Imposing Hard Constraints on Soft Snakes

P. Fua[*] and C. Brechbühler[†]

[*] Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025, USA
fua@ai.sri.com

[†] Communication Technology Lab.
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 closely related to Lagrangian methods but 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.

Keywords : Snakes, Deformable models, Constrained optimization, Consistency.

1 Introduction

We propose an approach to imposing generic hard constraints on "snake-like" deformable models [Terzopoulos *et al.*, 1987, Kass *et al.*, 1988] while both preserving the good convergence properties of snakes and avoiding having to solve large and ill-conditioned linear systems of equations.

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.

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—such as the attractive or repulsive forces that are often attached to the mouse cursor to guide snake optimization—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 that have proved so successful for feature delineation and surface modeling. For example, Sequential Quadratic Programming (SQP) [Fletcher, 1987] is widely recognized as one of the most powerful such techniques but updating the Lagrange multipliers requires solving a full $(n+m) \times (n+m)$ potentially ill-conditioned linear system—n being the number of state variables, normally much larger than m the number of constraints. It also requires the computation of the Hessian of the objective function which is hard to do when dealing with images because second derivatives of image gray values are notoriously noisy.

In the area of computer vision, one notable exception to the complexity problem is the approach proposed by Metaxas and Terzopoulos [1991] to enforce holonomic constraints[1] by modeling the second order dynamics of the system and stabilizing the constraint equations to prevent possible divergence using the Baumgarte method [Baumgarte, 1972]. Solving the system only requires dealing with matrices whose size is proportional to the number of constraints m.

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:

[1]Holonomic constraints are purely geometric constraints on a dynamic system.

- 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 algorithm is closely related to the two-phase algorithm proposed by Rosen [1978] and is an extension of a technique developed in [Brechbühler, 1995, Brechbühler *et al.*, 1995]. We will show that this can be achieved by solving mxm linear systems—where m is the number of constraints and that does not require computing the Hessian of the objective function. 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 [Cohen, 1995].

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, also known as the nonlinear equality-constrained problem (NEP), 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,

$$\begin{aligned} \text{NEP:}\quad & \text{minimize} && f(S) \\ & \text{subject to} && C(S) = 0 \ . \end{aligned} \tag{1}$$

While there are many powerful methods for nonlinear constrained minimization [Gill *et al.*, 1981, Culioli, 1994], 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. For example, Sequential Quadratic Programming (SQP) [Fletcher, 1987] is widely recognized as one of the most powerful such techniques, and we outline it in appendix. However, in our experience, it has a number of drawbacks for our specific application:

- The functions we try to optimize have severe nonconvexities. As a result, the iterations may become unstable, with rapidly diverging Lagrange multipliers and the constraints being violated ever worse. Sophisticated heuristics are required to overcome this problem. In their work, Metaxas *et al.* used the Baumgarte method with well-chosen parameters to stabilize the optimization.

- SQP requires the computation of the Hessian, which is hard to do when dealing with images: second derivatives of image gray values are notoriously noisy.
- SQP requires solving $(m+n) \times (m+n)$ linear systems of equations, which is unnecessarily large in cases where m is significantly smaller than n. In addition, these systems have zeros on their diagonals, which makes a tedious reordering of the matrix necessary for many sparse linear solvers to be able to deal with them.

For these reasons, we have developed [Brechbühler, 1995] the robust constrained optimization method described below that seems better suited to our problem.

2.1 Constrained Optimization in Orthogonal Subspaces

Solving a constrained optimization problem involves making two things happen concurrently: satisfying the constraints and minimizing the objective function. SQP attempts to do both at the same time. For our application, it has proved nore 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. This two-step approach is closely related to gradient projection methods first proposed by Rosen [1978].

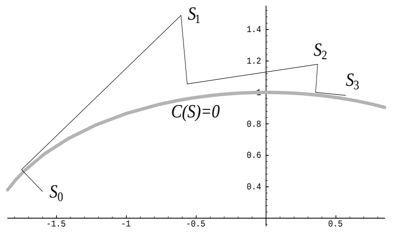

Figure 1: Constrained optimization. Minimizing $(x-0.5)^2+(y-0.2)^2$ under the constraint that $(x/2)^2+y^2=1$. The set of all states that satisfy the constraint $C(S)=0$, i.e. the constraint surface, is shown as a thick gray line. Each iteration consists of two steps: orthognal projection onto the constraint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

Projecting onto the constraint surface Let C be the constraints of Equation 1 and S be the current state. The first iteration step involves finding dS such that $C(S + dS) \approx 0$. We linearize the constraints and write

$$C(S + dS) \;\approx\; C(S) + A^T(S) \cdot dS \tag{2}$$

where A is the $n \times m$ Jacobian matrix of the constraints:

$$A = \frac{\partial C}{\partial S} = \nabla \cdot C^T = \begin{bmatrix} \frac{\partial c_1}{\partial s_1} & \cdots & \frac{\partial c_m}{\partial s_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial c_1}{\partial s_n} & \cdots & \frac{\partial c_m}{\partial s_n} \end{bmatrix} \;, \tag{3}$$

We can satisfy the constraints by taking Newton steps, that is iteratively solving the equation

$$A^T(S) \cdot dS = -C(S) \tag{4}$$

and incrementing S by dS. Equation 4, however, typically has many solutions because there are more variables than constraints. We choose the shortest possible dS, which restricts dS to the column space of A. Any component of dS orthogonal to this space would not change the validity of the solution, but would make it longer. Hence, dS can be written as $dS = AdV$, and dV is computed by solving the square symmetric positive definite $m \times m$ system

$$A^T(S)dS = A^T(S)A(S)dV = -C(S) \;. \tag{5}$$

In practice, because we go through several minimization iterations, performing a single Newton step at every iteration suffices to eventually enforce the constraints.

Minimizing the objective function Let S be the state vector after projection, $G = \nabla f$ the gradient of the objective function and A the Jacobian matrix of Equation 3. Computing G_Z, projection of G into the null space of A—that is the tangent subspace to the constraint surface—amounts to estimating Lagrange multipliers, that is, the coefficients that can be used to describe G as closely as possible as a linear combination of constraint normals. We solve the over-determined system $A\lambda = G$ in the least squares sense by solving the $m \times m$ system

$$A^T(S)A(S) \cdot \lambda = A^T(S)G \;. \tag{6}$$

$A\lambda$ is the component of G that is normal to the constraint surface, and we take G_Z to be $G - A\lambda$.

Then, f can be minimized by performing a line search in the G_Z direction. This amounts to steepest descent in the projected gradient direction. Alternatively, we can construct a new search direction in the way conjugate gradient does as a linear combination of G_Z and the previous search direction.

In short, each iteration of the optimization procedure involves the following two steps:

1. Take a Newton step to project the variables onto the constraint surface. This is achieved by solving the linear system
$$A^tAdV = -C(S)$$
and incrementing S by AdV.

2. Minimize f in a direction parallel to the projection of its gradient onto the tangent subspace to the constraint surface. To compute this direction, we first solve the linear system
$$A^T(S)A(S)\lambda = A^T(S)\nabla f$$
and take the direction to be $\nabla f - A\lambda$.

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.

2.2 Behavior of the Algorithm

We use the simple example of a chain falling under the influence of gravity to demonstrate the algorithm's behavior. The chain is modeled as a sequential list of twenty 2-D vertices $\mathcal{S} = \{(x_i, y_i),\ i = 1, \ldots, 20\}$ whose distances must remain constant. Assuming the endpoints are fixed, minimizing the chain's potential in the gravity field implies

$$\begin{array}{ll} \text{minimizing} & y_1/2 + \sum_{i=2}^{19} y_i + y_{20}/2, \\ \text{subject to} & (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \quad = \quad 0.1^2 \end{array}$$

We ran the optimization twice, once using the conjugate gradient approach to computing the search direction and once using steepest descent, that is, directly using G_Z. Stages of the optimization are shown in Figure 2. Figure 3 depicts the evolution of the objective function and squared constraint norm $\|C(S)\|^2$. Here again, because we perform only one Newton step at each iteration, the constraints are fully enforced only after a few iterations. For the first 35 iterations, steepest descent and conjugate gradient are roughly equivalent. Later, steepest descent slows down, whereas conjugate gradient reaches the final solution after about 46 iterations.

Our approach allows us to combine different kinds of constraints. To demonstrate this, we now also require the chain links to form right angles at vertices 4 and 13. We add the following two constraints:

$$\begin{array}{rcl} (\ x_4 - x_3\ )(\ x_5 - x_4\ ) + (\ y_4 - y_3\ )(\ y_5 - y_4\ ) & = & 0 \\ (x_{13} - x_{12})(x_{14} - x_{13}) + (y_{13} - y_{12})(y_{14} - y_{13}) & = & 0 \end{array}$$

Several optimization iterations are shown in Figure 4. Note that the objective function could be further reduced by flipping the corner at vertex 13 outward. However, doing so would mean temporarily violating a constraint, which our algorithm does not allow.

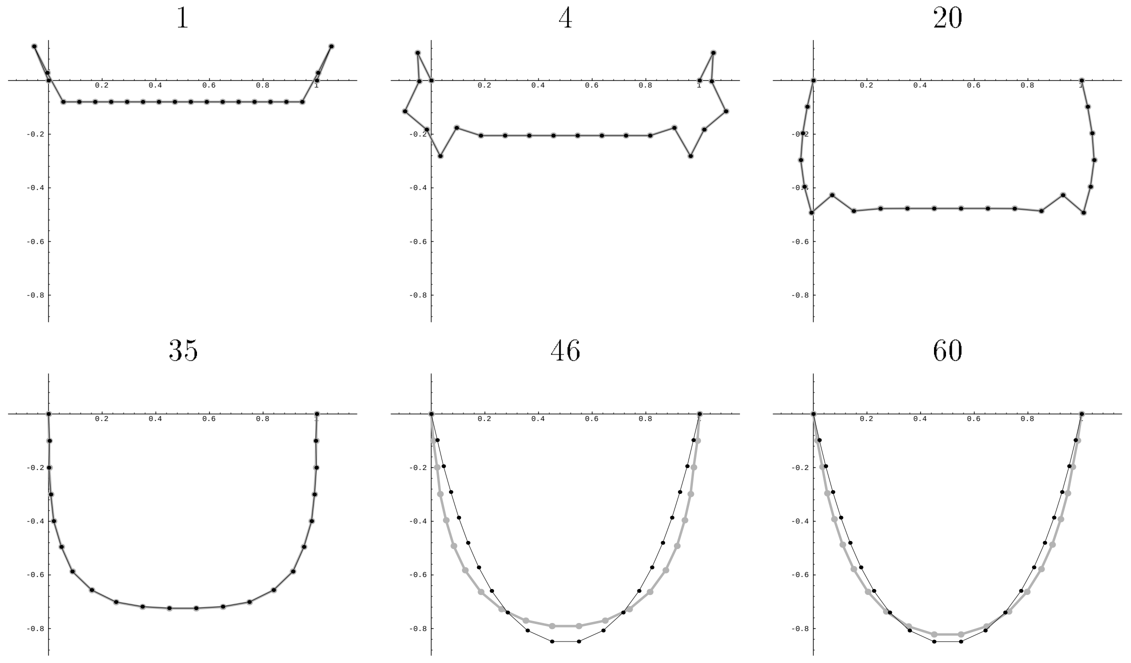

Figure 2: Minimizing the chain's potential energy while enforcing constant inter-vertex distances, using conjugate gradient (thin black chain) or steepest descent (thick gray chain): intermediate results after 1, 4, 20, 35, 46, and 60 iterations.

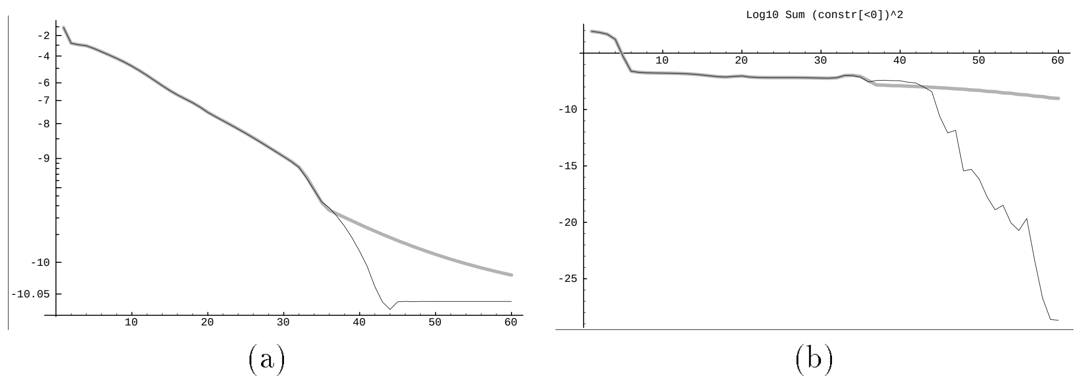

Figure 3: Evolution of the objective function and squared constraint norm. (a) Evolution of the potential energy $f(S)$ for the chain of Figure 2 while minimizing conjugate gradient (thin black line) or steepest descent (thick gray line). We use a nonlinear ordinate scale. (b) Logarithmic plot of the squared constraint norm $\|C(S)\|^2$.

By using an active set strategy, our optimization scheme can also solve inequality-constrained problems. For example, it can prevent the vertices from entering a forbidden circle and from moving

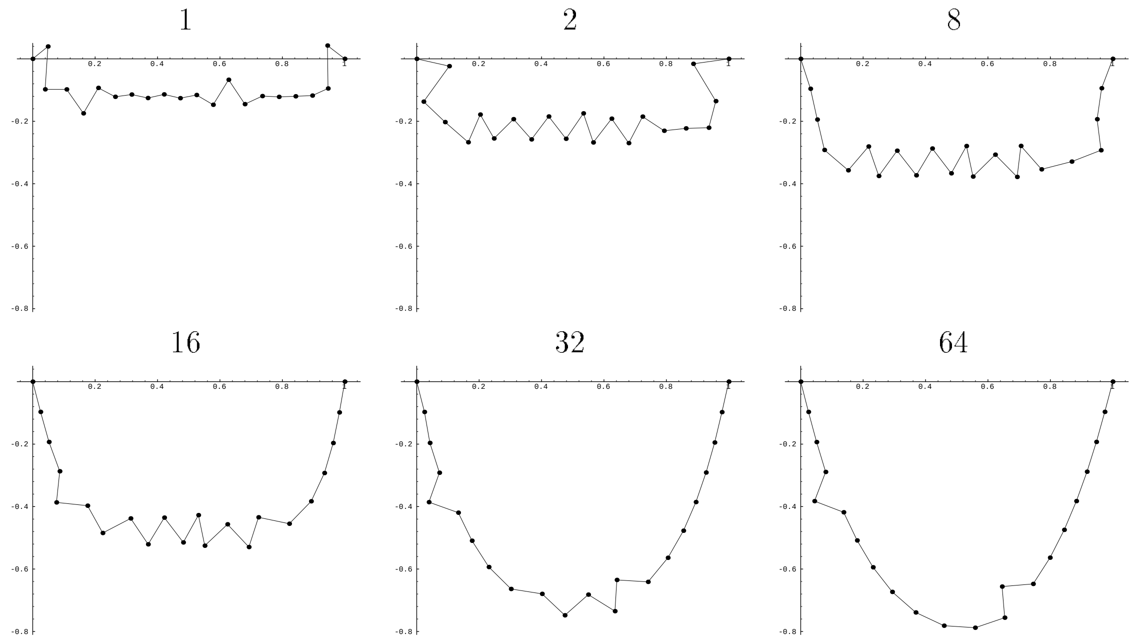

Figure 4: Chain with two "rusty" joints where the links are constrained to form 90-degree angles: intermediate results after 1, 2, 8, 16, 32, and 64 iterations.

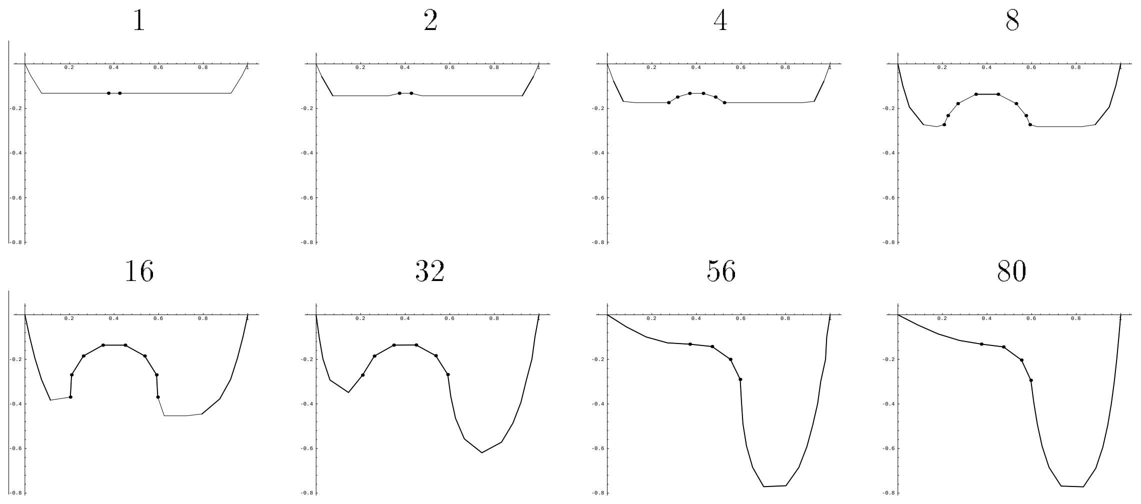

Figure 5: Minimizing the chain's potential energy under inequality constraints: intermediate results after 1, 2, 4, 8, 16, 32, 56, and 80 iterations. Active one-vertex constraints are indicated by black dots, two-vertex constraints by thick lines.

too far from their neighbors by bounding, but not fixing, the inter-vertex distance. The optimization can then be rephrased as

$$\begin{array}{lllll} \text{minimizing} & y_1/2 + \sum_{i=2}^{19} y_i + y_{20}/2 & & & \\ \text{subject to} & (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 & \leq & 0.1^2 & \text{(two-vertex constraint)} \\ & (x_i - 0.4)^2 + (y_i - 0.33)^2 & \geq & 0.2^2 & \text{(one-vertex constraint)}\ . \end{array}$$

Figure 5 depicts the optimization.

3 Snake Optimization

We first introduce our notations and briefly review traditional "snake-like" optimization [Kass *et al.*, 1988] and our own use of this technique for the modeling of what we refer to below as generalized snakes, that is, 2–D and 3–D linear features and 3–D surfaces. We then show how it can be augmented to accommodate the constrained-optimization algorithm described above and impose hard constraints on single snakes. Finally, we further extend it to allow the simultaneous and constrained optimization of 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 triangulations. More precisely, a polygonal curve, $\mathcal{C}$, is modeled as a sequential list of vertices, that is, in two dimensions, a list of n 2–D vertices $\mathcal{S}_2$ of the form

$$\mathcal{S}_2 = \{(x_i, y_i),\ i = 1, \ldots, n\}\ , \tag{7}$$

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

$$\mathcal{S}_3 = \{(x_i, y_i, z_i),\ i = 1, \ldots, n\}\ . \tag{8}$$

Similarly, we represent a surface $\mathcal{S}$ by a hexagonally connected set of 3–D vertices

$$\mathcal{S}_M = \{(x_i, y_i, z_i),\ i = 1, \ldots, n\} \tag{9}$$

called a *mesh*. Neighboring vertices are further organized into triangular planar surface elements called facets. Each vertex in the interior of the surface has exactly six neighbors, as shown in Figure 6(a).

In the remainder of the paper, we will refer to S, the vector of all x, y, and z coordinates that define the deformable model's shape as the *state vector* of the model. In practice, we take S to be the vector

$$\begin{aligned} S &= (x_1, x_2, ..., x_n, y_1, y_2, ..., y_n) \text{ in 2–D} \qquad (10) \\ S &= (x_1, x_2, ..., x_n, y_1, y_2, ..., y_n, z_1, z_2, ..., z_n) \text{ in 3–D} \end{aligned}$$

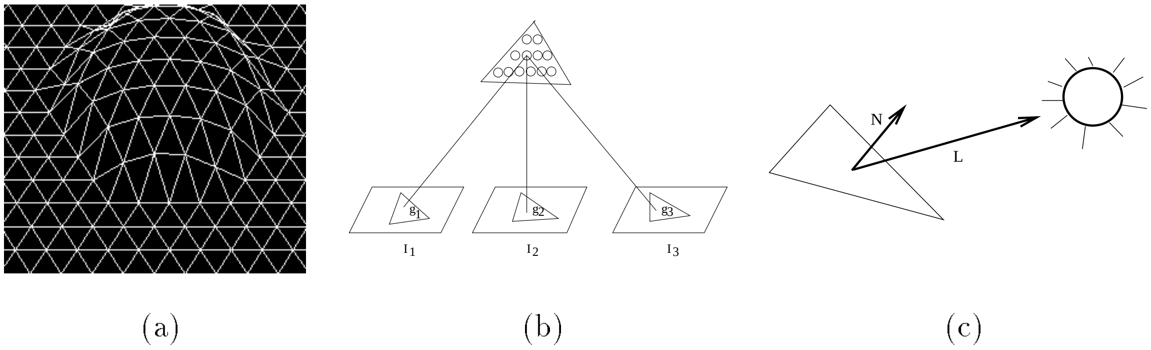

Figure 6: 3–D surfaces and energy terms. (a) A triangulated 3–D mesh, its shape depends on the x,y and z coordinates of its vertices. (b) Facets are sampled at regular intervals as illustrated here. The stereo component of the objective function is computed by summing the variance of the gray level of the projections of these sample points, the g_is. (c) The albedo of each facet is estimated using the facet normal $\overrightarrow{N}$, the light source direction $\overrightarrow{L}$, 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.

and we define the vectors X, Y, and Z as the vectors of all x, y and z coordinates, respectively.

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 [Fua, 1995].

For 3–D surfaces, we use an objective function that is the sum of a stereo term and a shape-from-shading term. As their behavior and implementation have already been extensively discussed elsewhere, we only describe them briefly and refer the interested reader to our previous publication [Fua and Leclerc, 1995]. 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. As shown in Figure 6(b), this comparison is done for a uniform 3–D 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 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 can take the complete objective function, $\mathcal{E}(S)$, to be a weighted average of these two components where the weighting is a function of texture within the projections of individual facets. However, to generate the results shown in Section 4, we have used only the stereo component of the objective function.

In all these cases, $\mathcal{E}(S)$ typically is a highly nonconvex function, and therefore difficult to optimize. As shown by Kass *et al.* [1988], it can effectively be optimized by

- introducing a quadratic regularization[2] term $\mathcal{E}_D = 1/2 S^t K_S S$ where K_S is a sparse stiffness matrix,
- defining the total energy $\mathcal{E}_T$

$$\begin{aligned} \mathcal{E}_T(S) &= \mathcal{E}_D(S) + \mathcal{E}(S) \\ &= 1/2 S^t K_S S + \mathcal{E}(S) \ , \end{aligned} \tag{11}$$

- embedding the curve in a viscous medium and solving the dynamics equation

$$\begin{aligned} \frac{\partial \mathcal{E}_T}{\partial S} + \alpha \frac{dS}{dt} &= 0 \ , \\ \text{with } \frac{\partial \mathcal{E}_T}{\partial S} &= \frac{\partial \mathcal{E}_D}{\partial S} + \frac{\partial \mathcal{E}}{\partial S} \ , \end{aligned} \tag{12}$$

  where α is the viscosity of the medium.

Since the regularization term $\mathcal{E}_D$ is quadratic, its derivative with respect to S is linear, and therefore Equation 12 can be rewritten as

$$\begin{aligned} K_S S_t + \alpha(S_t - S_{t-1}) &= -\left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} \\ \Rightarrow (K_S + \alpha I) S_t &= \alpha S_{t-1} - \left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} \ . \end{aligned} \tag{13}$$

In practice, α is computed automatically at the start of the optimization procedure so that a prespecified average vertex motion amplitude is achieved [Fua and Leclerc, 1990]. 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

$$dS_t = S_t - S_{t-1} \tag{14}$$

as the "snake step" taken at iteration t.

Furthermore, $\mathcal{E}_D$ can be chosen so that its derivatives with respect to X,Y, and Z are decoupled so that we can rewrite Equation 13 as a set of two or three differential equations in the two or three spatial coordinates:

$$(K + \alpha I) W_t = \alpha W_{t-1} - \left.\frac{\partial \mathcal{E}}{\partial W}\right|_{X_{t-1}} \tag{15}$$

[2]This term can be understood as a "deformation energy" that minimizes the overall curvature of the model, hence the notation.

where W stands for either X, Y, or Z, and K a sparse nxn matrix, n being the number of vertices.

In effect, this optimization method performs implicit Euler steps with respect to the regularization term [Kass *et al.*, 1988] 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 to constrained snake optimization by taking the objective function f to be the total energy $\mathcal{E}_T$ of Equation 11. 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. This is especially true in most of our applications 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 snakes, we decompose the second step of the optimization procedure of Section 2 into two steps. We first solve the unconstrained Dynamics Equation (Equation 13) as we do for unconstrained snakes. We then calculate the component of the snake step vector of Equation 14—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 12, 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.

$$\begin{aligned} A^T A dV &= -C(S_{t-1}) \\ S_{t-1} &\leftarrow S_{t-1} + A dV \end{aligned}$$

   Calculate the snake's total energy. If it has increased, revert to the previous position and increase the 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. Compute λ such that

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

and update S_t

$$S_t \leftarrow S_t - A\lambda$$

so that the snake step dS becomes

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

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

$$\begin{aligned} x_i - a_0 &= 0 \ , \\ y_i - b_0 &= 0 \ , \end{aligned} \tag{16}$$

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

$$\begin{aligned} \begin{vmatrix} x_i & a_0 & a_1 \\ y_i & b_0 & b_1 \\ 1 & 1 & 1 \end{vmatrix} &= 0 \\ \begin{vmatrix} x_{i+1} - x_{i-1} & a_1 - a_0 \\ y_{i+1} - y_{i-1} & b_1 - b_0 \end{vmatrix} &= 0 \end{aligned} \tag{17}$$

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.

In Figure 7, we use these spring and tangent constraints to contrast the behavior of our algorithm with one that attempts to impose these constraints by adding penalty terms to the energy function, that is, one that minimizes

$$\mathcal{E}_T(S) + \sum_{1 \leq i \leq m} \rho_i c_i(S)^2 \ , \tag{18}$$

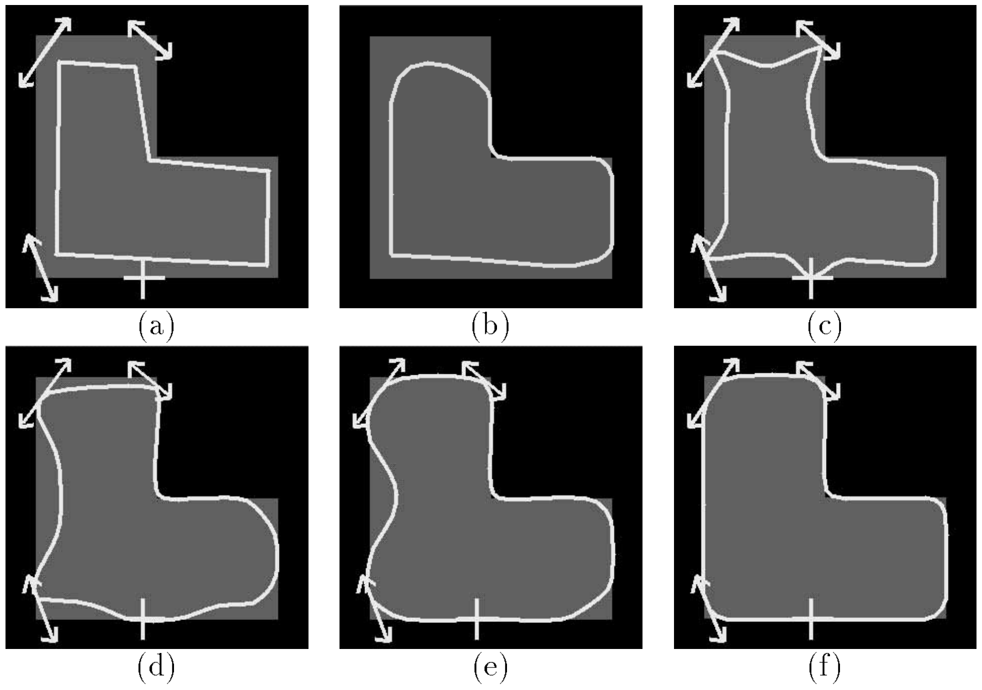

Figure 7: Imposing spring and tangent constraints on a 2–D snake. (a) An image of a polygon with an initial outline. The two-sided arrows represent tangent constraints (Equation 17), while the crosshair depicts an attractor constraint (Equation 16). (b) The result of unconstrained snake optimization. (c) The result of enforcing the constraints using penalty functions. (d,e,f) Three iterations of our constrained optimization algorithm.

where the ρ_i are arbitrarily chosen weights. The behavior shown in Figure 7(c) is typical. If the ρ_i are taken to be large enough to enforce the constraints, the image forces are almost completely ignored during the optimization, yielding a poor result. In essence, our method, like all those that rely on Lagrange multipliers, recomputes the weighting of each constraint—and not of its square—at each iteration so that this problem does not occur.

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. Given N snakes, we concatenate their respective state vectors $S_1, S_2, ..., S_N$ into a composite state vector $S = (S_1, S_2, ..., S_N)$ and compute for each snake the viscosity coefficient $\alpha_1, \alpha_2, ..., \alpha_n$ that would yield steps of the appropriate magnitude if each snake was optimized individually.

The three steps of an iteration of the optimization procedure then become

1. Project S onto the constraint surface as before and compute energy of each individual snake.

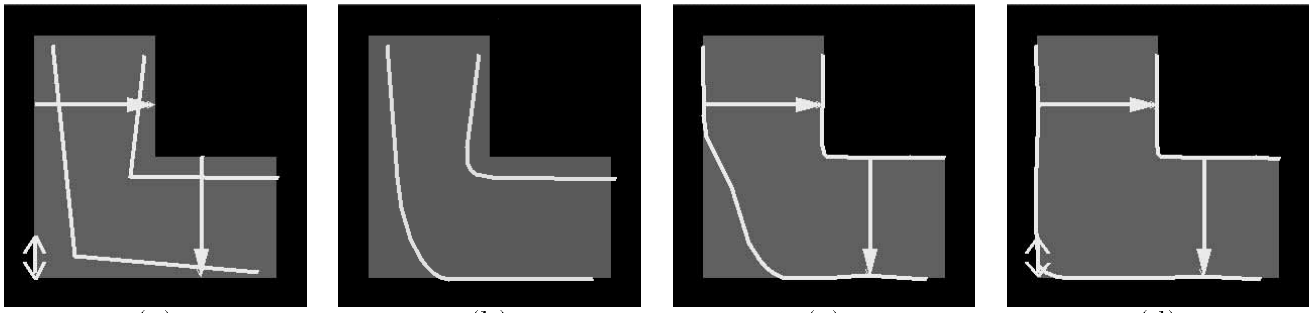
(a) (b) (c) (d)

Figure 8: Imposing distance and tangent constraints on a pair of 2–D snakes. (a) An image of a polygon with two initial outlines. The one-sided arrows represent distance constraints (Equation 19) that tie the two curves, while the two-sided arrow represents a tangent constraint (Equation 17) that constrains only one of the curves. (b) The result of unconstrained snake optimization. (c) The result of enforcing only the distance constraints. (d) The result of enforcing all the constraints.

For all snakes whose energy has increased, revert to the previous position and increase the viscosity.

2. Take a step for each snake individually:

$$\begin{aligned}(K_1 + \alpha_1 I)S_{1t} &= \alpha_1 S_{1t-1} - \left.\frac{\partial \mathcal{E}_1}{\partial S_1}\right|_{S_{1t-1}} \\ \ldots &= \ldots \\ (K_n + \alpha_n I)S_{nt} &= \alpha_n S_{nt-1} - \left.\frac{\partial \mathcal{E}_n}{\partial S_n}\right|_{S_{nt-1}} .\end{aligned}$$

3. Project the global step into the subspace tangent to the constraint surface as before.

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, that is, the individual snake optimizations, are well behaved, which is the case here.

To illustrate the behavior of this method, we reuse the image of Figure 7 and introduce a "distance" constraint between two snakes. Given a vector of length d, such as the ones depicted by arrows in Figure 8(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 endpoints. The distance constraint can then be written as

$$(x_i^1 - x_j^2)^2 + (y_i^1 - y_j^2)^2 + (z_i^1 - z_j^2)^2 - d^2 = 0 \ . \tag{19}$$

As shown in Figures 8(b,c), the algorithm exhibits good convergence properties even though the constraints are not linear but quadratic. It also allows us to effectively combine different types of constraints.

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

Figure 9(a) 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 17 and one of the attractor constraints of Equation 16, we can force convergence toward the desired edge.

We can also model the main road edges in the image of Figure 9 starting with the three rough approximations shown in Figure 10(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 19.

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.

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 11 and 12 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

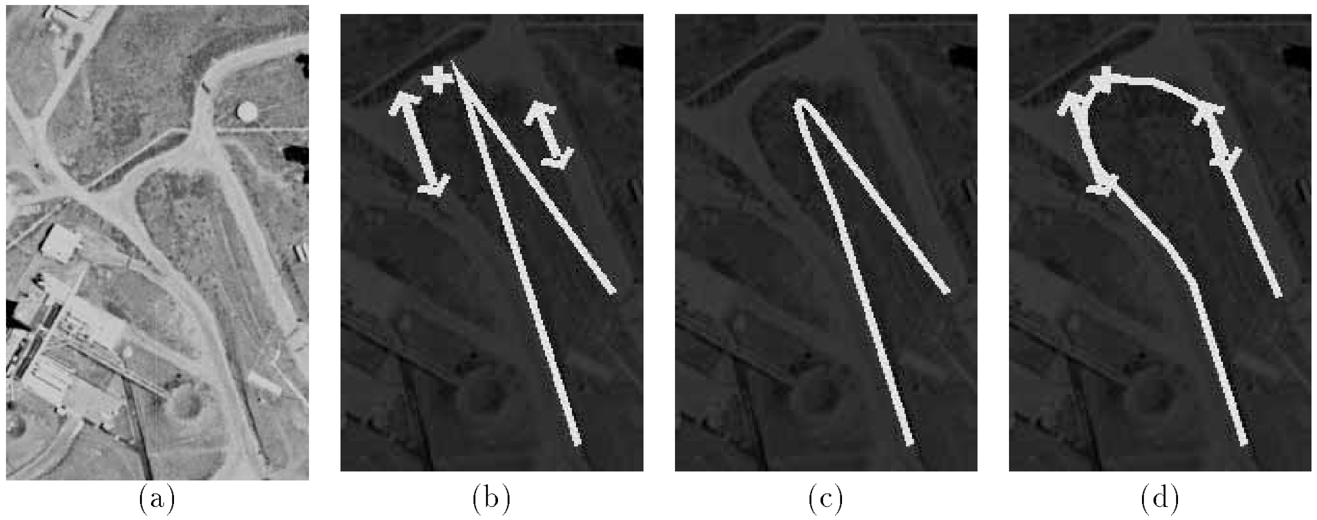

Figure 9: Modeling the edge of a road. (a) Aerial image of a set of roads. (b) A very rough approximation of one of the road's edges and a set of constraints. As in Figure 7, the two-sided arrows represent tangent constraints (Equation 17) while the crosshair depicts an attractor constraint (Equation 16). (c) The result of unconstrained snake optimization. (d) The result of constrained snake optimization using the constraints depicted by (b).

in 3–D space. This can expressed as

$$\begin{vmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{vmatrix} = 0 \quad , \tag{20}$$

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—ridgelines and roads—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 13 and 14, we show that the optimization under the consistency constraints of Equation 20 avoids the discrepancies that result from independent optimization of each feature.

In the example of Figure 13, the "ridge-snake" attempts to maximize the average edge gradient along its projections in all three images. In the case of Figures 12 and 14 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 as pairs of parallel edges. We do so by introducing pairs of snakes, constrained to remain parallel, that we call ribbon snakes. We also introduce a building and

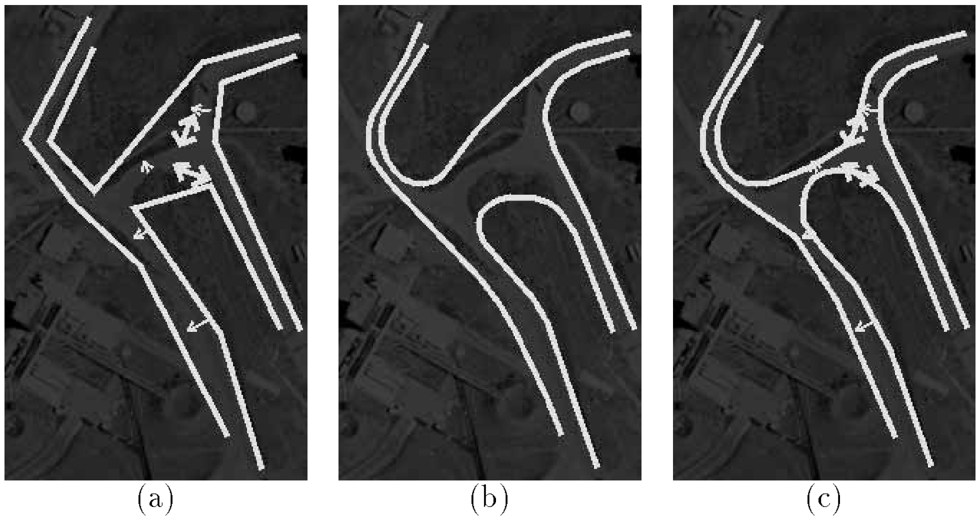

Figure 10: Modeling a set of road edges. (a) A set of three contours roughly approximating the edges of the main roads and a set of constraints. As before, the two-sided arrows represent "tangent constraints" (Equation 17) that apply to individual contours, while the thinner one-sided arrows represent distance constraints (Equation 19) that bind pairs of contours. (b) The result of unconstrained snake optimization. (c) The result of constrained snake optimization using the constraints depicted by (a).

use its base to further constrain the terrain. Figures 14(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 results shown in Figures 14(c,d,e,f). As shown in Figure 15, because the models for terrain and roads are consistent, we can now create synthetic views from different viewpoints that are such that the modeled road outlines still correspond to the road edges as they appear in the synthetic image. In other words, if used for simulation purposes, our approach would guarantee that there are no discrepancies between the synthetic images generated by the simulator and its internal representation of the objects.

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

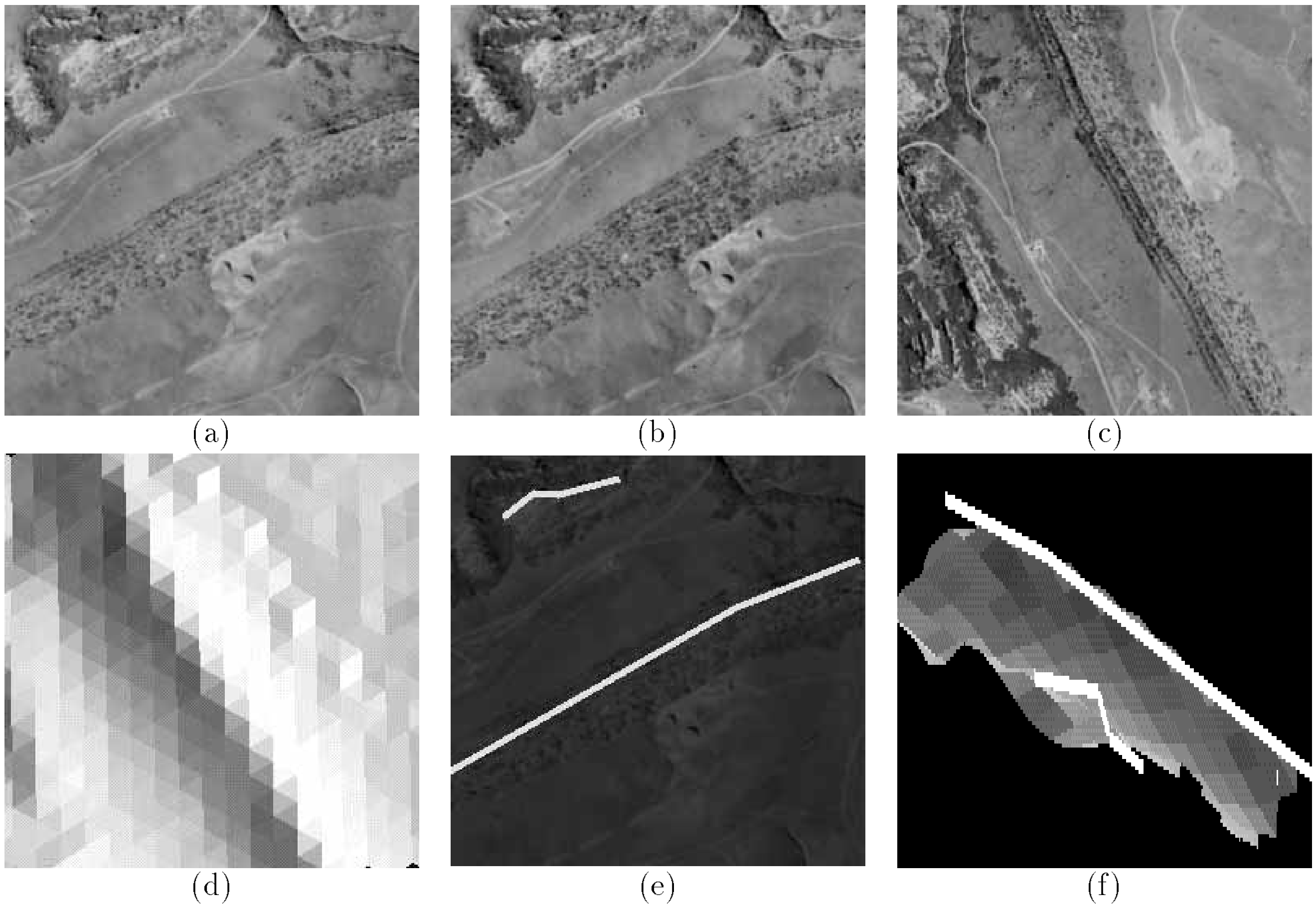

Figure 11: Rugged terrain with sharp ridge lines. (a,b,c) Three images of a mountainous site. (d) Shaded view of an initial terrain estimate. (e) Rough polygonal approximation of the ridgelines overlaid on image (a). (f) The terrain and ridgeline estimates viewed from the side (the scale in z has been exaggerated).

between them.

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

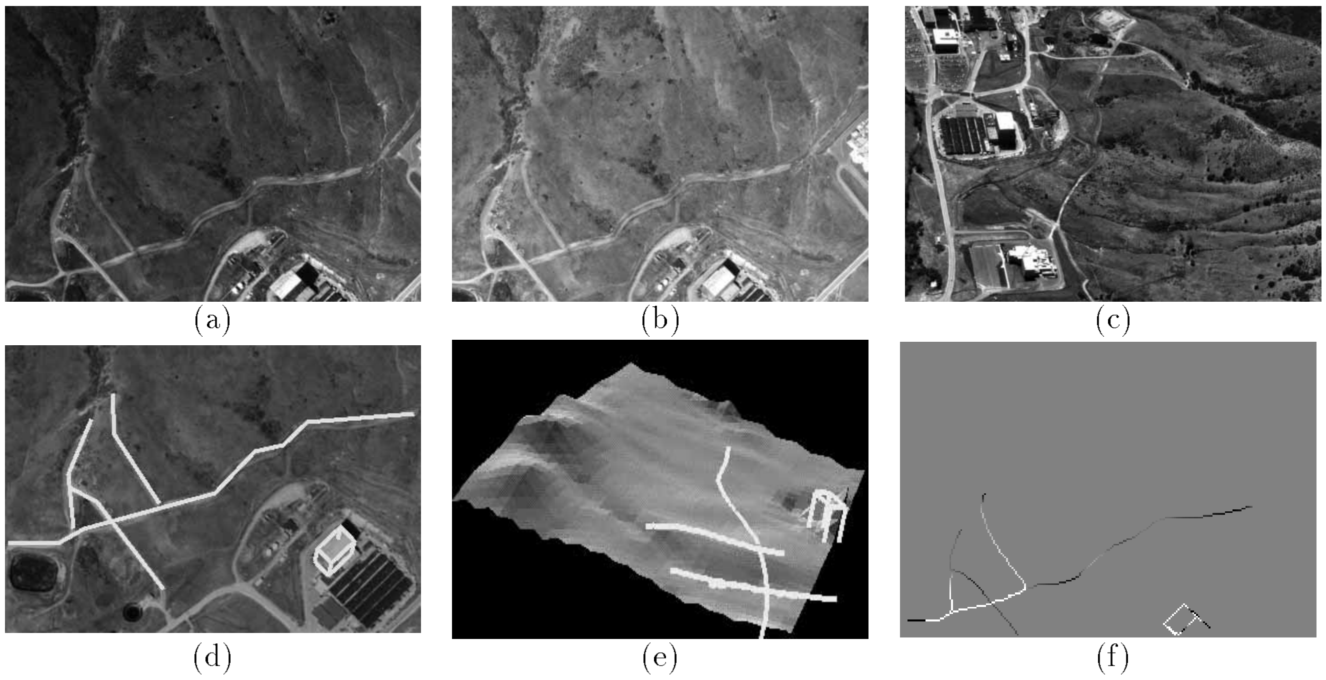

Figure 12: Building a site model. (a,b,c) Three images of a site with roads and buildings. (d) A rough sketch of the road network and of one of the buildings. (e) Shaded view of the terrain with overlaid roads after independent optimization of each. Note that the two roads in the lower right corner appear 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.

practices such as the fact that roads do not have arbitrary slopes.

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.

Acknowledgments

Support for the portion of this research conducted at SRI was provided by various contracts from the Advanced Research Projects Agency. We wish to thank Yvan Leclerc and Jean-Christophe Culioli for the advice they gave us during the development of the approach described here.

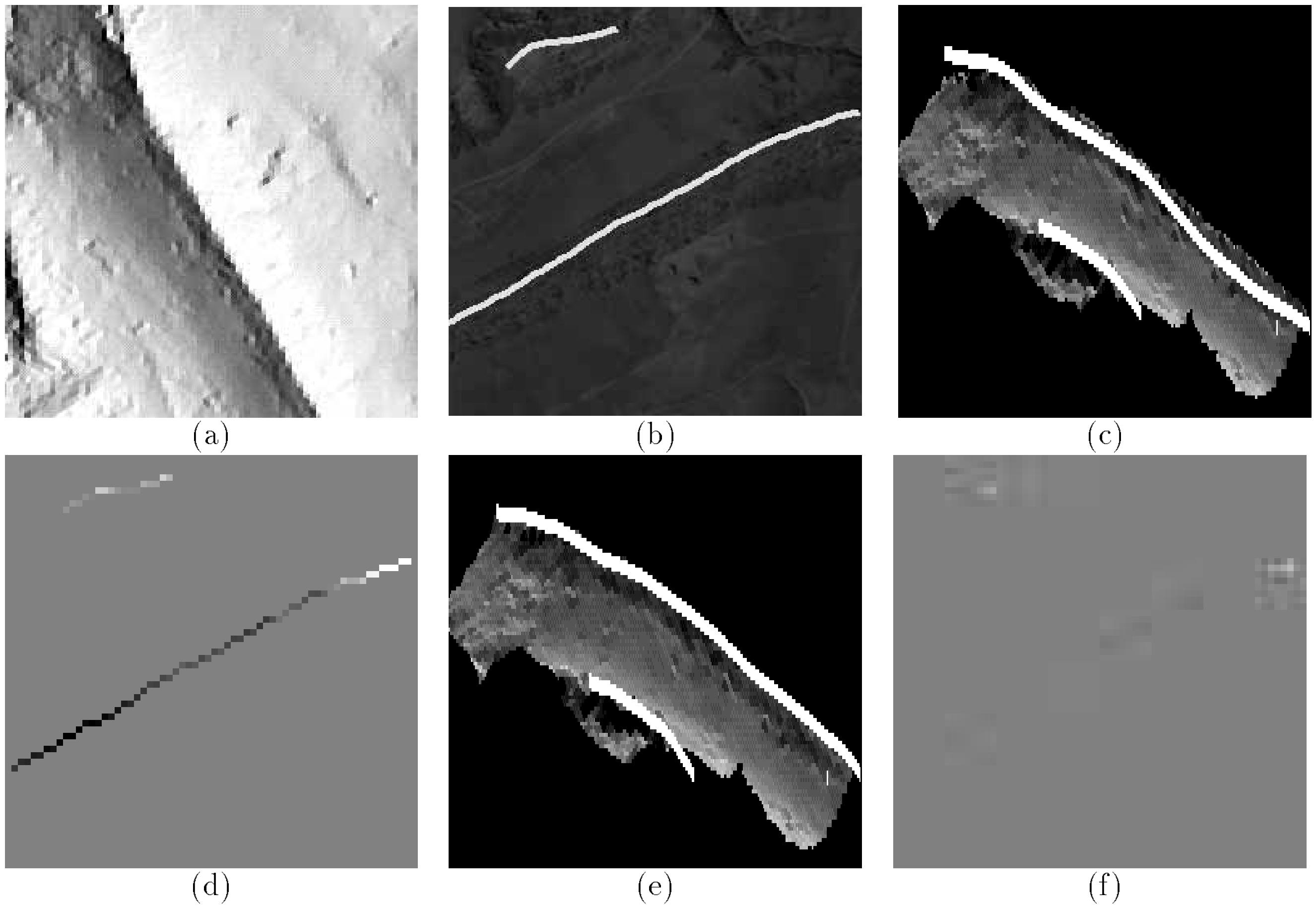

Figure 13: Recovering the 3–D geometry of both terrain and ridges. (a) Shaded view of the terrain after refinement. (b) Refined ridgeline after 3–D optimization. (c) Side view of the ridgeline and terrain after independent optimization of each one. Note that the shape of the ridgeline does not exactly 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).

Appendix: SQP, a Lagrange-Newton Algorithm

We summarize the optimization method presented in [Fletcher, 1987].

The Lagrangian function corresponding to the NEP of Equation 1 is defined as

$$l(S, \lambda) = f(S) - \lambda^T C(S) \ . \tag{21}$$

The *augmented* Lagrangian function includes a penalty term, the sum of the squared constraints

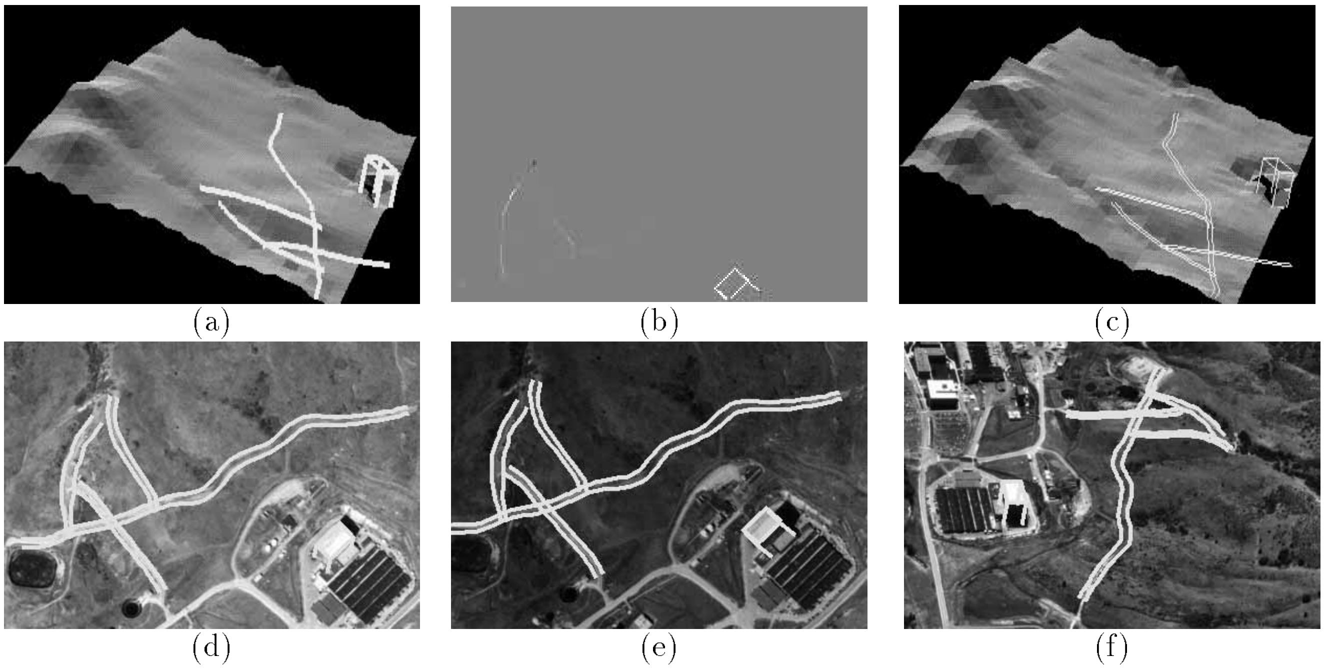

Figure 14: Recovering the 3–D geometry of both terrain and roads. (a) Shaded view of the terrain with overlaid low-resolution roads after optimization under consistency constraints. (b) Corresponding differences of elevation between features and underlying terrain. The image is stretched as the one of Figure 12(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.

multiplied by a penalty factor ρ.

$$l_A(S,\lambda,\rho) = f(S) - \lambda^T C(S) + \rho \cdot C(S)^T C(S) \tag{22}$$

At the solution (S^*, λ^*), the function $l(S,\lambda)$ is stationary with respect to S and λ. This can be written as

$$\bar{\nabla} l(S,\lambda) = 0. \tag{23}$$

Finding a zero of these equations using the Newton method is achieved by iteratively incrementing S and λ by dS and $d\lambda$, computed by solving

$$\bar{\nabla}\bar{\nabla}^T l \begin{pmatrix} dS \\ d\lambda \end{pmatrix} = -\bar{\nabla} l \ , \tag{24}$$

where the matrix on the left is the Hessian of l. This is equivalent to solving

$$\begin{pmatrix} W & -A \\ -A^T & 0 \end{pmatrix} \begin{pmatrix} dS \\ \lambda \end{pmatrix} = \begin{pmatrix} -G \\ C \end{pmatrix} , \tag{25}$$

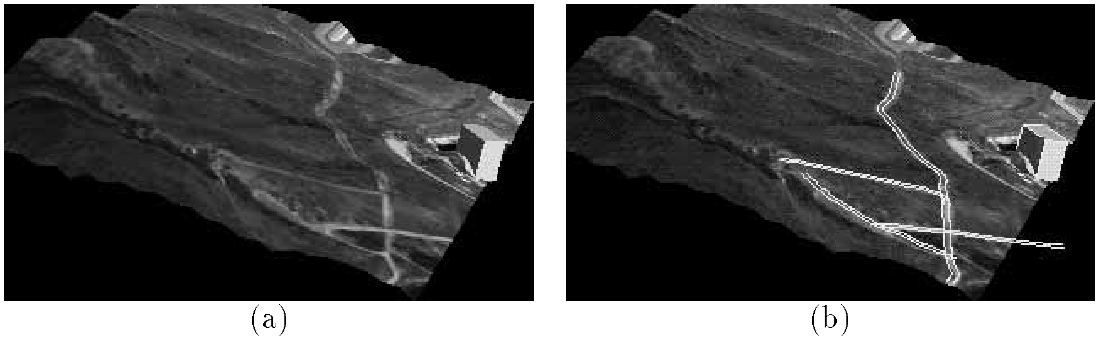

Figure 15: Generating synthetic views. (a) A synthetic view of the site generated using texture-mapping. (b) The same view with the roads modeled as ribbons overlaid on the image. Because the models for the road and the terrain are consistent, the modeled road outlines still correspond to the road edges in the synthetic image.

where

$$W = \nabla\nabla^T f(S) - \sum_i \lambda_i \nabla\nabla^T c_i(S)$$

is the Hessian $\nabla\nabla^T l$, dS the S increment, and λ the new estimate of the Lagrange multipliers.

The algorithm starts with initial estimates for S and λ. It then repeatedly solves the system of Equation 25 and updates $S \leftarrow S + dS$ until a convergence criterion is satisfied.

References

[Baumgarte, 1972] J. Baumgarte. Stabilization of Constraints and Integrals of Motion in Dynamical Systems. *Computational Methods Applied Mechanics Eng.*, 1:1–16, 1972.

[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.

[Brechbühler, 1995] C. Brechbühler. *Description and Analysis of 3-D Shapes by Parametrization of Closed Surfaces.* PhD thesis, ETH Zurich, Rämistrasse 101, CH-8092 Zürich, 1995. Diss. ETH No. 10979.

[Cohen, 1995] L. Cohen. Auxiliary Variables for Deformable Models. In *International Conference on Computer Vision*, pages 975–980, Cambridge, MA, June 1995.

[Culioli, 1994] J.C. Culioli. *Introduction à l'Optimisation.* Ellipses, 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 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. Also available as Tech Note 535, Artificial Intelligence Center, SRI International.

[Fua, 1995] P. Fua. Parametric Models are Versatile: The Case of Model Based Optimization. In *ISPRS WG III/2 Joint Workshop*, Stockholm, Sweden, September 1995.

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

[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.

[Metaxas and Terzopoulos, 1991] 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.

[Rosen, 1978] Rosen. Two-phase algorithm for non linear constraint problems. *Non Linear Programming*, 3:97–124, 1978.

[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.