Statistical Cue Integration in DAG Deformable Models

Siome Klein Goldenstein, *Member*, *IEEE*, Christian Vogler, and Dimitris Metaxas, *Member*, *IEEE*

Abstract—Deformable models are a useful modeling paradigm in computer vision. A deformable model is a curve, a surface, or a volume, whose shape, position, and orientation are controlled through a set of parameters. They can represent manufactured objects, human faces and skeletons, and even bodies of fluid. With low-level computer vision and image processing techniques, such as optical flow, we extract relevant information from images. Then, we use this information to change the parameters of the model iteratively until we find a good approximation of the object in the images. When we have multiple computer vision algorithms providing distinct sources of information (*cues*), we have to deal with the difficult problem of combining these, sometimes conflicting contributions in a sensible way. In this paper, we introduce the use of a directed acyclic graph (DAG) to describe the position and Jacobian of each point of deformable models. This representation is dynamic, flexible, and allows computational optimizations that would be difficult to do otherwise. We then describe a new method for statistical cue integration method for tracking deformable models that scales well with the dimension of the parameter space. We use *affine forms* and *affine arithmetic* to represent and propagate the cues and their regions of confidence. We show that we can apply the *Lindeberg* theorem to approximate each cue with a Gaussian distribution, and can use a *maximum-likelihood estimator* to integrate them. Finally, we demonstrate the technique at work in a 3D deformable face tracking system on monocular image sequences with thousands of frames.

Index Terms—Statistical cue integration, deformable model tracking, affine arithmetic, face tracking, directed acyclic graphs, deformable model representation.

1 Introduction

For years, engineers and computer scientists have been dealing with different abstractions to represent real solid objects inside a computer. Once the computer has an internal model of an object, we can analyze the object's properties and simulate its performance (with finite elements, for example), we can transform the object to achieve special effects, and we can even try to recognize the object and its actions. We call a *deformable model* any object whose shape changes according to a set of parameters.

Deformable models can represent a wide variety of shapes, from manufactured parts to the soft surface of the human body. In computer vision, deformable models constrain the tracking problem through the classes of deformations that they describe. In tracking, for example, we look for the value of the parameters for each frame, instead of the 3D positions of every point of the model. This restriction simplifies the problem. The accuracy and reliability of a deformable model tracking application is strongly dependent on how well the object under tracking fits the family of shapes described by the parameterization of the model. There is a fine balance between getting the smallest possible parameter space, thus dealing with more tractable problems, and capturing enough variability in the shape.

It is hard to store and manipulate all these models in a unified way and to parameterize them. Furthermore, with large-scale models, computational efficiency becomes a serious concern. The representation of models has been extensively investigated in computer graphics [19], [22], [18], but not in computer vision. In the first part of this paper, we present a flexible data structure for representing the models and associated parameterizations that is suitable for computer vision tracking applications. This data structure is based on a directed acyclic graph (DAG), which describes how points on the model depend on one another and on the parameters. It allows us to compute the positions and Jacobians of points efficiently, which are needed to track deformable models. Superficially, this data structure bears some similarities to graphical models, but differs from them in that we do not propagate probabilities through it. The concept of the DAG, however, is powerful enough that in future work it could be extended to propagate probabilities from the model parameters down to the point positions and associated Jacobians.

However, proper representation is only part of the picture. In deformable model tracking, each computer vision algorithm identifies a large number of local image displacements. These are then mapped into parameter space, using the local projected Jacobian (see (2) in Section 5.2) and summed up to form a single *generalized force*, also called a *cue*. As long as only one cue is used at a time, estimation of the model parameters is a straightforward process using a dynamical system (see Section 3). The picture changes dramatically, however, when multiple cues act on a model at the same time. The cues may affect different parameters, may yield conflicting information, and may even have their properties change over time, invalidating prior assumptions of the relative reliability of the cues.

Furthermore, due to the noise inherent in most low-level computer vision cues, different cues will exhibit different

• *The authors are with the Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Hill Center, Piscataway, NJ 08854-8019. E-mail: {siome, cvogler, dnm}@cs.rutgers.edu.*

Manuscript received 26 July 2002; revised 21 Feb. 2003; accepted 13 Mar. 2003.
Recommended for acceptance by J.M. Rehg.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 118241.

degrees of reliability at different points on the model surface. Even worse, usually the distribution of the noise is unknown. As a result, estimating the probability distributions for the optimal automated integration of cues to yield the best possible parameter estimate of the model is a difficult and open research problem.

In the second part of this paper, we address this problem. We describe a novel statistical approach to the estimation of the probability distributions of the cues. Our method is based on the interrelationships between *affine forms*, *affine arithmetic*, and *Gaussian probability distributions*. We develop a method for conversion between affine forms and Gaussians. This method supports automated cue integration, scales well with the dimension of the parameter space, and avoids most assumptions about the probability distribution of the noise in each of 2D image forces of the cues.

We use affine forms to represent the support of the local image contributions, while avoiding making assumptions about the actual shape of their probability distribution functions. We use affine arithmetic to convert them to generalized forces, and to sum them up to form the cue. Given certain conditions (see Section 5.2), we can use *Lindeberg's theorem* to show that the sum of the local image contributions making up a cue can be approximated by a Gaussian-distributed random variable, whose support is represented by an affine form. Moreover, we discuss how to bound the error in the approximation with the *Berry-Esseen theorem*. Once each cue is represented as Gaussian distribution, their integration becomes a task for a *maximum-likelihood estimator*.

The rest of this paper is organized as follows: We discuss related work, then give a brief introduction to the deformable model tracking framework. We follow with a detailed description of the DAG representation and the statistical cue estimation and integration framework. Finally, we validate our approach both quantitatively and qualitatively with face tracking experiments.

2 Related Work

Affine forms and affine arithmetic were developed in the 1990s as an alternative to classical interval arithmetic. Affine arithmetic provides tighter bounds than interval arithmetic in cascaded operations. Unlike interval arithmetic [32], it also preserves information about mutual dependencies between results. Since then, it has been used in numerical applications [31], electrical engineering [16], computer graphics [13], and computer vision [21].

Gaussian probability distributions are a widely used tool in engineering [17], as they have several desirable properties: preservation of linearity, compactness of representation via the mean and covariance matrix, and several convergence theorems, notably the central limit theorem.

A broad overview of the different methods for representation of objects in computer graphics can be found in [22]. One of the first constructive representations of objects, Constructive Solid Geometry, was a tree with Boolean operations to construct the volume of the solid object [35]. Hierarchical deformations have long been used to represent skeletons and articulated rigid bodies in both computer graphics and robotics [12], [18]. These kinds of hierarchies can be represented as a tree. When probabilities are propagated along a model, techniques such as *Bayesian Networks* [24] or *covariance propagation* [1] can be used. They are different from our DAG in that we do not yet use the structure to propagate probabilities. In computer graphics, a very similar method to our DAG approach has been used for dynamic control of models and their constraints [19].

Deformable models and their representations are an active area of research in both computer graphics and computer vision. *Snakes* were a landmark in the literature of computer vision [28] expanded to *active shapes* [11] and *active contours* [4] to allow statistical representation of the shape of the model and accomplish several tasks, from tracking to recognition. Stereo and shape from shading obtain initial fits [36] and, using a similar framework, [15] uses anthropometric data and inspired deformations to generate faces. A learning-based statistical model can help tracking of face models [6]. Eigen-based approaches can successfully track, fit, and even recognize objects [33], [5], [34]. In [9], the head is modeled as a cylinder, and in [3] as a plane, and in [34] tracking is used for animation, to mention just a few.

Cue integration is not a new topic. In [14], a two-cue integration algorithm is presented based on the use of constraints, in which optical flow is defined to be the constraining (i.e., most important) cue, and edges are defined to be the secondary cue. This framework requires an a priori user-based definition of which cue is the most important one. A voting approach for disambiguation of cue information, along with a very thorough review and comparison of several methods, is proposed in [7]. In this paper, we describe a method for automated cue integration that is general enough to merge contributions of cues that are structurally very dissimilar. Unlike previous work, our approach avoids making a priori assumptions about the distribution of noise in cues and it weights each cue's contribution dynamically depending on how much noise it contains.

There are several statistical approaches designed for tracking, estimation, and prediction. The *Kalman filter* [30], for example, treats the parameters and the observations as multivariate Gaussians, and also uses a linear predictive model. *Particle filter* [23] techniques (condensation [26]) propagate the evolution of non-Gaussian sampled distributions through nonlinear operations. Unfortunately, they require knowledge of the observations' distributions and the necessary number of samples of the distribution grows exponentially with the dimension of the parameter vector.

3 Deformable Models

The shape, position, and orientation of the surface of the model are controlled by a set of n parameters $\mathbf{q}$. For every point i on the surface of the model, there is a function F_i that takes the deformation parameters and finds

$$p_i = F_i(\mathbf{q}),$$

where p_i is the point position in the world frame.

In addition, most computer vision applications, such as deformable model tracking, require the first order derivatives, so we restrict F_i to the class of functions for which the first order derivative exists everywhere with respect to $\mathbf{q}$. This derivative is the *Jacobian* $\mathbf{J_i}$, where

$$\mathbf{J_i} = \begin{bmatrix} | & & | \\ \frac{\partial p_i}{\partial q_1} & \cdots & \frac{\partial p_i}{\partial q_n} \\ | & & | \end{bmatrix}.$$

Each column l of the Jacobian $\mathbf{J_i}$ is the gradient of p_i with respect to parameter q_l.

3.1 Fitting and Tracking

In principle, there is a clean and straightforward mathematical approach to track the parameters of deformable models across image sequences. Low-level computer vision algorithms generate desired 2D displacements on selected points on the model; that is, the differences between where the points are currently according to the deformable model and where they should be according to the measurements from the image. These displacements, also called the *image forces*, are then converted to one n-dimensional displacement $\mathbf{f}_g$ in the parameter space, called the *generalized force* and used as a force in a first-order massless Lagrangian system:

$$\dot{\mathbf{q}} = \mathbf{f}_g + F_{\text{internal}}(\mathbf{q}), \tag{1}$$

where $F_{\text{internal}}(\mathbf{q})$ is the result of internal forces of the model (i.e., elasticity). We integrate this system with the classical Euler integration procedure, which eventually yields a fixed point, where $\mathbf{f}_g = \mathbf{0}$. This fixed point corresponds to the desired new position of the model.

In order to use the system of (1), we have to accumulate all the 2D image forces from the computer vision algorithms into $\mathbf{f}_g$. First, we convert each image force $\mathbf{f}_i$ on a point p_i into a generalized force $\mathbf{f}_{g_i}$ in parameter space, which describes the effect that the single displacement at point p_i has on the model parameters. Obtaining the single generalized force $\mathbf{f}_g$ then simply consists of summing up all $\mathbf{f}_{g_i}$:

$$\mathbf{f}_g = \sum_i \mathbf{f}_{g_i}, \text{ where } \mathbf{f}_{g_i} = \sum_i \mathbf{B_i}^\top \mathbf{f}_i, \text{ and } \mathbf{B_i} = \left.\frac{\partial \mathbf{Proj}}{\partial p}\right|_{p_i} \mathbf{J_i}. \tag{2}$$

$\mathbf{B_i}$ is the projection of the Jacobian $\mathbf{J_i}$ from world coordinates into image coordinates via the image projection matrix $\mathbf{Proj}$ at point p_i.

Generating the generalized force in this manner works fine as long as all the image forces come from the same cue (i.e., the same low-level vision algorithm on the same image). The picture changes dramatically when there are multiple cues via multiple computer vision algorithms at work. In this case, each generates a distinct generalized force $\mathbf{f}_{g,c}$ and we have to integrate them into the single generalized force $\mathbf{f}_g$ of (1). Unfortunately, multidimensional cue integration is in no way a simple problem. In Section 5, we describe the problem further and develop a new method to integrate any number of cues statistically in an optimal way.

The concept of the dynamical system works well only if there is a close match between the model and the corresponding image. As a result, it is important to start the tracking process with a good initial "fit" of the model and its registration to the first image. Fitting is a very interesting, but hard problem in itself and is beyond the scope of this paper. For the purposes of the experiments run in this paper, we initially fit the model by manually specifying correspondences between selected points on the model and the image from the first frame.

4 Dynamic Representation of Deformable Models

The representation that we choose for deformable models has a significant effect on the tracking application in two areas: First, it affects how easy it is to reconfigure the application to track different models and, second, it affects computational performance. As the demands of the computer vision applications increase, so does the level of detail needed for the models. A model detailed enough to track subtle facial expressions; for example, requires thousands of polygons with dozens of parameters. With such large numbers of points, which are evaluated repeatedly during the integration of (1), it becomes essential to avoid repetitive computations, so we require a way to store common subexpressions. In addition, most points on the model depend only on a few parameters, so their Jacobians are accordingly sparse. Most significantly, however, with large models memory performance bottlenecks and cache sizes become a serious concern. Because of the large number of nodes involved, and the image processing that needs to be done on every frame, the model data structure and intermediate results are flushed from the CPU caches on a regular basis. Under these circumstances, special measures are needed to minimize the impact of cache misses.

To address these concerns, we now describe a *directed acyclic graph* data structure for the description of arbitrary deformable models. Its dynamic nature allows our tracking framework to adapt over time, to optimize the computations, and to minimize cache misses.

4.1 Directed Acyclic Graph (DAG)

At the model's core lies a discrete dynamic structure with a finite number of points, whose positions and Jacobians depend on one another and on the model parameters according to a set of mathematical functions. We organize the points and mathematical dependencies dynamically in a graph, which we evaluate bottom-up, on an as-needed basis.

There are two basic elements in this structure: *nodes* and *dependencies*. Nodes can be *normal nodes* and *ghost nodes*. Normal nodes correspond to actual points on the surface of the model. Ghost nodes, on the other hand, are not physical points on the model's surface. Rather, they are accumulators of common subexpressions, temporary results, or vectors. The dependencies are the mathematical building blocks that allow us to propagate the positions and Jacobians recursively through the deformable model.

Each node has one parent: a dependency, which in turn can have multiple nodes as parents. Each dependency represents a mathematical function that describes how the position and Jacobian of a particular node is determined by the positions and Jacobians of the respective parent nodes. Whenever we query a node for its position, or Jacobian, the node forwards the request to the respective parent dependency. The dependency, in turn, gets the position, or Jacobian, of the parent nodes, and calculates the resulting position, or Jacobian (using the chain rule). To avoid indeterminate expressions there are no loops;[1] thus, this structure is a *directed acyclic graph* (DAG). We now describe the most important dependencies in more detail.

4.2 Dependencies of the DAG

We need only a small set of simple dependencies to model affine deformations. They can approximate most complex deformations, through the smart use of ghost nodes to hold common subexpressions.

1. This is not strictly necessary, as there are recursive mathematical expressions that are well defined, but to allow this class of expression would increase the complexity of the data structure and affect computational feasibility.

Fixed Point. This dependency does not have any node as a parent. It takes three constants λ_1, λ_2, and λ_3 to define its position and Jacobian:

$$p_i = [\lambda_1\ \lambda_2\ \lambda_3]^\top \quad \text{and} \quad \mathbf{J_i} = \mathbf{0}. \tag{3}$$

Almost every recursive evaluation of the DAG eventually bottoms out at one or more fixed points.

Linear Combination of Points. This dependency requires a set of nodes $\{n_1, \ldots, n_k\}$ and a set of constants $\{\lambda_1, \ldots, \lambda_k\}$. Then, as the name implies:

$$p_i = \sum_{j=1}^{k} \lambda_j p_j \quad \text{and} \quad \mathbf{J_i} = \sum_{j=1}^{k} \lambda_j \mathbf{J_j}, \tag{4}$$

where p_j is the point represented by node n_j, and $\mathbf{J_j}$ is the associated Jacobian. This dependency is useful to specify the barycentric coordinates of a point with respect to a parameterized affine basis.

Add Parameterized Vector ("AddParVec"). This dependency adds a vector represented by three constants λ_1, λ_2, and λ_3, scaled by parameter l, to an existing node n_j.

$$p_i = p_j + q_l[\lambda_1\ \lambda_2\ \lambda_3]^\top \quad \text{and}$$
$$\mathbf{J_i} = \mathbf{J_j} + \begin{bmatrix} & 0 & \lambda_1 & 0 & \\ \cdots & 0 & \cdots \lambda_2 \cdots & 0 & \cdots \\ & \underbrace{0 \qquad \lambda_3}_{l} & & 0 & \end{bmatrix}. \tag{5}$$

This dependency is useful to specify a point with respect to a vector space basis. These three dependencies are only a small subset of all the ones that we have implemented, but capture most of the important deformations of a model in a tracking application.

4.3 Advantages of the DAG

The advantages of using this data structure to represent deformable models are manifold. It is fully dynamic, so it can easily be constructed and modified at runtime, instead of compile-time. As a result, it can be controlled through scripting languages, which facilitates adapting and reconfiguring the framework for different applications with a minimum amount of effort. An important consequence is that the dependencies can be selected and built programmatically through scripts, instead of having to be hand-picked. We take advantage of this facility by specifying the effect of deformations on regions of polygons in a declarative style, which our scripting library automatically translates into an appropriate DAG at runtime. Otherwise, specifying deformations on large models with thousands of polygons would be prohibitively cumbersome.

From an efficiency point of view, the data structure allows for several highly effective optimizations.

Common subexpressions. Each node has an associated storage area in which its position and Jacobian are cached. They are recomputed only when the cached information has become stale. As a result, by using a ghost node to represent a common subexpression, our framework ensures that it is computed only once.

Sparse matrices. Most of the nodes' Jacobians depend on very few parameters and, hence, are very sparse. Our framework collects for each node the set of parameters, on which it depends directly or indirectly. Based on this set, it determines which rows of the Jacobians it actually needs to compute, and which ones are simply set to zero. This optimization dramatically cuts down on the number of computations that need to be performed. It also potentially allows for saving space, but doing so is a double-edged sword because it interferes with the regularity of memory access patterns in the next optimization.

Cache behavior. As mentioned before, because of the size of the models and the nature of image processing, the data structures are flushed from the CPU caches on a regular basis. In the worst case, flushes happen at every iteration while the framework integrates the dynamical system, causing cache misses whenever the DAG is accessed. The misses can be minimized to some extent by clustering accesses to the DAG, but the latencies caused by cache misses are by far the more serious problem and are exacerbated by the ever increasing rift between CPU and main memory speed. Modern CPUs support hardware prefetching of cache lines, but, with the current state of the art, this strategy is effective only if the memory access patterns are regularly spaced and preferably linear.

It is in this area where using the DAG data structure pays off the most. Through a modified topological sort, our framework determines the order in which the nodes will be accessed during the calculations of the positions and Jacobians. It then lays out the associated storage areas in memory to ensure that the access patterns during the calculations are linear. It also schedules the read accesses to this information during the computations of the generalized forces to maximize the chance of regular patterns and intersperses them with cache prefetch instructions; with the net effect that the cache miss latencies are almost completely eliminated in typical tracking applications. This optimization would be very difficult to perform without a fully dynamical data structure.

4.4 Putting It Together: A Deformable Face Model

The basis of the deformable face is a static mesh of the face. Any model can be used, but the coarser the tessellation, the harder it is for computer vision algorithms to select good features. For our experiments, we started with a static geometric model of a head, publicly made available by the computer graphics group of the University of Washington[2] as part of [34]. We scissored a face mask out of the head model, and simplified it. The result was a static mask model of a generic face with 1,101 nodes and 2,000 faces. We defined parameters and associated regions for the raising and lowering of the eyebrows, for the smiling and stretching of the mouth, for the opening of the jaw, as well as seven parameters for the reference frame (a total of 11 parameters). For simplicity, we modeled the jaw movement as an affine transformation. This approximation is sufficient for small openings of the mouth, even though a rotation operation is the correct way to model the jaw.

Initially, all nodes on the static mesh are expressed as fixed points, as schematically shown in Fig. 1a, center. In theory, we could define a deformation for each node on the model by hand, but doing so would be extremely cumbersome.

2. http://www.cs.washington.edu/research/graphics/projects/realface/.

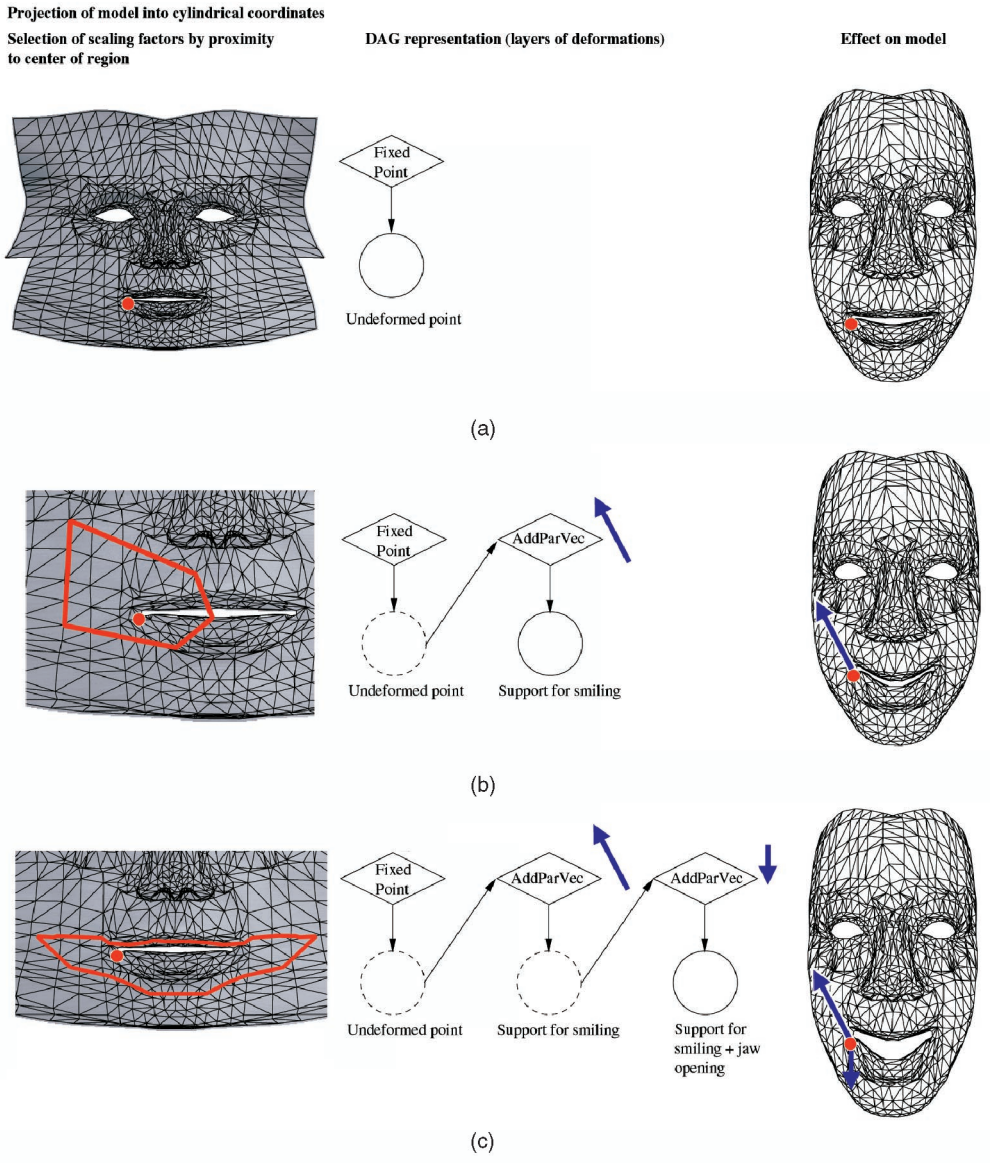


Fig. 1. Construction of the face model through the use of layers of deformation. Diamonds represent dependencies, dashed circles represent ghost nodes, and solid circles represent normal nodes. The left column shows the definition of regions in cylindrical coordinates. The center column shows the application of layered deformations for a node from these regions via adding parameterized vectors, whose magnitudes depend on the distance of the node from the center of the region. The right column shows the effect on the node in the deformable model.

Instead, we define deformations based on the concept of *regions*. We project the static mesh into a cylindrical coordinate space with the origin in the center of the head (Fig. 1a, left). In this space for each parameter, we mark one or more regions of nodes that are affected by it (see rows Fig. 1b and Fig. 1c, left).

As an example, consider the upward curving of the mouth in a smile. The general direction of the curving is upward and away from the corners of the lips, so we would like to model the deformations of those nodes with "Add Parameterized Vector" dependencies, which all move in the same direction $[u_1 u_2 u_3]^\top$ and share a common parameter that defines the extent of the upward curving. However, not all nodes are affected equally much: The nodes closest to the corners of the lips move the most, with the effect gradually diminishing as we consider nodes further away from this location. We therefore need to scale the direction vector by a constant λ that depends on how much a given node is affected by the smile. Thus, the final direction vector for the "Add Parameterized Vector" dependency (5) becomes $[\lambda_1 \lambda_2 \lambda_3]^\top = \lambda [u_1 u_2 u_3]^\top$.

We scale the direction vector depending on the distance of the node from the edge and the center of the marked region in the cylindrical coordinate space. For most deformations, we use a linear relationship to compute the scaling factors λ from the distances. However, for some deformations—especially lip curving when the jaw opens—an exponential decay function to compute λ has proven to be more accurate. Note that no matter what scaling function is used, these are computed at the construction time of the DAG and, thus, yield constants. As a concrete example, in Fig. 1b, left, the node in question is closer to the center of the region than in Fig. 1c, left. Thus, we scale the direction vector for the dependency in Fig. 1b, center, significantly more than for the one in Fig. 1c, center.

Because regions can overlap when a node is affected by multiple parameters, we chain the dependencies into *layers of deformations*, one per parameter. From top to bottom, the center column of Fig. 1 shows a progression of layers of deformations. We start with the fixed point from the static mesh. In the next row, we apply the deformation from the curving of the mouth in a smile. The original fixed point becomes a ghost node and the new position is calculated through the "Add Parameterized Vector" dependency from the old fixed point. In the bottom row, this process is cascaded to account for the movement of the node when the jaw is opened and the end result is a node that depends on the original fixed point and the addition of two parameterized vectors.

The regions are part of the declarative description of the face model that we briefly mentioned in Section 4.3. The computation of the scaling factors from the regions and the subsequent translation into the dependencies is done automatically by our scripting library at runtime.

5 Statistical Cue Integration

The low-level computer vision algorithms provide estimates of local two-dimensional forces to be applied to points of the model. These can be corrupted by noise and sometimes even by errors caused by local failures of the algorithm. As an example, consider the difference between point and edge tracking; the former works best in high-texture areas [37], whereas the latter works best on one-dimensional discontinuities [8]. As a result, the reliability of the estimates from a particular cue can vary dramatically, depending on which image region the information came from. Moreover, the noise characteristics of the image can affect the reliability of a cue as a whole, but some model parameters can be affected more than others. In our experiments, frequently one cue would deliver more reliable estimates of the model rotation than another and vice versa.

These considerations make it clear that combining the information from multiple cues is a very difficult problem. The standard approach to cue integration is simply to let each method convert the 2D image forces into a representative generalized force in parameter space and to average them into a single generalized force $\mathbf{f}_g$, which is then used in (1). This approach, however, completely ignores that the estimates from the cues exhibit different degrees of reliability. An alternative approach is to use one cue as a constraint [14], but this approach is still too inflexible in the case when the constraining cue becomes unreliable. A statistical approach seems like the obvious solution, but it opens another can of worms: How do we estimate the probability distributions of the contributions from the various image forces? Because the noise in the image forces is unknown, we cannot know what type of probability distribution best characterizes it.

To address these problems, we now describe a novel statistical technique for the estimation of cue probability distributions that does not assume anything about the noise in the image forces, except that it is uniformly bounded. This assumption is reasonable because of the very nature of the computer vision tracking application: There are *finite* displacements between successive frames, so the amount by which the model has to be moved from one frame to the next one is always finite.

The key element of this technique is the representation of the support of the noise with 2D regions. We use two-dimensional affine forms $\hat{\mathbf{f}}_{i,c}$ to represent these regions for a particular cue c:

$$\hat{\mathbf{f}}_{i,c} = \mathbf{a}_0 + \sum_{i=1}^{m} \mathbf{a}_i \varepsilon_i, \tag{6}$$

where $\mathbf{a}_0$ is a two-dimensional vector that represents the center of the region, $\mathbf{a}_i$ are also two-dimensional vectors, and ε_i are the *noise variables* (or *noise symbols*). They are independent from one another, although we do not know the shape of their distributions. Their support is $\varepsilon_i \in [-1, 1]$ and we let $E[\varepsilon_i] = 0$. Thus, affine forms describe a symmetric, convex polytope. By using the affine arithmetic counterparts to addition and multiplication by a scalar (see Appendix A), we propagate them through (2) to obtain the support of the generalized force for each cue as an n-dimensional affine form, where n is the dimension of the parameter space.

Even though we cannot say anything about the distributions underlying the image forces, we can make a strong statement about the probability distribution of the generalized force: *It can be approximated well by a Gaussian distribution*. The justification comes from *Lindeberg's theorem* which states that under certain conditions the sum of a large number of random variables converges to a Gaussian. We discuss in Section 5.2.1 that for our tracking application these conditions are satisfied and also bound the error of the approximation.

By approximating the generalized force from each cue with a Gaussian, we can integrate them with a maximum likelihood estimator (MLE). Overall, the integration consists of the high-level steps given in Algorithm 1. Next, we describe each step in more detail.

Algorithm 1. High-level overview of cue integration technique.
for each cue c **do**
  Select all points that cue c should to use.
  for each selected point p_i **do**
    $\hat{\mathbf{f}}_{i,c} \longleftarrow$ 2D affine form to be used as the force at point p_i.
  end for
  $\hat{\mathbf{f}}_{g,c} \longleftarrow \sum_{\forall i} \mathbf{B}_i^{\top} \hat{\mathbf{f}}_{i,c}$ {Calculates the generalized force as an affine form.}
  $\tilde{f}_{g,c} \longleftarrow$ Gaussian approximation of $\hat{\mathbf{f}}_{g,c}$.
end for
$\tilde{f}_g \longleftarrow$ MLE $\{\tilde{f}_{g,c}\}_{\forall c}$.
Apply mean of $\tilde{f}_g$ as generalized force in (1).

5.1 Computer Vision Cues as Regions

The first step is to estimate the image forces as 2D affine forms, instead of 2D vectors. We compute regions for the image forces in $\mathbb{R}^2$ which capture the area of confidence, in which the uncorrupted 2D force falls. There are rigorous statistical methods to estimate such regions [27], [29]; nevertheless, here we only use simple heuristics. Because affine forms are restricted to convex, symmetric regions, we may have to overestimate regions in order to fulfill these restrictions. Each low-level computer vision method has to select which points of the model have good local properties for their own requirements, thus avoiding obvious outliers. These properties are method-specific and constitute a broad research topic

of their own [20]. We now briefly describe how to estimate the affine forms for three different types of cues.

Edge Tracker. We compute the distance field on the result of a Canny edge detector. On each selected node, we apply a 2D force along the direction of the gradient of the distance field, with the value of the distance field as the force's magnitude. Thus, this force points to the closest edge point. However, we do not know whether the closest edge point is indeed the desired one or whether it is some other point along the same edge. For this reason, we model the region of uncertainty as a rectangle that is elongated along the direction of the edge, as shown in Fig. 2.

Point Tracker. A correlation-based point tracker takes an image and a position, and searches a point in the next image whose neighborhood is as similar as possible to the original neighborhood. We use a modified version of the KLT Tracker library[3] [39] to select and track points. The image force consists of the displacement of a point between successive frames. In this cue, errors occur when the deformable model has partially lost track in the previous frame, and, as a result, the point is being tracked from an incorrect position. We do, however, know with certainty that the position of the point is correct: in the first frame, to which we have fitted the model.

We can take advantage of this knowledge by tracking the point both from the first frame and the previous frame to the current one. In the common case, this approach yields two distinct positions for the point, with the correct one most likely being somewhere in the vicinity between them. To bound the region of confidence, we create a rectangle that is oriented along the line between these two positions, and that encompasses both of them.

Optical Flow. Variations of the classical optical flow algorithm [2] try to estimate the image motion through the optical flow constraint

$$\frac{\partial I}{\partial x}+\frac{\partial I}{\partial y}+\frac{\partial I}{\partial t}=0.$$

We estimate the region of confidence based on how good the gradient information is for the points that we select for optical flow. To extract this information, we compute the gradient crosscorrelation matrix

$$M=\begin{bmatrix}\sum_{j\in\Omega_i}\nabla x_j^2 & \sum_{j\in\Omega_i}\nabla x_j\nabla y_j\\ \sum_{j\in\Omega_i}\nabla x_j\nabla y_j & \sum_{j\in\Omega_i}\nabla y_j^2\end{bmatrix},$$

where ∇x_j is the horizontal gradient at point j, ∇y_j is the vertical gradient at point j, and Ω_i is a patch around the desired point i. The larger the eigenvalues for M are, the better the gradient information is along the corresponding eigenvector, and the higher the confidence is in our estimate. Accordingly, we define a rectangle with sides parallel to the eigenvectors as the region of confidence. The sizes of the sides vary inversely with the size of the eigenvalues.

5.2 Generalized Forces, Affine Forms, and Gaussians

Each cue c, as described in the last section, is transformed into a generalized force. Since $\mathbf{B_i}$ (the projected Jacobian of point i) is a matrix and $\hat{\mathbf{f}}_{i,c}$ (the 2D force of cue c applied to point i) is a 2D affine form, (2) is a set of linear operations over $\hat{\mathbf{f}}_{i,c}$ and the resulting generalized force $\hat{\mathbf{f}}_{g,c}$ is an affine form in parameter space.

3. http://vision.stanford.edu/~birch/klt/.

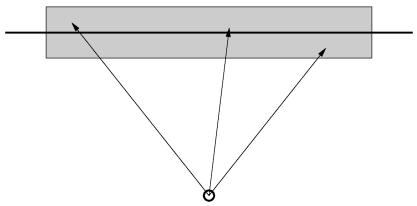


Fig. 2. Affine form for the image force in an edge detector. The size of the region along the edge is larger than the region perpendicular to it, describing the different confidences along these axes.

We now show that $\hat{\mathbf{f}}_{g,c}$ can be approximated well as a Gaussian random variable

$$\tilde{f}_{g,c}\ \approx\ \frac{1}{\sqrt{(2\pi)^n|\mathbf{\Lambda_c}|}}e^{-\frac{1}{2}(\hat{\mathbf{f}}_{g,c}-\mu_c)^\top\mathbf{\Lambda_c}^{-1}(\hat{\mathbf{f}}_{g,c}-\mu_c)}$$

with mean μ_c and covariance $\mathbf{\Lambda_c}$. Afterward, we describe a method to compute this approximation efficiently.

5.2.1 Proof of Gaussian Approximation: Lindeberg Theorem

The affine forms represent the bounded support of the generalized force. By definition, all noise variables of the affine form are independent (Appendix A), which implies that the image forces must be independent from another. We ensure this independence by selecting the points, for which we compute the image forces, sufficiently far apart from one another; in our experiments, we used a radius of 5 pixels.

The affine form is the sum of many independent random variables, whose support is a bounded 1D segment embedded in $\mathbb{R}^n$. Each noise variable has an unknown probability distribution and they are not necessarily identically distributed. In [20], we use the *Lindeberg theorem* [10, p. 205] to prove that, if all the l image forces f_i, applied at points p_i of the deformable model are independent and have bounded support, then the generalized force represented by an affine form, obtained according to (2), converges to a Gaussian as the number of image forces grows to infinity.

For estimating the error in the approximation, we can use the *Berry-Esseen theorem*, which bounds the maximum error between the distribution and its Gaussian approximation as a ratio between the sum of the absolute third moments and the cube of the sum of the standard deviations of the elements of the sum.

5.2.2 Mean of Gaussian Approximation

The estimation of the mean μ_c is a straightforward calculation. The generalized force is

$$\hat{\mathbf{f}}_{g,c}=\mathbf{a}_0+\sum_{i=1}^{m}\mathbf{a}_i\varepsilon_i,$$

where ε_i are the *noise variables* of the affine form (Appendix A). The mean vector is

$$\mu_c=E\left[\hat{\mathbf{f}}_{g,c}\right]=E[\mathbf{a}_0]+\sum_{i=1}^{m}E[\mathbf{a}_i\varepsilon_i]=\mathbf{a}_0+\sum_{i=1}^{m}\mathbf{a}_iE[\varepsilon_i],\quad(8)$$

but since let all noise variables ε_i have their mean at the origin,

$$\mu_c = \mathbf{a}_0. \tag{9}$$

5.2.3 Covariance of Gaussian Approximation

Finding the covariance matrix for the Gaussian approximation of an affine form is not a trivial task. Affine forms define *zonotopes* in higher dimensions, which means that the complexity of geometric algorithms (which use the points or faces of the region) grows exponentially in the dimension [20]. In [21], we used a heuristic to fine a hyperparallelepiped with minimum volume. Here, instead of interpreting the affine form geometrically, we take advantage of the expectation properties of the random variables. Using the definition of the covariance matrix $\mathbf{\Lambda_c}$ and (9):

$$\mathbf{\Lambda_c} = E\Big[(\hat{\mathbf{f}}_{g,c} - \mathbf{a}_0)(\hat{\mathbf{f}}_{g,c} - \mathbf{a}_0)^\top\Big]. \tag{10}$$

Each element λ_{ij} of $\mathbf{\Lambda_c}$ is

$$\begin{aligned}\lambda_{ij} &= E\Big[(\hat{\mathbf{f}}_{g,c} - \mathbf{a}_0)_i(\hat{\mathbf{f}}_{g,c} - \mathbf{a}_0)_j^\top\Big] \\ &= E\left[\left(\sum_{k=1}^{m} a_{ki}\varepsilon_k\right)\left(\sum_{l=1}^{m} a_{lj}\varepsilon_l\right)\right],\end{aligned}$$

where a_{ki} is the ith component of the vector $\mathbf{a}_k$ in $\hat{\mathbf{f}}_{g,c}$, and $(\hat{\mathbf{f}}_{g,c} - \mathbf{a}_0)_i$ is the one-dimensional affine form corresponding to the ith component of $(\hat{\mathbf{f}}_{g,c} - \mathbf{a}_0)$.

Expanding the sum, we observe that, because the ε are mutually independent and have zero mean, the cross terms are zero:

$$\lambda_{ij} = \sum_{k=1}^{m} a_{ki}a_{kj}E\big[\varepsilon_k^2\big] = \sum_{k=1}^{m} a_{ki}a_{kj}\sigma_{\varepsilon_k}^2, \tag{11}$$

or, if we assume that all noise variables have a common variance σ_ε^2,

$$\lambda_{ij} = \sigma_\varepsilon^2 \sum_{k=1}^{m} a_{ki}a_{kj}. \tag{12}$$

We build $\mathbf{\Lambda_c}$ using (11) or (12). Note that both equations are just a multiplication of an n-by-m matrix with its transpose, where the $\sigma_{\varepsilon_k}\mathbf{a}_k$ form the columns of the matrix. With a standard implementation of a matrix multiplication, this algorithm has complexity $\mathcal{O}(n^2m)$, where n is the dimension of the affine form and m is the number of noise variables. Beside the simplicity of implementation, the algorithm's most compelling advantage is that it provides an optimal estimate of the principal axes of the Gaussian distribution if the conditions of Lindeberg's theorem are satisfied. The reason is that, if these conditions are satisfied, Lindeberg's theorem tells us that the affine form indeed represents a Gaussian probability distribution, and the Gaussian estimated from (11) or (12) has both the same first-order and second-order moments as the affine form.

5.3 Generalized Force Integration

We have shown how to obtain the Gaussian representations $\tilde{f}_{g,c}$ for each cue c (see (7)). The remaining task is to combine their contributions into a single generalized force $\mathbf{f}_g$, so that we can integrate the dynamical system in (1) with contributions from multiple cues. We use an iterative Gaussian *maximum-likelihood estimator* (MLE) to combine all cues optimally.

Algorithm 2. Gaussian Maximum-Likelihood Estimator.

Initializes: $\hat{\mu}_1 \longleftarrow \mu_1, \quad \hat{\mathbf{\Lambda}}_\mathbf{1} \longleftarrow \mathbf{\Lambda_1}$

for $c = 2 \ldots nc$ **do** {Where $nc =$ number of cues}

$b_c \longleftarrow \hat{\mathbf{\Lambda}}_{c-1}\Big(\hat{\mathbf{\Lambda}}_{c-1} + \mathbf{\Lambda}_c\Big)^{-1}$

$\hat{\mu}_c \longleftarrow \hat{\mu}_{c-1} + b_c(\mu_c - \hat{\mu}_{c-1})$

$\hat{\mathbf{\Lambda}}_c \longleftarrow (\mathbf{I} - b_c)\hat{\mathbf{\Lambda}}_{c-1}$

end for

$\mu \longleftarrow \mu_{nc}, \quad \mathbf{\Lambda} \longleftarrow \hat{\mathbf{\Lambda}}_{nc}$

5.4 Computational Considerations

For each frame, solving (7) requires many iterations. The process would be computationally too expensive if all quantities had to be recomputed at each iteration. Fortunately, it has been shown that in normal situations, where the displacements between frames are not too large, the projected Jacobians $\mathbf{B_i}$ need to be calculated only once per frame, instead of per iteration [14].

If we assume that the shape of the affine form representing an image force $\hat{\mathbf{f}}_{i,c}$ does not change during the iterations over a single frame, the shape of $\mathbf{B}_i^\top\hat{\mathbf{f}}_{i,c}$ also remains the same. For many cues, this assumption is valid because the image is processed only once per frame. In this case, the covariance matrix of the Gaussian approximation of $\hat{\mathbf{f}}_{g,c}$ also remains the same. By taking advantage of this property, we can substantially optimize the statistical integration process: The full statistical computations need to be performed only at the first iteration. Subsequently, the unchanging covariance matrices can be cached, and only the means need to be recomputed. With this optimization, the overhead of the statistical integration method over the classical, averaging integration method becomes negligible, on the order of 5 percent of the total runtime.

6 Experiments and Validation

The system that we use as a base for all experiments has approximately $100,000$ lines of code, in four languages: C++, C, Lua, and TCL. The system uses *Lua* [25] for scripting. We provide bindings for most of our classes and algorithms, and Lua scripts configure elements, control the general flow of the simulation (the computationally expensive calculations are all done in C++), and provide the graphical user interface for several support utilities. The DAG model description files are implemented through Lua scripts as a mixture of declarative and procedural elements.

For our experiments, we used the model that we construct in Section 4.4. In Figs. 3 and 4, we show a few snapshots of the tracking sequences of two different subjects. In Fig. 3, we show a few snapshots of a tracking sequence with nearly 4,000 frames on a female subject, and in Fig. 4, we show snapshots of a sequence with 3,600 frames on a male subject. Both sequences were captured with a resolution of 640 x 480 pixels at 60Hz. The initial fitting of the mask was done using a few hand-picked correspondences between model landmarks and image points in the first frame, and did not give an accurate description of the points' positions along the Z-axis.

We used no predictive or corrective filter in these experiments. Furthermore, we imposed no constraints

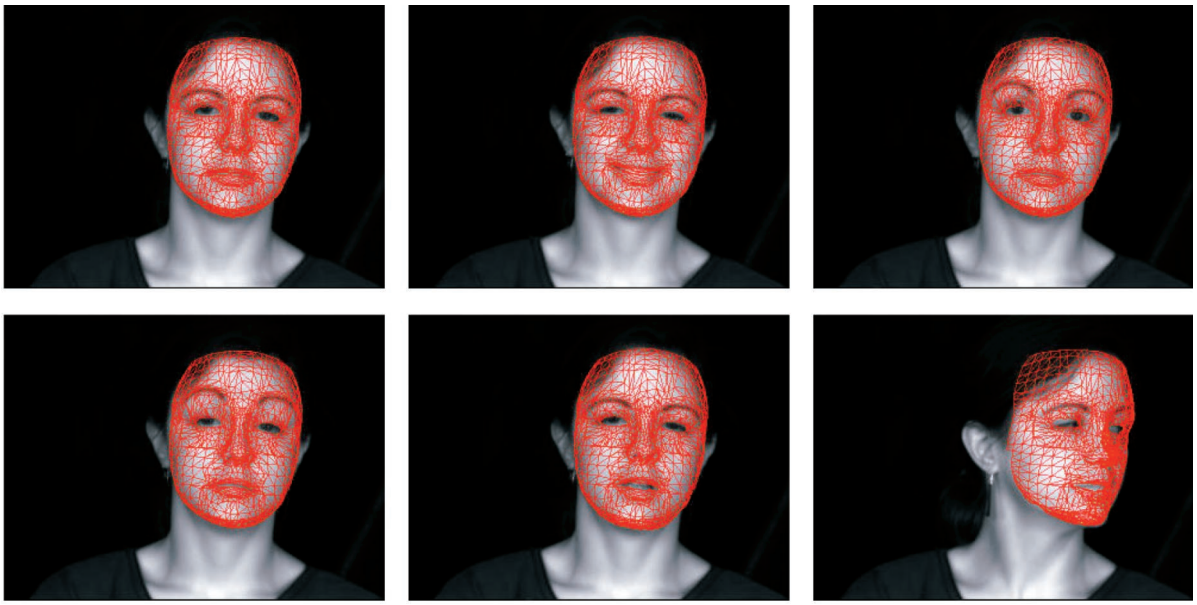


Fig. 3. Six snapshots of a tracking sequence of nearly 4,000 frames. The deformable model used here has 11 parameters.

(except for the normality of the orientation quaternion) on the values of the parameters.

6.1 Validation

To perform a quantitative validation we collected a special sequence of images. Using makeup, we drew markers (black dots) on the face of the subject before capture. Later, using a semiautomated procedure, we determined the 2D position of these markers on every frame of the sequence. The sequence is 1 minute long, and was captured at 60Hz (3,600 frames). During the tracking, we disallowed the selection of any points by the low-level vision algorithms that were too close to the markers, so that the markers would not affect the tracking process. We also specified the points on the model that corresponded to each marker, and projected them into image space at every frame, so as to estimate the 2D positions of the markers. The distance between our estimate and the real 2D position of the markers determined the error of the tracking.

In the standard deformable model framework, every point has an equal weight in the sum of the generalized forces ((2)). If we group the 2D image forces according to the type of cue (as we do in our statistical technique), standard cue integration can be viewed as the average weighted by the number of forces on each cue.

We compared the average errors that we gained from tracking with our statistical cue integration algorithm, and the standard method of averaging the cues under six different conditions: at the original image size of 640×480, at a rescaled size of 320×240, and at frame rates of 15Hz, 30Hz, and 60Hz. We simulated the lower frame rates by subsampling the image sequence over time. In Fig. 5, we see the

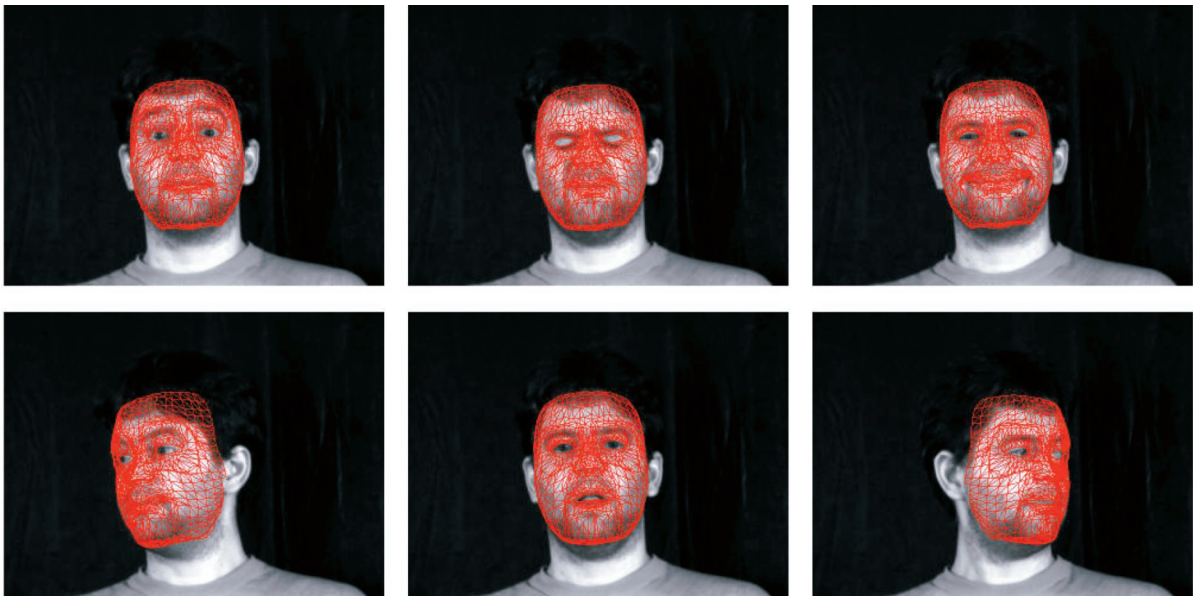


Fig. 4. Six snapshots of a tracking sequence of 3,600 frames. The deformable model used here has 11 parameters.

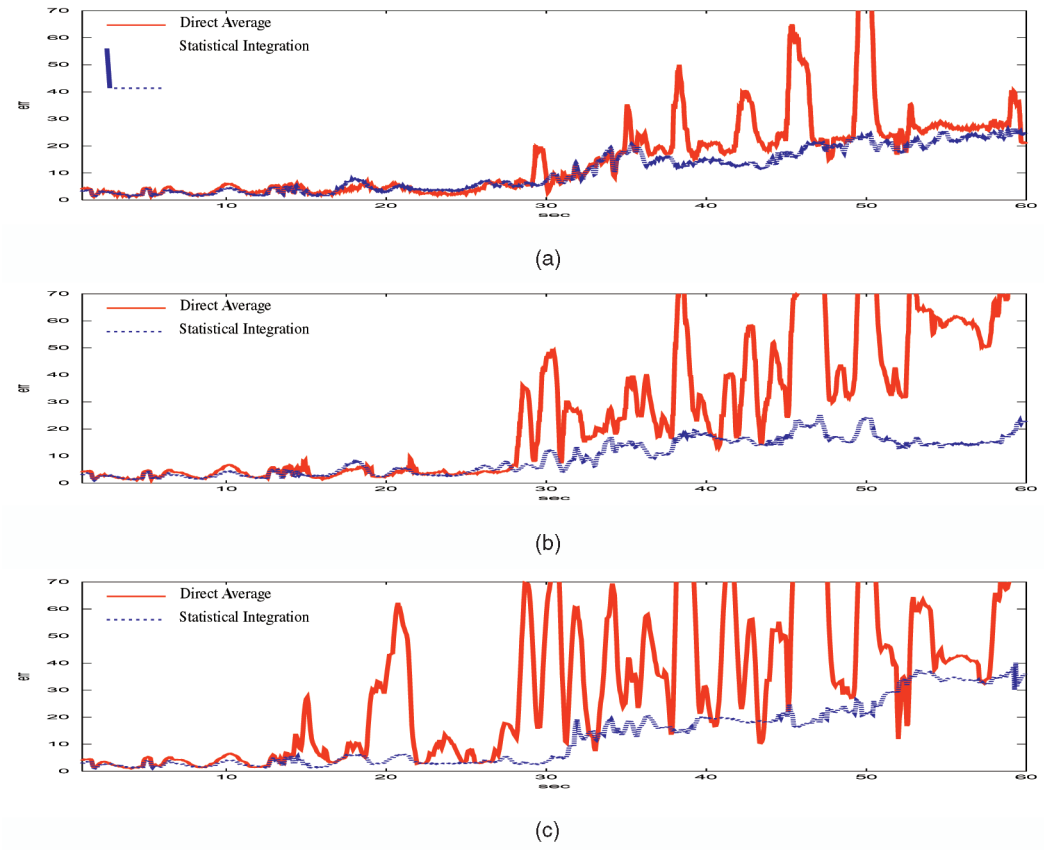


Fig. 5. (a) 60 Hz. (b) 30 Hz. (c) 15 Hz. Validation at full 640×480 resolution. Plot of the average error, in pixels, of the visible markers.

results at 640×480. It took our system at most 1.1 seconds per frame, of which 0.6 seconds were spent on image processing. We used 600 steps in the integration of (1). In Fig. 6, we see the results at 320×240. Our system used, at most, 0.7 seconds per frame (including 0.2 seconds for image processing), with 600 steps in the integration of (1).

Figs. 5 and 6 show that, as the frame rate of the sequences decreases, the traditional averaging method loses track earlier. These results are supported by our qualitative evaluation of unmarked tracking sequences, as well. Note that for both methods the overall average error in the validation sequences is larger than it would be under normal conditions because the markers were located in areas of interest; thus, we were forced to ignore many model points that would otherwise have provided the system with important tracking information.

We can draw the conclusion that the statistical cue integration makes the system significantly more robust compared to the standard averaging method. This effect gets more accentuated as the quality of the cues, image sizes, or frame rates decrease. For an explanation for the difference in robustness consider the contour of the face: Ideally, we would have a perfectly continuous edge, providing the edge tracker with a large number of selected points where to apply forces. Nevertheless, there are a few points along the contour with enough texture to be classified as corners, such as a birthmark, or the beginning of the earlobe. Although these are few in number, they can, individually, provide a much more reliable estimate of movement than edges.

In the standard method, the contribution to the parameters from the point tracker's corners in this region is diluted since the number of edge forces in that area is so much larger. On the other hand, in our statistical approach, the high confidence in the points is encoded in a very small affine region that represents the image force. These regions are mapped into parameter space and, eventually, affect the covariance matrix of the Gaussian. High confidence leads to small terms in the covariance matrix. The maximum-likelihood estimator essentially does a weighted average based on the inverse of the covariances: small covariances have larger relative weight, so the most reliable points are weighted the most. Analogous arguments apply to other regions and other cues.

The crux of the argument is that the statistical cue integration is capable of weighting the cues differently, and individually, for the many parameters, accounting for the different properties of the model and image. Since the parametric estimation of the cues is done continuously, our method is adaptive to the dynamic changes in the image sequence and keeps the tracking drift under control. For long sequences, as we show in this paper, even small improvements in the drift control are capable of substantial gains in tracking length.

The system behaves well and robustly as long as the model for the points is valid. As expected, problems arose when there was complex mouth movement while the face underwent large rotations (around 75 degrees). Two things contributed to this behavior: first, the bad depth estimate of the face model since the fit was accomplished from only one frontal view, and second, the simplified linear approximation of the jaw opening (see Section 4.4).

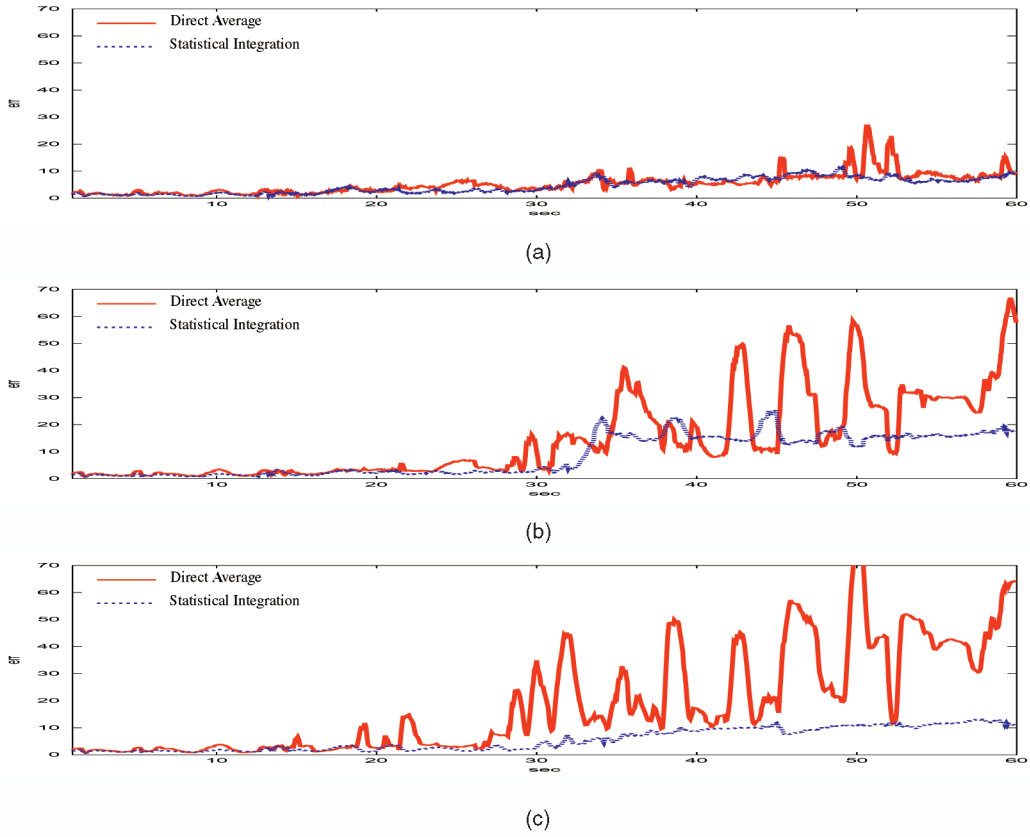


Fig. 6. (a) 60 Hz. (b) 30 Hz. (c) 15 Hz. Validation at scaled 320×240 resolution. Plot of the average error, in pixels, of the visible markers.

7 Conclusions and Future Work

It is hard to design efficient and flexible representations of a deformable model. The existing literature frequently concentrates on the high-level mathematical foundations and fails to describe the implementation details. To specify deformable models easily and to get good performance out of them, we need a powerful model specification technique. In this paper, we have described a representation based on a DAG that fills this gap. This data structure allows the description of complex deformations with a set of simple basic block operations. It allows fast calculation, on demand, of both positions and Jacobians. It allows us to construct the deformations with a mixed procedural and declarative approach, and can restructure them dynamically. Most critically for good performance, it supports CPU cache optimizations on-the-fly.

The next step in extending the DAG formulation is the creation of new algorithms for the automatic construction of the DAG from data, with the set of possible dependencies as a constraint. In this framework, the use of principal component analysis (PCA) for shape description [5] is just a special case (where only affine operations over the nodes are allowed). Since the deformations are described as a graph, another interesting problem is how to modify an existing DAG automatically; for example, to simplify it in order to obtain a more computationally efficient structure.

We have described a statistical technique that allows cue integration without having to know the probability distributions of the image forces. It scales well with the dimension of the parameter space and is general enough to integrate cues of very different, and even incompatible, origin. Although we have not explored it in this paper, our method is able to combine information from a 3D scanner or a stereo pair, with the normal monocular vision algorithms. Affine arithmetic can even be used in the stereo system, for example, to estimate a 3D region of confidence where the correspondences should be based on the 2D confidence regions of the matching correspondences. Our method requires the reformulation of the underlying basic computer vision techniques, so that the algorithms estimate convex symmetric regions instead of 2D forces. In spite of this reformulation, the computational considerations in Section 5.4 show that the effect on speed is small.

In this paper, the DAG formulation and estimation of cue probability distributions via affine arithmetic are two important, yet essentially separate building blocks for a deformable tracking system. Combining them will be one of the most important future applications because it will allow propagating the probability distribution of the parameter vector through the DAG to obtain probabilistic estimates of the node positions and Jacobians. Affine arithmetic is ideally suited for this task, for two reasons: First, it keeps track of correlations *across* different intermediate values, and, second it, can handle nonlinear node dependencies, such as rotations. Such a step would also make the DAG structure more similar to existing graphical models for probability propagation.

APPENDIX A
AFFINE FORMS AND AFFINE ARITHMETIC

Affine arithmetic is a numeric technique similar to *interval arithmetic*, in the sense that it propagates regions, instead of numbers, across arithmetic operations. The atom of affine arithmetic is called an *affine form*. An affine form $\hat{a}$ is represented as:

$$\hat{a} = a_0 + \sum_{i=1}^{m} a_i \varepsilon_i. \tag{13}$$

In $\mathbb{R}^1$, the coefficients a_i are real numbers, whereas in $\mathbb{R}^n$ they are n-dimensional vectors. The ε_i are symbolic real variables whose values are unknown, but guaranteed to lie in the interval $[-1 \ldots 1]$. The quantity a_0 is called the *central value* (mean), and the ε_i are called the *noise variables*. Each noise variable ε_i represents an independent component of the total uncertainty. In $\mathbb{R}^1$, $\hat{a}$ represents an interval and in $\mathbb{R}^n$ a convex polytope, whose number of faces depends on n and m.

For each operation on real numbers, we have to define a counterpart for affine forms. Affine operations like

$$\hat{z} = \alpha\hat{x} + \beta\hat{y} + \zeta, \tag{14}$$

are calculated exactly, where $\hat{x}$, $\hat{y}$, and $\hat{z}$ are affine forms represented by

$$\hat{x} = \sum_{i=1}^{m} x_i \varepsilon_i, \qquad \hat{y} = \sum_{i=1}^{m} y_i \varepsilon_i, \qquad \text{and } \hat{z} = \sum_{i=1}^{m} z_i \varepsilon_i,$$

and α, β, and ζ are real constants. The definition of this operation is

$$z_0 = \alpha x_0 + \beta y_0 + \zeta \quad \text{and} \quad z_i = \alpha x_i + \beta y_i. \tag{15}$$

Note that any operation defined on two affine forms also defines this operation on an affine form and a scalar, because a scalar s is trivially represented by the affine form $a_0 = s$.

Although for our cue integration technique, we only need affine operations, as specified in (14), other operations are also possible. A thorough description of how to do operations like reciprocation, multiplication, exponentiations, trigonometry, and even how to define a new operation, can be found in [38].

An affine form that is the result of an operation on other affine forms shares its noise variables with the affine forms of the operands. As a result, and in contrast to interval arithmetic, affine forms preserve interdependencies between values from intermediate computations. After a series of cascading operations, affine arithmetic usually provides tighter bounds than interval arithmetic.

As an example, consider a two-dimensional affine form f_{cj} as follows:

$$\mathbf{f}_{cj} = \begin{pmatrix} \hat{f}_x \\ \hat{f}_y \end{pmatrix} = \begin{pmatrix} 10 \\ 20 \end{pmatrix} + \begin{pmatrix} 2 \\ -3 \end{pmatrix}\varepsilon_1 + \begin{pmatrix} 1 \\ 0 \end{pmatrix}\varepsilon_2 + \begin{pmatrix} 0 \\ 1 \end{pmatrix}\varepsilon_3 + \begin{pmatrix} -1 \\ 4 \end{pmatrix}\varepsilon_4. \tag{16}$$

This representation, shown in Fig. 7, describes a vector whose mean is at $(10, 20)^\top$. If $\hat{f}_x$ and $\hat{f}_y$ were taken as independent, their spanned intervals would be $[6 \ldots 15]$ and $[12 \ldots 18]$, respectively (plotted as the light gray in Fig. 7).

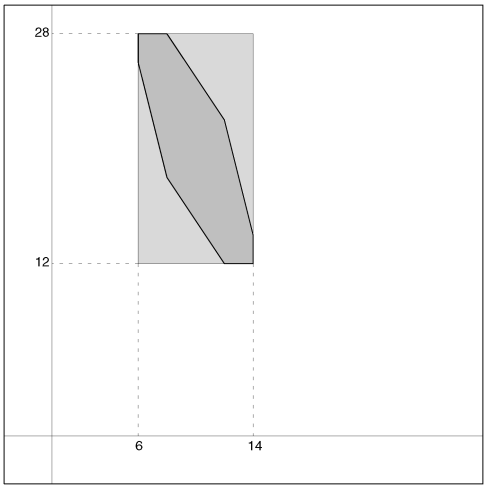


Fig. 7. Region defined by the 2D affine form of (16). In dark gray, we see the region of the affine form, while in the region of the interval counterpart is in light gray. Source: "Self-Validated Numerical Methods and Applications," Stolfi and Figueiredo, 1997 (used with permission).

However, because $\hat{f}_x$ and $\hat{f}_y$ share the noise variables ε_1 and ε_4, their variations are not independent. In fact, f_{cj} has to lie in the dark region of Fig. 7.

ACKNOWLEDGMENTS

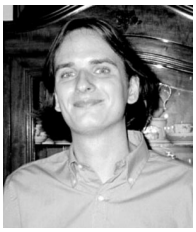
This work was supported in part by an ONR N-0014-02-1-0931, an US National Science Foundation Career Award NSF-9624604, NSF EIA-98-09209, AFOSR F49620-98-1-0434, and NSBRI-NBPF00202. S. Goldenstein was partially supported by CNPq—Brazilian's "*Conselho Nacional de Desenvolvimento Científico e Tecnológico*."

REFERENCES

[1] K. Arras, "An Introduction to Error Propagation: Derivation, Meaning, and Examples of Equation cy = fx cx fx," Technical Report EPFL-ASL-TR-98-01 R3, EPFL, 1998.
[2] J. Barron, D. Fleet, and S. Beauchemin, "Performance of Optical Flow Techniques," *Int'l J. Computer Vision*, vol. 12, pp. 43-77, 1994.
[3] M. Black and Y. Yacoob, "Recognizing Facial Expressions in Image Sequences Using Local Parameterized Models of Image Motion," *Int'l J. Computer Vision*, vol. 25, no. 1, pp. 23-48, 1997.
[4] A. Blake and M. Isard, *Active Contours: The Application of Techniques from Graphics, Vision, Control Theory and Statistics to Visual Tracking of Shapes in Motion.* Springer Verlag, 1999.
[5] V. Blanz and T. Vetter, "A Morphable Model for the Synthesis of 3D Faces," *Proc. SIGGRAPH*, pp. 187-194, Aug. 1999.
[6] M. Brand and R. Bhotika, "Flexible Flow for 3D Nonrigid Tracking and Shape Recovery," *Proc. Computer Vision and Pattern Recognition*, pp. 315-322, 2001.
[7] C. Bräutigam, "A Model-Free Voting Approach to Cue Integration," PhD thesis, Dept. of Numerical Analysis and Computing Science, KTH (Royal Institute of Technology), 1998.
[8] J. Canny, "A Computational Approach to Edge Detection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8, pp. 679-698, 1986.
[9] M. Cascia, S. Sclaroff, and V. Athitsos, "Fast, Reliable Head Tracking under Varying Illumination: An Approach Based on Registration of Texture-Mapped 3D Models," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 4, pp. 322-336, Apr. 2000.
[10] K.L. Chung, *A Course in Probability Theory*, second ed. Academic Press, Inc., 1974.
[11] T. Cootes and C. Taylor, "Active Shape Models—Their Training and Application," *Computer Vision and Image Understanding*, vol. 61, no. 1, pp. 38-59, 1995.

[12] J. Craig, *Introduction to Robotics: Mechanics and Control.* Addison Wesley, 1989.
[13] L. de Figueiredo, "Surface Intersection Using Affine Arithmetic," *Graphics Interface,* pp. 168-175, 1996.
[14] D. DeCarlo and D. Metaxas, "Optical Flow Constraints on Deformable Models with Applications to Face Tracking," *Int'l J. Computer Vision,* vol. 38, no. 2, pp. 99-127, July 2000.
[15] D. DeCarlo, D. Metaxas, and M. Stone, "An Anthropometric Face Model Using Variational Techniques," *Proc. SIGGRAPH,* pp. 67-74, 1998.
[16] L. Egiziano, N. Femia, and G. Spagnuolo, "New Approaches to the True Worst-Case Evaluation in Circuit Tolerance & Sensitivity Analysis: Part II—Calculation of the Outer Solution Using Affine Arithmetic," *Proc. Sixth Workshop Computer in Power Electronics (COMPEL),* 1998.
[17] W. Feller, *An Introduction to Probability Theory and Its Applications,* vol. II. John Willey & Sons, 1971.
[18] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice,* second ed. Addison Wesley, 1995.
[19] M. Gleicher and A. Witkin, "Supperting Numerical Computations in Interactive Contexts," *Graphics Interface,* pp. 138-145, 1993.
[20] S. Goldenstein, "Statistical Cue Estimation for Model-Based Shape and Motion Tracking," PhD thesis, Univ. of Pennsylvania, 2002.
[21] S. Goldenstein, C. Vogler, and D. Metaxas, "Affine Arithmetic Based Estimation of Cue Distributions in Deformable Model Tracking," *Proc. Computer Vision and Pattern Recognition,* pp. 1098-1105, 2001.
[22] J. Gomes, L. Darsa, B. Costa, and L. Velho, *Warping and Morphing of Graphical Objects.* Morgan Kauffman, 1999.
[23] N. Gordon, D. Salmon, and A. Smith, "A Novel Approach to Nonlinear/Nongaussian Bayesian State Estimation," *IEEE Proc. Radar Signal Processing,* vol. 140, pp. 107-113, 1993.
[24] D. Heckerman, "A Tutorial on Learning with Bayesian Networks," Technical Report MSR-TR-95-06, Microsoft Research 1995.
[25] R. Ierusalimschy, L.H. de Figueiredo, and W. Celes, "Lua—An Extensible Extension Language," *Software: Practice & Experience,* vol. 26, no. 6, pp. 635-652, 1996.
[26] M. Isard and A. Blake, "CONDENSATION: Conditional Density Propagation for Visual Tracking," *Int'l J. Computer Vision,* vol. 29, no. 1, pp. 5-28, 1998.
[27] G. Kamberova and M. Mintz, "Minimax Rules under Zero-One Loss for a Restricted Location Parameter," *J. Statistical Planning and Inference,* vol. 2, no. 79, pp. 205-221, 1999.
[28] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active Contour Models," *Int'l J. Computer Vision,* vol. 1, pp. 321-331, 1988.
[29] R. Mandelbaum, G. Kamberova, and M. Mintz, "Stereo Depth Estimation: A Confidence Interval Approach," *Proc. Int'l Conf. Computer Vision,* 1998.
[30] P. Maybeck, *Stochastic Models, Estimation, and Control.* Academic Press, 1979.
[31] F. Messine and A. Mahfoudi, "Use of Affine Arithmetic in Interval Optimization Algorithms to Solve Multidimensional Scaling Problems," *Proc. IMACS/GAMM Int'l Symp. Scientific Computing, Computer Arithmetic and Validated Numerics,* 1998.
[32] R. Moore, *Methods and Applications of Interval Analysis.* SIAM, 1979.
[33] H. Murase and S.K. Nayar, "Visual Learning and Recognition of 3D Objects from Appearance," *Int'l J. Computer Vision,* vol. 14, pp. 5-24, 1995.
[34] F. Pighin, R. Szeliski, and D. Salesin, "Resynthesizing Facial Animation through 3D Model-Based Tracking," *Proc. Int'l Conf. Computer Vision,* pp. 143-150, 1999.
[35] A. Requicha, "Representation for Rigid Solids: Theory, Methods, and Systems," *ACM Computing Surveys,* vol. 12, pp. 437-464, 1980.
[36] D. Samaras, D. Metaxas, P. Fua, and Y.G. Leclerc, "Variable Albedo Surface Reconstruction from Stereo and Shape from Shading," *Proc. Computer Vision and Pattern Recognition,* pp. 480-487, 2000.
[37] J. Shi and C. Tomasi, "Good Features to Track," *Proc. Computer Vision and Pattern Recognition,* pp. 593-600, 1994.
[38] J. Stolfi and L. Figueiredo, " Self-Validated Numerical Methods and Applications," *Proc. 21st Colóoquio Brasileiro de Matemática, (21st Brazilian Mathematics Colloquium) IMPA,* 1997.
[39] C. Tomasi and T. Kanade, "Detection and Tracking of Point Features," Technical Report CMU-CS-91-132, School of Computer Science, Carnegie Mellon Univ., 1991.

Siome Klein Goldenstein received the EE degree from the Universidade Federal do Rio de Janeiro, Brazil in 1996, the MSc degree in computer science from the Pontifícia Universidade Católica do Rio de Janeiro, Brazil in 1997, and the PhD degree in computer and information science from the University of Pennsylvania in 2002. He is currently a postdoctoral fellow at the Center for Computational Bioengineering, Imaging and Modeling (CBIM), Rutgers University. His main research interests lie in the study of motion for computer vision and computer graphics. He is a member of the IEEE and the IEEE Computer Society.



Christian Vogler did his undergraduate work in computer science and sign language linguistics at the University of Hamburg from 1992-1995. He received the MS degree in computer and information science in 1996, and the PhD degree in computer and information science in 2002, both from the University of Pennsylvania. He is a postdoctoral fellow at the Department of Computer Science at Rutgers University. He specializes in sign language and human motion recognition techniques, and in 3D facial parameter estimation with deformable models.

Dimitris Metaxas received a Diploma in electrical engineering from the National Technical University of Athens in 1986, the MSc degree in computer science from the University of Maryland in 1988, and the PhD degree in computer science from the University of Toronto in 1992. He is a professor in the Division of Computer and Information Science and the Department of Bioengineering at Rutgers University. From January 1998 to December 2001, he was an associate professor in the Department of Computer and Information Science at the University of Pennsylvania. From September 1992 to December 1997, he was an assistant professor in the same department. Dr. Metaxas is the director of the Center for Computational Bioengineering, Imaging and Modeling (CBIM) and while at PENN he was heading the Vision, Analysis, and Simulation Technologies Lab (VAST). He specializes in physics-based techniques for the modeling, estimation, and synthesis of the shape and motion for complex objects. He is the author of the book *Physics-Based Deformable Models: Applications to Computer Vision, Graphics and Medical imaging* published by Kluwer Academic Publishers in November 1996. He is an associate editor of *Graphical Models and Image Processing*, he is on the editorial board of *Medical Imaging*, and has been an associate editor for *Transactions on Pattern Analysis and Machine Intelligence*. Dr. Metaxas has organized several workshops and conferences in his area, he has served on the program committees of all major computer vision, medical imaging, and computer graphics conferences since 1992, and has more than 170 research publications in these areas. He received an US National Science Foundation Initiation Award (1993), an US National Science Foundation Career Award (1996), and an Office of Naval Research Young Investigator Proposal Award (1997). He has been invited nationally and internationally to speak on his research and his research has received several best paper awards and patents. He has graduated 17 PhD students and is a member of the ACM, the IEEE, and the IEEE Computer Society.