MODEL-BASED OPTIMIZATION: ACCURATE AND CONSISTENT SITE MODELING

P. Fua
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
fua@ai.sri.com

Commision III, Working Group 2

KEY WORDS: Vision Sciences, Cartography, Modeling, Aerial, Three-dimensional

ABSTRACT

Model-Based Optimization (MBO) is a paradigm in which an objective function is used to express both geometric and photometric constraints on features of interest. A parametric model of a feature, such as a road, a building, a river or the underlying terrain, is extracted from one or more images by adjusting the model's state variables until a minimum value of the objective function is obtained. The optimization procedure yields a description that simultaneously satisfies (or nearly satisfies) all constraints, and, as a result, is likely to be a good model of the feature.
Furthermore, because objects are all modeled in the same fashion, we can refine the models simultaneously and enforce geometric and semantic constraints between objects, thus increasing not only the accuracy but also the consistency of the reconstruction.
We believe that these capabilities will prove indispensable to automating the generation of complex object databases from imagery, such as the ones required for realistic simulations or intelligence analysis.

1 INTRODUCTION

Model-Based Optimization (MBO) is a paradigm in which an objective function is used to express both geometric and photometric constraints on features of interest. A parametric model of a feature (such as a road, a building, or coastline) is extracted from one or more images by adjusting the model's state variables until a minimum value of the objective function is obtained. The optimization procedure yields a description that simultaneously satisfies (or nearly satisfies) all constraints, and, as a result, is likely to be a good model of the feature.

The deformable models we use here are extensions of traditional snakes [Terzopoulos, *et al.*, 1987, Kass *et al.*, 1988, Fua and Leclerc, 1990]. They are polygonal curves or facetized surfaces to which is associated an objective function that combines an "image term" that measures the fit to the image data and a regularization term that enforces geometric constraints.

Because features and surfaces are all modeled in a uniform fashion, we can refine several models simultaneously and enforce geometric and semantic constraints between objects, thus increasing not only the accuracy but also the consistency of the reconstruction. The ability to apply such constraints is essential for the accurate modeling of complex sites in which objects obey known geometric and semantic constraints. In particular, when dealing with multiple objects, it is crucial that the models be both accurate and consistent with each other. For example, individual components of a building can be modeled independently, but to ensure realism, one must guarantee that they touch each other in an architecturally feasible way. Similarly, when modeling a cartographic site from aerial imagery, one must ensure that the roads lie on the terrain—and not above or below it—and that rivers flow downhill. To that end, we have developed a constrained-optimization scheme that allows us to impose hard constraints on our snakes at a very low computational cost while preserving their convergence properties.

We first introduce our generalized snakes. We then present our constrained-optimization scheme. Finally, we demonstrate its ability to enforce geometric constraints upon individual snakes and consistency constraints upon multiple snakes to produce complex and consistent site models.

2 GENERALIZED SNAKES

We model linear features as polygonal curves that may be described either as a sequential list of vertices, or, for more complex objects such as a road network or a 3-D extruded object, described by the network topology. In the latter case, to describe the object completely, one must supply not only the list of vertices but also a list of "edges" that defines the connectivity of those vertices. In addition, with some of these complex objects, one can also define "faces," that is, circular lists of vertices that must be constrained to remain planar.

Similarly, we model the terrain on which these features rest as triangulated surface meshes whose shape is defined by the position of vertices and can be refined by minimizing an objective function.

Our ultimate goal is to accommodate the full taxonomy of those "generalized snakes" described by Table 1. The algorithms described here are implemented within the Radius Common Development Environment (RCDE) [Mundy *et al.*, 1992].

2.1 Polygonal Snakes

A simple polygonal snake, $\mathcal{C}$, can be modeled as a sequential list of vertices, that is, in two dimensions, a list of 2–D vertices $\mathcal{S}_2$ of the form

$$\mathcal{S}_2 = \{(x_i \; y_i), \; i = 1, \ldots, n\} \; , \qquad (1)$$

Constraints/Type	Simple curve	Ribbon curve	Network	Triangulated meshes
Smooth	Low res. roads, rivers	High res. roads	Road network	Terrain
Polygonal	Man-made structures	City streets	Street Networks	
Planar	Planar structures	City streets	Street Networks	
Rectilinear	Roof tops, parking lots	City streets	Buildings	

Table 1: Snake taxonomy. The columns represent different types of snakes and the rows different kinds of constraints that can be brought to bear. The table entries are examples of objects that can be modeled using these combinations.

and, in three dimensions, a list of 3–D vertices $\mathcal{S}_3$ of the form

$$\mathcal{S}_3 = \{(x_i \; y_i \; z_i), \; i = 1, \ldots, n\} \; . \qquad (2)$$

In this paper, we refer to S, the vector of all x, y, and z coordinates of the 2–D or 3–D vertices that define the deformable model's shape as the model's *state vector*.

In the 2–D case, the "image energy" of these curves—the term we try to minimize when we perform the optimization is taken to be

$$\mathcal{E}_I(\mathcal{C}) = -\frac{1}{|\mathcal{C}|} \int_0^{|\mathcal{C}|} |\nabla \mathcal{I}(\mathbf{f}(s))| \, ds, \qquad (3)$$

where I represents the image gray levels, s is the arc length of $\mathcal{C}$, $\mathbf{f}(s)$ is a vector function mapping the arc length s to points (x, y) in the image, and $|\mathcal{C}|$ is the length of $\mathcal{C}$. In practice, $\mathcal{E}_I(\mathcal{C})$ is computed by integrating the gradient values $|\nabla \mathcal{I}(\mathbf{f}(s))|$ in precomputed gradient images along the line segments that connect the polygonal vertices.

In the 3–D case, illustrated by Figures 1 and 2(a), $\mathcal{E}_I(\mathcal{C})$ is computed by projecting the curve into a number of images, computing the image energy of each projection, and summing these energies.

2.2 Smooth Snakes and Ribbons

These snakes are used to model smoothly curving features such as roads or ridgelines.

2–D curves. Following Kass *et al.* [1988], we choose the vertices of such curves to be roughly equidistant and add to the image energy $\mathcal{E}_I$ a regularization term $\mathcal{E}_D$ of the form

$$\begin{aligned}\mathcal{E}_D(\mathcal{C}) = \mu_1 \textstyle\sum_i (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \\ + \mu_2 \textstyle\sum_i (2x_i - x_{i-1} - x_{i+1})^2 + (2y_i - y_{i-1} - y_{i+1})^2\end{aligned} \qquad (4)$$

and define the "total energy" $\mathcal{E}_T$ as

$$\mathcal{E}_T(\mathcal{C}) = \mathcal{E}_D(\mathcal{C}) + \mathcal{E}_I(\mathcal{C}) \; . \qquad (5)$$

The first term of $\mathcal{E}_D$ approximates the curve's tension, and the second term approximates the sum of the square of the curvatures, assuming that the vertices are roughly equidistant. In addition, when starting, as we do, with regularly spaced vertices, this second term tends to maintain that regularity. To perform the optimization we could use the steepest or conjugate gradient, but it would be slow for curves with large numbers of vertices. Instead, it has proven much more effective to embed the curve in a viscous medium and solve the equation of the dynamics

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial S} + \alpha \frac{dS}{dt} &= 0, \qquad (6)\\ \text{with } \frac{\partial \mathcal{E}}{\partial S} &= \frac{\partial \mathcal{E}_D}{\partial S} + \frac{\partial \mathcal{E}_I}{\partial S},\end{aligned}$$

where $\mathcal{E}$ is the energy of Equation 5, α the viscosity of the medium, and S the state vector that defines the current position of the curve. Since the deformation energy $\mathcal{E}_D$ in Equation 4 is quadratic, its derivative with respect to S is linear, and therefore Equation 6 can be rewritten as

$$\begin{aligned}K_S S_t + \alpha(S_t - S_{t-1}) &= -\left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} \\ \Rightarrow (K_S + \alpha I) S_t &= \alpha S_{t-1} - \left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} \; , \qquad (7)\end{aligned}$$

where

$$\frac{\partial \mathcal{E}_D}{\partial S} = K_S S,$$

and K_S is a sparse matrix. Note that the derivatives of $\mathcal{E}_D$ with respect to x and y are decoupled so that we can rewrite Equation 7 as a set of two differential equations of the form

$$(K + \alpha I) V - t = \alpha V_{t-1} - \left.\frac{\partial \mathcal{E}_I}{\partial V}\right|_{V_{t-1}} \; , \qquad (8)$$

where V stands for either X or Y, the vectors of the x and y vertex coordinates, and K is a pentadiagonal matrix. Because K is pentadiagonal, the solution to this set of equations can be computed efficiently in $O(n)$ time using LU decomposition and backsubstitution. Note that the LU decomposition need be recomputed only when α changes.

In practice, α is computed in the following manner. We start with an initial step size Δ_p, expressed in pixels, and use the following formula to compute the viscosity:

$$\alpha = \frac{\sqrt{2n}}{\Delta_p} \left| \frac{\partial \mathcal{E}}{\partial S} \right| , \qquad (9)$$

where n is the number of vertices. This ensures that the initial displacement of each vertex is on the average of magnitude Δ_p. Because of the nonlinear term, we must verify that the energy has decreased from one iteration to the next. If, instead, the energy has increased, the curve is reset to its previous position, the step size is decreased, and the viscosity recomputed accordingly. This procedure is repeated until the step size becomes less than some threshold value. In most cases, because of the presence of the linear term that propagates constraints along the whole curve in one iteration, it takes only a small number of iterations to optimize the initial curve.

3–D curves. To extend the smooth snakes to three dimensions, we add one term in z to the deformation energy of Equation 4. Since the derivatives of $\mathcal{E}_D$ with respect to x, y, and z are still decoupled, we can rewrite Equation 7 as a set of three differential equations of the form of Equation 8, where V now stands for either X, Y, or Z, the x, y, or z vertex coordinates.

The only major difference with the 2–D case is the use of the images' camera models. In practice, $\mathcal{E}_I(\mathcal{C})$ is computed by

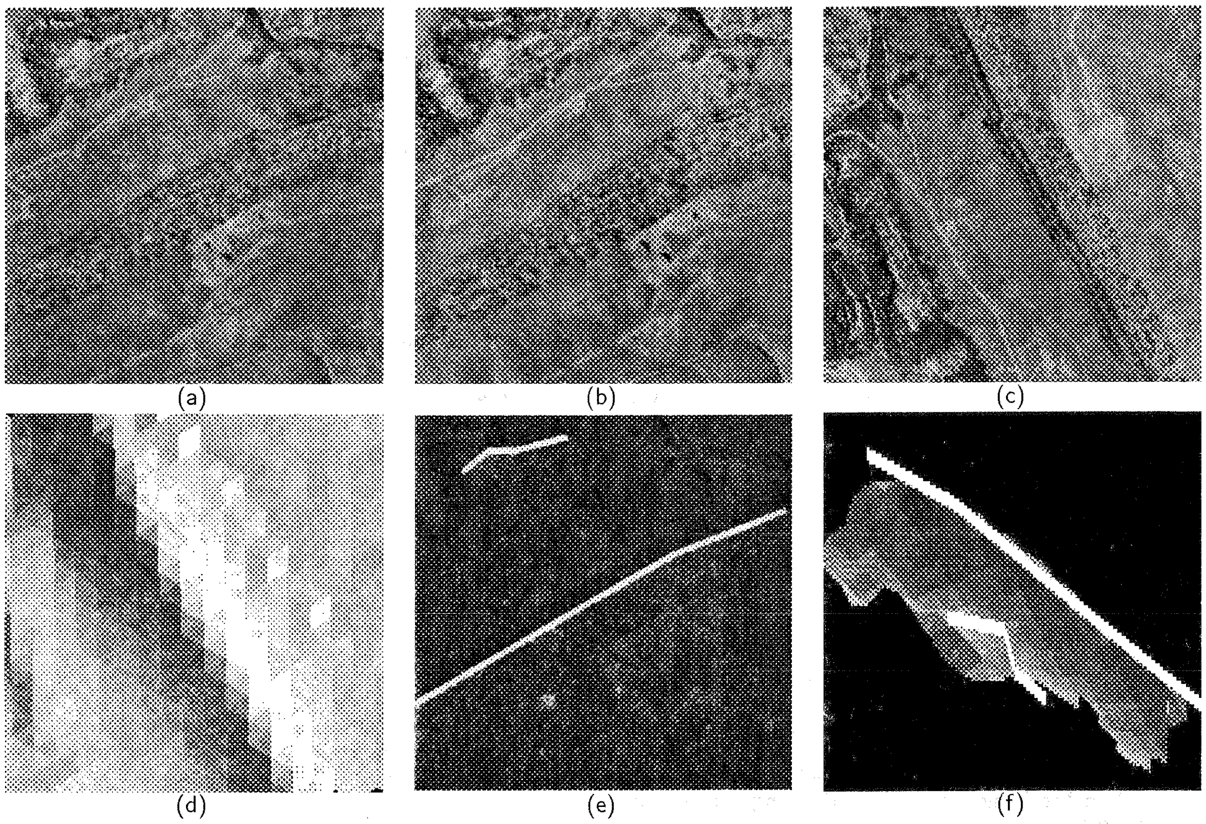

Figure 1: Rugged terrain with sharp ridge lines. (a,b,c) Three images of a mountainous site. (d) Shaded view of an initial terrain estimate. (e) Rough polygonal approximation of the ridgelines overlaid on image (a). (f) The terrain and ridgeline estimates viewed from the side (the scale in z has been exaggerated).

summing gradient values along the line segments linking the vertices' projections. These projections, and their derivatives, are computed from the state vector S by using the camera models. Similarly, to compute the viscosity, we use the camera models to translate the average initial step Δ_p, a number of pixels, into a step Δ_w expressed in world units and use the latter in Equation 9.

Ribbons 2–D snakes can also be extended to describe ribbon-like objects such as roads in aerial images. A ribbon snake is implemented as a polygonal curve forming the center of the road. Associated with each vertex i of this curve is a width w_i that defines the two curves that are the candidate road boundaries. The list of vertices can be written as

$$\mathcal{S}_2 = \{(x_i \; y_i \; w_i)\}, \; i = 1, \ldots, n\} \; . \qquad (10)$$

The state vector S becomes the vector of all x, y, and w and the average edge strength the sum of the edge strengths along the two boundary curves. Since the width of roads tends to vary gradually, we add an additional energy term of the form

$$\begin{aligned} \mathcal{E}_W(\mathcal{C}) &= \sum_i (w_i - w_{i-1})^2 \qquad (11) \\ \Rightarrow \frac{\partial \mathcal{E}_W}{\partial W} &= LW, \end{aligned}$$

where W is the vector of the vertices' widths and L a tridiagonal matrix. The total energy can then be written as

$$\mathcal{E}(\mathcal{C}) = \lambda_D \mathcal{E}_D(\mathcal{C}) + \lambda_W \mathcal{E}_W(\mathcal{C}) + \lambda_G \mathcal{E}_I(\mathcal{C}) \; ,$$

where λ_D and λ_W weigh the contributions of the two geometric terms. At each iteration the system must solve the three differential equations in the form of Equation 8, where V now stands for either X, Y, or W, the x, y, or w vertex coordinates.

2–D ribbons can be turned into 3–D ones in exactly the same way 2–D snakes are turned into 3–D ones. The state vector S becomes the vector of all x, y, z, and w and, at each iteration, the system must solve four differential equations, one for each coordinate.

2.3 Network Snakes

The 2–D and 3–D "network snakes" are a direct extension of the polygonal snakes of Section 2.1.

In the 2–D case, the extension is straightforward. A network snake is now defined by a list of n vertices $\mathcal{S}$ as before and a list of edges $\mathcal{A} = \{(i,j)$ where $1 \leq i \leq n$ and $1 \leq j \leq n\}$. Figure 3 depicts such a network snake. $\mathcal{E}_I(\mathcal{C})$ is computed as

$$\mathcal{E}_I(\mathcal{C}) = \sum_{(i,j) \in \mathcal{A}} \mathcal{E}_I^{i,j} / \sum_{(i,j) \in \mathcal{A}} L^{i,j} \; , \qquad (12)$$

where $\mathcal{E}_I^{i,j}$ is the sum of the edge gradients along the $((x_i, y_i)(x_j, y_j))$ segment and $L^{i,j}$ is its length. The snake is optimized using either steepest gradient descent or conjugate gradient.

In the 3–D case, one must take into account the fact that not all the network's edges are visible in all views. As a result one must also provide, for each projection of the snake into

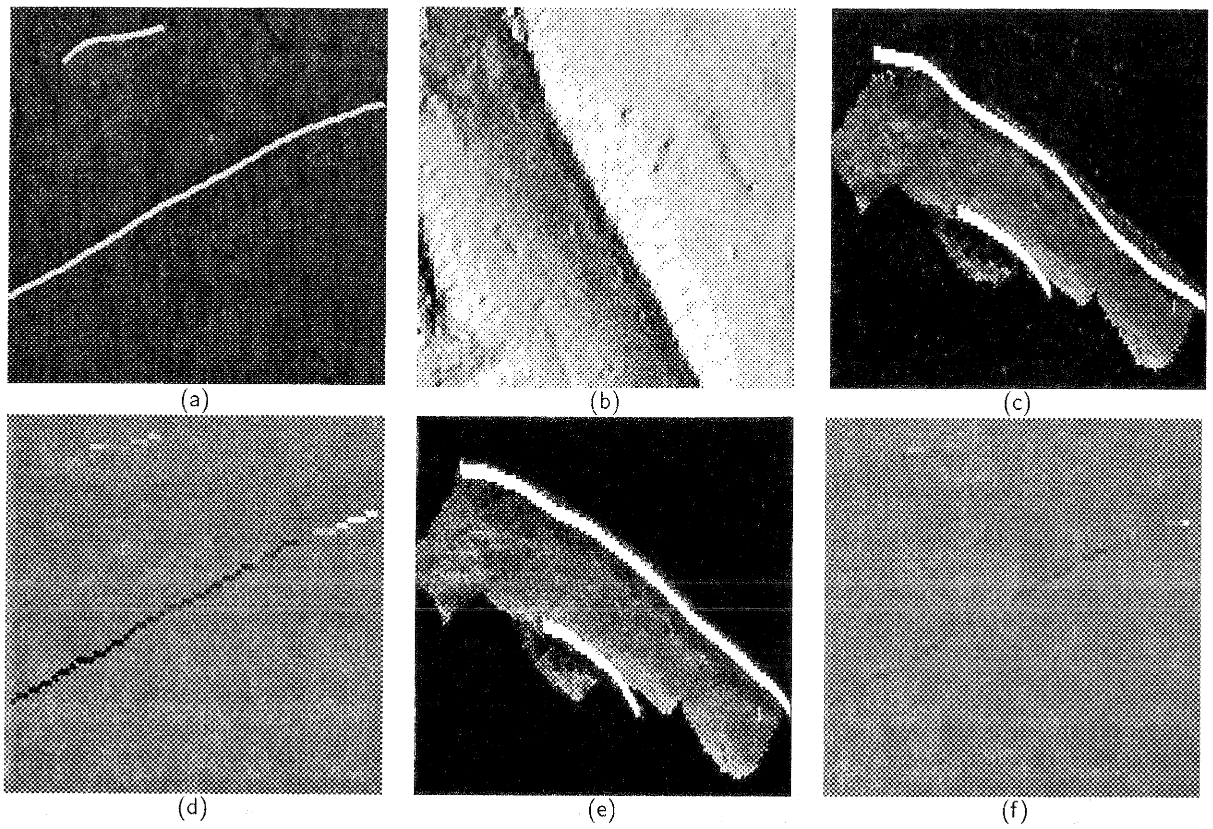

Figure 2: Recovering the 3–D geometry of both terrain and ridges. (a) Refined ridgeline after 3–D optimization. (b) Shaded view of the terrain after refinement. (c) Side view of the ridgeline and terrain after independent optimization of each one. Note that the shape of the ridgeline does not exactly match that of the terrain. (d) Differences of elevation between the recovered ridgeline and the underlying terrain. The image is stretched so that black and white represent errors of minus and plus 80 feet, respectively. (e) Side view after optimization under consistency constraints. (f) Corresponding difference of elevation image stretched in the same fashion as (d).

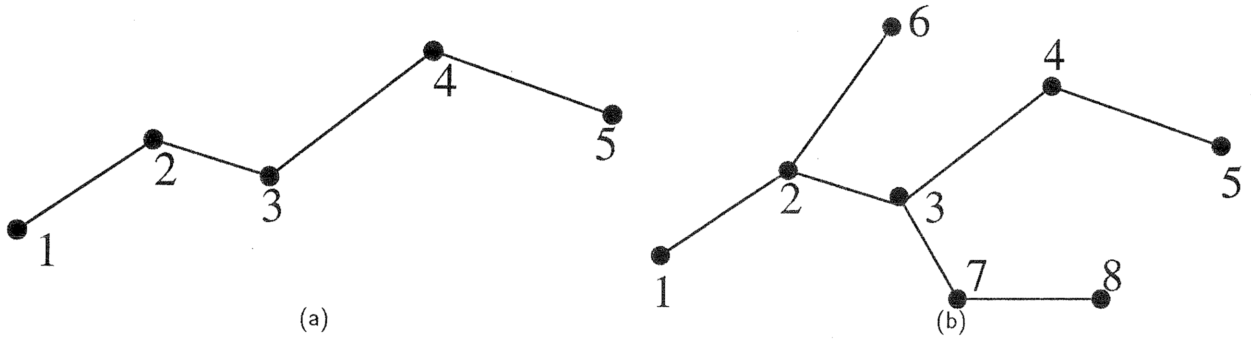

Figure 3: Snake topology. (a) A simple polygonal curve described by a sequential list of vertices v_i, $1 \leq i \leq 5$. (b) A network described by a list of vertices v_i, $1 \leq i \leq 8$, and a list of edges—((1 2) (2 3) (3 4) (4 5) (2 6) (3 7) (7 8)).

all the images, a list of visible edges. We compute this list by using the face-visibility methods embedded in RCDE as shown in Figure 4.

The number of degrees of freedom of generic 3–D networks can be reduced by forcing them to be planar. We do this either by defining a plane of equation

$$z = ax + by + c \tag{13}$$

and imposing that the vertices lie on such a plane or imposing planar constraints on sets of four vertices using the constrained-optimization approach introduced in Section 3.1. In both cases, we replace the n degrees of freedom necessary to specify the elevation of each vertex by the three degrees of freedom required to define the plane.

These 3–D networks can be further specialized to handle objects that are of particular interest in urban environments: trihedral corners found on building roofs and extruded objects that are used in RCDE to model building outlines. In Figure 5, we show several buildings modeled by roughly entering their outlines within RCDE and optimizing the shapes

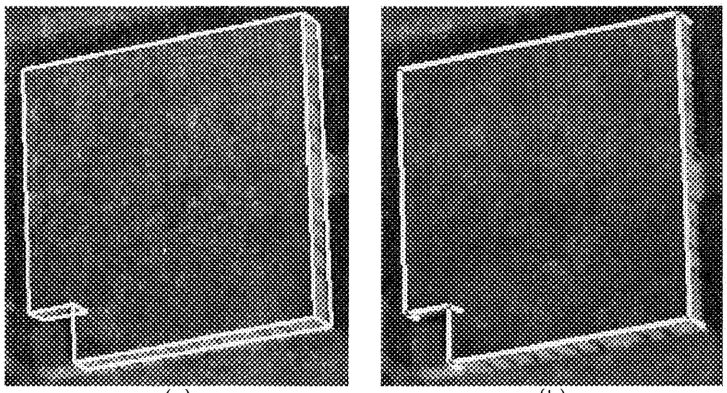

Figure 4: Edge visibility. (a) An RCDE "extruded object." Only the visible faces—that is, those whose normal is oriented towards the viewer—are drawn. Note that this heuristic does not account for nonconvexity, and as a result the faces in the lower left corner of the image are improperly drawn. (b) The network snake generated to optimize the extruded object. It includes roof edges and vertical wall edges. The edges at the back of the building are not drawn—and not used during the computations involving these views—because they belong to hidden faces. The edges at the base of the building are treated as invisible because their appearance is unreliable in typical imagery.

in three views simultaneously by using our extruded snakes. The use of the snakes has allowed us to perform this task much faster than we would have if we had had to precisely delineate all five buildings by hand. To produce this result, we have used the constrained-optimization technique of Section 3.1 to constrain the "wall" edges to remain vertical. We can also constrain the "roof outline" to be planar and the "roof edges" to form 90-degree angles. These constraints greatly reduce the number of degrees of freedom and allow for better convergence properties.

2.4 3–D Surface Meshes

Given the task of reconstructing a surface from multiple images whose vantage points may be very different, we need a surface representation that can be used to generate images of the surface from arbitrary viewpoints, taking into account self-occlusion, self-shadowing, and other viewpoint-dependent effects. Clearly, a single image-centered representation is inadequate for this purpose. Instead, an object-centered surface representation is required.

Many object-centered surface representations are possible. However, practical issues are important in choosing an appropriate one. First, the representation should be general-purpose in the sense that it should be possible to represent any continuous surface, closed or open, and of arbitrary genus. Second, it should be relatively straightforward to generate an instance of a surface from standard data sets such as depth maps or clouds of points. Finally, there should be a computationally simple correspondence between the parameters specifying the surface and the actual 3-D shape of the surface, so that images of the surface can be easily generated, thereby allowing the integration of information from multiple images.

A regular 3–D triangulation is an example of a surface representation that meets the criteria stated above, and is the one we have chosen for our previous work. In our implementation, all vertices except those on the edges have six neighbors and are initially regularly spaced. Such a mesh defines a surface composed of three-sided planar polygons that we call triangular facets, or simply facets. Triangular facets are particularly easy to manipulate for image and shadow generation; consequently, they are the basis for many 3-D graphics systems. These facets tend to form hexagons and can be used to construct virtually arbitrary surfaces. Finally, standard triangulation algorithms can be used to generate such a surface from noisy real data [Fua and Sander, 1992, Szeliski and Tonnesen, 1992].

Sources of information. A number of information sources are available for the reconstruction of a surface and its material properties. Here, we consider two classes of information.

The first class comprises those information sources that do not require more than one image, such as texture gradients, shading, and occlusion edges. When using multiple images and a full 3-D surface representation, however, we can do certain things that cannot be done with a single image. First, the information source can be checked for consistency across all images, taking occlusions into account. Second, when the source is consistent and occlusions are taken into account, the information can be fused over all the images, thereby increasing the accuracy of the reconstruction.

The second class comprises those information sources that require at least two images, such as the triangulation of corresponding points between input images (given camera models and their relative positions). Generally speaking, this source is most useful when corresponding points can be easily identified and their image positions accurately measured. The ease and accuracy of this correspondence can vary significantly from place to place in the image set, and depend critically on the type of feature used. Consequently, whatever the type of feature used, one must be able to identify where in the images that feature provides reliable correspondences, and what accuracy one can expect.

The image feature that we have chosen for correspondence (although it is by no means the only one possible) is simply intensity in radiometrically corrected images—for example, by filtering them. Clearly, intensity can be a reliable feature only when the albedo varies quickly enough on the surface and, consequently, the images are sufficiently textured.

Simple correlation-based stereo methods often use fixed-size

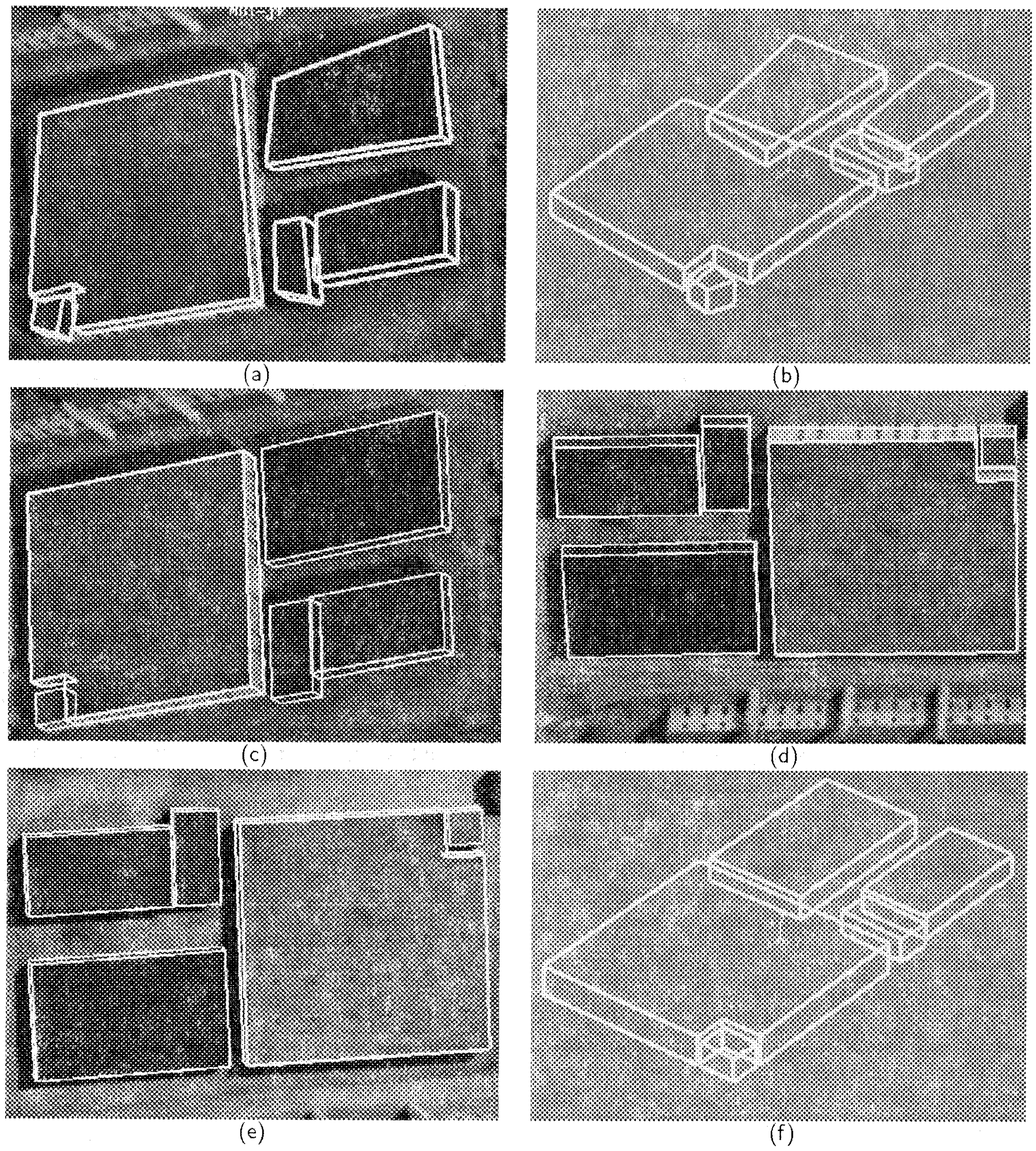

Figure 5: Buildings modeled by entering rough models within RCDE and optimizing them using the extruded snakes. (a) Rough initial sketches overlaid on one of the images. (b) A view from a different perspective. (c,d,e) Final building outlines overlaid on the three images we used to perform the 3–D optimization. (f) A view of the buildings from the perspective of (b).

windows in images to measure disparities, which will in general yield correct results only when the surface is parallel to the image plane. Instead, we compare the intensities as projected onto the facets of the surface. Consequently, the reconstruction can be significantly more accurate for slanted surfaces. Some correlation-based algorithms achieve similar results by using variable-shaped windows in the images [Quam, 1984, Nishihara, 1984, Kanade and Okutomi, 1990, Baltsavias, 1991, Devernay and Faugeras, 1994]. However, they typically use only image-centered representations of the surface.

Our approach is much more closely related to the least-squares approaches advocated by Wrobel [1991] and Heipke [1992], who both use a 2-1/2-D representation of the surface.

As for the monocular information source, we have chosen to use shading, where shading is the change in image intensity due to the orientation of the surface relative to a light source. We use this method because shading is most reliable when the albedo varies slowly across the surface; this is the natural complement to intensity correspondence, which requires quickly varying albedo. The complementary nature of these two sources allows us to accurately recover the surface geometry and material properties for a wide variety of images.

In contrast to our approach, traditional uses of shading information assume that the albedo is constant across the entire surface, which is a major limitation when applied to real images. We overcome this limitation by improving upon a

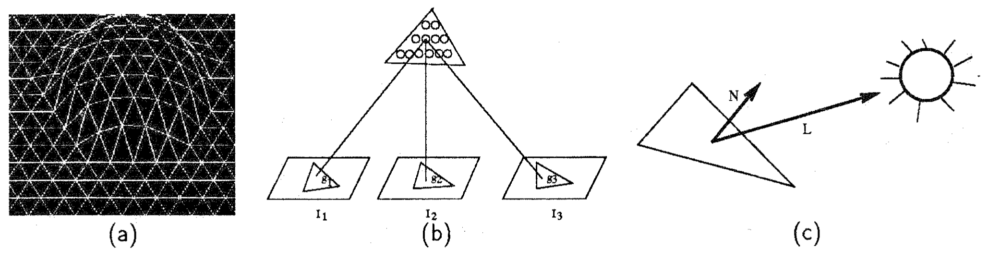

Figure 6: Mesh representation and computation of the image terms of the objective function: (a) Wireframe representation of the mesh. (b) Facets are sampled at regular intervals; the circles represent the sample points. The stereo component of the objective function is computed by summing the variance of the gray level of the projections of these sample points, the g_is. (c) Each facet's albedo is estimated using its normal N, the light source direction L, and the average gray level of the projection of the facet into the images. The shading component of the objective function is the sum of the squared differences in estimated albedo across neighboring facets.

method to deal with discontinuities in albedo alluded to in the summary of Leclerc and Bobick [1991]. We compute the albedo at each facet by using the normal to the facet, a light-source direction, and the average of the intensities projected onto the facet from all images. We use the local variation of this computed albedo across the surface as a measure of the correctness of the surface reconstruction. To see why albedo variation is a reasonable measure of correctness, consider the case when the albedo of the real surface is constant. When the geometry of the mesh is correct, the computed albedo should be approximately the same as the real albedo, and hence should be approximately constant across the mesh. Thus, when the geometry is incorrect, this will generally give rise to variations in the computed albedo that we can take advantage of. Furthermore, by using a *local* variation in the computed albedo, we can deal with surfaces whose albedo is not constant, but instead varies slowly over the surface.

Implementation. The triangulated 3–D mesh of vertices that represents a surface, $\mathcal{S}$, is a hexagonally connected set of vertices such as the one shown in Figure 6(a). The position of a vertex v_j is specified by its Cartesian coordinates (x_j, y_j, z_j). The mesh can be deformed by varying these coordinates to minimize an objective function that includes terms derived from stereo and shading information. Its state vector S is the vector of all x,y, and z coordinates.

The stereo component of the objective function is derived by comparing the gray levels of the points in all the images for which the projection of a given point on the surface is visible. It is similar to the term proposed by Wrobel [1991]. As shown in Figure 6(b), this comparison is done for a uniform sampling of the surface. This method allows us to deal with arbitrarily slanted regions and to discount occluded areas of the surface.

The shading component of the objective function is computed by using a method that does not invoke the traditional constant albedo assumption. Instead, it attempts to minimize the variation in albedo across the surface, and can therefore deal with surfaces whose albedo varies slowly. This term is depicted by Figure 6(c).

The stereo term is most useful when the surfaces are highly textured. Conversely, the shading term is most reliable where the surfaces have little or no texture. To account for this phenomenon, we take the complete objective function, $\mathcal{E}(\mathcal{S})$, to be a weighted average of these two components where the weighting is a function of texture within the projections of individual facets.

In general, $\mathcal{E}(\mathcal{S})$ is a highly nonconvex function of the vertex positions. To minimize $\mathcal{E}(\mathcal{S})$, we use the "snake-type" [Kass *et al.*, 1988] optimization technique of Section 2.2. We define the total energy of the mesh, $\mathcal{E}_T(\mathcal{S})$, as

$$\mathcal{E}_T(\mathcal{S}) = \mathcal{E}_D(\mathcal{S}) + \mathcal{E}(\mathcal{S}) \tag{14}$$

where $\mathcal{E}_D(\mathcal{S})$ is a regularization term analogous to the one of Equation 5. In practice, we take $\mathcal{E}_D$ to be a measure of the curvature or local deviation from a plane at every vertex. Because the mesh is regular, $\mathcal{E}_D$ can be approximated by using finite differences as a quadratic form [Fua and Leclerc, 1995]

$$\mathcal{E}_D(\mathcal{S}) = 1/2(X^T KX + Y^T KY + Z^T KZ)\,, \tag{15}$$

where X,Y, and Z are the vectors of the x,y, and z coordinates of the vertices, and K is a sparse and banded matrix. This regularization term serves a dual purpose. First, as before, it "convexifies" the energy landscape when λ_D is large and improves the convergence properties of the optimization procedure. Second, in the presence of noise, some amount of smoothing is required to prevent the mesh from overfitting the data, and wrinkling the surface excessively.

To speed the computation and prevent the mesh from becoming stuck in undesirable local minima, we typically use several levels of mesh sizes—three in the example of Figure 2(b)—to perform the computation. We start with a relatively coarse mesh that we optimize. We then refine it by splitting every facet into four smaller ones and reoptimizing. Finally, we repeat the split and optimization processes one more time.

3 ENFORCING CONSISTENCY

We now turn to the enforcing of geometric and consistency constraints on the multiple objects that may compose a complex site.

A traditional way to enforce such constraints is to add a penalty term to the model's energy function for each constraint. While this may be effective for simple constraints, this approach rapidly becomes intractable as the number of constraints grows, for two reasons. First, it is well known that minimizing an objective function that includes such penalty terms constitutes an ill-behaved optimization problem with poor convergence properties [Fletcher, 1987, Gill *et al.*, 1981]: the optimizer is likely to minimize the constraint terms while ignoring the remaining terms of the objective function. Second, if one tries to enforce several constraints of different natures, the penalty terms are unlikely to be commensurate and one has to face the difficult problem of adequately weighing the various constraints.

Using standard constrained optimization techniques is one way of solving these two problems. However, while there are many such techniques, most involve solving large linear systems of equations and few are tailored to preserving the convergence properties of the snake-like approaches of Sections 2.2 and 2.4. Exceptions are the approach proposed by Metaxas and Terzopoulos [1991] to enforce holonomic constraints by modeling the second-order dynamics of the system and the technique proposed by Amini *et al.* [1988] using dynamic programming.

Here, we propose a new approach to enforcing hard-constraints on our snakes without undue computational burden while retaining their desirable convergence properties.

3.1 Constrained Optimization in Orthogonal Subspaces

Formally, the constrained optimization problem can be described as follows. Given a function f of n variables $S = \{s_1, s_2, .., s_n\}$, we want to minimize it under a set of m constraints $C(S) = \{c_1, c_2, .., c_m\} = 0$. That is,

$$\text{minimize } f(S) \text{ subject to } C(S) = 0 \ . \qquad (16)$$

While there are many powerful methods for nonlinear constrained minimization [Gill *et al.*, 1981], we know of none that are particularly well adapted to snake-like optimization: they do not take advantage of the locality of interactions that is characteristic of snakes. We have therefore developed a robust two-step approach [Brechbühler *et al.*, 1995, Fua and Brechbuhler, 1996] that is closely related to gradient projection methods first proposed by Rosen [1961] and can be extended to snake optimization.

Solving a constrained optimization problem involves satisfying the constraints and minimizing the objective function. For our application, it has proved effective to decouple the two and decompose each iteration into two steps:

1. Enforce the constraints by projecting the current state onto the constraint surface. This involves solving a system of nonlinear equations by linearizing them and taking Newton steps.
2. Minimize the objective function by projecting the gradient of the objective function onto the subspace tangent to the constraint surface and searching in the direction of the projection, so that the resulting state does not stray too far away from the constraint surface.

Figure 7 depicts this procedure. Let C and S be the constraint and state vectors of Equation 16 and A be the $n \times m$ Jacobian matrix of the constraints. The two steps are implemented as follows:

1. To project S, we compute dS such that $C(S + dS) \approx C(S) + A^t dS = 0$ and increment S by dS. The shortest possible dS is found by writing dS as AdV and solving the equation $A^t AdV = -C(S)$.
2. To compute the optimization direction, we first solve the linear system $A^T(S)A(S)\lambda = A^T(S)\nabla f$ and take the direction to be $\nabla f - A\lambda$. This amounts to estimating Lagrange multipliers, that is, the coefficients that can be used to describe ∇f as closely as possible as a linear combination of constraint normals.

These two steps operate in two locally orthogonal subspaces, in the column space of A and in its orthogonal complement, the null space of A^T. Note that $A^T(S)A(S)$ is an $m \times m$ matrix and is therefore small when there are more variables than constraints, which is always the case in our application.

This technique has been used to enforce the geometric constraints in the example of Figure 5. Furthermore, it can be generalized to handle inequality constraints by introducing an "active set strategy." The inequality constraints that are strictly satisfied are deactivated, while those that are violated are activated and treated as equality constraints. This requires additional bookkeeping but does not appear to noticeably slow down the convergence of our constrained-optimization algorithm.

3.2 Constraining Snake Optimization

We could trivially extend the technique of Section 3.1 to the refinement of smooth curves and surfaces by taking the objective function f to be the total energy $\mathcal{E}_T$ of Equation 5. However, this would be equivalent to optimizing an unconstrained snake by using gradient descent as opposed to performing the implicit Euler steps that so effectively propagate smoothness constraints.

In practice, propagating the smoothness constraints is key to forcing convergence toward desirable answers. When a portion of the snake deforms to satisfy a hard constraint, enforcing regularity guarantees that the remainder of the snake also deforms to preserve it and that unwanted discontinuities are not generated. This is especially true in our application because many of the constraints we use can be satisfied by moving a small number of vertices, thereby potentially creating "kinks" in the curve or surface that subsequent optimization steps may not be able to remove without getting stuck in local minima.

Therefore, for the purpose of optimizing constrained smooth snakes, we decompose the second step of the optimization procedure of Section 3.1 into two steps. We first solve the unconstrained Dynamics Equation (Equation 7) as we do for unconstrained snakes. We then calculate the component of the snake step vector—the difference between the snake's current state and its previous one—that is perpendicular to the constraint surface and subtract it from the state vector. The first step regularizes, while the second prevents the snake from moving too far away from the constraint surface.

As in the case of unconstrained snakes, α, the viscosity term of Equation 7, is computed automatically at the start of the optimization and progressively increased as needed to ensure a monotonic decrease of the snake's energy and ultimate convergence of the algorithm.

Let S be the snake's state vector as described in Sections 2.2 and 2.4. An iteration of the optimization procedure involves the following three steps:

1. Take a Newton step to project S_{t-1}, the current state vector, onto the constraint surface.

   $$S_{t-1} \leftarrow S_{t-1} + AdV \text{ where } A^T AdV = -C(S_{t-1}) \ .$$

   If the snake's total energy has increased, back up and increase viscosity.
2. Take a normal snake step by solving

   $$(K_S + \alpha I)S_t \quad = \quad \alpha S_{t-1} - \left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}}$$

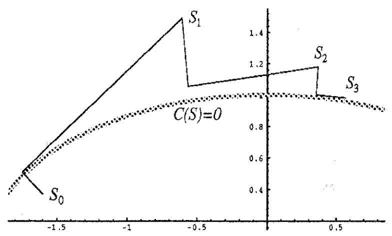

Figure 7: Constrained optimization. Minimizing $(x-0.5)^2+(y-0.2)^2$ under the constraint that $(x/2)^2+y^2=1$. The set of all states that satisfy the constraint $C(S)=0$, i.e. the constraint surface, is shown as a thick gray line. Each iteration consists of two steps: orthognal projection onto the constraint surface followed by a line search in a direction tangent to the surface. Because we perform only one Newton step at each iteration, the constraint is fully enforced after only a few iterations.

3. Ensure that dS, the snake step from S_{t-1} to S_t, is in the subspace tangent to the constraint surface.

$$S_t \leftarrow S_t - A\lambda \text{ where } A^tA\lambda = A^T(S_t - S_{t-1}) \ ,$$

so that the snake step dS becomes

$$\begin{aligned} dS &= (S_t - A\lambda) - S_{t-1} \\ \Rightarrow A^T dS &= 0. \end{aligned}$$

3.3 Multiple Snakes

Our technique can be further generalized to the simultaneous optimization of several snakes under a set of constraints that bind them. We concatenate the state vectors of the snakes into a composite state vector S and compute for each snake the viscosity coefficient that would yield steps of the appropriate magnitude if each snake was optimized individually. The optimization steps become

1. Project S onto the constraint surface as before and compute the energy of each individual snake. For all snakes whose energy has increased, revert to the previous position and increase the viscosity.
2. Take a normal snake step for each snake individually.
3. Project the global step into the subspace tangent to the constraint surface.

Because the snake steps are taken individually, we never have to solve the potentially very large linear system involving all the state variables of the composite snake but only the smaller individual linear systems. Furthermore, to control the snake's convergence via the progressive viscosity increase, we do not need to sum the individual energy terms. This is especially important when simultaneously optimizing objects of a different nature, such as a surface and a linear feature, whose energies are unlikely to be commensurate so that the sum of these energies would be essentially meaningless.

In effect, the optimization technique proposed here is a decomposition method and such methods are known to work well [Gill *et al.*, 1981] when their individual components, the individual snake optimizations, are well behaved, which is the case here.

4 CONSISTENT SITE MODELING

We demonstrate the ability of our technique to impose geometric constraints on 2–D and 3–D deformable models using real imagery. More specifically, we address the issue of optimizing the models of 3–D linear features such as roads, ridgelines, rivers, and the terrain on which they lie under the constraint that they be consistent with one another. In Figures 1 and 8 we present two such cases where recovering the terrain and the roads independently of one another leads to inconsistencies.

Because we represent the terrain as a triangulated mesh and the features as 3–D polygonal approximations, consistency can be enforced as follows. For each edge $((x_1,y_1,z_1),(x_2,y_2,z_2))$ of the terrain mesh and each segment $((x_3,y_3,z_3),(x_4,y_4,z_4))$ of a linear feature that intersect when projected in the (x,y) plane, the four endpoints must be coplanar so that the segments also intersect in 3–D space. This can expressed as

$$\begin{vmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{vmatrix} = 0 \ , \tag{17}$$

which yields a set of constraints that we refer to as *consistency constraints*.

In Figures 2 and 9, we show that the optimization under the constraints of Equation 17 avoids the discrepancies that result from independent optimization of each feature.

In the example of Figure 2, the "ridge snake" attempts to maximize the average edge gradient along its projections in all three images. In the case of Figures 8 and 9, the roads are lighter than the surrounding terrain. At low resolution, they can effectively be modeled as white lines, and the corresponding snakes attempt to maximize image intensity along their projections. At higher resolution, they are better modeled using the 3–D ribbon snakes of Section 2.2. We also introduce a building and use its base to further constrain the terrain. Figures 9(a,b) depict the result of the simultaneous optimization of the terrain and low-resolution roads. By supplying an average width for the roads, we can turn the lines into ribbons and reoptimize terrain and features under the same consistency constraints as before, yielding the result of Figure 9(c).

The case of rivers is somewhat more complex. Like roads, rivers are represented as linear features that must lie on the terrain. But, in addition, the system must ensure that they flow downhill and at the bottom of valleys. By introducing the active set strategy described at the end of Section 3.1, we have been able to impose such constraints and to generate the more complete site model of of Figure 10.

These examples illustrate the ability of our approach to model

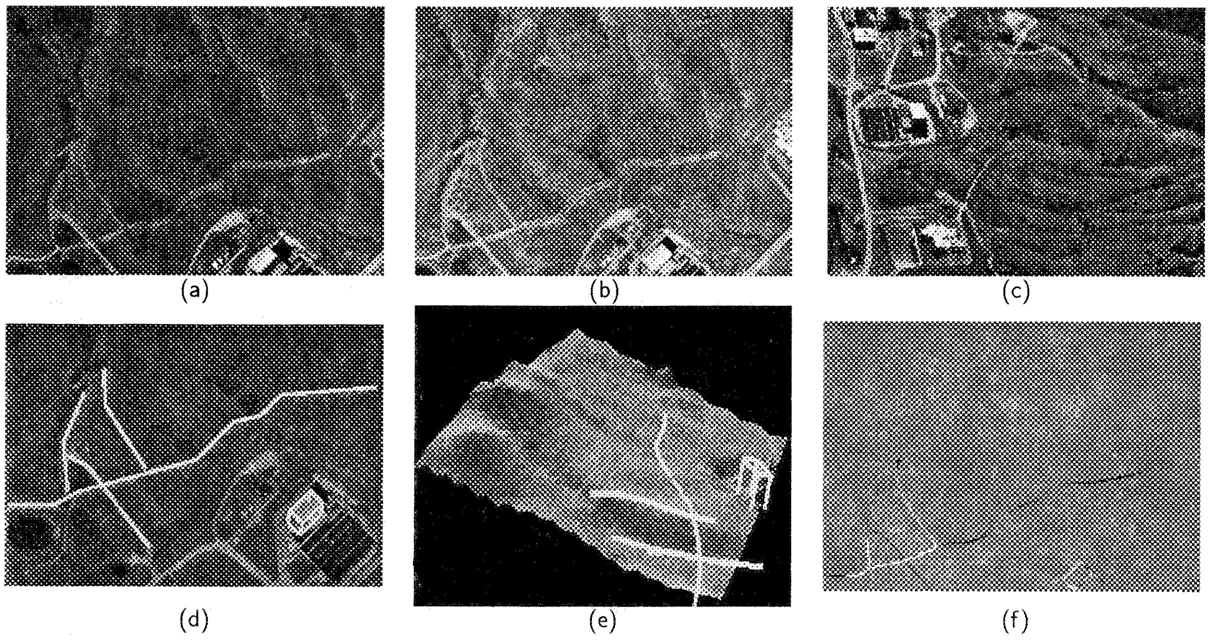

Figure 8: Building a site model. (a,b,c) Three images of a site with roads and buildings. (d) A rough sketch of the road network and of one of the buildings. (e) Shaded view of the terrain with overlaid roads after independent optimization of each. Note that the two roads in the lower right corner appear to be superposed in this projection because their recovered elevations are inaccurate. (f) Differences of elevation between the optimized roads and the underlying terrain. The image is stretched so that black and white represent errors of minus and plus 5 meters, respectively.

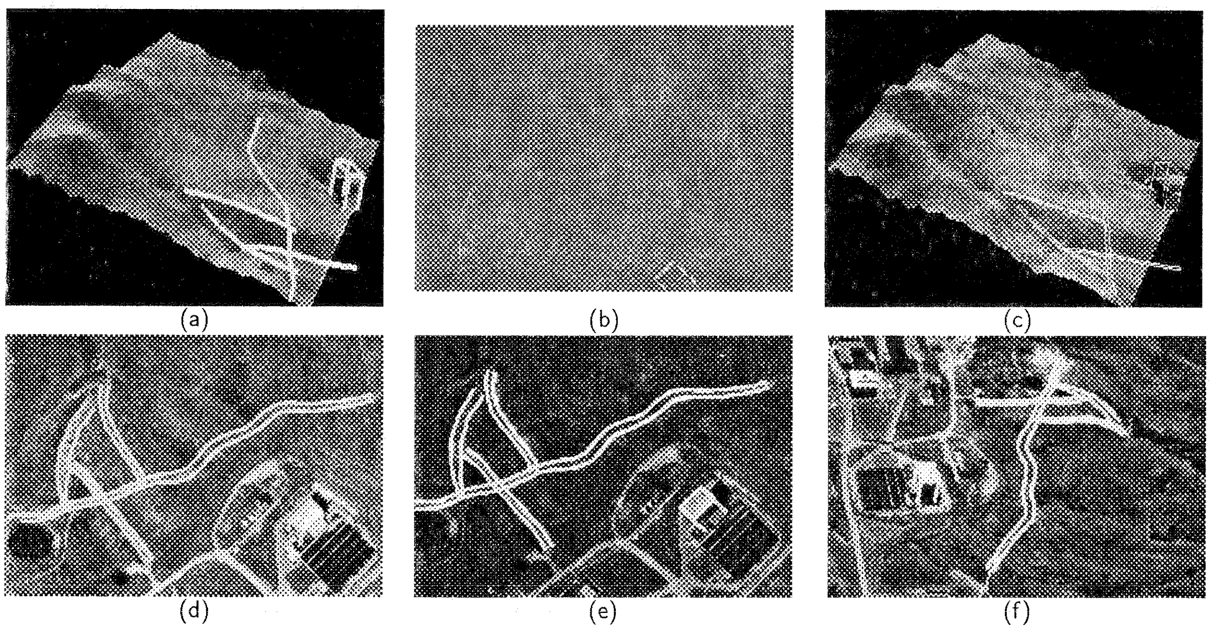

Figure 9: Recovering the 3–D geometry of both terrain and roads. (a) Shaded view of the terrain with overlaid low-resolution roads after optimization under consistency constraints. (b) Corresponding differences of elevation between features and underlying terrain. The image is stretched like the one of Figure 8(f). Note that only the roof of the building is significantly above the terrain. (c) The roads modeled as ribbons overlaid on the terrain. (d,e,f) The optimized roads overlaid on the original images.

different kinds of features in a common reference framework and to produce consistent composite models.

5 CONCLUSION

We have presented object modeling techniques for 2–D and 3–D linear features and 3–D surfaces that rely on parametric models and are extensions of traditional snakes. We have

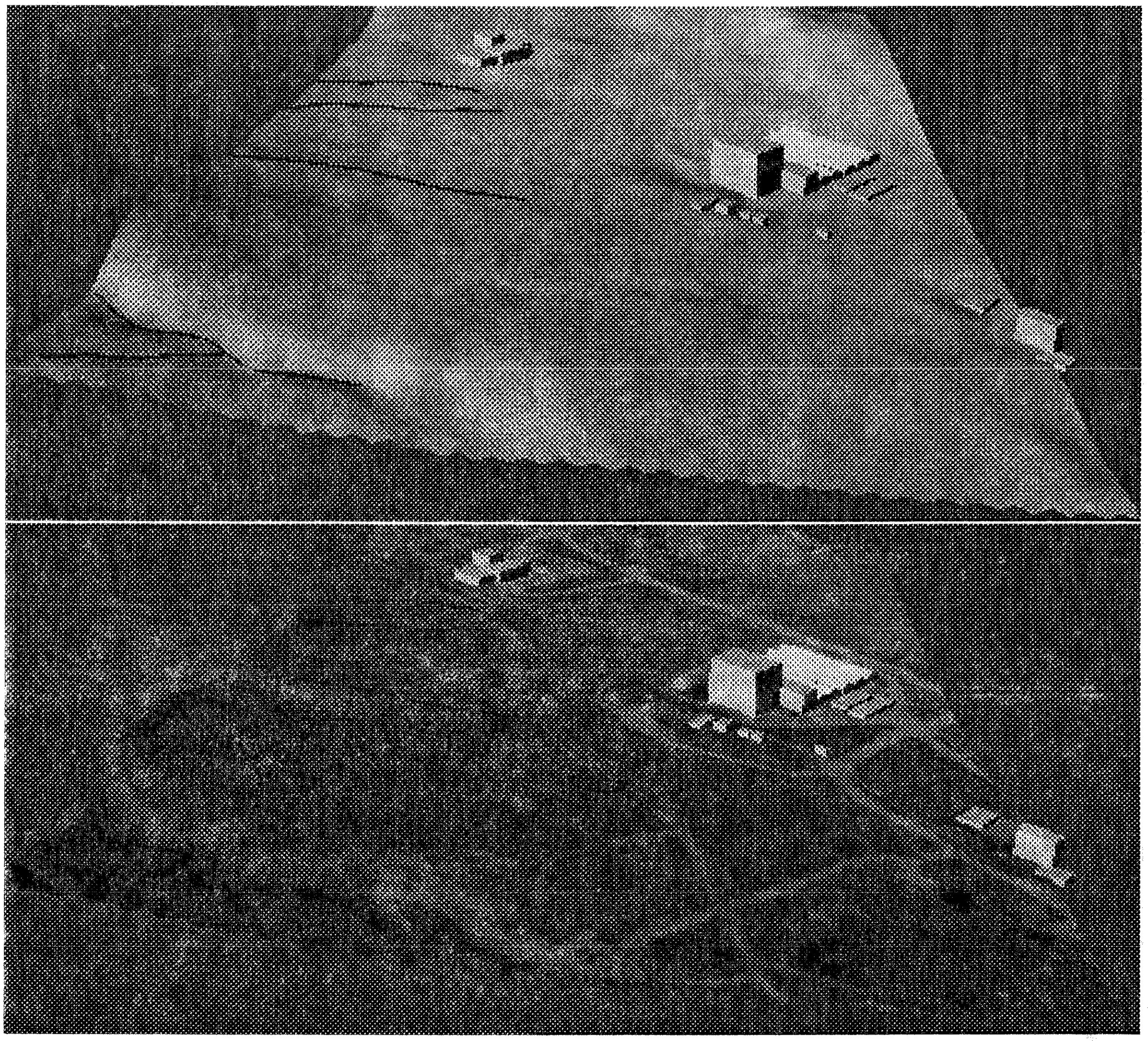

Figure 10: Composite model. Shaded and texture mapped views of an area of the site of Figure 8 including rivers, shown as black lines, and buildings.

shown that, using a constrained optimization method that allows us to enforce hard constraints on these deformable models at a low computational cost, we can generate consistent models of complex sites.

We believe that this last capability will prove indispensable to automating the generation of complex object databases from imagery, such as the ones required for realistic simulations or intelligence analysis. In such databases, the models must not only be as accurate—that is, true to the data—as possible but also consistent with each other. Otherwise, the simulation will exhibit "glitches," and the image analyst will have difficulty interpreting the models. Because our approach can handle nonlinear constraints, in future work we will use it to implement more sophisticated constraints than the simple geometric constraints presented here. When modeling natural objects, we intend to take physical laws into account. For example, rivers flow downhill and at the bottom of valleys; these characteristics should be used when modeling both the river and the surrounding terrain. In addition, when modeling man-made objects, we intend to take advantage of knowledge about construction practices, such as the fact that roads do not have arbitrary slopes.

We hope that the technique presented in this paper will eventually form the basis for a suite of tools for modeling complex scenes accurately while ensuring that the model components satisfy geometric and semantic constraints and are consistent with each other.

Acknowledgments

Support for this research was provided by various contracts from the Advanced Research Projects Agency. We wish to thank Tom Strat and Yvan Leclerc for their support.

REFERENCES

[Amini *et al.*, 1988] A.A. Amini, S. Tehrani, and T.E. Weymouth. Using Dynamic Programming for Minimizing the Energy of Active Contours in the Presence of Hard Constraints. In *International Conference on Computer Vision*, pages 95–99, 1988.

[Baltsavias, 1991] E. P. Baltsavias. *Multiphoto Geometrically Constrained Matching*. PhD thesis, Institute for Geodesy and Photgrammetry, ETH Zurich, December 1991.

[Brechbühler *et al.*, 1995] C. Brechbühler, G. Gerig, and O. Kübler. Parametrization of Closed Surfaces for 3-D Shape Description. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 61(2):154–170, March 1995.

[Devernay and Faugeras, 1994] F. Devernay and O. D. Faugeras. Computing Differential Properties of 3–D Shapes from Stereoscopic Images without 3–D Models. In *Conference on Computer Vision and Pattern Recognition*, pages 208–213, Seattle, WA, June 1994.

[Fletcher, 1987] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore, 2nd edition, 1987. A Wiley-Interscience Publication.

[Fua and Brechbuhler, 1996] P. Fua and C. Brechbuhler. Imposing Hard Constraints on Soft Snakes. In *European Conference on Computer Vision*, Cambridge, England, April 1996. Available as Tech Note 553, Artificial Intelligence Center, SRI International.

[Fua and Leclerc, 1990] P. Fua and Y. G. Leclerc. Model Driven Edge Detection. *Machine Vision and Applications*, 3:45–56, 1990.

[Fua and Leclerc, 1995] P. Fua and Y. G. Leclerc. Object-Centered Surface Reconstruction: Combining Multi-Image Stereo and Shading. *International Journal of Computer Vision*, 16:35–56, September 1995.

[Fua and Sander, 1992] P. Fua and P. Sander. Segmenting Unstructured 3D Points into Surfaces. In *European Conference on Computer Vision*, pages 676–680, Genoa, Italy, April 1992.

[Gill *et al.*, 1981] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, London a.o., 1981.

[Heipke, 1992] C. Heipke. Integration of Digital Image Matching and Multi Image Shape From Shading. In *International Society for Photogrammetry and Remote Sensing*, pages 832–841, Washington D.C., 1992.

[Kanade and Okutomi, 1990] T. Kanade and M. Okutomi. A Stereo Matching Algorithm with an Adaptative Window: Theory and Experiment. In *ARPA Image Understanding Workshop*, September 1990.

[Kass *et al.*, 1988] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[Leclerc and Bobick, 1991] Y. G. Leclerc and A. F. Bobick. The Direct Computation of Height from Shading. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Lahaina, Maui, Hawaii, June 1991.

[Metaxas and Terzopoulos, 1991] D. Metaxas and D. Terzopoulos. Shape and Nonrigid Motion Estimation through Physics-Based Synthesis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):580–591, 1991.

[Mundy *et al.*, 1992] J.L. Mundy, R. Welty, L. Quam, T. Strat, W. Bremmer, M. Horwedel, D. Hackett, and A. Hoogs. The RADIUS Common Development Environment. In *ARPA Image Understanding Workshop*, pages 215–226, 1992.

[Nishihara, 1984] H.K. Nishihara. Practical Real-Time Imaging Stereo Matcher. *Optical Engineering*, 23(5), 1984.

[Quam, 1984] L.H. Quam. Hierarchical Warp Stereo. In *ARPA Image Understanding Workshop*, pages 149–155, 1984.

[Rosen, 1961] Rosen. Gradient Projection Method for Nonlinear Programming. *SIAM Journal of Applied Mathematics*, 8:181–217, 1961.

[Szeliski and Tonnesen, 1992] R. Szeliski and D. Tonnesen. Surface Modeling with Oriented Particle Systems. In *Computer Graphics (SIGGRAPH)*, volume 26, pages 185–194, July 1992.

[Terzopoulos *et al.*, 1987] D. Terzopoulos, A. Witkin, and M. Kass. Symmetry-seeking Models and 3D Object Reconstruction. *International Journal of Computer Vision*, 1:211–221, 1987.

[Wrobel, 1991] B. P Wrobel. The evolution of Digital Photogrammetry from Analytical Phogrammetry. *Photogrammetric Record*, 13(77):765–776, April 1991.