Imposing Hard Constraints on Soft Snakes

P. Fua[1]* and C. Brechbühler[2]

[1] SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025 fua@ai.sri.com
[2] ETH-Zürich, Gloriastr. 35, CH-8092 Zürich, Switzerland, brech@vision.ee.ethz.ch

Abstract. An approach is presented for imposing generic hard constraints on deformable models at a low computational cost, while preserving the good convergence properties of snake-like models. We believe this capability to be essential not only for the accurate modeling of individual objects that obey known geometric and semantic constraints but also for the consistent modeling of sets of objects.

Many of the approaches to this problem that have appeared in the vision literature rely on adding penalty terms to the objective functions. They rapidly become untractable when the number of constraints increases. Applied mathematicians have developed powerful constrained optimization algorithms that, in theory, can address this problem. However, these algorithms typically do not take advantage of the specific properties of snakes. We have therefore designed a new algorithm that is tailored to accommodate the particular brand of deformable models used in the Image Understanding community.

We demonstrate the validity of our approach first in two dimensions using synthetic images and then in three dimensions using real aerial images to simultaneously model terrain, roads, and ridgelines under consistency constraints.

1 Introduction

We propose an approach to imposing generic hard constraints on "snake-like" deformable models [9] while both preserving the good convergence properties of snakes and avoiding having to solve large and ill-conditioned linear systems.

The ability to apply such constraints is essential for the accurate modeling of complex objects that obey known geometric and semantic constraints. Furthermore, 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.

* This work was supported in part by contracts from the Advanced Research Projects Agency.

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 [4, 8]: 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 that have proved so successful for feature delineation and surface modeling. Exceptions are the approach proposed by Metaxas and Terzopoulos [10] to enforce holonomic constraints by modeling the second order dynamics of the system and the technique proposed by Amini *et al.* [1] using dynamic programming.

In this work we propose a new approach to enforcing hard-constraints on deformable models without undue computational burden while retaining their desirable convergence properties. Given a deformable model, the state vector that defines its shape, an objective function to be minimized and a set of constraints to be satisfied, each iteration of the optimization performs two steps:

- Orthogonally project the current state toward the constraint surface, that is the set of all states that satisfy the constraints.
- Minimize the objective function in a direction that belongs to the subspace that is tangent to the constraint surface.

This can be achieved by solving small linear systems. This algorithm is closely related to the two-phase algorithm proposed by Rosen [11] and is an extension of a technique developed in [2]. The corresponding procedure is straightforward and easy to implement. Furthermore, this approach remains in the spirit of most deformable model approaches: they can also be seen as performing two steps, one attempting to fit the data and the other to enforce global constraints [3].

We view our contribution as the design of a very simple and effective constrained-optimization technique that allows the imposition of hard constraints on deformable models at a very low computational cost.

We first present the generic constrained optimization algorithm that forms the basis of our approach. We then specialize it to handle snake-like optimization. Finally, we demonstrate its ability to enforce geometric constraints upon individual snakes and consistency constraints upon multiple snakes.

2 Constrained Optimization

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,

$$\text{minimize } f(S) \text{ subject to } C(S) = 0 \tag{1}$$

While there are many powerful methods for nonlinear constrained minimization [8], 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 [2] that is closely related to gradient projection methods first proposed by Rosen [11] and can be extended to snake optimization.

2.1 Constrained Optimization in Orthogonal Subspaces

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 tangent subspace 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 1 depicts this procedure. Let C and S be the constraint and state vectors of Equation 1 and A be the $n \times m$ Jacobian matrix of the constraints. The two steps are implemented as follows:

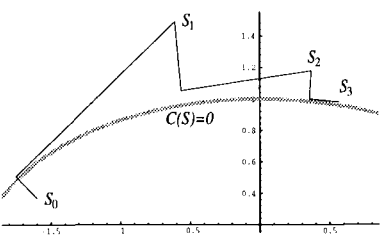


Fig. 1. Constrained optimization. Minimizing $(x - 0.5)^2 + (y - 0.2)^2$ under the constraint that $(x/2)^2 + y^2 = 1$. The set of all states that satisfy the constraint $C(S) = 0$, i.e. the constraint surface, is shown as a thick gray line. Each iteration consists of two steps: orthognal projection 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 only after a few iterations

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.

3 Snake Optimization

We first introduce our notations and briefly review traditional "snake-like" optimization [9]. We then show how it can be augmented to accommodate our constrained-optimization algorithm to impose hard constraints on single and multiple snakes.

3.1 Unconstrained Snake Optimization

In our work, we take 2–D features to be outlines that can be recovered from a single 2–D image while we treat 3–D features as objects whose properties are computed by projecting them into several 2–D images. We model 2–D and 3–D linear features as polygonal curves and 3–D surfaces as hexagonally connected triangulations.

We will 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.*

We recover a model's shape by minimizing an objective function $\mathcal{E}(S)$ that embodies the image-based information. For 2–D linear features, $\mathcal{E}(S)$ is the average value of the edge gradient along the curve. For 3–D linear features, $\mathcal{E}(S)$ is computed by projecting the curve into a number of images, computing the average edge-gradient value for each projection and summing these values [5]. For 3–D surfaces, we use an objective function that is the sum of a stereo term and a shape-from-shading term. [7].

In all these cases, $\mathcal{E}(S)$ typically is a highly nonconvex function, and therefore difficult to optimize. However, it can effectively be minimized [9] by

- introducing a quadratic regularization term $\mathcal{E}_D = 1/2S^tK_SS$ where K_S is a sparse stiffness matrix,
- defining the total energy $\mathcal{E}_T = \mathcal{E}_D(S) + \mathcal{E}(S) = 1/2S^tK_SS + \mathcal{E}(S)$,
- embedding the curve in a viscous medium and iteratively solving the dynamics equation $\frac{\partial \mathcal{E}_T}{\partial S} + \alpha\frac{dS}{dt} = 0$, where α is the viscosity of the medium.

Because $\mathcal{E}_D$ is quadratic, the dynamics equation can be rewritten as

$$K_SS_t + \left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} + \alpha(S_t - S_{t-1}) = 0 \Rightarrow (K_S + \alpha I)S_t = \alpha S_{t-1} - \left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} . \quad (2)$$

In practice, α is computed automatically at the start of the optimization procedure so that a prespecified average vertex motion amplitude is achieved [6]. The optimization proceeds as long as the total energy decreases. When it increases, the algorithm backtracks and increases α, thereby decreasing the step size. In the remainder of the paper, we will refer to the vector $dS_t = S_t - S_{t-1}$ as the "snake step" taken at iteration t.

In effect, this optimization method performs implicit Euler steps with respect to the regularization term [9] and is therefore more effective at propagating smoothness constraints across the surface than an explicit method such as conjugate gradient. It is this property that our constrained-optimization algorithm strives to preserve.

3.2 Constraining the Optimization

Given a set of m hard constraints $C(S) = \{c_1, c_2, .., c_m\}$ that the snake must satisfy, we could trivially extend the technique of Section 2 by taking the objective function f to be the total energy $\mathcal{E}_T$. However, this would be equivalent to optimizing an unconstrained snake 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.

Therefore, for the purpose of optimizing constrained snakes, we decompose the second step of the optimization procedure of Section 2 into two steps. We first solve the unconstrained Dynamics Equation (Equation 2) 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 2, 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.

An iteration of the optimization procedure therefore 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

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

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

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

so that the snake step dS becomes $dS = (S_t - A\lambda) - S_{t-1} \Rightarrow A^T dS = 0$.

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 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 [8] when their individual components, the individual snake optimizations, are well behaved, which is the case here.

4 Results

We demonstrate the ability of our technique to impose geometric constraints on 2–D and 3–D deformable models using real imagery.

4.1 2–D Features

To illustrate the convergence properties of our algorithm, we introduce two simple sets of constraints that can be imposed on 2–D snakes. The most obvious one forces the snake to go through a specific point (a_0, b_0). It can be written as the two constraints

$$x_i - a_0 = y_i - b_0 = 0 \ , \tag{3}$$

where i is the index of the snake vertex that is closest to (a_0, b_0) at the beginning of an iteration. In practice, the constraint always remains "attached" to the vertex that was closest initially and we refer to this constraint as an "attractor constraint." A slightly more sophisticated set of constraints achieves a similar purpose while allowing the point at which the snake is attached to slide. It is designed to force the snake to be tangent to a segment $((a_0, b_0), (a_1, b_1))$, and we will refer to it as a "tangent constraint." It can also be written as a set of two constraints

$$\begin{vmatrix} x_i & a_0 & a_1 \\ y_i & b_0 & b_1 \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} x_{i+1} - x_{i-1} & a_1 - a_0 \\ y_{i+1} - y_{i-1} & b_1 - b_0 \end{vmatrix} = 0 \tag{4}$$

where i in the index of the snake vertex that is both closest to the line segment and between the endpoints at the beginning of an iteration. The first constraint ensures that (x_i, y_i), (a_0, b_0), and (a_1, b_1) are collinear. The second ensures that the finite-difference estimate of the tangent vector is parallel to the segment's direction. The vertex at which the constraint is attached can slide along the segment and can slide off its edges so that a different vertex may become attached.

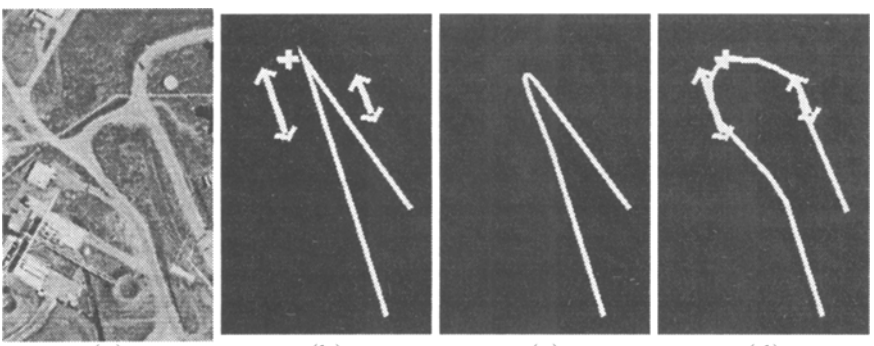


Fig. 2. Modeling the edge of a road. (a) Aerial image of a set of roads. (b) A very rough approximation of one of the road's edges and a set of constraints. The two-sided arrows represent tangent constraints (Equation 4) while the crosshair depicts an attractor constraint (Equation 3). (c) The result of unconstrained snake optimization. (d) The result of constrained snake optimization using the constraints depicted by (b).

Figure 2(b) depicts the very rough outline of the edge of a road. The outline is too far from the actual contour for a conventional snake to converge toward the edge. However, using two of the tangent constraints of Equation 4 and one of the attractor constraints of Equation 3, we can force convergence toward the desired edge.

To demonstrate the behavior of the multiple snake optimization, we also introduce a "distance" constraint between two snakes. Given a vector of length d, such as the ones depicted by arrows in Figure 3(a) and two snakes, let (x_i^1, y_i^1, z_i^1) and (x_j^2, y_j^2, z_j^2) be the vertices of each snake that are closest to the vector's

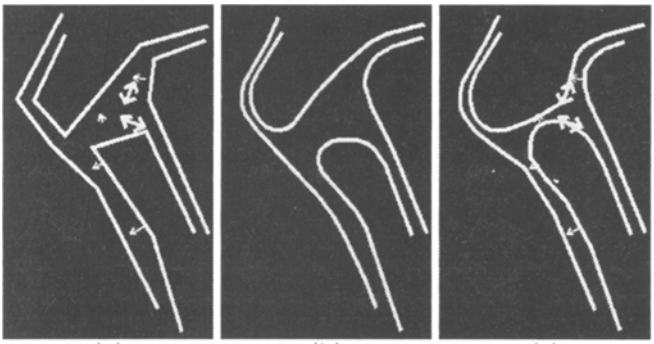


Fig. 3. Modeling a set of road edges. (a) A set of three contours roughly approximating the edges of the main roads and a set of constraints. As before, the two-sided arrows represent "tangent constraints" (Equation 4) that apply to individual contours, while the thinner one-sided arrows represent distance constraints (Equation 5) that bind pairs of contours. (b) The result of unconstrained snake optimization. (c) The result of constrained snake optimization.

endpoints. The distance constraint can then be written as

$$(x_i^1 - x_j^2)^2 + (y_i^1 - y_j^2)^2 + (z_i^1 - z_j^2)^2 - d^2 = 0 \ . \tag{5}$$

Using the same images as before, we can model the main road edges starting with the three rough approximations shown in Figure 3(a). Here again, these initial contours are too far away from the desired answer for unconstrained optimization to succeed. To enforce convergence toward the desired answer, in addition to the unary constraints—that is, constraints that apply to individual snakes—of the previous example, we can introduce binary constraints—that is, constraints that tie pairs of snakes—and optimize the three contours simultaneously. The binary constraints we use are the distance constraints of Equation 5.

In both of these examples, we were able to mix and match constraints of different types as needed to achieve the desired result without having to worry about weighting them adequately. Furthermore the algorithm exhibits good convergence properties even though the constraints are not linear but quadratic.

4.2 3–D Features

We now turn to the simultaneous optimization of 3–D surfaces and 3–D features. More specifically, we address the issue of optimizing the models of 3–D linear features such as roads and ridgelines and the terrain on which they lie under the constraint that they be consistent with one another. In Figures 4 and 5 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

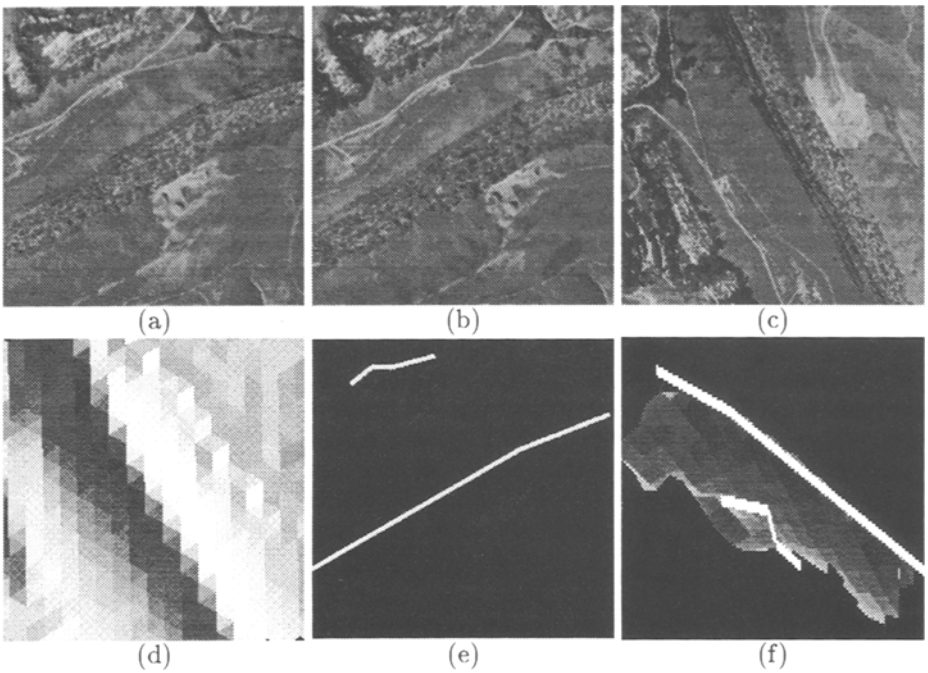


Fig. 4. Rugged terrain with sharp ridge lines. (a,b,c) Three images of a mountainous site. (d) Shaded view of an initial terrain estimate. (e) Rough polygonal approximation of the ridgelines overlaid on image (a). (f) The terrain and ridgeline estimates viewed from the side (the scale in z has been exaggerated).

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

$$\begin{vmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{vmatrix} = 0 \ , \tag{6}$$

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

In both examples shown here, we follow a standard coarse-to-fine strategy. We start with a rough estimate of both terrain and features and reduced versions of the images. We then progressively increase the resolution of the images being used and refine the discretization of our deformable models. In Figures 6 and 7, we show that the optimization under the constraints of Equation 6 avoids the discrepancies that result from independent optimization of each feature.

In the example of Figure 6, the "ridge-snake" attempts to maximize the average edge gradient along its projections in all three images. In the case of Figures 5 and 7 the roads are lighter than the surrounding terrain. At low resolution, they

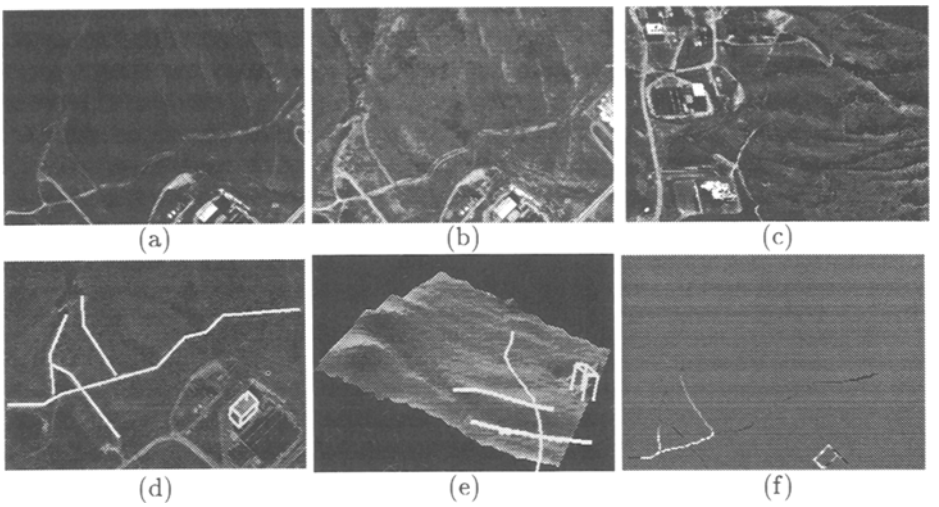


Fig. 5. Building a site model. (a,b,c) Three images of a site with roads and buildings. (d) A rough sketch of the road network and of one of the buildings. (e) Shaded view of the terrain with overlaid roads after independent optimization of each. Note that the two roads in the lower right corner appear 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.

can effectively be modeled as white lines, and the corresponding snakes attempt to maximize image intensity along their projections. We also introduce a building and use its base to further constrain the terrain. Figures 7(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 7(c).

These two examples illustrate the ability of our approach to model different kinds of features in a common reference framework and to produce consistent composite models.

5 Conclusion

We have presented a constrained optimization method that allows us to enforce hard constraints on deformable models at a low computational cost, while preserving the convergence properties of snake-like approaches. We have shown that it can effectively constrain the behavior of linear 2–D and 3–D snakes as well as that of surface models. Furthermore, we have been able to use our technique to simultaneously optimize several models while enforcing consistency constraints between them.

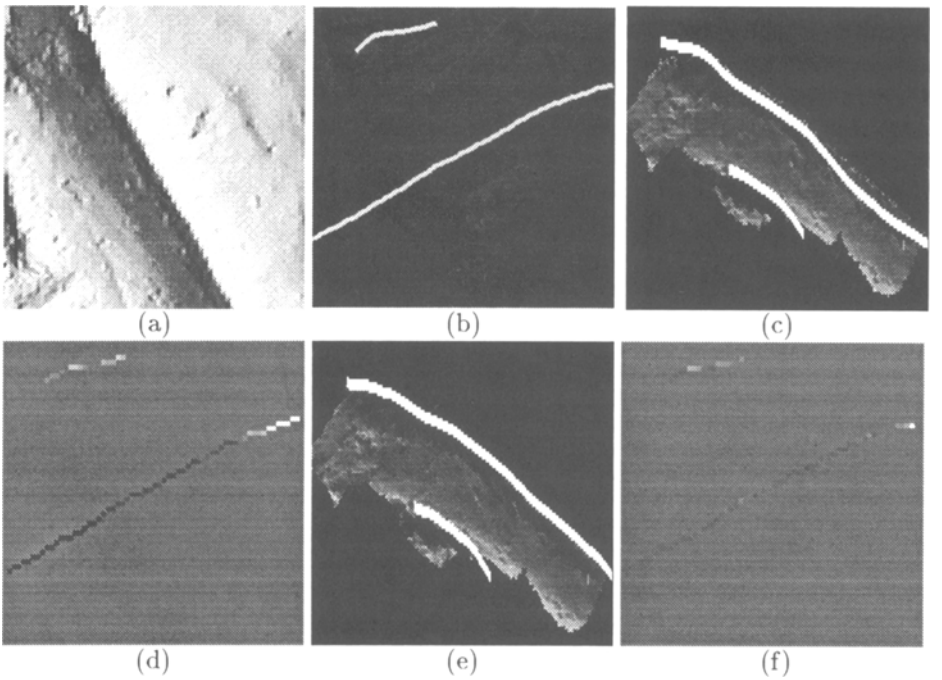
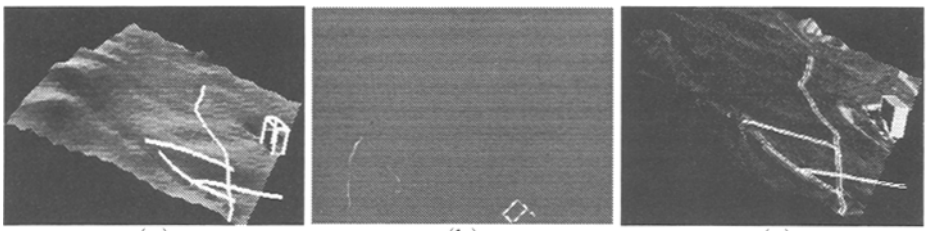


Fig. 6. Recovering the 3–D geometry of both terrain and ridges. (a) Shaded view of the terrain after refinement. (b) Refined ridgeline after 3–D optimization. (c) Side view of the ridgeline and terrain after independent optimization of each one. Note that the shape of the ridgeline does not exactly match that of the terrain. (d) Differences of elevation between the recovered ridge-line 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).

We believe that these last 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. 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; this 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.



(a) (b) (c)

Fig. 7. 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 as the one of Figure 5(f). Only the building's roof is significantly above the terrain. (c) Synthetic view of the site with the roads modeled as ribbons overlaid on the image.

Eventually, we hope that the technique presented in this paper will 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.

References

1. 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.
2. 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.
3. L. Cohen. Auxiliary Variables for Deformable Models. In *International Conference on Computer Vision*, pages 975–980, Cambridge, MA, June 1995.
4. R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore, 2nd edition, 1987. A Wiley-Interscience Publication.
5. P. Fua. Parametric Models are Versatile: The Case of Model Based Optimization. In *ISPRS WG III/2 Joint Workshop*, Stockholm, Sweden, September 1995.
6. P. Fua and Y. G. Leclerc. Model Driven Edge Detection. *Machine Vision and Applications*, 3:45–56, 1990.
7. 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.
8. P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization.* Academic Press, London a.o., 1981.
9. M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. *International Journal of Computer Vision*, 1(4):321–331, 1988.
10. D. Metaxas and D. Terzopoulos. Shape and Norigid Motion Estimation through Physics-Based Synthesis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):580–591, 1991.
11. Rosen. Gradient projection method for nonlinear programming. *SIAM Journal of Applied Mathematics*, 8:181–217, 1961.