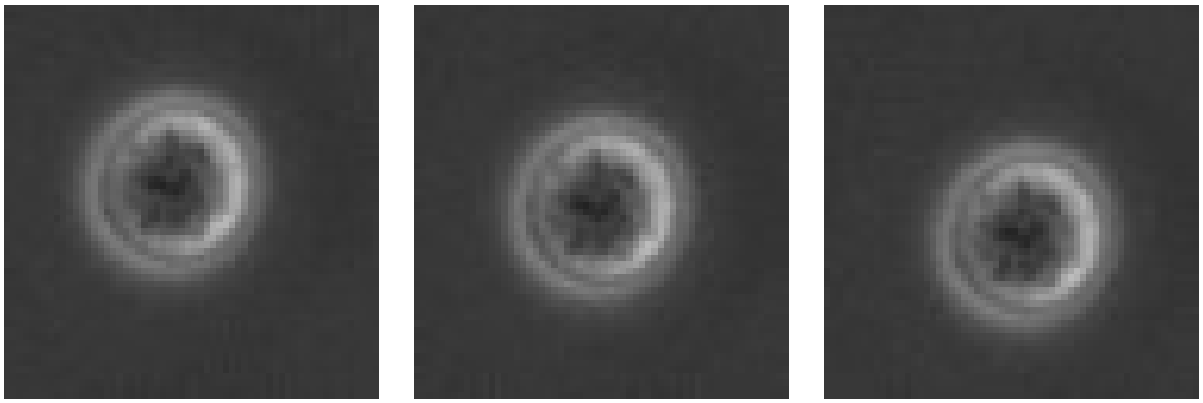
Test the algorithm with a 3D example to evaluate its performance.

8.1 Cell image example

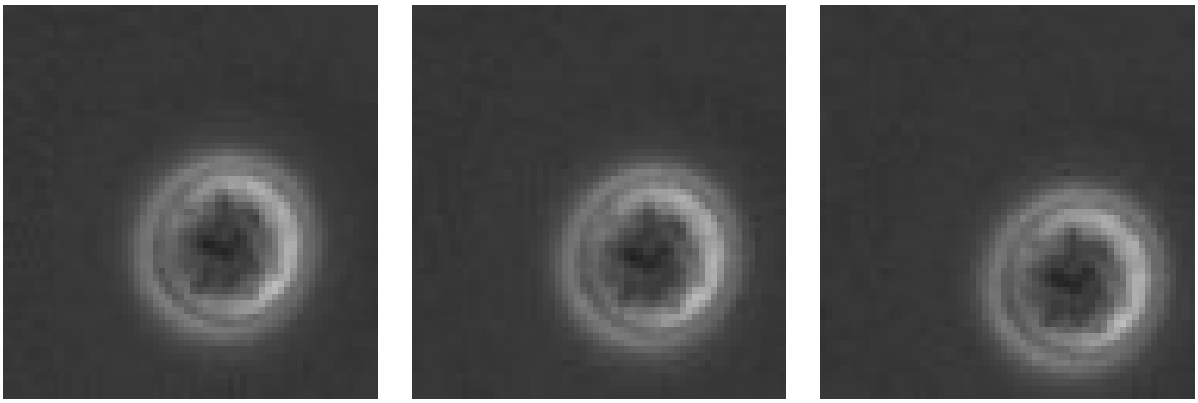
Consider the 9 cell images, shown in Figure 8.1, taken sequentially in time. From $t = 1$ to $t = 9$, the circular cell moved from the top-left corner to the bottom-right corner of the frame.



(a) Original image at $t = 1$ (b) Original image at $t = 2$ (c) Original image at $t = 3$



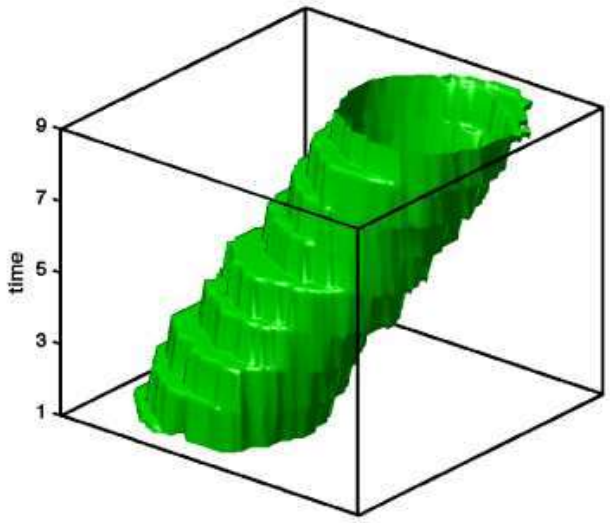
(d) Original image at $t = 4$ (e) Original image at $t = 5$ (f) Original image at $t = 6$



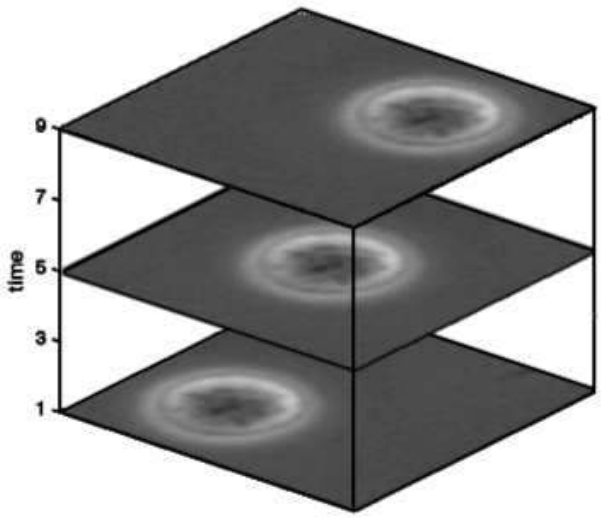
(g) Original image at $t = 7$ (h) Original image at $t = 8$ (i) Original image at $t = 9$

Figure 8.1: Original 3D image

Apply the algorithm to this 3D stack of images and we get 2 segments as shown in Figure 8.2(a). The green tube connecting opposite corners of the box is the segment representing the cell's location in space-time. Figure 8.2(b) shows 3 slices of the actual image, and they agree with Figure 8.2(a) in terms of the cell's location. So the segmentation result is correct.



(a) 2 segments found in 3D



(b) Corresponding image slices

Figure 8.2: $(\alpha, \tilde{\alpha}, \beta, \theta, \gamma, d_1, \sigma, \rho) = (10, 18, 200, 0.01, 0.01, 0.15, 6, 1)$

Chapter 9

Conclusion and future work

We have presented an algorithm that takes into account intensity and texture on different levels to produce a desirable segmentation. It differs from other multilevel segmentation methods in its use of AMG coarsening–a highly optimized coarsening method–and a saliency measure that identifies salient segments more accurately. The algorithm can also be readily extended to 3D problems, which is useful for tracking cell activities.

We implemented the algorithm mostly in Matlab, with the exception of the AMG coarsening step, which is programmed in C. Wherever possible, we vectorized functions and stored matrices in sparse forms in order to reduce the memory and time requirements. Currently the running time for segmenting a 512×512 image is just under 2 minutes, although this quantity depends heavily on the coarsening threshold θ.

The algorithm performs well on brightfield images containing few cells, but the quality of the segmentation deteriorates as more cells are introduced. This is mainly due to the low-contrast boundaries and the broken halos that surround the cells. One way to overcome this problem is to analyze boundary continuities between blocks using a top-down method [6].

Currently, the texture measure we use is isotropic, meaning that the algorithm cannot distinguish between two regions with the same pattern, but oriented differently. Figure 9.1 shows one such example.

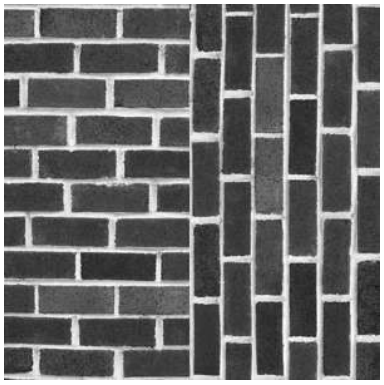


Figure 9.1: Two segments indistinguishable by the isotropic texture measure

To properly segment this image, we must incorporate more features of the blocks found, such as their lengths, widths and orientations. These are collectively known as the *shape elements* [3].

One drawback of the algorithm is that it is not fully autonomous. For every image we want to segment, there are 8 parameters to specify. Although some parameters usually stay fixed, having to explore within a multidimensional parameter space is still time-consuming and infeasible for larger problems.

In the future, we would like to include boundary continuity and shape elements in the segmentation. Even though this will increase the size of the parameter space, hopefully having more useful features will lessen the parameter sensitivity on less important features, so that we will be able to obtain better segmentation results without modifying more parameters. As for the 3D problems, we can use the unidirectional property of time to track multiple cells better.

Bibliography

[1] Paul Bao, Lei Zhang, and Xiaolin Wu. Canny edge detection enhancement by scale multiplication. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(9):1485–1490, September 2005. 2

[2] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A Multigrid Tutorial*. SIAM, 2 edition, 2000. 12

[3] Meirav Galun, Eitan Sharon, Ronen Basri, and Achi Brandt. Texture segmentation by multiscale aggregation of filter responses and shape elements. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:716–723, 2003. 4, 43

[4] Thrasyvoulos N. Pappas. An adaptive clustering algorithm for image segmentation. *IEEE Transactions on Signal Processing*, 40(4):901–914, April 1992. 2

[5] Eitan Sharon, Achi Brandt, and Ronen Basri. Fast multiscale image segmentation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:70–71, 2000. 4

[6] Eitan Sharon, Achi Brandt, and Ronen Basri. Segmentation and boundary detection using multiscale intensity measurements. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:469–476, 2001. 42

[7] Eitan Sharon, Meirav Galun, Dahlia Sharon, Ronen Basri, and Achi Brandt. Hierachy and adaptivity in segmenting visual scenes. *Nature*, 442:810–813, August 2006. 4, 9

[8] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000. 2