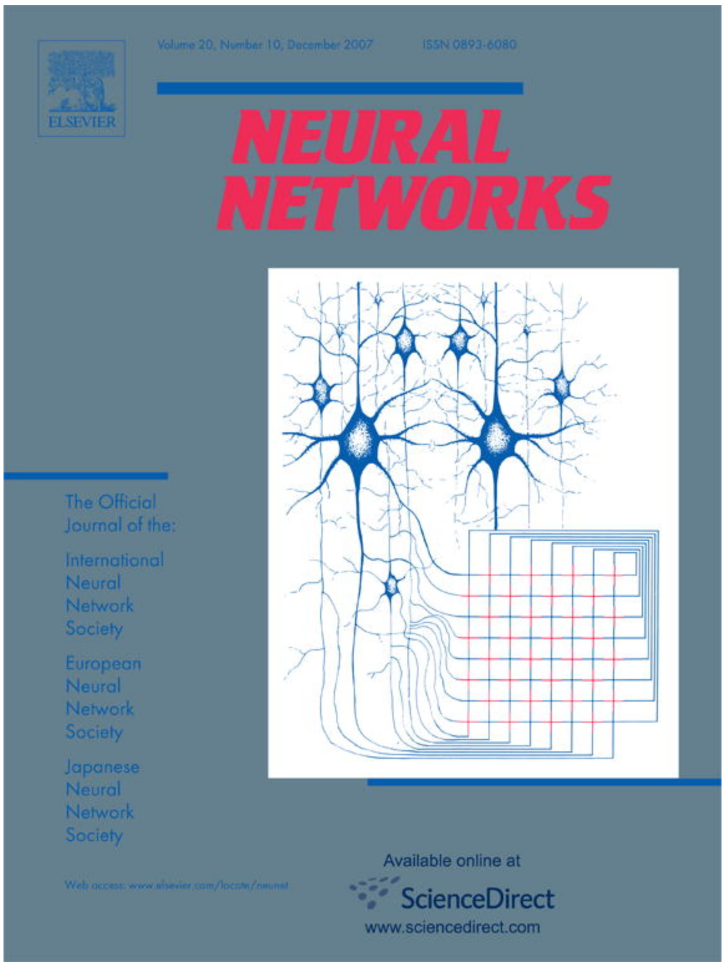
Volume 20, Number 10, December 2007
ISSN 0893-6080
ELSEVIER
NEURAL NETWORKS
The Official Journal of the:
International Neural Network Society
European Neural Network Society
Japanese Neural Network Society
Web access: www.elsevier.com/locate/neunet
Available online at
ScienceDirect
www.sciencedirect.com

Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 20 (2007) 1109–1131
Neural Networks
www.elsevier.com/locate/neunet

CONFIGR: A vision-based model for long-range figure completion

Gail A. Carpenter[*], Chaitanya Sai Gaddam, Ennio Mingolla

Department of Cognitive and Neural Systems, Boston University, 677 Beacon Street, Boston, MA 02215, USA

Received 14 February 2007; received in revised form 27 September 2007; accepted 27 September 2007

Abstract

CONFIGR (CONtour FIgure GRound) is a computational model based on principles of biological vision that completes sparse and noisy image figures. Within an integrated vision/recognition system, CONFIGR posits an initial recognition stage which identifies figure pixels from spatially local input information. The resulting, and typically incomplete, figure is fed back to the "early vision" stage for long-range completion via filling-in. The reconstructed image is then re-presented to the recognition system for global functions such as object recognition. In the CONFIGR algorithm, the smallest independent image unit is the visible pixel, whose size defines a computational spatial scale. Once the pixel size is fixed, the entire algorithm is fully determined, with no additional parameter choices. Multi-scale simulations illustrate the vision/recognition system. Open-source CONFIGR code is available online, but all examples can be derived analytically, and the design principles applied at each step are transparent. The model balances filling-in as figure against complementary filling-in as ground, which blocks spurious figure completions. Lobe computations occur on a subpixel spatial scale. Originally designed to fill-in missing contours in an incomplete image such as a dashed line, the same CONFIGR system connects and segments sparse dots, and unifies occluded objects from pieces locally identified as figure in the initial recognition stage. The model self-scales its completion distances, filling-in across gaps of any length, where unimpeded, while limiting connections among dense image-figure pixel groups that already have intrinsic form. Long-range image completion promises to play an important role in adaptive processors that reconstruct images from highly compressed video and still camera images.
© 2007 Elsevier Ltd. All rights reserved.

Keywords: Multi-scale image completion; Filling-in; Occlusion; BCS/FCS; Computer vision

1. Completing, connecting, and uniting image figures

In the process of recognizing objects, the human visual system encounters long-range featural gaps, derived from physiology, occlusion, and image sparseness. Early visual areas V1-V2-V4 compensate for such gaps by completing boundaries and filling-in features, as modeled, for example, by the Boundary Contour System/Feature Contour System (BCS/FCS) neural network (Cohen & Grossberg, 1984; Grossberg & Mingolla, 1985a, 1985b) and its many extensions.

The initial goal of the CONFIGR (CONtour FIgure GRound) project was to define a ready-to-use system for large-scale image completion that would build upon the function and design of the BCS/FCS model family. While partially accomplishing this goal, CONFIGR embodies substantially distinct design principles and functional capabilities. The new model carries out long-range contour completion via complementary processes that fill-in both figure and ground. The same general-purpose system also connects sparsely represented images (dots) and unifies occluded objects. A CONFIGR user need choose just one free parameter: the size of the smallest independent, or "visible," unit (pixel) in a given image. Once the pixel size, and hence the computational spatial scale, is specified, the algorithm is fully determined, even analytically computable. Intrinsic self-limiting mechanisms prevent spurious completions while permitting unimpeded filling-in across arbitrary distances.

In the experimental domains of perceptual psychophysics and cognitive neuroscience, CONFIGR filling-in more closely resembles amodal completion (e.g., filling-in of occluded contours and surfaces) than modal completion (e.g., filling-in of perceived brightness) (Pessoa & De Weerd, 2003). In the technological domains of sensor design and image reconstruction, CONFIGR suggests new algorithms for signal recovery from incomplete and inaccurate measurements

[*] Corresponding author. Tel.: +1 617 353 9483; fax: +1 617 353 7755.
E-mail addresses: gail@cns.bu.edu (G.A. Carpenter), gsc@cns.bu.edu (C.S. Gaddam), ennio@cns.bu.edu (E. Mingolla).

0893-6080/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2007.10.002

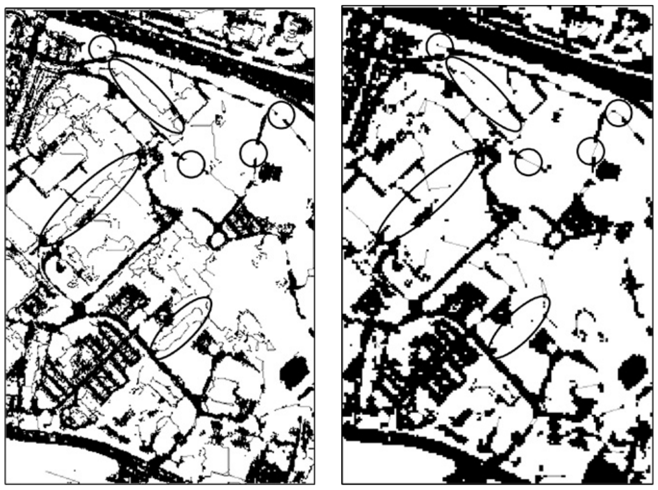
(a) Fine scale. (b) Coarse scale.

Fig. 1. (a) Road pixels (dark) in the benchmark Monterey image were initially located by an ARTMAP network using local visual features (Parsons & Carpenter, 2003). At this finest available spatial scale, CONFIGR completes figure contours (light pixels). (b) The coarse-scale image was created from 2 × 2 blocks of fine-scale pixels. At the coarse scale, CONFIGR fills-in road segments that it missed at the fine scale (circles), but misses contours that connect isolated pixels at the fine scale (ellipses).

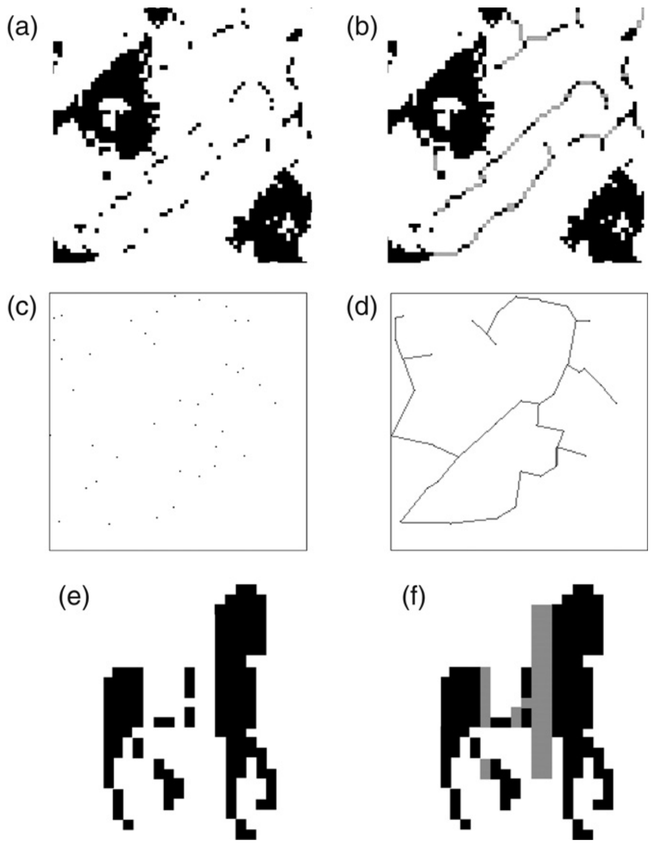

Fig. 2. ((a), (b)) Completion: Where is the road? In an image detail from Fig. 1(a), CONFIGR completes the figure *road*. ((c), (d)) Connection: Where are the links? CONFIGR connects 40 random dots. The original figure covers 0.1% of a 200 × 200 pixel square. ((e), (f)) Union: Where is the horse? For an image inspired by Magritte's painting *Blank Check*, CONFIGR unifies the occluded object.

(Candes, Romberg, & Tao, 2006), with potential applications to compressive imaging with sparse representations (Takhar et al., 2006) and new camera designs.

Fig. 1 shows how CONFIGR and ARTMAP (Carpenter, 2003; Carpenter, Grossberg, Markuzon, Reynolds, & Rosen, 1992) can work together in an integrated multi-scale vision/recognition system. This simulation illustrates how the system solves a potentially circular problem for a completion mechanism faced with a complex image, namely:

> Before object recognition takes place, how does the system know what should connect with what?

CONFIGR addresses this problem by assuming that recognition precedes, as well as follows, "early" vision, a sequence now also being explored in the experimental literature (Ahissar & Hochstein, 2004). Inputs to the long-range completion portions of the model are the result of the initial recognition step, which identifies figure pixels based on spatially local features. The vision system then completes the figure, thereby preparing it for global recognition. Various target classes might be defined as *figure*, as could pixel subsets of a given depth, color, or texture.

In Fig. 1(a), the recognition system first identifies *road* pixels from local features derived from the sensor input. The resulting image (dark pixels) contains many gaps caused by cars, overhanging trees, shadows, and errors. Initial *road* pixel locations are fed back to the CONFIGR vision system, which carries out long-range figure completion (light pixels).

Fig. 1(b) illustrates supplementary filling-in of the Monterey *road* figure at a coarser spatial scale, which merges 2 × 2 fine-scale pixel squares into one pixel via a simple smoothing algorithm. At the coarse scale, CONFIGR completes many of the gaps that the fine scale missed (circles). However, because the coarse scale treats isolated fine-scale pixels as noise, it erases many of the isolated figure pixels that form coherent connected contours (ellipses).

The Monterey simulation example suggests a strategy for multi-scale CONFIGR completion that defines the completed figure as the sum of the original image-figure pixels plus completions made at each plausible spatial scale. Note that the CONFIGR algorithm is identical across scales, once the pixel size is fixed.

Fig. 2(a), (b) shows a detail of fine-scale Monterey *road* pixels, and their CONFIGR completions. This image fragment (around the lowest ellipse in Fig. 1(a)) illustrates filling-in of a road that surrounds a building. At coarser spatial scales, nearly all of the initial road pixels, and hence their completions, are missing (Fig. 1(b)). Note, too, the absence of spurious figure completions in and around nearby paved areas.

The same CONFIGR algorithm that completes contours also connects random dots, with any degree of separation, forming coherent clusters (Fig. 2(c), (d)); and unites an occluded object (Fig. 2(e), (f)). Details of these and other examples illustrate CONFIGR mechanisms throughout the article. Sections 2 and 3 define the CONFIGR computational elements, and Section 4 specifies the algorithm. Section 5 illustrates the system's components with analytically computed examples, and Section 6 shows CONFIGR simulations,

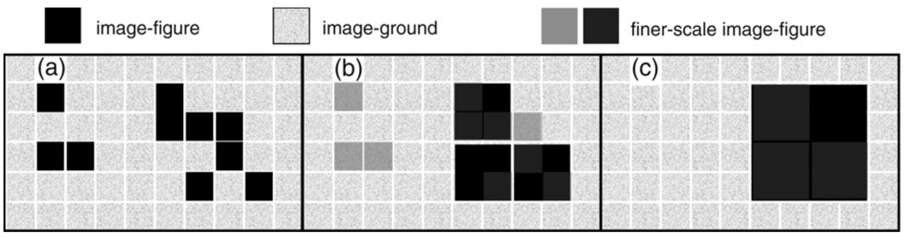

Fig. 3. Image pixels at three spatial scales: (a) fine, (b) medium, and (c) coarse. Each coarser-scale pixel is created from a 2 × 2 square of pixels in a finer-scale image via a smoothing algorithm. A coarser-scale pixel is labeled *image-figure* if the number of finer-scale image-figure pixels in the 2 × 2 square, plus half the number of image-figure pixels among the surrounding 12, is at least three.

including multi-scale Monterey examples and images where the random dots are denser than Fig. 2(c), (d). CONFIGR algorithm code is available in Matlab and C++ from http://cns.bu.edu/techlab/CONFIGR/. Section 7 discusses the special case of images that are exactly aligned with the vertical and horizontal computational directions of the algorithm, and Section 8 indicates future directions.

2. Figure, ground, and spatial scale

The first step of the CONFIGR algorithm is the specification of a computational spatial scale. This choice entails defining an integral square image unit, or *pixel*. For a given image, fewer pixels produce a coarser scale and more pixels produce a finer scale (Fig. 3). The pixel serves as the smallest independent, or "visible," unit of the image. The side of a pixel defines one unit of length.

The spatial scale (or, equivalently, the number of pixels in the image) is the only CONFIGR free parameter. Once this scale is fixed, system dynamics are fully determined and are, in fact, analytically computable. The same parameter-free computational algorithm serves for problems of completion, connection, and occlusion across all spatial scales. Parameter independence reflects the fact that CONFIGR is based on a minimal set of principles that produce a specified set of functional capabilities. It also promotes ease of use in a variety of image processing applications. Earlier BCS/FCS implementations, in contrast, typically required image-specific parameter selection, e.g., Gove, Grossberg, and Mingolla (1995, Appendix) and Mingolla, Ross, and Grossberg (1999, Table 1).

Pixels are initially labeled as *image-figure* or *image-ground*. This labeling may be the result of a pattern recognition procedure that identifies the initial set of figure pixels based on features that are spatially local. Alternative criteria for image-figure labels include color, depth, or edges. CONFIGR completes missing figure portions by relabeling some image-ground pixels as filled-figure or filled-ground (Fig. 4). The final figure at a given spatial scale is taken to be the sum of image-figure and filled-figure pixels. Complementary filling-in at different scales (Fig. 1) suggests a default strategy of summing filled-figure pixels across spatial scales to form the final percept.

3. CONFIGR computational elements

The basic CONFIGR architecture is designed to embody the simplest possible image elements. Accordingly, computations are restricted to vertical and horizontal directions. System analysis explores the computational capabilities of this minimal model system, whose limits suggest directions for future development—CONFIGR 2++ (Section 8).

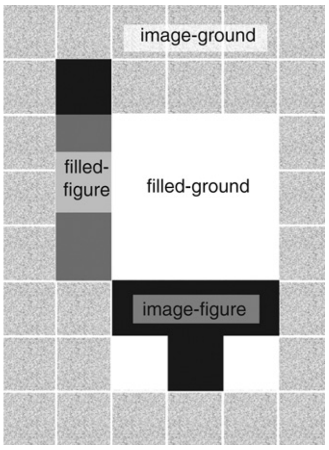

Fig. 4. CONFIGR pixel labels: image-figure (dark grey), filled-figure (lighter grey), image-ground (light grey texture), and filled-ground (white). The image shows the result of CONFIGR filling-in of three figure and eleven ground pixels, starting with five image-figure pixels. The Monterey fine-scale example (Fig. 1(a)) includes a similar configuration, with the image-figure pixels initially identified as *road* by the spatially local recognition system, and the final figure pixels forming a connected contour.

The computational elements that work together to define the CONFIGR model are now defined.

3.1. Rectangles, subpixels, and the grid

Defined by vertical and horizontal pixel boundaries, all CONFIGR image components are sums of rectangles. The *size of a rectangle* equals its height plus its width, with length measured in numbers of pixels along edges.

The smallest independent image rectangle is a pixel. Since one unit of length is defined as equal to the side of a pixel, the size of a pixel equals two at each spatial scale. Each pixel corresponds to a 5×5 array of square subunits, called *subpixels*, aligned as shown in Fig. 5. A *grid subpixel* is centered at the meeting point of four image pixels. CONFIGR system values at grid subpixels determine filling-in decisions.

3.2. Simple and complex cells

Together with left or right (for vertical) and up or down (for horizontal), the two CONFIGR directions produce four

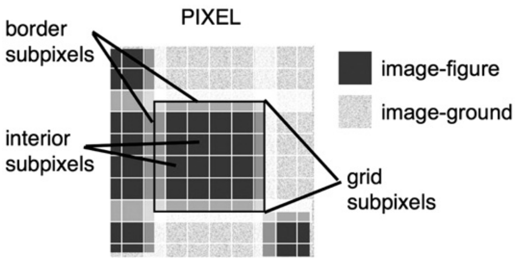

Fig. 5. Each pixel overlaps 16 full subpixels (interior), 16 half subpixels (border), and 4 quarter subpixels (grid).

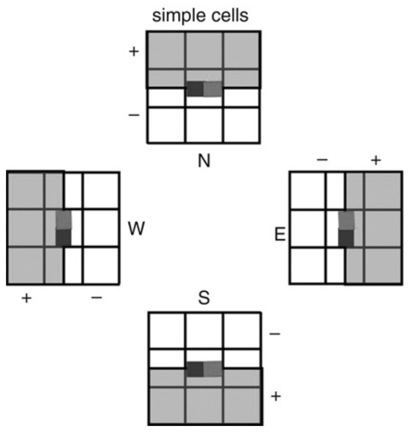

Fig. 6. N, S, E, and W simple cell receptive fields, with shaded activation masks. The dimensionless activation threshold $\in [\frac{1}{4}, \frac{1}{3})$. A dark-light bar in a center subpixel indicates the east or west (vertical) or north or south (horizontal) orientation of an active simple cell.

orientations: west (W) or east (E) and north (N) or south (S). To calculate the local orientations of the initial image-figure boundaries, each subpixel is associated with a set of *simple cells*, one for each orientation (N, S, E, W). For each orientation, the 3×3 subpixel square centered on the simple cell's image subpixel is its *receptive field*.

A simple cell's activation is determined by the image-figure pixels intersecting its receptive field. A simple cell sums image-figure subpixel fractions (+ or −) within its receptive field according to its orientation (Fig. 6). Within its receptive field, an east (E) simple cell, for example, sums image-figure subpixel fractions to the east of its midline; subtracts from this the image-figure subpixel fractions to the west of its midline; and divides this difference by the number of subpixels in the receptive field (9), to produce an *activation ratio* $\in [-\frac{1}{2}, +\frac{1}{2}]$.

A simple cell is (by definition) active if its activation ratio exceeds a given threshold. The *activation threshold* is set high enough ($\geq 1/4$) so that simple cells centered at image-figure corners are not active, but low enough ($< 1/3$) so that simple cells centered at border subpixels adjacent to these corners are active (Fig. 7(a)). An east (E) simple cell is active if the net number of image-figure subpixels to the east of its midline minus the number to the west is greater than 2.25, or one quarter the number of subpixels in the receptive field. A north, south, east, or west simple cell is active at each subpixel on the boundary between image-figure and image-ground, except for the grid subpixels that are located at convex or concave image-figure corners (Fig. 7(b)).

Complex cells sum the activation of two simple cells. That is, a *horizontal complex cell* subpixel is active if a north or a south simple cell is active at that subpixel. A *vertical complex cell* subpixel is active if an east or a west simple cell is active at that subpixel.

Vertical and horizontal complex cells are active in the same subpixel locations as simple cells (Fig. 7(b)). CONFIGR computations may be considered to be based upon either sums of simple cells or sums of complex cells. A light-dark bar that denotes an active simple cell in a subpixel center also denotes an active complex cell at the same location.

3.3. Lobe activation and propagation

Starting with BCS/FCS modeling (Cohen & Grossberg, 1984; Grossberg & Mingolla, 1985a, 1985b) and experimental investigations of the visual cortex (Von der Heydt, Peterhans, & Baumgartner, 1984), bipole cells have come to be viewed as fundamental computational units of visual processing. A typical bipole cell becomes active in response to activity in two adjacent lobes. Computations like those of bipole cells are key components of models of association fields (Field, Hayes, & Hess, 1993) and relatable contours (Kellman & Shipley, 1991).

CONFIGR incorporates particular types of lobes and bipole cells. This section defines model lobes, and Section 3.4 describes how lobes activate CONFIGR bipole cells. With computations restricted to N, S, E, and W orientations, lobe receptive fields can be modeled simply as subpixel strips. A *lobe receptive field* consists of a base subpixel and a line of subpixels extending N, S, E, or W from the base. The *lobe size* (3) is the number of subpixels in the line, including the base (Fig. 8). Initially, lobes receive input from complex [or simple] cells (iteration #0). Thereafter, a lobe iteratively receives input from active base subpixels of other lobes with the same orientation.

A lobe responds at its base subpixel (*lobe activation*) if at least two of its three receptive field subpixels receive input (Fig. 9(a)). Initially, an east or west lobe sums horizontal complex cell activity in its receptive field; and a north or south lobe sums vertical complex cell activity in its receptive field. A pair of perpendicular lobes is then active at each image-figure corner, and one or two lobes are active at other subpixels on the image-figure boundary. Where two image-figure pixels touch at corners, four lobes may be initially active at one base subpixel (Fig. 9(a, detail).

After initialization, lobes iteratively activate other lobes with the same orientation (Fig. 9(b)). At each step, a lobe of a given orientation responds at its base subpixel if lobes of that orientation are active in at least two of its three receptive field subpixels. Each iteration computes lobe activations for five subpixel (one pixel) steps, extending from one grid subpixel to the next.

In order to avoid spurious boundary effects, CONFIGR computations are carried only so far as simple cell and lobe receptive fields fit into the image. A half-pixel fringe suffices.

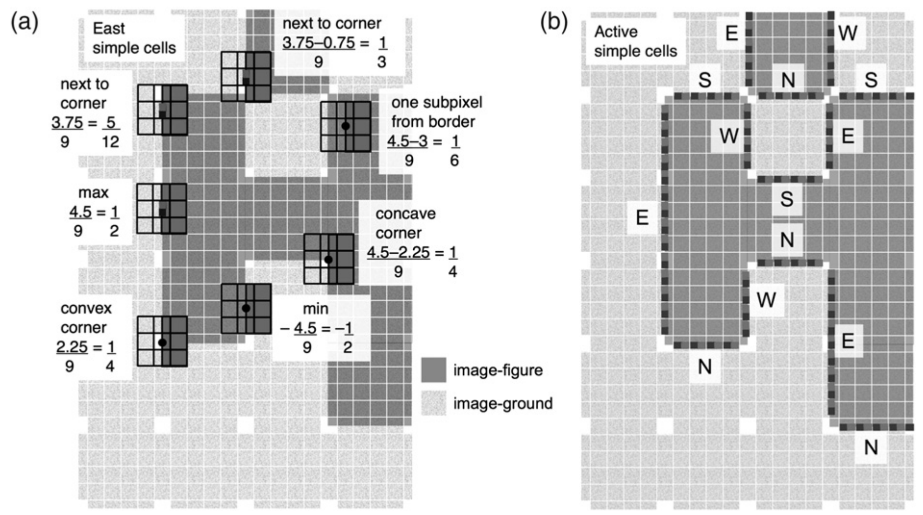
(a)
East simple cells
next to corner $\frac{3.75-0.75}{9} = \frac{1}{3}$
next to corner $\frac{3.75}{9} = \frac{5}{12}$
one subpixel from border $\frac{4.5-3}{9} = \frac{1}{6}$
max $\frac{4.5}{9} = \frac{1}{2}$
concave corner $\frac{4.5-2.25}{9} = \frac{1}{4}$
convex corner $\frac{2.25}{9} = \frac{1}{4}$
min $-\frac{4.5}{9} = -\frac{1}{2}$
image-figure
image-ground
(b)
Active simple cells
E
W
S
N
S
W
E
E
S
N
W
E
N
N

Fig. 7. Simple and complex cell activations. (a) A simple cell is active if its activation ratio exceeds an activation threshold $\in [\frac{1}{4}, \frac{1}{3})$. A dark-light bar indicates the center of the receptive field of an active simple cell, and a dot indicates an inactive simple cell. (b) Vertical complex cells are active at subpixels where either E or W simple cells are active, and horizontal complex cells are active where either N or S simple cells are active. Note that simple cells are not active at image-figure corners.

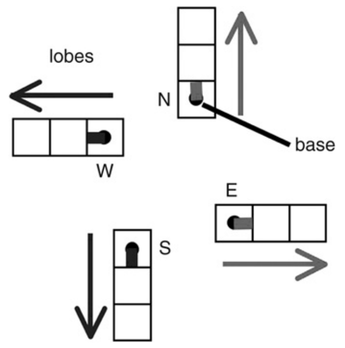

Fig. 8. Lobe receptive fields. A light or dark bar in a base subpixel indicates an active lobe, along with its north, south, east, or west orientation.

3.4. Lobe corners and empty rectangles

A classical bipole cell, with lobe receptive fields oriented 180° relative to one another, responds most strongly to collinear groupings. Recently, models (Grossberg & Mingolla, 1987; Grossberg & Swaminathan, 2004; Hansen & Neumann, 2004) and experiments (Pasupathy & Connor, 1999) that consider lobes at other relative orientations have investigated bipole cells that respond to other angles.

CONFIGR featural filling-in is based on bipole cells with lobes that are 90° relative to one another. A subpixel where one of these right-angle bipoles is active is called a *lobe corner*. A lobe corner (NE, NW, SE, or SW) is a grid subpixel with perpendicular active lobes (Fig. 10). CONFIGR defines two mutually exclusive classes of lobe corners: empty and filled. A lobe corner is *empty* if the pixel within its defining perpendicular lobes is image-ground. A lobe corner is *filled* if it is not empty; i.e., the pixel within its defining perpendicular lobes is image-figure or filled-figure or filled-ground. CONFIGR filling-in converts some empty corners to filled corners. Once a lobe corner is filled, it remains filled.

CONFIGR defines three mutually exclusive classes of empty corners: figure, ground, and wall (Figs. 9 and 11). Back-to-back empty corners are *wall corners*. An *empty ground corner* is an empty corner whose defining perpendicular lobes are flanked by pixels that are image-figure or filled-figure. For example, an empty NE lobe corner is an empty ground corner if the pixels to its north and east are image-figure or filled-figure. An empty corner that is neither a wall corner nor an empty ground corner is an *empty figure corner*. That is, it does not share a lobe with another empty corner; and at least one flanking pixel is image-ground or filled-ground.

An *empty rectangle* is an array of pixels that is spanned by a diagonal pair of empty lobe corners, neither of which is a wall corner. In addition, all of the rectangle's pixels are image-ground; and at least one lobe is active at each subpixel on the rectangle's edges. Fig. 11(b) contains three empty rectangles, of which one is spanned by a pair of empty figure corners and two are spanned by empty figure/empty ground corner pairs. The CONFIGR algorithm specifies which empty rectangles fill as figure and which fill as ground.

4. CONFIGR algorithm

CONFIGR realizes a set of computational principles, or rules, that govern lobe propagation and featural filling-in. The first rule states that formation of an empty corner stops lobe propagation. The second and third rules specify which empty rectangles fill as ground and which fill as figure. Other things being equal, ground fills more quickly than figure, and smaller rectangles fill before larger ones. An empty rectangle fills as ground if it contains an empty ground corner or if it is adjacent to one or more filled-ground pixels. After ground has filled-in on a given iteration, remaining empty rectangles fill as figure (Fig. 12). Examples in Section 5 illustrate these rules at work.

The steps of the CONFIGR algorithm will now be described. Matlab and C++ implementations of this system are available from: http://cns.bu.edu/techlab/CONFIGR/.

CONFIGR algorithm step 1: *Preparing the image*

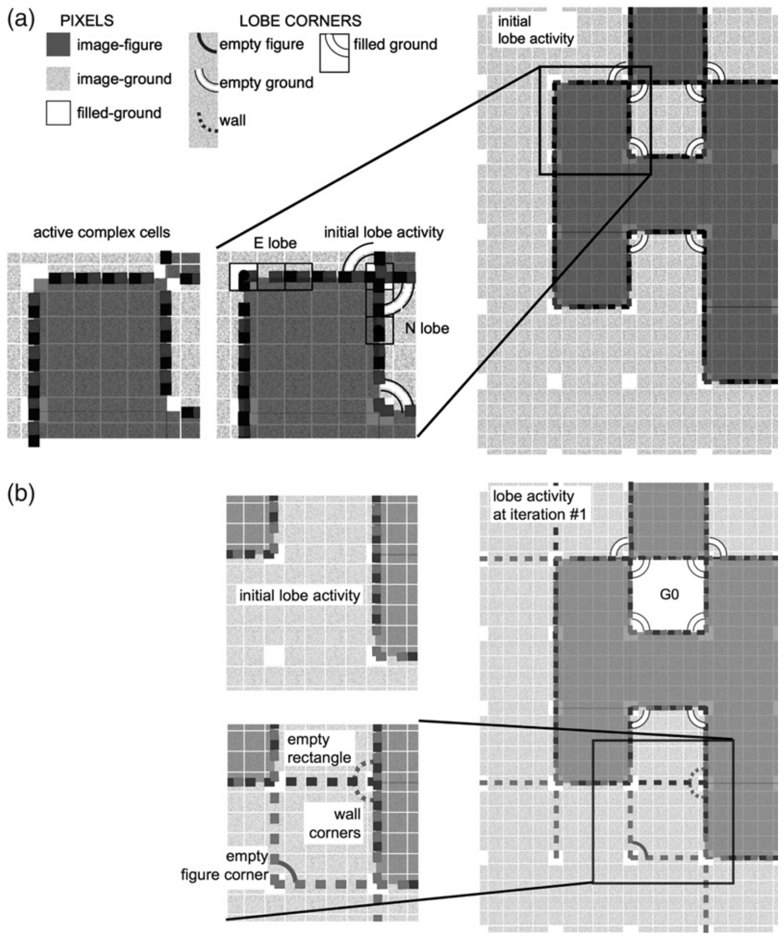

Fig. 9. Lobe activation. (a) Each lobe initially sums complex cell activity in its receptive field (iteration #0). After lobe initialization, each image-figure corner becomes a lobe corner. In this image fragment, initial lobe activation produces eight empty ground corners, one at each of the image-figure concavities. (b) After lobe initialization, an empty rectangle of size 2 (one pixel) fills as ground (G0). Lobe iteration #1 creates one empty figure corner and two wall corners. The empty figure corner and its diagonally opposite empty ground corner span an empty rectangle composed of two pixels. Recall that the rectangle's size (3) equals its height (2) plus its width (1).

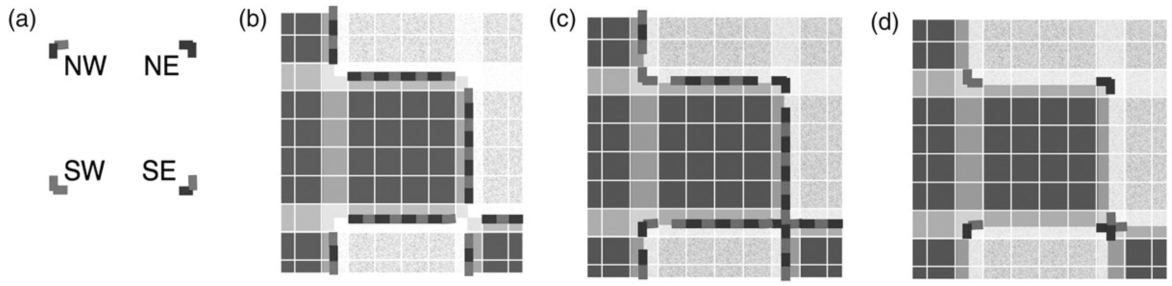

Fig. 10. Lobe corners. (a) Each lobe corner is defined by a pair of perpendicular active lobes, located at a grid subpixel. (b) Complex cell activations around the pixel shown in Fig. 5. (c) Initial lobe activations. (d) Initial activation produces seven lobe corners at convex and concave image-figure corners. Four of these are empty and three are filled. The grid subpixel at the lower right contains four lobe corners.

Image: Select a rectangular image.

Choose the spatial scale: Specify the size and location of one pixel in the designated image.

Storage matrices: Create matrices to store locations and variable values for pixels, subpixels, grid subpixels, simple and complex cells, and lobe corners.

Figure and ground: Label each pixel as image-figure (1) or image-ground (0).

Simple cell activation: At each subpixel and for each orientation (N, S, E, W), compute the simple cell activation (1 = active, 0 = inactive).

Complex cell activation: At each subpixel with an active simple cell, compute the vertical or horizontal complex cell activation (1 = active, 0 = inactive).

CONFIGR algorithm step 2: *Lobe initialization*

Lobe iteration number: Set the lobe iteration number to 0.

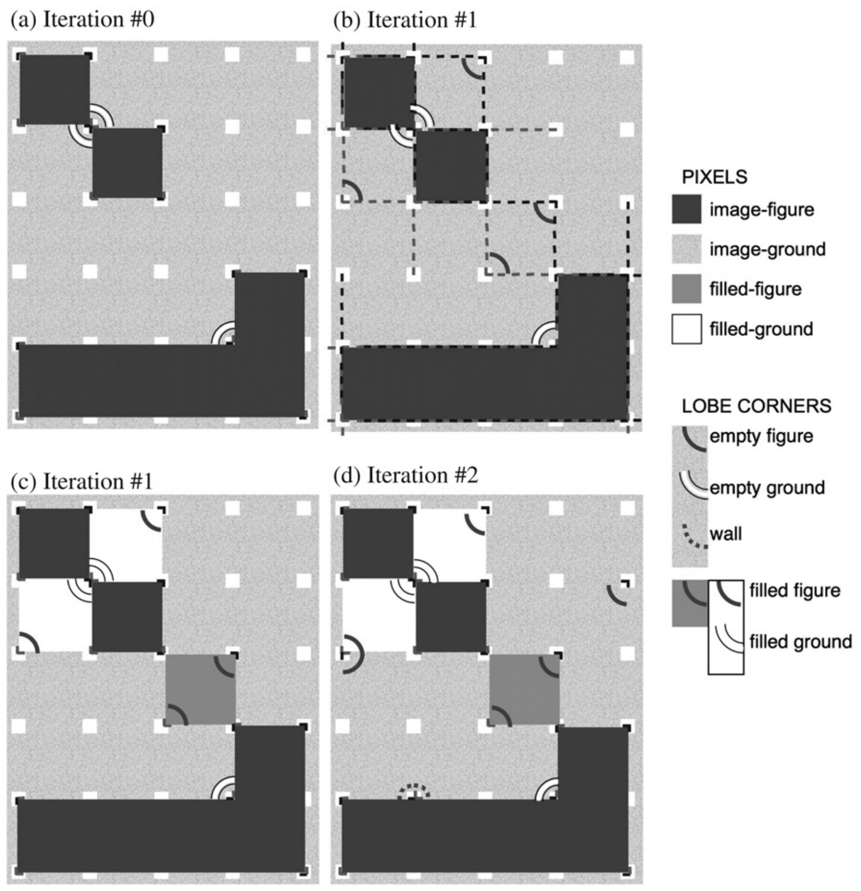

Fig. 11. Lobe corners, as computed by the CONFIGR algorithm. (a) Lobe initialization produces three empty ground corners, at the concave corners of the image figure. Each convex corner of the image figure also produces a lobe corner, which is already filled with figure. (b) Lobe iteration #1 produces four new lobe corners, and three empty rectangles. (c) By the end of iteration #1, two empty rectangles are filled as ground and one is filled as figure. (d) Lobe iteration #2 produces two back-to-back empty corners (wall corners), and two empty figure corners. The left-hand empty figure corner abuts another lobe corner, but this was filled on the previous iteration, so neither corner is a wall. Subsequent iterations produce no more lobe corners or filling-in. CONFIGR is thus seen to connect the figure components without creating spurious completions.

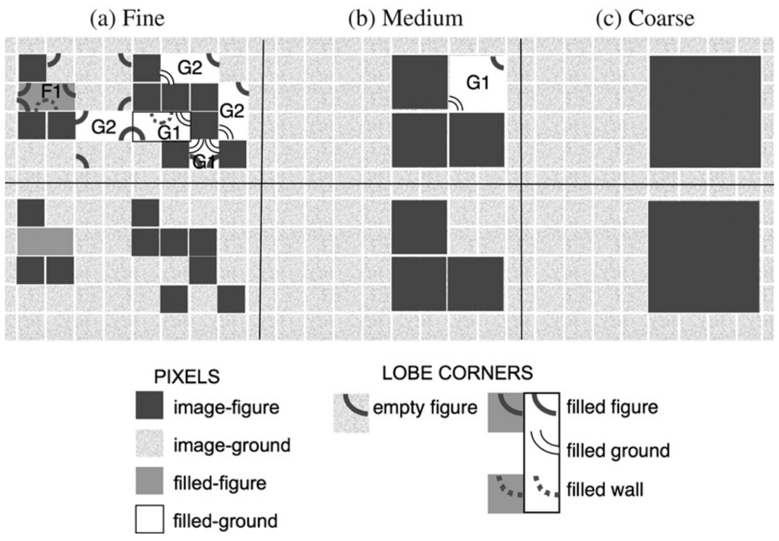

Fig. 12. Filling-in of the multi-scale images from Fig. 3. (a) At the fine spatial scale, CONFIGR connects the left-hand pixel group, forming two connected figure components. At lobe iteration #1, rectangles of size 2 and 3 fill as ground (G1), then another rectangle of size 3 fills as figure (F1). Three more rectangles (G2) fill as ground at iteration #2, two because they contain a ground corner and one because it is adjacent to a filled-ground pixel. No pixels fill as figure at the medium (b) and coarse (c) spatial scales. The lower row shows the final figure components after CONFIGR filling-in.

Lobe activation: At each subpixel and for each orientation (N, S, E, W), compute the initial lobe activation (1 = active, 0 = inactive).
Lobe corners: Specify each lobe corner type (NE, NW, SE, SW) and the location of its grid subpixel.
Corner labels: Label each lobe corner as a filled corner or an empty ground corner.
Empty rectangles: Mark each empty rectangle.
Filling-in as ground: Relabel as filled-ground the pixels of each empty rectangle.
Update corner labels: Relabel the four lobe corners spanning each filled rectangle as filled corners.

CONFIGR algorithm step 3: *Lobe iterations*

LOBE STOPPING RULE: Stop lobe propagation where two lobes form an empty corner.

Start lobe iterations{
Lobe iteration number: Increase the lobe iteration number by 1.
Lobe activation: At each subpixel and for each orientation (N, S, E, W), compute the lobe activation (1 = active, 0 = inactive). Repeat for a total of five subpixel steps.
New corners: For each new lobe corner, specify each lobe corner type (NE, NW, SE, SW) and the location of its grid subpixel.
Empty corner labels: Label each new lobe corner as empty figure, empty ground, or wall. Relabel as wall each existing empty figure corner that is back-to-back with a new wall corner.
Empty rectangles: List all NW empty corners that are not walls.
For each listed corner, search for SE empty corners that span empty rectangles with this lobe corner. When a NW–SE pair of empty corners spans an empty rectangle, mark it for potential filling-in on this iteration.
Repeat the search, marking each NE–SW pair of empty corners that spans an empty rectangle.
Sort the marked rectangles from smallest to largest.

GROUND FILLING RULE: An empty rectangle is eligible for filling-in as ground if it contains an empty ground corner or if it shares an edge with one or more filled-ground pixels.

Loop from smallest to largest empty rectangle (filling-in as ground){
Relabel as filled-ground the pixels of each empty rectangle that is the size of the loop's index and that is eligible for filling-in as ground. Relabel as filled each corner whose defining lobes are on the border or in the interior of the filled rectangle.

Update corners and rectangles

Relabel wall corners that have become empty figure or empty ground. For each such corner, add newly created empty rectangles to the marked list.
Relabel as filled-ground the pixels of each newly created empty rectangle of size equal to or smaller than the current loop size, if the rectangle is eligible for filling-in as ground. Relabel newly filled corners.
Remove from the list of marked rectangles all that are no longer empty, because they intersect newly filled rectangles.

Iterate corner and rectangle updates until no more changes occur.

}***End empty rectangle loop (filling-in as ground)***

FIGURE FILLING RULE: Remaining empty rectangles are eligible for filling-in as figure.

Loop from smallest to largest empty rectangle (filling-in as figure){
Relabel as filled-figure the pixels of each empty rectangle that is the size of the loop's index. Relabel as filled each corner whose defining lobes are on the border or in the interior of the filled rectangle.

Update corners and rectangles

After all rectangles of the loop size have been filled as figure, relabel affected corners.
Some empty corners that had previously been wall may now be empty figure or empty ground corners. For each such corner, add newly created empty rectangles to the marked list. Fill as ground newly created marked rectangles, if the rectangle contains an empty ground corner or is adjacent to one or more filled-ground pixels.
Remove from the list of marked rectangles all that are no longer empty, because they intersect newly filled rectangles.
Fill as figure each remaining newly created marked rectangle of size equal to or smaller than the current loop size.
Remove from the list of marked rectangles all that are no longer empty, because they intersect newly filled rectangles.

Iterate corner and rectangle updates until no more changes occur.

}***End empty rectangle loop (filling-in as figure)***
}***End lobe iterations***

5. CONFIGR algorithm illustrations

Examples in this section illustrate computations of the CONFIGR algorithm. These results are derived analytically. Only the Monterey and random dot examples (Section 6) are produced by computer simulation. Even in such large-scale examples, each CONFIGR detail can be readily checked by hand.

5.1. Filling-in as ground

Upon lobe initialization (iteration #0), the only possible type of empty rectangle is one that is surrounded by image-figure pixels, with four empty ground corners. As seen in the image fragment from Fig. 9, such an empty rectangle fills as ground (Fig. 13(a)).

Lobe iteration #1 produces an empty rectangle of size 3 (Fig. 13(b)). This rectangle contains empty ground corners and hence fills as ground. The five lobe corners (two ground, two wall, one figure) within the filled rectangle are then no longer empty. Diagonal lines run through the grid subpixels that are a city-block distance of 0, 1, 2, . . . , 10 from the NW image corner.

According to the lobe stopping rule, propagation from the lower left of the rectangle G1 ceases upon formation of the

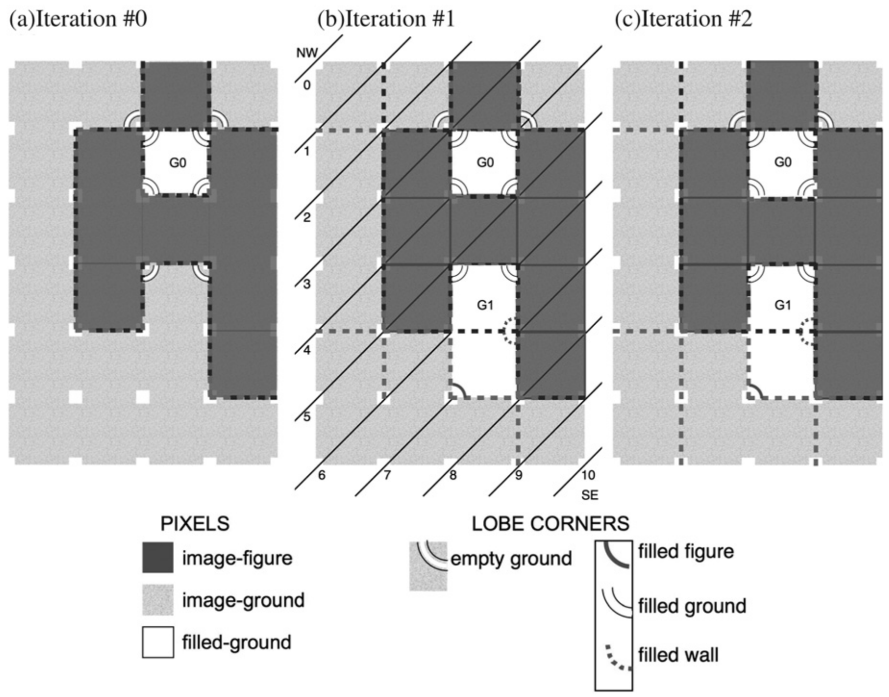

Fig. 13. Filling-in as ground in an image fragment. (a) G0 indicates a rectangle that fills as ground during the lobe initialization step (iteration #0). (b) At iteration #1, an empty rectangle fills as ground (G1). (c) Lobe activity after iteration #2. Lobe propagation from the lower central corner ceased when that corner formed during iteration #1.

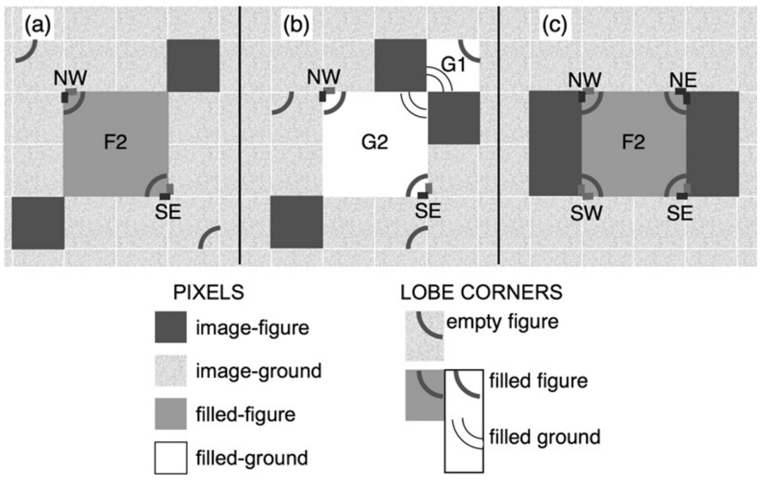

Fig. 14. Each image shows all the lobe corners and filled rectangles that CONFIGR would ever create, even if the lobe iterations were to continue indefinitely.

empty figure corner. Iteration #2 produces no additional empty corners or rectangles (Fig. 13(c)). The final CONFIGR figure consists of the original image-figure pixels, with no additional pixels filled as figure.

5.2. *Filling-in as figure*

Fig. 14 illustrates both ground and figure filling-in. In Fig. 14(a), starting with two image-figure pixels, CONFIGR creates two empty figure corners on lobe iteration #2. These corners span an empty rectangle, which fills as figure (F2). Iteration #3 creates two more empty figure corners, which span no empty rectangles and never fill-in. In Fig. 14(b), lobe initialization produces two empty ground corners, at concave corners of the image figure. Iteration #1 produces one empty figure corner (upper right), which pairs with one of the empty ground corners to span an empty rectangle, which fills as ground (G1). Iteration #2 produces four more empty figure corners, which span the central empty rectangle (size 4), plus two overlapping empty rectangles (size 5). After the smallest of these empty rectangles fills as ground (G2), the larger rectangles are no longer empty. In Fig. 14(c), active lobes generated by the opposing image-figure bars form four empty figure corners at iteration #2. The resulting empty rectangle fills as figure (F2), completing across the gap.

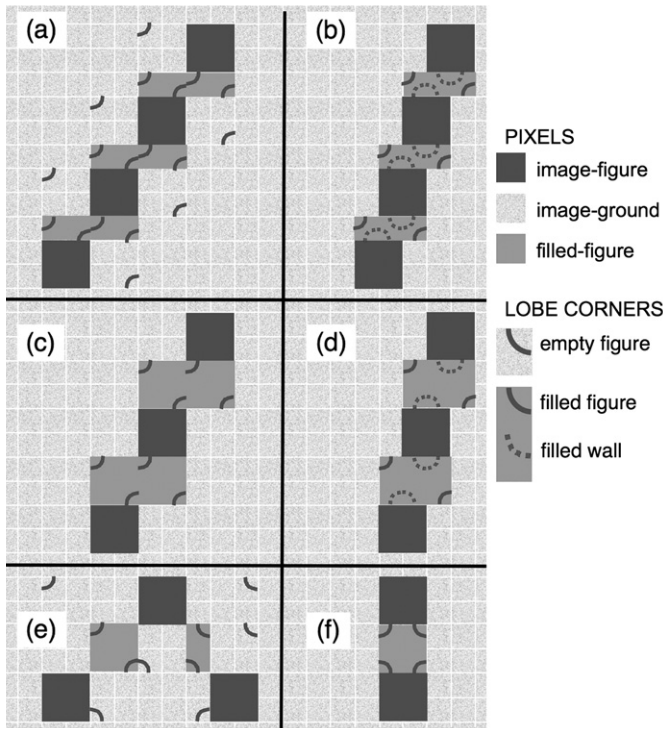

Fig. 15. Collinear grouping examples. In each case, CONFIGR mechanisms fill-in contours, limit lobe corner creation, and block spurious filling-in. All filling-in occurs on iteration #1 or #2. In (a) and (e), iterations #3–5 produce the additional lobe corners shown, but no empty rectangles.

5.3. Collinear groupings

Fig. 15 shows how CONFIGR computes collinear groupings of image-figure pixels arranged at various spacings and angles. The system employs different computational mechanisms for different cases. Fig. 15(b) and (d) create wall corners which help prevent spurious filling-in. The back-to-back corners in Fig. 15(e) are never walls, because the left corner is filled at iteration #2, before the right corner is created (iteration #3). In these examples, CONFIGR produces no empty ground corners and no filled-ground pixels.

5.4. Filling-in may relabel empty corners

An algorithmic implementation of the CONFIGR system needs to respect the fact that filling of an empty rectangle may change the status of other rectangles. For example, when two empty rectangles intersect, once the smaller fills, the larger is no longer empty, as in Fig. 14(b).

Filling-in may also change lobe corner labels. For example, if one of a pair of corners that form a wall fills-in, the second wall corner may become an empty ground or empty figure corner, which might then define a new empty rectangle.

Fig. 16 shows how ground filling-in (iteration #2) may cause an adjacent wall corner to be relabeled empty figure. At the next iteration, this corner is one of a pair of empty figure corners that span an empty rectangle. This rectangle fills as ground (G3) because of its adjacency to previously filled ground (G2).

The filling-in sequence in Fig. 17, all within iteration #1, illustrates how an empty rectangle, newly created by lobe relabeling, might fill immediately thereafter. When the central rectangle fills as ground (c), two adjacent wall corners become empty figure (d), creating two new empty rectangles. These fill immediately (e), because they are smaller than the size currently being filled within the algorithm's ground-filling loop.

Fig. 18 illustrates why the CONFIGR algorithm specifies that a rectangle newly filled as figure may reclassify an adjacent empty figure corner as an empty ground corner. Without this relabeling, the large rectangle (G3) would have filled, unreasonably, as figure. A similar pixel configuration occurs in the upper right section of the Monterey example (Fig. 1(a)). Fig. 18 demonstrates the CONFIGR computations that produced Fig. 4.

The example in Fig. 19(a), (b) is similar to another pixel combination in the upper right section of the Monterey image. It shows a second way in which the creation of empty ground corners during lobe iteration prevents unreasonable filling-in as figure. If the lobe corner (*) had, instead, been labeled empty figure, the large central rectangle would have filled as figure on iteration #5.

Fig. 19(c) serves as a reminder that a concave figure corner is an empty ground corner only if it is a lobe corner first. Creation of an empty ground corner after lobe initialization is a relatively rare but essential feature of the CONFIGR computation. The 40 random dot example (Fig. 2(d)) includes a pair of adjacent vertical filled rectangles similar to the two in Fig. 19(c).

5.5. CONFIGR component interactions

Figs. 20 and 21 further illustrate how CONFIGR lobe propagation, empty corner formation, and figure and ground filling-in work together to complete figures, starting with various image-figure pixel configurations.

In Fig. 20, for the U-shape, T-junction, and front porch examples, the final figure pixels are simply the original image-figure pixels. Both ground and wall corners help prevent spurious filling-in.

In Fig. 21(a), the central rectangle (size 5) completes as ground on lobe iteration #1. The upper and lower empty rectangles (size 2) then immediately fill as ground, because of their adjacency to filled-ground pixels. With a wider gap (Fig. 21(b)), CONFIGR produces two intersecting empty rectangles of size 6 on iteration #2. The horizontal one fills first as ground. Four wall corners are then relabeled as empty figure corners, producing the upper and lower empty rectangles (size 3). These fill as ground, by adjacency. With an even wider gap (Fig. 21(c)), two rectangles fill as ground at iteration #2. Iteration #3 produces four empty figure corners and eight wall corners. The central rectangle (size 7) fills as ground, again by adjacency.

Fig. 21(d) shows that, without the lower horizontal pixels, the upper bar in each image completes as figure. Since the central rectangle fills on a later iteration or not at all, adjacency does not cause the gap to fill as ground, as it did in (a)–(c). With the three-pixel gap, the two empty figure corners that emerge on iteration #3 fill as ground. An adjacent pair of empty figure corners emerge on iteration #4, but these do not span empty rectangles.

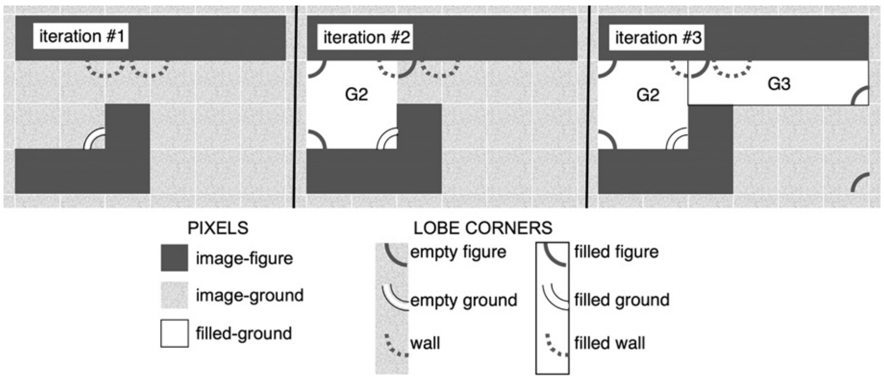

Fig. 16. Ground filling-in at iteration #2 causes one of the existing wall corners to become an empty figure corner. On iteration #3, this corner helps span an empty rectangle, which fills as ground.

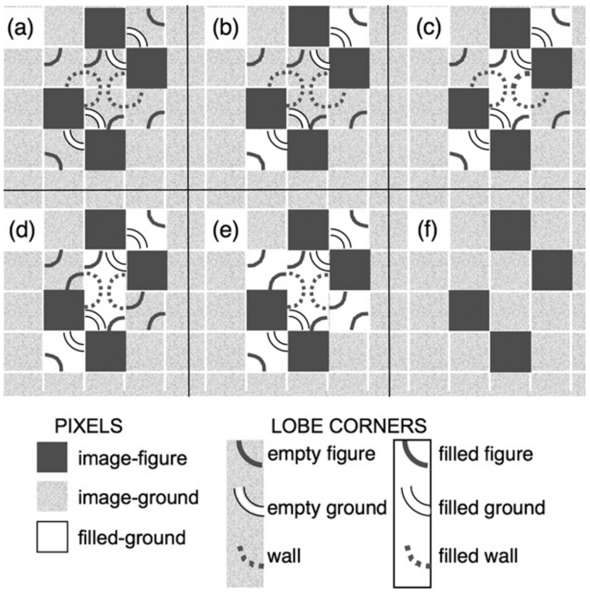

Fig. 17. Filling-in steps within lobe iteration #1. (a) Lobe initialization (iteration #0) produced four empty ground corners, and lobe iteration #1 produces six empty figure corners and six wall corners. These corners span three empty rectangles. (b) The CONFIGR algorithm's ground-filling loop first fills the two smallest empty rectangles, each of size 2. (c) The central rectangle (size 3) then fills as ground. (d) After four wall corners fill, the two remaining wall corners become empty figure corners, creating two more empty rectangles, each of size 2. (e) These two rectangles immediately fill, because they are smaller than the current loop's rectangle size. They fill as ground because they are adjacent to ground pixels. Subsequent lobe iterations produce no additional corners, empty rectangles, or filling-in for this image. (f) The final figure image is the same as the original image.

5.6. Local computations, global reorganization

The examples in Figs. 22 and 23 illustrate how a single pixel can qualitatively alter the global figure produced by CONFIGR. Each series (reading top to bottom, by column) shows the initial image-figure pixels and the series of lobe corners and filled rectangles that lead to the final figure.

In Fig. 22(a), the two image-figure pixels at the upper right form a self-contained unit, which blocks filling-in as figure elsewhere in the image. Removal of either of these pixels (Fig. 22(b), (c)) frees the remaining pixels to form a connected component. In both (b) and (c), iteration #1 produces an empty ground corner after flanking pixels fill as figure.

Fig. 23 further illustrates how a single image-figure pixel may reorganize the global figure percept. In Fig. 23(a), the right-hand pixel anchors a five-pixel grouping which becomes a self-contained unit, and so blocks filling-in of more figure pixels. If the five-pixel group were moved to the right, the two left-hand pixels would connect as a vertical figure component on iteration #1, but the left and right image sections would never connect. Fig. 23(b) shows how removing one image-figure pixel can reshape the global grouping. In Fig. 23(c), shifting the location of one of the image-figure pixels produces yet another final figure pattern. Like image (a), adding an extra image-figure pixel at the right of the bars would produce ground filling-in only. Similar filling-in patterns would persist for longer horizontal bars.

6. CONFIGR simulations: Roads, stars, and horses

The small-scale examples of Section 5 demonstrate the essential role of various mechanisms of the CONFIGR algorithm. Large-scale simulations now illustrate completion, connection, and union. Even in a large simulation, each computational detail can be checked by hand.

The implemented CONFIGR algorithm is available in Matlab and C++, from http://cns.bu.edu/techlab/CONFIGR/.

6.1. Roads: Completing contours at multiple spatial scales

The Monterey example (Fig. 1) illustrates how a multi-scale CONFIGR system, starting with noisy image-figure pixels, can complete contours in order to locate roads in a remotely sensed image. In large-scale simulations, rectangles filled as figure are indicated by a diagonal of grey pixels connecting the two corners lying between the two lobe corners that define the empty rectangle. In Fig. 14(a), for example, this diagonal would have connected the NE and SW corners of the filled rectangle F2. When two corner pairs complete one rectangle (e.g., Fig. 14(c)), the two intersecting diagonals are shown. When two filled rectangles are adjacent (e.g., Fig. 19(c)), two parallel diagonals appear. In multi-scale simulations, all diagonals are drawn at the fine scale.

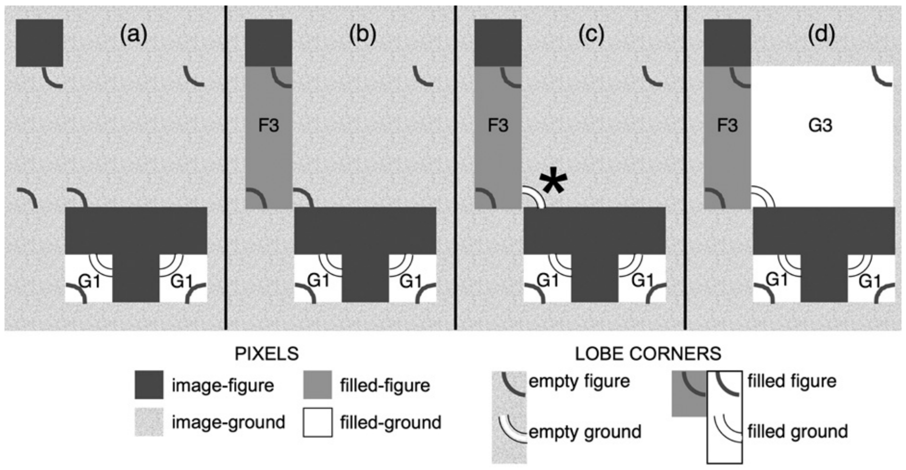

Fig. 18. Empty ground corner creation within an iteration, going from the smallest to the largest marked rectangle within the filling-in as figure loop of the CONFIGR algorithm. (a) Iteration #3 produces four empty figure corners, which span empty rectangles of size 4, 6, and 7. (b) The smallest empty rectangle fills first as figure (F3). (c) One of the lobe corners (*) is relabeled empty ground, since its defining perpendicular lobes are now flanked by figure pixels. (d) The remaining empty rectangle, which now contains an empty ground corner, immediately fills as ground.

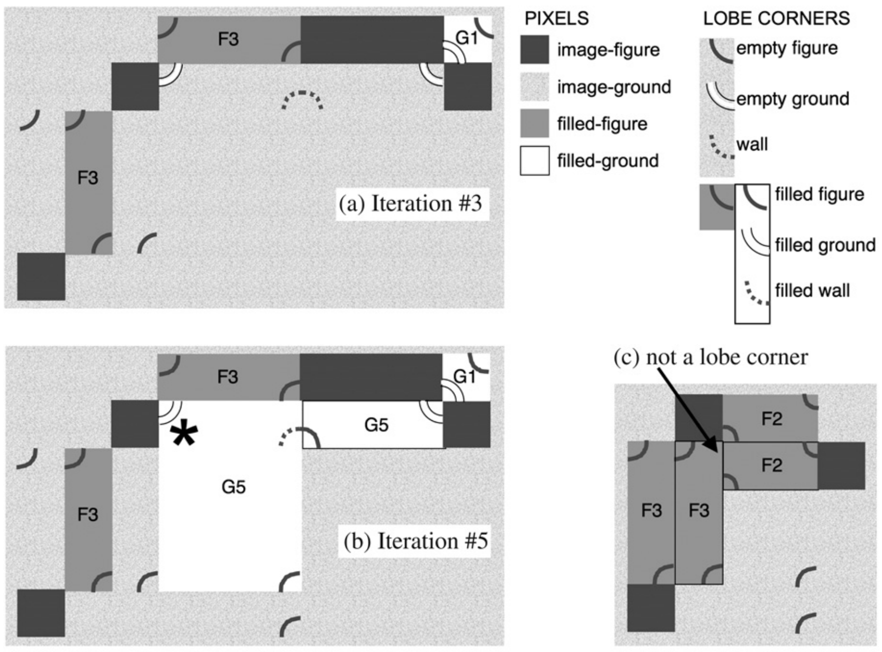

Fig. 19. Empty ground corners created after lobe initialization help prevent spurious filling-in as figure. (a) At iteration #3, two empty rectangles (F3) fill as figure. The same iteration produces the central wall corners and the lobe corner to their left, which is empty ground because it is flanked by two figure pixel. (b) Iteration #5 produces the central empty rectangle, which fills as ground. If its upper left-hand corner had been an empty figure corner, this large area would have filled as figure, even though it does not reasonably connect any image-figure pixels. After the large rectangle fills-in, the wall corner it contains becomes filled, and the wall corner to its right becomes an empty figure corner. The adjacent smaller rectangle then also fills as ground. Subsequent iterations produce no more lobe corners, empty rectangles, or filling-in. The final figure is an arc of twelve pixels connecting the six original image-figure pixels. (c) For this image, iteration #2 produces four empty figure corners, which span two empty rectangles that fill as figure (F2). Although the lower filled rectangle touches the corner of an image-figure pixel, the concave image corner that they surround is not a lobe corner, and hence it is not an empty ground corner. Iteration #3 produces six more empty figure corners, which span three empty rectangles. The two smallest (which together comprise the third) fill as figure (F3).

Fig. 24 shows a detail of CONFIGR filling-in at the fine and coarse spatial scales, for a fragment of the Monterey image located in and around the circle at the upper right. The fine and coarse scales exhibit complementary filling-in of figure (*road*) pixels. At the fine scale (Fig. 24(a)), image-figure details (concavity) in the central component cause filling-in as ground on iteration #1. This stops lobe propagation, and so inhibits connection to the road component located below and to its right.

The coarse scale (Fig. 24(b)) removes the concavity, with the central image-figure component reduced to a two-pixel rectangle. This piece connects to the component below and to its right at iteration #3. On the other hand, coarse-scale smoothing removes entirely the two-pixel image-figure rectangle seen above and to the left of the central component at the fine scale. At the fine scale, this small component anchors a road intersection, which CONFIGR connects to two

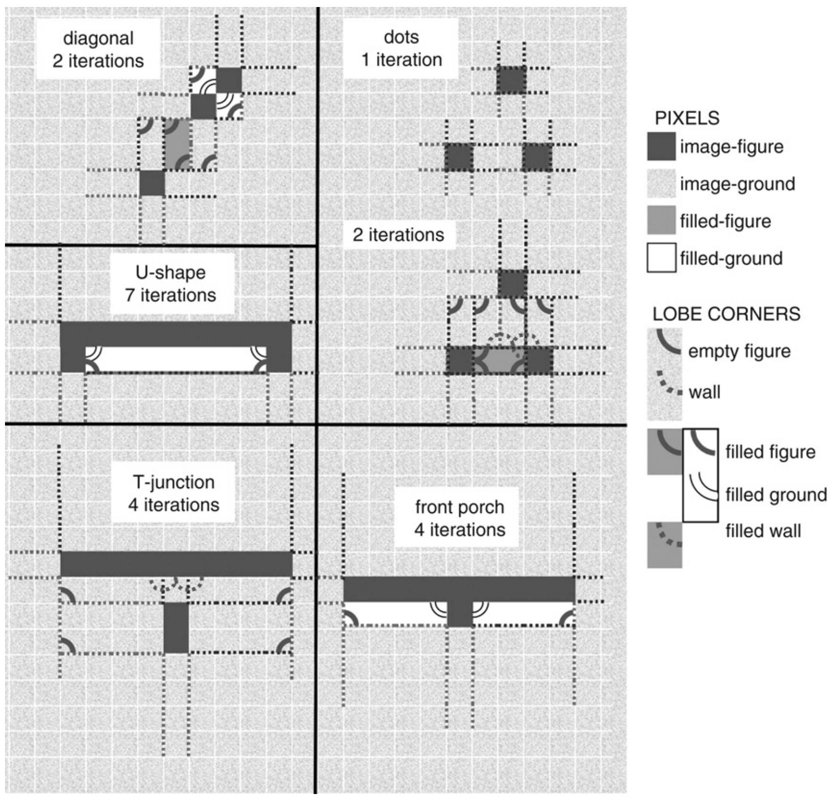

Fig. 20. Filling-in as ground and figure. Each image shows CONFIGR lobe propagation and corner formation. In each case, additional iterations would produce no more lobe corners or filling-in.

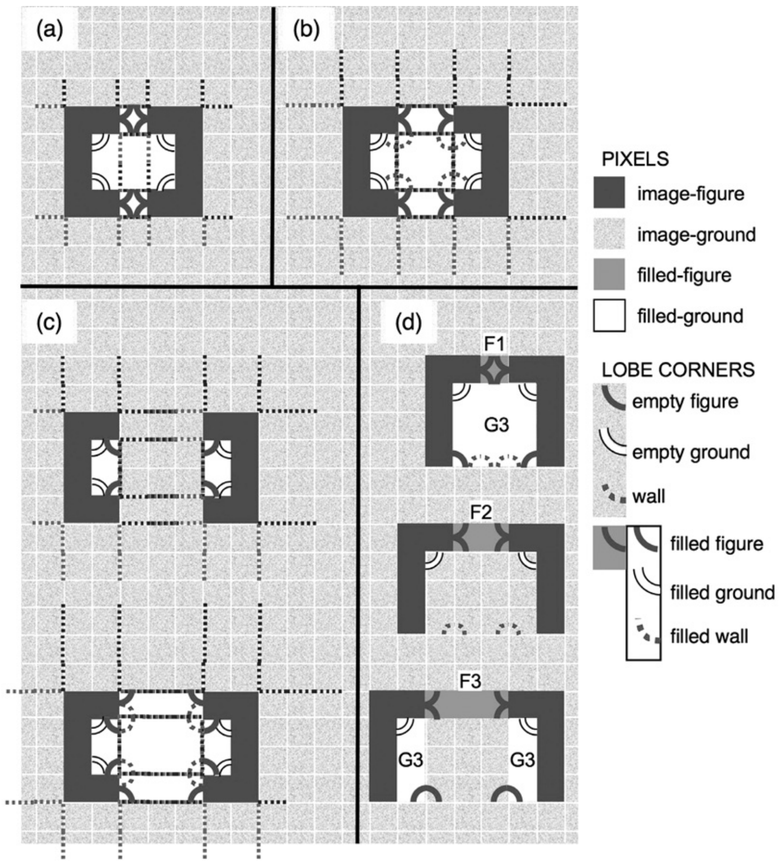

Fig. 21. Severed squares aligned with the vertical and horizontal lobe axes.

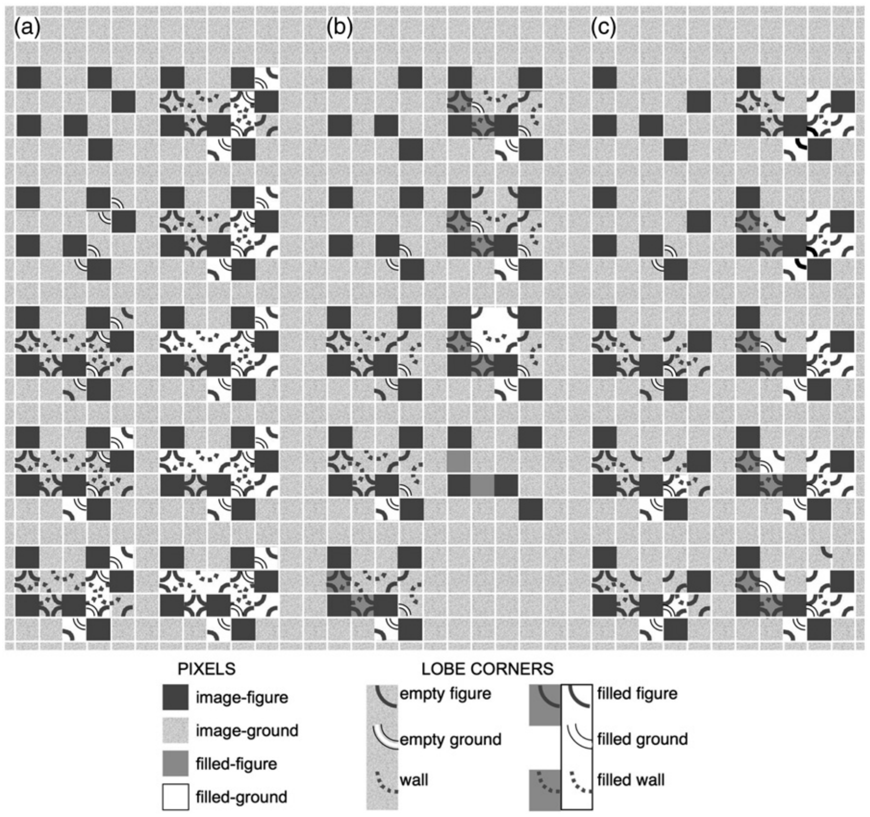

Fig. 22. Local perturbation produces global figure reorganization. (a) Within iteration #1, CONFIGR fills-in ground, but no additional figure, for a six-pixel image. Reading in ten steps (1–10) from top to bottom, then left to right by column, and starting as in the parallel bar example (Fig. 17), the original image (1) produces four empty ground corners at lobe initialization (2), then 11 empty figure corners and 11 wall corners at iteration #1 (3). Within the ground filling-in loop, two rectangles of size 2 fill as ground (4), then a rectangle of size 3 fills as ground (5). Filling of wall corners converts two adjacent wall corners to empty figure corners (6). The resulting empty rectangle of size 2 (lower right) immediately fills as ground, by adjacency to a filled-ground pixel (7). As rectangle size increases within the filling-in as ground loop, another empty rectangle (size 4) fills as ground (8). This in turn fills more wall corners and thus converts one more corner from wall to empty figure (9). The resulting empty rectangle of size 2 immediately fills as ground (10). Iteration #2 produces two more empty figure corners between the upper pixels, but these do not span empty rectangles. (b) Removing one image-figure pixel (upper right) changes the global configuration. Four image-figure pixels join as a single contour (5) during iteration #1 (3–6). Iteration #2 produces two empty figure corners (7) and an empty rectangle of size 4. The rectangle fills as ground because it contains the empty ground corner which was relabeled at iteration #1 (6), after two adjacent pixels filled as figure (5). (c) Removing the other image-figure pixel from the upper right produces the same connected figure on iteration #1 (3–9), but via different CONFIGR mechanisms. Iteration #3 (10) produces one empty figure corner, but no more filling-in.

nearby image-figure components. The algorithm connects road components at the lower right in both spatial scales.

Fig. 24(c) superimposes the original fine-scale image-figure pixels and the CONFIGR road completions from both the fine and the coarse spatial scales. Note that concavities in the road component in the upper left inhibit completion at both scales. The next coarser spatial scale removes the concavity, allowing this component to connect with the intersecting roads.

6.2. Stars: Connecting dots to form constellations

Fig. 2(c), (d) shows how CONFIGR joins 40 random dots, representing 0.1% a 200×200 pixel image, to form a connected figure component. The histogram in Fig. 25 shows the distribution of iteration numbers at which figure filling-in occurs, ranging from iteration #3 to iteration #38. Six dots connect to one other dot, 27 connect to two others, and seven connect to three others. In all, CONFIGR produces 41 figure connections, including one double vertical, like Fig. 19(c). This example has no initial figure concavities, and no filling-in as ground.

The random dot example in Fig. 2(c), (d) indicates that CONFIGR fill-in across arbitrary unobstructed distances while applying the same mechanisms for short-range connections. Although the final connected figure suggests optimization procedures such as those applied to the Traveling Salesman Problem, CONFIGR relies on local image-based computations, not the minimization of a global cost function.

Fig. 26 shows CONFIGR connection of 360 random dots, which include the previous 40 as a subset and which represent 0.9% of the square image. While most of the dots connect eventually, the complete figure includes four small self-contained components and one isolated dot (upper right). With this denser array of image-figure pixels, CONFIGR fills-in 30 rectangles as ground: 13 on iteration #1 and 17 more on iterations #2–14. Although the additional dots produce many more potential connections, filled-ground rectangles inhibit excessive filling-in as figure. CONFIGR produces figure connections on iterations #1–23, with the largest number (60) connecting on iteration #5. The number of figure connections (466) is more than 15 times the number of ground connections.

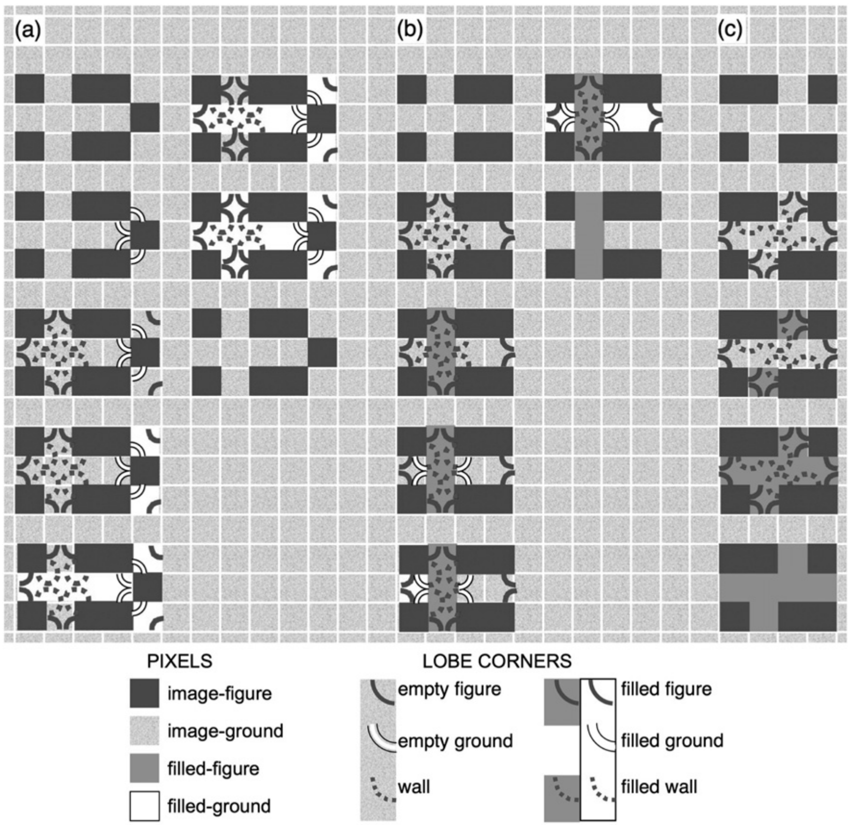

Fig. 23. Filling-in at iteration #1 for three related images. Each series starts with the image-figure pixels and ends with the complete set of figure pixels. (a) At the CONFIGR filling-in stage for a given lobe iteration, ground fills the eligible empty rectangles of all sizes, before any figure fills-in. Thus (5), the horizontal rectangle (size 5) fills before the intersecting vertical rectangle (size 4). Four relabeled wall corners (6) then produce two smaller empty rectangles (size 2), which fill as ground (7), by adjacency to ground. (b) Iteration #1 produces two intersecting empty rectangles of size 4 and size 5 (2). When the smaller one fills (3), the larger is no longer empty. After four wall corners are relabeled as empty ground (4), two new empty rectangles fill as ground (5, 6). (c) CONFIGR interprets image-figure pixels as pieces of a solid rectangle.

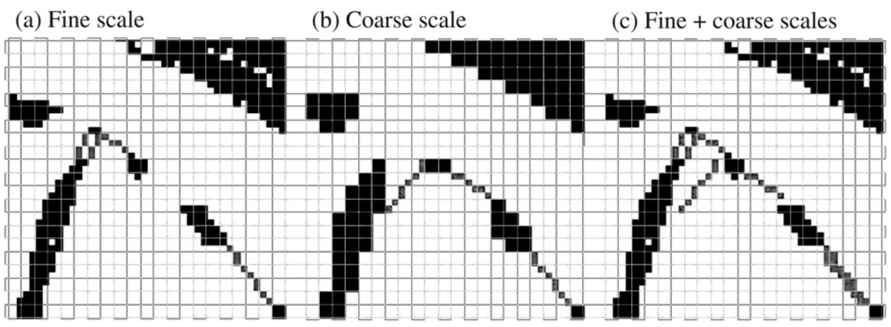

Fig. 24. Fragment from the upper right portion of the Monterey image (Fig. 1). Dark pixels are image-figure (*road*), and light pixels are diagonals of filled-figure rectangles. Coordinate lines are drawn at the coarse spatial scale, so each small box contains four fine-scale pixels.

With 3240 dots (8.1% of the square), image-figure crowding produces far more ground connections (1482) relative to the number of figure connections (2336). In Fig. 27, CONFIGR fills 650 rectangles as ground on iteration #1, 511 on iteration #2, and 321 more on iterations #3–7. Figure also fills-in quickly, with 1185 connections on iteration #1, 682 on iteration #2, 301 on iteration #3, 163 on iterations #4–6, and the final five on iterations #7–11.

In addition to completing its connections in just a few iterations, CONFIGR breaks this crowded random dot field into many self-contained components, or "constellations," which range in size from one image-figure dot, or "star," to hundreds. Connecting stars to form constellations is perhaps the most ancient and universal recorded example of filling-in. Fig. 28 demonstrates CONFIGR solutions to the problem of bringing coherence to the night sky. In each image, dark pixels represent stars of a constellation, and light pixels show CONFIGR connections.

In the constellations of Fig. 28, each set of stars is predefined and isolated for the CONFIGR algorithm. In a detail from the central portion of the 3240 dot simulation, Fig. 29 shows how CONFIGR can also self-organize its own constellations.

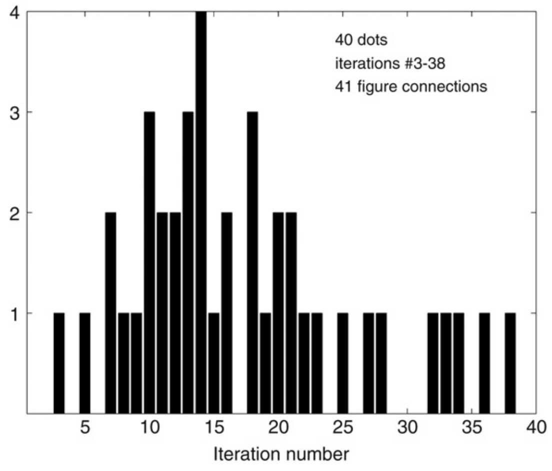

Fig. 25. Iteration numbers at which figure filling-in occurs for the 40 random dots of Fig. 2(c), (d).

Starting with a dense set of random dots, the system creates clusters of various sizes, leaving some dots isolated and incorporating others into large figures. The 60 × 120 pixel piece of the 200 × 200 square contains approximately 583 image-figure stars. CONFIGR clusters 61 of these stars into 23 small constellations (2–5 stars each) and clusters 62 stars into eight medium-sized constellations (7–17 stars each). Fourteen stars remain solo, and the remaining stars (approximately 446) are in nine large constellations, each extending beyond the rectangle's borders. As image-figure density increases, CONFIGR partitions the image into ever smaller self-contained clusters.

6.3. Horses: Unifying occluded objects

Magritte's painting *Blank Check* (1965) (http://www.planetperplex.com/img/magritte_blank_check.jpg) provides a vivid illustration of automatic segmentation of occluded objects. In that painting, a horse and rider are occluded by trees in a forest. The picture is globally impossible, but at first glance the horse appears to be a normally unified object. If the picture is manipulated to insert more occluding forest, the horse still seems unified—but only up to a point. If the separation is too great, the pieces of the horse never appear as a single object.

Dark image-figure pixels in Fig. 30(a) approximate the tan portions of the *Blank Check* horse. Light pixels show CONFIGR's filled-figure rectangles, which unify the complete object, though leaving open the space occupied by the rider. With the image-figure components further separated (Fig. 30(b)), CONFIGR still unites the occluded horse. If, however, the initial image-figure pieces are too far apart (Fig. 30(c)), CONFIGR does not span the gap.

As the sparse dot example illustrates (Fig. 2(c), (d)), CONFIGR does not set an *a priori* limit on the filling-in distance. Pixels can connect from arbitrary distances, if unimpeded. In Fig. 30(c), no image-figure or filled pixels block the central space. However, an empty rectangle by the front right leg (ellipse) fills as ground on iteration #8. In Fig. 30(b), the central empty rectangle forms on iteration #7. For any greater separation between the front and back of the horse, this rectangle would fill as ground, by adjacency to the filled-ground pixels by the leg.

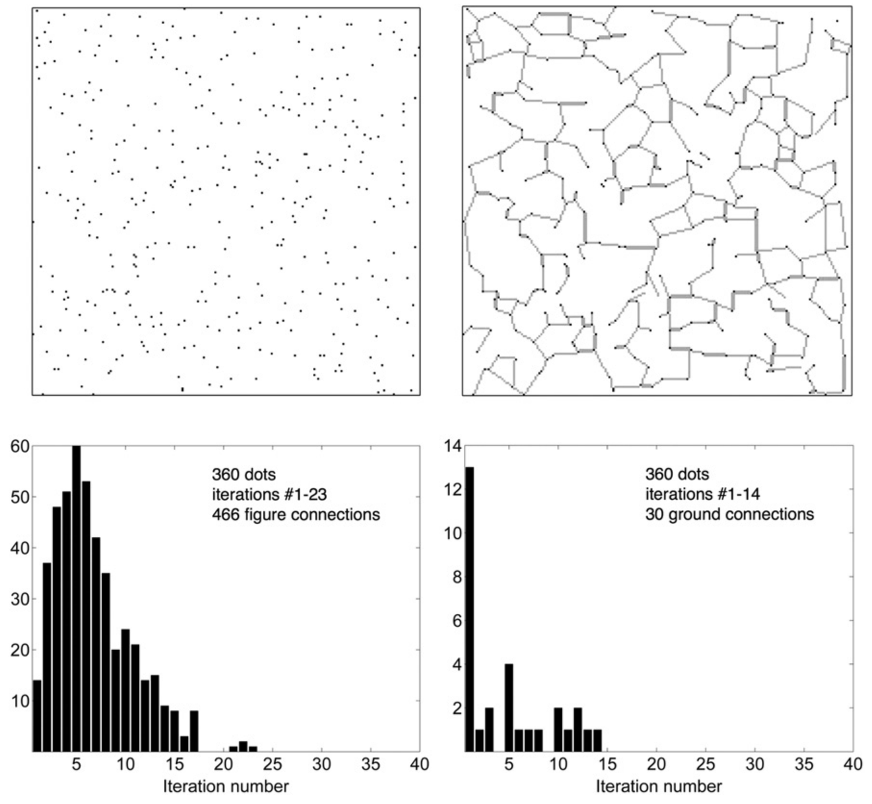

Fig. 26. CONFIGR connection of 360 random dots in a 200 × 200 pixel square, and iteration numbers at which figure and ground filling-in occurs.

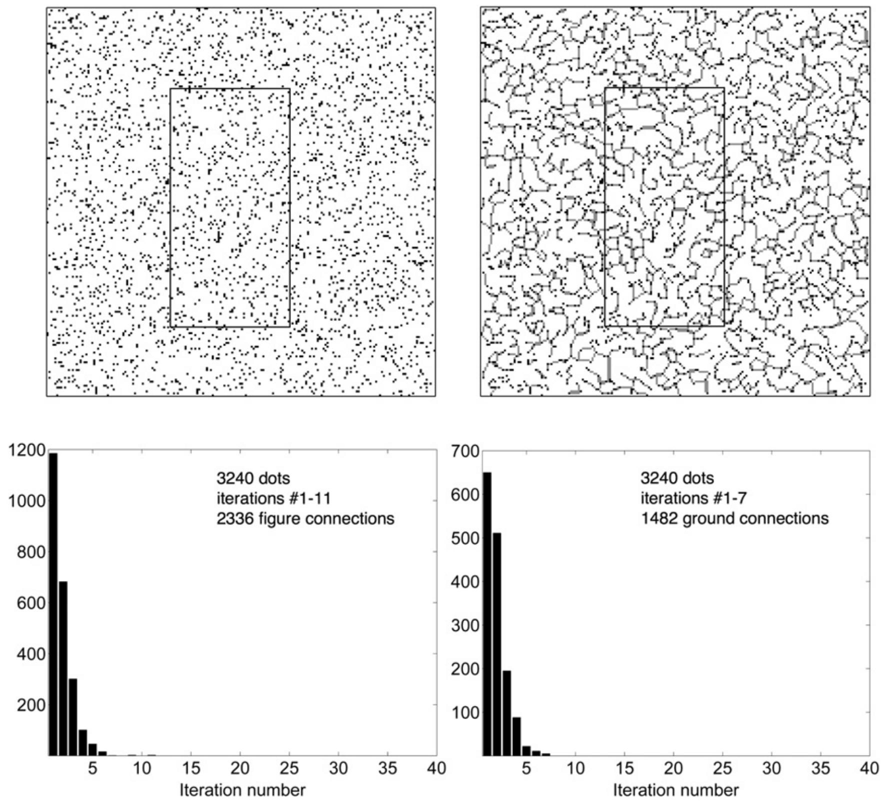

Fig. 27. CONFIGR connection of 3240 random dots in a 200 × 200 pixel square, and iteration numbers at which figure and ground filling-in occurs.

These images illustrate how the shape, detail, and scale of figure components contribute to the filling-in decisions. Note, too, that the image of Fig. 30(b), enlarged four-fold, would still fill-in as figure in the center, despite the wide separation, because the leg details would be similarly enlarged. This observation is reminiscent of the scale-invariant image completion extensively studied by Biederman (1987).

7. Images exactly aligned with lobe directions

Examples in this section highlight an anomaly of the CONFIGR algorithm, whereby vertical and horizontal image-figure elements that are aligned completely with the lobe directions fill-in differently from similar images that are tilted with respect to the lobes. Exactly vertical or horizontal bars rarely occur in natural images, and never in the Monterey or dots examples. The Monterey image, for instance (Fig. 1), includes many examples of parallel roads that fill appropriately as figure and ground. Nonetheless, the anomalous cases are notable, and point to directions for future system development: CONFIGR 2++ (Section 8).

7.1. Parallel bars

Given two parallel bars of image-figure pixels, CONFIGR typically fills-in figure and ground appropriately, as Fig. 31 shows for a series of bars that are tilted 45°. In these examples, CONFIGR completes the bars while employing a variety of mechanisms to block spurious filling-in as figure. In particular, filled-ground pixels "seal off" sides of bars, halting further lobe propagation from their interiors.

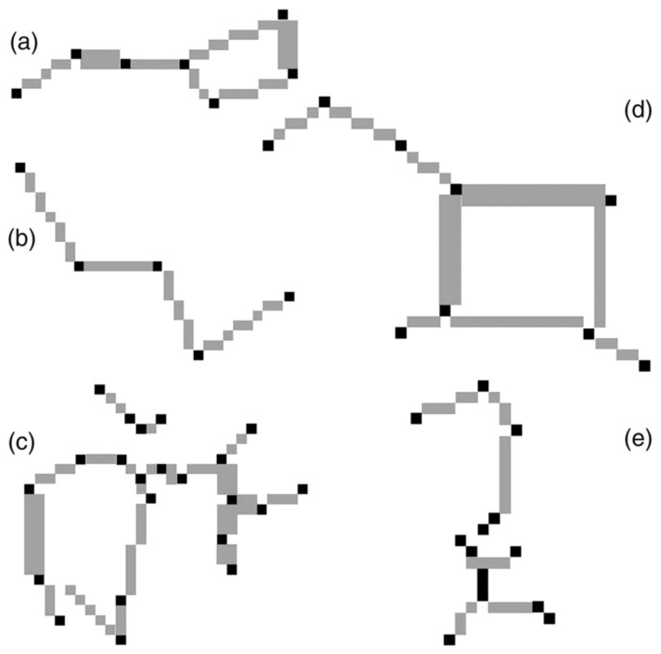

Fig. 28. CONFIGR constellations. (a) Big Dipper. (b) Cassiopeia. (c) Sagittarius. (d) Pegasus. (e) Orion.

Note that, in Fig. 31(b), iteration #1 produces three overlapping empty rectangles of size 4, requiring a tie-breaker in the algorithm. Either the two vertical (1 × 3) rectangles fill

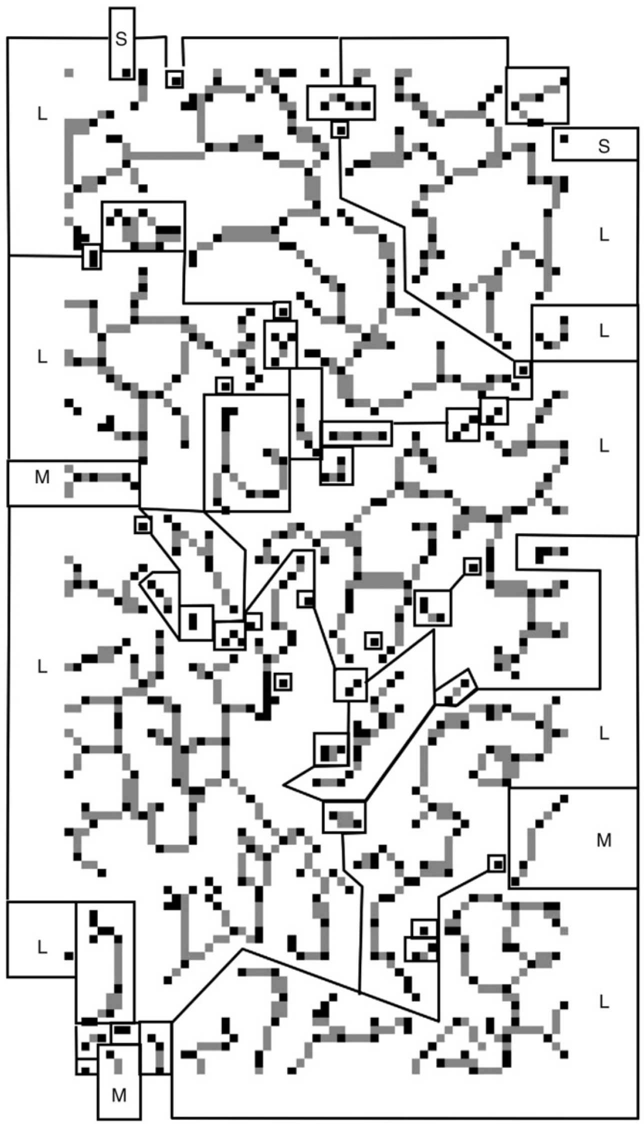

Fig. 29. CONFIGR self-organizes the randomly placed "stars," from the central rectangle of Fig. 27, into small (S), medium (M), and large (L) "constellations." Two small, three medium, and all nine large constellations extend beyond the borders of the rectangle.

as ground (as shown); or the central (2×2) rectangle fills first, converting two adjacent wall corners to empty figure corners and creating two new empty rectangles (1×1), which fill at once. In either case, all the central pixels fill as ground.

In contrast to the tilted parallel bar examples in Fig. 31, CONFIGR may fill the whole space between bars that are exactly horizontal or vertical. For short bars, this is basic long-range completion (Fig. 32(a)). However, when the image-figure bars are long (Fig. 32(b)), CONFIGR fills-in excessively between them, provided that the bars are exactly aligned with a lobe direction. The problem of central filled-figure pixels persists with missing image-figure pixels (Fig. 32(c)).

Fig. 33 further illustrates anomalous filling-in between bars that are exactly vertical or horizontal. If the bars are offset (Fig. 33(a)), filling-in may be even more extreme than when the bars are aligned (Fig. 32(b)). Missing pixels may produce a different filled-figure pattern, even if the gaps are filled at an

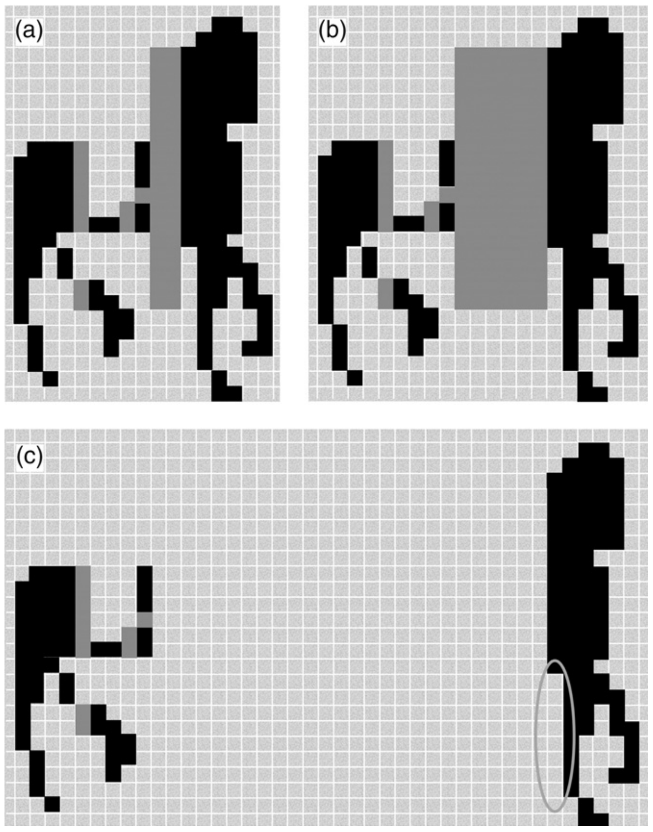

Fig. 30. CONFIGR unification of an occluded figure. (a) Image-figure pixels approximate the horse in Magritte's *Blank Check*. (b), (c) The scale of adjacent figure details determines the maximum separation for filling-in.

early iteration. In Fig. 33(b), pixels missing from the bars fill as figure on iteration #1. On iteration #5, the large empty rectangle of Fig. 33(a) now contains a smaller one, which fills as figure (F5). Adjacent wall corners then become empty ground corners, creating two new empty rectangles, which fill immediately as ground (G5).

When parallel bars are tilted, CONFIGR ground filling-in blocks spurious figure completions. Fig. 34 shows an image-figure pixel configuration which is similar to the ones in the upper left portion of the Monterey image (Fig. 1(a)). When the central rectangle (G4) fills, ground adjacency prevents filled-figure cross-talk between digitized bars that are approximately parallel and tilted with respect to the lobe directions. Here, all the empty ground corners are created at lobe initialization.

Bars that are just slightly tilted provide an intermediate case between the strictly horizontal bars of Figs. 32 and 33, and more tilted bars of Figs. 31 and 34. Fig. 35(a) shows a filled-figure connection between slightly tilted bars. While this filled-figure cross-talk (F2) is not as extreme as the complete filling-in of the exactly horizontal case, the extra connection may nonetheless be spurious. Fig. 35(b) shows appropriate filling-in generated by the same local image-figure pixel configuration.

7.2. Occluded squares

Fig. 21(a) showed a severed square aligned in the vertical and horizontal directions. CONFIGR does not connect the

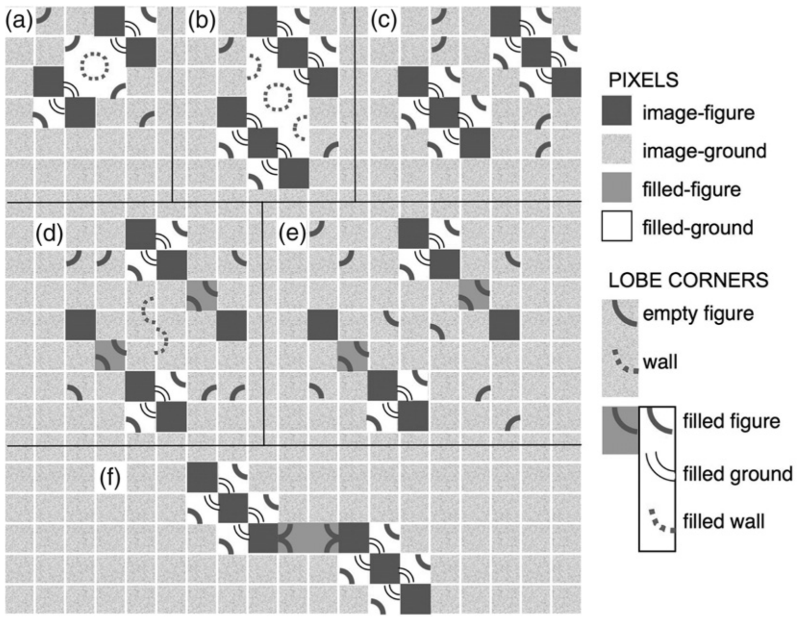

Fig. 31. Tilted parallel bars. (a) Bars are farther apart than in Fig. 17, and different CONFIGR algorithm elements block spurious figure filling-in. Here, three empty rectangles fill as ground on iteration #1. Iteration #2 produces two more empty figure corners, but no additional empty rectangles or filling-in. (b) With longer bars, iteration #1 again fills all empty rectangles as ground, and iteration #2 completes empty corner production. (c) With the bars even farther apart, filled-ground pixels stop most lobe propagation at iteration #1, without wall corners. At iteration #2, bar ends generate four additional empty corners, but no empty rectangles. (d) The gap in each bar fills as figure on iteration #1. Missing pixels imply that filled-ground pixels fail to block lobe propagation from the interior of each bar, but iteration #2 produces only wall corners in the center, as well as six new empty figure corners, and no additional filling-in. (e) With the bars-with-gaps farther apart, iteration #2 produces two central empty figure corners, but these span no empty rectangles. (f) When image-figure bars are sufficiently offset, CONFIGR may treat them as misaligned segments of a single contour, here connecting the end of one bar with the beginning of the other on iteration #2.

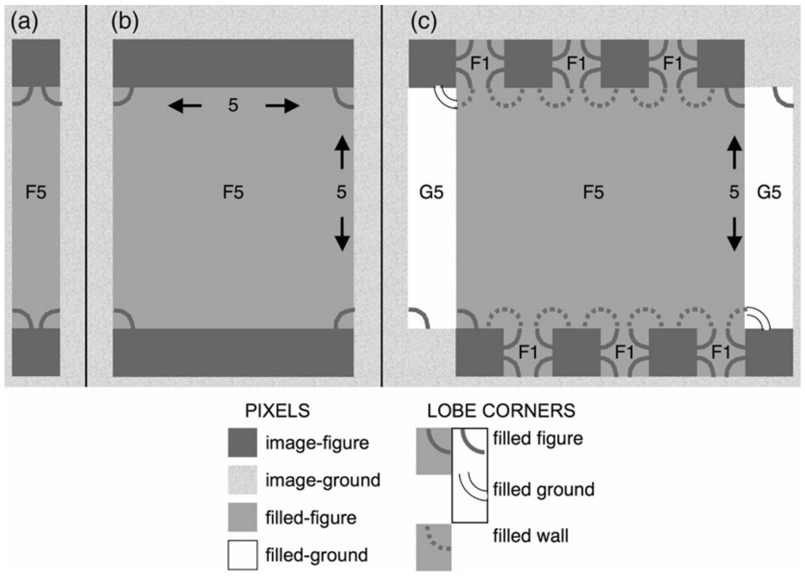

Fig. 32. Mechanisms that allow CONFIGR to connect pixel dots at a distance (a) also cause filling as figure between longer bars that are exactly horizontal or vertical (b). In (c), iteration #5 produces two rows of six wall corner pairs, plus four empty figure corners. After the smallest empty rectangle fills as figure (F5), the upper left and lower right wall corners become empty ground. The empty ground corners help span empty rectangles, which immediately fill as ground (G5).

edges of this image as figure, because the central rectangle fills first as ground. When the severed square is tilted, however (Fig. 36), its center does not fill as ground, so the edges are free to complete as figure. Iteration #2 produces four central wall corners, and two more empty figure corners, but no additional empty rectangles.

Similarly, Fig. 37 shows how CONFIGR unites digitized Kaniza squares tilted at various angles. However, when the

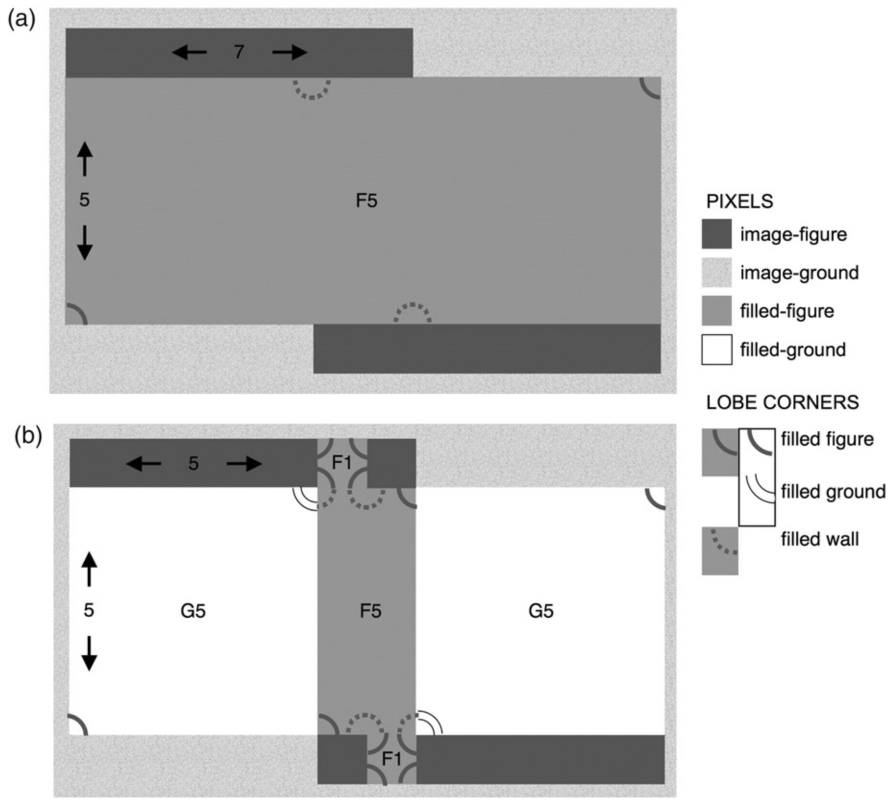

Fig. 33. Offset horizontal bars of length 7 fill-in between each other on iteration #5.

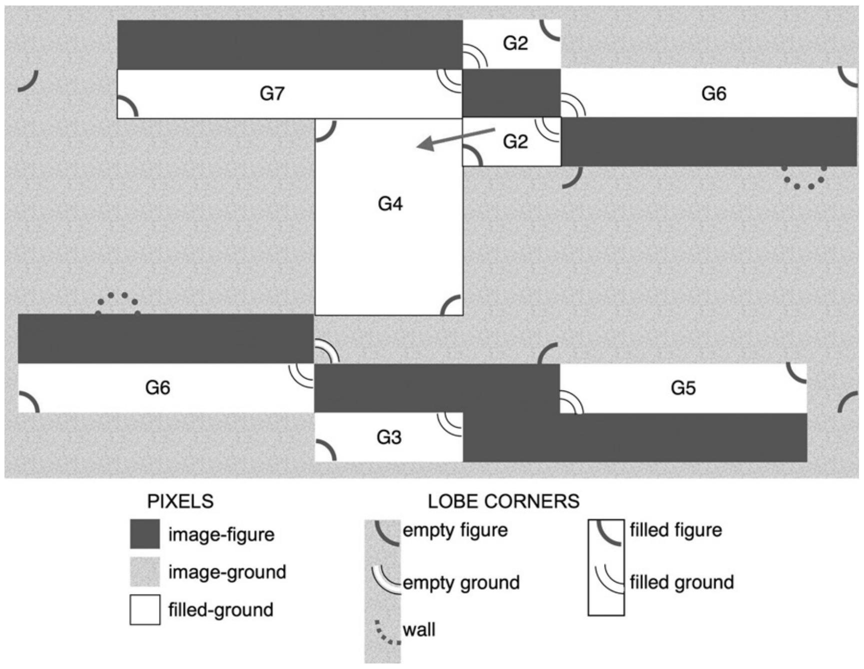

Fig. 34. For bars that are tilted and approximately parallel, CONFIGR blocks spurious filling-in as figure by filling empty rectangles as ground. The final figure is the same as in the original image.

square is exactly vertical and horizontal, the central square fills as ground, causing its edges also to fill as ground on the same iteration, by adjacency.

These examples further illustrate the special nature of images that are exactly aligned with the vertical and horizontal lobe directions.

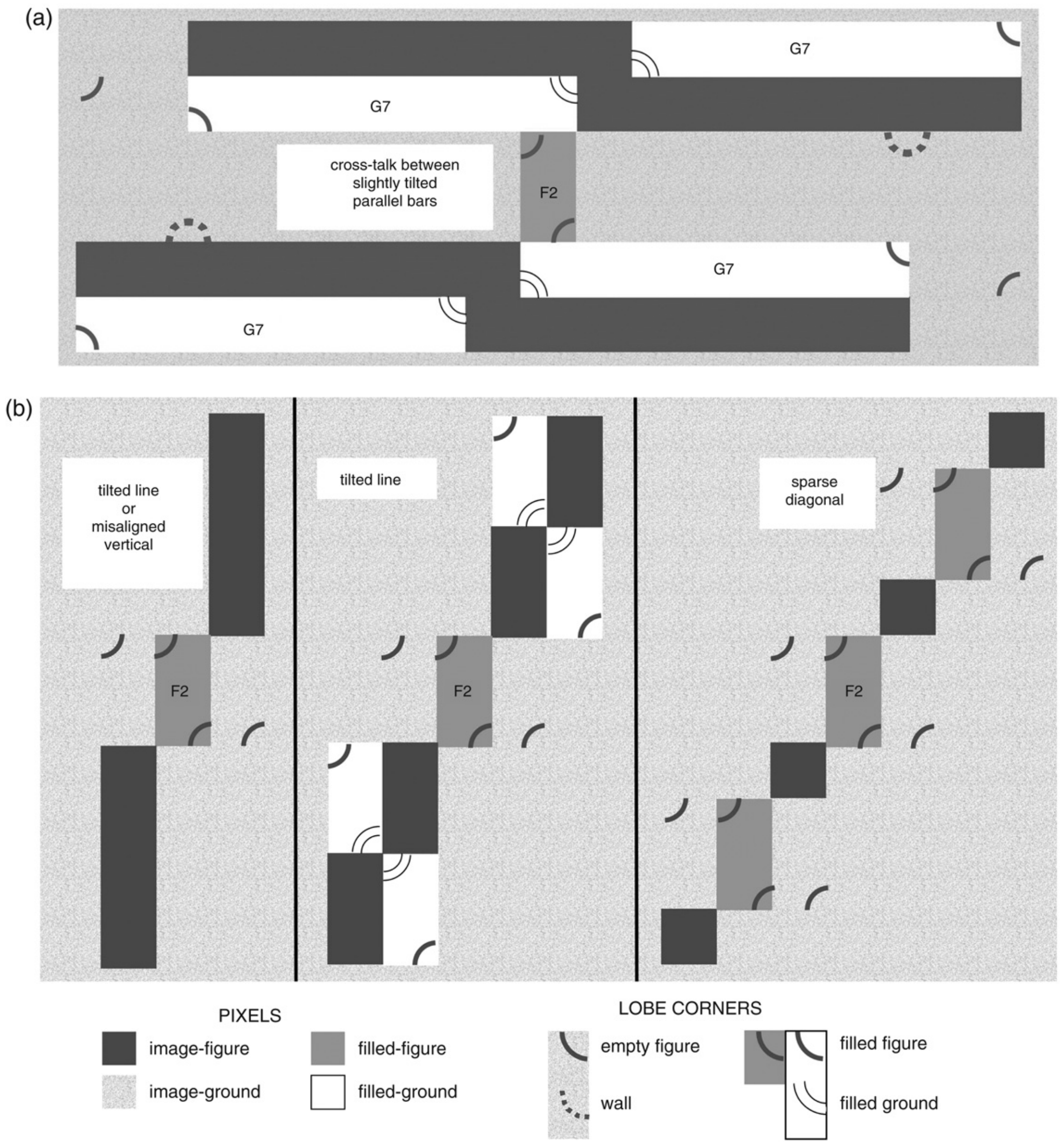

Fig. 35. The possibility of CONFIGR cross-talk is unavoidable at some scales, without another rule. (a) Slightly tilted parallel bars. (b) Correct filling-in as figure.

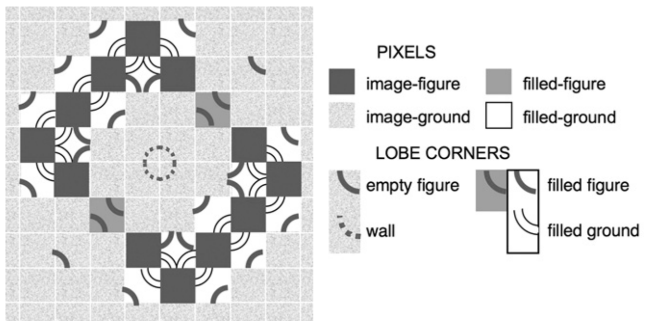

Fig. 36. CONFIGR completion of a severed image-figure square, tilted 45°.

8. Future directions: CONFIGR 2++

Image examples in Sections 5–7 are designed to highlight CONFIGR limitations as well as model capabilities. These examples point toward new design goals for the next generation. This section outlines questions of interest and possible design principles for future model development.

More lobe directions: Although CONFIGR can connect image-figure components in any direction, the model computes locally only in the horizontal and the vertical. These two directions derive from the sides of the square image pixels. A natural question for CONFIGR 2 is:

How can a vision system with more than two computational directions realize CONFIGR properties?

Images aligned with lobe directions: Examples in Section 7 illustrate the special nature of CONFIGR processing of images that are exactly vertical or horizontal. These figures suggest the more general question:

For any vision system based on a finite set of orientations, how do responses to image elements aligned with these orientations differ from responses to image elements placed at other orientations?

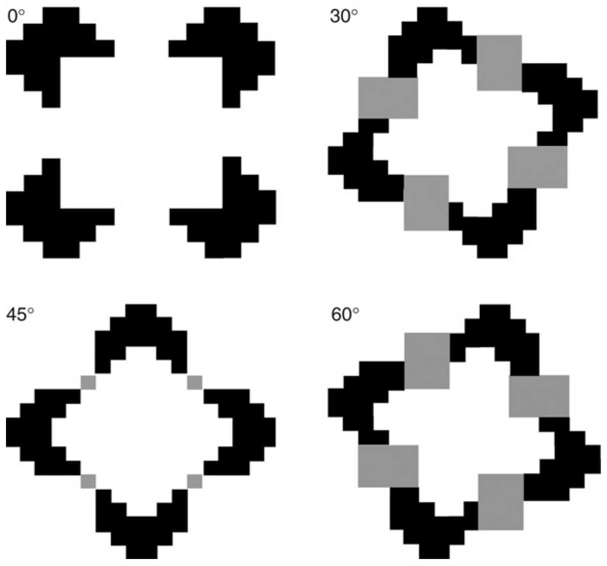

Fig. 37. Digitized Kaniza square tilted 0°, 30°, 45°, and 60°. CONFIGR completes the edges of the central square, except when the sides align exactly with the vertical and horizontal lobe directions.

Perturbing the image: Images that are strictly horizontal or vertical rarely occur in complex natural images such as Monterey (Fig. 1). Such images are, however, commonly constructed, e.g., for psychophysics experiments. To test robust CONFIGR predictions for these examples, the image angle might be randomly perturbed. The likelihood of an anomalous figure is then very small (Figs. 36 and 37).

Multiple spatial scales: The Monterey simulations (Figs. 1 and 24) apply a multi-scale CONFIGR strategy of adding the connections from all spatial scales to the original image figure to obtain the final figure. Although this multi-scale combination works well for the Monterey example, other examples might require alternative multi-scale procedures.

Global evidence: CONFIGR lobe computations are strictly local. The horizontal bar examples in Fig. 32 are designed to show that local evidence alone may be insufficient for some cases. CONFIGR 2 might address this issue by taking into account more global properties of the image figure. In such a system, short bars (Fig. 32(a)) would connect as figure, but the space between longer parallel bars (Fig. 32(b), (c)) would not fill. Similarly, CONFIGR 2 might permit the local vertical connections in Fig. 35(b), which are consistent with more global vertical image elements, but inhibit the spurious connection in Fig. 35(a), which is inconsistent with more global horizontal image elements. Multi-scale combinations might also help solve this problem.

Image boundary effects: In large-scale simulations, CONFIGR occasionally produces anomalies near image borders. "Connections" from only one image-figure pixel appear near the SW corner of the fine-scale Monterey example (Fig. 1(a)) and near the SE corner of the 360 dot example (Fig. 26). Though rare, these instances point to possible future system modifications, and suggest caution when interpreting results near image borders. Ideally, an image of interest should include a fringe (e.g., Fig. 27) which the final computation (Fig. 29) can employ to eliminate boundary effects.

Neural substrates and psychophysical predictions: Many elements of the CONFIGR system are new to the modeling literature. The recognition-vision-recognition sequence (Section 1) suggests that CONFIGR computations might occur in visual area V4. CONFIGR filling-in examples, particularly the role of concavities, suggest psychophysical experiments. Cognitive and neural considerations (Pessoa & De Weerd, 2003), as well as new computational challenges of large-scale simulations, will guide the design of CONFIGR 2 and beyond.

Acknowledgements

This work was supported by research grants from the Air Force Office of Scientific Research (AFOSR F49620-01-1-0423), the National Geospatial-Intelligence Agency (NMA 201-01-1-2016), the National Science Foundation (NSF SBE-0354378), and the Office of Naval Research (ONR N00014-01-1-0624).

References

Ahissar, M., & Hochstein, S. (2004). The reverse hierarchy theory of visual perceptual learning. *Trends in Cognitive Sciences*, *8*(10), 457–464.

Biederman, I. (1987). Recognition-by-components: A theory of human image understanding. *Psychological Review*, *94*(2), 115–147.

Candes, E., Romberg, J., & Tao, T. (2006). Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, *59*, 1207–1223.

Carpenter, G. A. Default ARTMAP. In *Proceedings of the international joint conference on neural networks* (pp. 1396–1401).

Carpenter, G. A., Grossberg, S., Markuzon, N., Reynolds, J. H., & Rosen, D. B. (1992). Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, *3*(5), 698–713.

Cohen, M. A., & Grossberg, S. (1984). Neural dynamics of brightness perception: Features, boundaries, diffusion, and resonance. *Perception and Psychophysics*, *36*(5), 428–456.

Field, D. J., Hayes, A., & Hess, Robert F. (1993). Contour integration by the human visual system: Evidence for a local "association field". *Vision Research*, *33*(2), 173–193.

Gove, A., Grossberg, S., & Mingolla, E. (1995). Brightness perception, illusory contours, and corticogeniculate feedback. *Visual Neuroscience*, *12*, 1027–1052.

Grossberg, S., & Mingolla, E. (1985a). Neural dynamics of perceptual grouping: Textures, boundaries, and emergent segmentations. *Perception and Psychophysics*, *38*(2), 141–171.

Grossberg, S., & Mingolla, E. (1985b). Neural dynamics of form perception: Boundary completion, illusory figures, and neon color spreading. *Psychological Review*, *92*(2), 173–211.

Grossberg, S., & Mingolla, E. (1987). Neural dynamics of surface perception: Boundary webs, illuminants, and shape-from-shading. *Computer Vision, Graphics, and Image Processing*, *37*(1), 116–165.

Grossberg, S., & Swaminathan, G. (2004). A laminar cortical model for 3D perception of slanted and curved surfaces and of 2D images: Development, attention, and bistability. *Vision Research*, *44*, 1147–1187.

Hansen, T., & Neumann, H. (2004). Neural mechanisms for the robust representation of junctions. *Neural Computation*, *16*(5), 1013–1037.

Kellman, P. J., & Shipley, T. (1991). A theory of visual interpolation in object perception. *Cognitive Psychology*, *23*, 141–221.

Mingolla, E., Ross, W., & Grossberg, S. (1999). A neural network for enhancing boundaries and surfaces in synthetic aperture radar images. *Neural Networks*, *12*, 499–511.

Parsons, O., & Carpenter, G. A. (2003). ARTMAP neural networks for information fusion and data mining: Map production and target recognition methodologies. *Neural Networks*, *16*, 1075–1089.

Pasupathy, A., & Connor, C. E. (1999). Responses to contour features in macaque area V4. *Journal of Neurophysiology*, *82*, 2490–2502.

Pessoa, L., & De Weerd, Peter (2003). *Filling-in: From perceptual completion to cortical reorganization*. New York: Oxford University Press.

Takhar, D., Laska, J. N., Wakin, M. B., Duarte, M. F., Baron, D., Sarvotham, S., et al. (2006). A new compressive imaging camera architecture using optical-domain compression. In *Proc. computational imaging IV*. SPIE, http://www.dsp.ece.rice.edu/cs/cscam-SPIEJan06.pdf.

Von der Heydt, R., Peterhans, E., & Baumgartner, G. (1984). Illusory contours and cortical neuron responses. *Science*, *224*(4654), 1260–1262.