Annotating floor plans using deformable polygons

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:26506438">http://nrs.harvard.edu/urn-3:HUL.InstRepos:26506438</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
Annotating Floor Plans Using Deformable Polygons

Kathy Ryall
Harvard University

Joe Marks
Digital Equipment Corporation

Murray Mazer
Digital Equipment Corporation

Stuart Shieber
Harvard University

December 16, 1993

Abstract

The ability to recognize regions in a bitmap image has applications in various areas, from document recognition of scanned building floor plans to processing of scanned forms. We consider the use of deformable polygons for delineating partially or fully bounded regions of a scanned bitmap that depicts a building floor plan. We discuss a semi-automated interactive system, in which a user positions a seed polygon in an area of interest in the image. The computer then expands and deforms the polygon in an attempt to minimize an energy function that is defined so that configurations with minimum energy tend to match the subjective boundaries of regions in the image. When the deformation process is completed, the user may edit the deformed polygon to make it conform more closely to the desired region. In contrast to area-filling techniques for delineating areal regions of images, our approach works robustly for partially bounded regions.

Keywords: 2-D graphics, drawing software, region finding, deformable templates, building geography.

1 Introduction

Location-aware technology and ubiquitous computing are major new trends in applications computing (Weiser, 1991; Wellner, Mackay, and Gold, 1993). One premise of this work is that the ability to track the movement of people and artifacts (e.g., computers, machinery, vehicles, etc.) will make possible new computer applications for better managing the home and workplace. Most current research in this area...
is focused on developing the hardware to make it all possible (e.g., the Digital-Olivetti Active Badge system (Want and Hopper, 1992; Want et al., 1992; Fishman and Mazer, 1992)). Less attention has been paid to software. Because location-aware computing depends upon the specific environmental or physical context in which it is employed, it requires software to support the input, representation, processing, and visualization of such physical settings. In this paper we focus on the input problem: our broad goal is to develop software techniques to facilitate the input of information about building geographies that is needed for location-aware computing.

In general, geographic information for buildings is available only from hard-copy floor plans.\footnote{Another potential source of geographic data for buildings is architectural CAD systems. However, this data source is less attractive than it might seem at first blush because data are available for recently constructed or renovated buildings only, and various systems use different data formats and representations.} (A sample bitmap scanned from a floor plan is shown in Figure 8.) Thus the data is not in symbolic form; it must be interpreted visually, a task that is easy for people but not for computers. The information required for location-aware computing is region data: the topology and geometry of “people” regions (e.g., offices, lobbies, or closets) and “computer” regions (e.g., the areas scanned by sensing devices). To extract this data from a scanned floor plan requires annotating the floor plan to delineate the relevant regions. The problem of region delineation is the one we address in this paper.

1.1 Existing Approaches

Several software tools exist today that can facilitate the delineation of areal regions on a floor plan. They all utilize one or more of the following four techniques:

1. Freehand drawing: The user redraws the floor plan from scratch with a mouse or digitizing tablet, delineating regions as part of the drawing process.

2. Tracing: The user traces out regions on the floor plan by hand, using a mouse or digitizing tablet.

3. Area filling: The user selects points on the floor plan image, and the computer fills in the fully enclosed areas surrounding the points. (The method is often called “flood filling”.)

4. Template fitting: The user positions and sizes predefined templates (usually rectangles) to cover the desired regions approximately.

All of these techniques have drawbacks. Freehand drawing is time-consuming, tedious, and error prone. Tracing is equally inefficient, though more accurate. Area filling would appear at first sight to be fast and accurate, but it only works for fully
enclosed regions: if the user wants to specify a region that is partially bounded —
the boundaries of the region might have been corrupted during the scanning process
(a frequent occurrence), or the region might be bounded in part by subjective con-
tours not depicted in the image — the filling technique will not work because the
filling process will “leak out” to the rest of the image through missing boundaries
or holes in existing boundaries. Finally, template fitting, though fast, is inherently
inaccurate.

1.2 Paper Overview

The area-filling approach has the attractive property of being semi-automatic: The
user chooses the point from which to start the filling process, which is then per-
formed automatically. However, the technique is brittle, as it relies on the unrealistic
assumption that subjective boundaries in an image (the boundaries of image areas
that are subjectively perceived as forming distinct regions) coincide with actual
closed pixel contours in the image. We propose an alternative method that utilizes
a similar division of labor between user and computer but relaxes this assumption.
The user specifies a location at which to place a deformable polygon; the polygon
then deforms automatically to conform to boundaries in the scanned image; when
the polygon achieves a static conformation, it is proposed by the system as a can-
didate region. The user is free to accept the region geometry as proposed by the
system, or edit it as appropriate. Relative to area filling, deformable polygons have
the advantage that they are robust with respect to corrupted or missing boundaries
in the image, their exact behavior can be customized through selection of parameter
settings, and they can take other application-specific problem characteristics into
account.

In the remainder of the paper we describe our overall approach (Section 2), the
algorithm for deforming polygons (Section 3), the system prototype we have built
to test our algorithm (Section 4), and some illustrative examples (Section 5).

2 Overall System Architecture

The approach to region identification that we take is a semi-automated method in
which the user interacts through a simple graphical interface. At a minimum, any
method for identification of a single region requires a user to indicate the rough
region of interest and to specify aspects of the region that cannot be determined a
priori: general notions of scale, complexity, and so forth. The method we propose
requires just this much input on the part of the user, and furthermore allows for
post-editing to correct any errors the system makes. In summary, the interaction is
schematically structured as a three-step process:
1. The user specifies values for a small set of parameters that roughly characterize the size and complexity of the region, and indicates the region of interest by clicking at a point in the region.

2. The system attempts to find the subjective boundaries of the region near the selected point.

3. The user may manually edit the boundaries found.

The novel contribution of our approach is the method by which the system finds the subjective boundaries of the region. As discussed above, the standard method, based on filling, has the problem that subjective boundaries that are not marked with explicit lines in the image are overrun so that neighboring regions are invaded. Our approach attempts to better characterize the notion of a subjective region by encapsulating the notion “quality of subjective region boundary” in a single function and by viewing the region-identification problem as the problem of optimizing this function.

3 Finding Subjective Boundaries

The approach is best seen by analogy. Suppose we are given a drawing of the two-room building given in Figure 1a. Note the door between the rooms. Because of this door, a filling algorithm started from any point would find a single region (as in Figure 1b).

Now imagine a surface defined so that the height of the surface at each black pixel in the image is quite high (say 10 feet) and at each interior pixel the surface is as many inches high as the pixel’s distance to the nearest black pixel, so that the surface is lowest just adjacent to the black pixels. A topographic map of such a surface is given in Figure 1c. Objects on such a surface tend to move downhill so as to locally minimize their potential energy. Consequently, if an infinitely stretchable rope were looped around the peak of the surface in the upper room (as in Figure 1d), it would flow towards the boundary, but would not invade the second room, as the height begins increasing again towards the lower peak at that point. The final settled position of the rope, also shown in Figure 1d, thus defines the subjective boundary of the upper room well. This example provides evidence that an approach based on energy minimization might be appropriate for characterizing subjective boundaries better than the simple filling technique.

Our method is based on just this metaphor of energy minimization, although the definition of the energy function is quite a bit more complicated. The definition of the energy function and the motivation for its design are the topic of the remainder of this section. As this is work in progress, we also comment on problems that remain in our approach, and untried alternatives that may be useful.
Figure 1: (a) A simple two-room drawing. (b) A filling algorithm started in the upper room would traverse the subjective boundary of the door, invading the second room as shown. (c) By defining an appropriate surface, where the black pixels are high, and the white pixels are assigned a height proportionate to the distance to the nearest black pixel (a “topographic map” of which is seen in the diagram) a better model of subjective boundaries can be generated. (d) An infinitely stretchable rope, looped around the surface’s peak in the upper room, flows downhill to the edges of the room, settling at the subjective boundary of the upper room.
Our system uses deformable polygons to simulate the deformation of the rope as it moves to lower energy configurations. Deformable polygons are similar in nature to snakes (Kass, Witkin, and Terzopoulos, 1988) and deformable templates (Yuille, Hallinan, and Cohen, 1992), specialized to the particularities of the task at hand, although snakes are a much more general and computationally expensive method whereas deformable templates are specialized to a different domain. The height of the surface is replaced by an energy assigned to each pixel, the rope is replaced by a polygon, and the deformation process is an iterative local minimization of the total energy of the polygon.

3.1 The Basic Energy Field

As described earlier, the surface that defines the energy at each pixel varies with the type of pixel. We distinguish three types of pixels:

- **Contour pixels** are the black pixels in the image, which are taken to represent overt region boundaries. Contour pixels are given a high positive energy value.

- **Edge pixels** are white pixels that are immediately adjacent to a contour pixel. These tend to form the bulk of a subjective region boundary. Edge pixels are given a small negative energy value.

- **Interior pixels** are all other white pixels. Informally speaking, their utility as part of a subjective boundary depends on their distance to the nearest boundary. Consequently, their energy is taken to be a linear function of their distance to the nearest contour pixel.

Figure 2a depicts the energies of a sequence of pixels taken from a cross-section of a room.

3.2 The Basic Energy Function

As briefly described above, the energy field over the image is used to determine an energy for a polygon placed in the image. The polygon is then deformed in order to minimize this energy. This section describes the basic energy function, a function that assigns to any given polygon configuration an energy value. Later sections describe, *inter alia*, modifications of this basic energy function that provide for finer control over the deformation process.

The basic energy function is given simply by the total energy of the polygon perimeter, that is, a line integral of the energy field over the boundary of the polygon. This polygon perimeter energy is an indication of how well the edges of the polygon conform to boundaries in the image.
Figure 2: (a) The energy surface defined at a cross-section of pixels. Contour pixels are given a high energy, edge pixels a small negative energy, and interior pixels an energy proportional to the distance to the nearest contour pixel. (b-d) The rising energy field through time. All non-contour energies start out negative (b), and rise until the edge pixels reach their ceiling height (c). Interior pixel energies continue to rise until the ceiling value for interior pixels is reached (d).
The perimeter energy can be efficiently computed by making use of the Cleary-Wyvill algorithm (Cleary and Wyvill, 1988) to determine which pixels the edges of the polygon pass through, and the length of the polygon edge that passes through the pixel. First we scan-convert each of the polygon’s edges via the Cleary-Wyvill algorithm. This provides information about which of the pixels the perimeter passes through and the length of the perimeter through each such pixel, as illustrated in Figure 3. Next we take a sum of the energies of the pixels weighted by the length of the perimeter through the pixel.

Because we are attempting to minimize the energy, polygon edges lying in edge pixels of the energy field are the most desirable. Thus, those polygons whose edges align with the edge pixels, and therefore conform to the boundaries in the underlying image will receive the lowest (and best) scores.

**Alternative:** Alternative energy functions might be used to characterize regions of a quite different sort from boundaries perceived as physically contained regions. For instance, an energy function might be defined to characterize the region covered by a motion sensor or a lighting device.
3.3 Iterative Minimization of the Energy Function

The process of subjective boundary determination begins when a “seed polygon” is placed somewhere within the region of interest. An iterative process attempts to minimize the total energy of the polygon by moving the vertices of the polygon to positions that are locally more optimal.

**Alternative:** We use a square vertex of a specified initial size for a seed polygon. Alternatively, the user might be given the ability to draw a polygon to be used as a seed or choose from among a set of templates, or to specify several seed positions the final boundaries of which, in union, comprise the specified region.

For each vertex of the polygon, in turn, the system calculates the energy of the polygon for each of nine positions of that vertex — the original vertex position and its eight neighbors, as shown diagrammatically in Figure 4.² (The calculation

²Of course, for vertices on the edge of the image, fewer possible positions are considered, so that the polygon remains within the image boundaries. For historical reasons, in the current implementation, this elimination of out-of-bounds pixel positions is achieved by adding a large penalty to the energy function.)
of the updated energy does not require recomputation of the energy for the entire polygon. Only the two edges that end at the moved vertex need be scan-converted.) The process is repeated to determine a new position independently for each of the vertices. When new positions for all of the vertices have been determined, the vertices are simultaneously moved to the new positions, and the process is repeated. When an entire pass over the set of vertices results in no repositionings of any vertices, the polygon has reached a local energy minimum and the process is halted. We use the term stable for a polygon in this condition.

At this point, the system cleans up the region definition by removing extraneous vertices. When three or more vertices are collinear, the interior vertices can be deleted without altering the shape of the polygon.

Independent updating of positions has the frailty that, although each separate vertex repositioning might be reasonable, the combination of the set of repositionings can be poor. For example, in Figure 5 moving vertices $a$ and $b$ independently does not introduce edge crossings, but the two moves in combination do. To remedy this problem, when such illegal combined moves are detected, we return the participating vertices to their positions at the start of the iteration. This test-undo process completes the iteration.

**Alternative:** An alternative to the independent movement of vertices is to move the vertices in round-robin fashion, which has the advantage that no additional test-undo process is needed because later repositionings are sensitive to the changes made earlier. The round-robin method has typically led, however, to severe problems with buckling, described below. Changes to the energy function might alleviate these problems, but this has not been verified.
3.4 The Dynamic Energy Field

An energy field in which all pixels have equal positive energy promotes decreasing the lengths of polygon edges. In such a field, minimizing the energy of a polygon would amount to decreasing the edge lengths causing the polygon to collapse on itself to a single point. In the energy field described in Section 3.1, as the polygon flows towards the boundary, the decrease in the energy function can make up for the increase in length due to the larger polygon. Unfortunately, if the polygon does not circumscribe a peak of the energy function, the polygon will still be minimized by collapsing to a single point.

One way to alleviate this problem is to promote, rather than penalize, a longer perimeter. If the energy function is shifted downward so that the energies of all interior pixels are negative, though still increasing away from the contours, the polygon is still attracted towards contour pixels, but since all interior pixels are negative, a longer perimeter rather than a shorter one is preferred.

The disadvantage of such an energy function is that gratuitous increases in perimeter are also promoted. The polygon will flow through openings so as to accumulate more perimeter, a return to the failing of the filling method. A middle ground is necessary, one that attempts to retain the advantages of each energy function. Initially, the negative energy function is used, thereby initiating a phase of polygon growth. The energies of all non-contour pixels are slowly increased during the course of the minimization until the interior-positive energy function is reached. This increase can be continued further for interior pixels until a separate ceiling value for them is reached. Figures 2(b–d) display the temporal sequence of energy functions.

Initially the polygon will expand as it attempts to increase its perimeter, and thus improve its energy score. As the interior pixel energies turn positive, polygon edges contract, attempting to move into negative-valued edge pixels. When the interior height ceiling has been hit, only the edge pixels remain negative (and attractive) to the polygon edges.

Thus our algorithm exhibits a two-phased behavior, expansion when the energies are negative followed by contraction when the rising energies become positive.

**Alternative:** Alternatives to this approach to the polygon-collapsing problem are also possible. A local maximum of the energy, a peak of the function, can be found directly by iteration and the polygon forced to circumscribe it. In order to eliminate the effect of perimeter length, the average energy along the perimeter, rather than the total energy, can be minimized. Making these changes more closely follows the original analogy with an infinitely stretchable rope.

A rising energy field changes the context in which the polygon is being manipu-
lated. Even if no single vertex move at iteration $n$ improves a polygon's energy, the values of the energy field at iteration $n + i$ may be different, resulting in a different total energy for the polygon. Thus we need to redefine our concept of stability to take the rising energy field into account. We do this by continuing the deformation process until no single vertex move can improve a polygon's energy and the height ceilings have been reached.

3.5 Buckling

We have already described how the polygon can take on certain undesirable shapes, namely, shapes with self-intersections, in the course of the iterative deformation. Another such problem arises because of the negative energies early in the deformation process. These negative energies promote increases in perimeter length, which can lead to "buckling" of the polygon, that is, highly non-convex polygons with large perimeter. Figure 6 shows how such buckles can arise. Although the rising height field should later eliminate this problem, the polygon can be caught in a local minimum of this sort and be unable to escape despite the rising height field. For this reason, it is important to promote convexity of the polygon, at least in the early stages of deformation. To this end, we add a term to the energy function during the expansion phase that disfavors concavity. (Note that a polygon may still become concave; if a vertex does not move, it may be "left behind", introducing a buckle into the polygon). During the contraction phase, however, interior pixels become positive. Any buckles introduced at this point must be the result of the polygon conforming to edge pixels in the energy field. Consequently, we permit the polygon to become concave during the contraction phase.

3.6 Adding Vertices

As the polygon grows, the edges become longer, and the ability of the polygon to conform to complex boundary shapes decreases. In order to allow the polygon to fit the boundary better, it is therefore desirable to increase the number of vertices dynamically. Again, a two-phase approach has been instituted. A maximum edge length is defined so that when an edge exceeds this length, a new vertex is added that bisects the edge. The position of the new vertex is chosen to be the pixel nearest the true bisection point that minimizes the energy function. It is sometimes preferable to allow addition of edges in only one of the two phases of deformation, or not at all. (We will show examples of this in Section 5.) Consequently, separate parameters of the algorithm control the addition of vertices in the two phases.
Figure 6: Polygon with a buckle at vertex v. During the deformation process, the system will move v to each of its eight neighbors in order to determine which position minimizes the energy function. Without a concavity penalty, the system will favor moving v to v2. However, in order to conform to the boundaries in the image, v needs to move towards the upper-left corner (i.e., towards v1).
3.7 The Algorithm in Summary

As a result of the considerations of the previous sections, we can summarize the energy function as the sum of the following terms, which are intended to reflect both how well the edges in the polygon conform to the boundaries in the image, and how well the polygon retains its original shape:

- **Total perimeter energy**: The total energy score of the polygon as a function of the rising energy field.
- **Out-of-bounds penalty**: A penalty to prevent the vertices from moving beyond the boundaries of the scanned image.
- **Self-intersection penalty**: A penalty to discourage the polygon from intersecting itself.
- **Concavity penalty**: A penalty that encourages the polygon to remain convex during the initial expansion phase.

Once a polygon has been seeded, the system iteratively deforms the polygon, attempting to find a local minimum of this dynamically varying energy function. Figure 7 contains pseudo-code for this final version of our algorithm.

Five user-specifiable parameters, alluded to in the above discussion, guide the deformation process, enabling the user to control the system’s identification of a variety of regions. These parameters are listed here for completeness.

- **Position**: The location of the seed polygon. The system is relatively insensitive to variation in the seed position.
- **Radius**: The radius of the seed polygon, which roughly specifies the overall scale of the region.
- **Maximum edge length**: The edge length above which to split edges by adding vertices. This can be thought of as a measure of how complex the region boundary is. For certain subjective boundaries, it is useful to limit the number of vertices in the polygon; the user can achieve this by setting the maximum edge length to infinity. An example is provided in Section 5.
- **Expansion vertex addition**: A flag which determines whether vertices should be added during the expansion phase.
- **Contraction vertex addition**: A corresponding flag for the contraction phase. In most cases, there is little difference between adding vertices in one phase as compared to both phases. It is usually for efficiency that we will choose one phase over both. Notice that if vertices are added, the edge length at which polygon edges are divided is the same for both phases.
input the bitmap image
calculate the energy field
repeat
  seed a region
  repeat
    add new vertices to polygon as appropriate
    increment the energy field as appropriate
    calculate polygon's total energy
    for each vertex
      for each of 9 neighboring positions
        recalculate the polygon's energy with
        the vertex at the new position,
        recording the neighboring position
        with the lowest energy
      update each vertex position to the neighboring
      position with the lowest energy
    test for and undo any illegal moves
  until polygon stable and height ceiling hit
delete collinear vertices
until all regions are characterized

Figure 7: Pseudo-code for the full algorithm.
Alternative: Various alternatives to these parameters might be entertained. The location parameter may be eliminable by exhaustive consideration of all local maxima of the initial energy field, by random or uniform sampling of seed positions, by region-growing (Ballard and Brown, 1982), or by some other method. Initial radius might be determined automatically by a greedy algorithm that iteratively increases the radius of the initial polygon until some vertex impinges on an edge pixel. The vertex-addition parameters might be simplified by eliminating the expansion phase as described in the alternative of Section 3.4.

4 The Prototype System

We have implemented this algorithm for region annotation in a prototype system. The interface for the system includes a viewing area for the bitmap image and a control panel for the various parameters. The user seeds a region by clicking with the mouse. The deforming polygon is displayed on top of the original bitmap image. The ability to view the deformation process is a key element of the interactive nature of our system. This visual information can help the user choose a better seed position if necessary, and determine more appropriate parameter settings to better guide the deformation process. After changing the input parameters, the user can also rerun the previous seed. In addition, the system permits the user to save out seed positions and parameter settings, which may prove useful for quickly annotating several (similarly designed) floors of a single building.

Another important feature of our system is that it supports user post-editing of candidate regions. A user can designate an area of interest by using the mouse to mark the region with a bounding box. The system then increases the spatial resolution of this area and presents the user with a new window containing the enlarged region. This is important because it is very difficult for the user to work at the screen-pixel level; by zooming in, the system provides the user with a greater degree of accuracy. The user can move, add, or delete vertices as desired. He then has the option to continue running the deformation process or accept the region as currently defined.

Once the user has accepted a region definition, the system will save this polygonal description as a vertex list. At this point, the user may annotate the region with various properties. Typical characteristics of building regions include room numbers, names of occupants, telephone information, lighting characteristics, etc.
5 Some Examples

In this section we give several examples, discussing the various input parameters used so as to give a flavor for the style of interaction and utility of the small number of parameters available to the user. The examples — a fully bounded region and two partially bounded regions — are based on the bitmap image shown in Figure 8, which was generated by scanning in a hardcopy of a floor plan. We discuss the use of the five input parameters listed in Section 3.7.

The first example, as shown in Figure 9, is a commonly shaped room for this floor plan. The default radius of one pixel, and a relatively small maximum edge length of 8 pixels, are used. Vertices were added during both the expansion and the contraction phase. The region was seeded near the center of the room. The system is able to identify the region, approximating the door swing (the upper left hand corner of the region) by a series of line segments. The candidate definition matches the boundaries in the underlying image, including the small alcove-like indentation in the bottom of the region.

The same settings will work to identify the second region, as indicated in Figure 10, which is similar in shape to the first region. There is, however, a key difference between the two regions. Region 2 is only partially bounded due to noise in the image. Both the bottom and right hand boundaries of the region are missing multiple pixels, so that filling techniques would not successfully delineate this region. Our system does find an appropriate boundary, however, for this and similar
Figure 9: Proposed definition for a fully bounded region.

Figure 10: Candidate definition for a partially bounded (noisy) region.
regions in the image.

Finally, we consider the third region. Assume the user is attempting to identify the large rectangular area in the middle of the image; it might be a lobby or some other type of common area. For this region, the user changes several of the parameters. First, the polygon's radius must be increased so that the initial polygon encloses the two small rectangles in the region. We used a radius of 50 pixels for this example. Next, we set the maximum edge length to infinity, preventing the system from adding additional vertices to the polygon during either phase. The system identifies the large rectangular region, inferring the missing boundaries in the original image.

Our system models simple polygons; in order to delineate toroidal shaped regions (for instance, to incorporate the two holes indicated by the two small rectangles in the middle of this region) we would need to combine two or more regions. Our system does not currently handle such regions.

6 Acknowledgements

The authors would like to acknowledge the aid of Joel Bartlett of Digital Equipment Corporation's Western Research Lab for valuable discussions and for providing sample bitmaps.
This material is based in part upon work supported by the National Science Foundation under a Presidential Faculty Fellow Grant No. IRI-9350192 to Stuart Shieber. In addition, author Ryall was supported in part by a summer internship at Digital’s Cambridge Research Laboratory.

References


